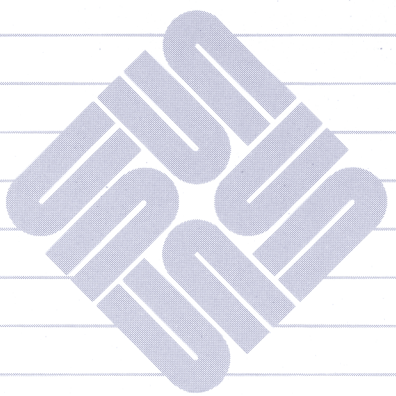
sun®
microsystems

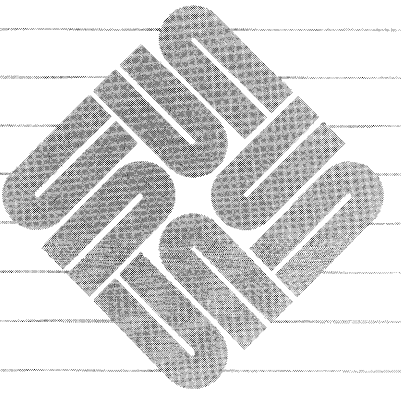
Change Pages and Addenda to the Docubox



Part. No. 800-3378-10
Revision A of 24 April 1989

sun®
microsystems

Change Pages and Addenda to the Docubox



Part. No. 800-3378-10
Revision A of 24 April 1989

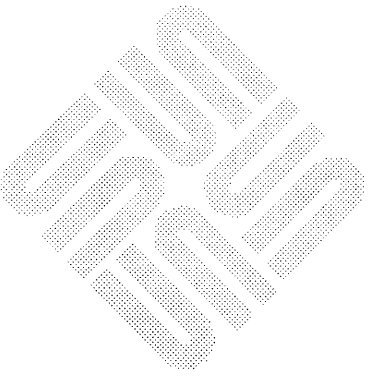
System Administration Addenda

Sun Microsystems, Inc. • 2550 Garcia Avenue • Mountain View, CA 94043 • 415-960-1300

Contents

Tables

Figures



System and Network Administration Addenda

This document contains new material and revised chapters for you to insert into your SunOS 4.0 *System and Network Administration* manual. Major sections include

- Additions to SunOS 4.0.3 Affecting System Administration
- Configuring the SunOS Kernel
- Using the Automounter
- The Sun Yellow Pages Service
- Setting Up Electronic Mail

Each section explains where to put it in your *System and Network Administration* manual.

SunOS 4.0.3 Changes That Affect Administration

SunOS Release 4.0.3 contains file system enhancements and new hardware support that affect system and network administration. The following section highlights these enhancements and contains procedures for implementing them in your environment. Topics discussed include:

- Changes to the SunOS file system.
- Floppy disk support for the Sun-3/80.
- Adding new clients of unsupported kernel architectures to existing networks.

These sections are supplemental to the information in your *System and Network Administration* manual. They do not replace text in the manual, unless otherwise noted.

0.1. File System Changes in Release 4.0.3

4.0.3 Inclusion Instructions

Include this section with Chapter 6, "The SunOS File System," under the section, "The `/usr` File System."

The `/usr` file system has been modified in Release 4.0.3 to reflect the addition of kernel architectures to support new Sun machines. Machines with the same application architecture but different kernel architectures cannot share certain executables. These unshareable executables now reside in the `kvm` directory hierarchy.

The `/usr/kvm` Directory

`/usr/kvm` contains the kernel architecture-dependent executables. Here is a typical `/usr/kvm`.

```
i386        libkvm.so.0.3   pdp11     sun2      u3b15
iAPX286     m68k            ps        sun3      u3b2
ld.so       machine         pstat     sun3x     u3b5
ldconfig    mc68010         sparc     sun4      vax
libkvm.a    mc68020         sun       u370      vmstat
            mdec/           stand/    u3b
```

The contents of `/usr/kvm` are described below.

The following are symbolic links to either `/bin/true` or `/bin/false`. They give the appropriate machine identity to the commands by the same name in `/usr/bin`.

```
sun2        m68k        u3b15
sun3        mc68010     u3b2
sun3x       mc68020     u3b5
sun4        sparc       vax
sun4c       pdp11
i386        u370
iAPX286     u3b
```

The commands `ps`, `pstat`, and `vmstat` display system statistics, such as process status and virtual memory usage. They are fully described in their appropriate man pages.

`machine` is a symbolic link to give the appropriate kernel architecture identity to `/usr/include/machine`.

`libkvm.a` and `libkvm.so.0.3` are shared libraries. The `ldconfig` command and `ld.so` link editor are used to link these shared libraries.

The directories `mdec` and `stand` contain executables that the machine uses when booting.

`/usr/kvm` holds directories that are specific to the server's kernel architecture. When you add a kernel architecture to the server, you install its `kvm` directory hierarchy into `/export/exec/kvm/sun`[*2,3,4,3x,4c*].

Here is a sample `kvm` directory for a Sun-3x machine. The directory's full pathname is `/export/exec/kvm/sun3x`.

```
boot        libkvm.so.0.3   pdp11     sun2      u3b15
i386        m68k            ps        sun3      u3b2
iAPX286     machine         pstat     sun3x     u3b5
ld.so       mc68010         sparc     sun4      vax
ldconfig    mc68020         stand     u370      vmstat
libkvm.a    mdec            sun       u3b
```

Note that the contents of this directory are the same as `/usr/kvm`, with the addition of `boot`. The `boot` directory contains that architecture's actual kernel: `vmunix` or `vmunix_small`.

The later subsection "Adding Clients to an Existing 4.0.3 Network" contains more information about kernel architectures.

0.2. Floppy Format for the Sun-3/80

4.0.3 Inclusion Instructions

Include this section at the end of Chapter 10, "Maintaining Disks with `format`."

You use the `fdformat` command to format your floppy disk to use with the SunOS. You must format all new blank disks before using them. After finishing with the floppy disk, use the `eject` command to eject it from the drive.

0.3. Formatting your Floppy Disk

The `fdformat` program formats and verifies each track on the diskette. `fdformat` terminates when it finds any bad sectors. The default for **`fdformat`** is to format a 1.44 megabyte high density diskette. Use the **`-L`** or the **`-l`** options to format low density diskettes.

NOTE `fdformat` *destroys all existing data on the disk.*

Mounting the Floppy Drive

In order for the `fdformat` command to work, you must mount the floppy drive device.

Follow these steps to mount the floppy drive:

1. Become superuser.
2. Go to the `/dev` directory.

```
novel# cd /dev
```

3. Mount the device.

```
novel# MAKEDEV fd0
```

You can now format your disk.

`fdformat` Syntax

Insert the floppy disk you want to format, then enter the `fdformat`command.

The basic syntax of `fdformat` is as follows:

```
novel# fdformat [-eflLv [device]
```

The **`-e`** option ejects the diskette when done.

The **`-f`** forces format to start without confirmation.

The **`-l`** formats a low density diskette (720 kilobyte) diskette.

The **`-L`** also formats a low low density diskette (720 kilobyte) diskette.

The **-v** verifies the floppy diskette after formatting. If the floppy fails verification, discard the diskette.

The *device* option names the special device file to use. This special device file should correspond to the correct unit number for the device. The default device for the Sun 3/80 is the internal floppy drive, **/dev/rfd0c**.

Error Message

If you try to format a floppy disk and get the following message, then your disk drive is probably not mounted.

```
/dev/rfdl0: no such file or directory
```

Go to the *Mounting the Floppy Drive* section, mount the drive, and try again.

Example

To format your floppy disk at high density followed by an automatic eject, type:

```
novel# fdformat -e /dev/rfd0c
```

0.4. Ejecting the Floppy Disk

The `eject` command is used when removable media devices do not have a manual eject button. You can specify the device by its name or by a nickname; if you do not specify a name or nickname, the default device is used.

eject can also display its default device and a list of nicknames.

`eject` **Syntax**

The syntax for `eject` is as follows:

```
novel# eject [-d|-f|-n|[device|nickname]
```

The **-d** option displays the name of the default device to be ejected.

The **-f** option forces the device to eject even if it is busy.

The **-n** option displays the nickname to the device name translation table.

The *device* specifies which device to eject by the name that appears in the directory **/dev**.

The *nickname* specifies which device to eject by the nickname known to this command.

The default filename for this command is **/dev/rfd0a**.

0.5. Adding Clients to an Existing 4.0.3 Network

4.0.3 Inclusion Instructions

This section replaces "Upgrading an NFS Server from Homogeneous to Heterogeneous" and "Adding and Removing Clients of an NFS Server," in Chapter 13, "The Sun Network File System."

The `setup_exec` and `setup_client` programs, described in *System and Network Administration* for Release 4.0, now support sun3x and sun4c kernel architectures.

Initially, you install Release 4.0.3 by running `suninstall` or `sunupgrade`. However, once the operating system runs successfully, you add clients to your network by running `setup_client` and `setup_exec`. You need to run both programs or just `setup_client`, depending on client architecture and the architectures already supported by your NFS servers. The following table explains when you should run each program:

Program Name	*When to Run It*
`setup_client`	To add a client to an existing 4.0.3 network, regardless of client or server architecture.
`setup_exec`	To add a client with a kernel architecture different from those supported by a running server.

The following subsections contain procedures for both programs. If your new client has a kernel architecture already supported by the server, skip the next subsection, and go on to the subsection, "Adding a New Client."

Adding a New Kernel Architecture

The `/usr` file system of a homogeneous NFS server contains programs that can only be used by machines with the same application architecture as the server. These executable programs are referred to as "architecture dependent." To upgrade the server to support clients with a new application architecture, you have to install executable programs for that architecture. This changes the server from homogeneous to heterogeneous. SunOS keeps additional architectures in the directory `/export/exec`. In previous releases of SunOS 4.0, you ran `setup_exec` to add these executables.

With Release 4.0.3, you also must run `setup_exec` to add support for machines with an already supported application architecture, but a new kernel architecture, thus different `/usr/kvm` directories. Because `/usr/kvm` contains crucial executables, you must run `setup_exec` to install `/usr/kvm` and `/usr/sys` for the new kernel architecture.

For example, suppose you install Release 4.0.3 on a homogeneous network of Sun-3/80s and Sun-3/470s. Then you want to add a Sun-3/60 to this network. You have to run `setup_exec` before adding the Sun-3/60. All three models have the same application architecture, `sun3`, but their kernel architectures are different— `sun3`, `sun3x`. Likewise, you must run `setup_exec` to add machines of the `sun4c` (SPARCsystem 330) kernel architecture to a Sun-4 server's existing network.

Procedures for Running `setup_exec`

Before you run `setup_exec`, make sure you have the appropriate release tape for the new client's kernel architecture. Then follow these procedures:

1. Install the release tape in a local or remote tape drive.
2. Become superuser on the NFS server that will serve the new client.
3. Type the following:

```
# /usr/etc/install/setup_exec
```

`setup_exec` displays the SOFTWARE FORM also used by `suninstall`:

```
Architecture Information :
     Type          :  [sun2]   [sun3]  [sun3x]   [sun4]   [sun4c]
     Path where executables reside :
     Path where kernel executables reside :
Media Information :
     Device Type : [st0]  [st1]  [st2]  [ar0]  [mt0]  [xt0]
     Drive Type  :  [local]  [remote]



Choice             :  [all]   [default]   [own choice]   [required]   [quit]









Are you finished with this form [y/n] ?
   [x/X=select choice] [space=next choice] [^B/^P=backward] [^F/^N=forward]
```

4. In response to "Type," type **x** before the appropriate kernel architecture type. The cursor automatically moves to the next prompt line.
5. In response to the prompt

```
Path where executables reside :
```

type the full pathname for the executables of the *application* architecture of

the client to be added. For example, type `/export/exec/sun3` for `sun3` and `sun3x` machines, or `/export/exec/sun4` for `sun4` and `sun4c`.

6. In response to the prompt

```
Path where kernel executables reside :
```

type the full pathname for the client's `kvm` directory. For example, for a Sun-3/470, you respond:

```
Path where kernel executables reside : /export/exec/kvm/sun3x
```

7. For media type, first type an **x** before the device abbreviation for your tape drive. Here is how you respond if you have mounted the tape in a Xylogics drive.

```
Device Type : [st0]  [st1]  [st2]  [ar0]  [mt0]  x[xt0]
```

8. Next, type an **x** before the location of the tape drive, such as:

```
Drive Type  :  x[local]  [remote]
```

9. For the Choice prompt, place an **x** before the "own choice" parameter.

10. Type **y** to indicate that you are finished. (Or, re-edit the form by pressing the keys indicated on the menu.) `setup_exec` then prompts you to choose the software you want from the selections it places on your screen, as follows:

```
CATEGORY      NAME                    BYTES       AVAIL BYTES     Y/N
=============================================================================
required      usr                     20971520    126444544       n
required      Kvm                     2653184     103166157       y
desirable     Sys                     2917376     23768064        y
desirable     Networking              961536      100221123       n
              .
              .
```

11. Type **y** only for the categories `kvm` and `sys`. Type **n** for all others. Then press RETURN to begin `setup_exec`.

When `setup_exec` finishes, you can remove the release tape. The newly installed `kvm` executables now reside in the `/export/exec` directory. Now you can add the new client machine through the `setup_client` program.

Adding and Removing Clients through `setup_client`

You actually add clients by issuing the `setup_client` command. This subsection explains how to set up clients of homogeneous and heterogeneous servers. Below are general procedures for running `setup_client`. You will find more specific procedures later in this section.

1. Ensure that the client is physically attached to the server's network, via Ethernet or similar media.
2. Become superuser on your NFS server.
3. Type the following:

```
# cd /usr/etc/install/script
# setup_client
```

to display the syntax for `setup_client`:

```
setup_client op clientname yptype size rootpath swappath homepath execpath \
kvmpath arch
where:
        op          = "add" or "remove"
        clientname  = name of the client machine
        yptype      = "master" or "slave" or "client" or "none"
        size        = size for swap
                      (e.g. 16M or 16m ==> 16777216 bytes
                            16000K or 16000k ==> 16384000 bytes
                            31250B or 31250b ==> 31250 blocks )
        rootpath    = parent pathname of client root (e.g. /export/root )
        swappath    = parent pathname of client swap (e.g. /export/swap )
        homepath    = parent pathname of client home (e.g. /home, remotehost:/home )
        execpath    = full pathname of exec directory
                      (e.g. /export/exec/sun2, /export/exec/sun3, etc)
        kvmpath     = full pathname of kvm directory
                      (e.g. /export/exec/kvm/sun3x)
        arch        = "sun2" or "sun3" or "sun3x"
                        or "sun4" or "sun4c" or "sun386"
```

You will see shortly how to specify these parameters for different types of machines.

4. Specify `setup_client` using the syntax shown above. `setup_client` creates the directories necessary for the client to operate, updates `/etc/exports` on the server, and creates an `/etc/fstab` file for the client. It then prompts you when it is finished.
5. On a network running YP, you need to update the `bootparams` database on the YP server. If your network does not run YP, go on to Step 6.
6. Boot the client machine.

`setup_client` revises the server's `/etc/exports` file so that it resembles the following:

```
/
/usr
/home
/usr/share
#
/export/root/client_name -root=client_name,access=client_name
/export/swap/client_name -root=client_name,access=client_name
#
/export/exec/kvm/client_kernel_arch
```

The client's `/etc/fstab` now contains entries similar to the following:

```
server:/export/root/client_name / nfs rw 0 0
server:/export/exec/client_app_arch /usr nfs ro 0 0
server:/export/exec/kvm/client_kernel_arch /usr/kvm nfs ro 0 0
server:/export/share /usr/share nfs ro 0 0
server:/home/server_name /home/server_name nfs rw 0 0
```

Adding a Client to a Heterogeneous Server

This example shows how to add a Sun-3/80 client called "felafel" to a network with YP that is supported by a Sun-4 NFS server called "grub." In doing this procedure, you upgrade server grub from homogeneous to heterogeneous.

1. Before running `setup_client`, ensure that client felafel is physically connected (via Ethernet or similar technology) to your server's network.
2. Become superuser on server grub.
3. Add the client's Internet address to `/etc/hosts`.
4. Add the client's Ethernet address to `/etc/ethers`.
5. Run `setup_exec`, if you have not done so already, to install the `kvm` directory for the Sun-3/80's `sun3x` kernel architecture.
6. Type the following to set up the new client:

```
# setup_client add felafel client 16m /export/root /export/swap \
/home /export/exec/sun3 /export/exec/kvm/sun3x sun3x
```

`setup_client` displays the following on your screen:

```
Start creating sun3x client "felafel" :

Updating bootparams ...
ATTENTION: /etc/bootparams on the yp master needs to be updated.

Creating root for client "felafel".

Creating 16m bytes of swap for client "felafel".

Updating /etc/exports to export "felafel" info.
exportfs: /usr/share: parent-directory (/usr) already exported

Updating /etc/exports to export "/export/exec/kvm/sun3x".
exportfs: /usr/share: parent-directory (/usr) already exported
exportfs: /export/exec/kvm/sun3x: parent-directory (/usr) already exported

Completed creating sun3x client "felafel".
```

7. Become superuser on the network's YP server.
8. Edit the file `/etc/bootparams` and add the following for the new client:

```
felafel     root=grub:/export/root/felafel \
            swap=grub:/export/swap/felafel
```

9. Update the `bootparams` maps by typing the following:

```
# cd /var/yp
# make bootparams
```

10. Boot the client machine.

Removing a Client

This example shows how to remove an existing client called "curry" from an NFS server.

1. Become superuser on the server and type the following:

```
# /usr/etc/install/script/setup_client remove curry none 16M \
/export/root /export/swap /home /export/exec /export/exec/kvm/sun4 sun4
```

2. Remove entries for client curry in `/etc/hosts` and `/etc/ethers`.

Adding a Client to a Homogeneous Server

This example shows how to add a Sun-4/110 called "peas" to a network without YP that is supported by a Sun-4/260 NFS server called "dinner."

1. Before running `setup_client`, ensure that client peas is physically connected (via Ethernet or similar technology) to your server's network.
2. Become superuser on server dinner.

3. Add the client's Internet address to `/etc/hosts`.
4. Add the client's Ethernet address to `/etc/ethers`.
5. Type the following to set up the new client:

```
# setup_client add peas none 16m /export/root /export/swap /home \
/export/exec/sun4 /export/exec/kvm/sun4 sun4
```

`setup_client` will display a series of messages similar to those shown above for adding a client to a heterogeneous server.

7. When `setup_client` is finished, you can go to the client machine and boot it up.

Converting a Standalone System to NFS Server

You can convert a standalone system running Release 4.0.3 to an NFS server by running `setup_exec` and `setup_client`. Use the same `setup_exec` and `setup_client` syntax for upgrading a standalone as you would for upgrading a server.

For example, suppose you have a Sun-4 standalone system running Release 4.0, and you want to upgrade it to an NFS server supporting both Sun-4 and SPARCsystem 330 4.0.3 clients. You need to run `setup_exec` to install `sun4c` executable files onto the system. You do not need to install `sun4` executable files, since `suninstall` has already placed these files in your standalone's `/usr` file system. If you want to load the `sun4c` executable files into `/export/exec/kvm`, specify `/export/exec/kvm/sun4c` for the path where kernel executables reside. Then run `setup_client` to add clients to the new server.

9

Reconfiguring the System Kernel

9

Reconfiguring the System Kernel

4.0.3 Inclusion Instructions

This section replaces Chapter 9, "Reconfiguring the System Kernel."

You should reconfigure your system's kernel after installing Release 4.0.3. The kernel provided for SunOS Release 4.0.3 supports many more devices than previous releases. Therefore, it is significantly larger than the kernels of earlier releases, and will occupy a considerable amount of memory. Tailoring the kernel for your own system significantly improves system performance.

Reconfiguring the kernel is not a difficult task. The `GENERIC` kernel configuration files for each architecture contain instructions that help you decide which kernel entries your particular system needs. If you carefully follow the instructions provided, particularly in the section, "Procedures for Reconfiguring the Kernel," and in the `README` file, also in the `/usr/share/sys/sun`*[2,3,3x,4]*`/conf` directory, you should have a streamlined, workable kernel without experiencing any problems.

This chapter discusses the following subjects:

- Reasons for reconfiguring the system's kernel.
- Major sections of the `GENERIC` kernel configuration file, from a broad perspective.
- Contents of `GENERIC` for each model of Sun computers.
- Procedures for putting the reconfigured kernel into operation.
- Procedures for changing swap space.

9.1. Why Reconfigure the Kernel?

There are two reasons to reconfigure the kernel:

- To free up memory that would otherwise be used by unused kernel modules, thus improving your system's performance.
- To tell the kernel about the hardware you added after installation or the software packages that require kernel modification to support. The SunOS Release 4.0.3 tapes contain the kernel configuration file for your Sun

workstation. When you build the kernel from the kernel file supplied on the release tape, the utilities involved create code that supports all hardware and software devices available for your Sun workstation architecture. This kernel is unnecessarily large; your particular system probably does not need all those/ items. Furthermore, on machines with a small amount of memory, using the kernel configuration file on the release tape wastes large amounts of main memory, seriously degrading system performance.

Modifying a copy of the GENERIC configuration file restricts the modified kernel to only those items that apply to your configuration. This smaller, customized kernel takes up less space in memory, giving larger effective memory size to programs. This improves system performance substantially, particularly if you intend to run SunView and other large applications or programs.

You also must reconfigure the kernel when you make major hardware or software additions to your system. For example, you edit the kernel configuration file when you add new hardware, such as a second disk or graphics controller. In addition, you need to edit the kernel configuration file when you add major software packages that may not have been selected when you initially ran `suninstall`. For example, if you decide to attach your standalone configuration to the network file system (NFS), you have to enable this option from the configuration file.

9.2. Parts of the Kernel Configuration File

This section examines the kernel configuration file from both a broad and narrow perspective. First, it explains how the configuration file is used during the kernel building process. Then an overview is given that describes the major sections of the configuration file. Finally, the annotated configuration section explains each line in the GENERIC configuration file for a Sun-2, Sun-3, Sun-3x, and Sun-4.

The Reconfiguration Process

Building a new system is a semi-automatic process. Most of it is handled by a configuration-build utility called `/usr/etc/config`, which generates the files you need to compile and link the kernel. Your major activity in the configuration process is to create the kernel configuration file *SYSTEM_NAME*. *SYSTEM_NAME* is actually the name of your machine or other identifying name. This file contains a description of the kernel you want `/usr/etc/config` to produce. `/usr/etc/config` then uses this information to create the directory `/usr/share/sys/sun`*[2,3,3x,4]/SYSTEM_NAME* and builds the kernel there.

Rather than creating the configuration file from scratch, you can copy and edit the GENERIC configuration file provided with your release in `/usr/share/sys/sun[2,3,3x,4]/conf.GENERIC` reflects the configuration file supplied by Sun, in that it contains all possible entries for your model of Sun workstation.

Major Sections of the GENERIC Configuration File

The GENERIC configuration file is divided into three sections:

- A system identification section that identifies the type of machine you have, including the name that you want to give the kernel. This section is similar for the GENERIC configuration file for each model type.

- A connections section that tells the kernel which CPU board and bus connections are available for attaching controllers.
- A devices section that contains lines specific to each type of controller, disk, tape, or other type device board that you want to configure.

To understand the format of the configuration file, it is best to begin by examining GENERIC. Note that the pound sign character (#) starts a comments that continues to the end of the line. Here is an example of the first few lines of the Sun-3 GENERIC file.

```
# @(#)GENERIC 1.82 88/02/08 SMI
#
# This config file describes a generic Sun-3 kernel, including all
# possible standard devices and software options.
#
# The following lines include support for all Sun-3 cpu types.
# There is little to be gained by removing support for particular
# cpu's, so you may as well leave them all in.
#
machine         "sun3"
cpu             "SUN3_160"      # Sun-3/75, Sun-3/140, Sun-3/160, or Sun-3/180
cpu             "SUN3_50"       # Sun-3/50
cpu             "SUN3_260"      # Sun-3/260 or Sun-3/280
cpu             "SUN3_110"      # Sun-3/110
cpu             "SUN3_60"       # Sun-3/60
cpu             "SUN3_E"        # Sun-3E (Eurocard VMEbus cpu)
#
# Name this kernel "GENERIC".
#
ident           GENERIC
#
# This kernel supports about eight users.  Count one
# user for each timesharing user, one for each window
# that you typically use, and one for each diskless
# client you serve.  This is only an approximation
# used to control the size of various kernel data
# structures, not a hard limit.
#
maxusers        8

#
# Include all possible software options.
#
# The INET option is not really optional, every kernel must include it.
#
options         INET            # basic networking support - mandatory
#
# The following options are all filesystem related.  You only need
# QUOTA if you have UFS.  You only need UFS if you have a disk.
# Diskless machines can remove QUOTA, UFS, and NFSSERVER.  LOFS is
# only needed if you're using the Sun Network Software Environment.
```

```
#
options         QUOTA           # disk quotas for local disks
options         UFS             # filesystem code for local disks
options         NFSCLIENT       # NFS client side code

          .
          .
          .
#
# Build one kernel based on this basic configuration.
# It will use the generic swap code so that you can have
# your root filesystem and swap space on any supported device.
# Put the kernel configured this way in a file named "vmunix".
#
config          vmunix          swap generic

#
# Include support for all possible pseudo-devices.
#
# The first few are mostly concerned with networking.
# You should probably always leave these in.
         .
         .
         .
```

You can see by these line groupings that the configuration file has three different types of entries:

- Lines that give a general description of the system (parameters global to the kernel image that this configuration generates)
- A line that describes items specific to each kernel image generated
- Lines that describe the devices on the system and connections to which these devices are attached.

The System Identification Section

The system identification section, which is shown in the illustration above, contains general system description lines and system-specific lines. They are described below.

General System Description Lines

The first six general description lines in the configuration file are mandatory for every Sun workstation. They are:

machine *type*

This field tells the kernel that the system is to run on the machine type specified. The legal *types* for a Sun workstation are "sun2", sun3", "sun-3x", and "sun4", and "sun4c". Note that the double quotes are considered part of the description.

cpu *type*

This field indicates that the system is to run on the CPU type specified. More than one CPU type can appear in the configuration file.

Legal types for a Sun-2 machine are:

```
"SUN2_120"   # Sun-1/100U, Sun-1/150U, Sun-2/120, Sun-2/170
"SUN2_50"    # Sun-2/50, Sun-2/160
```

Legal types for a Sun-3 machine are:

```
"SUN3_160"   # Sun-3/75, Sun-3/140, Sun-3/160, or Sun-3/180
"SUN3_50"    # Sun-3/50
"SUN3_260"   # Sun-3/260 or Sun-3/280
"SUN3_110"   # Sun-3/110
"SUN3_60"    # Sun-3/60
"SUN3_E"     # Sun-3E (Eurocard VMEbus cpu)
```

Legal types for a Sun-3x are:

```
"SUN3x_470"      # Sun-3/470, Sun-3/480, Sun-3/460
"SUN3x_80"       # Sun-3/80
```

Legal types for a Sun-4 machine are:

```
"SUN4_260"   # Sun-4/260, Sun-4/280
"SUN4_110"   # Sun-4/110
```

Legal types for a Sun-4c are:

```
"Sun4c_60"   # SPARCstation 1
```

ident *name*

This field tells the kernel the name you wish for the system identifier—the name for the machine or machines that run this kernel. You must include *name* in double quotes if it contains any numbers (for example, "SDST120"), or you will get a syntax error when you run `/usr/etc/config`. If the name is `GENERIC`, the kernel name will be taken from the configuration file name.

Note that when editing the `GENERIC` file, you do not have to name your kernel `GENERIC`. If you specify `options generic`, then you must use the `swap generic` clause on the `config` line. When you specify these two, the kernel prompts you during the boot process for the location of `/root` and `/swap`, for example, on the local disk or from an NFS server.

maxusers *number*

This field tells the kernel that the maximum expected number of simultaneously active users on this system is *number*. The number you specify is used to size several system data structures. The recommended value for `maxusers` on a server, regardless of model type is 8. The `GENERIC` configuration files for each model type give further instructions to help determine a specification for `maxusers` that is appropriate to your system's needs.

options *type*

The fields beginning with the word `options` tell the kernel to compile the selected software options into the system. *type* has a different value for each options line. For example, here are several options lines for a Sun-4 GENERIC kernel:

```
#
options         QUOTA           # disk quotas for local disks
options         UFS             # filesystem code for local disks
options         NFSCLIENT       # NFS client side code
options         NFSSERVER       # NFS server side code
options         LOFS            # loopback filesystem - needed by NSE
#
```

Refer to the GENERIC configuration file for your system to determine which options are appropriate for it.

System-Specific Description Lines

The next type of line in the kernel configuration file is a single line specifying the name of the file the kernel build procedure will create.

The line has the following syntax:

```
config kernelname config_clauses
```

Here *kernelname* indicates the name of the loaded kernel image. Its value is usually vmunix.

config_clauses are one or more specifications indicating where the root file system is located and where the primary paging (or swap) device is located. A *config_clause* may be one or more of the following:

```
root [on] root device
```

This clause specifies the location of the root file system.

```
swap [on] swap_device
```

This clause specifies the location of the primary swapping and paging area. Specifying swap generic enables you to place your root file system and swap space on any supported device. When specifying options generic, you must specify swap generic, as well.

```
dumps [on] dump_device
```

This clause specifies where the /export/dump kernel should place core images after a crash.

The "on" in the syntax of each clause is optional. Separate multiple *config_clauses* by white space. For example, the *config* line for a system with root on its first SMD disk (Partition a) and swap on Partition b of the same disk might be:

```
config vmunix root on xy0a swap on xy0b
```

Note also that the device names supplied in the clauses may be fully specified—as a device, unit, and file system partition—or underspecified. If underspecified, the `config` program uses built-in rules to select default unit numbers and file system partitions. (Chapter 16 explains rules that are followed for underspecified location of devices.) For example, the swap partition need not be specified at all if the root device is specified. This is because the default is to place the `/swap` partition in Partition b of the same disk where the root file system is located. Thus you could use the following *config_clause* to represent the same information as the previous clause:

```
config vmunix root xy0
```

For diskless clients:

Use the following `config_clause`

```
config vmunix root on type nfs
```

The Pseudo-Devices Section

This section lists all possible *pseudo devices* for your model. A pseudo-device is a collection of programs or a device driver that has no associated hardware. For example, here are three pseudo-devices needed to run SunView 1

```
pseudo-device   win128  # window devices, allow 128 windows
pseudo-device   dtop4   # desktops (screens), allow 4
pseudo-device   ms3     # mouse support, allow 3 mice
```

The `GENERIC` configuration file gives suggestions as to which pseudo-devices you may need.

The Connections Section

The next section in the configuration file lists the possible on board and bus connections, grouped together by the machine model. These connections, in conjunction with controllers, devices, and disks form a structure that enables your system to recognize various hardware attached to it. For each device or controller on a bus, you need to select the bus type it is connected to, as listed under connections for your machine type.

For a Sun-2, connections are divided into two groups, machine 1, which includes all workstations with a Multibus, and machine 2, which includes all workstations with a VMEbus. Sun-3, Sun-3x, and Sun-4 have separate connection lists for each model, or group of models, as you will see if you examine their GENERIC configuration files. Here is a segment from the Sun-3 GENERIC kernel connections section.

```
# connections for machine type 1 (SUN3_160)
controller      virtual 1 at nexus ?    # virtually addressed devices
controller      obmem 1 at nexus ?      # memory-like devices on the cpu board
controller      obio 1 at nexus ?       # I/O devices on the cpu board
controller      vme16d16 1 at nexus ?   # VME 16 bit address 16 bit data devices
controller      vme24d16 1 at nexus ?   # VME 24 bit address 16 bit data devices
controller      vme32d16 1 at nexus ?   # VME 32 bit address 16 bit data devices
controller      vme16d32 1 at nexus ?   # VME 16 bit address 32 bit data devices
controller      vme24d32 1 at nexus ?   # VME 24 bit address 32 bit data devices
controller      vme32d32 1 at nexus ?   # VME 32 bit address 32 bit data devices
```

```
# connections for machine type 2 (SUN3_50)
controller      virtual 2 at nexus ?
controller      obmem 2 at nexus ?
controller      obio 2 at nexus ?
```

Note that machine type 1 is a Sun-3/160 and machine type 2 is a Sun-3/50. The first three connections, `virtual`, `obmem`, and `obio` are used by Sun for specific devices. For example, the `fpa` floating point accelerator uses the `virtual` connection, and various graphics controllers use `obmem` and `obio`. For machine type 1, connections prefaced with "`vme`" are on the VMEbus. For example, the phrase vme32d16 indicates a 32 bit VMEbus with 16 bit data. Note however that the Sun-3/50 does not have a VME connection listed.

The easiest way to modify the connections section is to leave as is all connections lines listed for your machine type. Then, comment out each connection line for all other machine types. That way, as you add controllers and devices, the connections are already enabled and will be recognized by your system.

The Devices Section

The final section of the configuration file lists all devices that can be supported by a Sun-2, Sun-3, Sun-3x, or Sun-4. Devices are grouped into controllers and, if applicable, the disks and tapes that may be connected to them. Each device is listed on a separate device description line. The basic format of these lines is described as follows.

Device Description Lines

When reconfiguring the kernel configuration file, you need to specify each device on your machine so that the generated kernel recognizes these devices during the boot process. Devices may be hardware-related entities, that is, controller boards and devices attached to the controllers, or software pseudo-devices.

The device description lines tell the system what devices to look for and use, and how these devices are inter-connected. Each line has the following syntax:

dev_type dev_name `at` *connect_dev more info*

Below is a definition of each parameter on the device description line.

dev_type

This item specifies the device type. *dev_type* may be one of the following:

- **controller.** Usually a disk or tape controller.
- **disk** or **tape.** Devices connected to a controller.
- **device.** Hardware entity attached to the main system bus, for example, an Ethernet controller board.
- **pseudo-device.** Software subsystems or drivers with no associated hardware, such as the pseudo-tty driver and various network subsystems, such as the NFS and Internet subsystems.

dev_name

Standard device name and unit number of the device you are specifying (if the device is not a pseudo-device). For example, *dev_name* for the first Xylogics disk controller on a system is `xyc0`.

connect_dev

Connection to which this device is attached. Here are the possible connections for all Sun workstation configurations:

`virtual` Virtual preset

`obmem` On-board memory

`obio` On-board I/O

`mbio` Multibus I/0 (Sun-2 only)

`mbmem` Multibus Memory (Sun-2 only)

`vme16d16 (vme16)` VMEbus: 16 bit address/16 bit data (Sun-3 Sun-3x and Sun-4)

`vme24d16 (vme24)` VMEbus: 24 bit address/16 bit data (Sun-3 Sun-3x and Sun-4)

`vme32d16` VMEbus: 32 bit address/16 bit data (Sun-3 and Sun-4)

`vme16d32` VMEbus: 16 bit address/32 bit data (Sun-3 Sun-3x and Sun-4)

`vme24d32` VMEbus: 24 bit address/32 bit data (Sun-3 Sun-3x and Sun-4)

`vme32d32` VMEbus: 32 bit address/32 bit data (Sun-3 Sun-3x and Sun-4)

When modifying the configuration file, you must specify the connections that apply to your system, but do not need to specify connections that don't apply. If you are unsure of the type of system bus or data bus you have, refer to the hardware manuals that came with the system.

more_info

This is a sequence of the following:

```
csr addr drive number flags number priority level vector intr number
```

The above arguments are completely described in Chapter 16 of the *System and*

Network Administration manual because you need to supply values for them only if you are going to configure your own device drivers.

Briefly `csr` *addr* specifies the address of the csr (command and status registers) for a device. `drive` *number* specifies which drive the line applies to. `flags` *number*, `priority` *level*, and `vector` *intr number* are all values defined in the device driver.

Here is a sample set of lines describing Xylogics 450/451 disk controllers and disks.

```
controller  xyc0 at vme16d16 ? csr 0xee40 priority 2 vector xyintr 0x48
controller  xyc1 at vme16d16 ? csr 0xee48 priority 2 vector xyintr 0x49
disk        xy0 at xyc0 drive 0
disk        xy1 at xyc0 drive 1
disk        xy2 at xyc1 drive 0
disk        xy3 at xyc1 drive 1
```

Bus types and devices must hang off the appropriate controller, which in turn hangs off another controller until a configuration is formed that gets you to a bus type that hangs off a "nexus." Notice how each line in the connections section concludes with the words "`at nexus .`" On Sun systems, all bus types are considered to hang off a nexus. For example, the following SMD disk :

```
disk    xy0 at xyc0 drive 0
```

is attached to the Xylogics controller:

```
controller  xyc0 at vme16d16 ? csr 0xee40 priority 2 vector xyintr 0x48
```

that is attached to the bus type vme16d16, as listed in the following entry from the connections section:

```
controller      vme16d16 1 at nexus ?
```

In order to determine and note which standard devices are present on your machine, boot the GENERIC kernel after you have executed *suninstall*. If you want, you can delete, or comment out, those lines that pertain to devices not on your machine. Or you can configure your file with the devices that are on other machines that you will possibly want to boot from the same kernel.

9.3. Modifying the Kernel Configuration Files

This subsection shows the annotated `GENERIC` kernel configuration files for each model of Sun computer. Note how the GENERIC file contains comments that suggest which entries to pick for your particular system.

Note: Some parameters relating to the System V Inter-Process Communication (IPC) extensions may

also be tuned in the configuration file. These parameters do not appear in the GENERIC file but are documented in Chapter 16 of the *System Administration and Networking* manual.

If the comments indicate that the line is **mandatory**, you *must* include it in every system configuration file, either exactly as it stands, or, if commentary indicates variables, with the variables adjusted to fit your system. Some options shown as mandatory are only required if you have other related options selected for your system.

Modification Procedures

Here are suggested procedures for modifying the GENERIC kernel configuration file.

1. Go through the next subsections and find the copy of `GENERIC` that pertains to your model of Sun computer.
2. Read the annotated `GENERIC` file and determine how you want to modify each line. You can modify a line by doing one of the following:

- Changing its parameters so that it applies to your configuration. Usually you do this for lines indicated as **mandatory**. For example, the `maxusers` line is mandatory, but you can change its value from the default.
- "Commenting out" a line if it does not apply to your current configuration, but may apply to it in the future. To do this, type a pound sign (**#**) at the beginning of the line. The utilities that build the kernel ignore lines beginning with the pound sign.

  For example, suppose you have a SCSI-2 controller with one disk and one tape drive. You might want to comment out the lines that apply to a second SCSI-2 controller, second disk, and second tape drive. At a later date, you may add some or all of this equipment. All you need to do to have this equipment recognized is to remove the pound sign, then make the new kernel.
- Deleting any lines that will never apply to your configuration. For example, if you have a diskless client, you might want to delete lines applying to Xylogics disk and tape controllers. These controllers are often used with servers. If you do add a disk or tape to your machine, it will probably be enclosed in a "shoebox" containing SCSI controller, disk, and possibly, tape drive.

3. If you wish, mark up each line in the text, indicating the changes you want to make to the actual file.
4. Type the following:

```
# cd /usr/sys/sun[2,3,3x,4]/conf
```

   to go to the directory containing the GENERIC kernel configuration file.

5. Copy the file `GENERIC`. Call the new file *SYS_NAME*, where *SYS_NAME* represents the name you want to give to your system. Use the following command:

```
# cp GENERIC SYS_NAME
```

If your customized kernel is already in use and you now want to modify it, you should copy the customized kernel configuration file and edit the copy.

6. Change the permissions for *SYS_NAME* as follows:

```
# chmod +w SYS_NAME
```

7. Edit *SYS_NAME* using your preferred text editor. Use the notes you made as a guide while you make these changes. Make sure to include the proper device description lines for your machine.

Now you are ready to begin the reconfiguration process. Go on to the next major section, "Procedures for Reconfiguring the Kernel."

The Sun-2 GENERIC Kernel

The following is the GENERIC configuration file for a Sun-2 system.

```
#
# @(#)GENERIC 2.68 88/09/12 SMI
#
# This config file describes a generic Sun-2 kernel, including all
# possible standard devices and software options.
#
# The following lines include support for all Sun-2 cpu types.
# There is little to be gained by removing support for particular
# cpu's, so you may as well leave them all in.
#
machine     "sun2"
cpu     "SUN2_120"  # Sun-1/100U, Sun-1/150U, Sun-2/120, Sun-2/170
cpu     "SUN2_50"   # Sun-2/50, Sun-2/160
#
# Name this kernel "GENERIC".
#
ident       GENERIC
#
# This kernel supports about eight users.  Count one
# user for each timesharing user, one for each window
# that you typically use, and one for each diskless
# client you serve.  This is only an approximation
# used to control the size of various kernel data
# structures, not a hard limit.
#
maxusers    8

#
# Include all possible software options.
#
# The INET option is not really optional, every kernel must include it.
#
options     INET        # basic networking support - mandatory
```

```
#
# The following options are all filesystem related.  You only need
# QUOTA if you have UFS.  You only need UFS if you have a disk.
# Diskless machines can remove QUOTA, UFS, and NFSSERVER.  LOFS is
# only needed if you're using the Sun Network Software Environment.
#
options     QUOTA       # disk quotas for local disks
options     UFS     # filesystem code for local disks
options     NFSCLIENT   # NFS client side code
options     NFSSERVER   # NFS server side code
options     LOFS        # loopback filesystem - needed by NSE
#
# The following options are for accounting and auditing.  SYSAUDIT
# should be removed unless you are using the C2 security features.
#
options     SYSACCT     # process accounting, see acct(2) & sa(8)
options     SYSAUDIT    # C2 auditing for security
#
# The following options are for various System V IPC facilities.
# No standard software needs them, although some third party
# software relies on at least IPCSHMEM.
#
options     IPCMESSAGE  # System V IPC message facility
options     IPCSEMAPHORE    # System V IPC semaphore facility
options     IPCSHMEM    # System V IPC shared-memory facility
#
# The following option is only needed if you want to use the trpt
# command to debug TCP problems.
#
options     TCPDEBUG    # TCP debugging, see trpt(8)
#
# The following option includes the software DES support, needed if
# you're using secure NFS or secure RPC and you don't have a DES chip.
#
options     CRYPT       # software encryption (if no DES chip)

#
# Build one kernel based on this basic configuration.
# It will use the generic swap code so that you can have
# your root filesystem and swap space on any supported device.
# Put the kernel configured this way in a file named "vmunix".
#
config      vmunix      swap generic

#
# Include support for all possible pseudo-devices.
#
# The first few are mostly concerned with networking.
# You should probably always leave these in.
#
pseudo-device   pty     # pseudo-tty's, also needed for SunView
pseudo-device   ether       # basic Ethernet support
pseudo-device   loop        # loopback network - mandatory
```

```
#
# The next few are for SunWindows support, needed to run SunView 1.
#
pseudo-device   win128      # window devices, allow 128 windows
pseudo-device   dtop4       # desktops (screens), allow 4
pseudo-device   ms3     # mouse support, allow 3 mice
#
# The following is needed to support the Sun keyboard, with or
# without the window system.
#
pseudo-device   kb3     # keyboard support, allow 3 keyboards
#
# The following is needed to support the Sun dialbox.
#
pseudo-device   db              # dialbox support
#
# The following is for asynchronous tty support for the ALM-2 (aka MCP).
# If you have an ALM-2 (MCP) and it is being used to connect timesharing
# terminals, you will need this.
#
#pseudo-device  mcpa64
#
# The following is for the streams pipe device.  Currently nothing
# depends on this device so it is entirely optional.
#
pseudo-device   sp
#
# The following are for streams NIT support.  NIT is used by
# etherfind, traffic, rarpd, and ndbootd.  As a rule of thumb,
# NIT is almost always needed on a server and almost never
# needed on a diskless client.
#
pseudo-device   snit        # streams NIT
pseudo-device   pf      # packet filter
pseudo-device   nbuf        # NIT buffering module
#
# The following is for the "clone" device, used with streams devices.
# This is required if you include streams NIT support.
#
pseudo-device   clone

#
# The following sections describe what kinds of busses each
# cpu type supports.  You should never need to change this.
# (The word "nexus" is historical...)
#


# connections for machine type 1 (SUN2_120)
controller  virtual 1 at nexus ?    # virtually addressed devices
controller  obmem 1 at nexus ?  # memory-like devices on the cpu board
controller  obio 1 at nexus ?   # I/O devices on the cpu board
controller  mbmem 1 at nexus ?  # Multibus memory space
```

```
controller  mbio 1 at nexus ?   # Multibus I/O space

# connections for machine type 2 (SUN2_50)
controller  virtual 2 at nexus ?    # virtually addressed devices
controller  obmem 2 at nexus ?  # memory-like devices on the cpu board
controller  obio 2 at nexus ?   # I/O devices on the cpu board
controller  vme16 2 at nexus ?  # 16 bit address VMEbus (16 bit data)
controller  vme24 2 at nexus ?  # 24 bit address VMEbus (16 bit data)

#
# The following (large) section describes which standard devices this
# kernel supports.
#

#
# Support for 2 Xylogics 450/451 controllers with 2 drives each.
#
controller  xyc0 at mbio ? csr 0xee40 priority 2
controller  xyc0 at vme16 ? csr 0xee40 priority 2 vector xyintr 0x48
controller  xyc1 at mbio ? csr 0xee48 priority 2
controller  xyc1 at vme16 ? csr 0xee48 priority 2 vector xyintr 0x49
disk        xy0 at xyc0 drive 0
disk        xy1 at xyc0 drive 1
disk        xy2 at xyc1 drive 0
disk        xy3 at xyc1 drive 1
#
# Support for the SCSI-2 host adapter with 2 disks and 1 1/4" tape
# on the first SCSI controller and 1 disk and 1 1/4" tape on the
# second SCSI controller.
#
controller  sc0 at mbmem ? csr 0x80000 priority 2
controller  sc0 at vme24 ? csr 0x200000 priority 2 vector scintr 0x40
disk        sd0 at sc0 drive 0 flags 0
disk        sd1 at sc0 drive 1 flags 0
disk        sd2 at sc0 drive 8 flags 0
tape        st0 at sc0 drive 32 flags 1
tape        st1 at sc0 drive 40 flags 1
#disk       sf0 at sc0 drive 49 flags 2
#
# Support for the second SCSI-2 host adapter.
# Only supports one SCSI controller.
#
controller  sc1 at mbmem ? csr 0x84000 priority 2
disk        sd2 at sc1 drive 0 flags 0
disk        sd3 at sc1 drive 1 flags 0
tape        st1 at sc1 drive 32 flags 1
#disk       sf1 at sc1 drive 8 flags 2
#
# Support for the Sky floating point processor.
#
device      sky0 at mbio ? csr 0x2000 priority 2
device      sky0 at vme16 ? csr 0x8000 priority 2 vector skyintr 0xb0
#
```

```
# Support for the 2 tty lines (ttya, ttyb) on the cpu board.
# Needed when using a terminal for the console device.
# Flags=3 says to supply carrier in software for both lines.
#
device      zs0 at obio 1 csr 0x2000 flags 3 priority 3
device      zs0 at obio 2 csr 0x7f2000 flags 3 priority 3
#
# Support for the keyboard and mouse interface.  Needed when
# using a frame buffer as the console device or with SunView.
# You can remove this line if you don't use the standard Sun
# Workstation keyboard and mouse, but if you leave it in don't
# change it.
#
device      zs1 at obmem 1 csr 0x780000 flags 0x103 priority 3
device      zs1 at obio 2 csr 0x7f1800 flags 0x103 priority 3
#
# Support for tty lines on first Multibus SCSI-2 board.
#
device      zs2 at mbmem ? csr 0x80800 flags 3 priority 3
device      zs3 at mbmem ? csr 0x81000 flags 3 priority 3
#
# Support for tty lines on second Multibus SCSI-2 board.
#
device      zs4 at mbmem ? csr 0x84800 flags 3 priority 3
device      zs5 at mbmem ? csr 0x85000 flags 3 priority 3
#
# Support for 4 ALM's (Systech MTI-800/1600).  Flags set for
# all lines to be local, i.e., carrier supplied by software
# rather than by the device.
#
device      mti0 at mbio ? csr 0x620 flags 0xffff priority 4
device      mti1 at mbio ? csr 0x640 flags 0xffff priority 4
device      mti2 at mbio ? csr 0x660 flags 0xffff priority 4
device      mti3 at mbio ? csr 0x680 flags 0xffff priority 4
device      mti0 at vme16 ? csr 0x620 flags 0xffff priority 4
    vector mtiintr 0x88
device      mti1 at vme16 ? csr 0x640 flags 0xffff priority 4
    vector mtiintr 0x89
device      mti2 at vme16 ? csr 0x660 flags 0xffff priority 4
    vector mtiintr 0x8a
device      mti3 at vme16 ? csr 0x680 flags 0xffff priority 4
    vector mtiintr 0x8b
#
# Support for the on-board Intel 82586 Ethernet chip on the Sun-2/50.
#
device      ie0 at obio 2 csr 0x7f0800 priority 3
#
# Support for the first Multibus Intel Ethernet board.
#
device      ie0 at mbmem ? csr 0x88000 priority 3
#
# Support for the second Multibus Intel Ethernet board.
#
```

```
device      ie1 at mbmem ? csr 0x8c000 flags 2 priority 3
device      ie1 at vme24 ? csr 0xe88000 priority 3 vector ieintr 0x75
#
# Support for the first 3COM Ethernet board.
#
device      ec0 at mbmem ? csr 0xe0000 priority 3
#
# Support for the second 3COM Ethernet board.
#
device      ec1 at mbmem ? csr 0xe2000 priority 3
#
# Support for 2 Ciprico TapeMaster tape controllers with 1 tape drive each.
#
controller  tm0 at mbio ? csr 0xa0 priority 3
controller  tm0 at vme16 ? csr 0xa0 priority 3 vector tmintr 0x60
controller  tm1 at mbio ? csr 0xa2 priority 3
controller  tm1 at vme16 ? csr 0xa2 priority 3 vector tmintr 0x61
tape        mt0 at tm0 drive 0 flags 1
tape        mt1 at tm1 drive 0 flags 1
#
# Support for 2 Xylogics 472 tape controllers with 1 tape drive each.
#
controller  xtc0 at mbio ? csr 0xee60 priority 3
controller  xtc0 at vme16 ? csr 0xee60 priority 3 vector xtintr 0x64
controller  xtc1 at mbio ? csr 0xee68 priority 3
controller  xtc1 at vme16 ? csr 0xee68 priority 3 vector xtintr 0x65
tape        xt0 at xtc0 drive 0 flags 1
tape        xt1 at xtc1 drive 0 flags 1
#
# Support for 2 Sun Archive 1/4" tape controller boards.
#
device      ar0 at mbio ? csr 0x200 priority 3
device      ar1 at mbio ? csr 0x208 priority 3
#
# Support for the GP/GP+/GP2 graphics processors.
# Requires cgtwo as well.
#
device      gpone0 at vme24 ? csr 0x210000
#
# Support for either the Sun-2 color board, Sun-3 color board,
# or GP2 frame buffer.
#
device      cgtwo0 at vme24 ? csr 0x400000 priority 4 vector cgtwointr 0xa8
#
# Support for the Sun-1 color board.
#
device      cgone0 at mbmem ? csr 0xec000 priority 3
#
# Support for monochrome memory frame buffers on various machines.
#
device      bwtwo0 at obmem 1 csr 0x700000 priority 4   # 2/120
device      bwtwo0 at obio 2 csr 0x0 priority 4     # 2/50
device      bwone0 at mbmem ? csr 0xc0000 priority 3    # 1/100U
```

```
#
# Support for the Ikon Versatec printer controller.
#
device      vp0 at mbio ? csr 0x400 priority 2
#
# Support for 2 Systech VPC-2200 line printer controllers.
#
device      vpc0 at mbio ? csr 0x480 priority 2
device      vpc0 at vme16 ? csr 0x480 priority 2 vector vpcintr 0x80
device      vpc1 at mbio ? csr 0x500 priority 2
device      vpc1 at vme16 ? csr 0x500 priority 2 vector vpcintr 0x81
#
# Support for the parallel keyboard/mouse interface on the Sun-2/120
# cpu board.  Required if using the Sun-1 style parallel keyboard or
# mouse.
#
device      pi0 at obio 1 csr 0x1800
#
# Support for the hardware Data Ciphering Processor (aka the DES chip).
# Suggested if you make heavy use of secure RPC or secure NFS.
#
device      des0 at obio 1 csr 0x1000
device      des0 at obio 2 csr 0x7f1000
#
# Support for the time-of-day clock on the Sun-2/120 CPU board.
#
device      tod0 at obio 1 csr 0x3800
#
# Support for the time-of-day clock on the VME Sun-2 SCSI board.
#
device      tod0 at vme24 ? csr 0x200800
```

The Sun-3 GENERIC Kernel

The following is the GENERIC configuration file for a Sun-3.

```
#
# @(#)GENERIC 1.92 88/11/09 SMI
#
# This config file describes a generic Sun-3 kernel, including all
# possible standard devices and software options.
#
# The following lines include support for all Sun-3 cpu types.
# There is little to be gained by removing support for particular
# cpu's, so you may as well leave them all in.
#
machine     "sun3"
cpu     "SUN3_160"  # Sun-3/75, Sun-3/140, Sun-3/160, or Sun-3/180
cpu     "SUN3_50"   # Sun-3/50
cpu     "SUN3_260"  # Sun-3/260 or Sun-3/280
cpu     "SUN3_110"  # Sun-3/110
cpu     "SUN3_60"   # Sun-3/60
cpu     "SUN3_E"    # Sun-3E (Eurocard VMEbus cpu)
#
# Name this kernel "GENERIC".
#
ident       GENERIC
#
# This kernel supports about eight users.  Count one
# user for each timesharing serial port, one for each window
# that you typically use, and one for each diskless
# client you serve.  This is only an approximation
# used to control the size of various kernel data
# structures, not a hard limit.
#
maxusers    8

#
# Include all possible software options.
#
# The INET option is not really optional, every kernel must include it.
#
options     INET        # basic networking support - mandatory
#
# The following options are all filesystem related.  You only need
# QUOTA if you have UFS.  You only need UFS if you have a disk.
# Diskless machines can remove QUOTA, UFS, and NFSSERVER.  LOFS is
# only needed if you're using the Sun Network Software Environment.
#
options     QUOTA       # disk quotas for local disks
options     UFS     # filesystem code for local disks
options     NFSCLIENT   # NFS client side code
options     NFSSERVER   # NFS server side code
options     LOFS        # loopback filesystem - needed by NSE
#
```

```
# The following options are for accounting and auditing.  SYSAUDIT
# should be removed unless you are using the C2 security features.
#
options     SYSACCT     # process accounting, see acct(2) & sa(8)
options     SYSAUDIT    # C2 auditing for security
#
# The following options are for various System V IPC facilities.
# No standard software needs them, although some third party
# software relies on at least IPCSHMEM.
#
options     IPCMESSAGE  # System V IPC message facility
options     IPCSEMAPHORE    # System V IPC semaphore facility
options     IPCSHMEM    # System V IPC shared-memory facility
#
# The following option is only needed if you want to use the trpt
# command to debug TCP problems.
#
options     TCPDEBUG    # TCP debugging, see trpt(8)
#
# The following option includes the software DES support, needed if
# you're using secure NFS or secure RPC and you don't have a DES chip.
#
options     CRYPT       # software encryption (if no DES chip)

#
# Build one kernel based on this basic configuration.
# It will use the generic swap code so that you can have
# your root filesystem and swap space on any supported device.
# Put the kernel configured this way in a file named "vmunix".
#
config      vmunix      swap generic

#
# Include support for all possible pseudo-devices.
#
# The first few are mostly concerned with networking.
# You should probably always leave these in.
#
pseudo-device   pty     # pseudo-tty's, also needed for SunView
pseudo-device   ether       # basic Ethernet support
pseudo-device   loop        # loopback network - mandatory
#
# The next few are for SunWindows support, needed to run SunView 1.
#
pseudo-device   win128      # window devices, allow 128 windows
pseudo-device   dtop4       # desktops (screens), allow 4
pseudo-device   ms3     # mouse support, allow 3 mice
#
# The following is needed to support the Sun keyboard, with or
# without the window system.
#
pseudo-device   kb3     # keyboard support, allow 3 keyboards
#
```

```
# The following is needed to support the Sun dialbox.
#
pseudo-device   db              # dialbox support
#
# The following is for asynchronous tty support for the ALM-2 (aka MCP).
# If you have an ALM-2 (MCP) and it is being used to connect timesharing
# terminals, you will need this.  The number appended to mcpa should be
# the total number of serial lines provided by the ALM-2s in the system.
# For example, if you had eight ALM-2s this should read "mcpa128".
#
pseudo-device   mcpa64
#
# The following is for the streams pipe device.  Currently nothing
# depends on this device so it is entirely optional.
#
pseudo-device   sp
#
# The following are for streams NIT support.  NIT is used by
# etherfind, traffic, rarpd, and ndbootd.  As a rule of thumb,
# NIT is almost always needed on a server and almost never
# needed on a diskless client.
#
pseudo-device   snit        # streams NIT
pseudo-device   pf      # packet filter
pseudo-device   nbuf        # NIT buffering module
#
# The following is for the "clone" device, used with streams devices.
# This is required if you include streams NIT support.
#
pseudo-device   clone

#
# The following sections describe what kinds of busses each
# cpu type supports.  You should never need to change this.
# (The word "nexus" is historical...)
#

# connections for machine type 1 (SUN3_160)
controller  virtual 1 at nexus ?    # virtually addressed devices
controller  obmem 1 at nexus ?  # memory-like devices on the cpu board
controller  obio 1 at nexus ?   # I/O devices on the cpu board
controller  vme16d16 1 at nexus ?   # VME 16 bit address 16 bit data devices
controller  vme24d16 1 at nexus ?   # VME 24 bit address 16 bit data devices
controller  vme32d16 1 at nexus ?   # VME 32 bit address 16 bit data devices
controller  vme16d32 1 at nexus ?   # VME 16 bit address 32 bit data devices
controller  vme24d32 1 at nexus ?   # VME 24 bit address 32 bit data devices
controller  vme32d32 1 at nexus ?   # VME 32 bit address 32 bit data devices

# connections for machine type 2 (SUN3_50)
controller  virtual 2 at nexus ?
controller  obmem 2 at nexus ?
controller  obio 2 at nexus ?
```

```
# connections for machine type 3 (SUN3_260)
controller  virtual 3 at nexus ?
controller  obmem 3 at nexus ?
controller  obio 3 at nexus ?
controller  vme16d16 3 at nexus ?
controller  vme24d16 3 at nexus ?
controller  vme32d16 3 at nexus ?
controller  vme16d32 3 at nexus ?
controller  vme24d32 3 at nexus ?
controller  vme32d32 3 at nexus ?

# connections for machine type 4 (SUN3_110)
controller  virtual 4 at nexus ?
controller  obmem 4 at nexus ?
controller  obio 4 at nexus ?
controller  vme16d16 4 at nexus ?
controller  vme24d16 4 at nexus ?
controller  vme32d16 4 at nexus ?
controller  vme16d32 4 at nexus ?
controller  vme24d32 4 at nexus ?
controller  vme32d32 4 at nexus ?

# connections for machine type 7 (SUN3_60)
controller  virtual 7 at nexus ?
controller  obmem 7 at nexus ?
controller  obio 7 at nexus ?

# connections for machine type 8 (SUN3_E)
controller      virtual 8 at nexus ?
controller      obmem 8 at nexus ?
controller      obio 8 at nexus ?
controller      vme16d16 8 at nexus ?
controller      vme24d16 8 at nexus ?
controller      vme32d16 8 at nexus ?
controller      vme16d32 8 at nexus ?
controller      vme24d32 8 at nexus ?
controller      vme32d32 8 at nexus ?

#
# The following (large) section describes which standard devices this
# kernel supports.
#

#
# Support for 4 Xylogics 7053 controllers with 4 drives each.
#
controller      xdc0 at vme16d32 ? csr 0xee80 priority 2 vector xdintr 0x44
controller      xdc1 at vme16d32 ? csr 0xee90 priority 2 vector xdintr 0x45
controller      xdc2 at vme16d32 ? csr 0xeea0 priority 2 vector xdintr 0x46
controller      xdc3 at vme16d32 ? csr 0xeeb0 priority 2 vector xdintr 0x47
disk        xd0 at xdc0 drive 0
disk        xd1 at xdc0 drive 1
disk        xd2 at xdc0 drive 2
```

```
disk        xd3 at xdc0 drive 3
disk        xd4 at xdc1 drive 0
disk        xd5 at xdc1 drive 1
disk        xd6 at xdc1 drive 2
disk        xd7 at xdc1 drive 3
disk        xd8 at xdc2 drive 0
disk        xd9 at xdc2 drive 1
disk        xd10 at xdc2 drive 2
disk        xd11 at xdc2 drive 3
disk        xd12 at xdc3 drive 0
disk        xd13 at xdc3 drive 1
disk        xd14 at xdc3 drive 2
disk        xd15 at xdc3 drive 3
#
# Support for 2 Xylogics 450/451 controllers with 2 drives each.
#
controller  xyc0 at vme16d16 ? csr 0xee40 priority 2 vector xyintr 0x48
controller  xyc1 at vme16d16 ? csr 0xee48 priority 2 vector xyintr 0x49
disk        xy0 at xyc0 drive 0
disk        xy1 at xyc0 drive 1
disk        xy2 at xyc1 drive 0
disk        xy3 at xyc1 drive 1
#
# Support for the SCSI-2 host adapter with 2 disks and 1 1/4" tape
# on the first SCSI controller, and 2 disks and 1 1/4" tape on the
# second SCSI controller.
#
controller  sc0 at vme24d16 ? csr 0x200000 priority 2 vector scintr 0x40
tape        st0 at sc0 drive 32 flags 1
tape        st1 at sc0 drive 40 flags 1
disk        sf0 at sc0 drive 49 flags 2
disk        sd0 at sc0 drive 0 flags 0
disk        sd1 at sc0 drive 1 flags 0
disk        sd2 at sc0 drive 8 flags 0
disk        sd3 at sc0 drive 9 flags 0
disk        sd4 at sc0 drive 16 flags 0
disk        sd6 at sc0 drive 24 flags 0
#
# Support for the SCSI-3 host adapter and the on-board SCSI controller
# on several machines (e.g. 3/50).  Same device support as above.
#
controller  si0 at vme24d16 ? csr 0x200000 priority 2 vector siintr 0x40
controller  si1 at vme24d16 ? csr 0x204000 priority 2 vector siintr 0x41
controller  si0 at obio ? csr 0x140000 priority 2
tape        st0 at si0 drive 32 flags 1
tape        st1 at si0 drive 40 flags 1
tape        st2 at si1 drive 32 flags 1
tape        st3 at si1 drive 40 flags 1
disk        sf0 at si0 drive 49 flags 2
disk        sd0 at si0 drive 0 flags 0
disk        sd1 at si0 drive 1 flags 0
disk        sd2 at si0 drive 8 flags 0
disk        sd3 at si0 drive 9 flags 0
```

```
disk        sd4 at si0 drive 16 flags 0
disk        sd6 at si0 drive 24 flags 0
#
# Support for the SCSI-E host adapter used with the Sun-3/E.
# Same device support as above.
#
controller      se0 at vme24d16 ? csr 0x300000 priority 2 vector se_intr 0x40
tape        st0 at se0 drive 32 flags 1
tape        st1 at se0 drive 40 flags 1
disk        sd0 at se0 drive 0 flags 0
disk        sd1 at se0 drive 1 flags 0
disk        sd2 at se0 drive 8 flags 0
disk        sd3 at se0 drive 9 flags 0
#disk       sd4 at se0 drive 16 flags 0
#disk       sd6 at se0 drive 24 flags 0
#
# Support for the 2 tty lines (ttya, ttyb) on the cpu board.
# Needed when using a terminal for the console device.
# Flags=3 says to supply carrier in software for both lines.
#
device      zs0 at obio ? csr 0x20000 flags 3 priority 3
#
# Support for the keyboard and mouse interface.  Needed when
# using a frame buffer as the console device or with SunView.
# You can remove this line if you don't use the standard Sun
# Workstation keyboard and mouse, but if you leave it in don't
# change it.
#
device      zs1 at obio ? csr 0x00000 flags 0x103 priority 3
#
# Support for 4 ALM's (Systech MTI-800/1600).  Flags set for
# all lines to be local, i.e., carrier supplied by software
# rather than by the device.
#
device      mti0 at vme16d16 ? csr 0x620 flags 0xffff priority 4
    vector mtiintr 0x88
device      mti1 at vme16d16 ? csr 0x640 flags 0xffff priority 4
    vector mtiintr 0x89
device      mti2 at vme16d16 ? csr 0x660 flags 0xffff priority 4
    vector mtiintr 0x8a
device      mti3 at vme16d16 ? csr 0x680 flags 0xffff priority 4
    vector mtiintr 0x8b
#
# Support for 8 MCP boards.
# Note that the first four MCP's use the same vectors as the ALM's and thus
# ALM's cut into the total number of MCP's that can installed.
# Make sure the maxusers line above has at least one added to it for
# each serial port.
#
device      mcp0 at vme32d32 ? csr 0x01000000 flags 0x1ffff priority 4
    vector mcpintr 0x8b
device      mcp1 at vme32d32 ? csr 0x01010000 flags 0x1ffff priority 4
    vector mcpintr 0x8a
```

```
device      mcp2 at vme32d32 ? csr 0x01020000 flags 0x1ffff priority 4
    vector mcpintr 0x89
device      mcp3 at vme32d32 ? csr 0x01030000 flags 0x1ffff priority 4
    vector mcpintr 0x88
device      mcp4 at vme32d32 ? csr 0x01040000 flags 0x1ffff priority 4
    vector mcpintr 0xa0
device      mcp5 at vme32d32 ? csr 0x01050000 flags 0x1ffff priority 4
    vector mcpintr 0xa1
device      mcp6 at vme32d32 ? csr 0x01060000 flags 0x1ffff priority 4
    vector mcpintr 0xa2
device      mcp7 at vme32d32 ? csr 0x01070000 flags 0x1ffff priority 4
    vector mcpintr 0xa3
#
# Start of Network Interface Declarations
#
# N.B.: Diskless operation is only supported on the 1st existing network
#   interface in the following list. It must be reordered to use a
#   different interface.
#

#
# Support for the on-board Intel 82586 Ethernet chip on many machines
# (all but 3/50, 3/60, and 3/E).
#
device      ie0 at obio ? csr 0xc0000 priority 3
#
# Support for the Sun-3/E Intel Ethernet board.
#
device      ie0 at vme24d16 ? csr 0x31ff02 priority 3 vector ieintr 0x74
#
# Support for a second Intel Ethernet board, either a second
# Sun-3/E board or a Multibus Ethernet board used with a
# Multibus-to-VME adapter.  Used for Ethernet to Ethernet
# gateways.
#
device      ie1 at vme24d16 ? csr 0xe88000 priority 3 vector ieintr 0x75
#
# Support for the on-board LANCE Ethernet chip on the 3/50 and 3/60.
#
device      le0 at obio ? csr 0x120000 priority 3

#
# End of Network Interface Declarations
#

#
# Support for 2 Ciprico TapeMaster tape controllers with 1 tape drive each.
#
controller  tm0 at vme16d16 ? csr 0xa0 priority 3 vector tmintr 0x60
controller  tm1 at vme16d16 ? csr 0xa2 priority 3 vector tmintr 0x61
tape        mt0 at tm0 drive 0 flags 1
tape        mt1 at tm1 drive 0 flags 1
#
```

```
# Support for 2 Xylogics 472 tape controllers with 1 tape drive each.
#
controller  xtc0 at vme16d16 ? csr 0xee60 priority 3 vector xtintr 0x64
controller  xtc1 at vme16d16 ? csr 0xee68 priority 3 vector xtintr 0x65
tape        xt0 at xtc0 drive 0 flags 1
tape        xt1 at xtc1 drive 0 flags 1
#
# Support for the GP/GP+/GP2 graphics processors.
# Requires cgtwo as well.
#
device      gpone0 at vme24d16 ? csr 0x210000   # GP or GP+
device      gpone0 at vme24d32 ? csr 0x240000   # GP2
#
# Support for either the Sun-2 color board, Sun-3 color board,
# or GP2 frame buffer.
#
device      cgtwo0 at vme24d16 ? csr 0x400000 priority 4
            vector cgtwointr 0xa8
#
# Support for color memory frame buffers on various machine types.
#
# 3/110 on-board frame buffer
device      cgfour0 at obmem 4 csr 0xff000000 priority 4    # 3/110
# 3/60 P4 color frame buffer
device      cgfour0 at obmem 7 csr 0xff300000 priority 4    # 3/60
# 3/60 plug-in color frame buffer
device      cgfour0 at obmem 7 csr 0xff400000 priority 4    # 3/60
#
# Support for monochrome memory frame buffers on various machines.
#
device      bwtwo0 at obmem 1 csr 0xff000000 priority 4 # 3/160
device      bwtwo0 at obmem 2 csr 0x100000 priority 4   # 3/50
device      bwtwo0 at obmem 3 csr 0xff000000 priority 4 # 3/260
# 3/110 on-board frame buffer overlay plane
device      bwtwo0 at obmem 4 csr 0xff000000        # 3/110
device      bwtwo0 at obmem 7 csr 0xff000000 priority 4 # 3/60
device      bwtwo0 at obmem 8 csr 0x1000000         # 3/E
# 3/60 P4 color frame buffer overlay plane, or P4 monochrome frame buffer
device      bwtwo1 at obmem 7 csr 0xff300000 priority 4 # 3/60
# 3/60 plug-in color frame buffer overlay plane
device      bwtwo1 at obmem 7 csr 0xff400000        # 3/60
# 3/60 P4 24-bit color frame buffer
device          cgeight0 at obmem 7 csr 0xff300000 priority 4   # 3/60
# 3/60 P4 accelerated 8-bit color frame buffer
device      cgsix0 at obmem 7 csr 0xff000000 priority 4

#
# Support for the TAAC-1 Application Accelerator.
#
device      taac0 at vme32d32 ? csr 0x28000000
#
# Support for 2 Systech VPC-2200 line printer controllers.
#
```

```
device      vpc0 at vme16d16 ? csr 0x480 priority 2 vector vpcintr 0x80
device      vpc1 at vme16d16 ? csr 0x500 priority 2 vector vpcintr 0x81
#
# Support for the hardware Data Ciphering Processor (aka the DES chip).
# Suggested if you make heavy use of secure RPC or secure NFS.
#
device      des0 at obio ? csr 0x1c0000
#
# Support for the Floating Point Accelerator.
#
device      fpa0 at virtual ? csr 0xe0000000
```

The Sun-3x GENERIC Kernel

The following is a GENERIC configuration file for a Sun-3x.

```
#
# @(#)GENERIC 1.26 89/02/23 SMI
#
# This config file describes a generic Sun-3x kernel, including all
# possible standard devices and software options.
#
# The following lines include support for all Sun-3x cpu types.
# There is little to be gained by removing support for particular
# cpu's, so you may as well leave them all in.
#
machine     "sun3x"
cpu     "SUN3X_470" # Sun-3x/470, Sun-3x/480, or Sun-3x/460
cpu     "SUN3X_80"      # Sun-3x/80
#
# Name this kernel "GENERIC".
#
ident       GENERIC
#
# This kernel supports about eight users.  Count one
# user for each timesharing serial port, one for each window
# that you typically use, and one for each diskless
# client you serve.  This is only an approximation
# used to control the size of various kernel data
# structures, not a hard limit.
#
maxusers    8

#
# Include all possible software options.
#
# The INET option is not really optional, every kernel must include it.
#
options     INET        # basic networking support - mandatory
#
# The following options are all filesystem related.  You only need
# QUOTA if you have UFS.  You only need UFS if you have a disk.
# Diskless machines can remove QUOTA, UFS, and NFSSERVER.  LOFS is
# only needed if you're using the Sun Network Software Environment.
#
options     QUOTA       # disk quotas for local disks
options     UFS     # filesystem code for local disks
options     NFSCLIENT   # NFS client side code
options     NFSSERVER   # NFS server side code
options     LOFS        # loopback filesystem - needed by NSE
#
# The following options are for accounting and auditing.  SYSAUDIT
# should be removed unless you are using the C2 security features.
#
options     SYSACCT     # process accounting, see acct(2) & sa(8)
```

```
options     SYSAUDIT    # C2 auditing for security
#
# The following options are for various System V IPC facilities.
# No standard software needs them, although some third party
# software relies on at least IPCSHMEM.
#
options     IPCMESSAGE  # System V IPC message facility
options     IPCSEMAPHORE    # System V IPC semaphore facility
options     IPCSHMEM    # System V IPC shared-memory facility
#
# The following option is only needed if you want to use the trpt
# command to debug TCP problems.
#
options     TCPDEBUG    # TCP debugging, see trpt(8)
#
# The following option includes the software DES support, needed if
# you're using secure NFS or secure RPC and you don't have a DES chip.
#
options     CRYPT       # software encryption (if no DES chip)

#
# Build one kernel based on this basic configuration.
# It will use the generic swap code so that you can have
# your root filesystem and swap space on any supported device.
# Put the kernel configured this way in a file named "vmunix".
#
config      vmunix      swap generic

#
# Include support for all possible pseudo-devices.
#
# The first few are mostly concerned with networking.
# You should probably always leave these in.
#
pseudo-device   pty     # pseudo-tty's, also needed for SunView
pseudo-device   ether       # basic Ethernet support
pseudo-device   loop        # loopback network - mandatory
#
# The next few are for SunWindows support, needed to run SunView 1.
#
pseudo-device   win128      # window devices, allow 128 windows
pseudo-device   dtop4       # desktops (screens), allow 4
pseudo-device   ms3     # mouse support, allow 3 mice
#
# The following is needed to support the Sun keyboard, with or
# without the window system.
#
pseudo-device   kb3     # keyboard support, allow 3 keyboards
#
# The following is needed to support the Sun dialbox.
#
pseudo-device   db              # dialbox support
#
```

```
# The following is for asynchronous tty support for the ALM-2 (aka MCP).
# If you have an ALM-2 (MCP) and it is being used to connect timesharing
# terminals, you will need this.  The number appended to mcpa should be
# the total number of serial lines provided by the ALM-2s in the system.
# For example, if you had eight ALM-2s this should read "mcpa128".
#
pseudo-device   mcpa64
#
# The following is for the streams pipe device.  Currently nothing
# depends on this device so it is entirely optional.
#
pseudo-device   sp
#
# The following are for streams NIT support.  NIT is used by
# etherfind, traffic, rarpd, and ndbootd.  As a rule of thumb,
# NIT is almost always needed on a server and almost never
# needed on a diskless client.
#
pseudo-device   snit        # streams NIT
pseudo-device   pf      # packet filter
pseudo-device   nbuf        # NIT buffering module
#
# The following is for the "clone" device, used with streams devices.
# This is required if you include streams NIT support.
#
pseudo-device   clone

#
# The following sections describe what kinds of busses each
# cpu type supports.  You should never need to change this.
# (The word "nexus" is historical...)
#

# connections for machine type 1 (SUN3X_470)
controller  virtual 1 at nexus ?    # virtually addressed devices
controller  obmem 1 at nexus ?  # memory-like devices on the cpu board
controller  obio 1 at nexus ?   # I/O devices on the cpu board
controller  vme16d16 1 at nexus ?   # VME 16 bit address 16 bit data devices
controller  vme16d32 1 at nexus ?   # VME 16 bit address 32 bit data devices
controller  vme24d16 1 at nexus ?   # VME 24 bit address 16 bit data devices
controller  vme24d32 1 at nexus ?   # VME 24 bit address 32 bit data devices
controller  vme32d32 1 at nexus ?   # VME 32 bit address 32 bit data devices

# connections for machine type 2 (SUN3X_80)
controller      virtual 2 at nexus ?
controller      obmem 2 at nexus ?
controller      obio 2 at nexus ?

#
# The following (large) section describes which standard devices this
# kernel supports.
#
```

```
#
# Support for 4 Xylogics 7053 controllers with 4 drives each.
#
controller      xdc0 at vme16d32 ? csr 0xee80 priority 2 vector xdintr 0x44
controller      xdc1 at vme16d32 ? csr 0xee90 priority 2 vector xdintr 0x45
controller      xdc2 at vme16d32 ? csr 0xeea0 priority 2 vector xdintr 0x46
controller      xdc3 at vme16d32 ? csr 0xeeb0 priority 2 vector xdintr 0x47
disk        xd0 at xdc0 drive 0
disk        xd1 at xdc0 drive 1
disk        xd2 at xdc0 drive 2
disk        xd3 at xdc0 drive 3
disk        xd4 at xdc1 drive 0
disk        xd5 at xdc1 drive 1
disk        xd6 at xdc1 drive 2
disk        xd7 at xdc1 drive 3
disk        xd8 at xdc2 drive 0
disk        xd9 at xdc2 drive 1
disk        xd10 at xdc2 drive 2
disk        xd11 at xdc2 drive 3
disk        xd12 at xdc3 drive 0
disk        xd13 at xdc3 drive 1
disk        xd14 at xdc3 drive 2
disk        xd15 at xdc3 drive 3
#
# Support for 2 Xylogics 450/451 controllers with 2 drives each.
#
controller  xyc0 at vme16d16 ? csr 0xee40 priority 2 vector xyintr 0x48
controller  xyc1 at vme16d16 ? csr 0xee48 priority 2 vector xyintr 0x49
disk        xy0 at xyc0 drive 0
disk        xy1 at xyc0 drive 1
disk        xy2 at xyc1 drive 0
disk        xy3 at xyc1 drive 1
#
# Support for the SCSI-2 host adapter with 2 disks and 1 1/4" tape
# on the first SCSI controller, and 2 disks and 1 1/4" tape on the
# second SCSI controller.
#
controller  sc0 at vme24d16 ? csr 0x200000 priority 2 vector scintr 0x40
tape        st0 at sc0 drive 32 flags 1
tape        st1 at sc0 drive 40 flags 1
#disk       sf0 at sc0 drive 49 flags 2
disk        sd0 at sc0 drive 0 flags 0
disk        sd1 at sc0 drive 1 flags 0
disk        sd2 at sc0 drive 8 flags 0
disk        sd3 at sc0 drive 9 flags 0
disk            sd4 at sc0 drive 16 flags 0
disk            sd6 at sc0 drive 24 flags 0
#
# Support for the SCSI-3 host adapter. Same device support as above.
#
controller  si0 at vme24d16 ? csr 0x200000 priority 2 vector siintr 0x40
controller      si1 at vme24d16 ? csr 0x204000 priority 2 vector siintr 0x41
tape        st0 at si0 drive 32 flags 1
```

```
tape        st1 at si0 drive 40 flags 1
tape            st2 at si1 drive 32 flags 1
tape            st3 at si1 drive 40 flags 1
#disk       sf0 at si0 drive 49 flags 2
disk        sd0 at si0 drive 0 flags 0
disk        sd1 at si0 drive 1 flags 0
disk        sd2 at si0 drive 8 flags 0
disk        sd3 at si0 drive 9 flags 0
disk            sd4 at si0 drive 16 flags 0
disk            sd6 at si0 drive 24 flags 0
#
# Support for the SCSI-ESP host adapter for 3/80
#
controller  sm0 at obio ? csr 0x66000000 priority 2
tape        st0 at sm0 drive 32 flags 1
tape        st1 at sm0 drive 40 flags 1
#disk       sf0 at sm0 drive 49 flags 2
disk        sd0 at sm0 drive 0 flags 0
disk        sd1 at sm0 drive 1 flags 0
disk        sd2 at sm0 drive 8 flags 0
disk        sd3 at sm0 drive 9 flags 0
disk            sd4 at sm0 drive 16 flags 0
disk            sd6 at sm0 drive 24 flags 0
#
# Support for the 2 tty lines (ttya, ttyb) on the cpu board.
# Needed when using a terminal for the console device.
# Flags=3 says to supply carrier in software for both lines.
#
device      zs0 at obio ? csr 0x62002000 flags 3 priority 3
#
# Support for the keyboard and mouse interface.  Needed when
# using a frame buffer as the console device or with SunView.
# You can remove this line if you don't use the standard Sun
# Workstation keyboard and mouse, but if you leave it in don't
# change it.
#
device      zs1 at obio ? csr 0x62000000 flags 0x103 priority 3
#
# Support for 4 ALM's (Systech MTI-800/1600).  Flags set for
# all lines to be local, i.e., carrier supplied by software
# rather than by the device.
#
device      mti0 at vme16d16 ? csr 0x620 flags 0xffff priority 4
    vector mtiintr 0x88
device      mti1 at vme16d16 ? csr 0x640 flags 0xffff priority 4
    vector mtiintr 0x89
device      mti2 at vme16d16 ? csr 0x660 flags 0xffff priority 4
    vector mtiintr 0x8a
device      mti3 at vme16d16 ? csr 0x680 flags 0xffff priority 4
    vector mtiintr 0x8b
#
# Support for 8 MCP boards.
# Note that the first four MCP's use the same vectors as the ALM's and thus
```

```
# ALM's cut into the total number of MCP's that can installed.
# Make sure the maxusers line above has at least one added to it for
# each serial port.
#
device      mcp0 at vme32d32 ? csr 0x01000000 flags 0x1ffff priority 4
    vector mcpintr 0x8b
device      mcp1 at vme32d32 ? csr 0x01010000 flags 0x1ffff priority 4
    vector mcpintr 0x8a
device      mcp2 at vme32d32 ? csr 0x01020000 flags 0x1ffff priority 4
    vector mcpintr 0x89
device      mcp3 at vme32d32 ? csr 0x01030000 flags 0x1ffff priority 4
    vector mcpintr 0x88
device          mcp4 at vme32d32 ? csr 0x01040000 flags 0x1ffff priority 4
        vector mcpintr 0xa0
device          mcp5 at vme32d32 ? csr 0x01050000 flags 0x1ffff priority 4
        vector mcpintr 0xa1
device          mcp6 at vme32d32 ? csr 0x01060000 flags 0x1ffff priority 4
        vector mcpintr 0xa2
device          mcp7 at vme32d32 ? csr 0x01070000 flags 0x1ffff priority 4
        vector mcpintr 0xa3
#
# Support for the on-board Intel 82586 Ethernet chip
#
device      ie0 at obio ? csr 0x65000000 priority 3
#
# Support for a second Intel Ethernet board, either a second
# Sun-3/E board or a Multibus Ethernet board used with a
# Multibus-to-VME adapter.  Used for Ethernet to Ethernet
# gateways.
#
device      ie1 at vme24d16 ? csr 0xe88000 priority 3 vector ieintr 0x75
#
#
# Support for the on-board LANCE Ethernet (AM7990).
#
device      le0 at obio ? csr 0x65002000 priority 3
# Support for 2 Ciprico TapeMaster tape controllers with 1 tape drive each.
#
controller  tm0 at vme16d16 ? csr 0xa0 priority 3 vector tmintr 0x60
controller  tm1 at vme16d16 ? csr 0xa2 priority 3 vector tmintr 0x61
tape        mt0 at tm0 drive 0 flags 1
tape        mt1 at tm1 drive 0 flags 1
#
# Support for 2 Xylogics 472 tape controllers with 1 tape drive each.
#
controller  xtc0 at vme16d16 ? csr 0xee60 priority 3 vector xtintr 0x64
controller  xtc1 at vme16d16 ? csr 0xee68 priority 3 vector xtintr 0x65
tape        xt0 at xtc0 drive 0 flags 1
tape        xt1 at xtc1 drive 0 flags 1
#
# Support for the GP/GP+/GP2 graphics processors.
# Requires cgtwo as well.
#
```

```
device      gpone0 at vme24d16 ? csr 0x210000   # GP or GP+
device      gpone0 at vme24d32 ? csr 0x240000   # GP2
#
# Support for either the Sun-2 color board, Sun-3 color board,
# or GP2 frame buffer.
#
device      cgtwo0 at vme24d16 ? csr 0x400000 priority 4
            vector cgtwointr 0xa8
#
# Support for color memory frame buffers on various machine types
#
device      cgfour0 at obmem ? csr 0x50300000 priority 4
#
device          cgsix0 at obmem ? csr 0x50000000 priority 4
#
# Support for 24-bit color frame buffers on various machine types
#
device      cgeight0 at obmem ? csr 0x50300000 priority 4
#
# Support for monochrome frame buffers on various machines.
#
device      bwtwo0 at obmem ? csr 0x50300000 priority 4
#
# Support for the TAAC-1 Application Accelerator.
#
device      taac0 at vme32d32 ? csr 0x28000000
#
# Support for 2 Systech VPC-2200 line printer controllers.
#
device      vpc0 at vme16d16 ? csr 0x480 priority 2 vector vpcintr 0x80
device      vpc1 at vme16d16 ? csr 0x500 priority 2 vector vpcintr 0x81
#
# Support for the hardware Data Ciphering Processor (aka the DES chip).
# Suggested if you make heavy use of secure RPC or secure NFS.
#
device      des0 at obio ? csr 0x66002000
#
# Support for the Floating Point Accelerator.
#
device      fpa0 at virtual ? csr 0xe0000000
#
# Support Intel Floppy controller
#
controller      fdc0 at obio ? csr 0x6e000000 priority 6 vector fdintr 0x5c
device          fd0 at fdc0 drive 0 flags 0
#
# Support for the Parallel Printer Port.
#
device         pp0 at obio ? csr 0x6f000000 priority 1
```

The Sun-3/80 GENERIC Kernel

The following is the GENERIC kernel for a Sun-3/80. This kernel is a subset of the Sun-3x GENERIC kernel.

```
#
# @(#)SDST80 1.7 89/02/23 SMI
#
# This config file describes a generic Sun-3x kernel, including all
# possible standard devices and software options.
#
# The following lines include support for all Sun-3x cpu types.
# There is little to be gained by removing support for particular
# cpu's, so you may as well leave them all in.
#
machine     "sun3x"
cpu     "SUN3X_80"  # Sun-3x/80
#
# Name this kernel "GENERIC".
#
ident       GENERIC
#
# This kernel supports about eight users.  Count one
# user for each timesharing user, one for each window
# that you typically use, and one for each diskless
# client you serve.  This is only an approximation
# used to control the size of various kernel data
# structures, not a hard limit.
#
maxusers    8

#
# Include all possible software options.
#
# The INET option is not really optional, every kernel must include it.
#
options     INET        # basic networking support - mandatory
#
# The following options are all filesystem related.  You only need
# QUOTA if you have UFS.  You only need UFS if you have a disk.
# Diskless machines can remove QUOTA, UFS, and NFSSERVER.  LOFS is
# only needed if you're using the Sun Network Software Environment.
#
options     QUOTA       # disk quotas for local disks
options     UFS     # filesystem code for local disks
options     NFSCLIENT   # NFS client side code
options     NFSSERVER   # NFS server side code
options     LOFS        # loopback filesystem - needed by NSE
#
# The following options are for accounting and auditing.  SYSAUDIT
# should be removed unless you are using the C2 security features.
#
options     SYSACCT     # process accounting, see acct(2) & sa(8)
options     SYSAUDIT    # C2 auditing for security
```

```
#
# The following options are for various System V IPC facilities.
# No standard software needs them, although some third party
# software relies on at least IPCSHMEM.
#
options     IPCMESSAGE  # System V IPC message facility
options     IPCSEMAPHORE    # System V IPC semaphore facility
options     IPCSHMEM    # System V IPC shared-memory facility
#
# The following option includes the software DES support, needed if
# you're using secure NFS or secure RPC and you don't have a DES chip.
#
options     CRYPT       # software encryption (if no DES chip)

#
# Build one kernel based on this basic configuration.
# It will use the generic swap code so that you can have
# your root filesystem and swap space on any supported device.
# Put the kernel configured this way in a file named "vmunix".
#
config      vmunix      swap generic

#
# Include support for all possible pseudo-devices.
#
# The first few are mostly concerned with networking.
# You should probably always leave these in.
#
pseudo-device   pty     # pseudo-tty's, also needed for SunView
pseudo-device   ether       # basic Ethernet support
pseudo-device   loop        # loopback network - mandatory
#
# The next few are for SunWindows support, needed to run SunView 1.
#
pseudo-device   win128      # window devices, allow 128 windows
pseudo-device   dtop4       # desktops (screens), allow 4
pseudo-device   ms3     # mouse support, allow 3 mice
#
# The following is needed to support the Sun keyboard, with or
# without the window system.
#
pseudo-device   kb3     # keyboard support, allow 3 keyboards
#
# The following is for asynchronous tty support for the ALM-2 (aka MCP).
# If you have an ALM-2 (MCP) and it is being used to connect timesharing
# terminals, you will need this.
#
# pseudo-device mcpa64
#
# The following is for the streams pipe device.  Currently nothing
# depends on this device so it is entirely optional.
#
pseudo-device   sp
```

```
#
# The following are for streams NIT support.  NIT is used by
# etherfind, traffic, rarpd, and ndbootd.  As a rule of thumb,
# NIT is almost always needed on a server and almost never
# needed on a diskless client.
#
pseudo-device   snit        # streams NIT
pseudo-device   pf      # packet filter
pseudo-device   nbuf        # NIT buffering module
#
# The following is for the "clone" device, used with streams devices.
# This is required if you include streams NIT support.
#
pseudo-device   clone

#
# The following sections describe what kinds of busses each
# cpu type supports.  You should never need to change this.
# (The word "nexus" is historical...)
#

# connections for machine type 2 (SUN3X_80)
controller  virtual 2 at nexus ?
controller  obmem 2 at nexus ?
controller  obio 2 at nexus ?

#
# Support for the SCSI-ESP host adapter for 3/80
#
controller  sm0 at obio ? csr 0x66000000 priority 2
tape        st0 at sm0 drive 32 flags 1
tape        st1 at sm0 drive 40 flags 1
#disk       sf0 at sm0 drive 49 flags 2
disk        sd0 at sm0 drive 0 flags 0
disk        sd1 at sm0 drive 1 flags 0
disk        sd2 at sm0 drive 8 flags 0
disk        sd3 at sm0 drive 9 flags 0
disk            sd4 at sm0 drive 16 flags 0
disk            sd6 at sm0 drive 24 flags 0

#
# Support for the on-board LANCE Ethernet (AM7990).
#
device      le0 at obio ? csr 0x65002000 priority 3

device      zs0 at obio ? csr 0x62002000 flags 3 priority 3

#
# Support for the keyboard and mouse interface.  Needed when
# using a frame buffer as the console device or with SunView.
# You can remove this line if you don't use the standard Sun
# Workstation keyboard and mouse, but if you leave it in don't
# change it.
```

```
#
device      zs1 at obio ? csr 0x62000000 flags 0x103 priority 3

#
# Support for 3X/80 P4 frame buffers
#
# 3x/460 P4 color frame buffer
device      cgfour0 at obmem ? csr 0x50300000 priority 4
# 3x/460 P4 monochrome frame buffer
device      bwtwo0 at obmem ? csr 0x50300000 priority 4

#
# Support for the hardware Data Ciphering Processor (aka the DES chip).
# Suggested if you make heavy use of secure RPC or secure NFS.
#
# device        des0 at obio ? csr 0x66002000

#
# Support Intel Floppy controller
#
controller  fdc0 at obio ? csr 0x6e000000 priority 6 vector fdintr 0x5c
device      fd0 at fdc0 drive 0 flags 0

#
# Support for the Parallel Printer Port.
#
device         pp0 at obio ? csr 0x6f000000 priority 1
```

The Sun-4 GENERIC Kernel

The following is the GENERIC configuration file for a Sun-4.

NOTE If you wish to see the Sun-4c generic kernel, It is located in the same directory on your installation tape as the Sun-4. A copy is not included in this documentation.

```
#
# @(#)GENERIC 1.55 89/02/23 SMI
#
# This config file describes a generic Sun-4 kernel, including all
# possible standard devices and software options.
#
# The following lines include support for all Sun-4 cpu types.
# There is little to be gained by removing support for particular
# cpu's, so you may as well leave them all in.
#
machine     "sun4"
cpu     "SUN4_260"  # Sun-4/260, Sun-4/280
cpu     "SUN4_110"  # Sun-4/110
cpu     "SUN4_330"  # Sun-4/330
#
# Name this kernel "GENERIC".
#
ident       GENERIC
#
# This kernel supports about eight users.  Count one
# user for each timesharing serial port, one for each window
# that you typically use, and one for each diskless
# client you serve.  This is only an approximation
# used to control the size of various kernel data
# structures, not a hard limit.
#
maxusers    8

#
# Include all possible software options.
#
# The INET option is not really optional, every kernel must include it.
#
options     INET        # basic networking support - mandatory
#
# The following options are all filesystem related.  You only need
# QUOTA if you have UFS.  You only need UFS if you have a disk.
# Diskless machines can remove QUOTA, UFS, and NFSSERVER.  LOFS is
# only needed if you're using the Sun Network Software Environment.
#
options     QUOTA       # disk quotas for local disks
options     UFS     # filesystem code for local disks
options     NFSCLIENT   # NFS client side code
options     NFSSERVER   # NFS server side code
options     LOFS        # loopback filesystem - needed by NSE
#
```

```
# The following options are for accounting and auditing.  SYSAUDIT
# should be removed unless you are using the C2 security features.
#
options     SYSACCT     # process accounting, see acct(2) & sa(8)
options     SYSAUDIT    # C2 auditing for security
#
# The following options are for various System V IPC facilities.
# No standard software needs them, although some third party
# software relies on at least IPCSHMEM.
#
options     IPCMESSAGE  # System V IPC message facility
options     IPCSEMAPHORE    # System V IPC semaphore facility
options     IPCSHMEM    # System V IPC shared-memory facility
#
# The following option is only needed if you want to use the trpt
# command to debug TCP problems.
#
options     TCPDEBUG    # TCP debugging, see trpt(8)
#
# The following option includes the software DES support, needed if
# you're using secure NFS or secure RPC and you don't have a DES chip.
#
options     CRYPT       # software encryption (if no DES chip)

#
# Build one kernel based on this basic configuration.
# It will use the generic swap code so that you can have
# your root filesystem and swap space on any supported device.
# Put the kernel configured this way in a file named "vmunix".
#
config      vmunix      swap generic

#
# Include support for all possible pseudo-devices.
#
# The first few are mostly concerned with networking.
# You should probably always leave these in.
#
pseudo-device   pty     # pseudo-tty's, also needed for SunView
pseudo-device   ether       # basic Ethernet support
pseudo-device   loop        # loopback network - mandatory
#
# The next few are for SunWindows support, needed to run SunView 1.
#
pseudo-device   win128      # window devices, allow 128 windows
pseudo-device   dtop4       # desktops (screens), allow 4
pseudo-device   ms3     # mouse support, allow 3 mice
#
# The following is needed to support the Sun keyboard, with or
# without the window system.
#
pseudo-device   kb3     # keyboard support, allow 3 keyboards
#
```

```
# The following is needed to support the Sun dialbox.
#
pseudo-device   db              # dialbox support
#
# The following is for asynchronous tty support for the ALM-2 (aka MCP).
# If you have an ALM-2 (MCP) and it is being used to connect timesharing
# terminals, you will need this.  The number appended to mcpa should be
# the total number of serial lines provided by the ALM-2s in the system.
# For example, if you had eight ALM-2s this should read "mcpa128".
#
pseudo-device   mcpa64
#
# The following is for the streams pipe device.  Currently nothing
# depends on this device so it is entirely optional.
#
pseudo-device   sp
#
# The following are for streams NIT support.  NIT is used by
# etherfind, traffic, rarpd, and ndbootd.  As a rule of thumb,
# NIT is almost always needed on a server and almost never
# needed on a diskless client.
#
pseudo-device   snit        # streams NIT
pseudo-device   pf      # packet filter
pseudo-device   nbuf        # NIT buffering module
#
# The following is for the "clone" device, used with streams devices.
# This is required if you include streams NIT support.
#
pseudo-device   clone

#
# The following sections describe what kinds of busses each
# cpu type supports.  You should never need to change this.
# (The word "nexus" is historical...)
#

# connections for machine type 1 (SUN4_260)
controller  obmem 1 at nexus ?  # memory-like devices on the cpu board
controller  obio 1 at nexus ?   # I/O devices on the cpu board
controller  vme16d16 1 at nexus ?   # VME 16 bit address 16 bit data devices
controller  vme24d16 1 at nexus ?   # VME 24 bit address 16 bit data devices
controller  vme32d16 1 at nexus ?   # VME 32 bit address 16 bit data devices
controller  vme16d32 1 at nexus ?   # VME 16 bit address 32 bit data devices
controller  vme24d32 1 at nexus ?   # VME 24 bit address 32 bit data devices
controller  vme32d32 1 at nexus ?   # VME 32 bit address 32 bit data devices

# connections for machine type 2 (SUN4_110)
# NOTE: does not support bus-master devices.
controller  obmem 2 at nexus ?
controller  obio 2 at nexus ?
controller  vme16d16 2 at nexus ?
controller  vme24d16 2 at nexus ?
```

```
controller  vme32d16 2 at nexus ?
controller  vme16d32 2 at nexus ?
controller  vme24d32 2 at nexus ?
controller  vme32d32 2 at nexus ?

# connections for machine type 3 (SUN4_330)
controller  obmem 3 at nexus ?
controller  obio 3 at nexus ?
controller  vme16d16 3 at nexus ?
controller  vme24d16 3 at nexus ?
controller  vme32d16 3 at nexus ?
controller  vme16d32 3 at nexus ?
controller  vme24d32 3 at nexus ?
controller  vme32d32 3 at nexus ?

#
# The following (large) section describes which standard devices this
# kernel supports.
#

#
# Support for 4 Xylogics 7053 controllers with 4 drives each.
#
controller  xdc0 at vme16d32 ? csr 0xee80 priority 2 vector xdintr 0x44
controller  xdc1 at vme16d32 ? csr 0xee90 priority 2 vector xdintr 0x45
controller  xdc2 at vme16d32 ? csr 0xeea0 priority 2 vector xdintr 0x46
controller  xdc3 at vme16d32 ? csr 0xeeb0 priority 2 vector xdintr 0x47
disk        xd0 at xdc0 drive 0
disk        xd1 at xdc0 drive 1
disk        xd2 at xdc0 drive 2
disk        xd3 at xdc0 drive 3
disk        xd4 at xdc1 drive 0
disk        xd5 at xdc1 drive 1
disk        xd6 at xdc1 drive 2
disk        xd7 at xdc1 drive 3
disk        xd8 at xdc2 drive 0
disk        xd9 at xdc2 drive 1
disk        xd10 at xdc2 drive 2
disk        xd11 at xdc2 drive 3
disk        xd12 at xdc3 drive 0
disk        xd13 at xdc3 drive 1
disk        xd14 at xdc3 drive 2
disk        xd15 at xdc3 drive 3
#
# Support for 2 Xylogics 450/451 controllers with 2 drives each.
#
controller  xyc0 at vme16d16 ? csr 0xee40 priority 2 vector xyintr 0x48
controller  xyc1 at vme16d16 ? csr 0xee48 priority 2 vector xyintr 0x49
disk        xy0 at xyc0 drive 0
disk        xy1 at xyc0 drive 1
disk        xy2 at xyc1 drive 0
disk        xy3 at xyc1 drive 1
#
```

```
# Support for the SCSI-2 host adapter with 2 disks and 1 1/4" tape
# on the first SCSI controller, and 2 disks and 1 1/4" tape on the
# second SCSI controller.
#
controller  sc0 at vme24d16 ? csr 0x200000 priority 2 vector scintr 0x40
tape        st0 at sc0 drive 32 flags 1
tape        st1 at sc0 drive 40 flags 1
#disk       sf0 at sc0 drive 49 flags 2
disk        sd0 at sc0 drive 0 flags 0
disk        sd1 at sc0 drive 1 flags 0
disk        sd2 at sc0 drive 8 flags 0
disk        sd3 at sc0 drive 9 flags 0
disk        sd4 at sc0 drive 16 flags 0
disk        sd6 at sc0 drive 24 flags 0
#
# Support for the SCSI-3 host adapter.  Same device support as above.
#
controller  si0 at vme24d16 ? csr 0x200000 priority 2 vector siintr 0x40
controller  si1 at vme24d16 ? csr 0x204000 priority 2 vector siintr 0x41
tape        st0 at si0 drive 32 flags 1
tape        st1 at si0 drive 40 flags 1
tape        st2 at si1 drive 32 flags 1
tape        st3 at si1 drive 40 flags 1
#disk       sf0 at si0 drive 49 flags 2
disk        sd0 at si0 drive 0 flags 0
disk        sd1 at si0 drive 1 flags 0
disk        sd2 at si0 drive 8 flags 0
disk        sd3 at si0 drive 9 flags 0
disk        sd4 at si0 drive 16 flags 0
disk        sd6 at si0 drive 24 flags 0
#
# Support for the "SCSI weird" host adapter used with the Sun-4/110.
# Same device support as above.
#
controller  sw0 at obio 2 csr 0xa000000 priority 2
tape        st0 at sw0 drive 32 flags 1
tape        st1 at sw0 drive 40 flags 1
#disk       sf0 at sw0 drive 49 flags 2
disk        sd0 at sw0 drive 0 flags 0
disk        sd1 at sw0 drive 1 flags 0
disk        sd2 at sw0 drive 8 flags 0
disk        sd3 at sw0 drive 9 flags 0
disk        sd4 at sw0 drive 16 flags 0
disk        sd6 at sw0 drive 24 flags 0
#
# Support for the SCSI-ESP host adapter.
#
controller  sm0 at obio ? csr 0xfa000000 priority 2
tape        st0 at sm0 drive 32 flags 1
tape        st1 at sm0 drive 40 flags 1
#disk       sf0 at sm0 drive 49 flags 2
disk        sd0 at sm0 drive 0 flags 0
disk        sd1 at sm0 drive 1 flags 0
```

```
disk        sd2 at sm0 drive 8 flags 0
disk        sd3 at sm0 drive 9 flags 0
disk        sd4 at sm0 drive 16 flags 0
disk        sd6 at sm0 drive 24 flags 0
#
# Support for the 2 tty lines (ttya, ttyb) on the cpu board.
# Needed when using a terminal for the console device.
# Flags=3 says to supply carrier in software for both lines.
#
device      zs0 at obio ? csr 0xf1000000 flags 3 priority 3
#
# Support for the keyboard and mouse interface.  Needed when
# using a frame buffer as the console device or with SunView.
# You can remove this line if you don't use the standard Sun
# Workstation keyboard and mouse, but if you leave it in don't
# change it.
#
device      zs1 at obio ? csr 0xf0000000 flags 0x103 priority 3
#
# Support for 2 additional tty lines on board the Sun-4/330.
#
device      zs2 at obio 3 csr 0xe0000000 flags 0x3 priority 3
#
# Support for 4 ALM's (Systech MTI-800/1600).  Flags set for
# all lines to be local, i.e., carrier supplied by software
# rather than by the device.
#
device      mti0 at vme16d16 1 csr 0x620 flags 0xffff priority 4
    vector mtiintr 0x88
device      mti1 at vme16d16 1 csr 0x640 flags 0xffff priority 4
    vector mtiintr 0x89
device      mti2 at vme16d16 1 csr 0x660 flags 0xffff priority 4
    vector mtiintr 0x8a
device      mti3 at vme16d16 1 csr 0x680 flags 0xffff priority 4
    vector mtiintr 0x8b
#
# Support for 8 MCP boards.
# Note that the first four MCP's use the same vectors as the ALM's and thus
# ALM's cut into the total number of MCP's that can installed.
# Make sure the maxusers line above has at least one added to it for
# each serial port.
#
device      mcp0 at vme32d32 ? csr 0x01000000 flags 0x1ffff priority 4
    vector mcpintr 0x8b
device      mcp1 at vme32d32 ? csr 0x01010000 flags 0x1ffff priority 4
    vector mcpintr 0x8a
device      mcp2 at vme32d32 ? csr 0x01020000 flags 0x1ffff priority 4
    vector mcpintr 0x89
device      mcp3 at vme32d32 ? csr 0x01030000 flags 0x1ffff priority 4
    vector mcpintr 0x88
device          mcp4 at vme32d32 ? csr 0x01040000 flags 0x1ffff priority 4
        vector mcpintr 0xa0
device          mcp5 at vme32d32 ? csr 0x01050000 flags 0x1ffff priority 4
```

```
        vector mcpintr 0xa1
device          mcp6 at vme32d32 ? csr 0x01060000 flags 0x1ffff priority 4
        vector mcpintr 0xa2
device          mcp7 at vme32d32 ? csr 0x01070000 flags 0x1ffff priority 4
        vector mcpintr 0xa3
#
# Support for the on-board Intel 82586 Ethernet chip on many machines.
#
device      ie0 at obio ? csr 0xf6000000 priority 3
#
# Support for a second Intel Ethernet board, either a second
# Sun-3/E board or a Multibus Ethernet board used with a
# Multibus-to-VME adapter.  Used for Ethernet to Ethernet
# gateways.
#
device      ie1 at vme24d16 ? csr 0xe88000 priority 3 vector ieintr 0x75
#
# Support for the on-board LANCE Ethernet (AM7990) on the Sun-4/330.
#
device      le0 at obio ? csr 0xf9000000 priority 3
#
# Support for 2 Xylogics 472 tape controllers with 1 tape drive each.
#
controller  xtc0 at vme16d16 ? csr 0xee60 priority 3 vector xtintr 0x64
controller  xtc1 at vme16d16 ? csr 0xee68 priority 3 vector xtintr 0x65
tape        xt0 at xtc0 drive 0 flags 1
tape        xt1 at xtc1 drive 0 flags 1
#
# Support for the GP/GP+/GP2 graphics processors.
# Requires cgtwo as well.
#
device      gpone0 at vme24d16 ? csr 0x210000   # GP or GP+
device      gpone0 at vme24d32 ? csr 0x240000   # GP2
#
# Support for either the Sun-2 color board, Sun-3 color board,
# or GP2 frame buffer.
#
device      cgtwo0 at vme24d16 ? csr 0x400000 priority 4
            vector cgtwointr 0xa8
#
# Support for color memory frame buffers on various machine types.
#
device      cgfour0 at obio 2 csr 0xfb300000 priority 4
device      cgfour0 at obio 3 csr 0xfb300000 priority 4
#
# Support for monochrome memory frame buffers on various machines.
#
device      bwtwo0 at obio 1 csr 0xfd000000 priority 4
device      bwtwo0 at obio 2 csr 0xfb300000 priority 4
device      bwtwo0 at obio 3 csr 0xfb300000 priority 4
#
# Support for the TAAC-1 Application Accelerator.
#
```

```
device      taac0 at vme32d32 1 csr 0x28000000
device      taac0 at vme32d32 2 csr 0xF8000000
#
# Support for 24-bit, with overlay, P4 base framebuffer.
device      cgeight0 at obio 2 csr 0xfb300000 priority 4    # 4/110
device      cgeight0 at obio 3 csr 0xfb300000 priority 4    # 4/330
#
# Support for low end graphics option
device      cgsix0 at obio ? csr 0xfb000000 priority 4
#
# Support for 2 Systech VPC-2200 line printer controllers.
#
device      vpc0 at vme16d16 ? csr 0x480 priority 2 vector vpcintr 0x80
device      vpc1 at vme16d16 ? csr 0x500 priority 2 vector vpcintr 0x81
#
# Support for the hardware Data Ciphering Processor (aka the DES chip).
# Suggested if you make heavy use of secure RPC or secure NFS.
#
device      des0 at obio ? csr 0xfe000000
```

9.4. Procedures for Reconfiguring the Kernel

This section contains instructions for creating the new kernel after you modified the kernel configuration file. In order to perform these steps, first perform the following:

- Install Release 4.0.3 using `suninstall`.
- Log in as superuser.
- Create the file `/usr/share/sys/sun`*[2,3,4]*`conf/`*SYS_NAME* from the GENERIC or other template configuration file.
- Modify the `SYS_NAME` file according to the instructions in the previous section, and change the file's permissions.

Two sets of instructions follow: one for standalones and one for servers and their clients. Refer to the set that applies to your configuration.

Kernel Reconfiguration for Standalone Systems

For standalone machines, proceed as follows.

1. Go to the directory `/usr/share/sys/sun[2,3,3x,4]/conf` if you are not there already:

```
# cd /usr/share/sys/sun[2,3,3x,4]/conf
```

2. Run `/usr/etc/config`. Then change to the new configuration directory, and make the new system as shown below. (Remember to substitute your actual system image name for *SYS_NAME*):

```
# /usr/etc/config SYS_NAME
# cd ../SYS_NAME
# make
[ lots of output ]
```

3. Now save the old kernel and install the new one as follows:

```
# mv /vmunix /vmunix.old
# cp vmunix /vmunix
# /usr/etc/shutdown -h now

        The system goes through the halt sequence, then
        the monitor displays its prompt, at which point you
        can boot the system:

> b vmunix
```

4. If the system appears to work, this completes the upgrade procedure. If the new kernel doesn't seem function properly, boot `/vmunix.old`, copy it back to `/vmunix`, and fix your new kernel as follows:

```
# /usr/etc/shutdown -h now
> b vmunix.old -s
# mv /vmunix /vmunix.oops
# mv /vmunix.old /vmunix
# ^D      [ Brings the system up multi-user ]
```

It is advisable to keep a copy of the distributed GENERIC kernel available in case you have problems with any reconfigured kernel.

Kernel Reconfiguration for Servers and Their Clients

For server machines, proceed as follows.

1. Go to `/usr/share/sys/sun[2,3,3x,4]/conf` if you are not there already:

```
# cd /usr/share/sys/sun[2,3,3x,4]/conf
```

2. Run `/usr/etc/config`. Then change to the new configuration directory, and make the new system. (Remember to substitute your actual system image name for *SYS_NAME*):

```
# /usr/etc/config SYS_NAME
# cd ../SYS_NAME
# make
[ lots of output ]
```

3. Create kernel configuration files for your clients. (If you need help, refer to the previous section for the appropriate annotated configuration file for the client's architecture.) When editing the clients' configuration files (called *CLIENT_KERNEL_NAME* in the following steps), remember to include the entire set of devices used by all the machines:

```
# cd /export/exec/sun[2,3,3x,4]/sys/sun [2,3,3x,4]/conf
# cp TEMPLATE_NAME CLIENT_KERNEL_NAME
# chmod +w CLIENT_KERNEL_NAME
[ Edit CLIENT_KERNEL_NAME to reflect all clients' systems.
  Be especially careful with the device description lines. ]
# /usr/etc/config CLIENT_KERNEL_NAME
# cd ../CLIENT_KERNEL_NAME
# make
[ lots of output ]
```

4. Now you can position yourself in the directory that has the server's kernel in it, save your server's old kernel, install your new one, and try everything out:

```
# cd /usr/sys/sun[2,3,3x,4]SYS_NAME
# mv /vmunix /vmunix.old
# cp vmunix /vmunix
```

5. Next, install the appropriate client kernel under the client's root directory. Note that clients do not have to be halted at this time, but they must reboot in order to run the new kernel. To install the clients' kernel, save the original kernel (if there is one), install the new kernel image in `/export/root/client_name`, and then test it out by booting up one of the clients:

```
# cd /export/exec/sun[2,3,3x,4]/sys/sun[2,3,3x,4]CLIENT_KERNEL_NAME
# cp /export/root/client_name/vmunix /export/root/client_name/vmunix.old
[or wherever your client kernel is]
# mv vmunix /export/root/client_name/vmunix
[ On the client machine : ]
>b vmunix
```

6. Since at this point normal system performance is a highly, but not absolutely, certain indicator of a trouble-free kernel, if your system(s) appears to work you may proceed with some confidence. You have successfully completed installation.

   If, on the other hand, either of the new kernels does not seem to be functioning properly, halt all systems and boot from the original kernel. Then move the faulty kernel away and re-install the original in its place. Once you are booted up on the original, you can go about trying to fix the faulty kernel. For example, on the server type:

```
# /usr/etc/halt
> b vmunix.old -s
# cd /
# mv vmunix vmunix.bad
# mv vmunix.old vmunix
# ^D              [ Brings the system up multi-user ]
```

   For clients, halt all the clients on the server. You will have to correct the problem from the server.

On the server type:

```
# cd /export/root/client_name
# mv vmunix vmunix.bad
# mv vmunix.old vmunix
```

You may now boot up the clients and allow them to run while or until a new client kernel is made and ready to install; or if the clients can remain down, build

and install a new client kernel now.

9.5. Changing Swap Space

When you run `suninstall`, by default it sets up swap space for an NFS server or standalone system on Partition B. In addition, it sets up the swap space for clients in `/export/swap/`*client_name* to enable the client machine to swap over NFS. At this time, you can specify the size of the swap partition and client swap files, or have `suninstall` use the default.

After a system is in use, you may find the originally specified swap space for a machine is insufficient. Therefore, you will want to change swap size, either by increasing swap space on the first disk or repartitioning a second disk to include a swap partition. To do this on a server, you have to back up all the server's file systems, then rerun `suninstall`.

Procedures for Increasing Swap Space

To increase client swap space, you can use a program called `mkfile`. Its syntax is

```
mkfile [ -nv ] size[k|b|m] filename ...
```

The option `-n` tells `mkfile` to create an empty file; `-v` selects verbose mode, in which the system displays messages about what it is doing. The *size* argument specifies the size you want the file to be. The letters `k`, `b`, and `m` represent Kilobytes, Bytes, and Megabytes respectively. (Recommended swap size for a client machine is 10 Mbytes.) The *filename* argument, when configuring a client's swap space, should be the full pathname of the client machine's `swap` directory.

Here are steps you would take to change swap space for client raks.

1. Log in as superuser on your server machine.
2. Type the following:

```
# mkfile -n 10m /export/swap/raks
```

   This creates an empty file of 10 Mbytes in length in `/export/swap/raks`.

3. Reboot client raks.

Setting Up a Second Swap Partition

Another was you can increase swap size is to add or repartition a second disk with another swap partition. Use the `format` program to repartition the disk. Then add the following to the server's `/etc/fstab` file:

```
/dev/disk_abbrevb /dev/disk_abbrevb swap
```

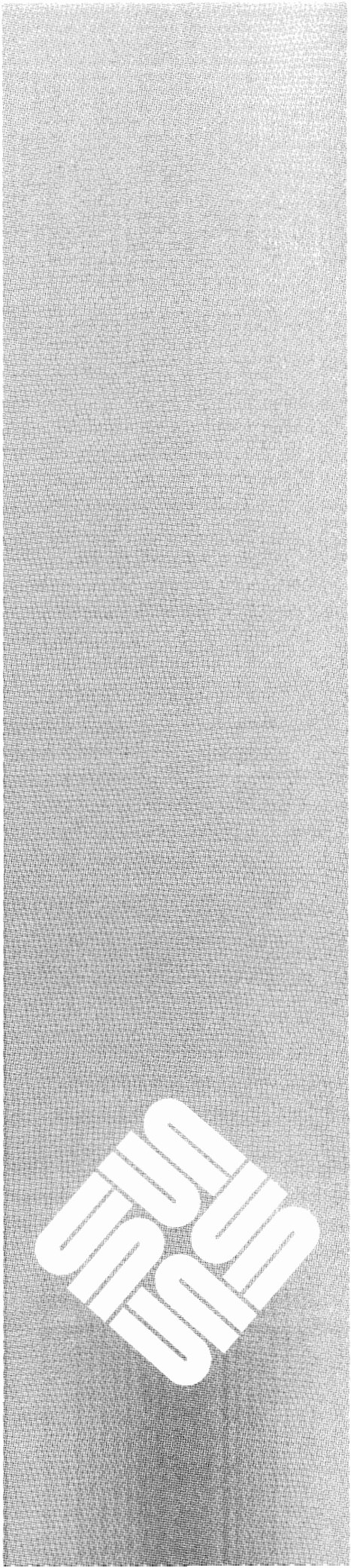
where *disk_abbrev* represents the disk type and number of the second disk.

Then check the server's `/etc/rc` file for the line

```
swapon -a
```

13

Addendum: Using the Automounter



13

Addendum: Using the Automounter

4.0.3 Inclusion Instructions

Add this first section to the end of Chapter 13, "The Sun Network File System."

You can mount file hierarchies shared through NFS a different method—automounting. The `automount` program enables users to mount and unmount remote directories on an as-needed basis. Whenever a user on a client machine running the automounter invokes a command that needs to access a remote file or directory, such as opening a file with an editor, the hierarchy to which that file or directory belongs is mounted and remains mounted for as long as it is needed. But whenever a certain amount of time has elapsed without the hierarchy being accessed, it is automatically unmounted. No mounting is done at boot- or run-level change-time, and the user no longer has to know the superuser password to mount a directory or even use the `mount` and `umount` commands. It is all done automatically and transparently.

Mounting some file hierarchies with `automount` does not exclude the possibility of mounting others with `mount`; A diskless machine *must* mount `/export/root`, `/export/swap` `/usr` and `/export/exec/kvm` through the `mount` program and `/etc/fstab` file. `mount`.

The next section explains how the automounter works. It also contains instructions for setting it up.

How the automounter works

Unlike `mount`, `automount` does not consult the file `/etc/fstab` for a list of hierarchies to mount. Rather, it consults a series of maps, which can be either direct or indirect. The names of the maps can be passed to `automount` from the command line, or from another (master) map.

Note The following explanation is written especially for advanced system administrators and programmers. If you do not have this type of experience, there is a summary at the end of this subsection to give you a bird's eye view of how the automounter works.

The automounter mounts everything under the directory `/tmp_mnt`, and provides a symbolic link from the requested mount point to the actual mount point under `/tmp_mnt`. For instance, if a user wants to mount a remote directory `src` under `/usr/src`, the actual mount point will be `/tmp_mnt/usr/src`, and `/usr/src` will be a symbolic link to that location. Note that, as with any other kind of mount, a mount affected through the automounter on a non-empty mount point will hide the original contents of the mount point for as long as the mount is in effect.

mount is in effect.

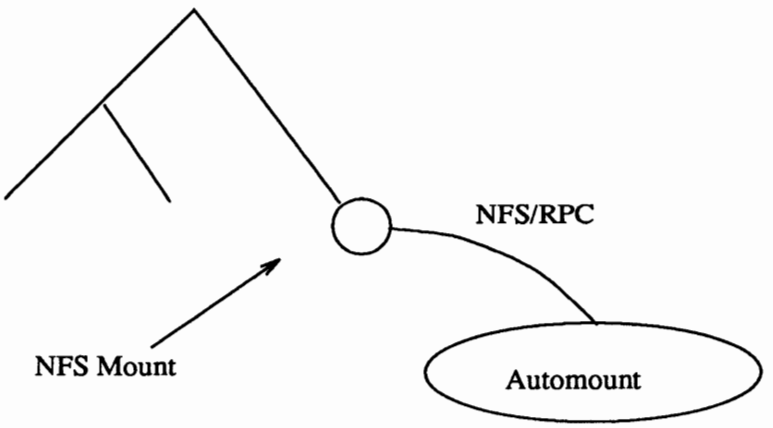
There are two distinct stages in the automounter's actions:

- The initial stage, boot time, when `rc.local` boots the automounter.
- The mounting stage, when a user tries to access a file or directory in a remote machine.

At the initial stage, when `rc.local` invokes `automount`, it opens a UDP socket and registers it with the `rpcbind` service as an NFS server port. It then forks off a server daemon that listens for NFS requests on the socket. The parent process proceeds to mount the daemon at its mount points within the file system (as specified by the maps). Through the `mount(2)` system call it passes the server daemon's port number and an NFS *file handle* that is unique to each mount point. The arguments to the `mount(2)` system call vary according to the kind of file system; for NFS file systems, the call is

```
mount ( "nfs", "/usr", &nfs_args );
```

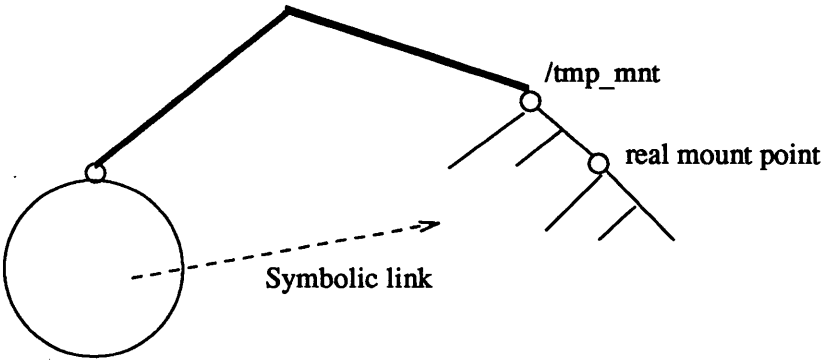
where `&nfs_args` contains the network address for the NFS server. By having the network address in `&nfs_args` refer to the local process (the automount daemon), `automount` in fact deceives the kernel into treating it as if it were an NFS server. Instead, once the parent process completes its calls to `mount(2)` it exits, leaving the daemon to serve its mount points:



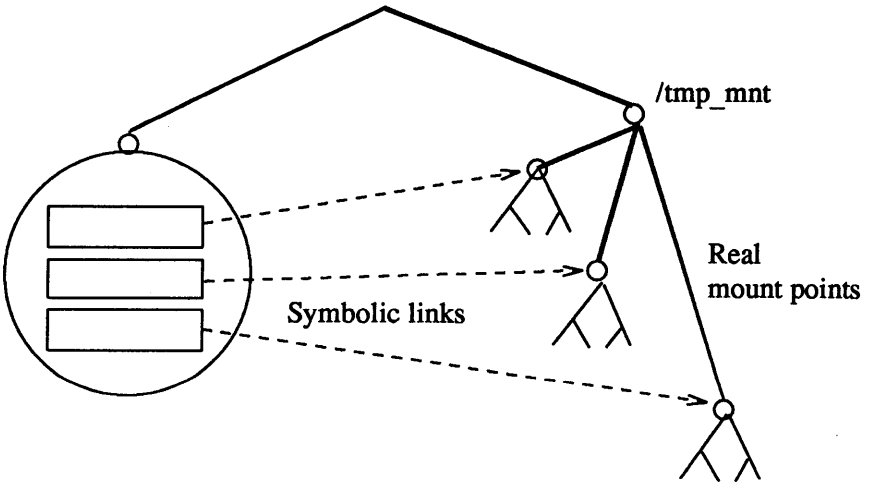
In the second stage, when the user actually requests access to a remote file hierarchy, the daemon intercepts the kernel NFS request and looks up the name in the map associated with the directory. Taking the location (*server:pathname*) of the remote file system from the map, the daemon then mounts the remote file system under the directory `/tmp_mnt`. It answers the kernel, telling it it is a symbolic link. The kernel sends an NFS READLINK request, and the automounter returns a symbolic link to the real mount point under `/tmp_mnt`.

The behavior of the automounter is affected by whether the name is found in a direct or an indirect map.

- If the name is found in a direct map, the automounter emulates a symbolic link, as stated above. It responds as if a symbolic link exists at its mount point. In response to a GETATTR, it describes itself as a symbolic link. When the kernel follows up with a READLINK it returns a path to the *real* mount point for the remote hierarchy in `/tmp_mnt`:



- If, on the other hand, the name is found in an indirect map, the automounter emulates a directory of symbolic links. It describes itself as a directory. In response to a READLINK, it returns a path to the mount point in `/tmp_mnt`, and a readdir(3) of the automounter's mount point returns a list of the entries that are *currently* mounted:



Whatever the case, that is, whether the map is direct or indirect, if the file hierarchy is already mounted and the symbolic link has been read recently, the cached symbolic link is returned immediately. Since the automounter is on the same host, the response is much faster than a READLINK to a remote NFS server. On the other hand, if the file hierarchy is not mounted, a small delay will occur while the mounting takes place.

Summary:

When `automount` is called from the command line or `rc.local`, `automount` forks a daemon to serve each mount point in the maps and makes the kernel believe that the mount has taken place. The daemon sleeps until a request is

made to access the corresponding file hierarchy. At that time the daemon does the following:

1. Intercepts the request
2. Mounts the remote file hierarchy
3. Creates a symbolic link between the requested mount point and the actual mount point under `/tmp_mnt`.
4. Passes the symbolic link to the kernel, and steps aside.
5. Unmounts the file hierarchy when a predetermined amount of time has passed with the link not being touched (generally five minutes), and resumes its previous position.

Preparing the Maps

A server never knows, nor cares, whether the files it shares are accessed through `mount` or `automount`. Therefore, you need not do anything different on the server for `automount` than for `mount`.

A client, however, needs special files for the automounter. As mentioned previously, `automount` does not consult `/etc/fstab`; rather, it consults the map file(s) specified at the command line (see below, "Invoking `automount`"). All automounter maps are located in the directory `/etc`. Their names all begin with the prefix `auto.`

There are three kinds of automount maps:

- master
- indirect
- direct

They are described below.

The Master Map

Each line in a master map, by convention called `/etc/auto.master`, has the syntax:

```
Mount-point     Map     [ Mount-options ]
```

where:

- *mount-point* is the full pathname of a directory.
- *map* is the name of the map the automounter should use to find the mount points and locations.
- *mount-options* is an optional, comma separated, list of options that regulate the mounting of the entries mentioned in the *map*, unless the *map* entries list other options.

A line whose first character is # is treated as a comment, and everything that follows until the end of line is ignored. A backslash (\) at the end of line permits splitting long lines into shorter ones.

Direct and Indirect Maps

Lines in direct and indirect maps have the syntax:

```
key     [ mount-options ]     location
```

where

- *key* is the pathname of the mount point.
- The *mount-options* are the options you want to apply to this particular mount.
- *location* is the location of the resource, specified as *server:pathname.*

As in the master map, a line whose first character is # is treated as a comment and everything that follows until the end of line is ignored. A backslash at the end of line permits splitting long lines into shorter ones.

The only formal difference between a direct and an indirect map is that the key in a direct map is a full pathname, whereas in an indirect pathname it is a simple name (no slashes). For instance, the following would be an entry in a direct map:

```
/usr/man    -ro,intr    goofy:/usr/man
```

and the following would be an entry in an indirect map:

```
parsley     -ro,intr    veggies:/usr/greens
```

As you can see, the *key* in the indirect map is begging for more information: where is the mount point *parsley* really located? That is why you must either provide that information at the command line or through another map. For instance, if the above line is part of a map called `/etc/auto.veggies`, you would have to call it up either as:

```
automount /veggies /etc/auto.veggies
```

or specify, in the master map:

```
/veggies    /etc/auto.veggies    -ro,soft,nosuid
```

In either case, you are associating a mount directory (`veggies`) with the entries (`parsley` in this case) mentioned in the indirect map `/etc/auto.veggies`. The end result is that the hierarchy `/usr/greens` from the machine `veggies` will be mounted on `/veggies/parsley` when needed.

Writing a Master Map

As stated above, the syntax for each line in the master map is

```
Mount-point     Map     [ Mount-options ]
```

A typical `auto.master` file would contain

```
#Mount-point   Map                Mount-options
/net           -hosts
/home          /etc/auto.home     -rw,intr,secure
/-             /etc/auto.direct   -ro,intr
```

The automounter recognizes some special mount points and maps, which are explained below.

Mount point /–

In the example above, the mount point `/-` is a filler that the automounter recognizes as a directive not to associate the entries in `/etc/auto.direct` with any directory. Rather, the mount points are to be the ones mentioned in the map. (Remember, in a direct map the key is a full pathname.)

Mount point `/home`

The mount point `/home` is to be the directory under which the entries listed in `/etc/auto.home` (an indirect map) are to be mounted. That is, they will be mounted under `/tmp_mnt/home`, and a symbolic link will be provided between `/home/`*directory* and `/tmp_mnt/home/`*directory*.

Mount point `/net`

Finally, the automounter will mount under the directory `/net` all the entries under the special map `-hosts`. This is a built-in map that does not use any external files except the hosts database `/etc/hosts`(or `hosts.byname` YP map.) Notice that since the automounter does not mount the entries until needed, the specific order is not important. Once the automount daemon is in place, a user entering the command

```
example $ cd /net/gumbo
```

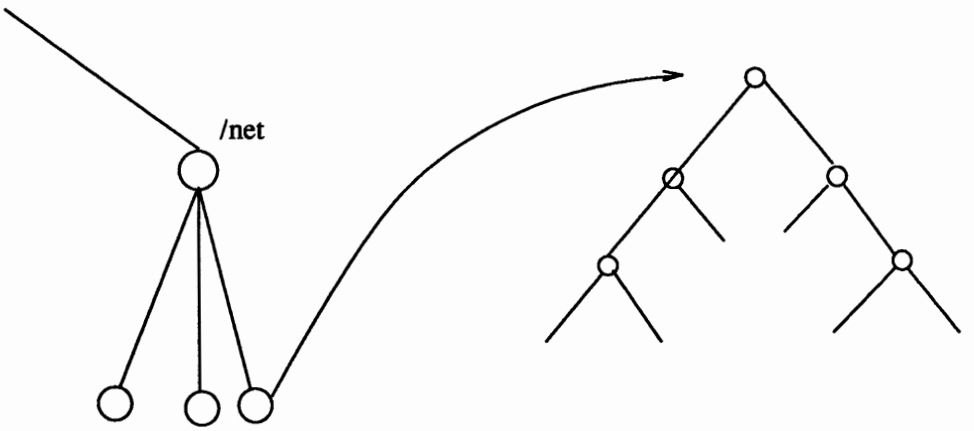
will change directory to the top of the hierarchy of files (i.e., the root file system) of the machine `gumbo` as long as the machine is in the hosts database. However, the user may not see under `/net/gumbo` all the files and directories. This is because the automounter can mount only the *shared* file systems of host `gumbo`, in accordance with the restrictions placed on the sharing.

The actions of the automounter when the command in the example above is issued are as follows:

1. `ping` the null procedure of the server's mount service to see if it's alive.
2. Request the list of shared hierarchies from the server.
3. Sort the shared list according to length of pathname (to ensure proper mounting order):

```
/usr/src
/export/home
/usr/src/sccs
/export/root/blah
```

4. Proceed down the list, mounting all the file systems at mount points in `/tmp_mnt` (creating the mount points as needed).
5. Return a symbolic link that points to the top of the recently mounted hierarchy:



Note that, unfortunately, the automounter has to mount all the file systems that the server in question advertises for sharing. Even if the request is as follows:

```
example $ ls /net/gumbo/usr/include
```

the automounter mounts all of `gumbo`'s shared systems, not just `/usr`.

In addition, unmounting that occurs after a certain amount of time has passed is from the bottom up.i This means if one of the directories at the top is busy, the automounter has to remount the hierarchy and try again later.

Nevertheless, the `-hosts` special map provides a very convenient way for users to access directories in many different hosts without having to use `rlogin` or `rsh`. (These remote commands have to establish communication through the network every time they are invoked.) Furthermore, they no longer have to ask you to modify their `/etc/fstab` files or mount the directories by hand as superuser.

Notice that both `/net` and `/home` are arbitrary names dictated by convention. The automounter will create them if they do not exist already.

Writing an indirect map

The syntax for an indirect map is:

```
key     [ mount-options ]     location
```

where *key* is the name (not the full pathname) of the directory that will be used as mount point. Once the key is obtained by the automounter, it is suffixed to the mount point associated with it either by the command line or by the master map that invokes the indirect map in question.

For instance, one of the entries in the master map presented above as an example, reads:

```
/home       /etc/auto.home  -rw,intr,secure
```

Here `/etc/auto.home` is the name of the indirect map that will contain the entries to be mounted under `/home`.

A typical `auto.home` map might contain:

```
#key      mount-options    location
willow                     willow:/home/willow
cypress                    cypress:/home/cypress
poplar                     poplar:/home/poplar
pine                       pine:/export/pine
apple                      apple:/export/home
ivy                        ivy:/home/ivy
peach     -rw,nosuid       peach:/export/home
```

As an example, assume that the map above is on host *oak*. If user *laura* has an entry in the password database specifying her home directory as `/home/willow/laura`, then whenever she logs into machine oak, the automounter will mount (as `/tmp_mnt/home/willow`) the directory `/home/willow` residing in machine willow. If one of the directories is indeed `laura`, she will be in her home directory, which is mounted read/write, interruptable and secure.

Suppose, however, that `laura`'s home directory is specified as `/home/peach/laura`. Whenever she logs into oak, the automounter mounts the directory `/export/home` from *peach* under `/tmp_mnt/home/peach`. Her home directory will be mounted read/write, nosuid. *Any option in the file entry overrides all options in the master map or the command line.*

Now, assume the following conditions occur:

- User laura's home directory is listed in the password database as `/home/willow/laura`.
- Machine `willow` shares its `home` hierarchy with the machines mentioned in `auto.home`.

- All those machines have a copy of the same `auto.home` and the same password database.

Under these conditions, user laura can run `login` or `rlogin` on any of these machines and have her home directory mounted in place for her.

Furthermore, now laura can also enter the command

```
% cd ~brent
```

and the automounter will mount brent's home directory for her (if all permissions apply).

On a network without YP, you have to change all the relevant databases (such as `/etc/passwd`) on all systems on the network in order to accomplish this. On a network running YP, propagate all the relevant databases throughout the network to ensure this.

Writing a Direct Map

The syntax for a direct map (like that for an indirect map) is:

```
key     [ mount-options ]       location
```

where:

- *key* is the *full* pathname of the mount point. (Remember that in an indirect map this is not a full pathname.)
- *mount-options* are optional but, if present, override — for the entry in question — the options of the calling line, if any, or the defaults. (See below, "Invoking `automount`").
- *location* is the location of the resource, specified as *server:pathname.*

Of all the maps, the entries in a direct map most closely resemble, in their simplest form, what their corresponding entries in `/etc/fstab` might look like. An entry that appears in `/etc/fstab` as:

```
dancer:/usr/local - /usr/local/tmp nfs - YES ro
```

appears in a direct map as:

```
/usr/local/tmp -ro dancer:/usr/local
```

The following is a typical `/etc/auto.direct` map:

```
/usr/local \
                    /bin      -ro,soft    ivy:/export/local/sun3 \
                    /share    -ro,soft    ivy:/export/local/share \
                    /src      -ro,soft    ivy:/export/local/src
/usr/man                      -ro,soft    oak:/usr/man \
                                          rose:/usr/man \
                                          willow:/usr/man
/usr/games                    -ro,soft    peach:/usr/games
/usr/spool/news               -ro,soft    pine:/usr/spool/news
/usr/frame                    -ro,soft    redwood:/usr/frame1.3 \
                                          balsa:/export/frame
```

Notice a couple of unusual features in this map. These are the subject of the next two subsections.

Multiple Mounts

A map entry can describe a multiplicity of mounts, where the mounts can be from different locations and with different mount options. Consider the first entry in the previous example. It is, in fact, one long entry whose readability has been improved by splitting it into three lines by using the backslash and indenting the continuation lines. This entry mounts `/usr/local/bin`, `/usr/local/share` and `/usr/local/src` from the server `ivy`, with the options read-only and soft. The entry could also read:

```
/usr/local \
                /bin      -ro,soft       ivy:/export/local/sun3 \
                /share    -rw,secure     willow:/usr/local/share \
                /src      -ro,intr       oak:/home/jones/src
```

where the options are different and more than one server is used.

Multiple mounts can be hierarchical. When file systems are mounted hierarchically, each file system is mounted on a subdirectory within another file system. When the root of the hierarchy is referenced, the automounter mounts the whole hierarchy. The concept of *root* here is very important. The symbolic link returned by the automounter to the kernel request is a path to the mount root. This is the root of the hierarchy that is mounted under `/tmp_mnt`. This mount point should theoretically be specified:

```
parsley     /    -ro,intr     veg:/usr/greens
```

In practice, it is not specified because in a trivial case of a single mount as above, it is assumed that the location of the mount point is *at* the mount root or "/." So instead of the above it is perfectly acceptable, indeed preferable, to enter

```
parsley     -ro,intr     veg:/usr/greens
```

The mount point specification, however, becomes important when mounting a hierarchy: here the automounter must have a mount point for each mount within the hierarchy. The example above is a good illustration of multiple, non-hierarchical mounts under `/usr/local` when the latter is already mounted.

The following illustration shows a true hierarchical mounting:

```
/usr/local \
                /          -rw,intr      peach:/export/local \
                /bin       -ro,soft      ivy:/export/local/sun3 \
                /share     -rw,secure    willow:/usr/local/share \
                /src       -ro,intr      oak:/home/jones/src
```

Note A true hierarchical mount can be problematic if the server for the root of the hierarchy goes down. Any attempt to unmount the lower branches will fail, since the unmounting has to proceed through the mount root, which also cannot be unmounted while its server is down.

The mount points used here for the hierarchy are `/`, `/bin`, `/share`, and `src`. Note that these mount point paths are relative to the *mount* root, not the host's *file system* root. The first entry in the example above has / as its mount point. It is mounted *at* the mount root. There is no requirement that the first mount of a hierarchy be at the mount root. The automounter will happily issue `mkdir`'s to build a path to the first mount point if it is not at the mount root.

Multiple Locations

In the example for a direct map, which was:

```
/usr/local \
                /bin       -ro,soft      ivy:/export/local/sun3 \
                /share     -ro,soft      ivy:/export/local/share \
                /src       -ro,soft      ivy:/export/local/src
/usr/man                   -ro,soft      oak:/usr/man \
                                         rose:/usr/man \
                                         willow:/usr/man
/usr/games                 -ro,soft      peach:/usr/games
/usr/spool/news            -ro,soft      pine:/usr/spool/news
/usr/frame                 -ro,soft      redwood:/usr/frame1.3 \
                                         balsa:/export/frame
```

the mount points `/usr/man` and `/usr/frame` list more than one location (three for the first, two for the second). This means that the mounting can be done from any of the replicated locations. This procedure makes sense only when you are mounting a hierarchy read-only, since theoretically you would like to have some control over the locations of files you write or modify. A good example is the man pages. In a large network, more than one server may export the current set of manual pages. It does not matter which server you mount them from, just so long as the server is up and running and sharing its file systems. In the example above, multiple mount locations are expressed as a list of mount locations in the map entry:

```
/usr/man -ro,soft oak:/usr/man rose:/usr/man willow:/usr/man
```

This could also be expressed as a comma separated list of servers, followed by a colon and the pathname (as long as the pathname is the same for all the replicated servers):

```
/usr/man -ro,soft oak,rose,willow:/usr/man
```

Here you can mount the man pages from the servers *oak*, *rose* or *willow*. From this list of servers the automounter first selects those that are on the local network and `ping`s these servers. This launches a series of RPC requests to the null procedure of the mount service in each server. (Note that the list does not imply any ordering.) The first server to respond is selected, and an attempt is made to mount from it.

This redundancy, very useful in an environment where individual servers may or may not be sharing their file systems, is enjoyed only at mount time. There is no status checking of the mounted-from server by the automounter once the mount occurs.. If the server goes down while the mount is in effect, the file system becomes unavailable. An option here is to wait five minutes until the auto-unmount takes place and try again. Next time around the automounter will choose one of the available servers. Another option is to use the `umount` command, inform the automounter of the change in the mount table (as specified in the later section, "The Mount Table"), and retry the mount.

Specifying Subdirectories

The earlier subsection, "Writing an Indirect Map", showed the following typical `auto.home` file:

```
#key      mount-options   location
willow                    willow:/home/willow
cypress                   cypress:/home/cypress
poplar                    poplar:/home/poplar
pine                      pine:/export/pine
apple                     apple:/export/home
ivy                       ivy:/home/ivy
peach     -rw,nosuid      peach:/export/home
```

Given this `auto.home` indirect file, every time a user wants to access a home directory in, say, `/home/willow`, all the directories under it will be mounted. Another way to organize an `auto.home` file is by user name, as in:

```
#key   mount-options   location
john                   willow:/home/willow/john
mary                   willow:/home/willow/mary
joe                    willow:/home/willow/joe
```

The above example assumes that home directories are of the form */home/user* rather than */home/server/user*. If a user now enters the following command:

```
% ls ~john ~mary
```

the automounter has to perform the *equivalent* of the following actions:

```
mkdir /tmp_mnt/home/john
mount willow:/home/willow/john /tmp_mnt/home/john
ln -s /tmp_mnt/home/john /home/john

mkdir /tmp_mnt/home/mary
mount willow:/home/willow/mary /tmp_mnt/home/mary
ln -s /tmp_mnt/home/mary /home/mary
```

However, the complete syntax of a line in a direct or indirect map is actually:

```
key   [ mount-option ]   server:pathname[:subdirectory]
```

Until now you used the form *server:pathname* to indicate the location. This is an ideal place for you to also indicate the subdirectory, like this:

```
#key   mount-options   location
john                   willow:/home/willow:john
mary                   willow:/home/willow:mary
joe                    willow:/home/willow:joe
```

Here `john`, `mary` and `joe` are entries in the *subdirectory* field. Now when a user refers to *john*'s home directory, the automounter mounts `willow:/home/willow`. It then places a symbolic link between `/tmp_mnt/home/willow/john` and `/home/john`.

If the user then requests access to *mary*'s home directory, the automounter sees that `willow:/home/willow` is already mounted, so all it has to do is return the link between `/tmp_mnt/home/willow/mary` and `/home/mary`. In other words, the automounter now only does:

```
mkdir /tmp_mnt/home/john
mount willow:/home/willow /tmp_mnt/home
ln -s /tmp_mnt/home/john /home/john

ln -s /tmp_mnt/home/mary /home/mary
```

In general, it is a good idea to provide a *subdirectory* entry in the *location* when different map entries refer to the same mounted file system from the same server.

Substitutions

If you have a map with a lot of subdirectories specified, like:

```
#key    mount-options location
john                  willow:/home/willow:john
mary                  willow:/home/willow:mary
joe                   willow:/home/willow:joe
able                  pine:/export/home:able
baker                 peach:/export/home:baker
        [.   .   .]
```

consider using string substitutions. You can use the ampersand character (&) to substitute the key wherever it appears. Using the ampersand, the above map now looks as follows:

```
#key    mount-options   location
john                    willow:/home/willow:&
mary                    willow:/home/willow:&
joe                     willow:/home/willow:&
able                    pine:/export/home:&
baker                   peach:/export/home:&
        [.   .   .]
```

If the name of the server is the same as the key itself, for instance:

```
#key     mount-options   location
willow                   willow:/home/willow
peach                    peach:/home/peach
pine                     pine:/home/pine
oak                      oak:/home/oak
poplar                   poplar:/home/poplar
        [.   .   .]
```

the use of the ampersand results in:

```
#key     mount-options   location
willow                   &:/home/&
peach                    &:/home/&
pine                     &:/home/&
oak                      &:/home/&
poplar                   &:/home/&
         [.   .   .]
```

Finally, notice that all the above entries have the same format. This permits you to use the catch-all substitute character, the asterisk (*). The asterisk reduces the whole thing to:

```
*    &:/home/&
```

where each ampersand is substituted by the value of any given key. Notice that once the automounter reads the catch-all key it does not continue reading the map, so that the following map would be viable:

```
#key     mount-options   location
oak                      &:/export/&
poplar                   &:/export/&
*                        &:/home/&
```

whereas in the next map the last two entries would always be ignored:

```
#key     mount-options   location
*                        &:/home/&
oak                      &:/export/&
poplar                   &:/export/&
```

You could also use key substitutions in a direct map, in situations like the following:

```
/usr/man          willow,cedar,poplar:/usr/man
```

which is a good candidate to be written as:

```
/usr/man          willow,cedar,poplar:&
```

Notice that the ampersand substitution uses the whole key string, so if the key in a direct map starts with a / (as it should), that slash is carried over, and you could not do something like

```
/progs  &1,&2,&3:/export/src/progs
```

because the automounter would interpret it as:

```
/progs  /progs1,/progs2,/progs3:/export/src/progs
```

Special Characters

Under certain circumstances you may have to mount directories whose names may confuse the automounter's map parser. An example might be a directory called `rc0:dk1`; this could result in an entry like:

```
/junk       -ro     vmsserver:rc0:dk1
```

The presence of the two colons in the location field will confuse the automounter's parser. To avoid this confusion, use a backslash to escape the second colon and remove its special meaning of separator:

```
/junk       -ro     vmsserver:rc0\:dk1
```

You can also use double quotes, as in the following example, where they are used to hide the blank space in the name:

```
/smile  dentist:"/front teeth"/smile
```

Environment Variables

You can use the value of an environmental variable by prefixing a dollar sign ($) to its name. You can also use braces to delimit the name of the variable from appended letters or digits.

The environmental variables can be inherited from the environment or can be defined explicitly with the `-D` command line option. For instance, if you want each client to mount client-specific files in the network in a replicated format, you could create a specific map for each client according to its name, so that the relevant line for host oak would be:

```
/mystuff    cypress,ivy,balsa:/export/hostfiles/oak
```

and in willow it would be:

```
/mystuff    cypress,ivy,balsa:/export/hostfiles/willow
```

This scheme is viable within a small network, but maintaining this kind of host specific maps across a large network would soon become unfeasible. The solution in this case would be to invoke the automounter with a command line similar to the following:

```
automount -D HOST=`hostname` ......
```

and have the entry in the direct map read:

```
/mystuff    cypress,ivy,balsa:/export/hostfiles/$HOST
```

Now each host would find its own files in the `mystuff` directory, and the task of centrally administering and distributing the maps becomes easier.

Invoking automount

Once the maps are written, you should make sure that there are no equivalent entries in `/etc/fstab`, and that all the entries in the maps refer to NFS shared files.

The syntax to invoke the automounter is:

automount [**–mnTv**] [**–D** *name*=*value*] [**–f** *master-file*] [**–M** *mount-directory*] [**–t** *sub-options*] \
[*directory map* [*–mount-options*]] . . .

The `automount`(8) man page contains a complete description of all options. The sub-options are the same as those for a standard NFS `mount`, excepting `bg` (background) and `fg` (foreground), which do not apply.

Given the following set of three maps:

- auto.master

```
#Mount-point   Map                Mount-options
/net           -hosts
/home          /etc/auto.home     -rw,intr,secure
/-             /etc/auto.direct   -ro,intr
```

- auto.home

```
#key      mount-options   location
willow                    willow:/home/willow
cypress                   cypress:/home/cypress
poplar                    poplar:/home/poplar
pine                      pine:/export/pine
apple                     apple:/export/home
ivy                       ivy:/home/ivy
peach     -rw,nosuid      peach:/export/home
```

- auto.direct

```
/usr/local \
                    /bin      -ro,soft    ivy:/export/local/sun3 \
                    /share    -ro,soft    ivy:/export/local/share \
                    /src      -ro,soft    ivy:/export/local/src
/usr/man                      -ro,soft    oak:/usr/man \
                                          rose:/usr/man \
                                          willow:/usr/man
/usr/games                    -ro,soft    peach:/usr/games
/usr/spool/news               -ro,soft    pine:/usr/spool/news
/usr/frame                    -ro,soft    redwood:/usr/frame1.3 \
                                          balsa:/export/frame
```

you can invoke the automounter (either from the command line or, preferably, from `rc.local`) in one of the following ways:

1. You can specify all arguments to the automounter without reference to the master map, as in:

```
automount /net -hosts /home /etc/auto.home -rw,intr,secure /- /etc/auto.direct -ro,intr
```

2. You can specify the same in the `auto.master` file, and instruct the automounter to look in it for instructions:

```
automount -f /etc/auto.master
```

3. You can specify more mount points and maps in addition to those mentioned in the master map, as follows:

```
automount -f /etc/auto.master /src /etc/auto.src -ro,soft
```

4. You can nullify one of the entries in the master map. (This is particularly useful if you use a map that you cannot modify and does not meet the needs of your machine):

```
automount -f /usr/lib/auto.master /home -null
```

5. You can replace one of the entries with your own:

```
automount -f /usr/lib/auto.master /home /myown/auto.home -rw,intr
```

In the example above, the automounter first mounts all items in the map `/myown/auto.home` under the directory `/home`. Then, when it consults the master file `/usr/lib/auto.master` and reaches the line corresponding to `/home` it simply ignores it, since it has already mounted on it.

Given the `auto.master` file of the previous example, commands (1) and (2) are equivalent *as long as your network does not have a distributed* `auto.master` *file*. This file is only available on networks running YP, and is fully described in a later section. If you network includes a distributed `auto.master` file, the second example would have to be modified in the following way to be equivalent to example 1:

```
automount -m -f /etc/auto.master
```

The `-m` option instructs the automounter not to consult the master file distributed by the YP. However, if you do not run YP, you do not have to specify the `-m` option. The automounter is completely silent when it does not find a distributed master file.

You can log in as superuser and type any of the above commands at shell level to start the automounter. Ideally, you should edit `rc.local` and include your preferred command there.

The Temporary Mount Point

The default name for all mounts is `/tmp_mnt`. Like the other names, this is arbitrary. It can be changed at invocation time by use of the `-M` option. For instance:

```
automount -M /auto ...
```

causes all mounts to happen under the directory `/auto`, which the automounter will create if it does not exist. It goes without saying that you should not designate a directory in a read only file system, as the automounter would not be able to modify anything then.

The Mount Table

Every time the automounter mounts or unmounts a file hierarchy, it modifies `/etc/mtab` to reflect the current situation. The automounter keeps an image in memory of `/etc/mtab`, and refreshes this image every time it performs a mounting or an automatic unmounting. If you use the `umount` command to unmount one of the automounted hierarchies (a directory under `/tmp_mnt`), the automounter should be forced to re-read the `/etc/mtab` file. To do that, enter the command:

```
example $ ps -ef | grep automount | egrep -v grep
```

This gives you the process ID of the automounter. The automounter is designed so that on receiving a SIGHUP signal it re-reads `/etc/mtab`. So, to send it that signal, enter:

```
% kill -1 PID
```

where *PID* stands for the process ID you obtained from the previous **ps** command.

Modifying the Maps

You can modify the automounter maps at any time, but remember that the automounter looks at the master and indirect maps only when it is invoked. If you want a modification to a maps to take effect immediately, you have to reboot the machine.

On the other hand, changes to a direct map should take effect the next time the automounter has to mount the modified entry. For instance, suppose you modify the file `/etc/auto.direct` so that the directory `/usr/src` is now mounted from a different server. The new entry takes effect immediately (if `/usr/src` is not mounted at this time) when you try to access it. If it is mounted now, you can wait until the auto unmounting takes place, and then access it. If this is not satisfactory, you can unmount with the `umount` command, notify `automount` that the mount table has changed (see above, "The Mount Table"), and then access it. The mounting should now be done from the new server.

Error Messages Related to `automount`

The following paragraphs are labeled with the error message you are likely to see if the automounter fails, and an indication of what the problem may be.

- `no mount maps specified`

  The automounter was invoked with no maps to serve, and it cannot find the YP `auto.master` map. This message is produced only when the `-v` option is given. Recheck the command, or restart YP if that was the intention.

- *mapname*`: Not found`

  The required map cannot be located. This message is produced only when the `-v` option is given. Check the spelling and pathname of the map name.

- `dir` *mountpoint* `must start with '/'`

  Automounter mountpoint must be given as full pathname. Check the spelling and pathname of the mount point.

- *mountpoint*`: Not a directory`

  The *mountpoint* exists but it is not a directory. Check the spelling and pathname of the mount point.

- `hierarchical mountpoint:` *mountpoint*

  Automounter will not allow itself to be mounted within an automounted directory. You will have to think of another strategy.

- `WARNING:` *mountpoint* `not empty!`

  The mountpoint is not an empty directory. This message is produced only when the `-v` option is given, and it is only a warning. It means that the previous contents of *mountpoint* will not be accessible.

- `Can't mount` *mountpoint*`:` *reason*

  Automounter cannot mount itself at *mountpoint*. The *reason* should be self-explanatory.

- *hostname*:*file system* already mounted on *mountpoint*

  Automounter has been mounted on an already mounted-on mountpoint and is attempting to mount the same file system there. This will happen if an entry in /etc/fstab also appears in an automounter map (either by accident or because the output of mount -p was redirected to fstab). Delete one of the redundant entries.

- WARNING: *hostname*:*file system* already mounted on *mountpoint*

  The automounter is mounting itself on top of an existing mount point (warning only).

- couldn't create *directory*: *reason*

  Couldn't create a directory. The *reason* should be self-explanatory.

- bad entry in map *mapname* "*map entry*"

- map *mapname*, key *map key*: bad

  The map entry is malformed, and the automounter cannot interpret it. Recheck the entry; perhaps there are characters in it that need escaping.

- *mapname*: *yp_err*

  Error in looking up an entry in a YP map.

- *hostname*: exports: *rpc_err*

  Error getting share list from *hostname*. This indicates a server or network problem.

- host *hostname* not responding

- *hostname*:*filesystem* server not responding

- Mount of *hostname*:*filesystem* on *mountpoint*: *reason*

  You will see these error messages after the automounter attempted to mount from *hostname* but either got no response or failed. This may indicate a server or network problem.

- *mountpoint* - *pathname from hostname: absolute symbolic link*

  When mounting a hierarchy, the automounter has detected that *mountpoint* is an absolute symbolic link (begins with "/"). The content of the link is *pathname*. This may have undesired consequences on the client. The contents of the link may be /usr.

- Cannot create socket for broadcast rpc: *rpc_err*

- Many_cast select problem: *rpc_err*

- Cannot send broadcast packet: *rpc_err*

- Cannot receive reply to many_cast: *rpc_err*

All these error messages indicate problems attempting to `ping` servers for a replicated file system. This may indicate a network problem.

- `trymany: servers not responding: reason`

  No server in a replicated list is responding. This may indicate a network problem.

- `Remount hostname:filesystem on mountpoint: server not responding`

  Attempted remount after unmount failed. Indicates a server problem.

- `NFS server (pidn@mountpoint) not responding still trying`

  An NFS request made to the automount daemon with PID *n* serving *mountpoint* has timed out. The automounter may be temporarily overloaded or dead. Wait a few minutes; if the condition persists, the easiest solution is to reboot the client. If not, you have to exit all processes that use automounted directories (or, change to a non automounted directory in the case of a shell), kill the current automount process and restart it again from the command line. If this does not work, you have to reboot.

14

The Sun Yellow Pages Service

14

The Sun Yellow Pages Service

4.0.3 Inclusion Instructions

Replace all of Chapter 14 in *System and Network Administration* with the following chapter.

Chapter 14 explains how to administer the Yellow Pages (YP) distributed network lookup service. Read the chapter if you administer a server, standalone, or client on a network running YP. Information covered includes:

- The YP environment
- Setting up YP servers
- Setting up a YP client
- Creating and updating maps
- Using the automounter
- Fixing YP problems
- How YP affects security

14.1. The YP Environment

YP service is based on information contained in the series of YP *maps*. Maps are built from administrative databases, which also form the basis for the ASCII files traditionally found in `/etc`. Each YP map has a `mapname` used by programs to access it. On a network running YP, at least one YP server maintains a set of YP maps for other hosts to query.

The YP Domain

A *YP domain* is a named set of YP maps. YP maps are located in a subdirectory of `/var/yp` on the YP server. This subdirectory will have the same name as the YP domain. For example, the directory `/var/yp/podunk.edu` contains maps for the "literature" domain.

YP Machine Types

By definition, a YP server is a machine with a disk storing a set of YP maps that it makes available to network hosts. You do not have to make your file server the YP server, unless, of course, it is the only machine on the network with a disk.

YP servers come in two varieties, master and slave. The machine designated as YP *master server* contains the master set of maps that you update as necessary. It also runs all the necessary programs to run YP. If you have only one YP server on your network, designate it as the master server.

You can designate additional YP servers on your network as *slave servers*. The slave server has an additional set of YP maps, which the master updates whenever you update the master's maps.

A server may be a master with regard to one map, and a slave with regard to another. However, randomly assigning maps to YP servers can cause a great deal of confusion. You are strongly urged to make a single server the master for all the maps you create within a single domain. The examples in this chapter assume that one server is the master for all maps in the domain.

YP clients run processes that request data from maps on the servers. Clients do not care which server is the master, since all YP servers should have the same information. The distinction between master and slave server only applies to where you make the updates.

To find out which YP server is currently providing service to a client, use the `ypwhich` command as follows:

```
% ypwhich hostname
```

where *hostname* is the name of the client.

YP Maps

Information in YP maps is organized in a format similar to databases, that of the SunOS `dbm` files. The `ypfiles`(5) and `dbm`(3) man pages completely explain the `dbm` file format.

As in all `dbm` files, a YP map has an identifying *mapname* and is implemented with two files, *mapname*`.dir` and *mapname*`.pag`. The file ending in `.pag` contains the actual map entries.

Each YP map contains a set of values and their associated keys, which programs use to look up the values. For example, machines query the map `hosts.byname` to find a machine's Internet address. Like the `/etc/hosts` file, `hosts.byname` is based on information in the hosts database.

SunOS Release 4.0.3 supplies a default group of YP maps in the subdirectory *YP_domainname* of the directory `/var/yp`. Table 14-1 lists these maps.

Table 14-1 *A Basic Set of YP Maps*

bootparams	netgroup.byhost	passwd.byuid
ethers.byaddr	netgroup.byuser	protocols.byname
ethers.byname	netgroup	protocols.bynumber
group.bygid	netid.byname	publickey.byname
group.byname	netmasks.byaddr	publickey
hosts.byaddr	networks.byaddr	rpc.bynumber
hosts.byname	networks.byname	services.byname
mail.aliases	passwd.byname	ypservers
mail.byaddr		

You may want to use all these maps or only specific maps. YP can also serve any number of maps you create or add when you install other software products.

Most of the default maps are built from the same network databases as the familiar administrative files in `/etc`. YP services make updating these databases much simpler. You no longer have to change administrative files on every machine each time you modify the network environment. For example, when you add a new machine to a network without YP, you have to update `/etc/hosts` on every machine on the network. With YP, you update the `hosts.byname` and `hosts.byaddr` maps on the YP master only. The master updates the YP slaves, if any. Then programs that previously consulted `/etc/hosts` send a remote procedure call to the YP servers for the same information. The YP server refers to `hosts.byaddr` and `hosts.byname`, then sends the requested information to the client.

You can use the `ypcat` command to display the values in a map. Its basic format is

```
ypcat mapname
```

where *mapname* is the name of the map you want to examine. The rest of this chapter and the `ypcat`(8) man page describe more options for `ypcat`.

You can use the `ypwhich` command introduced earlier to find out which server is the master of a particular map. Type the following

```
% ypwhich -m map.name
```

where *map.name* is the name of the map whose master you want to find. SunOS responds by displaying the name of the server. For complete information, refer to the `ypwhich`(8) man page.

The following table describes the default YP maps, information they contain, and whether SunOS consults the corresponding administrative files when YP is running.

Table 14-2 *YP Map Descriptions*

Map Name	Corresponding Admin File?	Description
`bootparams`	`bootparams`	Contains pathnames of files clients need during booting: `root`, `swap`, possibly others. With YP, SunOs does not consult `/etc/bootparams` after map is created.
`ethers.byaddr`	`ethers`	Contains machine names and Ethernet addresses. SunOS does not consult `/etc/ethers` after map is created.
`ethers.byname`	`ethers`	Same as `ethers.byaddr`, but machine name appears twice in the key.
`group.bygid`	`group`	Contains group security information with group ID as key. With YP, SunOS consults local `/etc/group` first, then map.
`group.byname`	`group`	Contains group security information with group name as key. With YP, SunOS consults local `/etc/group` first, then map.
`hosts.byaddr`	`hosts`	Contains host name, user name, and IP address. With YP, SunOS uses map except during booting, when it consults `/etc/hosts.`
`hosts.byname`	`hosts`	Contains hostname and IP address. With YP, SunOS uses map except during booting, when it consults `/etc/hosts.`
`mail.aliases`	`aliases`	Contains aliases and mail addresses. With YP, SunOS consults local `/etc/aliases` first, then map.
`mail.byaddr`	`aliases`	Contains machine name and alias. With YP, SunOS consults local `/etc/aliases` first, then map.
`netgroup.byhost`	`netgroup`	Contains group name and host names. SunOS does not consult `/etc/netgroup` after map is created.
`netgroup.byuser`	`netgroup`	Same as `netgroup.byhost`.
`netgroup`	`netgroup`	Contains group name, user name, host name. SunOS never consults `/etc/netgroup` after map is created.
`netid`	None	Contains machine name and mail address (including IP domain name).
`netmasks.byaddr`	`netmasks`	Contains network mask to be used with IP subnetting. SunOS never consults `/etc/netmasks` after map is created.

Table 14-2 *YP Map Descriptions— Continued*

Map Name	Corresponding Admin File?	Description
`networks.byaddr`	`networks`	Contains names of networks known to your system and their IP addresses. SunOS never consults `/etc/networks` after map is created.
`networks.byname`	`networks`	Same as `networks.byaddr`
`passwd.byname`	`password`	Contains password information with user name as key. With YP, SunOS consults local `/etc/passwd`, then map.
`passwd.byuid`	`password`	Contains password information with user ID as key. With YP, SunOS consults local `/etc/passwd`, then map.
`protocols.byname`	`protocols`	Contains network protocols known to your network. SunOS never consults `/etc/protocols` after map is created.
`protocols.bynumber`	`protocols`	Contains network protocols and identifying information. SunOS never consults `/etc/protocols` after map is created.
`publickey.byname`	`publickey`	Contains public and secret keys. SunOS never consults `/etc/publickey` after map is created.
`publickey`	`publickey`	Same as `publickey.byname`.
`rpc.bynumber`	`rpc`	Contains number and name of rpc calls known to your system. SunOS never consults `/etc/rpc` after map is created.
`services.byname`	`services`	Lists Internet services known to your network. SunOS never consults `/etc/services` after map is created.
`ypservers`	`None`	Lists YP servers known to your network.

YP Binding

YP clients get information from the YP server through the binding process—a slightly different process from NFS binding. Here is what happens during YP binding:

1. A program run on the client requests information that is normally provided by a YP map.
2. A C library routine on the client looks in the directory `/var/yp/binding/`*domainname* to find out which server the client should bind to.

3. The client library routine initiates binding by forwarding the request to the appropriate YP server.
4. The `ypserv` daemon on the YP server handles the request by consulting the appropriate map.
5. `ypserv` then sends the requested information back to the client.

YP and the Concept of Naming

YP is one example of a network service that performs *naming*. At its simplest, naming is the process of looking up information about an entity in a file—which is what you do manually in an environment without YP. Within the YP environment, naming occurs when a program uses YP to find out the identity of, or information about, an object. YP gets this information, such as a host's Internet address, the mailing address of a user, or whether a user is a member of a netgroup, from the appropriate YP map.

The most common example of naming is called *name to address mapping*. This occurs when a program on one host needs to access another machine, and uses YP to locate the remote host's Internet address. In this case, YP consults the `host.byname` map. Other examples of naming occur when one machine uses YP to find out what type of services another machine provides, such as mail services.

YP provides naming solely within your local domain. To provide naming service across domains, your local domain must run the Domain Name Server (DNS). DNS is a network application service like NFS and YP. It is part of SunOS, but it is not installed during the `suninstall` process. You must set DNS up separately, as explained in Chapter 23, "Domain Name Service."

Commands Used for Maintaining YP

In addition to maps, YP service also includes specialized daemons, system programs, and commands, which are introduced below. The remainder of this chapter describes them in detail, as do the *SunOS Reference Manual* and *Network Programming* manual.

`ypserv`	Looks up requested information in a map. `ypserv` is a daemon that runs on YP servers with a complete set of maps. At least one `ypserv` daemon must be present on the network for YP service to function.
`ypbind`	Initiates binding; it is the YP binder daemon. `ypbind` is present on both clients and servers. It initiates binding by trying to find a `ypserv` process that serves maps within the requesting client's domain. `ypserv` must run on each YP server. `ypbind` must run on all clients.
`ypinit`	Automatically creates maps for a YP server from files located in `/etc`. It also constructs the initial maps that are not built from files in `/etc`, such as `ypservers`. Use `ypinit` to set up the master YP server and the slave YP servers for the first time.

`make`	Updates YP maps. It is a version of the `make` command that reads the Makefile in `/var/yp`. You can use `make` to update all maps based on the files in `/etc` or to update individual maps. The man page `ypmake`(8) describes `make` functionality for YP.
`makedbm`	Enables you to update maps that are not built from the Makefile in `/var/yp`. `makedbm` takes an input file and converts it into `dbm .dir` and `.pag` files—valid YP maps. You can also use `makedbm` to "disassemble" a map, so that you can see the key-value pairs that comprise it.
`ypxfr`	Moves a YP map from one server to another, using YP itself as the transport medium. You can run `ypxfr` interactively, or periodically from a `crontab` file. (See Chapter 8 for information about running `crontab`.)
`yppush`	Copies a new version of a YP map from the YP master server to its slaves. You run it on the master YP server.
`ypset`	Tells a `ypbind` process to get YP services for a domain from a named YP server. This is not for casual use.
`yppoll`	Tells which version of a YP map is running on a server that you specify.
`ypcat`	Displays the contents of a YP map.
`ypmatch`	Prints the value for one or more specified keys in a YP map. You cannot specify which YP server's version of the map you are seeing.
`ypwhich`	Shows which YP server a client is using at the moment for YP services, or which YP server is master of a particular map.
`ypupdated`	Changes YP information. It is a daemon normally started up by `inetd`.

14.2. Preparing the YP Domain

Before you configure machines as YP servers or clients, you must prepare the YP domain by:

- Giving it a name
- Designating which machines will serve or be served by the YP domain
- Preparing the network databases from which the maps in the YP domain are built.

Setting the Domain Name and Host Name

The first step in setting up YP is to give your YP domain a name. Next, make a list of network hosts that will give or receive YP service. Determine which machine should be master server for the YP domain. List which hosts on the network, if any, are to be slave servers. Finally, list the all hosts that are to be YP clients. You probably will want all hosts in your network's administrative domain to to receive YP services. If this is the case, give the YP domain the same name as the network administrative domain. Refer back to Chapter 12, "The Sun Network Environment," for a full description of network domain names.

Before a machine can use YP services, its correct YP domain name and host name must be specified in two booting scripts: `/etc/rc.local` and `/etc/rc.boot`. `suninstall` automatically updates these scripts for every machine you have it configure. Thereafter, when these machines boot, their host names and YP domain names are already set.

However, if you do not run `suninstall`, you must manually set the YP domain name and host name on every machine to use YP services. Here is how to do this for the server.

1. Log in as superuser.
2. Edit `/etc/rc.local` and find the line in the file that begins:

   ```
   /bin/domainname
   ```

3. Add your YP domain name after `domainname`, for example

   ```
   /bin/domainname ypdomain.dancer.com
   ```

   Then close the file.

5. Edit `/etc/rc.boot`. Find the section that looks like:

   ```
   HOME=/; export HOME
   hostname=
   ```

   The value after `hostname=` should be the name of the master server. If the name is different or does not appear, type in the server's name. Then close the file.

6. Reboot the master server, then finish setting it up, as described in the next subsection.

Repeat the process above for any slave servers and for all clients of the YP domain.

Changing the YP Domain Name

If you need to change the default domain name set during `suninstall`, run the `domainname` command on the command line. The syntax of `domainname` is:

```
% /usr/bin/domainname name-of-domain
```

Running `domainname` without an argument displays the local machine's YP domain name. To change a machine's default domain, log in to it as superuser.

Then run

```
# domainname name-of-domain
```

Supply the new domain name for the *name-of-domain* argument.

Preparing Network Databases for YP Service

Network databases exist in two forms, YP maps and administrative files in `/etc`. Until you implement YP, each host accesses these databases in the form of the local `/etc` files, which could contain potentially out-of-date information. Therefore, it is a good idea to have all hosts on your network access the YP maps.

You need to take two steps to enforce this policy. First, you must ensure that the `ypbind` daemon is running on all hosts—both servers and clients. Second, you must abbreviate or eliminate the files in `/etc` that are built from the same databases as the YP maps. These files are:

```
passwd
hosts
ethers
group
networks
protocols
services
netgroup
aliases
netmasks
```

Preparing Files on YP Clients

Here are the changes, if any, that you need to make to `/etc` files on all YP clients. Note that the files `networks`, `protocols`, `ethers`, `services`, and `netgroup` need be present on any YP clients.

Preparing `hosts.equiv`

The `hosts.equiv` file does correspond to an equivalent YP map. However, you can add escape sequences to it that reference YP. This reduces problems with `rlogin` or `rsh`, which are sometimes caused by the communicating machines having different `/etc/hosts.equiv` files.

To let anyone log on to a machine, have `hosts.equiv` contain a single line, with only the character, + (plus) on it. Alternatively, you can exercise more control over logins by designating trusted groups. Both the + character and trusted groups are described in Chapter 13. YP assumes that the trusted group name appearing after the @ sign is a netgroup defined in the map `netgroups`.

If a machine's `hosts.equiv` does not have escape sequences, remote access is determined by the entries in the file. YP maps are not consulted.

Preparing `.rhosts`

`.rhosts` also does not have an equivalent YP map. Chapter 13 fully discusses its format and restrictions. Because it controls remote root access to the local machine, unrestricted access to `.rhosts` is not recommended. Make the list of trusted hosts explicit, or use netgroup names for the same purpose. You can not use secondary hostnames in your `.rhosts`, `hosts.equiv`, or `netgroup`

files. You can, however, use secondary hostnames in `/etc/hosts`. All of the above files are related in that they enable local machines to access remote machines in some fashion.

Preparing `hosts`

In order to receive YP service, the client's `hosts` file must contain entries for the local host's name, and the local loopback name. SunOS accesses `/etc/hosts` at boot time before the client's `ypbind` daemon starts up. Once `ypbind` is running, `/etc/hosts` is not accessed at all. Rather, SunOS consults information in the `hosts.byaddr` and `hosts.byuser` maps. Refer to Chapter 12 for more information on `/etc/hosts`.

Preparing `passwd`

Programs first consult a YP client's local `/etc/passwd` file to determine access permission before consulting the YP maps. Therefore, every client's `/etc/passwd` should contain entries for root and the primary users of the machine. `/etc/passwd` should also have the + escape entry to force the use of the `passwd.byname` and `passwd.byuid` YP maps. If there is no + entry, programs will not consult the YP maps at all.

You may also want to add an entry for "daemon," to allow file-transfer utilities to work, and for "operator," to let a dump operator log in. A sample YP client's `/etc/passwd` file looks like:

```
root:9wxntql2tHT.k:0:1:Operator:/:/bin/csh
nobody:*:-2:-2::/:
daemon:*:1:1::/:
sys:*:2:2::/:/bin/csh
bin:*:3:3::/bin:
uucp:*:4:4::/var/spool/uucppublic:
news:*:6:6::/var/spool/news:/bin/csh
sync::1:1::/:/bin/sync
stefania:7kjDXZD/Hug2s:624:20:Stefania:/home/dancer/samba:/bin/csh
+::0:0:::
```

The last line informs the library routines to use the YP maps. If you remove that line, you will disable YP password access.

Earlier entries in `/etc/passwd` take precedence over, or *mask*, later entries with the same user name or same user ID. Therefore, please note the order of the entries for the daemon and sync user names (which have the same user ID) and duplicate it in your own file.

Preparing `group`

For a YP client, you can reduce `/etc/group` to a single line:

```
+:
```

The + escape sequence forces all translation of group names and group IDs to be made via the YP maps. This is the recommended procedure.

Preparing Network Databases on the Master Server

Before running the programs that create the YP master server, you need to take several precautions regarding the administrative files based on the network databases. First, check the following files in the new server's `/etc` directory to make sure they reflect an up-to-date picture of your system:

```
passwd
hosts
ethers
group
networks
protocols
services
```

An entry for the daemon user ID must be present in `/etc/passwd` for both master and slave server. Furthermore, that entry must precede any other entries with the same user ID, as described previously for setting up the client's `/etc/passwd` file.

If you know how you want to set up the groups in the `netgroup` database, edit `/etc/netgroup` before you run `ypinit`. Otherwise, `ypinit` makes an empty `netgroup` map. Finally, make sure your `aliases` database is complete by verifying that `/etc/aliases` contains all aliases for every host the master is to serve. Refer to Chapter 16 and the `aliases`(5) man page for more information.

14.3. Setting Up YP Servers and Clients

This section explains how to set up and administer machines on your network to use YP services. Topics covered include:

- Setting up a master YP server
- Setting up a slave YP server
- Setting up a YP client

You can initiate YP service when installing a new version of the operating system or by manually setting up YP on a running network.

Setting Up a Master YP Server

This next section explains how to set up the master server for your YP domain. The steps below apply to both a YP master server configured through `suninstall` and a master that you want to configure manually.

Starting YP Service with `ypinit`

`ypinit` builds a fresh set of YP maps on the master or slave server, depending on the options you give it. It builds the maps from the files in `/etc` and from information it receives at runtime. (The `ypservers` map is built in the latter way.) The next procedure shows how to use `ypinit` to designate servers and build the maps for your YP domain.

1. Log in as superuser on the new master server.
2. Type

```
# cd /var/yp
# /usr/etc/ypinit
```

3. `ypinit` asks whether you want the procedure to exit at the first non-fatal error or continue despite non-fatal errors.

   If you choose to have the procedure die, you can fix the problem and restart `ypinit`. This is recommended if you are running `ypinit` for the first time. If you prefer to continue, you can try to fix all problems that may occur by hand or fix some, then restart `ypinit`.

4. `ypinit` prompts for a list of other hosts to become YP servers. List all YP slave servers, even if at some point one might become the master server. You need not add any other hosts, but if you expect to set up up more YP servers, add them now. You will save yourself time later; and there is little runtime penalty for designating all your servers now.

5. If you previously disabled YP, after you finish running `ypinit`, edit the new YP server's `/etc/rc.local` file. Remove the comments (# signs) from the lines that refer to the `ypbind` command. These lines are

```
#if [-f /usr/etc/ypbind]; then
    .
    .
fi
```

   You must do this in order to have YP work on the server.

For security reasons, you may want to restrict access to the master YP machine to a smaller set of users than that defined in its `/etc/passwd` file. To do this, copy the complete file and give the copy a name and path other than `/etc/passwd`. Edit out undesired users from the remaining `/etc/passwd`. For a security-conscious system, this smaller file should not include the YP escape entry (+) discussed later in this chapter.

Creating the Master Server

Now that the master maps are created, you can actually create the master server and begin YP service. To do this, you have to run `ypinit` again, then start up `ypserv` on the server. When a client requests information from the server, `ypserv` is the daemon that actually looks up the data in the YP maps.

1. From `/var/yp`, type the following

```
# /usr/etc/ypinit -m
```

   to set up the master server.

2. Next, type

```
# /usr/etc/ypserv
```

   to start providing YP services. The next time you boot the server, `ypserv` will automatically start up from `/etc/rc.local`.

Setting Up a YP Slave Server

Your network can have one or more YP slave servers. Before actually running `ypinit` to create the slave servers, you should take several precautions.

The domain name for each YP slave must be the same as for the YP master server. `suninstall` sets this up automatically for each machine you designate as a YP slave. If you did not set up YP through `suninstall`, use the `domainname` command without arguments on each YP slave to make sure they are consistent with the master server. Make any changes to the domain name necessary, as described in the previous section, "Setting Up the YP Domain." Also, do not forget to set each slave server's host name, if you did not set up YP through `suninstall`.

As you did with the YP master server, you must also check every slave server's `/etc/passwd` file. Make sure that an entry for the daemon user name exists and that it precedes other entries with the same user ID.

Finally, you must make sure that the network is working properly before you set up a slave YP server. In particular, check that you can use `rcp` to send files from the master YP server to YP slaves.

Now you are ready to create a new slave server.

1. Log in to the slave server as superuser.
2. Change directory to `/var/yp`.
3. Type the following:

```
# /usr/etc/ypinit -s master
```

   where *master* is the host name of an existing YP server. Ideally, the named host really is the master server, but it can be any host with a stable set of YP maps.

4. `ypinit` will not prompt you for a list of other servers, as it does when you create the master server. However, it does let you choose whether or not to halt at the first non-fatal error. `ypinit` then creates a copy of the master's YP map set in the slave server's `/var/yp/`*domainname* directory.
5. When `ypinit` terminates, make copies of the following files:

```
/etc/passwd
/etc/hosts
/etc/group
/etc/networks
/etc/protocols
/etc/netgroup
/etc/services
```

   For example, you might type:

```
# cp /etc/passwd /etc/passwd-

OR

# cp /etc/passwd /etc/passwd.old
```

6. Edit the original files (not those with the - or `.old` extension) as described in the previous section, "Preparing Files for YP Service." This ensures that processes on the slave server actually use the YP services, rather than files in the local `/etc`. In this way, you ensure that the YP slave server is also a YP client.
7. Back up copies of the edited files, as well. For instance, you might type:

```
# cp /etc/passwd /etc/passwd+
```

8. Type

```
# /usr/etc/ypserv
```

to begin YP services on the slave server. The next time you reboot the YP slave, `ypserv` will start automatically from `/etc/rc.local`.

Repeat the procedures above for each machine you want configured as a YP slave server.

Setting Up a YP Client

The first step in creating the YP client is to declare it as such to the network. If you create the YP client through `suninstall`, then you simply specify it as such on the client form. Remember that machines with disks, including file or other types of servers, can be configured as YP clients.

To add a new machine to an already running network, run the `setup_client` program, as described in Chapter 13. Use the *yp_type* argument of `setup_client` to specify the YP services the machine is to give or receive: `master`, `slave`, or `client`.

Once you have established the machine as a YP client, do the following:

1. Edit the client's local files, as described in "Preparing Files for YP Service," if you have not done so already.
2. Add entries for the client in the following files:

```
/etc/ethers
/etc/hosts
/etc/bootparams
/etc/netgroup (If applicable)
```

3. Have the client use the `yppasswd` command to create a new password in the `yppasswd` maps.

4. Type

```
% ps -aux
```

to see if `ypbind` is running. If it is not, start it, then check `/etc/rc.local` to make sure that you have removed the commenting from the `ypbind` entry.

With the relevant files in `/etc` abbreviated and `ypbind` running, the processes on the machine will be clients of the YP servers.

At this point, you must have configured a YP server on the network. Otherwise, processes on the client hang if no YP server is available while `ypbind` is running.

Note the possible alterations to files in the client's `/etc` directory, as described in "Preparing Files for YP Service." Because some files may be modified or may no longer exist, it is not always obvious how the files corresponding to YP maps are being used. Refer to the man pages for `passwd`, `hosts`, `netgroup`, `hosts.equiv`, and `group` These entries describe how the escape conventions for each file force data to be included or excluded from the equivalent YP map.

In particular, notice how changing passwords in the `/etc/passwd` file or running the `passwd` command only affects the local client's environment. To change the YP password maps, you must run the `yppasswd` command. Its syntax is:

```
% yppasswd login_name
```

When you type the above command, `yppasswd` prompts you to type your new password twice, as does the local `passwd` command. However, when you have successfully responded, `ypasswd` puts your new password in the `passwd.byname` and `passwd.byuid` maps. Your YP password can be different from the password on your own machine.

14.4. Administering YP Maps

This section describes how to maintain the maps of an existing YP domain. Subjects discussed include:

- Updating YP maps
- Propagating a YP map
- Adding maps to an additional YP server
- Moving the master map set to a new server

Updating Existing Maps

After you have installed YP, you will discover that some maps require frequent updating while others never need to change. For example, the password and host-related maps are guaranteed to change constantly on a large company's network. On the other hand, the netmasks and protocols-related maps probably will change little, if at all.

When you need to update a map, you use one of two updating procedures, depending on whether the map is standard or non-standard. A *standard* map is a map in the default set created by `ypinit` from files in `/etc`. *non-standard* maps may be any of the following:

- Included with an application purchased from a vendor.
- Created specifically for your site.
- Existing in a form other than ASCII.

The following text explains how to use various updating tools. In practice, you probably will only use them if you add non-standard maps or change the set of YP servers after the system is already up and running.

Modifying Maps based on `/etc` Files

Use the following procedure for updating all *standard* maps.

1. Become superuser on the master server. (Always modify YP maps on the master server.)
2. Edit the file in `/etc` that corresponds to the map you want to change. (Refer back to Table 15-2 if you are not sure of the corresponding file.)
3. Type the following:

```
# /var/yp
# make
```

The `make` command will then update your map according to the changes you made in its corresponding file. After updating a map, you have to propagate it, as explained in the next section, "Propagating a YP Map."

Creating and Modifying Non-Standard Maps

To update a non-standard map, you edit its corresponding ASCII file. Then you rebuild the updated map using the `/usr/etc/yp/makedbm` command. (The `makedbm`(8) man page fully describes this command.)

There are two different methods for using `makedbm`:

- Redirect the command's output to a temporary file, modify the file, then use the modified file as input to `makedbm`.
- Have the output of `makedbm` operated on within a pipeline that feeds into `makedbm` again directly. This is appropriate if you can update the disassembled map with either `awk`, `sed`, or a `cat` append.

You can use either of two possible procedures for creating new maps. The first uses an existing ASCII file as input; the second uses standard input.

Updating Maps Built from Existing ASCII Files

Assume that an ASCII file `/var/yp/mymap.asc` was created with an editor or a shell script on the YP master. You want to create a YP map from this file, and locate it in the `home_domain` subdirectory. To do this, you type the following on the master server:

```
% cd /var/yp
% /usr/etc/yp/makedbm mymap.asc home_domain/mymap
```

The mymap map now exists in the directory home_domain.

Adding entries to mymap, is simple. First, you must modify the ASCII file mymap.asc. (If you modify the actual dbm files without modifying the corresponding ASCII file, the modifications are lost.) Type the following:

```
% cd /var/yp
% <edit mymap.asc>
% /usr/etc/yp/makedbm mymap.asc home_domain/mymap
```

When you finish updating the map, propagate it to the slave servers, as described in the section "Propagating a YP Map."

Updating Maps Built from Standard Input

When no original ASCII file exists, create the YP map from the keyboard by typing input to makedbm, as shown below:

```
ypmaster% cd /var/yp
ypmaster% /usr/etc/ypmakedbm - home_domain/mymap
al ar
bl br
cl cr
<ctl D>
```

When you need to modify such a map, you use makedbm -u to disassemble the map and create a temporary ASCII intermediate file. You type the following:

```
% cd /var/yp
% /usr/etc/yp/makedbm -u home_domain/mymap > mymap.temp
```

The resulting temporary file mymap.temp, has one entry on each line. You can edit it as needed, using your preferred editing tools.

To update the map, you give the name of the modified temporary file to makedbm as follows:

```
% /usr/etc/yp/makedbm mymap.temp home_domain/mymap
% rm mymap.temp
```

When makedbm finishes, propagate the map to the slave servers, as described in the section "Propagating a YP Map."

The preceding paragraphs explained how to use some tools, but in reality almost everything you actually have to do can be done by ypinit and /var/yp/Makefile, unless you add non-standard maps to the database, or change the set of YP servers after the system is already up and running.

Whether you use the Makefile in /var/yp or some other procedure Makefile— is one of many possible— the goal is the same: a new pair of

well-formed dbm files must end up in the domain directory on the master YP server.

Propagating a YP Map

When you *propagate* a YP map, you move it from place to place—most often from the master to all YP slave servers. Initially `ypinit` propagates the maps from master to slaves, as described previously. From then on, you must transfer updated maps from master to slaves by running the `ypxfr` command. You can run `ypxfr` three different ways: periodically through the root `crontab` file; by the `ypserv` daemon; and interactively on the command line.

`ypxfr` handles map transference in tandem with the `yppush` program. `yppush` always runs on the master server. The Makefile in the `/var/yp` directory automatically runs `yppush` after you change the master set of maps.

`yppush`'s function is to copy, or *push* a new version of a YP map from the YP master to slave(s). After making the copies, `yppush` contacts each slave server in the `ypservers` map and sends it a "transfer map" request. When the request is acknowledged by the slave, the `ypxfr` program actually transfers the new map to the slave.

Using `crontab` with `ypxfr`

Maps have differing rates of change. For instance, `protocols.byname` may not change for months at a time, but `passwd.byname` may change several times a day in a large organization. When you schedule map transference through the `crontab` command, you can designate the intervals when individual maps are to be propagated.

To periodically run `ypxfr` at a rate appropriate for your map set, put the appropriate `ypxfr` entries in a copy of the `/var/spool/cron/crontabs/root` file. Then run the `crontab` command to give the new file to the `cron` daemon. `ypxfr` contacts the master server and transfers the map only if the master's copy is more recent than the local copy. (Refer to Chapter 8 for information about creating `crontab` files through the `crontab` command.)

Maps with unique change characteristics can be checked and transferred by explicitly invoking `ypxfr` within `/var/spool/cron/crontabs/root`.

Using Shell Scripts with `ypxfr`

As an alternative to creating separate `crontab` entries for each map, you may prefer to have `/var/spool/cron/crontabs/root` run a shell script that periodically updates all maps. SunOS provides sample map updating shell scripts, `ypxfr_1perday`, `ypxfr_1perhour`, and `ypxfr_2perday` in the directory `/usr/etc/yp`. You can easily modify or replace these shell scripts to fit your site's requirements. Here is the default `ypxfr_1perday` shell script:

```
#! /bin/sh
#
# ypxfr_1perday.sh - Do daily yp map check/updates
#

PATH=/bin:/usr/bin:/usr/etc:/usr/etc/yp:$PATH
export PATH

# set -xv
ypxfr group.byname
ypxfr group.bygid
ypxfr protocols.byname
ypxfr protocols.bynumber
ypxfr networks.byname
ypxfr networks.byaddr
ypxfr services.byname
ypxfr ypservers
```

This shell script lists the maps you probably will want to update once per day.

Run the same shell scripts in `/var/spool/cron/crontabs/root` on each slave server configured for the YP domain. Alter the exact time of execution from one server to another to prevent the checking from bogging down the master.

If you want to transfer the map from a particular slave server, use the `-h`*host* option of `ypxfr` within the shell script. Here are the commands you put in the script:

```
cd /var/yp
/usr/etc/yp/ypxfr -h host [-c] mapname
```

where *host* is the name of the server with the maps you want to transfer, and `mapname` is the name of the requested map. If you use the `-h` option without specifying *host*, `ypxfr` will try to get the map from the master server.

You can use the `-s`*domain* option to transfer maps from another domain to your local domain. These maps should be essentially the same across domains. For example, two YP domains might share the same `services.byname` and `services.byaddr` maps.

Directly Invoking `ypxfr`

The third method of invoking `ypxfr` is to run it as a command. Typically, you do this only in exceptional situations — for example when setting up a temporary YP server to create a test environment, or when trying to quickly get a YP server that has been out of service consistent with the other servers.

Logging ypxfr's Activities

ypxfr's transfer attempts and the results can be captured in a log file. If /var/yp/ypxfr.log exists, results are appended to it. No attempt to limit the log file is made. To turn off logging, remove the log file.

Propagating a YP Map to Another Domain

You can propagate a YP map to another domain, but to do so, you must run the Domain Name System. Chapter 23, "Domain Name Service," explains how to do this.

Adding New YP Maps to the Makefile

Adding a new YP map entails getting copies of the map's dbm files into the /var/yp/*domain_name* directory on each of the YP servers in the domain. The actual mechanism w described above in "Propagating a YP Map". This section only describes how to update the Makefile so that propagation works correctly.

After deciding which YP server is the master of the map, modify /var/yp/Makefile on the master server so that you can conveniently rebuild the map. Actual case-by-case modification is too varied to describe here, but typically a human-readable ASCII file is filtered through awk, sed, and/or grep to make it suitable for input to makedbm. Refer to the existing /var/yp/Makefile for examples, and the description of make in the *Programming Utilities and Libraries* manual for full information.

Use the mechanisms already in place in /var/yp/Makefile when deciding how to create dependencies that make will recognize; specifically, the use of .time files allows you to see when the Makefile was last run for the map.

To get an initial copy of the map, you can run yppush on the YP master server. The map must be globally available before clients begin to access it. If the map is available from some YP servers, but not all, you will see unpredictable behavior from client programs.

Adding a New YP Server to the Original Set

After YP is running, you may need to create a YP slave server that you did not include in initial set given to ypinit. The following procedure explains how to do this:

Log in to the master server as superuser, and follow these directions.

1. Go to the YP domain directory by typing:

```
# cd /var/yp/domain_name
```

2. Add the new server's name to the ypservers map. Disassemble ypservers, as follows:

```
# /usr/etc/yp/makedbm -u ypservers > /tmp/temp_file
```

   makedbm converts ypservers from dbm format to the temporary ASCII file /tmp/*temp_file*.

3. Edit /tmp/*temp_file* using your preferred text editor. Add the new slave server's name to the list of servers. Then close the file.

4. Run the `makedbm` command with *temp_file* as the input file and `ypservers` as the output file.

```
# /usr/etc/yp/makedbm /tmp/temp_file ypservers
```

Here `makedbm` converts `ypservers` back into `dbm` format.

5. Set up the new slave server's YP domain directory by copying the YP map set from the master server. To do this, remote log in to the new YP slave, and run the `ypinit` command:

```
# rlogin ypslave
ypslave# cd /var/yp
ypslave# /usr/etc/ypinit -s ypmaster
```

6. Verify that the `ypservers` map is correct (since there is no ASCII file for `ypservers`) by typing the following:

Note: If a host name is not in *ypservers* it will not be warned of updates to the YP map files.

```
ypslave# cd /var/yp/domain_name
ypslave# /usr/etc/yp/makedbm -u ypservers
```

Here `makedbm` will display each entry in `ypservers` on your screen. When you are finished, complete the steps described above in the section "Setting Up A Slave Server."

Changing a Map's Master Server

To change a map's master, you first have to build it on the new YP master. The old master's name occurs as a key-value pair in the existing map. Therefore, using the existing copy at the new master or transferring a copy to the new master with `ypxfr` is insufficient. You have to reassociate the key with the new master's name. If the map has an ASCII source file, you should copy it in its current version to the new master.

Here are instructions for remaking a sample YP map called `jokes.bypunchline`.

1. Log in to the new master as superuser, and type the following:

```
newmaster# cd /var/yp
```

2. `/var/yp/Makefile` must have an entry for the new map before you specify the map to `make`. If this isn't the case, edit the Makefile now.

3. Type the following:

```
newmaster# make jokes.bypunchline
```

4. If the old master will remain a YP server, `rlogin` in to it, and edit `/var/yp/Makefile`. Comment out the section of `/var/yp/Makefile` that made `jokes.bypunchline` so that it is no longer made there.

5. If `jokes.bypunchline` only exists as a `dbm` file, remake it on the new master by disassembling a copy from any YP server, then running the disassembled version through `makedbm`:

```
newmaster# cd /var/yp
newmaster# ypcat -k jokes.bypunchline |\
/usr/etc/yp/makedbm - mydomain/jokes.bypunchline
```

After making the map on the new master, you must send a copy of it to the other slave servers. However, do not use `yppush`— the other slaves will try to get new copies from the old master, rather than the new one. A typical method for circumventing this (you may find others) is to transfer a copy of the map from the new master back to the old master. Become superuser on the old master server and type:

```
oldmaster# /usr/etc/yp/ypxfr -h newmaster jokes.bypunchline
```

Now it is safe to run `yppush`. The remaining slave servers still believe that the old master is the current master; they will attempt to get the current version of the map from the old master. When they do so, they will get the new map, which names the new master as the current master.

If this method fails, you can try this cumbersome but sure-fire option. Log in as superuser on each YP server and execute the `ypxfr` command shown above. This will certainly work, but you should consider it the worst case solution.

14.5. Administering Users on a YP Network

SunOS Release 4.1 provides a number of features for administering YP in a secure environment. If your site requires tight security, refer to the *Security Features Guide*. You may want to use some of the features it discusses, such as secure file systems and C2-like security. If your network requires average security, refer first to Chapter 13. This next section covers only YP matters; it assumes you are familiar with the security information in Chapter. 13

How YP Maps Affect Security

Security on a system running YP depends on how programs consult the files in `/etc` on which the YP maps are based. A machine's local files are consulted first. These include

```
/etc/passwd
/etc/group
/etc/aliases
```

Then the programs consult maps in the YP domain that correspond to the local files. For example, a machine checks its own `/etc/aliases` file for mail aliases, then checks the `mail.aliases` YP map.

The `passwd` file is another example of how local files take precedence in a YP environment. When users run the `passwd` command to change their passwords, they change their machines' local `/etc/passwd` files ONLY. Suppose a user tries to log in to a host where he or she does not have an entry in the local `/etc/passwd`. Even if this `/etc/passwd` file has the + entry to pull in information from the YP password maps, the `passwd` command still prints the

error message

```
Not in passwd file.
```

The following files in each machine's `/etc` are considered global files:

```
/etc/hosts
/etc/networks
/etc/ethers
/etc/services
/etc/netmasks
/etc/protocols
/etc/netgroup
```

They contain network wide data. On a network with YP, a machine no longer accesses these files for information. It refers to the corresponding YP maps instead. However, when booting, each machine needs an entry in `/etc/hosts` for itself.

Changing the YP Password

Users must run the `yppasswd` command to change their passwords in the YP password file. Before they can do this, you must start the `yppasswdd` daemon by adding an entry for `yppasswdd` in `rc.local` on the master server for the password maps.

Go to the master server, edit `rc.local`, and add the following:

```
/usr/etc/rpc.yppasswdd /var/yp/domainname/passwd -m \
    passwd DIR=/var/yp/domainname
```

where *domainname* is the name of your YP domain directory. Then reboot the master server to start up the `yppasswdd` daemon.

To actually change the YP password, have the user type:

```
% yppasswd user_name
```

The system then prompts the user to type the old and new passwords, as the local `passwd` command does. Refer to the `yppasswdd`(8) man page for more information about the daemon and to `yppasswd`(1) for information about the `yppasswd` command.

How Netgroups Affect Security on a YP Network

On a network with YP, the `/etc/netgroup` file on the master YP server is used for generating three YP maps: `netgroup`, `netgroup.byuser` and `netgroup.byhost`. `netgroup` contains the basic information in `/etc/netgroup`. The two other YP maps contain information in a format that speeds lookup of netgroups given the host or user.

Various programs use the `/etc/netgroup`-based YP maps for permission checking during remote mount, login, remote login, and remote shell creation. These programs include: `login`, `mountd`, `rlogin`, and `rsh`. `login` consults them for user classifications if it encounters netgroup names in `/etc/passwd`. The `mountd` daemon consults them for machine classifications if it encounters

netgroup names in /etc/exports. rlogin and rsh consult the netgroup maps for both machine and user classifications if they encounter netgroup names in the /etc/hosts.equiv or /.rhosts file.

Refer to Chapter 12 for more information about etc/netgroup.

Adding a New User to a YP Server

Adding a new user to a network already running YP is not exactly the same as adding a new client to a network, as described in Chapter 13. It involves adding not only a new user but, possibly, a new client machine to the YP maps.

The first step in adding a new user to a YP network is to update the yppasswd file. Follow these procedures;

1. Log in as root on the master YP server.
2. Edit the master YP server's /etc/passwd file. Use the vipw command to add a new line to the password file, as follows: vipw(brings the password file into the vi editor and prevents anyone else from editing it until you are done)

```
# /usr/etc/vipw
```

Later the user's password file entry will be copied to the /etc/passwd in their client machine's / directory. Without an entry in the local /etc/passwd, the user cannot log in should YP fail.

The following example shows an entry in the YP password map passwd.byname.

```
roger:3u0mRdrJ4tEVs:1947:10:The Boss:/home/shams/roger:/bin/csh
```

Note that the fields in the YP password maps are similar to the local passwd file described in Chapter 13. Here are suggestions as to what these fields should contain:

Login name	Should be the same as user name in the local /etc/passwd file, and unique within the network domain.
Encrypted password	This is the password created by the yppasswd command. Have all new users run yppasswd. If a user forgets his or her password, you can make this field empty, enabling them to log in without a password and create a new one with passwd. An asterisk in this field matches no password.
User ID	A number that must be unique within the network domain, which you assign to this user. Failure to keep ID's unique prevents files on different machines from being moved between directories because the system will respond as if the directories are owned by two

different users. Also, file ownership may become confused when an NFS server exports a directory to an NFS client whose password file contains users with user IDs that match those of different users on the NFS server.

Group ID	IDs which you assigned to groups created in the local `/etc/group` files.
User Information	Same as the local `/etc/passwd` file.
Home Directory	Same as the local `/etc/passwd` file.
Login Shell	Same as the local `/etc/passwd` file.

3. After you have updated the master server's password file, propagate the password YP maps as follows:

```
# cd /var/yp
# make passwd
```

Note that if your site uses the C2 secure configuration option, it will split `passwd` into two files. In C2 secure environment, only processes running with superuser privileges can access the file containing the encrypted password.

Making the Home Directory

After you update the YP password file, create an entry for the user on the local machine. Then create a home directory, as shown in Chapter 13. Note that if the YP password maps have not yet been updated on the machine's YP server, the following error message appears when you attempt to change ownership of the home directory.

```
unknown user id:  username
```

In that case, you can use the following set of commands:

```
# cd /home/servername
# mkdir roger
# chown userid# roger
```

You use the ID number from the password file entry instead of login name to change the ownership of the user's home directory.

14.6. Fixing YP Problems

This section explains how to clear problems encountered on networks running YP. It has two parts, covering problems seen on a YP client and those seen on a YP server.

Debugging a YP Client

Before trying to debug a YP client, review the first part of the chapter, which explains the YP environment. Then look for the subheading in this section that best describes your problem.

Hanging Commands on the Client

The most common problem of YP clients is for a command to hang and generate console messages such as:

```
yp: server not responding for domain <domainname>.  Still trying
```

Sometimes many commands begin to hang, even though the system as a whole seems okay and you can run new commands.

The message above indicates that `ypbind` on the local machine is unable to communicate with `ypserv` in the domain *domainname*. This happens when a machine running `ypserv` has crashed. It may also occur if the network or YP server is so overloaded that `ypserv` cannot get a response back to the client's `ypbind` within the timeout period.

Under these circumstances, every client on the network will experience the same or similar problems. The condition is temporary in most cases. The messages will usually go away when the YP server reboots and restarts `ypserv`, or when the load on the YP server or network itself decreases.

However, commands may hang and require direct action to clear them. The following list describes the causes of such problems and gives suggestions for fixing them:

- The YP client has not set, or has incorrectly set, `domainname` on the machine. Clients must use a domain name that the YP servers know.

  On the client type `domainname` to see which domain name is set. Compare that with the actual domain name in `/var/yp` on the YP master server. If a machine's `domainname` is not the same as the server's, the machine's `domainname` entry in `rc.local` is incorrect. Log in as superuser, edit the client's `rc.local`, and correct the `domainname` entry. This assures domain name is correct every time the machine boots. Then set `domainname` manually by typing:

  ```
  # domainname good_domain_name
  ```

- If your domain name is correct and commands still hang, make sure your local network has at least one YP server machine. Some network domain consist of two or more local networks joined by routers. A client can automatically bind only to a `ypserv` process on its local network, not on another accessible network. At least one YP server for a client's domain must running on the client's local network. Two or more YP servers improve availability and response characteristics for YP services.

- If your local network has a YP server and commands still hang, make sure the server is up and running. Check other machines on your local network. If several clients also have problems, suspect a server problem. Try to find a client machine behaving normally, and type the `ypwhich` command on it.

If `ypwhich` does not respond, kill it and go to a terminal on the YP server. Type:

```
# ps ax | grep yp
```

Look for `ypserv` and `ypbind` processes. If the server's `ypbind` daemon is not running, start it up by typing:

```
# /usr/etc/ypbind
```

If a `ypserv` process is running, type

```
# ypwhich
```

on the YP server. If `ypwhich` does not respond, `ypserv` has probably hung, and you should restart it. While logged in as superuser, type the following:

```
# kill -9 [some pid # from ps]
# /usr/etc/ypserv
```

If `ps` shows no `ypserv` process running, start one up.

YP Service is Unavailable

When most machines on the network appear to be okay, but one client cannot receive YP service, that client may experience many different symptoms. For example, some commands appear to operate correctly while others terminate with an error message about the unavailability of YP. Other commands limp along in a backup-strategy mode particular to the program involved. Still other commands or daemons crash with obscure messages or no message at all. Here are messages a client in this situation may receive:

```
samba% ypcat myfile
ypcat: can't bind to YP server for domain <domainname>.
    Reason: can't communicate with ypbind.
```

```
% /usr/etc/yp/yppoll myfile
Sorry, I can't make use of the yellow pages.  I give up.
```

On the problem client, run **`ls -l`** on a directory containing files owned by many users, including some not in the client's `/etc/passwd` file—for example `/usr`. If the resulting display lists file owners not in the local `/etc/passwd` as numbers, rather than names, this also means that YP service is not working.

These symptoms usually indicate that the client's `ypbind` process is not running. Run **`ps ax`** and check for `ypbind`. If it you do not find it, log in as superuser and start it as follows:

```
# /usr/etc/ypbind
```

YP problems should disappear.

ypbind Crashes

If `ypbind` crashes almost immediately each time it is started, look for a problem in some other part of the system. Check for the presence of the `portmap` daemon by typing:

```
% ps ax | grep portmap
```

If it is not running, reboot.

If `portmap` itself will not stay up or behaves strangely, look for more fundamental problems. Check the network software in the ways suggested in the section on Ethernet debugging in Chapter 12, "The SunOS Network Environment."

You may be able to communicate with `portmap` on the problematic client from a machine operating normally. From the functioning machine, type:

```
% rpcinfo -p client
```

If `portmap` on the problematic machine is okay, `rpcinfo` produces the following output:

```
program vers proto   port
 100007    2   tcp   1024  ypbind
 100007    2   udp   1028  ypbind
 100007    1   tcp   1024  ypbind
 100007    1   udp   1028  ypbind
 100021    1   tcp   1026  nlockmgr
 100024    1   udp   1052  status
    .
    .
    .
```

Your machine will have different port numbers. The four entries for the `ypbind` process are:

```
 100007    2   tcp   1024  ypbind
 100007    2   udp   1028  ypbind
 100007    1   tcp   1024  ypbind
 100007    1   udp   1028  ypbind
```

If they are not displayed, `ypbind` has been unable to register its services. Reboot the machine and run `rpcinfo` again. If the `ypbind` processes are there and they change each time you try to restart `/usr/etc/ypbind`, reboot the system, even if the `portmap` daemon is running. If the situation persists after reboot, call Sun customer support for help.

`ypwhich` Displays are Inconsistent

When you use `ypwhich` several times on the same client, the resulting display varies because the YP server changes. This is normal. The binding of YP client to YP server changes over time when the network or the YP servers are busy. Whenever possible, the network stabilizes at a point where all clients get acceptable response time from the YP servers. As long as your client machine gets YP service, it does not matter where the service comes from. For example, one YP server machine gets its own YP services from another YP server on the network.

Debugging a YP Server

Before trying to debug your YP server, read the beginning part of this chapter about the YP environment. Then look in this subsection for the heading that most closely describes the server's problem.

Servers Have Different Versions of a YP Map

Because YP propagates maps among servers, occasionally you find different versions of the same map at YP servers on the network. This version discrepancy is normal if transient, but abnormal otherwise.

Most commonly, normal map propagation is prevented if it occurs when a YP server or router between YP servers is down. When all YP servers and the routers between them are running, `ypxfr` should succeed.

If a particular slave server has problems updating maps, log in to that server and run `ypxfr` interactively. If `ypxfr` fails, it will tell you why it failed, and you can fix the problem. If `ypxfr` succeeds, but you suspect it has occasionally failed, create a log file to enable logging of messages. As superuser type:

```
ypslave# cd /var/yp
ypslave# touch ypxfr.log
```

This saves all output from The output resembles the output `ypxfr` displays when run interactively, but each line in the log file is timestamped. (You may see unusual orderings in the timestamps. That is okay— the timestamp tells you when `ypxfr` started to run. If copies of `ypxfr` ran simultaneously but their work took differing amounts of time, they may actually write their summary status line to the log files in an order different from that which they were invoked. Any pattern of intermittent failure shows up in the log. When you have fixed the problem, turn off logging by removing the log file. If you forget to remove it, it will grow without limit.

While still logged in to the problem YP slave server, inspect the system `crontab` file, `/var/spool/cron/crontabs/root`, and the `ypxfr*` shell scripts it invokes. Typos in these files cause propagation problems, as do failures to refer to a shell script within `/var/spool/cron/crontabs/root`, or failures to refer to a map within any shell script.

Also, make sure that the YP slave server is in the map `ypservers` within the domain. If it is not, it still operates perfectly as a server, but `yppush` will not tell it when a new copy of a map exists.

If the YP slave server's problem is not obvious, you can work around it while you debug using `rcp` or `tftp` to copy a recent version the inconsistent map from any healthy YP server. Be sure not do this remote copy as root, but you can

probably do it while logged in as daemon. For instance, here is how you might transfer the map "busted:"

```
ypslave# chmod go+w /var/yp/mydomain
ypslave# su daemon
$ rcp ypmaster:/var/yp/mydomain/busted.\* /var/yp/mydomain
$ ^D
ypslave# chown root /var/yp/mydomain/busted.*
ypslave# chmod go-w /var/yp/mydomain
```

Here the * character has been escaped in the command line, so that it will be expanded on ypmaster, instead of locally on ypslave. Notice that the map files should be owned by root, so you must change ownership of them after the transfer.

ypserv Crashes

When the ypserv process crashes almost immediately, and does not stay up even with repeated activations, the debug process is virtually identical to that previously described in the subsection "ypbind Crashes.". Check for the existence of the portmap daemon as follows:

```
ypserver% ps ax | grep portmap
```

Reboot the server if you do not find the daemon. If it is there, type:

```
% rpcinfo -p yp_server
```

and look for output such as:

```
program vers proto   port
 100004    2   udp   1027  ypserv
 100004    2   tcp   1024  ypserv
 100004    1   udp   1027  ypserv
 100004    1   tcp   1024  ypserv
 100007    2   tcp   1025  ypbind
 100007    2   udp   1035  ypbind
 100007    1   tcp   1025  ypbind
 100007    1   udp   1035  ypbind
 100009    1   udp   1023  yppasswdd
 100003    2   udp   2049  nfs
 100024    1   udp   1074  status
 100024    1   tcp   1031  status
 100021    1   tcp   1032  nlockmgr
 100021    1   udp   1079  nlockmgr
 100020    1   udp   1082  llockmgr
 100020    1   tcp   1033  llockmgr
 100021    2   tcp   1034  nlockmgr
 100012    1   udp   1104  sprayd
 100011    1   udp   1106  rquotad
 100005    1   udp   1108  mountd
 100008    1   udp   1110  walld
 100002    1   udp   1112  rusersd
 100002    2   udp   1112  rusersd
 100001    1   udp   1115  rstatd
 100001    2   udp   1115  rstatd
 100001    3   udp   1115  rstatd
```

Your machine will have different port numbers. The four entries representing the `ypserv` process are:

```
 100004    2   udp   1027  ypserv
 100004    2   tcp   1024  ypserv
 100004    1   udp   1027  ypserv
 100004    1   tcp   1024  ypserv
```

If they are not present, `ypserv` has been unable to register its services. Reboot the machine. If the `ypserv` processes are there, and they change each time you try to restart `/usr/etc/ypserv`, reboot the machine. If the situation persists after reboot, call Sun for assistance.

14.7. Turning Off Yellow Pages Services

If `ypserv` on the master is disabled, you can no longer update any the YP maps. If you choose to turn off YP on a network currently running it, you can disable it by simply renaming the `/usr/etc/ypbind`file to `/usr/etc/ypbind.orig`. `suninstall` does this automatically if you tell it you do not want to run YP. Type the following:

```
% cd /etc
% mv /usr/etc/ypbind /usr/etc/ypbind.orig
```

To disable YP on a particular YP slave or master, type the following on the server in question:

```
% mv /usr/etc/ypserv /usr/etc/ypserv.orig.
```

Again, `suninstall` does this automatically if you do not select yp.

Refer back to Chapter 13, "The Sun Network File System," for information about the files you need to maintain for a server should you decide to disable YP.

14.8. The `publickey` Map

This file consists of three fields in the following format:

```
user name    user's public key : user's secret key
```

where *user name* may be the name of a user or of a machine, *user public key* is that key in hexadecimal notation, and *user secret key* is that key also in hexadecimal notation.

Since nobody expects you to be conversant in hexadecimal notation, the program `newkey` is provided to make things easier. Simply become superuser at the master server and invoke `newkey` for a given user:

```
# newkey -u username
```

or for the super user in a given host machine:

```
# newkey -h hostname
```

and at the prompt enter the appropriate login password. The program will then create a new public/secret key pair in `/etc/publickey`, encrypted with the login password of the given user.

Users can later on modify their own entries, or can even create them, by using the program `chkey`. The user simply types:

```
% chkey
```

and then responds to prompts from the command. A typical `chkey` session would look like this:

```
willow% chkey
Generating new key for username
Password: user enters password
Sending key change request to server...
Done.
willow%
```

Note that in order for `newkey` and `chkey` to be able to run properly, the daemon `ypupdated` must be running in the master server. If it is not running at this point, enter

```
# /usr/lib/netsvc/yp/ypupdated
```

and also make sure that the appropriate file in `/etc/rc?.d` contains the lines

```
if [ -f /usr/lib/netsvc/yp/ypupdated -a -d /var/yp/`domainname` ]
then
    /usr/lib/netsvc/yp/ypupdated
    (echo -n ' ypupdated')  >/dev/console
fi
```

The `ypupdated` daemon consults the file `/var/yp/updaters` for information about which maps should be updated and how to go about it. In the case of the `publickey` map, changes to `/etc/publickey` affected through `newkey` or `chkey` are mediated by `/usr/etc/yp/udpublickey`.

Finally, ensure that the master YP server contains the empty file `/etc/netid`. You do not have to do anything to this file, but it must exist for public key encryption to work.

Automounting with YP

You can use YP along with the automounter to provide a single `auto.master` file for an entire network. You simply write the automounter files as they would reside in a machine's `/etc` directory.

A typical `auto.master` file might contain

```
#Mount-point   Map                 Mount-options
/net           -hosts
/home          /etc/auto.home      -rw,intr,secure
/-             /etc/auto.direct    -ro,intr
```

A typical `auto.home` map might contain:

```
#key      mount-options   location
willow                    willow:/home/willow
cypress                   cypress:/home/cypress
poplar                    poplar:/home/poplar
pine                      pine:/export/pine
apple                     apple:/export/home
ivy                       ivy:/home/ivy
peach     -rw,nosuid      peach:/export/home
```

Here is a typical `/etc/auto.direct` map:

```
/usr/local \
                    /bin      -ro,soft    ivy:/export/local/sun3 \
                    /share    -ro,soft    ivy:/export/local/share \
                    /src      -ro,soft    ivy:/export/local/src
/usr/man                      -ro,soft    oak:/usr/man \
                                          rose:/usr/man \
                                          willow:/usr/man
/usr/games                    -ro,soft    peach:/usr/games
/usr/spool/news               -ro,soft    pine:/usr/spool/news
/usr/frame                    -ro,soft    redwood:/usr/frame1.3 \
                                          balsa:/export/frame
```

Refer back to the "Using the Automounter" section in Chapter 13, "The Sun Network File System," for full information about these files.

15

Addendum: Setting Up Electronic Mail

15

Addendum: Setting Up Electronic Mail

4.0.3 Inclusion Instructions

This section is a replacement for the first part of Chapter 15, "Electronic Mail and Communications." Replace all pages from the beginning of the chapter to the heading "Dial-up Networks."

SunOS electronic mail is handled by `sendmail`, a general internetwork mail routing service. `sendmail` features aliasing and forwarding, automatic routing to network gateways, and flexible configuration.

This section explains how to install the mail system on your computer. For a more detailed explanation, refer to Chapter 18, "Sendmail Installation and Operation."

The mail system consists of the following commands and files:

`/usr/ucb/mail`	UCB mail program, described in `mail`(1)
`/usr/bin/mailtool`	Window-based interface to
`/usr/lib/sendmail`	Mail routing program
`/usr/lib/sendmail.mx`	Mail routing program linked with the domain name service resolver.
`/etc/sendmail.cf`	Configuration file for mail routing
`/usr/lib/sendmail.main.cf`	Sample configuration file for main machines (see below)
`/usr/lib/sendmail.subsidiary.cf`	Sample configuration file for subsidiary machines (see below)
`/var/spool/mail`	Mail spooling directory for delivered mail
`/var/spool/mqueue`	Spool directory for mail going out over the network

`/var/spool/mqueue`	Spool directory for mail going out over the network
`/var/spool/secretmail`	Secure mail directory
`/usr/bin/xsend`	Secure mail sender
`/usr/bin/xget`	Secure mail receiver
`/usr/bin/enroll`	To receive secure mail messages
`/etc/aliases`	Mail forwarding information
`/usr/ucb/newaliases`	Symbolic link to `/usr/lib/sendmail`.
`/usr/ucb/biff`	Mail notification enabler
`/usr/etc/in.comsat`	Mail notification daemon
`/usr/etc/in.syslog`	Error message logger, used by `sendmail`

Users send mail by using the `mail` command or `mailtool`, a user interface fully described in *Mail and Messages: Beginner's Guide*, to edit the messages sent and received. Both pass these messages to `sendmail` for routing. (Refer to `mail`(1) and `sendmail`(8) for detailed information about these programs.)

`sendmail` processes each piece of mail, using information in the configuration file `/etc/sendmail.cf` to determine network name syntax, aliasing and forwarding information, and network topology. Local mail is delivered by giving it to the program `/bin/mail`, which adds it to the mailboxes in the directory `/var/spool/mail`, using a locking protocol to avoid problems with simultaneous updates. The file `/var/spool/mail/`*username* normally contains mail for each user name. After the mail is delivered, the local mail notification daemon `/usr/etc/in.comsat` notifies users who have the command `biff y` in their `.login` files, or who use `mailtool`, that mail has arrived.

Only you can read your mail file. However, anyone with the superuser password can read others' files, including their mail. To send mail that is secure against any possible perusal (except by a code-breaker), use the secret mail facility, which encrypts the mail so that no one can read it. The man page `xsend`(1) fully describes this facility. Note that `xsend` does not work over the network.

Domain Names and `sendmail`

By default, `sendmail` uses Internet standard domain names, first described in Chapter 12, "The SunOS Network Environment." These standards make it possible for any Internet system in the world to send or receive mail with any other Internet system. This is why each Internet system must have a unique name. As explained in Chapter 12, a domain is an administrative division and has nothing to do with the connectivity of the network. Typically, all machines in given domain are connected to each other, but this is only an administrative convenience.

As you have seen previously, Internet names are divided into domains and subdomains. There are only a small number of top-level domains, for example `.COM` and `.EDU`. In some countries, the name of the country is the top level

domain. For example, some systems in the United Kingdom use the top level domain `.UK`. In the United States, the top level domain `.US` is used for some personal systems.

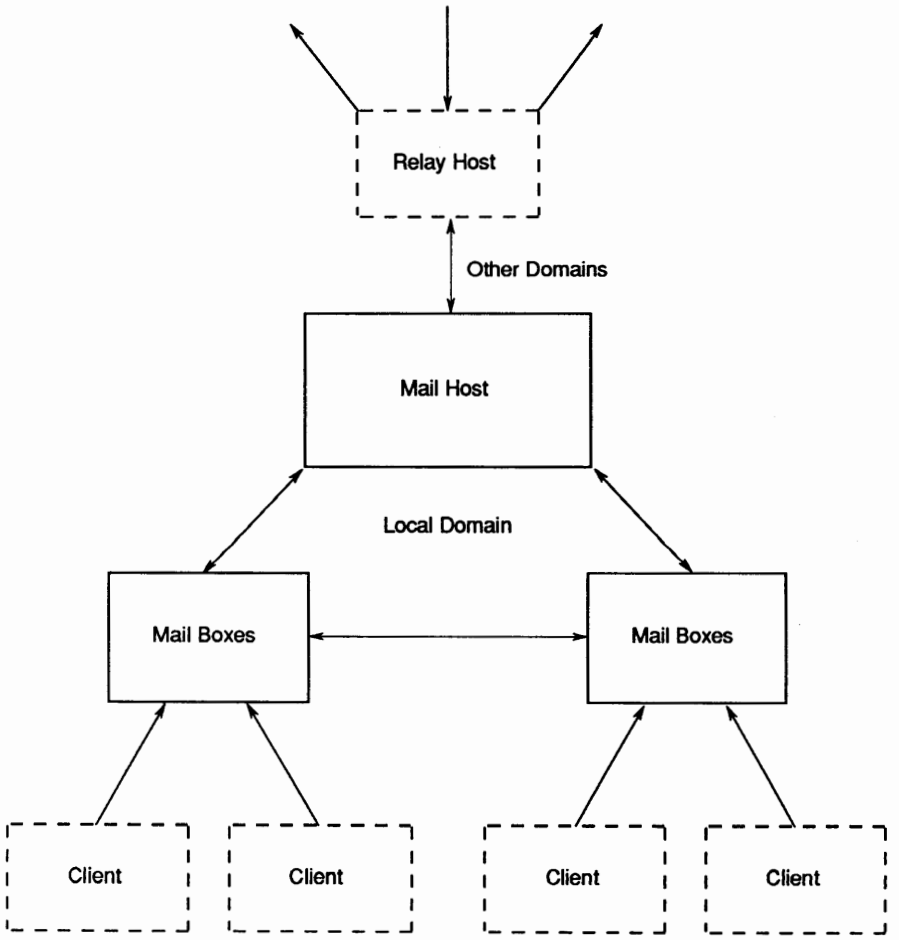
Domains nest inside one another like directories, except that the names go from right to left. Thus `joe.sun.COM` is the name of host joe in subdomain sun, which is in domain `.COM`, in the same way that `/etc/hosts.equiv` is file `hosts.equiv` in directory `/etc`.

If you have not done so already, you should register your domain with the Network Information Center (NIC) at SRI International, as described in Chapter 12.

15.1. Configuring Machines for Mail Operation

From a `sendmail` perspective, three types of machines exist in a network: at least one mailbox server, mail clients, and a mail host. Figure 15-1 illustrates their relationship to one another.

Figure 15-1 *A Typical Electronic Mail Configuration*



This section defines these machines and explains how to configure them.

Setting Up an NFS Mailbox Server and Its Clients

A *mailbox server* is any machine that actually stores mailboxes in the directory `/var/spool/mail`. In Release 4.0, client machines can mount their mailboxes through NFS from a mailbox server. This enables users to log in to any machine, including the server and read their mail. You can designate an NFS server as a mailbox server by having it export `/var/spool/mail`. However, other types of machines, even diskless clients, can operate as mailbox servers.

`/var/spool/mail` holds the individual mailboxes for users on the network. Each file called `/var/spool/mail/`*user_name* contains the individual mailbox for a particular user.

Any NFS server can be a mailbox server, including machines from companies other than Sun or machines running earlier releases of SunOS. To set up a machine as an NFS mailbox server, you need to edit `/etc/exports`, so that the server exports `/var/spool/mail`.

The mailbox server is responsible for sending outgoing mail from a client. When a client sends mail, the mailbox server puts it in a queue for delivery. Thereafter, the client can reboot or even power down, yet its mail will safely reach the recipient—baring other network problems, of course. When the recipient gets the client's mail, the path in the message's "From:" line contains the name of the mailbox server. If the recipient chooses to respond, the response goes to the user's mailbox in `/var/spool/mail` on the server, not directly to the client.

Converting Clients to Use Mailbox Servers

The following instructions explain how to set up the mailbox server's clients:

1. Make sure clients are halted so that no mail is lost during the conversion.
2. Log in as superuser on the mailbox server.
3. If any mailboxes exist on the client, move them to the mailbox server's mailbox directory. Type the following:

```
# mv /export/root/client_name/var/spool/mail/* /var/spool/mail
```

4. Edit `fstab` in each client's `/export/root/`*client_name*`/etc` directory, adding an entry of the form:

```
mailbox_server:/var/spool/mail  /var/spool/mail
```

   where *mailbox_server* is the server's name.

5. When you are finished editing their `/etc/fstab` files reboot each client to have the new `/etc/fstab` file take affect.

Once you have performed these tasks, mail operations can proceed. The mailbox server receives incoming mail from `sendmail`. The `/bin/mail` program running on the server puts the mail in the appropriate mailbox in `/var/spool/mail`. Each client mounts its mailbox through the `/var/spool/mail` entry in its `/etc/fstab` file.

Setting Up the Mailbox Server Alias

The next step in setting up mailbox servers is to assign an alias to the mailbox server. You do this by editing the `/etc/aliases` file. (The later section, "Setting Up the Postmaster Alias" fully describes `/etc/aliases`.)

Suppose machine "ballet" is the mailbox server for a particular organization or group. The existing `/etc/aliases` file on the network might resemble:

```
root: sysadmin@ballet
shamira: shamira@raks
benny: benny@samba
```

where "ballet" is the mailbox server name and "raks" and "samba" are client names.

To set up the mailbox server alias for server ballet, you would first log in to ballet. Then change the client names to the mailbox server names, so that `/etc/aliases` might look like:

```
root: sysadmin@ballet
shamira: shamira@ballet
benny: benny@ballet
```

Be very careful that all aliases in the file resolve to names on the mailbox server, not the clients. In other words, if you leave the entry for user shamira as

```
shamira: shamira@raks
```

and do not change it to

```
shamira: shamira@ballet
```

shamira's outgoing mail will take an extra trip over the network between raks and ballet.

Setting Up the Mailhost and Subsidiary Machines

The next step in mail configuration is to define which machine on your network is the the main mail machine, or *mailhost*, and which machines are mail subsidiaries. Subsidiary mail machines directly distribute mail to recipients in the same domain. However, when a mail subsidiary encounters mail destined for a machine in a different domain, the subsidiary sends it to the mailhost for delivery.

The mailhost must have the file `/usr/lib/sendmail.main.cf` installed as `/etc/sendmail.cf` in its root file system. Subsidiary mail machines have the file `/usr/lib/sendmail.subsidiary.cf` installed as `/etc/sendmail.cf` in their root file systems. The mailhost/subsidiary configuration simplifies administration by reducing the number of machines with custom mail configuration files. In most cases, you will find the default `/usr/lib/sendmail.main.cf` and `/usr/lib/sendmail.subsidiary.cf` appropriate for your network. However, you may want to make a few simple changes to customize the `sendmail.cf` files for your particular network. Should you want to tailor these configuration files for your network, refer to Chapter 18, "Sendmail Installation and Operation," for more information.

A good candidate for mailhost is a machine attached to an Ethernet and to phone lines, or a machine configured as a router to the Internet. Note that if you have a non-networked standalone in a time-sharing configuration, treat it like the mailhost in a one machine network. Similarly, if you have several machines on an Ethernet and none have phones, pick one as the mailhost and leave the others as subsidiaries. You might connect your network to other domains in the future, perhaps by using the Sunlink MHS product.

Configuring Subsidiary Machines

`suninstall` installs the default subsidiary configuration file, `/usr/lib/sendmail.subsidiary.cf` on each machine. Thus, you do not have to do anything more to configure a subsidiary machine, unless you want to use a configuration file other than the default. Refer to Chapter 18 for specific instructions. Then put the subsidiary configuration file in `/export/root/proto.local/etc/sendmail.cf`.

You can make simple changes to the subsidiary configuration file to fit the needs of your particular network. For example, you can change the way in which the network domain name appears in mail messages.

By default, the visible part of the network domain name (as described in Chapter 12) is used for both outgoing and incoming mail. If you want the domain name to appear differently for incoming and outgoing mail, you need to add the appropriate lines to `sendmail.cf`. In this file, lines beginning with the letters "Dm" set the domain name for outgoing mail; lines beginning with "Cm" set the domain name for incoming mail.

Note: The following instructions also apply when you are configuring the mail host

Suppose your network has the domain name `hq.dance.com`. Its displayed domain name on mail messages would be `dance.com`. To change the domain name on outgoing and incoming mail, do the following:

1. Log in as superuser on the subsidiary machine.
2. Edit `sendmail.cf`.
3. If you want to change the displayed domain name on outgoing mail, add a line beginning with `Dm`, in this case

   ```
   Dmdance.com
   ```

   Change the name following `Dm` to the name you want displayed.
4. To change the domain name for incoming mail, add a line beginning with `Cm`, in this case

   ```
   Cmdance.com
   ```

   Replace the name following `Cm` to the one you want accepted, for example:

   ```
   Cmdance.uucp
   ```

   You can state a list of acceptible domain name aliases after `Cm`. Thereafter, `sendmail` will recognize that incoming mail sent to any of these domain names should in fact be sent to the local domain.

On a subsidiary machine with phone lines, you can edit the `/etc/sendmail.cf` file so that `sendmail` routes mail received from `uucp` to certain hosts via the local phone lines. This is more efficient than having all `uucp` traffic go through the mailhost. (The second half of this chapter explains how to set up `uucp`.) Follow these procedures.

1. Log in as superuser on the subsidiary machine with a phone line.
2. Edit `sendmail.cf` and find the following line:

```
# local UUCP connections -- not forwarded to mailhost
CV
```

3. Put the names of the local `uucp` sites on the end of the CV line, or create additional CV lines.

   For example, you could do the following:

```
CV rome prussia georgia
```

Configuring the Mail Host

The first step in creating the mail host is to have the file `/usr/lib/sendmail.main.cf` become the mail configuration file for that machine. Follow this procedure:

1. Log in as superuser on the machine to become mail host.
2. Type the following:

```
# cp /usr/lib/sendmail.main.cf /etc/sendmail.cf
```

   Thereafter, `sendmail` will read `/etc/sendmail.cf` and recognize that this machine is the mail host.

3. Place the `mailhost` name for your new mail host in the `/etc/hosts` file on the master YP server, or on all hosts on a network without YP.

   For example, suppose you've selected the machine ballet as the mailhost. Its entry in `/etc/hosts` should look like:

```
129.255.99.99   ballet mailhost
```

You can optionally edit `/etc/sendmail.cf` on the mailhost to suit your particular network. Your network may have a `uucp` or Ethernet connection with a machine called a *relay host* that will relay mail to you. The relay host runs at least one mail-related protocol called a *mailer*. Each mailer specifies a policy and the mechanics to use when delivering mail. `sendmail` provides for several different kinds of mailers. The sample `sendmail.cf` file defines several available mailers, such as `smartuucp`, `ddn`, `ether`, and `uucp`. You can add others, as described in Chapter 18.

Once you have found a willing relay host, look for the following block of lines in `sendmail.cf`:

```
# major relay mailer
DMsmartuucp

# major relay host
DRddn-gateway
CRddn-gateway
```

Change the line `DMsmartuucp` to the name of the mailer that you connect to the mail host, for example, `uucp` or `ddn`, and the name following the letters `DR` to the name of the relay host.

For example, if your local network includes a machine called "cmu-cs-vlsi" that is on the Internet, you might use the following entry:

```
# major relay mailer
DMddn

#major relay host
DRcmu-cs-vlsi
CRcmu-cs-vlsi
```

On the other hand, your relay host might be `uucp` host ucbvax, which means you might use the following entry:

```
# major relay mailer
DMuucp

#major relay host
DRucbvax
CRucbvax
```

This change enables you to mail to an address such as "charlie@MIT.EDU." Even though your mail host may not be on the Internet, the message will arrive. If you are using Sunlink/MHS, refer to your installation manual for more information.

Setting Up the Postmaster Alias

Someone at your site, possibly yourself, should have the responsibility for handling problems with electronic mail. This person should have the alias *Postmaster*, a title which is recognized throughout the electronic mail community. For example, you can send mail to `postmaster` at other network sites if you have problems with mail originating at those sites.

The `/etc/aliases` File

The `/etc/aliases` file on a local machine contains all names by which a machine or person is known; `sendmail` reads it to determine mailing addresses. It is completely described in the man page `aliases`(5). You can use uppercase letters in names to the left of the colon in `/etc/aliases`. However, it is much safer to only use lowercase letters. `/etc/aliases` has

local aliases for individuals and groups, and special aliases. A local alias has the form

```
name: address[, address]
```

where *name* is the person or group's name and *address* is the address of a recipient in the group. A typical local alias might be:

```
benny:benny@samba
```

A special alias has the form:

```
owner-aliasname: address
```

`postmaster` is a special alias. Every local `/etc/aliases` file should have a postmaster entry; here is the default entry:

```
# Following alias is required by the mail protocol, RFC 822
# Set it to the address of a HUMAN who deals with this system's mail problems
postmaster: root
```

To establish the postmaster alias, change "`root`" to the user name of the person who will act as postmaster. Then messages directed to "postmaster" arrive with that person's other mail. If this postmaster also manages a domain with more than one host, add the postmaster alias to `/etc/aliases` on all hosts, or, for networks running YP, in the `mail.aliases` map on the YP master server.

If you manage the mail system for several domains, change `/etc/aliases` on all of them to forward postmaster mail to the machine where the postmaster usually reads mail. Suppose you are user amina, the network postmaster, and you use machine raks. You would use the following entry for postmaster in `/etc/aliases`:

```
postmaster: amina@raks
```

As postmaster, you may not want to have users' mail mixed in with your personal mail. Here are procedures that help you avoid this by redirecting postmaster mail into a separate file on your machine.

1. Log in as superuser on each mail client. (If your network runs YP, you only have to perform Steps 1 and 2 on the master YP server.)
2. Add an alias such as:

```
postmaster: sysadmin@raks
```

   to each mail client's `/etc/aliases`. This entry tells `sendmail` to direct mail to the postermaster alias to `sysadmin` on machine raks.
3. Log in as superuser on your own machine.

4. Add your own local mail alias to `/etc/aliases`. For example, postmaster amina would add the following entry to `/etc/aliases` on her own machine, raks

```
sysadmin:/home/amina/mailadm
```

This entry defines an alias for sysadmin: the file `/home/amina/mailadm`.

5. Exit superuser and log in with your own user name.

6. Type the following to create the file `mailadm`:

```
% touch /home/amina/mailadm
% chmode og+w
```

7. Type the following to have the file read:

```
% mail -f mailadm
```

You can also create aliases for people or groups of people called *mailing lists* when setting up the postmaster alias. The actual `/etc/aliases` file contains instructions for doing this.

Each time you edit `/etc/aliases`, you must run the `newaliases` program to rebuild the local alias database. If your network runs YP, run `make` in `/var/yp` on the master YP server after updating domain wide aliases.

Handling Undelivered Mail

By default, any time a message is returned as undeliverable by `sendmail`, a copy of the message header is sent to the postmaster. You can optionally disable this feature by editing the `sendmail.cf` file. Look for the following lines:

```
# CC my postmaster on error replies I generate
OPPostmaster
```

and change them as shown:

```
# CC my postmaster on error replies I generate
OPnobody
```

As shown above, you might want to create a separate file for undelivered mail, instead of mixing it with your personal mail.

Special Considerations for Networks with YP

Networks running YP have additional steps you must take and additional services you can add to `sendmail` service. Most importantly, after you have finished modifying `/etc/hosts` and `/etc/aliases` as described previously, remember to propagate the YP maps associated with these files. Type the following:

```
# cd /var/yp
# make
```

You can now use `mail.aliases` for domain-wide mail aliases. Thereafter, you should not have to remember hostnames when sending mail. `mail.aliases` will usually contain a copy of all the aliases known to a central mail machine.

Using Inverted YP Domain Names

The inversion of the YP domain-wide aliases can be used to simplify mail going outside the current domain. For example, consider the following `/etc/aliases` entries:

```
benny:benny@samba
gkelly:kelly@jazz
```

e: On networks with YP, you do not have to explicitly state a hostname. `sendmail` gets this information from the `mail.aliases` map.

When user kelly sends mail to user benny, the mail header displays the sender name, `kelly@jazz`. However, if kelly sends a message to a user called `placido@song.com`, the sender name on the message will be `gkelly@dance.com`. "`dance.com`" is the visible part of the domain name.

For incoming mail, `sendmail` normally replaces the left side of an alias entry with the right side. Thus the left side of the alias `gkelly` is replaced by the right side of the alias, `kelly@jazz`.

`sendmail` performs inverted alias processing on outgoing mail, replacing the right side of the alias with the left. This prevents the specific mailbox servers on a network from being visible outside the local domain. Therefore, if user kelly decides to switch to using host jazz as a mailbox server, user `placido@song.com` can still reply to `gkelly@dance.com`. The mail will automatically go to the right mailbox server.

Mail and the Internet Domain Name Server

In a domain running YP, the YP maps `hosts.byname` and `host.byaddr` resolve host names and addresses. You may wish to use the Internet domain name service for machines directly connected to the Internet. Consider doing this only if you can route between your machine and one of more of the "official" root servers. This will be the case if you are on the ARPANET, MILNET, or NSFNET.

Even if your are on one of these networks, you should still keep the `sendmail.cf` files on all clients and mailbox servers with standard configurations. You only need to customize the mail host `sendmail.cf` to take advantage of domain name service.

Install `/usr/lib/sendmail.mx` in place of `/usr/lib/sendmail` on the mail host to use domain name service. Each machine running `sendmail.mx` must have either `/etc/resolv.conf` or `/etc/named.boot` set up properly to allow name resolution or at least a caching server. Refer to Chapter 22 and the `resolv.conf`(5) man page for more information.

Testing Your Mail Configuration

First, reboot all systems whose configuration files you have changed. Then, send test messages from various machines on the network. To do so, go to a particular machine and type the following:

```
samba% /usr/lib/sendmail -v </dev/null names
```

This command sends a null message to the specified recipient name, and displays messages while it runs. Try the following tests:

- Send mail to yourself or other people on the local machine by addressing the message in the command above to a regular user name, such as `root`.
- If you are on an Ethernet, send mail to someone on another machine, as in `root@jazz`. Try this in three directions: from the main machine to a subsidiary machine, vice versa, and from a subsidiary machine to another subsidiary machine, if more than two are on the network.
- If you have a relay host, send some mail to another domain from the mailhost. This ensures that the relay mailer and host are selected properly.
- If you have set up a `uucp` connection on your phone line to another host, you can send mail to someone at that host and they can send mail back, or call you on the phone if they receive it.

  Try having them send mail to you. For example, you could send to ucbvax!azhar if you have a connection to ucbvax. `sendmail` will not be able to tell you whether the message really got through, since it hands the message to `uucp` for delivery. You have to ask the human at the other end if mail has been received. To get an idea of the message's progress, look in the file `/var/spool/uucp/LOGFILE`. For more information, refer to Chapter 21.
- Send a message to "postmaster" on various machines and make sure that it comes to your postmaster's mailbox, so that when other sites send you mail as postmaster, you will see it.

Diagnosing Problems with Mail Delivery

Here are some tools you can use for diagnosing mail problems.

The `/usr/lib/sendmail` command has a number of options for troubleshooting problems. For example, you can type the following:

```
% /usr/lib/sendmail -v -bv recipient
```

to verify the aliases and the deliverability of a given *recipient*. Here is an example of the resulting display from this command:

```
% /usr/lib/sendmail -v -bv shamira@raks
shamira... aliased to   mwong
mwong... aliased to                shamira@raks
shamira@raks... deliverable
```

`sendmail` also includes a test mode invoked by the `bt` option, as in

```
% /usr/lib/sendmail -bt
```

Here are instructions for using test mode.

1. Invoke test mode by typing the following:

```
% /usr/lib/sendmail -bt
ADDRESS TEST MODE
Enter <ruleset> <address>
```

2. Respond to the last prompt by typing a zero, then the mail address you want to test, for example, benny@samba.

```
> 0 benny@samba
rewrite: ruleset  3   input: "benny" "@" "samba"
rewrite: ruleset  6   input: "benny" "<" "@" "samba" ">"
        .
        .
        .
```

   The diagnostic information that `sendmail` displays is fully described in Chapter 18.

The `mconnect` program is another diagnostic tool that you can use to open connections to other `sendmail` systems over the network. `mconnect` runs interactively, so that you can issue it various diagnostic commands. For example, the `VRFY` command of `SMTP` (the protocol used to deliver mail over the network) performs the same operation as the `-bv` option to `sendmail`, and `VERB` invokes verbose mode like the `-v` option.

Other diagnostic tools include:

- `Received` lines in the header of the message.

  These trace which systems the message was relayed through. Note that in the `uucp` network, many sites do not update these lines, and that in the Internet, the lines often get rearranged. You can straighten them out by looking at the date and time in each line. Don't forget to account for different time zones.

- Messages from " MAILER-DAEMON."

  These typically report delivery problems.

- The system log, for delivery problems in your group of workstations.

  `sendmail` always records what it is doing in the system log, as explained in the next subsection. You might want to modify the `crontab` file to run a shell script nightly that searches the log for SYSERR messages and mails any that it finds to postmaster. In this way, problems are often fixed before anyone notices them, and the mail system runs more smoothly.

The System Log

The system log is supported by the `syslog` program, which is fully described in the man page `syslog`(8). Just as you define a machine called "mailhost" to handle mail relaying, you can define a machine called "loghost" in `/etc/hosts` to hold all the logs for an entire YP domain.

Format

Each line in the system log consists of a timestamp, the name of the machine that generated it (for logging from several machines over the Ethernet), and a message.

Levels

`syslog` can log a large amount of information. The log is arranged as succession of levels. At the lowest level, only unusual occurrences are logged. At the highest level, even the most mundane and uninteresting events are recorded for posterity. As a convention, log levels under ten are considered "useful;" log levels above ten are usually for debugging purposes.

Index

Writing Device Drivers

Sun Microsystems, Inc. • 2550 Garcia Avenue • Mountain View, CA 94043 • 415-960-1300

Part No: 800-1780-10
Revision A, of 24 April 1989

PART ONE: Regular Device Drivers

2

Hardware Context

Computer I/O architectures are far more dependent upon bus structure than they are upon CPU type, and device drivers, oriented as they are towards I/O, must have intimate knowledge of the bus characteristics of the machines on which they are running. For example, many Multibus machines do not support vectored interrupts [1] and thus drivers for interrupt driven devices which are intended to run on Multibus machines must provide polling facilities. Fortunately, the Sun kernel provides facilities (described in the *Other Kernel/Driver Interfaces* section of the *Overall Kernel Context* chapter) by which a driver can determine the type of the machine upon which it's running.

2.1. Multibus Machines

Multibus Memory Address Space and I/O Address Space

The MC680X0 family of processors does all its I/O via a process known as "memory mapping." What this means is that the processor sees no difference between memory and peripheral devices — all input-output operations are performed by storing data and fetching data from the same memory space. The Multibus, on the other hand, was originally designed for processors, like those of the Intel 8080 family, which have two separate address spaces. Such processors have one kind of instruction for storing data in memory or fetching data from memory (instructions such as MOV), and another, different kind of instruction (such as IN and OUT) for transferring data to or from peripheral devices. Reflecting the architecture of such processors, the Multibus has two address spaces.

Multibus memory space
: is used for memory or devices that look like memory. Many devices — commonly known as "memory mapped" devices — are designed to be accessed as memory, and drivers for such devices can "map" them into user virtual memory space and then perform device I/O by simply reading and writing the device's memory as part of normal address space. Such memory-mapped drivers tend to be quite simple, and so it's notable that devices not explicitly designed to be memory mapped can, under a restricted set of circumstances, be driven by memory mapping. The restrictions are,

[1] The Multibus itself, as it turns out, actually does support vectored interrupts, but not in a way that can reasonably be used with the MC680X0 family of processors.

however, fairly severe. Such drivers cannot, for example, have `xxioctl()` routines. See the *Mapping Devices Without Device Drivers* section of the *Driver Development Topics* manual for more details. The Sun-2 Color Board is a good example of a device that *is* designed to be memory mapped, and a listing of its driver can be found in the *Sample Driver Listings* appendix.

Multibus I/O address space
is another "space" entirely separate from normal memory. Typically used as an area to which device registers can be mapped, I/O space was originally introduced to keep such registers out of limited primary address space by providing a means of making peripherals, rather than system memory, respond to the bus whenever given I/O control lines were asserted by the CPU. (Such a setup also reduces hardware costs by keeping the number of address lines small.) Devices which have their control and status registers mapped to Multibus I/O address space are said to be "I/O mapped" devices.

The MC680X0 family, of course, no longer suffers the addressing limitations that made the dual-space architecture of the Multibus so attractive. The VMEbus, in similar regard, is no longer structured around separate "memory" and "I/O" spaces. (The term "I/O space" does continue to be used, from time to time, with reference to VMEbus-based systems and devices. Such use, however, is largely by way of analogy with Multibus systems, and it shouldn't be taken too literally).

Be aware that generic Multibus memory space can be either 20-bit or a 24-bit. (Sun normally uses 20-bit Multibus memory addresses, though when a Multibus card is installed in a VMEbus system with a VMEbus/Multibus adapter, 24-bit addresses are used). In similar regard, a generic Multibus can provide either an 8-bit or 16-bit I/O space, and Sun uses only the 16-bit Multibus I/O space. Note, however, that some older Multibus boards accept only 8-bit Multibus I/O addresses.

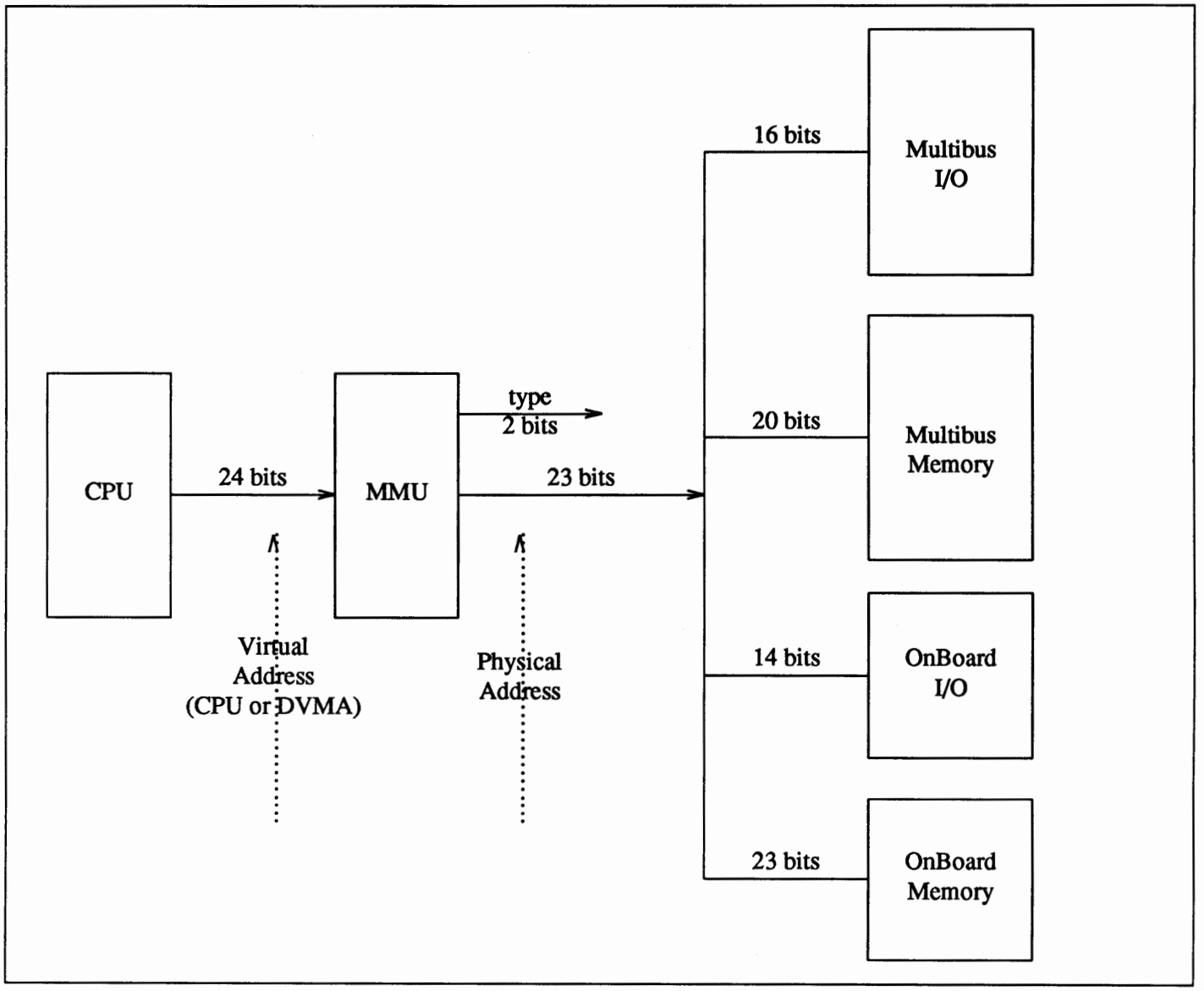
Sun Multibus systems actually have four "address spaces," corresponding to the four types of memory (each type has an identifying number associated with it, a number which is used by the MMU in computing PTE's (Page Table Entries). See the *Sun-2 Address Mapping* section of the *Driver Development Topics* chapter for details. Though you will seldom deal with the on-board address spaces, you're best off understanding what they are. The following table thus contains not only the two Multibus spaces, but the "on board" memory and I/O spaces as well. It's within these spaces, resident on the CPU board itself, that SunOS is run.

Table 2-1 *Sun-2 Multibus Memory Types*

Type	*Description*	*Address Size*	*Address Range*
0	On-Board Memory	23 bits	0x0 – 0x7FFFFF
1	On-Board I/O Space	14 bits	0x0 – 0x3FFF
2	Multibus Memory	20 bits	0x0 – 0xFFFFF
3	Multibus I/O Space	16 bits	0x0 – 0xFFFF

The following schematic view of the Sun-2 Multibus may help the driver developer to visualize the larger hardware context within which drivers operate (when running on a Sun-2 Multibus machine.)

Figure 2-1 *Sun-2 Multibus Address Spaces*



Note some significant aspects of addressing layout as indicated in this table.

- The Memory Management Unit is at the *center* of the picture, a position that reflects its importance in the addressing scheme of all Sun machines, VMEbus based as well as Multibus based. (The centrality of the MMU will become quite clear when you later set out to allocate a physical address to your device, and then examine/set it with the PROM monitor.)

- Secondly, the input address of the MMU is a 24-bit *virtual address*. It may originate with the CPU, or come from a DMA bus master; it makes no difference.
- The output is a 23-bit *physical address* and a 2-bit *address type*. The address type specifies one of the four address spaces indicated at the right of the diagram.
- The four address spaces are to the right. The space corresponding to the incoming virtual address is a function of both the address and the memory type. Note that only the top two memory spaces (Multibus I/O and Multibus Memory) are accessible by way of the Multibus; the two On-Board memory spaces are accessed directly and are seldom of concern to non-Sun driver developers.

Programs can only reference driver address spaces in terms of virtual addresses which are then translated by the MMU into physical addresses within the appropriate physical address space.

Allocation of Multibus Memory

Here are some notes about the allocation of Multibus Memory resources in the Sun system.

No devices may be assigned addresses below `0x40000` in Multibus memory space since the CPU uses these addresses for DVMA. (See the end of this chapter for a discussion of DVMA).

The table on the next page shows a map of how Multibus Memory space is laid out in the Sun system. Note that this memory map, as well as all of those that follow, is only a general guide. To be sure that you are not installing a device at a location that will put it in conflict with existing devices, it's necessary to check the configuration of the specific systems into which it will be installed. The best way to do so is to check the local config file for the physical addresses of the devices installed within the bus of interest. This will probably give you enough information, but if you still think that there may be a conflict, and if you have a Sun source license, you can check the driver header files to determine the amount of space consumed on the bus by existing devices. With the exception of the Sky board, these devices can be rearranged. Also note the possibility that your machine will have devices attached to it, and taking up bus space, even though those devices do not appear in the config file. This possibility exists because the *xx*`mmap()` system call can sometimes be used to drive a device without installing it in the formal sense — see the *Mapping Devices Without Device Drivers* section of the *Driver Development Topics* chapter for more details.

Table 2-2 *Sun-2 Multibus Memory Map*

Address	*Device*
0x00000 – 0x3FFFF	DVMA Space (256 Kilobytes)
0x40000 – 0x7FFFF	Sun Ethernet Memory (#1) (256 Kilobytes)
0x80000 – 0x83800	SCSI (#1) (16 Kilobytes)
0x84000 – 0x87800	SCSI (#2) (16 Kilobytes)
0x88000 – 0x8B800	Sun Ethernet Control Info (#1) (16 Kilobytes)
0x8C000 – 0x8F800	Sun Ethernet Control Info (#2) (16 Kilobytes)
0x90000 – 0x9F800	*** **FREE** *** (64 Kilobytes)
0xA0000 – 0xAF800	Sun Ethernet Memory (#2) (64 Kilobytes)
0xB0000 – 0xBF800	*** **FREE** *** (64 Kilobytes)
0xC0000 – 0xDF800	Sun Model 100/150 Frame Buffer (128 Kilobytes)
0xE0000 – 0xE1800	3COM Ethernet (#1)
0xE2000 – 0xE3800	3COM Ethernet (#2)
0xE4000 – 0xE7C00	*** **FREE** *** (16 Kilobytes)
0xE8000 – 0xF7800	Reserved for Color Devices (64 Kilobytes)
0xF8000 – 0xFF800	*** **FREE** *** (16 Kilobytes)

Allocation of Multibus I/O Space

Multibus I/O address space is specified in the config file as `mbio`. From the PROM monitor, Multibus I/O space begins at `0xEB0000`, and extends to `0xEC0000`.

Prior to Sun Release 3.0, the system made the assumption that any address lower than `0x10000` that it found in its config file was a Multibus I/O address. With current releases this is no longer true; now the bus type of every address must be explicitly given.

The following table of generic Multibus I/O usage, like the table above, is intended only as a guide.

Table 2-3 *Sun-2 Multibus I/O Map*

Address	*Device Type*
0x0040 – 0x0047	Interphase Disk Controllers
0x00A0 – 0x00A3	CPC TapeMaster Controllers
0x0200 – 0x020F	Archive Tape Drives
0x0400 – 0x047F	Ikon 10071-5 Multibus/Versatec Interface
0x0480 – 0x057F	Systech VPC-2200 Versatec/Centronics Interfaces
0x0620 – 0x069F	Systech MTI-800/1600 terminal Interface
0x2000 – 0x200F	Sky Board
0xEE40 – 0xEE4F	Xylogics 450/451 Disk Controller
0xEE60 – 0xEE6F	Xylogics 472 Multibus Tape Controller

2.2. VMEbus Machines

VMEbus machine architecture is generally more complex than Multibus machine architecture — it makes no distinction between I/O space and Memory space, but on the other hand it supports multiple address spaces. It does so for reasons of both cost and flexibility. The VMEbus was designed to be cost-effective for a range of applications. It is expensive (in terms of money, power, and board space) to provide the hardware for a full 32-bit address space. If installed devices only respond to 16-bit addresses, it makes sense to be able to put them all into a 16-bit address space and save the cost of 16-bits' worth of address decoders and the like. The 24 and 32-bit address spaces are similar compromises between cost and flexibility.

The driver writer has to understand which address space his board uses (generally, this is completely out of his/her control), and make an appropriate entry in the config file. For DMA devices, the driver writer has to know the address space that the board uses for its DMA transfers (this is usually a 32 or 24-bit space).

Sun-2 VMEbus Address Spaces

The Sun-2 VMEbus machines are based upon the 24-bit subset of the generic VMEbus — they support only a 16-bit and a 24-bit address space. These address spaces are known as `vme16d16` (16 address bits and 16 data bits respectively) and `vme24d16` (24 address bits and 16 data bits). Sun-2 VMEbus machines also contain on-board memory and I/O space, of course, but these aren't accessed by way of the VMEbus and are only barely relevant to the driver developer.

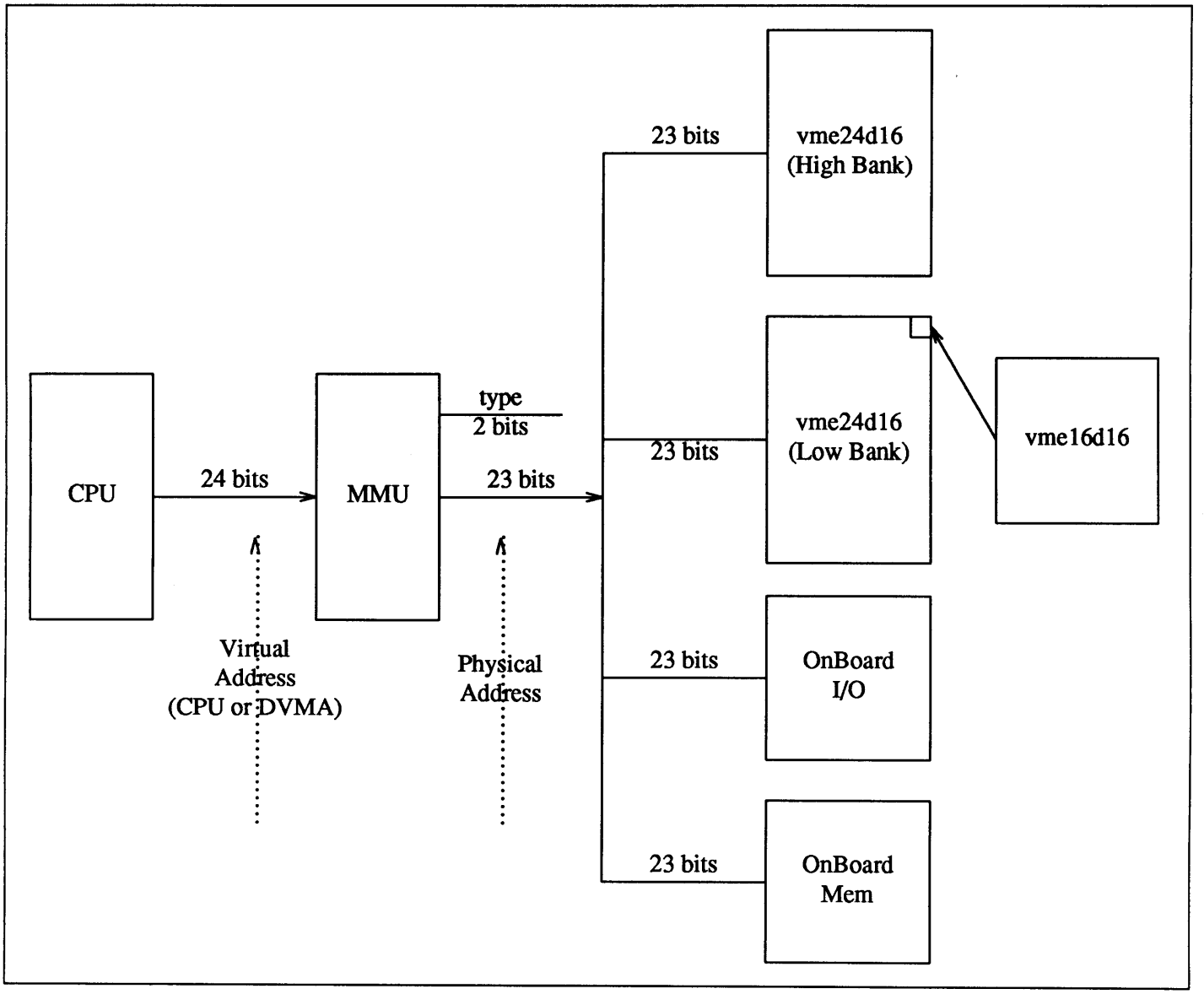
There are four types of memory on Sun-2 VMEbus machines:

Table 2-4 *Sun-2 VMEbus Memory Types*

Description	*Address Size*	*Address Range*
On-Board Memory	23 bits	0x0 - 0x7FFFFF
On-Board I/O Space	23 bits	0x0 - 0x7FFFFF
`vme24d16`	23+1 bits	0x0 - 0xFEFFFF
`vme16d16` — Stolen from top 64K of `vme24d16` (0x0 - 0xFFFF)		

The four address spaces are laid out as follows:

Figure 2-2 *Sun-2 VMEbus Address Spaces*



Note a few details:

- In all Sun-2 machines (as in Sun-3's, Sun-3x's and Sun-4's), the address input into the MMU is a virtual address, and may originate with either the CPU or a DVMA (Direct Virtual Memory Access) bus master. (See the *Sun Main-Bus DVMA* section, later in this chapter, for a discussion of DVMA).
- Unlike Sun-2 Multibus systems, in which each memory type maps cleanly to one address space, `vme24d16` maps to two different memory banks. Addresses from `0x0` to `0x7FFFFF` are "type 2" memory, while those from `0x800000` and up are "type 3". This is because Sun-2 VMEbus machines have only 23 output address bits, and this trick is necessary to generate the full range of a 24-bit address space. (See *Sun-2 Address Mapping* in the

Driver Development Topics chapter for more details).

- Multibus boards, connected to VMEbus to Multibus adapters, can be plugged into physical memory anywhere within `vme24d16` (which means that they can also be in `vme16d16`).
- The 24 bits in the `vme24d16` address space are referred to in the above table as 23+1 bits. This is because, as should be clear in the diagram above, the Sun-2 MMU outputs only the lower 23 bits of the address, and the 24th bit is actually one of the MMU's type bits.
- Note especially that `vme16d16` *is stolen from* `vme24d16`. It's selected by addresses in the form `0xFFXXXX`, that is, addresses which have the 8 high bits set.

Sun-3/Sun-3x/Sun-4 Address Spaces

Sun-3, Sun-3x and Sun-4 machines are all based on the full 32-bit VMEbus, so let's begin their discussion with a listing of the address types supported by the generic VMEbus. In all these memory references, we are referring to virtual VMEbus addresses, not Sun physical memory locations.

Table 2-5 *Generic VMEbus (Full Set)*

VMEbus-Space Name	*Address Size*	*Data Transfer Size*	*Physical Address Range*
`vme32d16`	32 bits	16 bits	`0x0 – 0xFFFFFFFF`
`vme24d16`	24 bits	16 bits	`0x0 – 0xFFFFFF`
`vme16d16`	16 bits	16 bits	`0x0 – 0xFFFF`
`vme32d32`	32 bits	32 bits	`0x0 – 0xFFFFFFFF`
`vme24d32`	24 bits	32 bits	`0x0 – 0xFFFFFF`
`vme16d32`	16 bits	32 bits	`0x0 – 0xFFFF`

Not all of these spaces are commonly used, but they are all nevertheless supported by the Sun-3 and Sun-4 lines. The following table indicates their sizes and physical address mappings.

Table 2-6 *Sun-3/Sun-4 VMEbus Address types*

Type	*Address-Space Name*	*Address Size*	*Address Range*
0	On-board Memory	32 bits	`0x0 – 0xFFFFFFFF`
1	On-board I/O	24 bits	`0x0 – 0xFFFFFF`
2	`vme32d16`	32 bits	`0x0 – 0xFEFFFFFF`
3	`vme32d32`	32 bits	`0x0 – 0xFEFFFFFF`
2	`vme24d16` — Stolen from top 16M of `vme32d16` (0x0 - 0xFEFFFF)		
2	`vme16d16` — Stolen from top 64K of `vme24d16` (0x0 - 0xFFFF)		
3	`vme24d32` — Stolen from top 16M of `vme32d32` (0x0 - 0xFEFFFF)		
3	`vme16d32` — Stolen from top 64K of `vme24d32` (0x0 - 0xFFFF)		

The Sun-3x is different than the Sun-3 and Sun-4 in that the hardware does not use page table entries (PTE's) with a type identifier to map the devices into physical memory. The Sun-3x uses absolute physical address when mapping devices.

Therefore the type field is not used as an identifier of physical address mapping. The next two tables show the virtual VME addresses and the corresponding physical addresses for the specific ranges. Note for the Sun-3x there is no `vme32d16` entry and there is a hole in the address space usage from the end of the on-board I/O area to the beginning of the vme16d16 area.

Table 2-7 *Sun-3x VMEbus Address types*

Type	*Address-Space Name*	*Address Size*	*Address Range*
0	On-board Memory	32 bits	`0x0 – 0xFFFFFFFF`
1	On-board I/O	24 bits	`0x0 – 0x00FFFFFF`
2	`vme24d16`	32 bits	`0x0 – 0xFEFFFFFF`
3	`vme32d32`	32 bits	`0x0 – 0xFEFFFFFF`
2	`vme16d16` — Stolen from top 64K of `vme24d16` (0x0 - 0xFFFF)		
3	`vme24d32` — Stolen from top 16M of `vme32d32` (0x0 - 0xFEFFFF)		
3	`vme16d32` — Stolen from top 64K of `vme24d32` (0x0 - 0xFFFF)		

Table 2-8 *Sun-3x Physical Address map*

Type†	*Address-Space Name*	*Address Size*	*Address Range*
	On-board Memory	32 bits	`0x00000000 – 0x57FFFFFF`
	On-board I/O	32 bits	`0x58000000 – 0x6EFFFFFF`
	`vme16d16`	32 bits	`0x7C000000 – 0x7C00FFFF`
	`vme16d32`	32 bits	`0x7D000000 – 0x7D00FFFF`
	`vme24d16`	32 bits	`0x7E000000 – 0x7EFFFFFF`
	`vme24d32`	32 bits	`0x7F000000 – 0x7FFFFFFF`
	`vme32d32`	32 bits	`0x80000000 – 0xFFFFFFFF`

†*Types are not used with the Sun-3x architecture.*

Sun-3/Sun-3x/Sun-4 space overlays are much more complex than those of the Sun-2, as is evident from both the tables above and the diagrams below. The principle, however, is the same — when a space overlays a larger space, its memory is stolen from that larger space and is considered by the MMU to be in the the overlaid space. One simply cannot address above `0xFF000000` in 32-bit VMEbus space or above `0x00FF0000` in 24-bit VMEbus space.

As the following diagrams illustrate, Sun-3 and Sun-4 addressing schemes are almost identical. They differ only in the size of the virtual address which — output by the CPU or a DVMA Bus Master — is fed to the MMU.

The Sun-3x, which has the MMU on the CPU chip, is a different hardware architecture than the Sun-3's and Sun-4's. There is a full 32 bit input to the MMU from the CPU, and all 32 bits are used for input to the OnBoard and vme modules. No devices use the `vme32d16` so it is not part of the memory map.

Figure 2-3 *Sun-3 VMEbus Address Spaces*

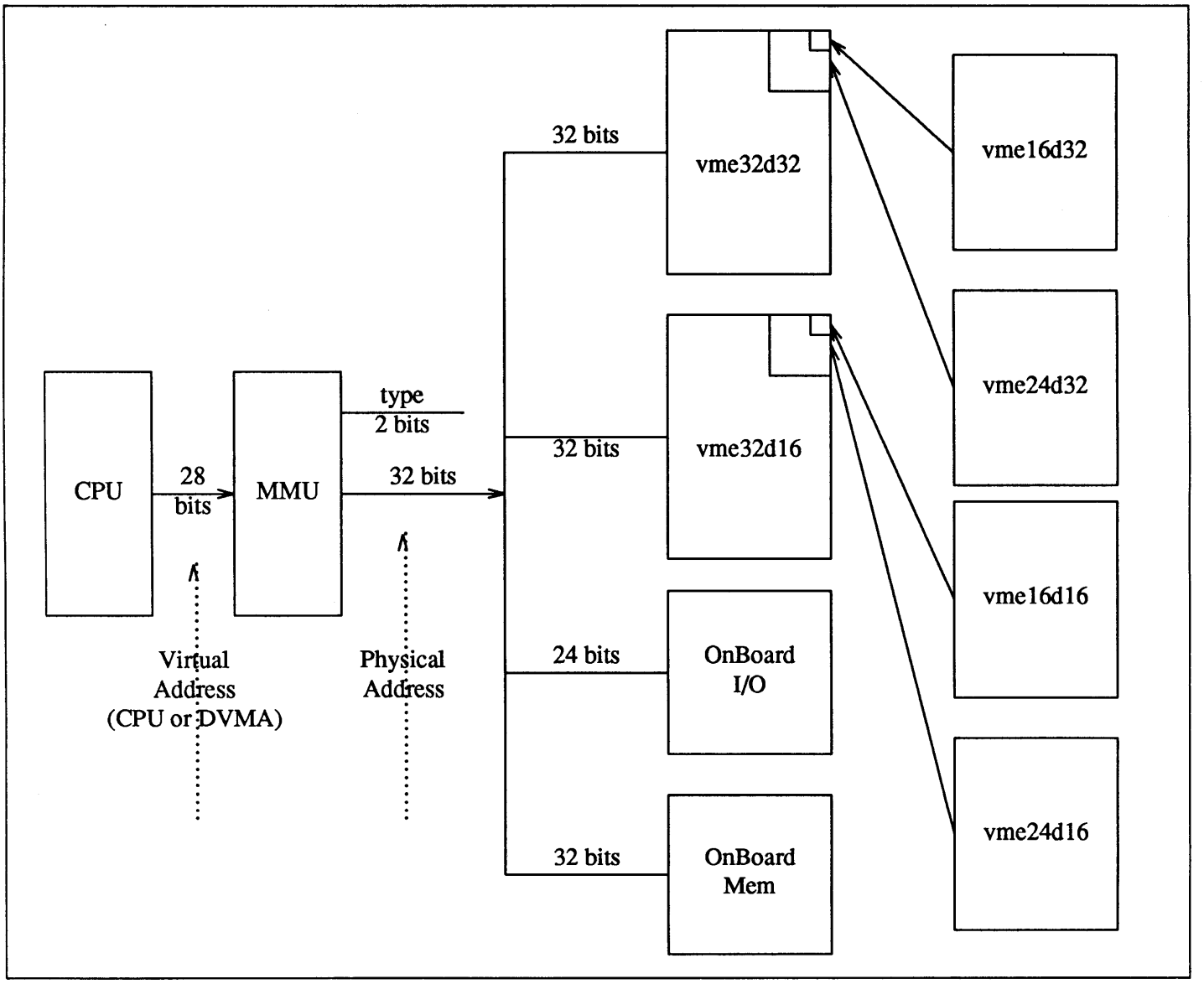


Figure 2-4 *Sun-3x VMEbus Address Spaces*

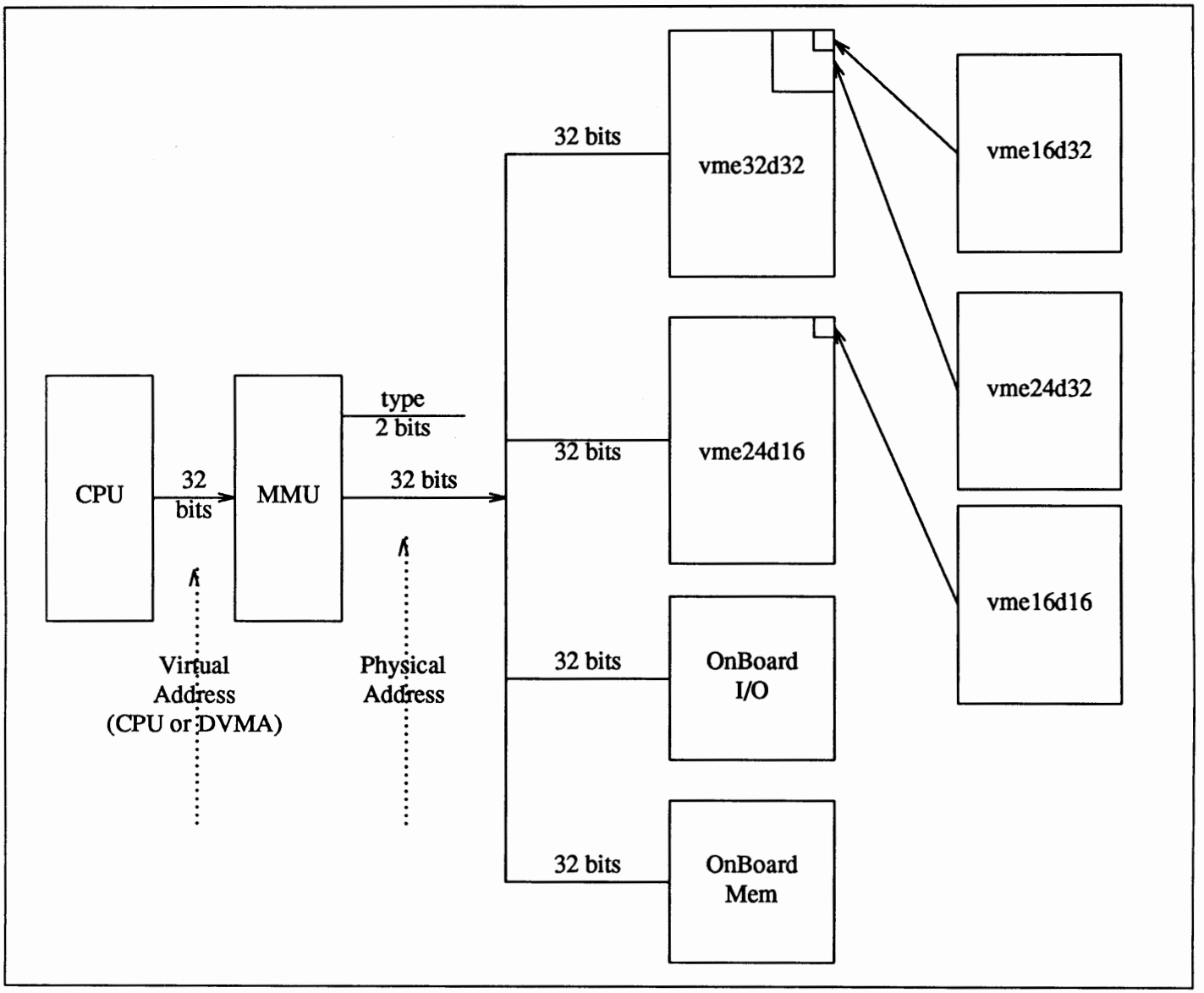
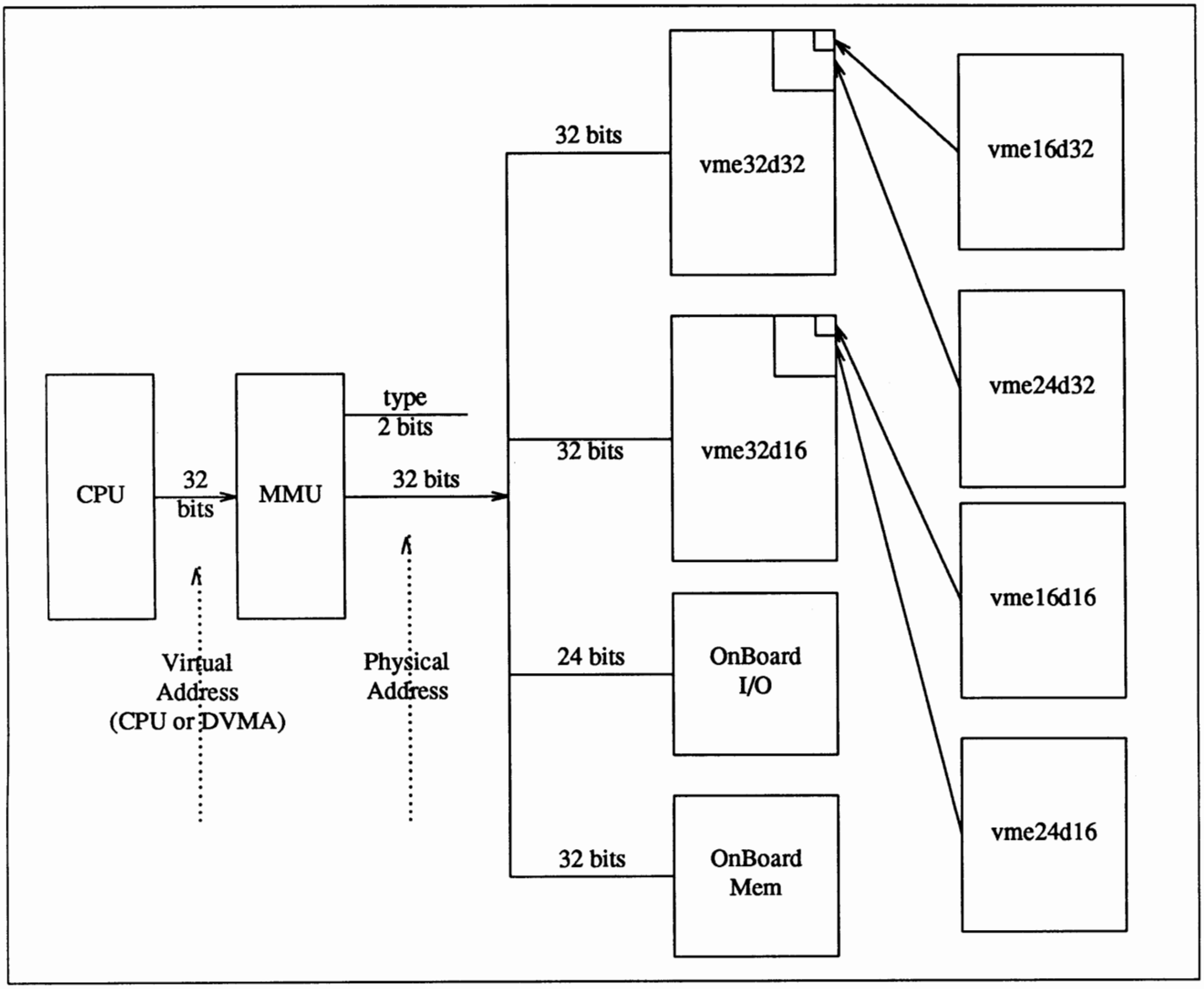


Figure 2-5 *Sun-4 VMEbus Address Spaces*



Allocation of VMEbus Memory

This section summarizes the typical use of the 16, 24 and 32-bit VMEbus address spaces by Sun devices. Note that the usages summarized here are only for the generic configuration, and there's no guarantee that they match the exact usage on your machine. They will, however, help you to decide where to attach your device. The "Allocated From" field shows whether bus space is allocated from the high end of the given range or from the low end. The idea is to keep the maximum size "hole" in the middle in case the boundary needs to be shifted later.

Table 2-9 *16-bit VMEbus Address Space Allocation*

Address Range	*Allocated From*	*Description of Use*
0x0000-0x7FFF	Low	Reserved for OEM/user devices
0x8000-0xFFFF	High	Reserved for Sun devices

16-bit VMEbus space is mapped into the topmost 64K of 24-bit VMEbus space at 0x00FF0000 to 0x00FFFFFF (on Sun-2s) or 0xFFFF0000 to 0xFFFFFFFF (on Sun-3's, Sun-3x's, and Sun-4's). Note: The Multibus/VMEbus Adapter will map the Multibus I/O addresses of Multibus cards that use Multibus I/O into the same addresses in the 16-bit VMEbus space. This may place the standard Multibus addresses for some cards into the OEM/user area in the above table. These addresses can be changed, if necessary, by physically readdressing the device and then changing its entry in the config file.

Table 2-10 *24-bit VMEbus Address Space Allocation*

Address Range	*Allocated From*	*Description of Use*
0x000000-0x0FFFFF		CPU board DVMA space
0x100000-0x1FFFFF		Reserved by Sun
0x200000-0x2FFFFF	Low	Reserved for small Sun devices
0x300000-0x3FFFFF	High	Reserved for large Sun devices
0x400000-0x7FFFFF	(Taken)	Reserved for huge Sun devices
0x800000-0xBFFFFF	High	Reserved for huge OEM/user devices
0xC00000-0xCFFFFF	Low	Reserved for large OEM/user devices
0xD00000-0xDFFFFF	High	Reserved for small OEM/user devices
0xE00000-0xEFFFFF		Multibus-to-VMEbus memory space
0xF00000-0xFEFFFF		Reserved for the Future
0xFF0000-0xFFFFFF		Stolen by 16-bit VMEbus space

Table 2-11 *32-bit VMEbus Address Space Allocation (Sun-3's, Sun-3x's, and Sun-4's)*

Address Range	*Description of Use*
0x00000000 - 0x000FFFFF	DVMA Space
0x00100000 - 0x7FFFFFFF	Reserved by Sun
0x80000000 - 0xFEFFFFFF	Reserved for OEM/user devices
0xFF000000 - 0xFFFFFFFF	Stolen by vme24d16

These same assignments apply to both 16-bit-data and 32-bit-data VMEbus accesses. Note that, at least in the GENERIC kernel, there are some Sun devices (`tm0, tm1, vpc0, vpc1 and mti0-4`) installed in the OEM/user area. It's always best to check, when choosing an installation address, that you aren't going to conflict with an already installed device.

Table 2-12 *VMEbus Address Assignments for Some Devices*

Device	*Addressing*	*Addresses Used*
VMEbus SKY Board	`vme16d16`	0x8000 - 0x8FFF (Sun-2 only)
VMEbus SCSI Board	`vme24d16`	0x200000 - 0x2007FF
VMEbus TOD Chip	`vme24d16`	0x200800 - 0x2008FF (Sun-2 only)
Graphics Processor	`vme24d16`	0x210000 - 0x210FFF
Sun-2 Color Board	`vme24d16`	0x400000 - 0x4FF7FF

The VMEbus Sky board occupies addresses `8000-8FFF` in 16-bit address space, and it requires that the high nibble of the address be '8'. Unlike other pre-installed devices, it cannot be moved.

This table is, of course, not complete. There is always a variety of devices on the bus, as can be easily determined by examining the config file. This table, however, does include the standard devices that use a significant amount of space on the VMEbus. Note that, in machines which came after the Sun-2 line, several of these devices have been replaced by on-board devices and have thus disappeared from the VMEbus address space.

The Sun VMEbus to Multibus Adapter

Multibus devices that are to be attached to VMEbus machines must be attached to a VMEbus to Multibus adapter. (The Adapter works for most, but not all, Multibus boards). An adapter can be used to take over *one and only one* chunk of `vme24d16`. However, that chunk can overlap all or part of `vme16d16` (because `vme16d16` is a proper subset of `vme24d16`). In any case, the adapter must be told how much space the board attached to it actually expects, for by default it will take over a full megabyte. Note that the Multibus Adapter supports fully vectored interrupts, and that drivers for Multibus devices attached by way of adapters need not poll, since the adapters contain switches by which Multibus devices can be assigned vectors.

Interrupt Vector Assignments

The table below shows the assignments of interrupt vectors for those devices that can supply interrupts through the VMEbus vectored interrupt interface. To pick one for your device, examine the kernel config file for an unused number in the range reserved for customer use, `0xC8` to `0xFF`.

Table 2-13 *Vectored Interrupt Assignments*

Vector Numbers			*Description*
0x40	*thru*	0x43	sc0, sc? si0, si? — SCSI Host Adapters
0x48	*thru*	0x4B	xyc0, xyc1, xyc? — Xylogics Disk Controllers
0x4C	*thru*	0x5F	future disk controllers
0x60	*thru*	0x63	tm0, tm1, tm? — TapeMaster Tape Controllers
0x64	*thru*	0x67	xtc0, xtc1, xtc? — Xylogics Tape Controllers
0x68	*thru*	0c6F	future tape controllers
0x70	*thru*	0x73	ec? — 3COM Ethernet Controller
0x74	*thru*	0x77	ie0, ie1, ie? — Sun Ethernet Controller
0x78	*thru*	0x7F	future ethernet devices
0x80	*thru*	0x83	vpc? — Systech VPC-2200
0x84	*thru*	0x87	vp? — Ikon Versatec Parallel Interface
0x88	*thru*	0x8B	mti0, mti? — Systech Serial Multiplexors
0x8C	*thru*	0x8F	dcp1, dcp? — SunLink Comm. Processor
0x90	*thru*	0x9F	zs0, zs1 — Sun-3/Sun-3x Terminal/Modem Controller
0xA0	*thru*	0xA3	future serial devices
0xA4	*thru*	0xA7	pc0, pc1, pc2, pc3 — SunIPC
0xA8	*thru*	0xAB	future frame buffer devices
0xAC	*thru*	0xAF	future graphics processors
0xB0	*thru*	0xB3	sky0, ? — SKY Floating Point Board
0xB4	*thru*	0xB7	SunLink Channel Attach
0xB8	*thru*	0xC7	Reserved for Sun Use
0xC8	*thru*	0xFF	Reserved for Customer Use

2.3. ATbus Machines

The Intel 80386 processor handles I/O devices placed in either memory space or in I/O space. On the 80386, memory-mapped I/O provides additional programming flexibility. Any memory instruction can access any I/O port located in the memory space. For example, the MOV instruction transfers data between any register and any port. The AND, OR, and TEST instructions manipulate bits in the internal registers of a device.

On some devices, reading a register will not read back what was written. Therefore, instructions such as AND, OR, and TEST can, in some cases, produce unexpected results because the instruction reads a good location, changes it, and writes it back. See the *Other Device Peculiarities* section, ahead.

Memory-mapped I/O can use the full complement of instructions. The 16 MB memory of AT memory exists in the 4 GB physical address space of the Sun386i at `0xE000 0000`. For example, a device that, on an AT, shows up in memory at `D0 0000` will show up in the Sun386i physical memory at `0xE0D0 0000`. Virtual addresses are assigned during the autoconfiguration process.

If an I/O device is mapped into the I/O space then the IN, OUT, INS, and OUTS instructions are used to communicate to and from the device. All I/O transfers

are performed via the AL (8-bit), AX (16-bit), or EAX (32-bit) registers. The first 256 bytes of the I/O space are directly addressable. The entire 64 Kbyte I/O space is indirectly addressable through the DX register.

The Sun386i has 21 interrupt channels, but only 11 are available to devices on the AT bus. The following list of interrupt channel assignments shows all of the interrupt channels.

Table 2-14 *Interrupt Channel Assignments*

*AT Channel**	*Assignee*
3	AT Pin B25
4	AT Pin B24
5	AT Pin B23
6	Not available (system diskette)
7	Not available (parallel port)
8	SCSI
9	AT Pin B04
10	AT Pin D03
11	AT Pin D04
12	AT Pin D05
13	Not available (Ethernet)
14	AT Pin D07
15	AT Pin D06

* *Available to AT Cards*

When you add an AT card to the AT bus, you must select one of the values in the Channel column for the AT card's jumpers. For example, if you select channel 10 for a serial card, the "device" line in the config file might look as follows:

```
device  ns0 at atio ? csr 0x3f8 irq 10 priority 6
```

The Sun386i does not permit two AT cards to use the same interrupt channel.

Some cards will also use DMA and will have jumpers to select a DMA channel to use. The following list shows that DMA channels 0-3 and channel 5 are available for AT cards. Note that channel 0 and 5 can be used with 16-bit DMA devices; 1, 2, and 3 can be used only with 8-bit DMA devices. Note also that channels 4, 6, and 7 are pre-assigned.

Table 2-15 *Sun386i DMA Channel Assignments*

Channel	*Assignee*	*Size (bits)*
0	AT Bus	16
1	AT Bus	8
2	AT Bus	8
3	AT Bus	8
4	Software	Not Available
5	AT Bus	16
6	Ethernet	16
7	SCSI	16

For example, you might set up a controller that uses DMA channel 3. For this, the "controller" line in the config file might look like: this:

```
controller wds0 at atio ? csr 0x320 dmachan 3 irq 3 priority 3
```

The Sun386i does not permit two AT cards to use the same DMA channel.

In these examples, "priority" refers to the `spl` levels used in the driver. That is, the phrase "priority 3" implies that the driver uses `spl3()` to protect its critical regions.

Loadable Drivers

On Sun386i machines, device drivers can be dynamically loadable. That is, they can be attached to a system without rebuilding its kernel and without having to bring the system down and restart it. See the *Adding and Removing Loadable Drivers* section of the *Configuring the Kernel* chapter for details.

DOS and SunOS Environments

The Sun386i system supports both DOS drivers and SunOS drivers.

You can attach a DOS device driver in the standard DOS way, but it will be usable only from within the DOS environment. Usually, all you need to do is to first plug in an add-in board. Then you insert an installation diskette (which comes with the board) into Drive A> and re-boot the system. The device driver is already compiled and linked. Generally, the diskette contains programs called "INSTALL" or something similar. You execute this program by typing its name. It copies the driver file from the diskette to the hard disk. At the same time, this procedure will modify the disk's `config.sys` file.

The DOS system must be re-booted. The device driver will automatically be loaded into memory, its options will be parsed, and the driver will be initialized.

NOTE *The DOS driver on the Sun386i is running under SunOS and DOS, but the driver is unaware of this. SunOS might switch control to another task during device operation, so strict timing dependencies could fail. Real time devices, for example, may not work properly. If a peripheral and controller have strict timing requirements, their drivers should be written in the standard SunOS style. DOS drivers do not run at the elevated priority of SunOS drivers.*

SunOS drivers, of course, are parts of the system kernel. Thus the timing requirements of most devices can be met under SunOS. SunOS drivers are accessible from the DOS environment.

2.4. Hardware Peculiarities to Watch Out For

There is a variety of device peculiarities that the driver developer must be aware of. The most common of them are related to the Multibus and Multibus-based devices, but there are others as well.

Multibus Device Peculiarities

The IEEE Multibus is a source of problems for two separate reasons. The first of these, discussed immediately below, is the fact that the Multibus has a different notion of byte order than does the either Motorola MC680X0 family or the Sun SPARC processor (the reduced instruction set CPU upon which Sun-4 machines are built). The second is simply that the Multibus has been around for a long time, and thus brings with it a variety of older devices, many of which have addressing limitations and other characteristics which make for a less than perfect fit with the Sun architecture.

Multibus Byte-Ordering Issues

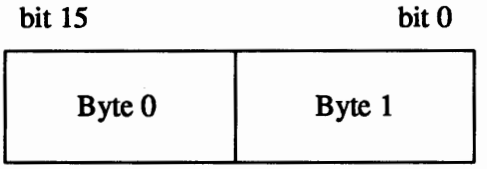
The Sun-2, Sun-3, and Sun-3x processors are members of the Motorola MC680X0 family, while Sun-4 processors are based on the SPARC CPU. All of these processors address bytes within words by what we shall call *IBM conventions* — the most significant byte of a word is stored at the lowest addressed byte of the word. The Multibus, on the other hand, uses *DEC conventions* — the least significant byte of a word is stored at the lowest address, and significance increases with address.

This class of byte-addressing conventions leads to two separate problems, with two separate solutions:

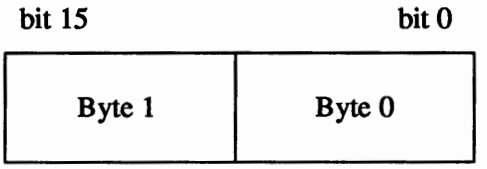
- The first problem occurs when you're moving a single *byte* across the interface between the MC680X0/SPARC and the IEEE Multibus. Because the two devices don't agree about the end of the word that the byte actually appears in, you have to change the byte address before the move — what you want to do, in effect, is move every byte to the other side of the word which it occupies — the most CPU-efficient way of doing so is to toggle the least significant bit of every byte address.
- The second problem, also related to the Multibus, is a higher level version of the first. It occurs when machine *words* with significant internal structure (or structures that contain words) are moved across the bus interface. (If you write only words, and the device uses only words, there's no problem). The Multibus byte-ordering incompatibility will cause structures to be scrambled when they're moved across the bus interface, unless the bytes within them are physically swapped first.

Here are a few pictures describing the problems in detail:

Motorola (IBM) Byte Ordering



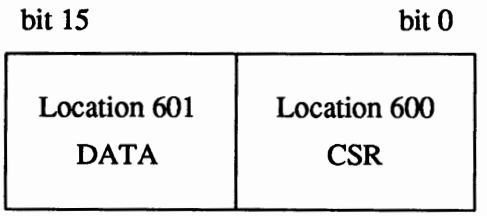
Multibus (DEC) Byte Ordering



That is, the MC680X0 and SPARC CPUs place byte 0 in bits 8 through 15 of the 16-bit word, whereas the Multibus places byte 1 in those bits. If you did everything with the CPU, or everything on the Multibus, there wouldn't be any conflict, since things would be consistent. However, as soon as you cross the boundary between them, the byte order is reversed. Thus, you have to toggle the least significant bit of the address of any *byte* destined for the Multibus — this will have the effect of swapping adjacent addresses and thus reordering the bytes.

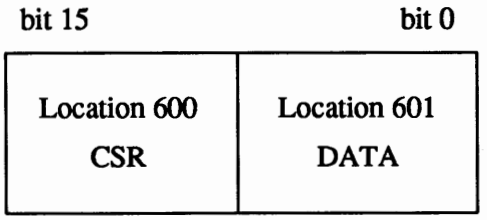
To clarify this, consider an interface for a hypothetical Multibus board containing only two 8-bit I/O registers, namely a control and status register (csr) and a data register (we actually use this design later on in our example of a simple device driver). In this board, we place the command and status register at Multibus byte location 600, and the data register at Multibus byte location 601. The Multibus picture of that device looks like this:

Hypothetical Board Registers



But the MC680X0 and SPARC processors view that device as looking like this:

Hypothetical Board Registers



so that if you were to read location 600 from the point of view of the processor, you'd really end up reading the DATA register off the Multibus instead. So, when we define the *skdevice* data structure for that board, we define it by starting with the register definition in the device manual, and then swapping bytes to take account of the expected byte swapping:

```
struct skdevice {
    char    sk_data;    /* 01: Data Register */
    char    sk_csr;     /* 00: command(w) and status(r) */
};
```

This rule (flipping the least significant bit of the address) holds good for all *byte* transfers which cross the line between the MC680X0/SPARC CPU and the Multibus.

Other Multibus-related Peculiarities

- Many Multibus device controllers are geared for the 8-bit 8080 and Z80 style chips and don't understand 16-bit data transfers. Because of this, such controllers are quite happy to place what's really a word quantity (such as a 16-bit address which must be two-byte aligned in the MC680X0) starting on an odd byte boundary. Some devices use 16-bit or 20-bit addresses (many don't know about 24-bit addresses), and it often happens that you have to chop an address into bytes by shifting and masking, and assign the halves or thirds of the address one at a time, because the device controller wants to place word-aligned quantities on odd-byte boundaries. Note also that many Multibus boards are geared for the 8086 family with its segmented address scheme. An 8086 (20-bit) address really consists of a 4-bit segment number and a 16-bit address; you usually have to deal with the 4-bit part and the 16-bit part separately. For a good example of what we're talking about here, see the code for `vp.c` in the *Sample Driver Listings* appendix to this manual.
- Although there are a myriad of vendors offering Multibus products, remember that the Multibus is a "standard" that evolved from a bus for 8-bit systems to a bus for 16-bit systems. Read vendors' product literature *carefully* (especially the fine print) when selecting a Multibus board. The memory address space of the Multibus is *supposed to be* 20 or 24 bits wide and the I/O address space of the Multibus is *supposed to be* 16 bits wide. In practice, some older boards are limited to 16 bits of address space and 8 bits

of I/O space. In particular, watch for the following addressing peculiarities:

- For a memory-mapped board, ensure that the board can actually handle a full twenty bits of addressing. Older Multibus boards often can only handle sixteen address lines. The Sun system assumes there is a 20-bit Multibus memory space out there. If the Multibus board you're talking to can only handle 16-bit addresses, it will ignore the upper four address lines, and this means that such a board "wraps around" every 64K, which means that on a Sun the addresses that such a board responds to would be replicated sixteen times through the one-megabyte address space on the Multibus. This may conflict with some other device.
- Some Sun-2 Multibus systems, notably Sun-2/170s, have a backplane structure that complicates the installation of 24-bit memory-mapped devices. The internal "bus" on these systems (often called the P2 bus) is divided into multiple segments, each mapped to a portion of the backplane slots. In such systems, 24-bit memory-mapped devices must be installed in a different segment than that used by standard Sun-2 devices. See the *Sun-2/170 Configuration Guide* for more information.
- For an I/O-mapped board (one that uses I/O registers), make sure that the board can handle 16-bit I/O addressing. Some older boards support only 8-bit I/O addressing. In our system, the address spaces of such boards would find themselves replicated every 256 bytes in the I/O address space. Trying to fit such a board into the Sun system would severely curtail the number of I/O addresses available in the system.

- Finally, watch out for boards containing PROM code that expects to find a CPU bus master with an Intel 8080, 8085, or 8086 on it. Such boards are of course useless in the Sun system.

Sun-4/SPARC Peculiarities

There are two peculiarities which are specific to machines built upon the Sun SPARC CPU (currently, just Sun-4's) which can impact device drivers. For more information about the Sun-4 machine architecture, see *Porting C, Fortran and Pascal Programs to the Sun-4*.

- The first problem is structure alignment. In MC680X0 family processors, structures are aligned on half-word boundaries, but on Sun-4's, the structure-alignment requirements are imposed by the most strictly-aligned structure components. For example, a structure containing only bytes and characters has no alignment restrictions, while a structure containing a `double` word must be constructed so as to guarantee that that this word falls on a 64-bit boundary.

  Programmers must be aware of these rules when writing drivers, for Sun-4 compilers will pad structures to enforce them, and such padding will not always be correct for structures intended to map to device registers. Also, structures must be carefully designed if drivers are to be portable across machine architectures.

- The second problem is data alignment. In MC680X0 family processors, characters are aligned on byte boundaries, while integers of all sizes are

aligned on 16-bit boundaries. In Sun-4 machines, in contrast, all quantities must be aligned on their "natural" boundaries: 16-bit half words on 16-bit boundaries, 32-bit words on 32-bit boundaries and 64-bit `double` words on 64-bit boundaries.

In normal programs, details such as these are handled by the compiler. In drivers, however, more care must be taken. SPARC (unlike the MC68010) doesn't break down 32-bit transactions into successive 16-bit transactions. Thus, there are times when 32-bit entities have to be broken down by the driver if they are to get across the bus correctly. More specifically, 32-bit or 64-bit alignment is not possible in the 16-bit VMEbus spaces, and thus 32-bit and 64-bit data access does not exist. In the 32-bit VMEbus spaces, all data paths exist.

Other Device Peculiarities

There are other device peculiarities of interest to the driver developer. These peculiarities are particularly unfortunate in that they tend to require special handling of various kinds — byte swapping, bit shuffling, timing delays, etc. — whenever the driver contacts the device. Such special handling precludes the most obvious and desirable means of interfacing the driver to the device, by mapping the device registers into a C-structure declaration and then accessing them by way of references to structure fields.

- One of the most infuriating of these peculiarities is internal sequencing logic. Devices with this strange characteristic (a vestige of microcomputer systems with extremely limited address space) map multiple internal registers to the same externally addressable address. There are various kinds of internal sequencing logic:
  - The Intel 8251A and the Signetics 2651 alternate the same external register between *two* internal mode registers. Thus, if you want to put something in the first mode register of an 8251, you do so by writing to the external register. This write will, however, have the invisible side effect of setting up the sequencing logic in the chip so that the next read/write operation refers to the alternate, or second, internal register.
  - The NEC PD7201 PCC has multiple internal data registers. To write a byte into one of them, it's necessary to first load the first (register 0) with the number of the register into which the following byte of data will go — you then send that byte of data and it goes into the specified data register. The sequencing logic then automatically sets up the chip so that the next byte sent will go into data-register 0.
  - Another chip of a similar ilk is the AMD 9513 timer. This chip has a data pointer register for pointing at the data register into which a data byte will go. When you send a byte to the data register, the pointer gets incremented. The design of the chip is such that you *can't read the pointer register to find out what's in it*!
- In fact, it's often true that device registers, when read, don't contain the same bits that were last written into them. This means that bitwise operations (like `register &= ~XX_ENABLE`) that have the side effect of

generating register reads must be done in a software copy of the device register, and then written to the real device register.

- Another problem is timing. Many chips specify that they can only be accessed every so often. The Zilog Z8530 SCC, which has a "write recovery time" of 1.6 microseconds, is an example. This means that a delay has to be enforced (with DELAY) when writing out characters with an 8530. Things can get worse, however, for there are instances when it's unclear what delays are needed, and in such cases it's left to the driver developer to determine them empirically.
- And peripheral devices can contain chips that use a byte-ordering convention different from that used by the Sun system into which they're installed. The Intel 82586, for example, supports DEC byte-ordering conventions; this makes it perfectly compatible with Multibus-based, but not VMEbus-based, Sun machines. Drivers for such peripheral devices will have to swap bytes, as indicated above, and to take care that, in doing so, they don't inadvertently reorder the bits in any control fields greater than 16 bits in length.
- Finally, there are some common interrupt-related peculiarities worth noting:
  - When a controller interrupts, it does *not* necessarily mean that both it *and* one of its slave devices are ready. Some controllers are designed in this way, but others interrupt to indicate that the controller or one of its devices *but not necessarily both* is ready.
  - Not all devices power up with interrupts disabled and then start interrupting only when told to do so.
  - While there should be a way to determine that a board has actually generated an interrupt — an attention bit or something equivalent — some devices have no such thing.
  - Finally, an interrupting board should shut off its interrupts when told to do so (and also after a bus reset). Not all do.

2.5. DMA Devices

Many device controller boards are capable of what is known as Direct Memory Access or DMA. This means that the CPU can tell the device controller for such devices the address in memory where a data transfer is to take place and the length of the data transfer, and then instruct the device controller to start the transfer. The data transfer then takes place without further intervention on the part of the processor. When it's complete, the device controller interrupts to say that the transfer is done.

Sun Main-Bus DVMA

NOTE *Sun-2, Sun-3, Sun-3x, and Sun-4 machines use Direct Virtual Memory Access (DVMA) to allow devices on the Main Bus (either a VMEbus or a Multibus) to perform DMA transfers from and to system virtual address space. In the Sun386i system, however, the Memory Management Unit (MMU) is incorporated directly on the Intel 80386 chip itself; devices need to use physical addresses. Sun386i*

DMA is discussed in the next Section.

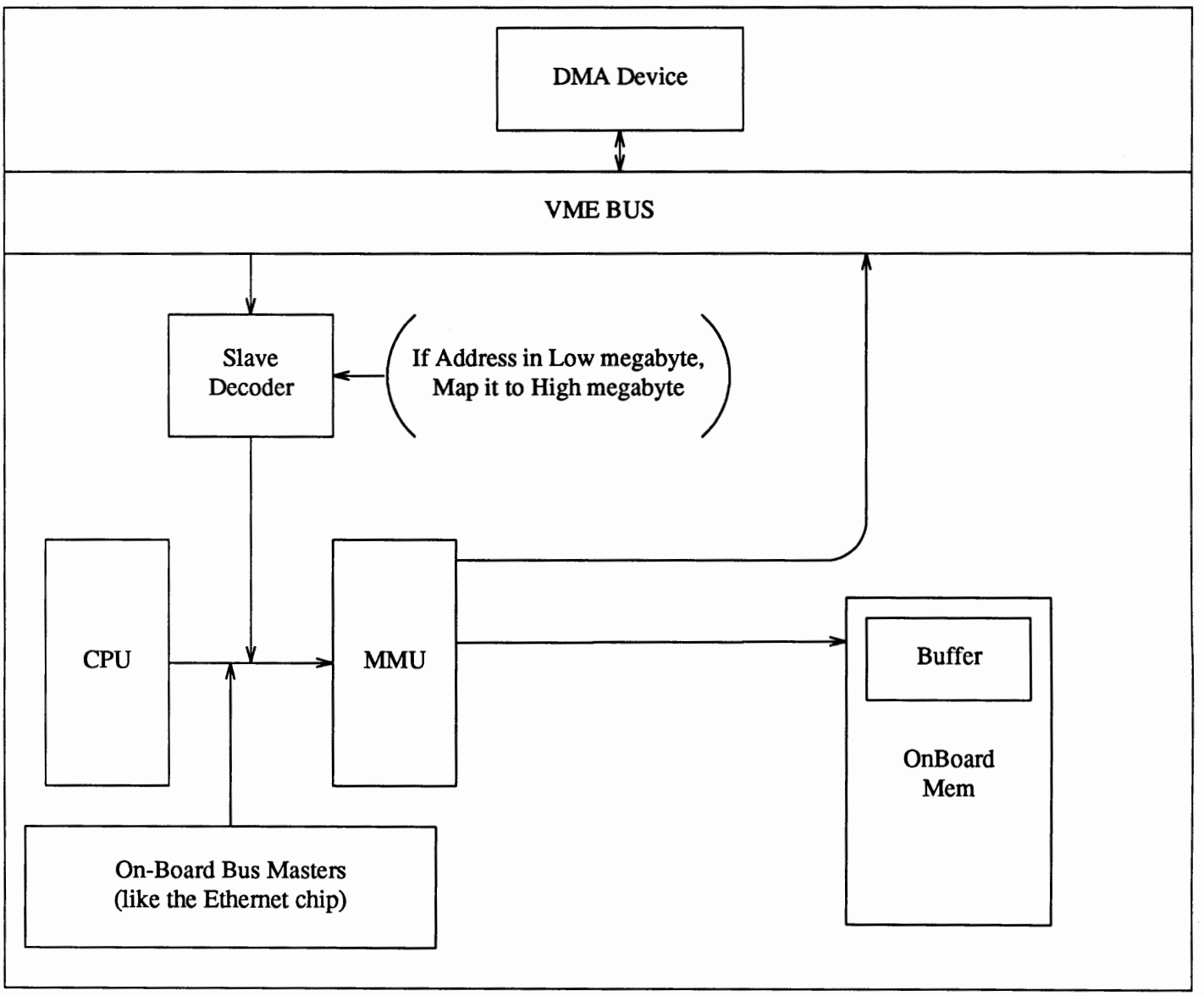
Direct Virtual Memory Access (DVMA) is a mechanism provided by the Sun Memory Management Unit to allow devices on the Main Bus (either a VMEbus or a Multibus) to perform DMA directly to Sun processor memory. It also allows Main Bus master devices to do DMA directly to Main Bus slaves without the extra step of going through processor memory. DVMA works by ensuring that the addresses used by devices are processed by the MMU, just as if they were virtual addresses generated by the CPU. This allows the system to provide the same memory protection and mapping facilities to DMA devices as it does to the system CPU (and thus to programs).

When setting up a driver to support DMA, it's necessary to know the device's DMA address size. This address size is the primary factor used in determining which of the system address spaces will host the device. Multibus devices generally have a DMA address size of 20 bits, while VMEbus devices generally have a 24 or 32-bit DMA address size.

- Since, on Sun-2 Multibus machines, DMA addresses are generally 20-bits long, the system DVMA hardware responds to the first 256K of Multibus address space (`0x0` to `0x3FFFF`). When an address in this range appears on the bus, the DVMA hardware adds `0xF00000` to it (the system places the Multibus memory address space at `0xF00000` in the system's virtual address space) and then uses the MMU to map to the location in physical memory that will be used for the data transfer.
- On Sun-2 VMEbus systems, the DVMA hardware responds to the entire lower megabyte of VMEbus address space (`0x0` to `0xFFFFF`). The system maps addresses in this range into the most significant megabyte of system virtual address space (`0xF00000` to `0xFFFFFF`).
- On the Sun-3, Sun-3x, and Sun-4 systems the DVMA hardware responds to the lowest megabyte of VMEbus address space *in both the 24-bit and 32-bit VMEbus spaces.* It maps addresses in this megabyte into the most significant megabyte of system virtual address space (`0x0FF00000` to `0xFFFFFFF` for the Sun-3 and `0xFFF00000` to `0xFFFFFFFF` for the Sun-3x and Sun-4). The Sun-3, Sun-3x, and Sun-4 DVMA hardware use supervisor access for checking protection.

The driver writer must account for these mappings, as should be evident from the diagram below.

Figure 2-6 *System DVMA*



Devices can only make DVMA transfers in memory buffers which are from (or redundantly mapped into — see below) the low-memory areas reserved as DVMA space. The memory-management hardware will then recognize references to these areas and map them into the high megabyte of system virtual address space, an area known as DVMA space. Likewise, if a driver needs to allocate space for a DMA transfer, it must do so by way of a mechanism that guarantees its allocation from DVMA space. There are several ways of making this guarantee:

- `rmalloc()` can be used with the `iopbmap` argument. This will get a small block of memory from the beginning of the DVMA space. Such small blocks of memory are usually used for control information, and not for large

blocks of data.

- For a large buffer, the driver can statically declare a `buf` structure (which is a buffer header that contains a pointer to the data) and then use `mbsetup()` to allocate a buffer for it from DVMA space. This mechanism is primarily intended for block devices but is perfectly adaptable for use by character devices that need large DMA buffers.

When dealing with addresses which are in DVMA space, the driver must strip off the high bits by subtracting the external variable `DVMA`, which contains the address of DVMA (declared as an array of characters). `DVMA` is initialized by the system to either `0x00F00000` (for Sun-2s) or `0xFFF00000` (for Sun-3's, Sun-3x's and Sun-4's). If the driver fails to make this adjustment, the device will attempt to use a null address — in the high megabyte — and the CPU board will not respond to it.

NOTE *Addresses received by way of* `mbsetup()` *(and* `MBI_ADDR()`*) do not have to be adjusted in this fashion, as* `mbsetup()` *will have already adjusted them to be relative to the start of DVMA space.*

When the device, in turn, uses the address, the address reference comes down the bus and through a slave decoder, which adds the machine-specific offset to it to map it back into the high megabyte of system virtual memory.

Sun DMA is called DVMA because the addresses which the device uses to communicate with the kernel are virtual addresses like any others. The driver, as part of the kernel, is privy to implementation dependent information, and knows that it must chop off the high-bits of any address intended for the device. This allows the MMU to recognize the addresses destined for the Main Bus and to act accordingly. The device, however, knows nothing of this except that its buffers are mapped to the high megabyte of system virtual memory.

User processes, it should be noted, cannot do DVMA directly into their own address spaces. The kernel, however, provides a way of getting around this limitation by supporting the redundant mapping of physical memory pages into multiple virtual addresses. In this way, a page of user memory (or, for that matter, a page of kernel memory) can be mapped into DVMA space in such a way that transferred data immediately appears in (or immediately comes from) the address space of the process requesting the I/O operation. All that a driver need do to support such direct user-space DVMA is to set up the kernel page maps with the routine `mbsetup()` — the details of the mapping will then be automatically handled by the kernel.

If you wish to do DMA over the Main Bus, you must make the appropriate entries in the kernel memory map. There are two functions, `mbsetup()` and `mbrelse()`, to help with this chore.

DMA on ATbus Machines

The Sun386i uses the Intel 80386 chip. This chip has an integrated MMU, so the I/O devices cannot access the Sun MMU address-translation facility and therefore must use physical addresses to access memory directly.

To do DMA on the Sun386i, you must make certain changes in the kernel's memory map (its page tables). Use the `mbsetup()`, `dma_setup()`,

`mbrelse()`, and `dma_done()` routines to make these changes. The changes you must make to the kernel memory map are described with these routines in the *Kernel Support Routines* appendix.

Writing Device Drivers

Sun Microsystems, Inc. • 2550 Garcia Avenue • Mountain View, CA 94043 • 415-960-1300

Part No: 800-1780-10
Revision A, of 24 April 1989

5

Driver Development Topics

5.1. Installing and Checking the Device

The central processor board (CPU) of the Sun Workstation has a set of PROMs containing a program generally known as the "Monitor". (See the appropriate *PROM Commands* chapter of the *PROM User's Manual* for detailed descriptions of the monitor commands and their syntax). The monitor has three basic purposes:

1) To bring the machine up from power on, or from a hard reset (monitor `k2` command).

2) To provide an interactive tool for examining and setting memory, device registers, page tables and segment tables.

3) To boot SunOS, stand-alone programs, or the kernel debugger `kadb`.

If you simply power up your computer and attempt to use its monitor to examine your device's registers, you will likely fail. This is because, while you may have correctly installed your device (a process that includes specifying its virtual memory mapping in the config file) those mappings are SunOS specific, and don't become active until SunOS is booted. The PROM will, upon power up, map in a set of essential system devices — like the keyboard — but your device is almost certainly not among them.

When installing a new device, you will use the monitor primarily as a means of examining and setting device registers. But before even beginning the development of your driver, it's a good idea to attach your device to the system bus and use the monitor to manually probe and test it. This will give you a chance to become familiar with the details of its operation, and to ensure that it works as you expect it to.

Setting the Memory Management Unit

Upon power-up, the PROM monitor:

- Maps the beginning of on-board memory, up to 6 megabytes, to low virtual addresses starting at virtual `0x0`.
- *Sun-2 machines only.* Maps the bus spaces into virtual address space, for the purpose of supporting Multibus devices. Multibus IO space is mapped from `0xEB0000` to `0xEBFFFF` on Sun-2 Multibus machines. On Sun-2 VMEbus machines, `vme16d16` is mapped from `0xEB0000` to `0xEBFFFF` so that Multibus cards attached by way of VMEbus adapter cards can be accessed. These two address spaces, Multibus I/O and `vme16d16`, are *not*

remapped by the SunOS kernel. This means, for example, that kernel virtual address `0xEBEE40` can be used to talk to a device at `0xEE40` in Multibus IO space without setting up a mapping. (This shortcut is *only* possible for the two 16-bit Sun-2 spaces).

Later, using the autoconfiguration process, SunOS makes a pass through the config file (actually, through the `ioconf` file that was produced as output by `config` when it processed the config file). For each device, SunOS selects an unused virtual address (using an algorithm that doesn't presently concern us) and maps it into the device's physical address as specified in the config file.

SunOS then calls the *xx*`probe()` routine for each device, passing it the chosen virtual address. In this way, *xx*`probe()` is kept from having any knowledge of the physical address to which the device is mapped. *xx*`probe()` then determines whether or not the device is present. If it isn't, the virtual address can be reused.

To test a device, ignore the SunOS mappings and use the monitor to manually set the MMU to map your device registers to a known address in physical memory. Then you can use the monitor to verify its proper operation. This verification process will consist primarily of using the monitor's `O` (open a byte), `E` (open a word) and `L` (open a long word) commands to examine and modify the device's registers. Note that, in Sun-4 machines, words and long words are both 32 bits in length.

The process of setting up the device for initial testing consists of three discrete steps.

- The selection of an appropriate virtual address for the testing of the device.
- The determination of the physical address of the device, as well as the address space that it occupies.
- The use of the monitor to map the system's virtual address to the device's physical address. Detailed discussion of these three steps follow.

Since SunOS initializes the MMU in the course of its autoconfiguration process, it's possible to test a device by actually installing it, and then booting and halting SunOS. (You can halt SunOS by pressing the 'L1' and 'A' keys simultaneously, or, on a terminal console, by hitting the <BREAK> key). Having gotten to the monitor by this route, the MMU will be initialized to its SunOS run-time state. You can then use the monitor to test the device, or, if you wish, boot `kadb`. *(A hard reset — the monitor's* `k2` *command — will set the to MMU to its pre-SunOS power-up state). But while using the SunOS memory maps may occasionally be useful, it's not what you want to do during the first stages of device integration.*

Selecting a Virtual Address

First, understand that the MMU, when mapping a virtual address to a physical address, is actually mapping to a page of physical memory and an offset within that page. The low-order bits of a virtual address, those that specify the offset, *do not get mapped* — an address that is X bytes from the beginning of its virtual page will be X bytes from the beginning of whatever physical page it gets

mapped into.

The mapping mechanism is essentially the same for all Sun systems, although the details of address size and page mapping differ. This can be seen in the following diagrams:

Figure 5-1 *Sun-2 Address Mapping*

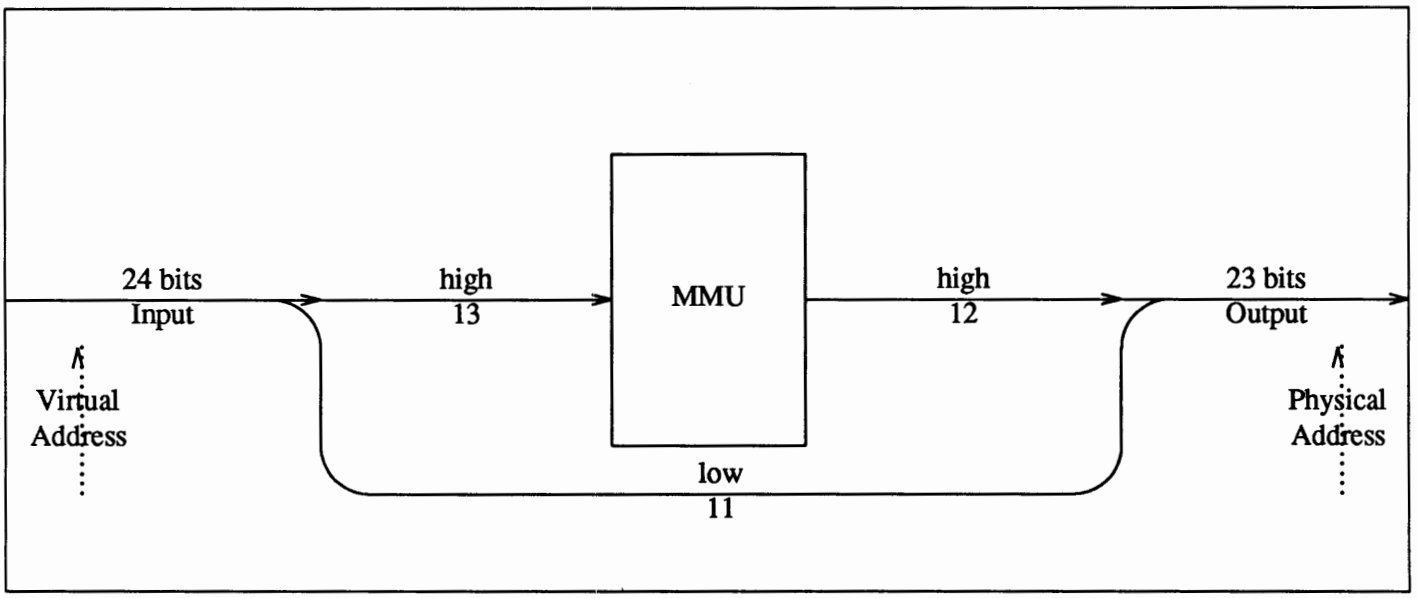


Figure 5-2 *Sun-3 Address Mapping*

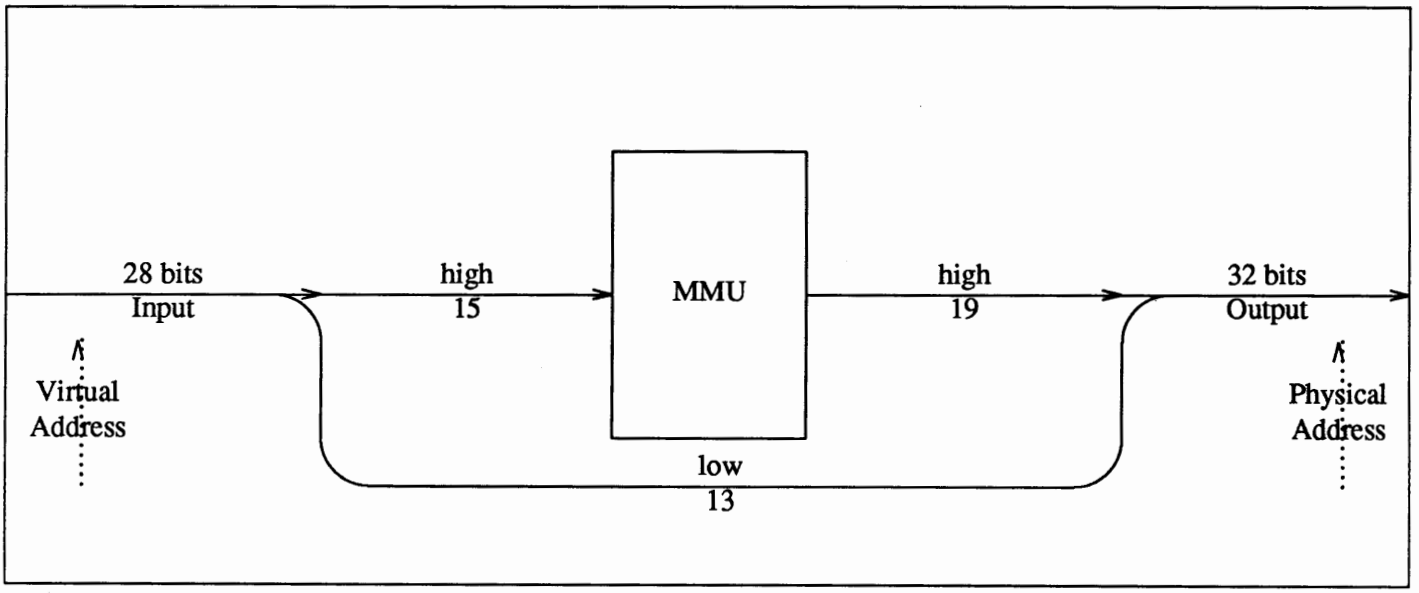


Figure 5-3 *Sun-3x/Sun-4 Address Mapping*

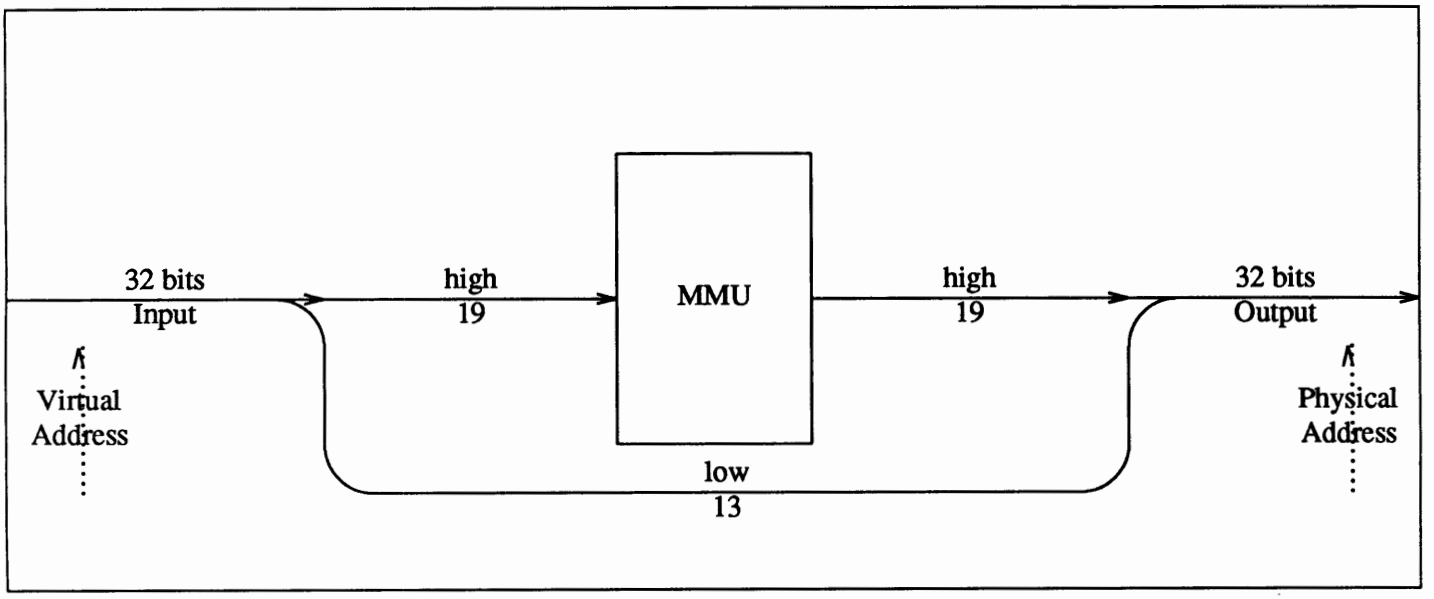
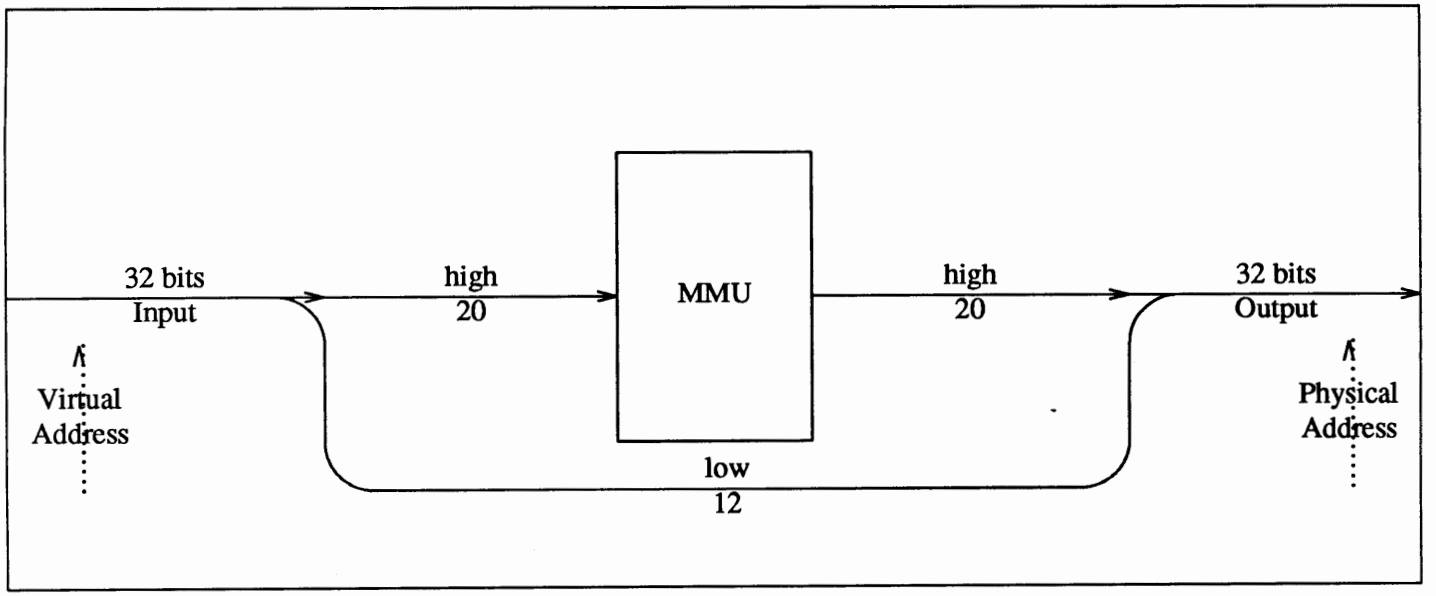


Figure 5-4 *Sun386i Address Mapping*



The easiest way to select a virtual address for PROM-monitor testing is to use one between `0x4000` and `0x100000` on Sun-2, Sun-3, Sun-3x and Sun-4 systems, or `0x20000` and `0x100000` on Sun386i systems. Addresses in these ranges are unused by the monitor in the respective Sun models, and are thus

available. (Note that these addresses, while convenient for testing, are not those that the kernel will choose when your device is finally installed).

It's most convenient to select a virtual address which has only zero's in its low-order bits. This way you select the first address in a virtual page. The low-order bits in the address you choose will remain unchanged. With `'X'` representing the unmapped low-order bits (11 for a Sun-2, 13 for a Sun-3, Sun-3x or Sun-4, 12 for a Sun386i) the test address `0x4000` is, in binary:

```
   Sun-2 :             0000 0000 0010 0XXX XXXX XXXX   (24 bits)
   Sun-3 :        0000 0000 0000 100X XXXX XXXX XXXX   (28 bits)
   Sun-3x:  0000 0000 0000 0000 100X XXXX XXXX XXXX   (32 bits)
   Sun-4 :  0000 0000 0000 0000 100X XXXX XXXX XXXX   (32 bits)
 Sun386i :  0000 0000 0000 0000 0100 XXXX XXXX XXXX   (32 bits)
```

Finding a Physical Address

Your board may be preconfigured to some address. If it is, then use that address unless it conflicts with the address of an already installed device. If it does, you will have to find an unused physical address at which you can install your device. To do so, examine the kernel config file for the system upon which you are working. Tables in the *Hardware Context* chapter show memory layouts corresponding to typical configurations, but if your system has departed at all from the norm, you will have to consult your kernel's config file (to determine where devices have been installed) and the header files for the corresponding device drivers (to determine how much space they consume on the bus).

Selecting a Virtual to Physical Mapping

When selecting a virtual to physical mapping, it's best if you understand a bit about the internals of the Memory Management Unit. The Sun-2, Sun-3, and Sun-4 all use the same proprietary MMU architecture. The Sun-3x uses the MMU that is on the same chip as the CPU. This MMU works completely different than the Sun MMU.

The following description is about the Sun MMU operation as it pertains to the Sun-2, Sun-3 and Sun-4. There is also an example of how to perform a mappings using sample numbers. The Sun-3x description follows the Sun-2/ Sun-3/Sun-4 description and includes a page mapping example.

Sun-2/Sun-3/Sun-4 Virtual to Physical Mapping

Up to this point we've only stressed that the MMU maps the top bits of the virtual address, leaving the offset bits unchanged. Now it will be necessary to explain the mapping process in more detail.

Some new concepts are necessary to discuss the details of virtual to physical memory mapping.

- The *context register (of real concern only on the Sun-2)* is a register specifying which of memory *contexts* should be used when mapping virtual addresses to physical addresses. Sun-2 and Sun-3 Context Registers contain 3 bits, and specify one of eight memory contexts; Sun-4/260 Context Registers contain four bits, and specify one of 16 memory contexts. Each SunOS *process segment* (containing either code, data or stack) is kept within a single memory context.

- Sun-3s have user and kernel address spaces in the same hardware context. That is to say, there is only one virtual address space, a portion of which is used by the kernel and the rest by user processes. Sun-4 virtual address spaces are divided into two chunks. One of them is at the top of the addressable virtual memory space and the other is at the bottom. The size of the unused space between these two spaces varies with the model — in the Sun-4/260 each of the two virtual address spaces is 512 megabytes in size, and the space between them consumes 15 Gigabytes.
- Sun-2s, on the other hand, segregate kernel and user processes into separate hardware contexts with separate address maps. Kernel processes are run in the supervisor context (context 0) and only processes in context 0 have access to the I/O devices.

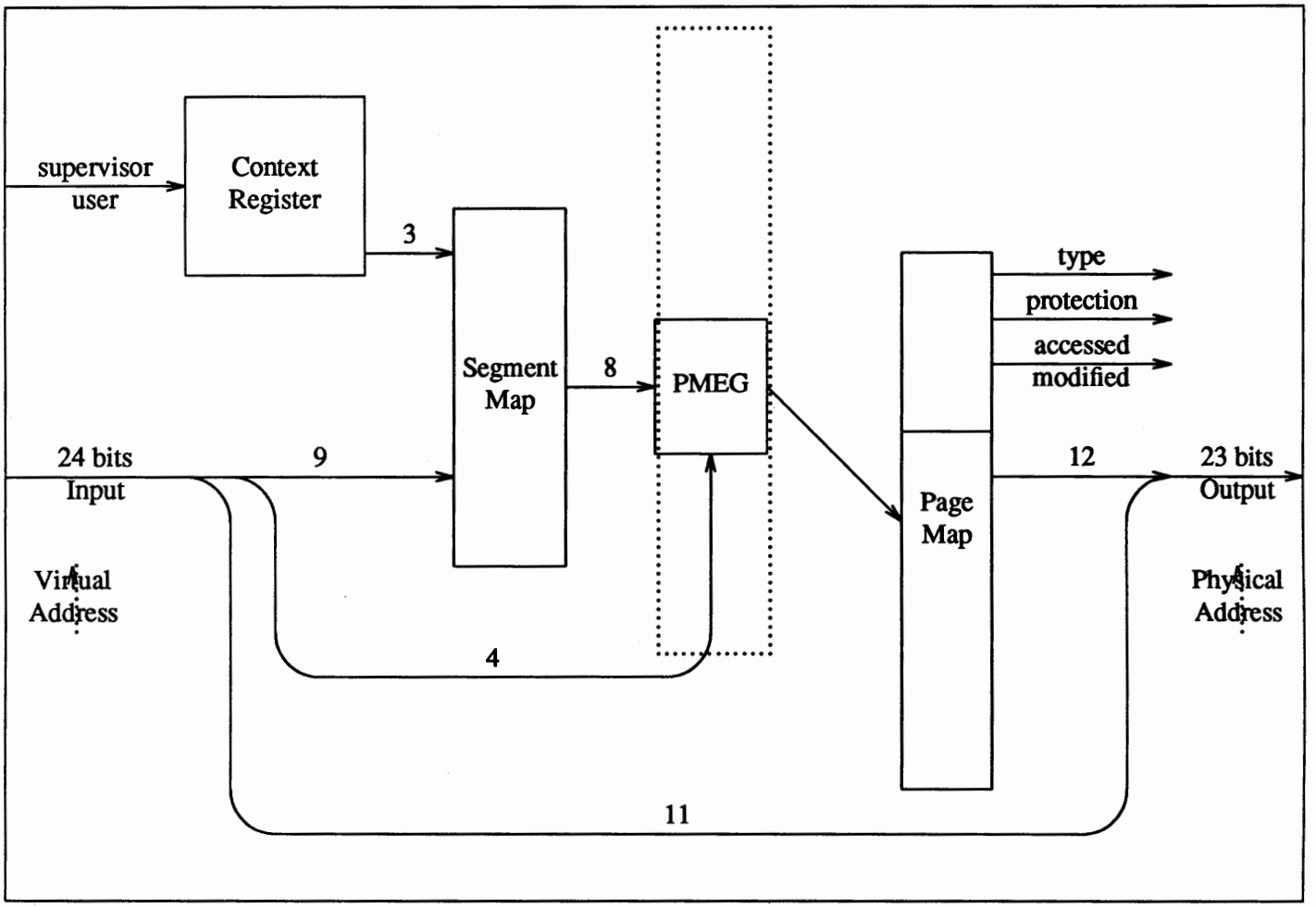
- The *segment map* is used in conjunction with the *context register* to select the *page map entry group* (PMEG) corresponding to the virtual address being mapped. The eight bits in the segment register specify one of a group of 256 PMEGs.
- Within each *page map entry group* there are 16 *page table entries.*
- The *page map* maps the PMEG returned from the segment mapping with a second subfield of the incoming virtual address to exactly specify a single *page table entry* describing the physical page within which the virtual address is mapped.
- The *page table entry* (PTE) is the final output of the MMU. A PTE specifies the physical address of a page, as well as its type (e.g., on-board memory space), protection, and the state of its *access* and *modified* flags.

Note (for Sun-2 machines only): when testing your device, it's necessary to ensure both that you are in supervisor state and that you are in context zero (the kernel context). The monitor normally initializes to supervisor state, but if you enter it by way of an abort from SunOS, you will remain in whatever context you were in at the time of the abort. To be on the safe side, begin all of your monitor sessions with the command `S5`*. This will put you into supervisor data state, where you want to be. Note one important exception to this rule: if you've* `mmap()`*'ed the device into your (user) program's address space and want to check that this worked, you must use the* `S1` *command instead of the* `S5` *command. This will cause user function codes to be used when accessing page maps and data.*

Sun-2 Address Mapping

Note the following diagram of the Sun-2 MMU:

Figure 5-5 *Sun-2 MMU*

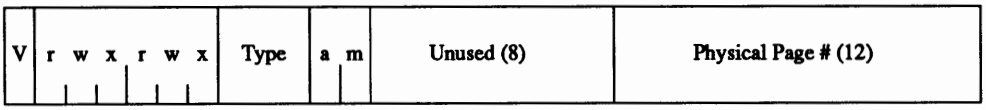


Note that:

- The lower 11 bits of the incoming virtual address are passed through the MMU without being mapped — these are the bits that specify the position within the page, regardless of whether that page is physical or virtual.
- Multiple segment maps can specify the same PMEG, and often do.
- The PTE, on the output side of the MMU, specifies a variety of kinds of status information for the specified page, as well as the top bits of its physical address.

The process of mapping a virtual to a physical address consists, in practice, of plugging the right number into the right PTE. The monitor provides a simple means of addressing the right PTE, but you will have to calculate the right value to plug into it.

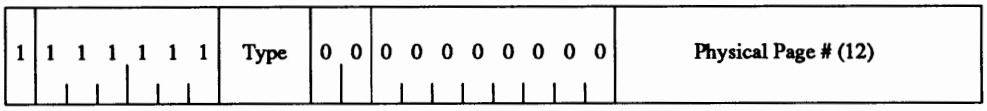
On Sun-2 systems, hardware PTEs are 32-bit numbers with the following structure.



Most of the PTEs that we will deal with will have similar structures, and so we can begin by making a "template" bit mask that we can use to construct our standard PTEs. One acceptable mask will assume values as follows:

```
V (valid) = 1
rwxrwx = 111111
(a/m) accessed/modified = 00
unused = 00000000
```

Thus, we can see that our template will be:



This gives us a mask of `0xFE000000` (if we assume that the type field is `0000`). Now, as already mentioned, there are four types of memory, represented in the PTE by values of 0, 1, 2 and 3 in the type field indicated above. (Types 0 and 1 have the same meaning in both Multibus and VMEbus machines, but types 2 and 3 do not.) Type 2 is used, on Sun-2 VMEbus machines, to designate the first 8 megabytes of the 24-bit VMEbus space — `0x0` to `0x7FFFFF` — and type 3 is used to designate the second 8 megabytes — `0x800000` to `0xFFFFFF`. (But remember that the top 64K of the 24-bit space is stolen for the 16-bit space). This use of two memory types to designate physical memory is necessary because the Sun-2 physical address size, 23 bits, is not sufficient to address all 16 megabytes of `vme24d16`.

Table 5-1 *Sun-2 PTE Masks*

Type	Description	Mask
0	On Board Memory	0xFE000000
1	On Board I/O Space	0xFE400000
2	(Multibus) Memory Space	0xFE800000
3	(Multibus) I/O Space	0xFEC00000
2	(VMEbus) VMEbus Low	0xFE800000
3	(VMEbus) VMEbus High	0xFEC00000

To determine the value which we need to plug into the PTE, we must add the appropriate mask to the appropriate physical page number, thus giving us the full 32-bit number that we need. Here, we will cease to explain details and simply give a series of rules for calculating physical page numbers.

```
If Sun-2 Multibus:

    If Multibus I/O Space, use Type-3 Template
    If Multibus Memory Space, use Type-2 Template

    Physical Page Number = Physical Address >> 11
```

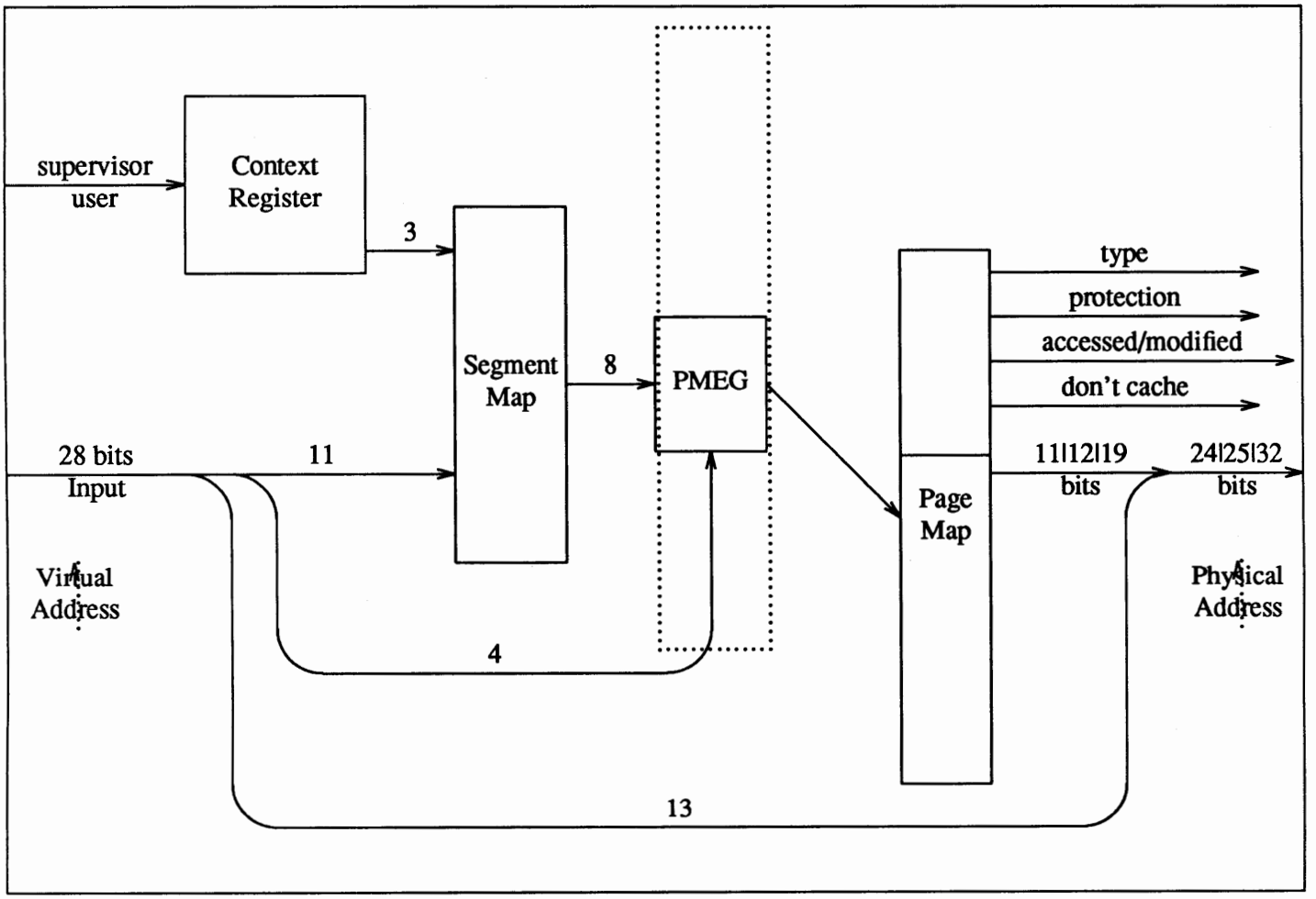
```
If Sun-2 vme24d16:

    If Physical Address >= 0x800000
        Use Type-3 Template
        Physical Page Number =
            (Physical Address - 0x800000) >> 11

    If Physical Address < 0x800000
        Use Type-2 Template
        Physical Page Number = Physical Address >> 11
```

```
If Sun-2 vme16d16

    Use Type-3 Template
    Physical Page Number =
        (Physical Address + 0x7F0000) >> 11
```

Sun-3 and Sun-4 Address Mapping

Consider the following diagram of address mapping on the Sun-3.

Figure 5-6 *Sun-3 MMU*

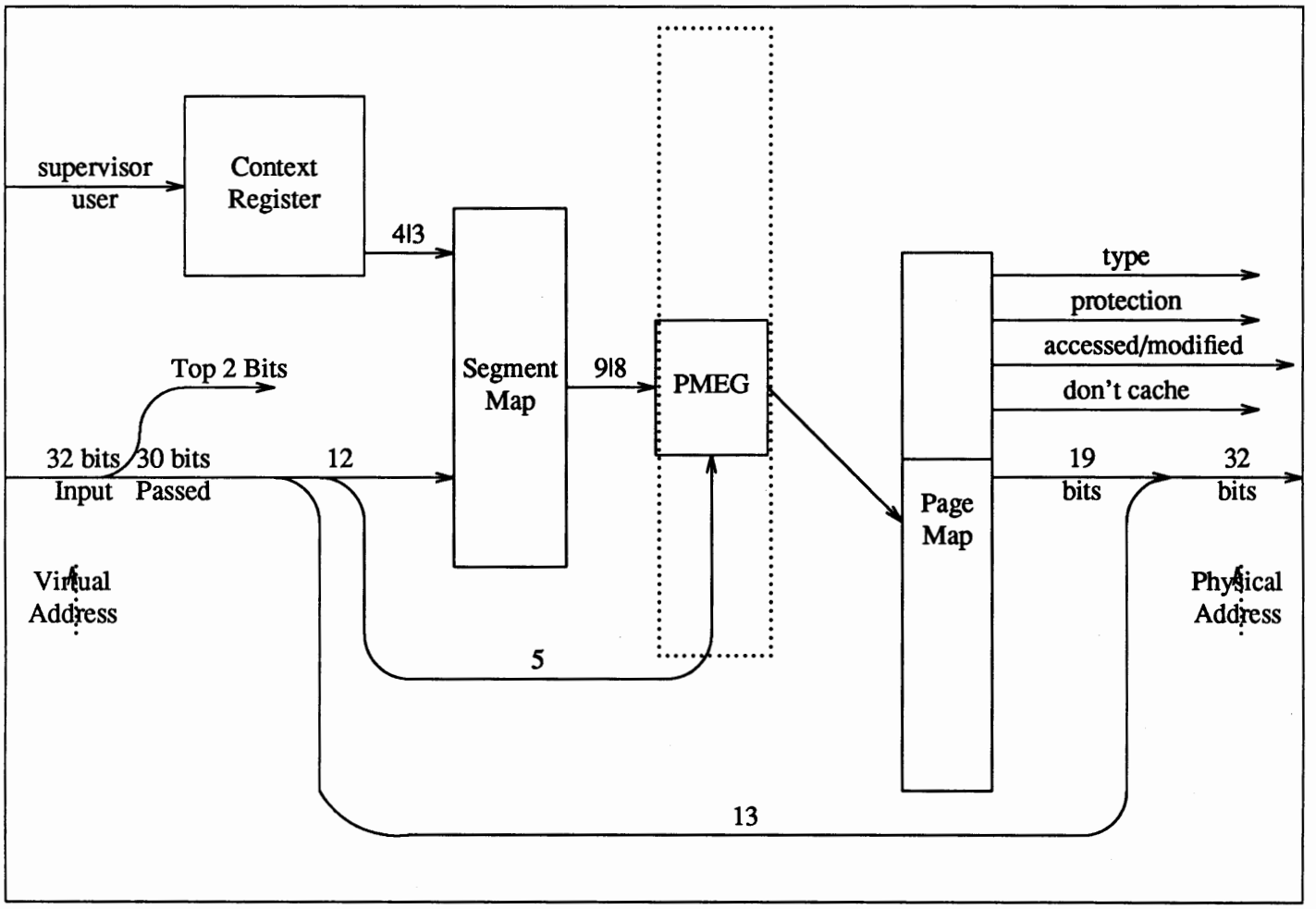


As you can see, the general scheme is the same as it was in the Sun-2, but the details have changed:

- The MMU is getting a 28-bit virtual address as its input, as opposed to a 24-bit address in the Sun-2.
- The number of mode and permission bits in the PTE has been reduced.
- The number of high-order bits reported out of the MMU, and thus the size of the physical address, is variable. The address size is fixed for any given Sun-3 machine, and varies only with the model — there are different kinds of Sun-3 machines and they have different physical address sizes.

The Sun-4 MMU is almost the same:

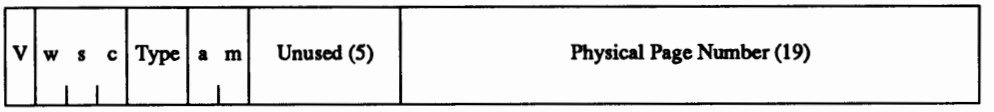
Figure 5-7 *Sun-4 MMU*



As you can see, the Sun-4 MMU is largely identical to the Sun-3 MMU. The differences are that:

- The Sun-4 MMU gets a 32-bit virtual address as its input, as opposed to a 28-bit address in the Sun-3. The top two bits are immediately shunted off. They must be either `00` or `11`, and are used to specify one of the two "chunks" in the virtual address space. (See *Selecting a Virtual to Physical Mapping* above).
- The number of bits coming off the Context Register is 4 (to specify one of 16 contexts) on Sun-4/260s and 3 (to specify one of 8 contexts) on Sun-4/110s.
- The number of bits coming off the Segment map is 9 for Sun-4/260s and 8 for Sun-4/110s.

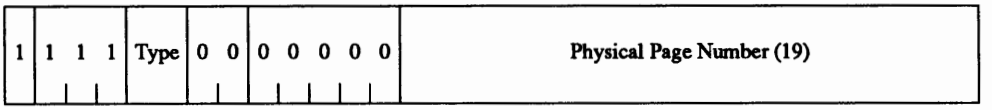
On both Sun-3 and Sun-4 systems, PTEs are 32-bit numbers with the following structure.



As we did with Sun-2 PTEs, we will make a "template" bit mask that we can use to construct our standard PTEs. One acceptable mask assumes values as follows:

```
V (valid) = 1
w/s (write ok/supervisor only) = 11
c (don't cache) = 1
(a/m) accessed/modified = 00
unused = 00000
```

(A one (1) in the don't cache position only disables caching if the type is zero (0), since other types of pages are never cached). With the above values, our template will be:



This gives us a mask of `0xF0000000` (if we assume that the type field is `00`). Thus, the four masks for the four types of memory are:

Table 5-2 *Sun-3/Sun-4 PTE Masks*

Type	Description	Mask
0	On Board Memory	0xF0000000
1	On Board I/O Space	0xF4000000
2	vme16d16	0xF8000000
2	vme24d16	0xF8000000
2	vme32d16	0xF8000000
3	vme16d32	0xFC000000
3	vme24d32	0xFC000000
3	vme32d32	0xFC000000

To determine the value to be plugged into the PTE, we must add the appropriate mask to the appropriate physical page number, thus giving us the full 32-bit number that we need. Here, again, we will give rules instead of details.

```
If vme16d16
   or vme24d16
   or vme32d16

    Use Type-2 Template
```

```
If vme16d32
   or vme24d32
   or vme32d32

    Use Type-3 Template
```

```
If vme32d16
   or vme32d32

    Physical Page Number = Physical Address >> 13
```

```
If vme24d16
   or vme24d32

    Physical Page Number =
        (Physical Address +0xFF000000) >> 13
```

```
If vme16d16
   or vme16d32

    Physical Page Number =
        (Physical Address +0xFFFF0000) >> 13
```

Sun-3x Virtual to Physical Mapping

In the previous CPU board designs, such as the Sun-3 architecture, a discrete MMU was designed and implemented to handle Demand Paging (off chip). That MMU was implemented mostly in hardware, with a dedicated register for the Context and separate high speed RAM for the Segment and Page values. In the Sun-3x architecture where the MC68030 is used as the CPU, a fully programmable Memory Management Unit (MMU) integrated into the silicon (on the 68030 chip) will be used to handle demand paging. A similar MMU has been offered by Motorola for some time (the MC68851 MMU) but was not used by Sun due to certain architectural incompatibilities.

This Memory Management Unit is drastically different in operation from the popular discrete version of its processors. Some of the MMU's most significant changes involve how the Translation Tables are initialized, accessed, and updated and also the way the Address Translation procedure, or Table Walk, is completed. This next discussion provides the process of how the firmware builds, initializes, and updates the entries in the MMU Translation Tables, how the Table Walk is accomplished, and how the MMU performs Address Translation. An example is shown how to use the monitor to map virtual addresses into physical addresses to access devices through the PROM.

The MMU handles the translation of addresses from virtual to physical using translation tables stored at arbitrary locations in memory. The MMU has an

Address Translation Cache (ATC) that holds recently used virtual to physical address translations. When the CPU passes a virtual address to the MMU for translation, it will first search the ATC for the corresponding physical address. If the requested entry is not in the ATC, the processor will search the translation tables in main memory for the information. An ATC access operates in parallel with the other on-chip caches, namely the CPU's Instruction Cache and Data Cache. In order for the MMU to operate correctly, its internal registers must be initialized to a known state.

The MMU has several internal registers that are initialized to known values before the MMU is Enabled (Address Translation Enabled) and during various Reset (k2 or power-on) operations. These registers include the CPU Root Pointer (CRP), the Supervisor Root Pointer (SRP), and the Translation Control (TC) register, all of which are initialized while the MMU is Disabled (Translation Disabled). The CRP and SRP are discussed in the Motorola 68030 Manual, but for now it is important to say that these registers contain the starting addresses for the MMU's table walk.

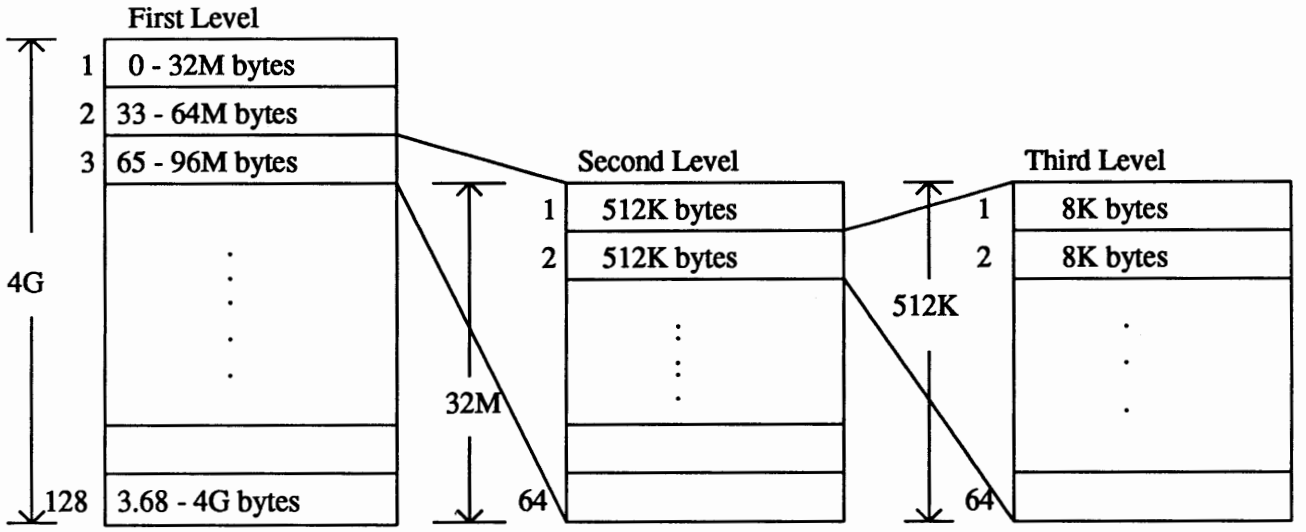
The Table Walk

The MMU's principal function is address translation, which involves converting a virtual or logical address to a physical address. This process is known as a Table Walk. For the Sun-3x architecture a three level MMU has been designed and requires that a three level table walk be initiated to perfrom address translation. This process terminates when either an INVALID Entry or PAGE Descriptor is encountered. The three levels of address translation are referred to as TIA, TIB, and PAGE respectively.

The three level table walk is needed to evenly divide the four gigabyte addressing range of the MC68030. This could have been accomplished several different ways, but a specified design goal was to have the Firmware, the Executive Diagnostic and the Unix Operating System all use the same Translation Table format.

The first level of lookup, the TIA table entry, must be able to map in the entire four gigabyte addressing range all at once. The largest block of virtual memory that is required at any one time will be 32 megabytes. By dividing 4 gigabytes by 32 megabytes we get 128 entries for the first level of address translation. For the second level of translation, the TIB entries will take each of the 32 megabyte TIA entries and divide them by 64. This will allow each TIA entry to be accessed as 64 separate 512 Kbytes (1/2 megabyte) blocks. Each of the 64 TIB entries are then divided into 64 again which results in 8 Kbyte page sizes.

It is because of this table traverse that the name Table Walk is used. Each virtual address is translated to a physical one by taking parts of the virtual address and using them as indexes into the three tables, the resulting output being a Page Table Entry (PTE) which will determine that exact physical address. See the table below for how the entire virtual address range is divided into 8 Kbyte ranges.



The beginning of the table walk starts with a pointer to the location of the MMU tables in main memory. The PMMU has two pointers, one that is used by the CPU (CPU Root Pointer), and one that is used by the CPU while in supervisor state (Supervisor Root Pointer). For the firmware's use, both the CRP and the SRP are initialized to the same value, which means they will both point to the base of the MMU tables.

When the MMU is Enabled, the CPU will pass virtual addresses to be translated to the MMU. If the requested entry is not in the ATC, a table walk of the translation table will be initiated. The table walk sequence is described below.

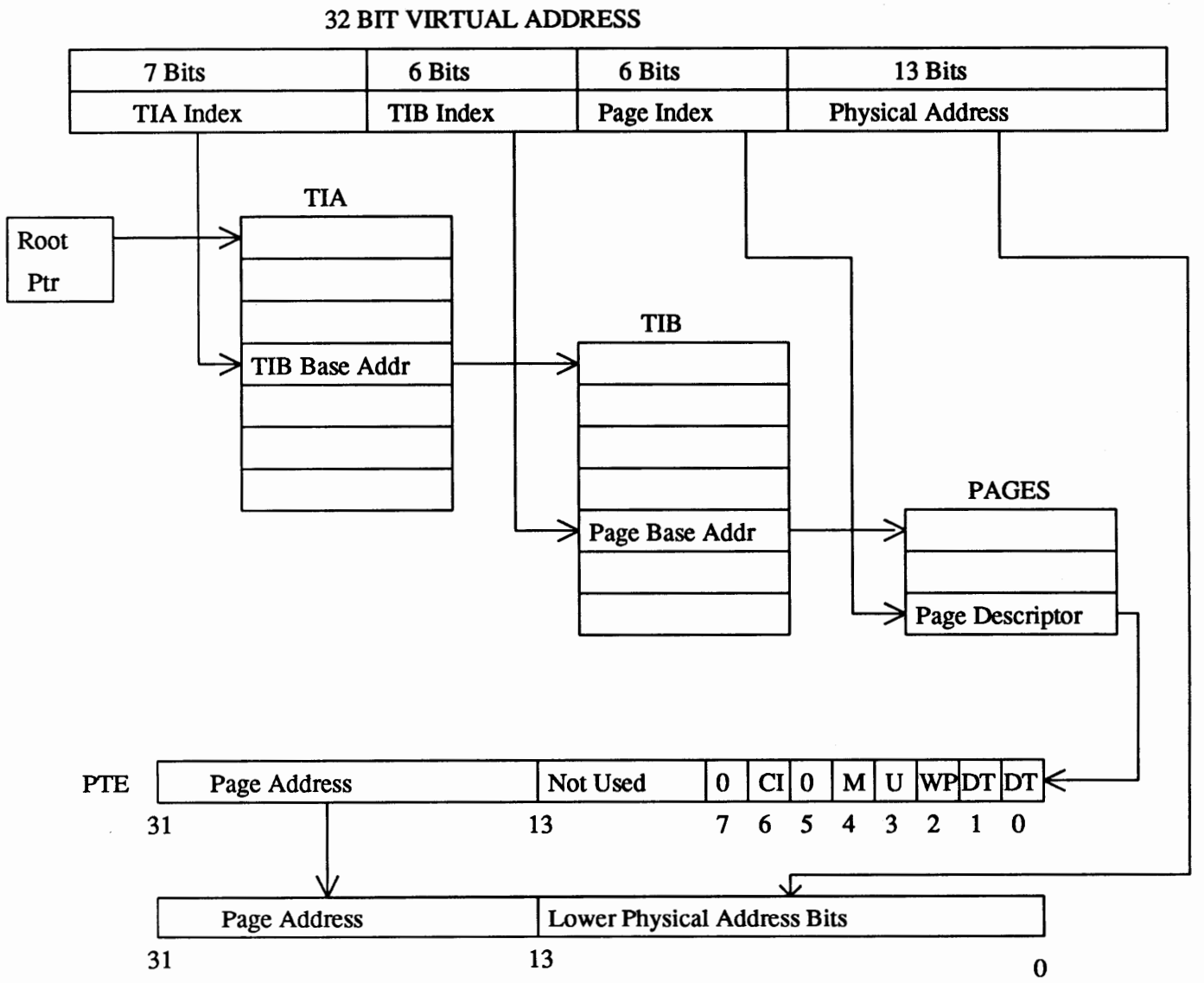
Step One: The CRP contains the base address of the TIA table in memory. The top seven bits of the Virtual Address are used to calculate the index into the TIA table. This index is added to the CRP to generate the specific TIA table entry. The TIA entry contains the base address of the TIB table for the next step.

Step Two: The next six bits of the virtual address are used as an index into the TIB table. When added to the base address from the TIA table the specific TIB table entry is generated. The TIB entry contains the base address of the PAGE Table.

Step Three: The next six bits of the virtual address are used as an index into the PAGE table. The base address from the TIB table plus the index result in the PAGE Table Entry (PTE). The PTE contains a 32 bit PAGE Descriptor of which 19 bits are the Page address, 5 are unused, and the remaining 8 are Status bits.

The Physical address is calculated by taking the top 19 bits from the PTE and the lower 13 bits from the Virtual address. These 13 bits are an offset into the physical memory page that is selected from the 19 bits.

The table walk is completed by passing the physical address back to the CPU. If an INVALID descriptor is ever encountered the table walk terminates.



A Few Example PTE Calculations

Example One: You wish to map a device which you have attached at physical `0x280008` onto bus type `vme24d16` which will be mapped into virtual memory at address `0xE000000`. What is the corresponding PTE?

Sun-3 Solution

Well, since we are mapping the device into `vme24d16`, we will use `0xF8000000` as the template. Then, following the Sun-3 rules, as given above, we add the physical address to `0xFF000000`. This yields `0xFF280008`. In binary, this is:

```
1111 1111 0010 1000 0000 0000 0000 1000
```

Shifting this right by 13 yields:

```
XXXX XXXX XXXX X111 1111 1001 0100 0000
```

Adding the template, 0xF8000000, we get values for the 13 bits that are undefined from the shift. Thus the PTE is:

```
1111 1000 0000 0111 1111 1001 0100 0000
```

Which is 0xF807F940.

A final note: we've now calculated the PTE that maps the virtual page beginning at 0xE000000 to the physical page containing 0x280008. To get the virtual address by which to access the device it's necessary to take the lower 13 bits of the physical installation address — the bits that are just passed through the MMU — and add them to virtual 0xE000000. The lower 13 bits of physical 0x280008 are 0008, and adding them to 0xE000000 yields 0xE000008, the virtual address by which the device can be accessed.

Sun-3x Solution

Our variables are:

```
physical address    280008
virtual address     E000000
bus type            vme24d16
```

The base address for vme24d16 for the Sun-3x, which is in Table 2-8 in Chapter 2, is 0x7e000000 So we add the physical memory address to the vme base pointer which gives us a specific physical address.

```
vme24d16    7E000000
physical      280008
            --------
physical    7E280008
```

Then we take off the top 19 bits to mask out just the vme page, which gives us the physical page of memory. We then need to logically 'or' in some status bits to allow us to write to this page. The value 1 enables the write status.

```
physical    7E280008
and mask    7E280000
            --------
page        7E280000
or flag            1
            --------
PTE         7E280001
```

To use the monitor to perform the mapping, use the 'p' command for displaying and changing the Page Table. The syntax is

```
p[virtual address]
```

where the virtual address is the original virtual memory given in the problem initially. The monitor returns the current PTE and asks you for a new value. The newly calculated PTE is input, which modifies the PTE to map to a new physical memory location

```
monitor cmd      >pE000000<cr>
return value     xxxxxxxx
new PTE          ?7E280001<cr>
exit monitor
```

Now every reference to the virtual memory location `E000008` will be mapped to the device. Note that since the original physical address was folded into the virtual address and then was masked, we still have the 8 offset at the end of the memory reference to index into the physical page of memory to access the device.

Example Two: You wish to map physical `0xEE48` on bus type `vme16d32` on a Sun-3. Using virtual address `0xE000000`, what is the PTE?

Sun-3 Solution

Since we are mapping the device into `vme16d32`, we will use `0xFC000000` as the template. Then, following the Sun-3 rules, as given above, we add the physical address to `0xFFFF0000`. This yields `0xFFFFEE48`. In binary, this is:

```
1111 1111 1111 1111 1110 1110 0100 1000
```

Shifting this right by 13 yields:

```
XXXX XXXX XXXX X111 1111 1111 1111 1111
```

Adding the template, `0xFC000000`, we get values for the 13 bits that are undefined from the shift. Thus the PTE is:

```
1111 1100 0000 0111 1111 1111 1111 1111
```

Which is `0xFC07FFFF`.

To get the virtual address by which to access the device at physical `0xEE48`, add its lower 13 bits, `0xE48`, to `0xE000000` — this yields `0xE000E48`.

Sun-3x Solution

Using the same steps above, this is how the solution looks:

```
physical     EE48
virtual      E000000
bus type     vme16d32

vme16d32     7D000000
physical         EE48
             --------
physical     7D00EE48


physical     7D00EE48
and mask     7D00E000     0111 1101 0000 0000 111
             --------
masked page  7D00E000
or flag             1
             --------
PTE          7D00E001
```

This is the new PTE value that can be used in the monitor as shown in the previous example.

Getting the Device Working and in a Known State

Before you even *think* about writing any code you should check out your device. You must get to know it, finding out early if it has any peculiarities that will affect its driver. It may, for example, have addressing and data-bandwidth limitations. Or, if it's a bus master, it may not implement the *release on request* bus-arbitration scheme the Sun supports. *Know the peculiarities of your device early, and then test it to verify that it's working before proceeding further with driver development.*

Make sure that the board is set up as specified in the vendor's manual. Device characteristics which, in general, have to be set properly before the device can successfully be used include:

- I/O register addresses for I/O mapped Multibus boards,
- Memory base addresses for Multibus boards that use Multibus memory space,
- Address and data widths,
- Interrupt levels,
- Interrupt vector numbers for VMEbus device,
- VMEbus address modifiers,
- The bus grant level for VMEbus devices should be set at 3.

Then, take down your system and power it off. Plug the device into the card cage and attempt to bring the system back up. If you can't boot the system, then there's a problem. Perhaps the board isn't really working, or perhaps it's responding to addresses used by other system devices. You must resolve this problem before proceeding further.

Take SunOS down again and attempt to contact the device using the PROM monitor. To do so, you will need to set up a PTE on the Sun-2, Sun-3, or Sun-4 which maps to the device's physical installation address. Use the procedures given above to calculate a PTE, then:

- Issue the monitor command that puts you into *supervisor data* state. This will be **s B** for Sun-4 machines and **s5** for all others. So, if you have a Sun-3, give the

  ```
  >s5
  ```

  command.

- Calculate, using the procedures given above, the PTE appropriate to the physical address you've chosen.

- Set the position in the kernel page map that corresponds to your physical address to contain the calculated PTE. This will map your chosen physical address, thus putting you in contact with your device. You may use the monitor's **p** command to perform this mapping. The **p** command takes a virtual address as its argument, displays the PTE that corresponds to that virtual address, and gives you the option of modifying the PTE. For example:

  ```
  >pF32000
  ```

  selects the page map entry that corresponds to the virtual address of `0xF32000` and displays it. It also displays a '?', which indicates that you may type in a new value to replace the one displayed. (See the appropriate *PROM Commands* chapter of the *PROM User's Manual* for more details). Note that all virtual addresses within a page select the same PTE.

Having contacted the device from the monitor, try some of the following:

- Try reading from the device status register(s), if there are any.

- Try writing to the device control and data registers(s), if there are any. Then try reading the data back to see if it got written properly (this assumes, of course, that the device allows the reading of these register(s).

- Try actually getting the device to do something by sending it data.

- If the device is a controller with separate slave devices, then switch a slave on and off and watch for changes in the controller status bits.

Your goal is to try to actually operate the device, for a moment, from the monitor. For example, if you have a line printer, try to print a line with a few characters. Be aware that bit and byte ordering issues are critical in this process. The reason you're doing this is to ensure that the device works and that you understand the way it works. When you understand the device's peculiarities, you can proceed to write a driver for it.

A Warning about Monitor Usage

When you use the monitor's **o**, **e** or **l** commands to open a location, the monitor *reads* the present contents of that location and displays them before giving you the option to rewrite them. In the best of all possible worlds, this would present no problems, *but many devices don't respond to reads and writes in as straightforward a fashion as does normal memory.*

For example, the Intel 8251A and the Signetics 2651 use the same externally addressable register to access *two* separate internal mode registers, and they have internal state logic that alternates accesses to the external register between the two internal registers. So suppose that you want to put something in mode register 1 of the 8251. You open the external register, the monitor displays its contents, and you then do your write. If, being cautious, you then read the external register to check that the data you wrote is there, you will find that it's not — because the read will sequence you on to the second register.

To deal correctly with such devices, it's necessary to use the monitor's "write without looking" facility and then read the locations back later to check them. You can write without looking with any of the monitor commands that "open" an area of memory; all that's necessary is that you enter a `value` after the `address` argument. For example:

```
>l [address] [value]
```

This will cause `value` to be written into `address` without first reading its current contents. For more information on hardware peculiarities and the problems that they can cause for the monitor, the *Hardware Peculiarities to Watch Out For* section of the *Hardware Context* chapter.

5.2. Installation Options for Memory-Mapped Devices

Memory-Mapped Device Drivers

Memory-mapped devices are the simplest types of devices to write drivers for. Frequently, however, their essential simplicity isn't obvious from a quick glance at their source code. This is because many memory-mapped devices are frame buffers, and frame-buffer drivers must set up and manage the low-level interface for the Sun window system as well as the standard device interface. Consequently, they tend to be littered with declarations and manipulations related to the "pixrect" (pixel rectangle) system. See the *Pixrect Reference Manual* for more details.

Memory-mapped devices are most frequently installed into Sun systems with simple drivers that map them into user address space (there are sometimes alternatives to such drivers, as you will see below). Such memory-mapped drivers don't really do much. Obviously, *xx*`probe()` and *xx*`mmap()` must exist, for the kernel must be able to check the device installation and perform the actual device mapping. And, in addition, *xx*`intr()` must be real if the device is interrupt driven. But *xx*`open()` and *xx*`close()` are usually stubs, and *xx*`read()` and *xx*`write()` can be calls to `nulldev`.

Keep in mind that the major purpose of a memory-mapped driver is to support the `mmap()` system call. This is very important because user processes which call window code must first map the frame buffer into their address space. They do so with the `mmap()` system call, which is translated by the kernel into a series of calls to the driver's `mmap` routine. Each of these calls returns page table entry information which the kernel needs to map a single page (the next page) of frame-buffer memory into a virtual address space. Here's some very simple driver *xx*`mmap()` code.

```
/*ARGSUSED*/
cgonemmap(dev,off,prot)
    dev_t dev;
    off_t off;
    int prot;
{
    return (fbmmap(dev,off,prot,NCGONE,cgoneinfo,CG1SIZE));
}

/*ARGSUSED*/
int fbmmap(dev, off, prot, numdevs, mb_devs, size)
    dev_t dev;
    off_t off;
    int prot, numdevs;
    struct mb_device **mb_devs;
    int size;
{
    int kpfnum;

    if ((u_int) off >= size)
        return -1;

    kpfnum =
      hat_getkpfnum(mb_devs[minor(dev)]->md_addr + off);
    return kpfnum;
}
```

dev is, of course, the device major and minor number, and *off* is the offset into the frame buffer (passed down from the user's `mmap()` system call). *prot* is also passed down from the user's call, but it is not currently used. As you can see, there's a bit of shuffling around and then a call to `hat_getkpfnum`, which returns a Page Frame Number which *xx*`mmap()` is expected to return.

Note that `mb_dev->md_addr` is the address of the frame buffer from the Main Bus device structure. This is the device installation address as given in the kernel config file. The offset is checked to be sure the user isn't mapping beyond the end of the frame buffer.

Mapping Devices Without Device Drivers

Under a restricted set of circumstances, it's possible to avoid writing a device driver altogether by using the `mmap()` system call to overlay the device's registers and memory onto user memory. Having done this, you can read and write the registers — as if they were normal user memory — from a user program.

What this really amounts to is piggybacking the new device onto an another, system standard, virtual memory device (and its driver). The `mmap()` routine of a system virtual memory device is then used to do the user-device mapping, and the "installation" is accomplished without the development of a driver specific to the user device. Instead, a user level program is written, one that calls the `mmap()` system call.

The restrictions on this shortcut are, however, fairly severe.

- The device must not require any special handling of the type that would go into *xx*`ioctl()`.
- The device (including all its control registers) must work with user function codes, since that's what it will get when mapped into and then accessed from user space.

NOTE *MC680X0 processors, SPARC processors and the Intel 80386 all run in either 'user' or 'supervisor' state. Many devices, in turn, restrict certain of their operations, and will only perform them when the processor is in supervisor state. The Sun CPU is in supervisor state only when executing kernel code. This means that device drivers, which are part of the kernel, can issue device commands which are not available from user processes. Also note that, when the CPU is in supervisor state, as it is when driver code is executing, the device will receive different VMEbus address modifier codes than when the CPU is in user state. For details about these codes see the VMEbus specification.*

- The device must not require any other sort of special handling — it cannot, for example, be multiplexed, interrupt driven, or do DMA.
- Finally, there are security problems associated with this sort of installation. Since the system virtual-memory devices are normally owned by and restricted to the superuser, your programs will either have to change their permissions to allow normal users to access them, or will have to run with superuser privileges. The former strategy is usually not acceptable in the long run, because it creates a gaping hole in the security of the system. And it's far from clear that the second alternative is desirable either.

The virtual-memory devices of interest here are those that support mapping over the entire range of a virtual address space. They are:

Table 5-3 *Virtual Memory Devices*

Machine Type	Memory Device Name
Multibus (Sun-2 only)	`mbmem`
Multibus (Sun-2 only)	`mbio`
VMEbus (All Sun's)	`vme16d16`
VMEbus (All Sun's)	`vme24d16`
VMEbus (Sun-3 and Sun-4)	`vme32d16`
VMEbus (Sun-3/Sun-3x/Sun-4)	`vme16d32`
VMEbus (Sun-3/Sun-3x/Sun-4)	`vme24d32`
VMEbus (Sun-3/Sun-3x/Sun-4)	`vme32d32`
ATbus (Sun386i only)	`atmem`

In addition, there are memory pseudo-devices that support access to the on-board devices that users are allowed to access. These are `/dev/fb`, `/dev/mem` and `/dev/kmem` (See the `mem(4)` manual page for details).

`/dev/fb` is a memory device which, on any given system, is set up to address the local frame-buffer device. It can be used as if it were a system memory device — on any given system, `/dev/fb` can be `mmap()`'ed into user memory and then written to, with the effect of writing the local frame buffer memory.

To use `mmap()` with one of the system memory devices, you must do three things:

- Open the device.
- Calculate the *offset* which you will need to call `mmap()`. This offset is merely the device address on the appropriate system memory device rounded to a page boundary. That is to say that you get the offset from the device manual and/or the switches on the device itself.
- Call `mmap()` to allocate virtual space and map in the physical bus address of your device, which you must know. (See the *Hardware Context* chapter for a discussion on how to pick a good physical address from the information in the system config file).

The following example program uses `/dev/fb` rather than one of the virtual memory devices. It makes a good example because it maps the system frame buffer into user memory so that it can then be written from a user program. It uses `mmap()` to set things up, but doesn't bother with calling `munmap()`, because unmapping occurs automatically when the memory device is closed. This close occurs implicitly when the program ceases execution. (The machine segment size is 128K for the Sun-2 and Sun-3; 256K for the Sun-4; and 4Mbytes for the Sun386i. Areas greater than the machine segment size should be mapped only with special care. The Sun-3x has no segment size so any input value will work. For details, see the discussion of `mmap()` in the *User Support Routines* appendix).

Once the device has been mapped into user space it can be treated as a piece of local user memory. (Remember that memory accesses performed by way of this mechanism will be reflected — at the device level — as non-privileged (user) accesses. This is because `mmap()` accesses inherit the privilege of the process that calls `mmap()`. Thus, if memory is mapped by a driver, subsequent accesses to it will have the standard supervisor data access privilege, but if it's called from a user process, as described here, subsequent accesses will be non-privileged. Attempts to access supervisor-only device registers without supervisor privilege might produce a bus error, i.e., they're inaccessible from a user program, and thus a kernel level driver must be written to manipulate them. The device will also receive different address modifier codes when accessed from a user process than when accessed via a device driver).

```
#include <stdio.h>
#include <sys/file.h>
#include <sys/mman.h>
#include <sys/types.h>

/* Width and Height of Frame Buffer in Bits */
#define WIDTH 1152
#define HEIGHT 900

main()
{
    int fd;
    unsigned len;
    char *addr;

    /* Open the frame-buffer device */
    if ((fd = open("/dev/fb",O_RDWR)) < 0)
        syserr("open");

    /* Compute total number of bytes */
    len = ((WIDTH * HEIGHT)/8);

    /*
     * offset must be page aligned.  /dev/fb
     * is already aligned with frame-buffer memory
     */
    offset = 0;

    /* Map device memory to user space */
    addr = mmap((caddr_t)0, len, PROT_READ|PROT_WRITE,
        MAP_SHARED, fd, 0);
    if (addr == (caddr_t)-1)
        syserr("mmap failed");

    writeFB(addr);
    exit(0);
}
```

```
writeFB(addr) /* Write to frame buffer */
    char *addr;
{
    char color;
    int i,j;

    color = 0xFF;
    for (i = 0; i < HEIGHT; i++) {
        color = ~color;
        for (j = 0; j < WIDTH/8; j++)
            *addr++ = color;
    }
}

syserr(msg) /* print system call error message and terminate */
    char *msg;
{
    extern int errno, sys_nerr;
    extern char *sys_errlist[];

    fprintf(stderr,"ERROR: %s (%d", msg, errno);
    if (errno > 0 && errno < sys_nerr)
        fprintf(stderr, "; %s)\n", sys_errlist[errno]);
    else
        fprintf(stderr,")\n");
    exit(1);
}
```

NOTE *This example uses the special memory device* `/dev/fb`, *since this device is always set up to address the frame buffer memory.*

So, despite the plethora of limitations on the sorts of devices that can be installed by way of mapping them into user space, it's quite an easy thing to do. If your device characteristics are such that this is an option, you may well wish to take it. And even if such an installation isn't an attractive long-term option (for example, because of unacceptable security problems) it may still be attractive as a short-term alternative to driver development. Even in environments where security considerations make it unacceptable in the long term, it can allow you to get your device up and running very quickly. Sometimes this counts for a lot.

Direct Opening of Memory Devices

It should be noted, for the purpose of completeness, that there's another approach to avoiding driver development, one that's even easier than the use of `mmap()` described here, and even more limited. That is, it's possible to simply open the virtual memory device that contains your board, to seek to the location of its registers, and then to read and write those registers as if they were regular memory.

This approach has most of the same problems as does the use of `mmap()`, *and is notable mainly because, with it, the device receives supervisor function codes. It does, however, introduce new problems.* It doesn't give you the same degree of control as does `mmap()`, and you often need that control when dealing with

devices. When you use mmap(), the device actually becomes part of your user memory space, and it's left to the compiler to generate exactly the I/O accesses which you implicitly specify in your structure and variable declarations. You can always access exactly what you want, and the accesses occur directly as *move byte* and *move word* operations. Thus they are very fast.

When, however, you simply open a system memory device as a file and then read and write to it, your communication with your board is mediated by the I/O system. The I/O systems will always try to do the "right thing" (if you request I/O at an odd address or for an odd number of bytes it will perform byte access as appropriate; otherwise it will use short integers), but it still doesn't give you the kind of control that can be had using mmap(). Furthermore, I/O operations involve lots of code, and take *hundreds* of times as long as direct references to mmap() 'ed references, which proceed by way of the MMU and use low-level store and move instructions to directly access device registers and memory as physical memory.

So the bottom line is that, unless you need to access a device only a few times, or if you need to receive supervisor function codes (and the corresponding VMEbus address-modifier codes) and performance isn't critical, you can do your installation by opening a system memory device and then seeking to your device registers and memory space. Otherwise, use mmap() or write a driver. If you do decide to use the open()/lseek() method, do so with low-level I/O rather than with the standard I/O library. The standard I/O library implements a buffered I/O scheme which will add considerably to your problems.

The following user program is similar to the example above, in that it writes the same pattern to the memory of a frame buffer. This time, though, the write is done by way of the I/O system rather than by using mmap(), and the frame buffer is taken to be installed at *OFFSET* (whatever the device physical installation address is) in the vme24d16 memory space.

NOTE *Since all Sun VMEbus machines have a built-in, on-board frame buffer, this example is only meaningful for color frame buffers. On Sun-2 Multibus machines, however, this code would work with* /dev/obmem *and an offset of* BW2MB_FB.

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/param.h>
#include <sys/buf.h>
#include <sys/file.h>

void syserr();
long lseek();

/* Width and Height of Frame Buffer in Bits */
#define WIDTH 1152
#define HEIGHT 900

main()
{
```

```
    int fd;

    /* Open the system memory device containing the frame buffer */
    if ((fd = open("/dev/vme24",O_RDWR)) < 0)
         syserr("open");

    /* Seek to the frame buffer memory */
    if (lseek(fd, (long)OFFSET, L_SET) == -1L)
        syserr("lseek");

    writeFB(addr);
    exit(0);
}

writeFB(fd) /* Write to frame buffer */
    int fd;
{
    char color;
    int i,j;

    color = 0xFF;
    for (i = 0; i < HEIGHT; i++) {
        color = ~color;
        for (j = 0; j < WIDTH/8; j++) {
            if (write(fd, &color, 1) == -1)
                syserr("write");
        }
    }
}
```

5.3. Debugging Techniques

As described above, it's a good idea to begin debugging by using the monitor to check that the device has been installed at the intended address, and that it works, before proceeding to debug your device driver. This allows you to avoid debugging the device simultaneously with the driver, and experience that you'd like to avoid for as long as possible. Alternatively, if you're confident in both your device and the correctness of your installation, you can simply make a new kernel, boot it and proceed with debugging. In this case you should put some `printf()` messages — see below — into the *xx*`probe()` routine. Then you can at least see the device get contacted and initialized.

Debugging drivers is significantly more difficult than debugging regular user programs, for a number of reasons:

- In the first place, device drivers are part of the system kernel. This means that the system is not protected from their errors. Addressing errors, for example, will frequently trip hardware traps and crash the system.
- As mentioned above, there's the possibility that the device hardware will be buggy. For this reason, you can't really trust your environment in the same way as you can when writing a user program on a mature computer system.

- Some devices behave in rather peculiar ways. (See *A Warning about Monitor Usage*, above).
- Finally, the debugging environment in the kernel is thinner than it is in user space. There is a kernel debugger, `kadb`, and this a a big step towards making life easier for driver developers. Still, life remains more difficult when debugging in kernel space.

  It's possible to prototype drivers in user address space by using techniques similar to those described in the Mapping Devices Without Device Drivers section of this chapter. The same constraints given there apply to prototyping. In particular, it's not possible to run an interrupt routine, or to probe for non-existent devices without generating bus errors from prototype drivers in user space. If the device generates no interrupts, and if it doesn't do DMA, the entire driver might be able to run in user space.

For all these reasons, you should give extra care to desk-checking your code, and check a reference manual when not absolutely sure of the meaning of a given construction. Don't take chances.

Also, make changes incrementally. Don't try to save time by making many changes at once. You will save time in the long run if you take the time to add and test a few parts at a time. Keep your feet on solid ground.

Use trace output from `printf()`, as described below. Drivers can act in surprising ways, and the best way to proceed is by making the flow of operations highly visible.

NOTE *On the Sun386i system, the loadable drivers feature makes driver development much easier because the code-compile-reboot-test cycle is reduced to code-compile-load-test.*

Debugging with `printf()`

With the availability of `kadb`, the kernel debugger, the importance of `printf()` in the debugging of device drivers has been significantly reduced. Still, even with `kadb` available, `printf()` statements remain useful as means of providing synchronous tracing of overall driver flow and structure. `kadb` can be made to provide a similar sort of tracing (by tying print commands to strategically chosen breakpoints) but this won't altogether eliminate the `printf()` statement. The `printf()` has long found application in driver debugging, and, as a matter of taste and experience, some programmers will continue to use it. For this reason, we will discuss its use in some detail.

The kernel `printf()` sends its message directly to the systems console, without going through the tty driver. As a consequence, the printing is uninterruptible—the characters aren't buffered. Furthermore, `printf()` runs at high priority, and no other kernel or user process activity takes place while its output is being produced. `printf()` thus radically limits overall system performance (though this is usually ok while device drivers are being debugged).

The window systems should not be up when you use `printf()` to debug a driver because its output will go to the console window. On

the Sun386i system, it is best to set the global variable `newlog` to 0.

There is a second kernel print statement, `uprintf()`. `uprintf()`, however, is of little use to driver developers. It attempts to print to the current user tty as identified in the `user` structure, and prints to the console only if there's no current user tty (at which point it becomes identical to `printf()`). `uprintf()` cannot be called from lower-half routines, which run in interrupt context and cannot make any assumptions about the `user` structure (where `uprintf()` looks to determine the current user tty). `uprintf()` is most useful for production drivers, like tape drivers that encounter media errors, which want to report errors not to a programmer but to the user.

There are occasions in which the use of `printf()` *(or* `uprintf()`*) statements will change the behavior of your driver.* `printf()` *statements, for example, can affect the timing of operations in the driver being tested as well as in other drivers. The output may be so slow relative to other device operations that interrupts are lost and system failures are introduced; thus, it is frequently impossible to synchronously trace a device interrupt routine. Driver code may begin to fail only when* `printf()`*s are introduced, or, even worse, only when* `printf()`*s are disabled. If you're debugging a tty driver, you may even face a situation where* `printf()`*-based tracing generates new calls to the driver being debugged. Thus, there are situations in which it cannot be used. In such situations, you should use* `kadb` *or the techniques suggested below in the section on Asynchronous Tracing.*

The best way to use `printf()` statements for tracing driver execution is by setting things up so that you can toggle printing by using the kernel debugger, `kadb` (see below) to set and reset print-control variables. Doing so is very simple. At the top of the driver source file, include statements like:

```
#ifdef XXDEBUG
int xxdebug = 0;
#define XXDPRINT if (xxdebug > 0) printf
#endif
```

(It's important that the variables like `xxdebug` be global, so that you can later access them freely from the debugger — remember that all drivers are part of one program, the kernel, and name your print-control variables so as to avoid naming conflicts).

Then, instead of calling `printf()` inside the driver routines, call `XXDPRINT`. Each call should be in the form:

```
#ifdef XXDEBUG
XXDPRINT("driver name...",...);
#endif
```

which will only call `printf()` if `XXDEBUG` is defined and `xxdebug` is set to a value greater than 0.

Make sure that each call to `XXDPRINT` identifies the driver, for it's possible that you, or some other programmer, will want to see debugging output from several drivers at once. And leave the debugging code in for a while after you're

finished — bugs may surface later.

Having set things up like this, you can turn the `printf()`'s on or off at any time by using `kadb` to set unset or change the print-control variable `xxdebug`. Or you can use `adb` if you wish, running it at user level in a separate window:

```
example adb -w /vmunix /dev/kmem
```

(`adb` won't allow you to set breakpoints in the kernel, but it will allow you to set and unset variables — you can change the value of `xxdebug`, or even reset a variable which has caused your driver to hang). *Remember that you're in the kernel and BE CAREFUL.*

Incidentally, `/dev/kmem` represents the kernel *virtual* address space, which is why it's used here. `adb -k /vmunix /dev/mem`, in contrast, generates a view of the *physical* address space, because `/dev/mem` represents the physical memory. This latter command is useful for examining core files.

Good places to put `printf()` statements include:

- driver routine entry points
- before critical subroutine calls
- upon reading status information from the device
- before writing of commands or data to the device
- at intermediate points in complex routines
- at routine exit points

Note again that you don't have to restrict yourself to a single `xxdebug` variable, or to binary tests that check to see if a variable is on or off. You can use as many variables, and as many values for each variable, as necessary to reflect the functional divisions most appropriate to your driver. It might even be useful to get truly esoteric, and send certain trace statements directly to the user tty (by calling `uprintf()`) while the rest use `printf()` and go to the console.

Event-Triggered Printing

In the above discussion, the `xxdebug` variable was initialized by the compiler, and toggled with a debugger. However, it's just as easy to have the driver routines themselves set a trigger variable under pre-chosen conditions.

For example, if you wanted to enable tracing after a given *condition* had occurred, you could declare `xxdebug`, just as was shown above, but define `XXDPRINT` somewhat differently:

```
#ifdef XXDEBUG
int xxdebug = 0;
#define XXDPRINT(v,msg,a1,a2) \
    if (xxdebug > (v)) printf(msg,a1,a2);
#endif
```

and then, in the code that checks for the condition:

```
#ifdef XXDEBUG
if (condition) xxdebug = 1;
#endif
```

Then to call XXDPRINT:

```
#ifdef XXDEBUG
XXDPRINT(0,"driver name...\n",a,b);
#endif
```

One major disadvantage of using the kernel `printf()` is that its output doesn't go through a device driver, and thus can't be paused with Control-S or redirected to a file. It's possible, then, that `printf()` will overwhelm you with output. There are a number of things that you can do if you run into this problem:

- If you haven't used multivalued print-control variables, then do so. This gives you more control than you have with simple on/off print control, and will allow you to reduce the amount to trace noise.
- You can use a debugger to set the global variable `noprintf`. This will keep `printf()`'s output from being sent to the console, but that output will still go to a buffer where kernel error messages are kept before being transferred to `/var/adm/messages`. You can examine the message buffer at your leisure, in one of two different ways:
  - From a user window, you can use `dmesg`.
  - From `kadb` (or `adb` on `/dev/kmem`) you can type `msgbuf+10/s`.
- It's also possible to reconfigure your system so that it uses a hardcopy terminal as its console over a RS-232 line. Then, you won't lose any of the `printf()` output.
- Best of all, you can get another machine and connect it to your machine over a RS-232 line. Having done so, use `tip` to open a window on the second machine *as the console of the test machine*. You can then use `tip`'s record feature (see the `tip` man page) to make a record of all the stuff that `printf()` is sending to the test machine's console.

Asynchronous Tracing

As mentioned above, there are occasions when timing problems forbid the use of the printf statement. In these cases, it's a good idea to give up any attachment that you might have to `printf()` statements and use `kadb`.

Or, if you prefer, it's possible to deal with timing problems by using `kadb` to patch the `noprintf` variable, and then to check the message buffer to see what's going on. Doing so:

- allows you to continue using the debugging code that you installed before encountering the timing problem, and

- presents you with a sequential list of the events in your driver, a list spelled out in English phrases and including interrupt-level events.

Or, you can simply use `kadb` for everything.

kadb — A Kernel Debugger

NOTE `kadb` *does not work with versions of the kernel earlier than 3.2.*

`kadb` is an interactive debugger similar in operation to `adb`. `kadb` differs in several key respects from `adb`. It runs as a standalone program under the PROM monitor, rather than as a user process in user address space. And it allows you to set breakpoints and single step in the kernel!

Thus, running a kernel under `kadb` is significantly different than running it under `adb -k`. The `k` option to `adb` merely makes it simulate the kernel memory mappings while `kadb` actually runs in the kernel address space. And unlike `adb`, which runs at user level and as a separate process from the process being debugged, `kadb` runs in system space as a *coprocess*. It shares not only the kernel address space but its CPU supervisor mode as well.

`kadb`, for all intents and purposes, is a version of `adb`. It has the same command syntax and almost the same command set. Thus, you can see the `kadb` and `adb` manual pages, as well as *Debugging Tools for the Sun Workstation*, for more details on its use. Note, however, the following points of special interest to driver developers:

- All interrupts are disabled while interacting with `kadb` (except non-maskable interrupts). Thus, when using `kadb` to examine memory, the kernel remains stable. However, while single stepped instructions are being executed, the actual standing priority of the kernel is temporarily restored, and interrupts can get dispatched, run and return. You won't notice unless you have a break point set in the interrupt routine, which works just fine.
- `kadb` is installed so that, when a program is being run under it, an abort sequence (L1-A) will transfer control not to the PROM monitor but to `kadb` itself. Once in `kadb`, you can abort again and be transferred to the monitor. The transfer is direct and immediate, so you can use the monitor to examine control spaces (e.g. page and segment maps) which are not accessible from `kadb`. The monitor `c` command will return you to `kadb`.
- `kadb` runs in the same virtual memory space as the kernel itself, and with the CPU in supervisor mode. This means that `kadb` uses the kernel memory maps when calculating virtual addresses, and that it can directly examine kernel structures. This is in contrast to the situation with `adb -k`, where software copies of the page table entries are used to map virtual addresses to physical pages.
- `kadb`'s memory view is almost the same as that resulting from `adb /vmunix /dev/kmem`. In other ways, however, `kadb` is much different. To give just one example: on Sun-3 and Sun-3x machines, where users and supervisors share the virtual address space, `kadb` allows the user to examine the user virtual address space (this is *not* true with `adb -k`).

- Finally, be aware that `kadb` — as a consequence of the way that `adb` works — always does 32-bit memory reads. Even if you tell `kadb` to read a byte it will read a long. This leads to a lack of control that can cause problems when reading device registers. (This problem does not exist on the Sun386i. On the Sun386i, when `kadb` is told to read a byte, it does. Within `kadb`, the `B` command is used to read a single byte and the `v` command to write one).

5.4. Device Driver Error Handling

There are various types of errors: "expected" errors like those generated by *xx*`probe()` routines, transient errors in operations that can reasonably be retried, fatal errors that require controlled shutdowns, and others. The kinds of errors that you will face depends upon the kinds of drivers that you write and the peculiarities of your devices; few generalizations can usefully be made.

To further complicate matters, the detection and treatment of errors varies greatly from device to device. You should begin by carefully reading your device specification manual to determine the error indications that can arise and the responses that should be made when they do. Most devices have at least an error bit in the control/status register, and usually more detailed error information is available. Ideally, you should understand all potential errors, avoid those that you can and recover from the rest. This ideal isn't always achievable, but try not to leave any obvious holes. *If you do nothing else, check for device errors and use the kernel* `printf()` *function to report them to the system console.*

There are various error reporting and management mechanisms available to the driver developer. Most of them have already been mentioned as they've become relevant; here they are collected and summarized:

Error Recovery

It's difficult to generalize about error-recovery mechanisms, for they are largely device specific. It's worth noting, however, that:

- Some errors are worth retrying and some aren't; the matter is entirely device specific.
- Error-recovery routines should be able to run at the interrupt level. This is because errors can occur either synchronously or asynchronously with respect to the dispatch of device commands, and, therefore, recovery routines must be callable from interrupt routines.
- If you do implement error recovery logic, you must do so consistently. The data structure that contains retry-status information must be global, and must be protected by critical sections. Error-recovery routines, like interrupt routines in general, must take special pains to protect data-structure integrity; indeed, they must *restore* such integrity upon encountering errors they can't recover from.

Error Returns

There are three mechanisms by which driver routines can report errors up to their calling routines. The first, of course, is by way of the values that the driver routines return to their callers. The second, and most important, is the error-reporting mechanism based upon the buffer-header. *This is the only mechanism that can be used when returning errors from* `xx`*`strategy()`*, `xx`*`start()`*, and `xx`*`intr()`*. (See the discussion of `xx`*`intr()`* error reporting in the *Summary of Device Driver Routines* chapter. Finally, it is possible to directly set the global error register, `u.u_error`, from routines in the top half of the driver.

Error Signals

It is sometimes desirable to have a driver send a software interrupt to the process or processes. It's possible, for example, that a device will fail in an unrecoverable fashion — in this case it's perhaps a good idea to signal the user processes, rather than merely returning an extraordinary error code. It's also possible (though rare) for a driver to encounter serious errors from which it can recover by restarting the device — user processes may also need to be notified in this case. The kernel `psignal()` and `gsignal()` routines can signal either a single process or all the processes in a given process group.

Error Logging

When you use the kernel `printf()` statement to report errors to the console, those errors are also placed into a system error-message buffer accessible to the `dmesg` daemon. `dmesg` can be, and typically is, run every 30 minutes by the `crontab` daemon, for the purpose of appending the messages in the buffer to `/var/adm/messages`. Note that the message buffer is small, and that if a lot of entries are being written into it, some of them will get lost before being transferred into `/var/adm/messages`.

Kernel Panics

The most unequivocal way of dealing with an error is to panic when you get it. The `panic()` routine is provided to help you do so in a somewhat controlled fashion — `panic()` is called only on unresolvable fatal errors. It prints "panic: mesg" on the console, and then reboots. (Or, if you're running under the debugger, it transfers control to `kadb`). `panic()` also keeps track of whether it's already been called, and avoids attempts to sync the disks (by flushing all pending write buffers) if it has, since this can lead to recursive panics.

The final production version of a driver should call `panic()` only when "impossible" situations are encountered; lesser errors should be recovered from. During debugging, though, `panic()` can be used to implement a passable assert mechanism.

```
#ifdef XXDEBUG
if (inconsistent condition)
    panic("Assertion Failed: ...");
#endif
```

(It's possible to write a fancier assert mechanism, for example by having an ASSERT macro which calls an `assert()` routine which prints error context information and then calls `panic()`, but this minimal hack will perhaps do).

Finally, note that in cases where it's *very* important to halt the system *immediately* after detecting an inconsistent condition, `kadb` can be used. The driver code can test for the inconsistent condition, and then set a debugging variable:

```
if (inconsistent condition)
    junk = 1;
```

`kadb` can then be used to set a breakpoint at the machine instruction generated from the assignment to `junk`.

5.5. System Upgrades

System upgrades generally have minimal effects on user-written device drivers. The changes that are necessary are rare and release specific.

Some changes must be made if user-written drivers are to work with new release software. In Release 2.0, for example, there was a minor change in one of the bus-interface structures. There wasn't much involved in adapting user-written drivers, but it had to be done.

In other cases, changes are optional. When VMEbus machines were introduced, for example, drivers had to be adapted to run on them; however, it was possible to upgrade Multibus machines without rewriting user-written drivers.

In any case, any release upgrades that imply changes — either optional or mandatory — to user-written device drivers will be documented in the *System Summary and Change Notes* for the release in question.

5.6. Loadable Drivers

The Sun386i supports loadable drivers. This feature allows you to add a device driver to a running system without rebooting the system or rebuilding the kernel. The loadable drivers feature reduces time spent on driver development, and makes it easier for users to install drivers from other vendors.

This section explains how to convert a non-loadable driver to be a loadable driver.

Conversion of a non-loadable driver to a loadable driver requires an initialization or "wrapper" module to be written. The module `zzinit.c` below is an example of a wrapper module that contains the same kind of information ordinarily provided by a config file and by the linker. Almost all wrappers are identical to the example below. Usually, only the actual structure initialization values are different.

The following module is a typical example of an initialization routine for a driver named `zz` that has one controller and one device on that controller.

```
#include <sys/types.h>
#include <sys/conf.h>
#include <sys/buf.h>
#include <sys/param.h>
#include <sys/errno.h>
#include <sundev/mbvar.h>
#include <sun/autoconf.h>
```

```
#include <sun/vddrv.h>

extern zzopen(), nulldev(), zzstrategy(), zzdump();
extern zzsize(), zzread(), zzwrite(), zzioctl();
extern zzint(), nodev(), seltrue();

extern struct mb_driver zzcdriver;    /* defined in driver */

/*
 * Driver block device entry points (normally in <sun/conf.c>)
 */
struct bdevsw zzbdev = {
    zzopen, nulldev, zzstrategy, zzdump, zzsize, 0
};

/*
 * Driver character device entry points (normally in <sun/conf.c>)
 */
.
struct cdevsw zzcdev = {
    zzopen, nulldev, zzread, zzwrite, zzioctl, nodev,
    nulldev, seltrue, 0
};

/*
 * Controller structure (normally in ioconf.c) (see <sundev/mbvar.h>)
 */
struct mb_ctlr zzcctlr[] = {
    &zzcdriver, 0, 0, (caddr_t) 0x00001000, 2, 6,
    SP_ATMEM, 0
};

/*
 * Device structure (normally in ioconf.c) (see <sundev/mbvar.h>)
 */
struct mb_device zzcdevice[] = {
    &zzcdriver, 0, 0, 0, (caddr_t) 0x00000000, 0, 0, 0x0,
    0, 0x0
};

/*
 * The following structure is defined in <sun/vddrv.h>
 *
 * If the number of controllers is 0, then the address of the
 * controller structure array must be NULL. Similarly, if the number
 * of devices is 0, then the address of the device structure array
 * must be NULL. The bdevsw or cdevsw entries can be NULL if there
 * is no block or character device for the driver.
 */
struct vdldrv vd = {
    VDMAGIC_DRV,      /* Type of module. This one is a driver. */
    "zzdrv",          /* Name of the module. */
    zzcctlr,          /* Address of the mb_ctlr structure array */
```

```
    &zzcdriver,         /* Address of the mb_driver structure */
    zzcdevice,          /* Address of the mb_device structure array */
    1,                  /* Number of controllers */
    1,                  /* Number of devices */
    &zzbdev,            /* Address of the bdevsw entry */
    &zzcdev,            /* Address of the cdevsw entry */
    0,                  /* Block device number. 0 means let system choose. */
    0,                  /* Char. device number. 0 means let system choose.*/
};

/*
 * This is the driver entry point routine. The name of the default entry point
 * is xxxinit. It can be changed by using the "-entry" command to modload.
 *
 * inputs: function code - VDLOAD, VDUNLOAD, or VDSTAT.
 *      pointer to kernel vddrv structure for this module.
 *      pointer to appropriate vdioctl structure for this function.
 *      pointer to vdstat structure (for VDSTAT only)
 *
 * return: 0 for success. VDLOAD function must set vdp->vdd_vdtab.
 *      non-zero error code (from errno.h) if error.
 *
 */

xxxinit(function_code, vdp, vdi, vds)
    unsigned int function_code;
    struct vddrv *vdp;
    addr_t vdi;
    struct vdstat *vds;
{
    switch (function_code) {
        case VDLOAD:
            vdp->vdd_vdtab = (struct vdlinkage *)&vd;
            return (0);
        case VDUNLOAD:
            return (unload(vdp, vdi));
        case VDSTAT:
            return (0);
        default:
            return (EIO);
    }
}

static unload(vdp, vdi)
        struct vddrv *vdp;
        struct vdioctl_unload *vdi;
{
    extern struct buf zztab;

    struct buf *dp;

    dp = &zztab;
    if (dp->b_actf) {
```

```
        return(-1); /* The driver still has an active request. */
    }

    /* The driver can do any device shutdown stuff that it needs to do */

    return(0);
}
```

Your driver routines can be placed in the wrapper module if you like. If your driver is big, it is more appropriate to break it into several modules.

If you decide to place your driver in the wrapper module, then the driver can be compiled with the following command line:

```
example# cc -c -O -Dsun386 -Di386 -DTTYSOFTCAR -DWEITEK \
    -DVDDRV -DCRYPT -DVPIX -DIPCSHMEM -DIPCSEMAPHORE \
    -DIPCMESSAGE -DLOFS -DNFSSERVER -DNFSCLIENT -DUFS \
    -DINET -DSUN386 -DKERNEL -Umc68000 -Di386bug zzinit.c
```

However, if the driver consists of more than one module, then you must use the link editor, `ld`(1), with the `-r` option to preserve relocation information. For example you might type:

```
example# cc -c -O -Dsun386 -Di386 -DTTYSOFTCAR -DWEITEK \
    -DVDDRV -DCRYPT -DVPIX -DIPCSHMEM -DIPCSEMAPHORE \
    -DIPCMESSAGE -DLOFS -DNFSSERVER -DNFSCLIENT -DUFS \
    -DINET -DSUN386 -DKERNEL -Umc68000 -Di386bug zzinit.c

example# cc -c -O -Dsun386 -Di386 -DTTYSOFTCAR -DWEITEK \
    -DVDDRV -DCRYPT -DVPIX -DIPCSHMEM -DIPCSEMAPHORE \
    -DIPCMESSAGE -DLOFS -DNFSSERVER -DNFSCLIENT -DUFS \
    -DINET -DSUN386 -DKERNEL -Umc68000 -Di386bug zz1.c

example# cc -c -O -Dsun386 -Di386 -DTTYSOFTCAR -DWEITEK \
    -DVDDRV -DCRYPT -DVPIX -DIPCSHMEM -DIPCSEMAPHORE \
    -DIPCMESSAGE -DLOFS -DNFSSERVER -DNFSCLIENT -DUFS \
    -DINET -DSUN386 -DKERNEL -Umc68000 -Di386bug zz2.c

example# ld -r -o zz.o zzinit.o zz1.o zz2.o
```

Thus the object module can be created either by the `cc`(1) command, when the driver resides in one module, or by the `ld`(1) command, when the driver resides in several modules.

In either case the resulting object file (`zzinit.o` or `zz.o`) is a normal `COFF` file and can then be used with the `modload` command.[1] The driver is just like

[1] "COFF" = Common Object File Format, a UNIX object-file standard to which Sun386i assembler and link-editor output files (normally `a.out`) comply. See `coff`(5).

any other program, except its text segment starts somewhere in the range `0xFD000000` to `0xFE000000`.

PROM User's Manual Addenda and Errata

Sun Microsystems, Inc. • 2550 Garcia Avenue • Mountain View, CA 94043 • 415-960-1300

Part No: 800-1736-10
Revision A of 24 April 1989

Contents

Tables

Figures

PROM User's Manual Addenda and Errata

The *PROM User's Manual* describes PROM monitor commands, self-tests and extended tests for Sun-2 through Sun-386i systems. This text adds to or modifies the information found there, for Sun-3 systems with the new, 2.8 version of the Boot PROM, and all Sun-3/400 series, the Sun-3/80 and SPARCsystem 330s.

New Monitor Command Feature

In the past, all numerical PROM monitor commands were entered in hexadecimal. If you have PROM version 2.8, or a Sun-3/400 series system, a Sun-3/80 or SPARCsystem 330, you may now enter decimal or ASCII values after the PROM monitor prompt (>). This feature is particularly useful when using the monitor `q` command to program the EEPROM, which sometimes requires that you convert letters and decimal numbers to hexadecimal values before you enter them.

To enter a decimal value after a PROM monitor command, simply precede the value with the "%" character:

```
>q 050 %20
```

To enter an ASCII character, simply precede it with the "@" character:

```
>q 022 @i
>q 023 @e
```

If the value you enter is not preceded by a % or @ character, the monitor program treats that the value as hexadecimal.

EEPROM Security Feature

Chapter 10 of the *PROM User's Manual* describes the Sun-3 and Sun-4 EEPROM. There is now a Security Mode Select Feature, located at EEPROM address 0x492. This feature provides a *non-secure* mode that permits the use of all PROM monitor commands.

It also provides a *command secure* mode that permits the use of PROM monitor commands (other than **c**, for continue, or **b**, for boot, without parameters) only when a password is entered. In the command secure mode, you may operate your workstation normally, including powering down, booting, terminating with the `L1-A` command, and re-booting. You may not, however, perform any unusual operations, such as booting a non-standard kernel, running diagnostics,

or changing EEPROM or CPU board memory contents, without entering a password.

Finally, this feature provides a *fully secure* mode that does not permit use of the PROM monitor (other than the **c** command with no parameters) without entering a password. To power-up, re-boot, or perform any other PROM monitor operation, you must supply a password. This mode allows you to control access to the workstation by turning it off. The workstation does not automatically boot on power-up.

CAUTION **In fully secure mode, even a default boot cannot be completed unless the password is entered. Once the SunOS is halted, you cannot restore it until you enter the correct password after the prompt. If the password is unknown, the system CPU board must be serviced as a failed board.**

The Password

If you should attempt to enter a PROM monitor command such as q, for example, on a system that is set for one of the secure modes, your interaction might look like this:

```
>q 020                    (you want to look at EEPROM offset address 020)
>Mon Pass:                (you enter the correct password)
>EEPROM 020: 00 ?         (the contents of location 020 are shown)
```

Or, if you enter an incorrect password, your interaction might look like this:

```
>q 020
>Mon Pass: (you enter the wrong password)
>Invalid
           (there is a slight delay)
>
```

You may now try again or enter an unprotected command.

To install or change a password, the system must be in non-secure mode, or you must know the existing password for a secure system. You then use the PROM monitor **q** command to enter the password in EEPROM offset location 490 for Sun-3 and Sun-4 (SPARC) systems, or 160 for the Sun386i.

NOTE *When you attempt to change the values stored in the EEPROM monitor password locations, you will be prompted with this message:*

```
Modifiying security location(s). Are you sure?(y/N)
```

If you enter **y** *for yes, the change you entered is written to the location shown. If you enter* **n** *for no, nothing is written to EEPROM, and the contents of the next location are displayed.*

To enter a monitor password, use the @ character, described at the beginning of this document, in order to enter the letters that make up the password. If you do not use the @ character, you must enter the hexadecimal equivalent of the letters. Following is an example of the way you would enter a password called *mypasswd* on a Sun-3 or Sun-4 system. You enter one letter per location, followed with : Return. To exit the command, you may enter any non-

hexadecimal character, such as period, as shown.

```
>q 493
> EEPROM 493: 00? @m
Modifiying security location(s). Are you sure?(y/N) Yes
> EEPROM 494: 00? @y
Modifiying security location(s). Are you sure?(y/N) Yes
> EEPROM 495: 00? @p
Modifiying security location(s). Are you sure?(y/N) Yes
> EEPROM 496: 00? @a
Modifiying security location(s). Are you sure?(y/N) Yes
> EEPROM 497: 00? @s
Modifiying security location(s). Are you sure?(y/N) Yes
> EEPROM 498: 00? @s
Modifiying security location(s). Are you sure?(y/N) Yes
> EEPROM 499: 00? @w
Modifiying security location(s). Are you sure?(y/N) Yes
> EEPROM 49a: 00? @d
Modifiying security location(s). Are you sure?(y/N) Yes
> EEPROM 49b: 00? .
>
```

NOTE *If a password was already stored in locations 493-49a, hexadecimal values would appear in place of the zeroes in the example above.*

The password you enter must either fill the eight bytes (locations 493-49a) with a character or a zero.

Note that changes to the security mode and password do not take effect until the PROM monitor mode is re-entered.

It is recommended that the password is changed before the security mode is changed. For more information on using the EEPROM **q** command, refer to the *PROM User's Manual*, Sun PN 800-1736, or the `Monitor(8S)` section of the *SunOS Reference Manual.*

Changing Security Modes

The EEPROM offset location 492 (or 162 for a Sun386i) contains a value that determines the security mode. The table below shows the interpretation of values found in that location. The "0x" denotes a hexadecimal value.

0x1	command secure
0x5e	fully secure
all other values	non-secure

Because the PROM monitor password is stored as text, it is recommended that the `chmod` and `/etc/chown` commands be used so that the `/dev/eeprom` device file may be accessed by the super-user. To accomplish this, enter:

```
%su
Password: enter your super-user password
# cd /dev
# /etc/chown root eeprom
# chmod 600 eeprom
```

EEPROM Layout for PROM Security

Here is a table that shows the EEPROM offset locations for the Sun-3, Sun-4 and Sun386i systems.

Sun386i Offset	*Sun-3,-4 Offset*	*Field*	*Function*
0x160-1	0x490-1	bad_login	The bad login counter stores the number of invalid password attempts. The maximum value is 65535 and the counter does not roll over to 0.
0x162	0x492	secure	The values 1 and 0x5e correspond to command secure and fully secure, respectively. Any other value is non-secure.
0x163-a	0x493-a	password	If the password is shorter than 8 bytes, the password string is fenced with a null character.

3-D Logo EEPROM Parameter

If your system has a CG6 board, you may set EEPROM location 0x020 to 0x06 so that, upon power-up, the Sun logo appears to be three-dimensional. Refer to the *PROM User's Manual* monitor **q** command description for information on changing EEPROM values.

Sun-3 Extended Tests

Chapter 9 of the *PROM User's Manual* describes Sun-3 extended tests. For workstations with the boot PROM version 2.8, many of these tests are unavailable, and the user interface has changed. The new PROM version tests only the devices needed to boot the operating system. Therefore, when you invoke the extended tests from the monitor prompt,

```
>x
```

the menu of extended tests look something like this:

```
Extended Test Menu: (Enter 'q' to exit)

Cmd - Test

ie  -  Intel Ethernet Test
mk  -  Mouse/Keyboard Ports Test
rs  -  Serial Ports Test
```

NOTE *For CPU boards with the AMD AM7990 (Lance) Ethernet chip, the first choice will be*

```
ae  -  AMD Ethernet Test
```

New Boot Path Extended Tests

In order to invoke the disk and tape bootpath tests when a version 2.8 PROM is installed on a Sun-3 CPU board, you must enter an asterisk after the boot command, from the monitor prompt:

```
>b*device ()
```

The extended test appropriate for the named *device* will then be executed, and any error messages displayed on the screen. *device* could be one of the following:

sd	*for SCSI disk*
st	*for SCSI tape*
xd	*for Xylogics 7053 Disk Controller*
xt	*for Xylogics tape*
xy	*for Xylogics 450/451 Disk Controller*

Sun-3/400 Series and Sun-3/80 PROM

For Sun-3/400 series and Sun-3/80 users, the text that follows is intended to replace Chapters 7 and 8 in the *PROM User's Manual*, Sun PN 800-1736-10. These chapters cover changes to PROM monitor commands and self-tests associated with the Sun-3/400 product. In addition, the information in Chapter 13 of the PROM manual, "Sun-4 Extended Test System", applies to the Sun-3/400 series workstation. If you have a Sun-3/400 series system, Chapter 9, "Sun-3 Extended Test System" does not apply.

The Power-Up Test Sequence

In order to perform the power-up tests, two assumptions must be met. The CPU (the non-PMMU portion of the MC68030 for the Sun-3/400 series system) must be functional and the ability to fetch instructions from the Boot PROM must be intact.

Powering up a Sun-3 workstation resets the CPU to *boot* state, which means that all instruction fetches are forced to the Boot PROMs. Execution of the minimum-confidence power-up tests begin immediately. These tests do not employ any memory until memory has been successfully checked.

The objective of the power-up test sequence is to determine whether or not the CPU board logic and main memory are functional. Following the successful completion of the power-up tests and subsequent workstation initialization, an attempt is made to boot the SunOS operating system, an EEPROM-specified program, or an operator-specified stand-alone program.

If hardware problems are detected during the verification process, the PROM monitor prompt should appear.

For Sun-3/400 series systems, if the Diagnostic Switch at the rear of the system is in NORM position, you will not be able to interact with the self-tests. You may read the LEDs on the CPU board edge (described later in this chapter) to determine whether or not a test is failing, and you will see a rotating diagonal symbol after the

```
Testing _ megabytes of memory...
```

message on the console during the memory tests. The quantity of memory checked during a power-up with the diagnostic switch on NORM is dependent on EEPROM programming. The EEPROM chapter in the *PROM User's Manual* and the `eeprom` command description in the *SunOS Reference Manual* explain how to set the parameter that controls the quantity of memory tested.

For the Sun-3/80, which has no diagnostic switch, an EEPROM parameter must be set in order to execute a diagnostic boot-up and view the self-test display on a terminal. Refer to the "Diagnostic Power-Up" section for more information on this setting.

If the workstation contains a large amount of main memory, self-tests may last as long as eight minutes. The table below compares the self-test duration of various Sun-3 systems.

Table 1 *Self-Test Execution Time Comparisons*

System Type	*Clock Rate in MHz*	*Memory Size in Megabytes*	*Self-test Duration in Minutes*
Sun-3/160	16 MHz	4 Meg	.5 Min
Sun-3/75	16 MHz	8 Meg	.75 Min
Sun-3/60	20 MHz	24 Meg	1.75 Min
Sun-3/80	16 MHz		11 Sec (est.)
Sun-3/260	25MHz	64 Meg	2.5 Min
Sun-3/400 series	33MHz	128 Meg	2.75 Min (est.)
SPARC system		128 Meg	8.0 Min (est.)

If the Diagnostic Switch is in the NORM position, (or the Sun-3/80 EEPROM parameter set), the power-up tests execute successfully and, if you do *not* terminate the default boot sequence, an attempt is made to down-load the SunOS operating system.

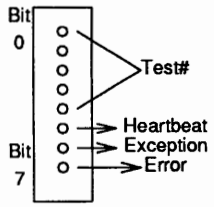
A display something like this appears on the *workstation's* screen to indicate that power-up tests are successful:

```
Selftest Completed Successfully.

        Sun Workstation, Model Sun-3/___ Series
        Type-X keyboard
        ROM Rev __, _ MB memory installed, Serial #_______
        Ethernet address __:__:__:__:__:__

Testing __ megabytes of memory...Completed.

Auto-boot in progress...
```

Diagnostic LEDs

One requirement of Sun-3 firmware is to assign a unique test number to most of the power-up tests and display that number in bits *zero* through *four* of the diagnostic LEDs as the test is running. Given that there are fewer test numbers than there are power-up tests, power-up tests that check the same part of hardware share a test number.

If one of these power-up tests should fail, bit *seven* of the diagnostic LEDs also lights up. Bit seven serves as an indicator that there is a hardware problem. The LED display permits the service person not only to conclude whether or not there is a problem, but to determine which type of power-up test is failing.



For the sake of completeness, LED *five* is the *heart beat* LED. After the power-up tests have been completed, but prior to invocation of the SunOS operating system or an EEPROM-specified program, LED 5 will blink on and off to indicate that the IU is actually executing instructions. LED *six* indicates whether or not the failure is an exception (i.e. unexpected trap or unexpected interrupt). The diagram to the left shows LED designations for Sun-3/400 series systems.

Sun-3/80 LED

Sun-3/80 systems have one green LED that blinks while a test is in progress, and turns OFF when there is an error condition. When the light stays ON, everything is functioning satisfactorily:

LED status	Visual	Condition
Blinking	⊖	Testing
OFF	○	Error
ON (Green)	●	Ok

The Boot Sequence

Following the initialization of the workstation, the default boot sequence is executed. There are two issues that must be considered here. One has to do with *what is to be down-loaded* while the other has to do with *where it is to be loaded from.*

Assuming *no* operator intervention, the position of the Diagnostic Switch (on all Sun-3's except the Sun-3/80) will determine *what* is to be booted. If the Diagnostic Switch is in the NORM position, the SunOS operating system is booted. Otherwise, if the Diagnostic Switch is in the DIAG position, the EEPROM-specified program is booted. Be aware that the PROM monitor program is invoked if no EEPROM-specified program is available.

If you are in the PROM monitor mode, you may specify *what* is to be booted and *where* it is to be booted from. See command **b** (boot) in the chapter titled *Sun-3 PROM Monitor Commands* for a description of how to boot user-specified programs from user-specified devices.

If you are interacting with a Sun-3 workstation through the console to reach the monitor program, you may enter `L1-a` (or [Break] on a terminal) IMMEDIATELY after the

```
Testing __ megabytes of memory...Completed.
```

message.

CAUTION **Do not use `L1-A` procedure once the automatic boot has started; the file systems may be damaged if disks are powered-on and the operating system has started to run.**

If the operating system is already booted, use the procedures described in Chapter 3 of the *PROM User's Manual* to start the monitor.

Once you are in the monitor mode (symbolized by the > prompt), you should do the following:

```
> g 0
panic: zero
Syncing disks... done
```

Press L1-a or [Break] again when the message above finishes.

Next you may see this message:

```
dumping to dev somevalue, offset somevalue

Abort at somevalue
>
```

The firmware also determines from what boot device the program will be loaded. If the Diagnostic Switch is in the NORM position and the content of EEPROM location 0x18 is equal to 0x12 (an arbitrarily chosen value), Sun-3 firmware will attempt to boot the SunOS operating system from the boot path specified in the EEPROM, beginning at location 0x19. If the boot path is missing or contains an error, the monitor program is invoked.

If the Diagnostic Switch is in the NORM position and the content of EEPROM location 0x18 is *not* equal to 0x12, Sun-3 firmware attempts to boot the SunOS operating system using the following boot device polling sequence:

1. Xylogics Disk (450-451).
2. SCSI Disk.
3. Ethernet.

If the Diagnostic Switch is in the DIAG position, the firmware assumes that both the path name of the file containing the to-be-loaded program and the boot device are specified in the EEPROM, beginning at EEPROM location 0x22. If either the file name or the boot device is not present or is in error, the monitor is invoked. If the Diagnostic Switch is in the DIAG position, you may connect a terminal to Serial A or B and interact with the self-tests and the Extended Test System, if required.

Diagnostic Power-Up

NOTE *There is no diagnostic switch on a Sun-3/80; an EEPROM setting is required to allow a diagnostic boot-up that displays test names and errors through Serial Port A. Enter the PROM monitor mode and use the* **q** *command to write* **12** *to location 0x70b. Any other value causes tests to run without error reporting to the terminal. Refer to "Displaying and Modifying Memory" for information on use of the* q *command.*

If the Diagnostic Switch exists and is on DIAG, the self-test is executed as it is when the switch is "off" or on NORM, except that all of memory is tested. In addition, self-test status information is directed only to serial ports A and B, using the MMU (Memory Management Unit) bypass until all hardware required for the Video Monitor has been successfully tested.

Any hardware failures during the selftests will invoke scope loops to permit troubleshooting the failure. An RS-232 terminal with its characteristics set to 9600 Baud, 8 data bits, 1 stop bit and no parity should be connected to Serial Port A of the CPU board if you wish to view self-test status and interact with the system. If you use Serial Port B you must set the terminal baud rate to 1200.

Limited interaction with the self-test program is possible in this mode and the following characters will invoke the following actions when entered from the terminal connected to serial port A during self test. Each of these commands is documented below.

ESCAPE Key — Sun-3/400 Series and Sun-3/80

Pressing this key any time during the self-test sequence prior to the display of the `Selftest Passed` message causes the self-test sequence to abort and a warning message is displayed. The necessary memory sizing and other setup is done, and then, for the Sun-3/400 series, the program displays the PROM Monitor menu. For the Sun-3/80, this message is displayed:

```
<Warning: selftests aborted by user>

<Initializing Main Memory 0x0000000C Megabytes Initialized>

Type a Character within 10 seconds to enter the Menu Tests...(e for e
mode)
EEPROM: Using RS232 A port.
Selftest Completed.
```

Note that this abort can be done if the test is running normally (without any errors) or if the test is presently looping on any error encountered.

Control-q Key — Sun-3/400 Series Only

Holding down the Control key while pressing the `q` key any time during the execution of a particular Sun-3/400 Series self-test causes that one self-test to stop and execution of the next self-test to begin. For example, if the Control-q sequence was entered during the `MEMORY WRITE/WRITE/READ TEST`, that test would terminate and the next test (`MEMORY ADDRESS TEST`) would begin. Note that this action can be performed if the test is running normally (without any errors) or if the test is presently looping on any error encountered.

Control-l Key — Sun-3/400 Series and Sun-3/80

Holding down the Control key while pressing the `l` key any time during a self-test causes that particular self-test to terminate and control to be transferred to the `TEST LOOP MENU`. From within this menu any particular self-test can be executed and looped directly, without executing the self-tests that normally come before it. Once a test is invoked from the `TEST LOOP MENU`, a test control flag is set so that once that test has finished it will go back to the beginning of that one particular test and start it all over again. In order to proceed to the next self-test after a particular self-test has been invoked from this menu, enter the Control-q sequence or the Esc key to go to next test or to the Monitor (as described above). See the section on the Test Loop Menu for more details.

More Interactive Self-Test Commands

b Press the **b** (a mnemonic for *burn-in*) key, prior to the display of the `...Completed` or `Selftest Finished` message, to execute the power-up test sequence indefinitely. This option is useful during the manufacturing burn-in stage.

For the Sun-3/80, when the last selftest is finished the message tests`Automatically`continuing will be displayed and the self-test will be restarted again, using the System Enable Register read test as the first test. The SCC and terminal I/O tests are bypassed in burn-in mode since operator interaction is required. This looping sequence will continue until an ESCAPE is entered, a reset is done, or the "*b*" key pressed again to turn burn-in mode off. The burn-in mode can be toggled by successive pressing of the "*b*" key. Note that this burn-in mode key can be processed during a test that is running normally (no errors), if the test is presently looping on an error encountered, or at the end of selftest when the "Selftest Finished" is displayed and an operator input is requested.

s Press the **s** key prior to the display of the `...Completed` message to *re-start* the power-up test sequence.

Space Bar
If one of the power-up tests fails, it will continue to re-execute forever unless interrupted. Press the [space bar] to terminate the failed test and execute the next power-up test.

Control-m
Holding down the [Control] key while pressing **m** causes this test to end and sets a flag that allows execution of only the quick memory tests.

Control-b
Holding down the [Control] key while entering **b** causes this test to end and sets a test control flag so that all the tests that require the Bus Error circuitry to work will be skipped. This test is also the equivalent of a "Control-m". It assumes that 8MB of ECC memory are present on the board, because the test is executed before memory has been sized. This test displays no error messages. The Bus Error dependent tests skipped are:

Bus Error Register Test
The Level 1 to Level 3 Interrupt tests
P4 Video RAM Board Tests.

A Successful Sun-3/80 Self-Test

For a Sun-3/80, After the selftest have completed the final status of the selftest sequence will be displayed, as shown by the following example:

```
        END OF SELFTEST # 0x00000005 (SELFTEST FAILED)
        #PASSED = 0x00000004, #FAILED = 0x00000001

<Initializing Main Memory 0x0000000C Megabytes Initialized>

Type a Character within 10 seconds to enter the Menu Tests...(e for echo mode)
```

Test Loop Menu — Sun-3/400 Series and Sun-3/80

The Test Loop Menu is a menu appears when you press Control-l (loop) key either during Sun-3/400 series or Sun-3/80 self-test execution. Test. The menu allows the operator to transfer self-test control directly to a specified self-test and loop that test continually. This option is intended to support debugging a specific failing test or section of hardware without running all of the previous self-tests.

Once in the Test Loop Menu the following display will be shown on the terminal:

```
   <<< TEST LOOP MENU >>>
(a)System Enable Register Test
(b)I/O Mapper RAM Write/Write/Read Test
(b)I/O Mapper RAM Address Test
(c)I/O Mapper RAM 3-Pattern Test
(t)ECC Memory Forced CE Test
(u)ECC Memory Forced UE Test
<<< Press ,Space for more, <Esc> to quit or select an option >>>
```

or, for the Sun-3/80

```
   <<< TEST LOOP MENU >>>
(a)System Enable Register Test
(b)I/O Mapper RAM Write/Write/Read Test
(b)I/O Mapper RAM Address Test
(c)I/O Mapper RAM 3-Pattern Test
(w)P4 Enable plane Write/Write Read Test
(x)P4 Color Plane Write/Write/Read Test
<<< Press ,Space for more, <Esc> to quit or select an option >>>
```

At this point you may either press the space bar to see more self-tests, or enter the key code for the test you want to execute in loop mode. Once a test is selected, that test is invoked and executed continually, regardless of whether the test passed or failed. From this point you can use any of the special control keys to continue the selftest if desired: control-q to continue the self-test in non-loop mode or Esc to abort all self-tests and go to the PROM Monitor.

Successful Diagnostic Boot Display

For a Sun-3/400 series or Sun-3/80 system, upon self-test completion a status message that looks something like this is displayed:

```
END OF SELFTEST #0x00000005 (SELFTEST FAILED)

#PASSED=0x00000004,#FAILED=0X00000001
```

NOTE *There is no diagnostic switch on a Sun-3/80; an EEPROM setting is required to allow a diagnostic boot-up. Enter the PROM monitor mode and use the* **q** *command to write* **12** *to location 0x70b. Refer to "Displaying and Modifying Memory" for information on use of the* q *command.*

During a diagnostic boot, after self-test has completed successfully, you will be prompted to enter the extended tests, as shown below.

```
Selftest passed.

Optional Menu Tests

Type a character within 10 seconds to enter Menu Tests...(e for echo mode)
```

If you press a terminal key during this time the Extended Test Menu is displayed on the terminal screen. See the "Sun-3 Extended Test Sequence" chapter in this document for details.

If you do not assert control of the system by pressing a terminal key within the 10 seconds delay time the Boot PROM program will next display the Sun logo and message on the console.

Remote Testing

If you press **e** on a *dumb terminal* keyboard, during the ten second period, all subsequent output will appear on *both* the console video monitor *and* a terminal attached to Serial Port A or B.

The purpose of this feature is to enable an individual at a remote site, using a terminal attached to a non-local machine by way of a telephone line with modems at each end, and the individual on-site to observe the system's behavior simultaneously. Once a system is in this mode, the local and non-local parties can communicate by employing the # command. Specifically, if the person at the remote site would like to send a message to the person on-site, he or she would simply type the # before typing the actual message. In order to terminate the message, the person at the remote site would enter a second #. At this point, the on-site person would be able to respond by prefixing and terminating their response with the #.

The # option will be very useful in the situation where an on-site customer has contacted a remote repair depot regarding a potential hardware problem. The repair person at the remote repair depot could initiate diagnostics on the non-local machine and allow both parties to observe the output. Beyond that, the repair person and the customer could communicate by way of the # command.

Diagnostic Self-test Sequence

The following text lists diagnostic mode self-tests for each Sun-3 system. If the diagnostic switch is enabled, the name of each test appears on the terminal until all self-tests are complete. The tests differ slightly according to the system architecture. The examples that follow represent each Sun-3 workstation.

Figure 1 *Sun-3/75, 3/140, 3/150, 3/160, and 3/110 Diagnostic Boot Sequence*

```
Boot PROM Selftest

  PROM Checksum Test
  DVMA Reg Test
  Context Reg Test
  Segment Map Wr/Rd Test
  Segment Map Address Test
  Page Map Test
  Memory Path Data Test
  NXM Bus Error Test
  Interrupt Test
  TOD Clock Interrupt Test
  MMU Access Bit Test
  MMU Access/Modify Bit Test
  MMU Invalid Page Test
  MMU Protected Page Test
  Parity Test
  Memory Size = 0x00000xxx Megabytes
  Memory Test (testing xxxxxxxx MBytes)

Selftest passed

Optional Menu Tests

Type character within 10 seconds to enter menu tests... (e for echo mode)
```

Figure 2 *Sun-3/50 and Sun-3/60 Diagnostic Boot Sequence*

```
Boot PROM Selftest

  PROM Checksum Test
  Context Reg Test
  Segment Map Wr/Rd Test
  Segment Map Address Test
  Page Map Test
  Memory Path Data Test
  NXM Bus Error Test
  Interrupt Test
  TOD Clock Interrupt Test
  MMU Access Bit Test
  MMU Access/Modify Bit Test
  MMU Invalid Page Test
  MMU Protected Page Test
  Parity Tests
  Memory Size = 0x00000xxx Megabytes
  Memory Test (testing xxxxxxxx MBytes)

Selftest passed

Optional Menu Tests

Type a character within 10 seconds to enter Menu Tests... (e for echo mode)
```

Figure 3 *Sun-3/260 and Sun-3/280 Diagnostic Boot Sequence*

```
Boot PROM Selftest

  PROM Checksum Test
  DVMA Reg Test
  Context Reg Test
  Segment Map Wr/Rd Test
  Segment Map Address Test
  Page Map Test
  Memory Path Data Test
  NXM Bus Error Test
  Interrupt Test
  TOD Clock Interrupt Test
  MMU Access Bit Test
  MMU Access/Modify Bit Test
  MMU Invalid Page Test
  MMU Protected Page Test
  ECC Error Tests
  Cache Data 3 Pat Test
  Cache Tags 3 Pat Test
  Memory Size = 0x00000xxx Megabytes
  Memory Test (testing xxxxxxxx MBytes)

Selftest passed

Optional Menu Tests

Type character in next 10 seconds to enter menu tests... (e for echo mode)
```

Figure 4 *Sun-3/400 Series Diagnostic Boot Sequence*

```
System_name Boot Prom Self-Test
(Hit key for character echo, space bar for next test, control-m, control-b)
System Enable Register Read Test (pass 0x00000001)
PROM Checksum Test
I/O Mapper RAM Write/Write/Read Test (pass 0x00000001)
I/O Mapper RAM Address Test (pass 0x00000001)
I/O Mapper RAM 3-Pattern Test (pass 0x00000001)
Bus Error Register Test (pass 0x00000001)
Level 1 Interrupt Test (pass 0x00000001)
Level 2 Interrupt Test (pass 0x00000001)
Level 3 Interrupt Test (pass 0x00000001)
TOD Clock Interrupt Test (pass 0x00000001)
<Sizing Main Memory>
<Memory Size = 0x00000010>
Memory Write/Write/Read Test (pass 0x00000001)
Memory Address Test (pass 0x00000001)
Memory 3-Pattern Test (pass 0x00000001)
Memory Read Byte Alignment Test (pass 0x00000001)
Memory Write Byte Alignment Test (pass 0x00000001)
Parity Memory No Error Test (pass 0x00000001)
Parity Memory Forced Error Test (pass 0x00000001)
ECC Memory No Error Test (pass 0x00000001)
ECC Memory Forced CE Test (pass 0x00000001)
ECC Memory Forced UE Test (pass 0x00000001)
Central Cache Tag RAM Write/Write/Read Test (pass 0x00000001)
Central Cache Tag RAM Address Test (pass 0x00000001)
Central Cache Tag RAM 3-Pattern Test (pass 0x00000001)
Central Cache Data RAM Write/Write/Read Test (pass 0x00000001)
Central Cache Data RAM Address Test (pass 0x00000001)
Central Cache Data RAM 3-Pattern Test (pass 0x00000001)
Central Cache Data RAM Read Byte Alignment Test (pass 0x00000001)
Central Cache Data RAM Write Byte Alignment Test (pass 0x00000001)
Central Cache Read Hit Test (pass 0x00000001)
Central Cache Invalid Read Miss Test (pass 0x00000001)
Central Cache Valid Read Miss Test (pass 0x00000001)
Central Cache Write Hit Test (pass 0x00000001)
Central Cache Write Miss, No Writeback Test (pass 0x00000001)
Cache Write Miss, Writeback Test (pass 0x00000001)
Central Cache Line Cross Invalid Read Miss Test (pass 0x00000001)
Central Cache Line Cross Write Miss Writeback Test (pass 0x00000001)
Central Cache Writeback Timeout Test (pass 0x00000001)
Block Copy (Source=Cache Miss,Dest=Cache Miss) Test (pass 0x00000001)
Block Copy (Source=Cache Miss,Dest=Cache Hit) Test (pass 0x00000001)
Block Copy (Source=Cache Hit,Dest=Cache Miss) Test (pass 0x00000001)
Block Copy (Source=Cache Hit,Dest=Cache Hit) Test (pass 0x00000001)
```

```
Memory Write/Write/Read Test (Central Cache on) {pass 0x00000001}
IOC Tag RAM Write/Write/Read Test {pass 0x00000001}
IOC Tag RAM Address Test {pass 0x00000001}
IOC Tag RAM 3-Pattern Test {pass 0x00000001}
IOC Data RAM Write/Write/Read Test {pass 0x00000001}
IOC Data RAM Address Test {pass 0x00000001}
IOC Data RAM 3-Pattern Test {pass 0x00000001}
IOC Data RAM Read Byte Alignment Test {pass 0x00000001}
IOC Data RAM Write Byte Alignment Test {pass 0x00000001}
VME Loopback Test {pass 0x00000001}
VME Loopback and DVMA Test {pass 0x00000001} (not for Sun-3/460)
IOC Read Hit Test {pass 0x00000001}
IOC Invalid Read Miss Test {pass 0x00000001}
IOC Write Hit Test {pass 0x00000001}
IOC Write Miss, No Writeback Test {pass 0x00000001}
IOC Write Miss, Writeback Test {pass 0x00000001}
IOC Read Miss, Writeback Test {pass 0x00000001}
IOC Valid Write Hit (Central Cache Match,Unmod) Test {pass 0x00000001}
IOC Invalid Write Miss (Central Cache Match,Unmod) Test {pass 0x00000001}
IOC Invalid Read Miss (Central Cache Match,Unmod) Test {pass 0x00000001}
IOC Invalid Read Miss (Central Cache Match,Modified) Test {pass 0x00000001}
IOC Valid Read Miss (Central Cache Match),Writeback Test {pass 0x00000001}
IOC Flush (Valid, Modified) Test {pass 0x00000001}
IOC Flush (Valid, Not Modified) Test {pass 0x00000001}
IOC Flush (Not Valid, Not Modified) Test {pass 0x00000001}
IO Mapper Invalid Page (IO.DT) Test {pass 0x00000001}
IOC Write Miss, Writeback (Write Protect) Test {pass 0x00000001}
IOC Invalid Read Miss (IO Mapper IO.EN = 0) Test {pass 0x00000001}
IOC Write Miss (IO Mapper IO.EN = 0) Test {pass 0x00000001}
IOC Random Data Block Write Test {pass 0x00000001}
IOC Random Data Block Read (Central Cache off) Test {pass 0x00000001}
IOC Random Data Block Read (Central Cache on) Test {pass 0x00000001}
<P4 Low Resolution Color Video RAM Board Detected>
P4 Overlay Frame Buffer Write/Write/Read Test {pass 0x00000001}
P4 Overlay Frame Buffer Address Test {pass 0x00000001}
P4 Overlay Frame Buffer 3-Pattern Test {pass 0x00000001}
P4 Overlay Frame Buffer March Test {pass 0x00000001}
P4 Overlay Frame Buffer Read Byte Alignment Test {pass 0x00000001}
P4 Overlay Frame Buffer Write Byte Alignment Test {pass 0x00000001}
P4 Enable Plane Write/Write/Read Test {pass 0x00000001}
P4 Color Plane Write/Write/Read Test {pass 0x00000001}

Selftest passed

Type character within 10 seconds to enter Extended Tests System (e for echo mode)
```

Figure 5 *Sun-3/80 Diagnostic Boot Sequence*

```
Sun-3/80 Boot PROM Selftest Revision xx
  (Press <Esc> to abort tests or <Cntrl-L> for the Loop Menu)
  System Enable Register Read Test (pass 0x00000001)  hh:mm:ss
  PROM Checksum Test (pass 0x00000001)  hh:mm:ss
  Clock/Calendar Device (pass 0x00000001)  hh:mm:ss
  I/O Mapper RAM Write/Write/Read Test (pass 0x00000001)  hh:mm:ss
  I/O Mapper RAM Address Test (pass 0x00000001)  hh:mm:ss
  I/O Mapper RAM 3-Pattern Test (pass 0x00000001)  hh:mm:ss
  <Memory Size = 0x00000010>  hh:mm:ss
  Memory Address Test (pass 0x00000001)  hh:mm:ss
  Memory Read Byte Alignment Test (pass 0x00000001)  hh:mm:ss
  Memory Write Byte Alignment Test (pass 0x00000001)  hh:mm:ss
  I/O Mapper RAM Write/Write/Read Test (pass 0x00000001)  hh:mm:ss
  I/O Mapper RAM Address Test (pass 0x00000001)  hh:mm:ss
  I/O Mapper RAM 3-Pattern Test (pass 0x00000001)  hh:mm:ss
  Bus Error Register Test (pass 0x00000001)  hh:mm:ss
  Level 1 Interrupt Test (pass 0x00000001)  hh:mm:ss
  Level 2 Interrupt Test (pass 0x00000001)  hh:mm:ss
  Level 3 Interrupt Test (pass 0x00000001)  hh:mm:ss
  Configuration Memory Check (pass 0x00000001)  hh:mm:ss
  LANCE Controller Check (pass 0x00000001)  hh:mm:ss
  EPS SCSI Check (pass 0x00000001)  hh:mm:ss
  Floppy Controller Check (pass 0x00000001)  hh:mm:ss
  Printer Controller Check (pass 0x00000001)  hh:mm:ss
  <P4 Low Resolution Color Video RAM Board Detected>
  P4 Overlay Frame Buffer Write/Write/Read Test (pass 0x00000001) hh:mm:ss
  P4 Overlay Frame Buffer Address Test (pass 0x00000001)  hh:mm:ss
  P4 Overlay Frame Buffer 3-Pattern Test (pass 0x00000001)  hh:mm:ss
  P4 Overlay Frame Buffer March Test (pass 0x00000001)  hh:mm:ss
  P4 Overlay Frame Buffer Read Byte Alignment Test (pass 0x00000001) hh:mm:ss
  P4 Overlay Frame Buffer Write Byte Alignment Test (pass 0x00000001) hh:mm:ss
  P4 Enable Plane Write/Write/Read Test (pass 0x00000001)  hh:mm:ss
  P4 Color Plane Write/Write/Read Test (pass 0x00000001)  hh:mm:ss

  END OF SELFTEST #0x00000005 (SELFTEST PASSED/FAILED)
  #PASSED = 0x00000004, #FAILED = 0x00000001

  Initializing Main Memory n Megabytes Initialized

  Type a character within 10 seconds to enter menu tests.
```

Refer to the *PROM User's Manual* for descriptions of the self-tests named for workstations other than the Sun-3/400 series and the Sun-3/80.

Here is a summary of the LED displays for the Sun-3/400 series and Sun-3/80 during the power-up self-tests. Many of the tests are the same for both systems; the test description will indicate which tests is system-specific, In case of error in a Sun-3/400 series test, the LED in bit position 7 also lights. The text that follows this chart shows both the normal test and error displays. As indicated, many tests share the same LED display.

Sun-3/80 LED

LED status	Visual	Condition
Blinking	⊖	Testing
OFF	○	Error
ON (Green)	●	Ok

The Sun-3/80 has one green LED only, which lights steadily during normal function, blinks during a test, and turns off when there is an error. The messages that appear on a terminal attached to Serial Port A describe the specific test in progress and the nature of any errors found.

The following table explains the Sun-3/400 series LED code.

LED Display ●= ON, o= OFF 7 6 5 4 3 2 1 0	***SelfTest being Performed.***
● ● ● ● ● ● ● ●	A reset will set LEDs to this state.
o o o o o o o ●	Keyboard/Mouse SCC Write/Read Test
o o o o o o ● ●	System Enable Register Read Test
o o o o o ● o o	PROM Checksum Test
o o o o o ● o ●	I/O Mapper RAM Test(s)
o o o o o ● ● o	Bus Error Register Test
o o o o o ● ● ●	Interrupt Test(s)
o o o o ● o o o	ECC Memory Sizing and Test(s)
o o o o ● o o ●	Parity Memory Sizing and Test(s)
o o o o ● o ● o	ECC Memory Forced Error Test(s)
o o o o ● o ● ●	Central Cache Tag RAM Test(s)
o o o o ● ● o o	Central Cache Data RAM Test(s)
o o o o ● ● o ●	Central Cache Hit/Miss Test(s)
o o o o ● ● ● o	Block Copy Test(s)
o o o o ● ● ● ●	Memory Write/Write/Read Test(Central Cache on)
o o o ● o o o o	IOC Tag RAM Test(s)
o o o ● o o o ●	IOC Data RAM Test(s)
o o o ● o o ● o	VME Loopback Test
o o o ● o o ● ●	VME Loopback and DVMA (not for 3/460)
o o o ● o ● o o	IOC Read/Write/Flush Test(s)
o o o ● o ● o ●	P4 Overlay Frame Buffer Test(s)

Self-Test Descriptions

The following paragraphs describe each individual test, and the messages and indications generated if it fails. Some tests do not apply to both the Sun-3/400 series and Sun-3/80; refer to the Diagnostic Boot Sequence display examples on previous pages for lists of applicable self-tests.

LED Register Test

The first Sun-3/400 series test is indicated by a slow loop through the Diagnostic LEDs, from LED 7 through LED 0. It is intended to determine if the CPU is able to fetch instructions correctly from the Diagnostic PROM and transfer data across the data bus to the LED status register.

NOTE *If all the LED s on a Sun-3/400 series remain lighted, you could have a low voltage or board seating problem, or the system could contain the wrong Boot PROM.*

NOTE *The diagrams that follow depict the LED states for each self-test. Note that the LEDs are shown here for convenience in reading as binary numbers; the least significant bit is on the right. In most systems, the LEDs are read with bit 0 on the left for desktop installations and with bit 0 on top for deskside installations.*

The tests are described here in the order that they are executed.

UART SCC (Z8530) Port A,B Write/Read Test

This test checks the ability of the Sun-3/400 series workstation CPU to communicate with port A and B of the Zilog Z8530 Serial Communications Chip (SCC). It performs a write/read test of port A and B SCC chip internal registers WR12/RR12. The purpose of this test is to test the path between the CPU and the SCC so that all subsequent tests may display the test name and error status to a terminal attached to Serial Port A.

The test enters a scope loop on a data compare error, but no error messages are displayed on the terminal during this test.

The diagram below summarizes the LED states for a Sun-3/400 series CPU board.

Hexadecimal Value Of LEDs	*Visual Representation*	*Condition*
0x00	**bit 7 oooooooo bit 0**	**okay**
0x80	**•ooooooo**	**error**

Keyboard/Mouse SCC (Z8530) Port A,B Write/Read Test

This test checks the ability of the CPU to communicate with port A and B of the Zilog Z8530 Serial Communications Chip (SCC). It performs a write/read test of port A and B SCC chip internal registers WR12/RR12. Note that this test is for the SCC used for the keyboard and mouse.

The test enters a scope loop on a data compare error, but no error messages are displayed on the terminal during this test because this is a test of the path to the SCC chip.

The diagram below summarizes the LED states for a Sun-3/400 series workstation.

Test Number	*Hexadecimal Value Of LEDs*	*Visual Representation*	*Condition*
1	**0x01**	**bit 7 ○○○○○○○● bit 0**	**okay**
1	**0x81**	**●○○○○○○●**	**error**

System Enable Register Read Test

This test reads the System Enable Register and verifies that all bits read are zero, ignoring the Diagnostics Switch bit.

The test enters a scope loop on data compare errors, with the following terminal message:

```
exp xxxxxxxx, obs xxxxxxxx, xor xxxxxxxx
```

The diagram below summarizes the LED states.

Test Number	*Hexadecimal Value Of LEDs*	*Visual Representation*	*Condition*
3	**0x03**	**bit 7 ○○○○○○●● bit 0**	**okay**
3	**0x83**	**●○○○○○●●**	**error**

PROM Checksum Test

The checksum of all locations in both PROMs with the exception of the last word is calculated and compared to an expected value that is stored in the last word of the second PROM. If this test fails, the PROMs should be replaced.

The test enters a scope loop on data compare errors, with the following terminal message:

Checksum error: Exp xxxxxxxx, Obs xxxxxxxx

The diagram below summarizes the LED states.

Test Number	*Hexadecimal Value Of LEDs*	*Visual Representation*	*Condition*
4	**0x04**	**bit 7 ooooo•oo bit 0**	**okay**
4	**0x84**	**•oooo•oo**	**error**

I/O Mapper Write/Write/Read Test

This test verifies the address and data paths to the IO Mapper RAM, as well as address bit and data bit uniqueness.

For each test address of IO Mapper RAM:
(base+0x0,base+0x01,base+0x02,base+0x04,base+0x08,...,base+iomapper_size)

For each data pattern at each test address:
(0x0,0x1,0x2,0x4,0x10,0x20,...0x80000000)
The test does the following:

- Writes test data to the test address.
- Writes inverted test data to test address + 0x04.
- Reads back data from the test address and compares.

Upon error, the test loops through the steps shown above with constant test data and a constant test address.

This test enters a scope loop on data compare errors, with the following terminal message:

```
addr xxxxxxxx, exp xxxxxxxx, obs xxxxxxxx
```

The diagram below summarizes the LED states.

Test Number	*Hexadecimal Value Of LEDs*	*Visual Representation*	*Condition*
5	**0x05**	**bit 7 ooooo•o• bit 0**	**okay**
5	**0x85**	**•oooo•o•**	**error**

I/O Mapper RAM Address Test

This test writes the complete I/O Mapper RAM address space with the longword address as the data, then reads back the entire address space and verifies that no addresses are overwritten. This is a test for addressing uniqueness of the I/O Mapper RAM.

This test enters a scope loop on data compare errors, with the following terminal message:

```
addr xxxxxxxx, exp xxxxxxxx, obs xxxxxxxx
```

The LED display shown for the I/O Mapper Write/Write/Read Test applies to this test also.

I/O Mapper RAM 3-Pattern Test

This test writes the complete I/O Mapper RAM address space with a repeated three-long-word pattern sequence, then reads back the entire address space and verifies the data.

This test enters a scope loop on data compare errors, with the following terminal message:

```
addr xxxxxxxx, exp xxxxxxxx, obs xxxxxxxx
```

The LED display shown for the I/O Mapper Write/Write/Read Test applies to this test also.

Bus Error Register Test

This test does the following:

1. Verifies that attempting to read an invalid memory address causes a bus error, with appropriate data written to the BUSERR register.
2. Verifies that attempting to read an FPA device address while the ENABLE-FPA bit in the System Enable Register is off causes a bus error, with appropriate data written to the BUSERR register.

This test enters a scope loop upon error, with one or more of the following terminal error messages:

```
error1: No Bus Error Reading Invalid Memory Address, read addr = 0xyyyyyyyy.
error2: Bus error reg TIMEOUT bit not set.
error3: No Bus Error Reading Disabled FPA, read addr = 0xyyyyyyyy.
error4: Bus error reg FPAENERR bit not set.
```

The diagram below summarizes the LED states.

Test Number	*Hexadecimal Value Of LEDs*	*Visual Representation*	*Condition*
6	**0x06**	**bit 7 ooooo●●o bit 0**	**okay**
6	**0x86**	**●oooo●●o**	**error**

Level 1 Interrupt Test

This test forces a Level 1 soft interrupt to verify that an autovector Level 1 interrupt will occur.

This test enters a scope loop upon error, with the following error message:

```
error: No level 1 interrupt occurred.
```

The diagram below summarizes the LED states.

Test Number	*Hexadecimal Value Of LEDs*	*Visual Representation*	*Condition*
7	0x07	bit 7 ooooo●●● bit 0	okay
7	0x87	●ooooo●●	error

Level 2 Interrupt Test

This test forces a level 2 soft interrupt to verify that an autovector Level 1 interrupt will occur.

The test enters a scope loop upon error, with the following terminal error message:

```
error: No level 2 interrupt occurred.
```

The LED display for this test is the same as that for the Level 1 Interrupt test.

Level 3 Interrupt Test

This test forces a level 3 soft interrupt to verify that an autovector Level 3 interrupt will occur.

The test enters a scope loop upon error, with the following error message:

```
error: No level 3 interrupt occurred.
```

The LED display for this test is the same as that for the Level 1 Interrupt test.

TOD Clock Interrupt Test

This test enables the TOD CLock to interrupt and then verifies that the interrupt occurs.

This test enters a scope loop upon error, with the following error message:

```
error: No TOD interrupt occurred.
```

The LED display for this test is the same as that for the Level 1 Interrupt test.

Memory Write/Write/Read Test

This test verifies the address and data paths to Main Memory, as well as address bit and data bit uniqueness.

For each test address of main memory:
(0x0,0x1,0x2,0x4,0x8,0x10,0x20,...size_of_mem_installed)

For each data pattern at each test address:
(0x0,0x1,0x2,0x4,0x10,0x20,...0x80000000)
The test does the following:

- Writes test data to test address.
- Writes inverted test data to test address + 0x04.
- Reads back data from test address and compares.

Upon error, the test loops through the steps shown above with constant test data and a constant test address.

This test enters a scope loop on data compare errors, with the following message:

```
addr xxxxxxxx, exp xxxxxxxx, obs xxxxxxxx
```

The diagram below summarizes the LED states.

Test Number	*Hexadecimal Value Of LEDs*	*Visual Representation*	*Condition*
8	**0x08**	**bit 7 oooo•ooo bit 0**	**okay**
8	**0x88**	**•ooo•ooo**	**error**

Memory Address Test

This test writes the complete Main Memory address space with the longword address as the data, then reads back the entire address space and verifies that no addresses are overwritten. This is a test for addressing uniqueness of the Main Memory RAM.

This test enters a scope loop on data compare errors, with the following terminal message:

```
addr xxxxxxxx, exp xxxxxxxx, obs xxxxxxxx
```

The LED display is that same as that shown for the Memory Write/Write/Read Test.

Memory 3-Pattern Test

This test writes the complete Main Memory address space with a repeated three long word pattern sequence, then reads back the entire address space and verifies the data.

This test enters a scope loop on data compare errors, with the following terminal message:

```
addr xxxxxxxx, exp xxxxxxxx, obs xxxxxxxx
```

The LED display is that same as that shown for the Memory Write/Write/Read Test.

Memory Read Byte Alignment Test

This test verifies that byte, word, and long word read operations produce the appropriate byte-aligned data.

The test enters a scope loop on data compare errors, with the following message:

```
byte misalignment: addr xxxxxxxx, exp xxxxxxxx, obs xxxxxxxx
```

The LED display is that same as that shown for the Memory Write/Write/Read Test.

Memory Write Byte Alignment Test

This test verifies that byte, word, and long word read operations produce the appropriate byte-aligned data for various byte-aligned write operations.

This test enters a scope loop on data compare errors, with the following terminal message:

```
byte misalignment: addr xxxxxxxx, exp xxxxxxxx, obs xxxxxxxx
```

The LED display is that same as that shown for the Memory Write/Write/Read Test.

Parity Memory No Error Test

This test will execute only if a Parity Memory board was detected when main memory was sized.

The test does the following:

1. Determines if Parity memory is installed. If not, it bypasses the test.
2. Initializes all Parity memory with zero to initialize the check bits.
3. For each test address 0x00000000,0x00000001,0x00000002,...,0x00040000, 0x00800000,0x00800001,0x00800002,...0x00840000, or up to total Parity memory installed (16MB max):

   For each data pattern 0x00000001,0x00000003,0x00000007,...0xffffffff at one specific test address:

   The test:
   (a) Writes 0x0 to Memory Error Address register to clear previous interru
   (b) Enables Parity checking on Parity board.
   (c) Enables Level 7 interrupts in Memory Error Control Register.
   (d) Enables system interrupts in System Interrupt Register.
   (e) Verifies that Memory Error Control register was properly set.
   (f) Writes longword test data to test address.
   (g) Reads longword from test address.
   (h) Verifies that no Level 7 interrupt occurred.
   (i) Verifies that Memory Error Control register was not modified.

Upon error, the test loops through steps a - i with a constant test address and constant test data.

Possible error messages for this test are:

```
error1: Memory Error Control Reg not written:
exp=0x00000000, obs=0x00000000, testaddr=0x00000000, testdata=0x00000000
error2: Unexpected parity error (level 7) interrupt:
testaddr=0x00000000, testdata=0x00000000
Memory Error Control Reg: obs=0x00000000
Parity Memory Error Reg:  obs=0x00000000
Memory Error Address Reg: obs=0x00000000
error3: Bad Memory Error Control Reg after memory write:
exp=0x00000000, obs=0x00000000, testaddr=0x00000000, testdata=0x00000000
```

The diagram below summarizes the LED states.

Test Number	*Hexadecimal Value Of LEDs*	*Visual Representation*	*Condition*
9	0x09	bit 7 oooo•oo• bit 0	okay
9	0x89	•ooo•oo•	error

Parity Memory Forced Error Test

This test will execute only if a Parity Memory board was detected when main memory was sized.

For each test address 0x00000000,0x00000001,0x00000002,...,0x00040000, 0x00800000,0x00800001,0x00800002,...0x00840000 or up to total Parity memory installed (16MB max):

For each data pattern 0x1,0x2,0x4,...0x00000100 at one specific test address:

The test does the following:

Writes 0x0 to Memory Error Address register to clear previous interrupt.
Enables Parity memory board.
Writes longword 0x0 to test address, test address + 4 (good parity).
Enables Parity checking and Parity Test on Parity board.
Enables Level 7 interrupts in Memory Error Control Register.
Enables system interrupts in system interrupt Register.
Writes longword test data to test address.
Reads longword from test address.
Verifies that Level 7 interrupt occurred.
Verifies that Memory Error Address register captured test address.
Verifies that Parity Memory Error register was set correctly
Verifies that Memory Error Control register was set correctly.

Upon error, the test loops through the steps shown above with a constant test address and constant test data.

Possible error messages for this test are:

```
error1: no parity error (level 7) interrupt occurred when bad parity forced.
addr = 0x00000000, wr/rd data = 0x00000000

error2: Memory Error Address Reg: exp=0x00000000, obs=0x00000000
addr = 0x00000000, wr/rd data = 0x00000000

error3: Parity Memory Error Reg: exp=0x00000000, obs=0x00000000
addr = 0x00000000, wr/rd data = 0x00000000

error4: Bad Memory Error Control Reg after memory write:
exp=0x00000000, obs=0x00000000, testaddr=0x00000000, testdata=0x00000000
```

The LED display shown for the Parity Memory No Error Test applies to this test also.

ECC Memory No Error Test

This test does the following:

- Checks if ECC memory is installed. If not, it bypasses this test.
- Determines the starting address and size of total ECC memory.
- Initializes all ECC memory with zero to initialize ECC check bits.
- For each test address 0x00000000,0x00000001,0x00000002,...,0x00040000, 0x00800000,0x00800001,0x00800002,...0x00840000, 0x01000000,0x01000001,0x01000002,...0x01040000, 0x01800000,0x01800001,0x01800002,...0x01840000, ...,

0x08800000,0x08800001,0x08800002,...0x00880000, or up to total memory installed (144MB max) and for each data pattern 0x00000001,0x00000003,0x00000007,...0xffffffff at one specific test address, the test does the following:

(a) Enables ECC checking on all ECC boards.
(b) Enables Level 7 interrupts in the Memory Error Control Reg.
(c) Enables system interrupts in the System Interrupt Register.
(d) Verify that Memory Error Control register was properly set.
(e) Reads a longword from test address.
(f) Verifies that no Level 7 interrupt occurred.
(g) Verifies that the Memory Error Control register was not modified.

Upon error, the test loops through steps a - g with a constant test address and constant test data.

Possible error messages for this test are:

```
error1: Memory Error Control Reg not written:
exp=0x00000000, obs=0x00000000, testaddr=0x00000000, testdata=0x00000000

error2: Unexpected ECC error (level 7) interrupt:
testaddr=0x00000000, testdata=0x00000000
Memory Error Control Reg: obs=0x00000000
Memory Error Address Reg: obs=0x00000000
Syndrome Reg (Board 0):   obs=0x00000000

error3: Bad Memory Error Control Reg after memory write:
exp=0x00000000, obs=0x00000000, testaddr=0x00000000, testdata=0x00000000
```

The diagram below summarizes the LED states.

Test Number	*Hexadecimal Value Of LEDs*	*Visual Representation*	*Condition*
10	**0x0a**	**bit 7oooo●o●obit 0**	**okay**
10	**0x8a**	**●ooo●o●o**	**error**

ECC Memory Forced CE Test

This test does the following:

- Gets the starting address and size of total ECC memory.
- For each test address 0x00000000,0x00800000,0x01000000,...0x07800000, (starting at ECC memory base), or up to total ECC memory installed, the test:

  (a) Resets syndrome register.
  (b) Enables CE, level 7 interrupts in the Memory Error Control Register.
  (c) Verifies that Memory Error Control register was properly set.
  (d) Writes data to memory.
  (e) Writes check bits to the Diagnostic registers.
  (f) Enables ECC checking, DM1 on memory board.
  (g) Enables system interrupts in the System Interrupt Register.
  (h) Reads a longword from test address.
  (i) Verifies that a Level 7 interrupt occurred.
  (j) Verifies that the Memory Error Control register was modified correctly.
  (k) Verifies that the CE bit was set in the Memory board syndrome register.
  (l) Verifies that the syndrome code in the syndrome register was correctly set.

Upon error, the test loops through steps a - l.

Possible error messages for this test are:

```
error1: Memory Error Control Reg not written:
exp=0x00000000, obs=0x00000000, testaddr=0x00000000

error2: No ECC error (level 7) interrupt occurred after CE forced:
testaddr=0x00000000

error3: Bad Memory Error Control Reg after CE forced:
exp=0x00000000, obs=0x00000000, testaddr=0x00000000

error4: CE bit in ECC syndrome reg not set after CE forced:
testaddr=0x00000000

error5: Bad ECC Syndrome Reg Syndrome code (b31:24) after CE forced:
testaddr=0x00000000, exp=0x00000000, obs 0x00000000
```

The LED display for this test is the same as that for the ECC Memory No Error Test.

ECC Memory Forced UE Test

This test does the following:

- Determines the starting address and size of total ECC memory.
- For each test address 0x00000000,0x00800000,0x01000000,...0x07800000, (starting at ECC memory base), or up to total ECC memory installed, the test:

  (a) Resets the Syndrome Register.
  (b) Enables Level 7 interrupts in the Memory Error Control Register.
  (c) Verifies that the Memory Error Control register was properly set.
  (d) Writes data to memory.
  (e) Writes check bits to the Diagnostic registers.
  (f) Enables ECC checking, DM1 on the memory board.
  (g) Enables system interrupts in the System Interrupt register.
  (h) Reads a longword from test address.
  (i) Verifies that a Level 7 interrupt occurred.
  (j) Verifies that the Memory Error Control register was correctly modified.
  (k) Verify that the syndrome code in Syndrome register was correctly set.

Upon error, the test loops through steps a -k.

Possible error messages for this test are:

```
error1: Memory Error Control Reg not written:
exp=0x00000000, obs=0x00000000, testaddr=0x00000000.

error2: No ECC error (level 7) interrupt occurred after UE forced:
testaddr=0x00000000

error3: Bad Memory Error Control Reg after UE forced:
exp=0x00000000, obs=0x00000000, testaddr=0x00000000

error5: Bad ECC Syndrome Reg Syndrome code (b31:24) after UE forced:
testaddr=0x00000000, exp=0x00000000, obs 0x00000000
```

The LED display for this test is the same as that for the ECC Memory No Error Test.

Central Cache Tag RAM Write/Write/Read Test

This test verifies the address and data paths to the Central Cache Tag RAM, as well as address bit and data bit uniqueness.

For each test address of Central Cache TAG RAM:
(base+0x0,base+0x10,base+0x40,base+0x80,...,base+cache_size)

For each data pattern at each test address:
(0x00000000,0x00010000,0x00020000,0x00040000,...0x80000000)

The test does the following:

(a) Writes test data to test address.
(b) Writes inverted test data to test address + 0x10 (next tag).
(c) Reads back data from test address and compare.
NOTE: Only bits 16:31 of data read back are valid!

Upon error, the test loops through steps a - c with constant test data and a constant test address.

This test enters a scope loop on data compare errors, with the following terminal message:

```
addr xxxxxxxx, exp xxxxxxxx, obs xxxxxxxx
```

The diagram below summarizes the LED states.

Test Number	*Hexadecimal Value Of LEDs*	*Visual Representation*	*Condition*
11	**0x0b**	**bit 7 oooo●o●● bit 0**	**okay**
11	**0x8b**	**●ooo●o●●**	**error**

Central Cache Tag RAM Address Test

This test writes the complete Central Cache Tag RAM address space with the longword address as the data, then reads back the entire address space and verifies that no addresses are overwritten. Only the most significant 16 bits (b31:16) are verified, as the least significant 16 (b15:00) are invalid for reads. This is a test for addressing uniqueness of the Central Cache Tag RAM.

This test enters a scope loop on data compare errors, with the following message:

```
addr xxxxxxxx, exp xxxxxxxx, obs xxxxxxxx
```

The LED display for this test is the same as that for the Central Cache Tag RAM Write/Write/Read Test.

Central Cache Tag RAM 3-Pattern Test

This test writes the complete Central Cache Tag RAM address space with a repeated three-long-word pattern sequence, then reads back the entire address space and verifies the data. Only the most significant 16 bits (b31:16) are verified, because the least significant 16 (b15:00) are invalid for reads.

This test enters a scope loop on data compare errors, with the following terminal message:

```
addr xxxxxxxx, exp xxxxxxxx, obs xxxxxxxx
```

The LED display for this test is the same as that for the Central Cache Tag RAM Write/Write/Read Test.

Central Cache Data RAM Write/Write/Read Test

This test verifies the address and data paths to the Central Cache Data RAM, as well as address bit and data bit uniqueness.

For each test address of Central Cache Data RAM:
(base+0x0,base+0x01,base+0x02,base+0x04,base+0x08,...,base+cache_size)

For each data pattern at each test address:
(0x0,0x1,0x2,0x4,0x10,0x20.....0x80000000)
The test does the following:

(a) Writes test data to test address.
(b) Writes inverted test data to test address + 0x04.
(c) Reads back data from test address and compares.

Upon error, the test loops through steps a - c with constant test data and a constant test address.

This test enters a scope loop on data compare errors, with the following message:

```
addr xxxxxxxx, exp xxxxxxxx, obs xxxxxxxx
```

The diagram below summarizes the LED states.

Test Number	*Hexadecimal Value Of LEDs*	*Visual Representation*	*Condition*
11	**0x0b**	**bit 7 oooo●o●● bit 0**	**okay**
11	**0x8b**	**●ooo●o●●**	**error**

Central Cache Data RAM Address Test

This test writes the complete Central Cache Data RAM address space with the longword address as data, then reads back the entire address space and verifies that no addresses are overwritten. This is a test for addressing uniqueness of the Central Cache Data RAM.

This test enters a scope loop on data compare errors, with the following message:

```
addr xxxxxxxx, exp xxxxxxxx, obs xxxxxxxx
```

The diagram below summarizes the LED states.

Test Number	*Hexadecimal Value Of LEDs*	*Visual Representation*	*Condition*
12	**0x0c**	**bit 7 oooo••oo bit 0**	**okay**
12	**0x8c**	**•ooo••oo**	**error**

Central Cache Data RAM 3-Pattern Test

This test writes the complete Central Cache Data RAM address space with a repeated three-long- word pattern sequence, then reads back the entire address space and verifies the data.

This test enters a scope loop on data compare errors, with the following message:

```
addr xxxxxxxx, exp xxxxxxxx, obs xxxxxxxx
```

The LED display for this test is the same as that shown for the Central Cache Data RAM Address Test.

Central Cache Data RAM Read Byte Alignment Test

This test verifies that byte, word, and long word read operations produce the appropriate byte-aligned data.

The test enters a scope loop on data compare errors, with the following message:

```
byte misalignment: addr xxxxxxxx, exp xxxxxxxx, obs xxxxxxxx
```

The LED display for this test is the same as that shown for the Central Cache Data RAM Address Test.

Central Cache Data RAM Write Byte Alignment Test

This test verifies that byte, word, and long word read operations produce the appropriate byte-aligned data for various byte-aligned write operations.

This test enters a scope loop on data compare errors, with the following terminal message:

```
byte misalignment: addr xxxxxxxx, exp xxxxxxxx, obs xxxxxxxx
```

The LED display for this test is the same as that shown for the Central Cache Data RAM Address Test.

Central Cache Read Hit Test

This test verifies that a data operand read from system memory with the memory block address in the cache valid causes a read from the cache and not from system memory.

The test does the following:

1. Turns off EN_CACHE,EN_IOCACHE,EN_DVMA,EN_VME_LOOP in the System Enable Register.
2. Clears Central Cache tags, data, and the first 64 Kbytes of system memory.
3. For each test address 0x0,0x1,0x2,0x4.....0x08000000 (128MB), or up to maximum memory installed, the test:

   (a) Writes 00-0f as data into memory for cache line of interest.
   (b) Writes ff-f0 as data into cache data RAM for cache line of interest.
   (c) Writes test address, valid bit ON in cache tag RAM.
   (d) Turns on central cache.
   (e) Reads from test address.
   (f) Turns off central cache.
   (g) Verifies that the data read is from cache, not system memory.
   (h) Verifies cache tag is still correct (unmodified).
   (i) Verifies cache data is still correct (unmodified).

Upon error, the test loops through steps a - i with a constant test address.

Possible error messages for this test are:

```
error1: Bad read data on cache valid read:
testaddr=0x00000000,readaddr=0x00000000,exp=0x00000000,obs=0x00000000

error2: Bad cache tag ram after cache valid read:
testaddr=0x00000000,tagramaddr=0x00000000,exp=0x00000000,obs=0x00000000

error3: Bad cache data ram after cache valid read:
testaddr=0x00000000,dataramaddr=0x00000000,exp=0x00000000,obs=0x00000000
```

The diagram below summarizes the LED states.

Test Number	*Hexadecimal Value Of LEDs*	*Visual Representation*	*Condition*
13	**0x0d**	**bit 7 oooo●●o● bit 0**	**okay**
13	**0x8d**	**●ooo●●o●**	**error**

Central Cache Invalid Read Miss Test

This test verifies that doing a data operand read from system memory with the memory block address in the cache invalid causes a read from system memory and not the cache. The test does the following:

1. Turns off EN_CACHE,EN_IOCACHE,EN_DVMA,EN_VME_LOOP in the System Enable Register.
2. Clears Central Cache tags, data, and the first 64 Kbytes of system memory.
3. For each test address 0x0,0x1,0x2,0x4....0x08000000 (128MB), or up to maximum memory installed, the test:

    (a) Writes 00-0f as data into memory for cache line of interest.
    (b) Writes ff-f0 as data into cache data RAM for cache line of interest.
    (c) Writes test address, valid bit OFF in cache tag RAM.
    (d) Turns on central cache.
    (e) Reads from test address.
    (f) Turns off central cache.
    (g) Verifies that the data read is from system memory, not cache.
    (h) Verifies cache tag is updated correctly.
    (i) Verifies cache data is updated correctly.

Upon error, the test loop through steps a - i with a constant test address. Possible error messages for this test are:

```
error1: Bad read data on cache invalid read:
testaddr=0x00000000, readaddr=0x00000000,exp=0x00000000,obs=0x00000000

error2: Bad cache tag ram after cache invalid read:
testaddr=0x00000000,tagramaddr=0x00000000,exp=0x00000000,obs=0x00000000

error3: Bad cache data ram after cache invalid read:
testaddr=0x00000000,dataramaddr=0x00000000,exp=0x00000000,obs=0x00000000
```

The LED display for this test is the same as that shown for the Central Cache Read Hit Test.

Central Cache Valid Read Miss Test

This test verifies that a data operand read from system memory with a valid modulo 64 Kbyte memory block address in the cache causes a read from system memory and not the cache.

The test does the following:

1. Turns off EN_CACHE,EN_IOCACHE,EN_DVMA,EN_VME_LOOP in the System Enable Register.
2. Clears Central Cache tags, data, and the first 64 Kbytes of system memory.
3. For each test address 0x0,0x1,0x2,0x4....0x08000000 (128MB), or up to maximum memory installed, the test:

    (a) Writes 00-0f as data into the memory line for test address (x).
    (b) Writes 50-5f as data into the memory line for addr x+64KB.
    (c) Writes ff-f0 as data into the cache data RAM for addr x.
    (d) Writes the test address, valid bit ON in cache tag RAM.
    (e) Turns on central cache.
    (f) Reads from address x+64KB.
    (g) Turns off central cache.
    (h) Verifies that the data read is from memory, not cache.
    (i) Verifies that the cache tag is updated correctly
    (j) Verifies that cache data is updated correctly.

Upon error, the test loops through steps a - j with a constant test address. Possible error messages for this test are:

```
error1: Bad system memory after cache valid write hit:
testaddr=0x00000000,memaddr=0x00000000,exp=0x00000000,obs=0x00000000

error2: Bad cache tag ram after cache valid write hit:
testaddr=0x00000000,tagramaddr=0x00000000,exp=0x00000000,obs=0x00000000

error3: Bad cache data ram after cache valid write hit:
testaddr=0x00000000,dataramaddr=0x00000000,exp=0x00000000,obs=0x00000000
```

The LED display is the same as that for the Central Cache Read Hit Test.

Central Cache Write Hit Test

This test verifies that doing a data operand write to system memory with the memory block address in the cache valid causes a write to the cache and not to system memory.

The test does the following:

1. Turns off EN_CACHE, EN_IOCACHE, EN_DVMA, EN_VME_LOOP in the System Enable Register.
2. Clears Central Cache tags, data, and the first 64 Kbytes of system memory.
3. For each test address 0x0,0x1,0x2,0x4....0x08000000 (128MB), or up to maximum memory installed, the test:
   (a) Writes the longword test address as data into memory.
   (b) Writes longword zero into cache data RAM.
   (c) Writes the address, valid bit ON in cache tag RAM.
   (d) Turns on central cache.
   (e) Writes the test address with inverted longword address as data.
   (f) Turns off central cache.
   (g) Verifies that the system memory location was unmodified, since the write should have gone only to the cache (no write-through).
   (h) Verifies that the cache tag was updated correctly (valid,dirty,addr bits).
   (i) Verifies that cache data was written correctly.

Upon error, the test loops through steps a - i with a constant test address. Possible error messages for this test are:

```
error1: Bad system memory after cache valid write hit:
testaddr=0x00000000,memaddr=0x00000000,exp=0x00000000,obs=0x00000000

error2: Bad cache tag ram after cache valid write hit:
testaddr=0x00000000,tagramaddr=0x00000000,exp=0x00000000,obs=0x00000000

error3: Bad cache data ram after cache valid write hit:
testaddr=0x00000000,dataramaddr=0x00000000,exp=0x00000000,obs=0x00000000
```

The LED display is the same as that shown for the Central Cache Read Hit Test.

Central Cache Write Miss, No Writeback Test

This test verifies that a data operand write to system memory address (X+64KB) with a valid, not dirty memory block address (X) in the cache causes a write to the cache and not to system memory, and that no writeback to system memory occurs.

The test does the following:

1. Turns off EN_CACHE, EN_IOCACHE, EN_DVMA, EN_VME_LOOP in the System Enable Register.
2. Clears Central Cache tags, data, and the first 64 Kbytes of system memory.
3. For each test address 0x0,0x1,0x2,0x4....0x08000000 (128MB), or up to maximum memory installed, the test:

   (a) Writes the longword address as data into memory.
   (b) Writes the inverted longword address as data into cache data RAM.
   (c) Writes the correct address, valid bit ON in cache tag ram.
   (d) Turns on central cache.
   (e) Writes to (address+64KB) with the inverted longword address as data.
   (f) Turns off central cache.
   (g) Verifies that the system memory location was unmodified, since the write should have gone only to the cache (no write-through).
   (h) Verifies that the cache tag was updated correctly (valid,dirty,addr bits).
   (i) Verifies that cache data was written correctly.

   Upon error: loop through steps (a) --> (i) with constant test address. Possible error messages for this test are:

```
error1: Bad system memory after cache valid write miss:
testaddr=0x00000000,memaddr=0x00000000,exp=0x00000000,obs=0x00000000

error2: Bad cache tag ram after cache valid write miss:
testaddr=0x00000000,tagramaddr=0x00000000,exp=0x00000000,obs=0x00000000

error3: Bad cache data ram after cache valid write miss:
testaddr=0x00000000,dataramaddr=0x00000000,exp=0x00000000,obs=0x00000000
```

The LED display is the same as that shown for the Central Cache Read Hit Test.

Central Cache Write Miss, Writeback Test

This test verifies that a data operand write to system memory address (X+64KB) with a valid and dirty memory block address (X) in the cache causes a write to the cache and also a writeback to system memory. The test does the following:

1. Turns off EN_CACHE,EN_IOCACHE,EN_DVMA,EN_VME_LOOP in the System Enable Register.
2. Clears Central Cache tags, data, and the first 64 Kbytes of system memory.
3. For each test address 0x0,0x1,0x2,0x4....0x08000000 (128MB), or up to maximum memory installed, the test:

   (a) Writes the longword address as data into memory.
   (b) Writes inverted longword address as data into cache data RAM.
   (c) Writes address, valid bit ON, dirty bit ON in cache tag RAM.
   (d) Turns on central cache.
   (e) Writes to (address+64KB) with inverted longword address as data.
   (f) Turns off central cache.
   (g) Verifies that system memory location was modified, i.e. a writeback of data initially in cache ram occurred.
   (h) Verifies cache tag was updated correctly (valid,dirty,addr bits).
   (i) Verifies cache data was written correctly.

   Upon error, the test loops through steps a - i with a constant test address.

   Possible error messages for this test are:

```
error1: Bad read data on line cross cache invalid read:
testaddr=0x00000000,readaddr=0x00000000,exp=0x00000000,obs=0x00000000

error2: Bad cache tag ram after line cross cache invalid read:
testaddr=0x00000000,tagramaddr=0x00000000,exp=0x00000000,obs=0x00000000

error3: Bad cache data ram after line cross cache invalid read:
testaddr=0x00000000,dataramaddr=0x00000000,exp=0x00000000,obs=0x00000000
```

The LED display is the same as that shown for the Central Cache Read Hit Test.

Central Cache Line Cross Invalid Read Miss Test

This test verifies that doing a longword data operand read from system memory across a 16-byte line boundary with invalid memory block addresses in the cache causes a read from system memory and not the cache, and that both 16-byte lines are copied into the cache. The test does the following:

1. Turns off EN_CACHE,EN_IOCACHE,EN_DVMA,EN_VME_LOOP in the System Enable Register.
2. Clears Central Cache tags, data, and the first 64 Kbytes of system memory.
3. For each test address 0x0,0x1,0x2,0x4....0x08000000 (128MB), or up to maximum memory installed, the test:
   (a) Writes the longword address as data into memory for the two 16-byte lines starting at the address of interest (8 longword writes).
   (b) Writes the inverted longword address as data into cache data RAM for the two 16-byte lines starting at the address of interest (8 longword writes).
   (c) Writes the two correct addresses, valid bit OFF in cache tag RAM.
   (d) Turns on central cache.
   (e) Reads a longword from (address + 0x0e), to access data across the line boundary.
   (f) Turns off central cache.
   (g) Verifies that the data read is the same as the address, and therefore, is not read from cache, but rather from system memory.
   (h) Verifies that the cache tag is updated correctly (both entries).
   (i) Verifies that cache data is updated correctly (all 32 bytes).

   Upon error, the test loops through steps a - i with a constant test address.

   Possible error messages for this test are:

```
error1: Bad read data on line cross cache invalid read:
testaddr=0x00000000,readaddr=0x00000000,exp=0x00000000,obs=0x00000000

error2: Bad cache tag ram after line cross cache invalid read:
testaddr=0x00000000,tagramaddr=0x00000000,exp=0x00000000,obs=0x00000000

error3: Bad cache data ram after line cross cache invalid read:
testaddr=0x00000000,dataramaddr=0x00000000,exp=0x00000000,obs=0x00000000
```

The LED display is the same as that shown for the Central Cache Read Hit Test.

Central Cache Line Cross Write Miss Writeback Test

This test Verifies that a longword data operand write to system memory (X+64KB) across a 16-byte line boundary with a valid and dirty memory block address (X) in the cache causes a write to the cache with the new data and a writeback to memory with the old data in the cache. It verifies that both 16-byte lines are correct in memory as well as cache. The test does the following:

1. Turns off EN_CACHE,EN_IOCACHE,EN_DVMA,EN_VME_LOOP in the System Enable Register.
2. Clears Central Cache tags, data, and the first 64 Kbytes of system memory.
3. For each test address 0x0,0x1,0x2,0x4....0x08000000 (128MB), or up to maximum memory installed, the test:
   (a) Writes the longword address as data into memory for the two 16-byte lines at address X and for two lines at address (X+64KB).
   (b) Writes the inverted longword address as data into cache data RAM for the two 16-byte lines starting at the address of interest.
   (c) Writes the two correct addresses, valid bit ON and dirty bit ON in cache tag RAM.
   (d) Turns on central cache.
   (e) Writes a longword to (address+64KB+0x0e), to access data across the line boundary and to cause a writeback.
   (f) Turns off central cache.
   (g) Verifies that memory was modified through writeback of data previously in cache (all 32 bytes).
   (h) Verifies cache tag is updated correctly (both entries).
   (i) Verifies cache data is updated correctly (all 32 bytes).

   Upon error: loop through steps (a) --> (i) with constant test address. Possible error messages for this test are:

```
error1: Bad memory data on line cross cache write miss writeback:
testaddr=0x00000000,memmaddr=0x00000000,exp=0x00000000,obs=0x00000000

error2: Bad cache tag ram after line cross cache write miss writeback:
testaddr=0x00000000,tagramaddr=0x00000000,exp=0x00000000,obs=0x00000000

error3: Bad cache data ram after line cross cache write miss writeback:
testaddr=0x00000000,dataramaddr=0x00000000,exp=0x00000000,obs=0x00000000
```

The LED display is the same as that shown for the Central Cache Read Hit Test.

Central Cache Writeback Timeout Test

This test verifies that a data operand write to system memory address 0x0 with a valid and dirty memory block address 0x20000000 (a nonexistent memory address) in the cache causes a write to the cache and a writeback timeout due to attempting to writeback to a nonexistent memory address. The test does the following:

1. Turns off EN_CACHE,EN_IOCACHE,EN_DVMA,EN_VME_LOOP in the System Enable Register.
2. Clears Central Cache tags, data, and the first 64 Kbytes of system memory.
3. For each test address 0x0,0x1,0x2,0x4....0x08000000 (128MB), or up to maximum memory installed, the test:
   (a) Write line for addr x (test addr) with 00-0f data.
   (b) Write cache data ram line for addr x with ff-f0 data.
   (c) Write address (x+0x9000000), VALID, DIRTY in cache tag ram. Maximum addressable memory for Sun-3/400 is 144 MB = 0x9000000.
   (d) Turn on central cache.
   (e) Write to address x with longword 0x50515253 as data.
   (f) Turn off central cache.
   (g) Verify WBTIMOUT bit Memory Error Control Reg was set.
   (h) Verify Memory Error Addr Reg captured invalid writeback addr.
   (i) Verify that entire line at addr x was not modified.
   (j) Verify cache tag was updated correctly (valid,dirty,addr bits).
   (k) Verify cache data was written correctly.

   Upon error: loop continuously through steps a - k. Possible error messages for this test are:

```
error1: No level 7 interrupt occurred.
testaddr=0x00000000,tagramaddr=0x00000000,dataramaddr=0x00000000

error2: Memory Error Control Reg Asynch Timeout bit not set.
testaddr=0x00000000,exp=0x00000000,obs=0x00000000

error3: Bad Mem Err Addr Reg after cache invalid writeback:
testaddr=0x00000000,exp=0x00000000,obs=0x00000000
```

The LED display is the same as that shown for the Central Cache Read Hit Test.

Block Copy (Source=Cache Miss,Dest=Cache Miss) Test

This test verifies that doing a block copy read from memory followed by a block copy write to memory with all Central Cache entries invalidated causes the proper blocks of data to be transferred in memory. The test does the following:

1. Turns off EN_CACHE,EN_IOCACHE,EN_DVMA,EN_VME_LOOP in the System Enable Register.
2. Clears Central Cache tags, data, and the first 64 Kbytes of system memory.
3. For each test address 0x0,0x1,0x2,0x4....0x08000000 (128MB), or up to maximum memory installed, the test:
   (a) Writes the line for addr x (test addr) in memory with 00-0f data.
   (b) Writes the line for addr x+0x10 in memory with ff-f0 data.
   (c) Writes the line for addr x+0x20 in memory with all zero data.
   (d) Does a block copy read from addr x.
   (e) Does a block copy write to addr x+0x10.
   (f) Verifies that the line data for addr x was unmodified.
   (g) Verifies that the line data for addr x+0x10 was modified correctly.
   (h) Verifies that the line data for addr x+0x20 was unmodified.

   Upon error, the test loops through steps a - h with a constant test address. Possible error messages for this test are:

```
error1: Bad system memory where no write expected:
testaddr=0x00000000,memaddr=0x00000000,exp=0x00000000,obs=0x00000000

error2: Bad system memory where block write expected:
testaddr=0x00000000,memaddr=0x00000000,exp=0x00000000,obs=0x00000000

error3: Bad system memory where no write expected:
testaddr=0x00000000,memaddr=0x00000000,exp=0x00000000,obs=0x00000000
```

The diagram below summarizes the LED states.

Test Number	*Hexadecimal Value Of LEDs*	*Visual Representation*	*Condition*
13	**0x0d**	**bit 7 oooo●●o● bit 0**	**okay**
13	**0x8d**	**●ooo●●o●**	**error**

Block Copy (Source=Cache Miss,Dest=Cache Hit) Test

This test verifies that doing a block copy read from memory followed by a block copy write to memory with a valid and clean Central Cache entry for the destination causes the proper blocks of data to be transferred in memory and the Central Cache block entry invalidated. The test does the following:

1. Turns off EN_CACHE,EN_IOCACHE,EN_DVMA,EN_VME_LOOP in the System Enable Register.
2. Clears Central Cache tags, data, and the first 64 Kbytes of system memory.
3. For each test address 0x0,0x1,0x2,0x4....0x08000000 (128MB), or up to maximum memory installed, the test:
   (a) Writes the line for addr x (test addr) in memory with 00-0f data.
   (b) Writes the line for addr x+0x10 in memory with ff-f0 data.
   (c) Writes the line for addr x+0x20 in memory with all zero data.
   (d) Writes the Central Cache line for addr x+0x10 with 50-5f data.
   (e) Writes the Central Cache tag for addr x+0x10 valid, clean.
   (f) Turns on Central Cache.
   (g) Does a block copy read from addr x.
   (h) Does a block copy write to addr x+0x10.
   (i) Turns off Central Cache.
   (j) Verifies that the line data for addr x was unmodified.
   (k) Verifies that the line data for addr x+0x10 was modified correctly.
   (l) Verifies that the line data for addr x+0x20 was unmodified.
   (m) Verifies that the Central Cache tag for addr x+0x10 was invalidated.

   Upon error, the test loops through steps a - m with a constant test address. Possible error messages for this test are:

```
error1: Bad system memory where no write expected:
testaddr=0x00000000,memaddr=0x00000000,exp=0x00000000,obs=0x00000000

error2: Bad system memory where block write expected:
testaddr=0x00000000,memaddr=0x00000000,exp=0x00000000,obs=0x00000000

error3: Bad system memory where no write expected:
testaddr=0x00000000,memaddr=0x00000000,exp=0x00000000,obs=0x00000000

error4: Central Cache tag not invalidated:
testaddr=0x00000000,tagaddr=0x00000000,exp=0x00000000,obs=0x00000000
```

The LED display is the same as that shown for the Block Copy (Source=Cache Miss,Dest=Cache Miss) Test.

Block Copy (Source=Cache Hit,Dest=Cache Miss) Test

This test verifies that doing a block copy read from memory followed by a block copy write to memory with a valid and clean Central Cache entry for the source causes the proper block of data to be transferred from Central Cache to memory and that the Central Cache block entry for the source is unchanged. The test does the following:

Turns off EN_CACHE,EN_IOCACHE,EN_DVMA,EN_VME_LOOP in the System Enable Register.

Clears Central Cache tags, data, and the first 64 Kbytes of system memory.

For each test address 0x0,0x10,0x20,0x40,...,0x08000000 (128MB), or up to maximum memory installed, the test:
(a) Writes the line for addr x (test addr) in memory with 00-0f data.
(b) Writes the line for addr x+0x10 in memory with 50-5f data.
(c) Writes the line for addr x+0x20 in memory with all zero data.
(d) Writes the Central cache line for addr x with ff-f0 data.
(e) Writes the Central cache line for addr x+0x10 with af-a0 data.
(f) Writes the Central cache tag for addr x valid, clean.
(g) Turns on Central Cache.
(h) Does a block copy read from addr x.
(i) Does a block copy write to addr x+0x10.
(j) Turns off Central Cache.
(k) Verifies that the line data for addr x was unmodified.
(l) Verifies that the line data for addr x+0x10 was modified correctly.
(m) Verifies that the line data for addr x+0x20 was unmodified.
(n) Verifies that the Central Cache tag for line addr x was unchanged.

Upon error, the test loops through steps a - n with a constant test address. Possible error messages for this test are:

```
error1: Bad system memory where no write expected:
testaddr=0x00000000,memaddr=0x00000000,exp=0x00000000,obs=0x00000000

error2: Bad system memory where block write expected:
testaddr=0x00000000,memaddr=0x00000000,exp=0x00000000,obs=0x00000000

error3: Bad system memory where no write expected:
testaddr=0x00000000,memaddr=0x00000000,exp=0x00000000,obs=0x00000000

error4: Central Cache tag UNEXPECTEDLY invalidated:
testaddr=0x00000000,tagaddr=0x00000000,exp=0x00000000,obs=0x00000000
```

The diagram below summarizes the LED states.

Test Number	*Hexadecimal Value Of LEDs*	*Visual Representation*	*Condition*
14	0x0e	bit 7 oooo●●●o bit 0	okay
14	0x8e	●ooo●●●o	error

Block Copy (Source=Cache Hit,Dest=Cache Hit) Test

This test verifies that doing a block copy read from memory followed by a block copy write to memory with the a valid and clean Central Cache entry for the source and destination causes the proper block of data to be transferred from Central Cache to memory and that the Central Cache block entry for the destination address is invalidated. The test does the following:

1. Turns off EN_CACHE,EN_IOCACHE,EN_DVMA,EN_VME_LOOP in the System Enable Register.
2. Clears Central Cache tags, data, and the first 64 Kbytes of system memory.
3. For each test address 0x0,0x10,0x20,0x40,...,0x08000000 (128MB), or up to maximum memory installed, the test:
   (a) Writes the line for addr x (test addr) in memory with 00-0f data.
   (b) Writes the line for addr x+0x10 in memory with 50-5f data.
   (c) Writes the line for addr x+0x20 in memory with all zero data.
   (d) Writes the Central cache line for addr x with ff-f0 data.
   (e) Writes the Central cache line for addr x+0x10 with af-a0 data.
   (f) Writes the Central cache tag for addr x,x+0x10 valid, clean.
   (g) Turns on Central Cache.
   (h) Does a block copy read from addr x.
   (i) Does a block copy write to addr x+0x10.
   (j) Turns off Central Cache.
   (k) Verifies that the line data for addr x was unmodified.
   (l) Verifies that the line data for addr x+0x10 was modified correctly.
   (m) Verifies that the line data for addr x+0x20 was unmodified.
   (n) Verifies that the Central Cache tag for addr x is unchanged.
   (o) Verifies that the Central Cache tag for addr x+0x10 was invalidated.

   Possible error messages for this test are:

```
error1: Bad system memory where no write expected:
testaddr=0x00000000,memaddr=0x00000000,exp=0x00000000,obs=0x00000000

error2: Bad system memory where block write expected:
testaddr=0x00000000,memaddr=0x00000000,exp=0x00000000,obs=0x00000000

error3: Bad system memory where no write expected:
testaddr=0x00000000,memaddr=0x00000000,exp=0x00000000,obs=0x00000000

error4: Central Cache tag UNEXPECTEDLY invalidated:
testaddr=0x00000000,tagaddr=0x00000000,exp=0x00000000,obs=0x00000000

error5: Central Cache tag not invalidated:
testaddr=0x00000000,tagaddr=0x00000000,exp=0x00000000,obs=0x00000000
```

The LED display for this test is the same as that for the Block Copy (Source=Cache Hit,Dest=Cache Miss) Test.

Memory Write/Write/Read Read Test (Central Cache on)

This test verifies the address and data paths to Main Memory, as well as address bit and data bit uniqueness.

For each test address of main memory:
(0x0,0x1,0x2,0x4,0x8,0x10,0x20....size_of_mem_installed)

For each data pattern at each test address:
(0x0,0x1,0x2,0x4,0x10,0x20....0x80000000)
The test does the following:
(a) Writes test data to test address.
(b) Writes inverted test data to test address + 0x04.
(c) Reads back data from test address and compares.

CAUTION **THIS RAM TEST IS BEING DONE WITH CENTRAL CACHE ENABLED!**

Upon error: loop through steps a - c with constant test data and a constant test address. This test enters a scope loop on data compare errors, with the following message:

```
addr xxxxxxxx, exp xxxxxxxx, obs xxxxxxxx
```

The diagram below summarizes the LED states.

Test Number	*Hexadecimal Value Of LEDs*	*Visual Representation*	*Condition*
15	**0x0f**	**bit 7 oooo●●●● bit 0**	**okay**
15	**0x8f**	**●ooo●●●●**	**error**

IOC Tag RAM Write/Write/Read Test

This test performs three passes of write/write/read testing on the entire IOC Tag RAM. Each pass consists of writing a longword test pattern to the address under test, then the 1's complement of that pattern to the next long word, and finally, reading the original long word back and comparing it with what was written. The test address is then incremented by one longword and the process is repeated until the end of RAM is reached. The first pass is done with a test pattern of 0x5a972c5a, the second pass is done with a test pattern of 0x5a5a972c, and the third pass is done with a test pattern of 0x2c5a5a97.

This test enters a scope loop on data compare errors, with the following terminal message:

```
addr xxxxxxxx, exp xxxxxxxx, obs xxxxxxxx
```

The diagram below summarizes the LED states.

Test Number	*Hexadecimal Value Of LEDs*	*Visual Representation*	*Condition*
16	**0x10**	**bit 7 ooo●oooo bit 0**	**okay**
16	**0x90**	**●oo●oooo**	**error**

IOC Tag RAM Address Test

This test writes the complete I/O Cache Tag RAM address space with the long-word address as the data, then reads back the entire address space and verifies that no addresses are overwritten. This is a test for addressing uniqueness of the I/O Cache Tag RAM.

This test enters a scope loop on data compare errors, with the following message:

```
addr xxxxxxxx, exp xxxxxxxx, obs xxxxxxxx
```

The LED display is the same as that shown for the IOC Tag RAM Write/Write/Read Test.

IOC Tag RAM 3-Pattern Test

This test writes the complete I/O Cache Tag RAM address space with a repeated three long word pattern sequence, then reads back the entire address space and verifies the data.

This test enters a scope loop on data compare errors, with the following message:

```
addr xxxxxxxx, exp xxxxxxxx, obs xxxxxxxx
```

The LED display is the same as that shown for the IOC Tag RAM Write/Write/Read Test.

IOC Data RAM Write/Write/Read Test

This test performs 3 passes of write/write/read testing on the entire IOC Data RAM. Each pass consists of writing a longword test pattern to the address under test, then the 1's complement of that pattern to the next long word, and finally, reading the original long word back and comparing it with what was written. The test address is then incremented by one longword and the process is repeated until the end of RAM is reached. The first pass is done with a test pattern of 0x5a972c5a, the second pass is done with a test pattern of 0x5a5a972c, and the third pass is done with a test pattern of 0x2c5a5a97.

This test enters a scope loop on data compare errors, with the following terminal message:

```
addr xxxxxxxx, exp xxxxxxxx, obs xxxxxxxx
```

The diagram below summarizes the LED states.

Test Number	*Hexadecimal Value Of LEDs*	*Visual Representation*	*Condition*
17	**0x11**	**bit 7 ooo•ooo• bit 0**	**okay**
17	**0x91**	**•oo•ooo•**	**error**

IOC Data RAM Address Test

This test writes the complete I/O Cache Data RAM address space with the long-word address as the data, then reads back the entire address space and verifies that no addresses are overwritten. This is a test for addressing uniqueness of the I/O Cache Data RAM.

This test enters a scope loop on data compare errors, with the following message:

```
addr xxxxxxxx, exp xxxxxxxx, obs xxxxxxxx
```

The LED display is the same as that shown for the IOC Data RAM Write/Write/Read Test.

IOC Data RAM 3-Pattern Test

This test writes the complete I/O Cache Data RAM address space with a repeated three-long-word pattern sequence, then reads back the entire address space and verifies the data.

This test enters a scope loop on data compare errors, with the following message:

```
addr xxxxxxxx, exp xxxxxxxx, obs xxxxxxxx
```

The LED display is the same as that shown for the IOC Data RAM Write/Write/Read Test.

IOC Data RAM Read Byte Alignment Test

This test verifies that byte, word, and long-word read operations produce the appropriate byte-aligned data. The test enters a scope loop on data compare errors, with the following terminal message:

```
byte misalignment: addr xxxxxxxx, exp xxxxxxxx, obs xxxxxxxx
```

The LED display is the same as that shown for the IOC Data RAM Write/Write/Read Test.

IOC Data RAM Write Byte Alignment Test

This test verifies that byte, word, and long word read operations produce the appropriate byte-aligned data for various byte-aligned write operations. The test enters a scope loop on data compare errors, with the following terminal message:

```
byte misalignment: addr xxxxxxxx, exp xxxxxxxx, obs xxxxxxxx
```

The LED display is the same as that shown for the IOC Data RAM Write/Write/Read Test.

VME Loopback Test

This test verifies that the VME loopback function works for writes and reads. The test does the following:

1. Turns off EN_CACHE,EN_IOCACHE,EN_DVMA,EN_VME_LOOP in the System Enable Register.
2. For each test address (0x0,0x4,0x8,0x10,0x20,0x00040000....0x00080000) and for each data pattern (0x0,0x1,0x2,,...,0x80000000) at each test address, the test:
   (a) Turns on VME-loopback in System Enable Register.
   (b) Writes the address, using DMVA offset.
   (c) Reads the address, using DVMA offset.
   (d) Reads the address again, using DVMA offset.
   (e) Turns off VME-loopback in System Enable Register.
   (f) Verifies that the data read is the same as the data that was written.

   Upon error: loop through steps a - f with a constant address and data. Possible error messages for this test are:

```
error1: testaddr=0x00000000,testdata=0x00000000,exp=0x00000000,obs=0x00000000
```

The diagram below summarizes the LED states.

Test Number	*Hexadecimal Value Of LEDs*	*Visual Representation*	*Condition*
18	**0x12**	**bit 7 ooo•oo•o bit 0**	**okay**
18	**0x92**	**•oo•oo•o**	**error**

VME Loopback and DVMA Test

This test is not used on a Sun-3/460 workstation.

This test verifies that the VME loopback function works for DVMA writes and reads.

The test does the following:

1. Turns off EN_CACHE,EN_IOCACHE,EN_DVMA,EN_VME_LOOP in System Enable Register.
2. For each test address (0x0,0x4,0x8,0x10,0x20,0x00040000....0x00080000) and for each data pattern (0x0,0x1,0x2....0x80000000) at each test address, the test does the following:
   (a) Writes IO Mapper entry for test address.
   (b) Turns on DVMA, VME-loopback in System Enable Register.
   (c) Writes the address, using DMVA offset.
   (d) Reads the address, using DVMA offset.
   (e) Turns off DVMA in System Enable Register.
   (f) Reads the address again, using DVMA offset.
   (g) Turns off VME-loopback in System Enable Register.
   (h) Verifies that the data read is the same as the data that was written.

   Upon error, the test loops through steps a - h with constant addresses and data. Possible error messages for this test are:

```
error1: testaddr=0x00000000,testdata=0x00000000,exp=0x00000000,obs=0x00000000
```

The diagram below summarizes the LED states.

Test Number	*Hexadecimal Value Of LEDs*	*Visual Representation*	*Condition*
19	**0x13**	**bit 7 ooo•oo••bit 0**	**okay**
19	**0x93**	**•oo•oo••**	**error**

IOC Read Hit Test

This test verifies that a data operand read from system memory with a valid memory block address in the IO Cache causes a read from the IO Cache and not from system memory. The test does the following:

1. Turns off EN_CACHE,EN_IOCACHE,EN_DVMA,EN_VME_LOOP in the System Enable Register.
2. Clears IOC data, tag RAM.
3. Clears Central Cache tags, data, and 1st 64KB of system memory.
4. For each address longword aligned address within each of 128 pages (0x0,0x4,0x8,0x10....0x1000,0x2000,0x2004....0x3000....0x000f1000). the test does the following:

   (a) Write the longword address as data into memory.
   (b) Write the inverted longword address as data into cache data ram.
   (c) Write the address, valid bit ON in cache tag ram.
   (d) Write IO Mapper entry for test address.
   (e) Turn on IO Cache, DVMA, VME-loopback in System Enable Register.
   (f) Read the address, using DVMA offset.
   (g) Turn off IO Cache, DVMA in System Enable Register.
   (h) Read the address again, using DVMA offset.
   (i) Turn off VME-loopback in System Enable Register.
   (j) Verify that the data read is the inverse of the address, and therefore, is not read from system memory, but rather from cache.
   (k) Verify cache tag is still correct (valid,unmodified).
   (l) Verify cache data is still correct (unmodified).

   Upon error, the test loops through steps a - l with a constant test address. Possible error messages for this test are:

```
error1: Bad read data on cache valid read:
testaddr=0x00000000,dataramaddr=0x00000000,exp=0x00000000,obs=0x00000000

error2: Bad IOC tag ram after cache valid read:
testaddr=0x00000000,tagramaddr=0x00000000,exp=0x00000000,obs=0x00000000

error3: Bad IOC data ram after cache valid read:
testaddr=0x00000000,dataramaddr=0x00000000,exp=0x00000000,obs=0x00000000
```

The diagram below summarizes the LED states.

Test Number	*Hexadecimal Value Of LEDs*	*Visual Representation*	*Condition*
20	**0x14**	**bit 7 ooo•o•oo bit 0**	**okay**
20	**0x94**	**•oo•o•oo**	**error**

IOC Invalid Read Miss Test

This test verifies that a data operand read from system memory with an invalid memory block address in the IO Cache causes a read from system memory and not from the IO Cache. The test does the following:

1. Turns off EN_CACHE,EN_IOCACHE,EN_DVMA,EN_VME_LOOP in System Enable Register.
2. Clears IOC data, tag RAM.
3. Clears Central Cache tags, data, and 1st 64KB of system memory.
4. For each address longword aligned address within each of 128 pages (0x0,0x4,0x8,0x10....0x1000,0x2000,0x2004....0x3000....0x000f1000), the test does the following:
   (a) Writes 00-0f as data into memory for cache line of interest.
   (b) Writes ff-f0 as data into cache data ram for cache line of interest.
   (c) Writes the address, valid bit OFF in cache tag ram.
   (d) Writes IO Mapper entry for test address.
   (e) Turns on IO Cache, DVMA, VME-loopback in System Enable Register.
   (f) Reads the address, using DVMA offset.
   (g) Turns off IO Cache, DVMA in System Enable Register.
   (h) Reads the address again, using DVMA offset.
   (i) Turns off VME-loopback in System Enable Register.
   (j) Verifies that the data read is the same as the address, and therefore is not read from io cache, but rather from system memory.
   (k) Verifies cache tag is updated correctly.
   (l) Verifies cache data is updated correctly (entire line).

   Upon error: loop through steps a - l with a constant test address. Possible error messages for this test are:

```
error1: Bad read data on cache invalid read:
testaddr=0x00000000,dataramaddr=0x00000000,exp=0x00000000,obs=0x00000000

error2: Bad IOC tag ram after cache invalid read:
testaddr=0x00000000,tagramaddr=0x00000000,exp=0x00000000,obs=0x00000000

error3: Bad IOC data ram after cache invalid read:
testaddr=0x00000000,dataramaddr=0x00000000,exp=0x00000000,obs=0x00000000
```

The LED display is the same as that for the IOC Read Hit Test.

IOC Write Hit Test

This test verifies that a data operand write to system memory with a valid memory block address in the IO Cache causes a write to the IO Cache and not to system memory. The test does the following:

1. Turns off EN_CACHE,EN_IOCACHE,EN_DVMA,EN_VME_LOOP in the System Enable Register.
2. Clears IOC data, tag RAM.
3. Clears Central Cache tags, data, and the first 64KB of system memory.
4. For each address longword aligned address within each of 128 pages (0x0,0x4,0x8,0x10,....0x1000,0x2000,0x2004,....0x3000,...0x000f1000). the test does the following:
   (a) Write 00-0f (16 bytes) into memory for cache line of interest.
   (b) Write all zeros into cache data ram for cache line of interest.
   (c) Write the address, valid bit ON in cache tag ram.
   (d) Write IO Mapper entry for test address.
   (e) Turn on IO Cache, DVMA, VME-loopback in System Enable Register.
   (f) Write address (with DVMA offset) with inverse address as data.
   (g) Turn off IO Cache, DVMA, VME-loopback in System Enable Register.
   (h) Verify that system memory location was unmodified, since the write should have gone only to the cache (no write-through).
   (i) Verify cache tag was updated correctly (valid,dirty,addr bits).
   (j) Verify cache data was written correctly.

   Upon error: loop through steps a - j with a constant test address. Possible error messages for this test are:

```
error1: Bad system memory after cache valid write hit:
testaddr=0x00000000,memaddr=0x00000000,exp=0x00000000,obs=0x00000000

error2: Bad IOC tag ram after cache valid write hit:
testaddr=0x00000000,tagramaddr=0x00000000,exp=0x00000000,obs=0x00000000

error3: Bad IOC data ram after cache valid write hit:
testaddr=0x00000000,dataramaddr=0x00000000,exp=0x00000000,obs=0x00000000
```

The LED display is the same as that for the IOC Read Hit Test.

IOC Write Miss, No Writeback Test

This test verifies that a data operand write to system memory with an invalid memory block address in the IO cache causes a write to IO cache and not to system memory. The test does the following:

1. Turns off EN_CACHE,EN_IOCACHE,EN_DVMA,EN_VME_LOOP in the System Enable Register.
2. Clears IOC data, tag RAM.
3. Clears Central Cache tags, data, and the first 64KB of system memory.
4. For each address longword aligned address within each of 128 pages (0x0,0x4,0x8,0x10,...0x1000,0x2000,0x2004,...0x3000,...0x000f1000), the test does the following:

   (a) Writes 00-0f (16 bytes) into memory for cache line of interest.
   (b) Writes all zeros into cache data ram for cache line of interest.
   (c) Writes the address, valid bit OFF in cache tag ram.
   (d) Writes IO Mapper entry for test address.
   (e) Turns on IO Cache, DVMA, VME-loopback in System Enable Register.
   (f) Writes address (with DVMA offset) with inverse address as data.
   (g) Turns off IO Cache, DVMA, VME-loopback in System Enable Register.
   (h) Verifies that system memory location was unmodified, since the write should have gone only to the cache (no write-through).
   (i) Verifies cache tag was updated correctly (valid,dirty,addr bits).
   (j) Verifies cache data was written correctly.

   Upon error: loop through steps a - j with a constant test address. Possible error messages for this test are:

```
error1: Bad system memory after cache write miss:
testaddr=0x00000000,memaddr=0x00000000,exp=0x00000000,obs=0x00000000

error2: Bad IOC tag ram after cache write miss:
testaddr=0x00000000,tagramaddr=0x00000000,exp=0x00000000,obs=0x00000000

error3: Bad IOC data ram after cache write miss:
testaddr=0x00000000,dataramaddr=0x00000000,exp=0x00000000,obs=0x00000000
```

The LED display is the same as that for the IOC Read Hit Test.

IOC Write Miss, Writeback Test

This test verifies that a data operand write to system memory with a different, valid and dirty memory block address in the IO cache causes a writeback to system memory.

The test does the following:

1. Turns off EN_CACHE,EN_IOCACHE,EN_DVMA,EN_VME_LOOP in the System Enable Register.
2. Clears IOC data, tag RAM.
3. Clears the Central Cache tags, data, and the first 64KB of system memory.
4. For each address longword aligned address within each of 128 pages (0x0,0x4,0x8,0x10,...0x1000,0x2000,0x2004,...0x3000,...0x000f1000) the test:

   (a) Writes the line for memory address x (writeback addr) with ff-f0 data.
   (b) Writes the line for memory address x+0x10 with zero data.
   (c) Writes the line in data cache for addr x with 00-0f data.
   (d) Writes the address (x), VALID, DIRTY in cache tag RAM.
   (e) Writes an IO Mapper entry for the test address.
   (f) Turns on IO Cache, DVMA, VME-loopback in the System Enable Reg.
   (g) Writes the address (with DVMA offset) with inverse address as data.
   (h) Turns off the IO Cache, DVMA, VME-loopback in System Enable Reg.
   (i) Verifies that the line for addr x was written back to memory.
   (j) Verifies that system memory for location (x+0x10) was unmodified.
   (k) Verifies that the cache tag was updated correctly (valid,dirty,addr bits).
   (l) Verifies that cache data was written correctly.

   Upon error, the test loops through steps a - l with a constant test address. Possible error messages for this test are:

```
error1: Bad system memory where writeback should have occurred:
testaddr=0x00000000,memaddr=0x00000000,exp=0x00000000,obs=0x00000000

error2: Bad system memory where NO writeback should have occurred:
testaddr=0x00000000,memaddr=0x00000000,exp=0x00000000,obs=0x00000000

error3: Bad IOC tag ram after cache write miss:
testaddr=0x00000000,tagramaddr=0x00000000,exp=0x00000000,obs=0x00000000

error4: Bad IOC data ram after cache write miss:
testaddr=0x00000000,dataramaddr=0x00000000,exp=0x00000000,obs=0x00000000
```

The LED display is the same as that for the IOC Read Hit Test.

IOC Read Miss, Writeback Test

This test Verifies that a data operand read from system memory with a different valid and dirty memory block address in the IO cache causes a writeback to system memory. The test does the following:

1. Turns off EN_CACHE,EN_IOCACHE,EN_DVMA,EN_VME_LOOP in the System Enable Register.
2. Clears IOC data, tag RAM.
3. Clears Central Cache tags, data, and the first 64KB of system memory.
4. For each address longword aligned address within each of 128 pages (0x0,0x4,0x8,0x10,...0x1000,0x2000,0x2004,...0x3000,...0x000f1000).

   the test does the following:

   (a) Writes the line for memory address x (writeback addr) with ff-f0 data.
   (b) Write the line for memory address x+0x10 with 50-5f data.
   (c) Writes the line in data cache for addr x with 00-0f data.
   (d) Writes the address (x), VALID, DIRTY in cache tag ram.
   (e) Writes the IO Mapper entry for the test address.
   (f) Turns on IO Cache, DVMA, VME-loopback in System Enable Register.
   (g) Reads from address x+0x10 (with DVMA offset).
   (h) Turns off IO Cache, DVMA in System Enable Register.
   (i) Reads from address x+0x10 (with DVMA offset).
   (j) Turns off VME-loopback in System Enable Register.
   (k) Verifies data read in was from memory, not cache.
   (l) Verifies that the line for addr x was written back to memory.
   (m) Verifies that the system memory line for addr (x+0x10) was unmodified.
   (n) Verifies that the cache tag was updated correctly (valid,addr bits).
   (o) Verifies that cache data was written correctly.

   Upon error, the test loops through steps a - o with a constant test address. Possible error messages for this test are:

```
error1: Bad read data on cache valid read miss:
testaddr=0x00000000,memaddr=0x00000000,exp=0x00000000,obs=0x00000000

error2: Bad system memory where writeback should have occurred:
testaddr=0x00000000,memaddr=0x00000000,exp=0x00000000,obs=0x00000000

error3: Bad system memory where NO writeback should have occurred:
testaddr=0x00000000,memaddr=0x00000000,exp=0x00000000,obs=0x00000000

error4: Bad IOC tag ram after cache valid read miss:
testaddr=0x00000000,tagramaddr=0x00000000,exp=0x00000000,obs=0x00000000

error5: Bad IOC data ram after cache valid read miss:
testaddr=0x00000000,dataramaddr=0x00000000,exp=0x00000000,obs=0x00000000
```

The LED display is the same as that for the IOC Read Hit Test.

IOC Valid Write Hit (Central Cache Match,Unmod) Test

This test Verifies that a data operand write to system memory with a valid memory block address in the IO Cache *and* in the Central Cache causes a write to the IO Cache and not to system memory, and that the entry in the Central Cache is invalidated. The test does the following:

1. Turns off EN_CACHE,EN_IOCACHE,EN_DVMA,EN_VME_LOOP in the System Enable Register.
2. Clears IOC data, tag RAM.
3. Clears Central Cache tags, data, and the first 64KB of system memory.
4. For each longword aligned address within each of 128 pages (0x0,0x4,0x8,0x10,...0x1000,0x2000,0x2004,...0x3000,...0x000f1000).

   the test does the following:

   (a) Writes the line for memory address x with 00-0f data.
   (b) Writes the line in IO cache for addr x with all zero data.
   (c) Writes the address (x), VALID in IO cache tag ram.
   (d) Writes the address (x), VALID in Central cache tag ram.
   (e) Writes the IO Mapper entry for the test address.
   (f) Turns on Central Cache, IO Cache, DVMA, VME-loopback in the System Enable register.
   (g) Writes to the test address (with DVMA offset) with the inverted address as data.
   (h) Turns off the Central Cache, IO Cache, DVMA, and VME-loopback in the System Enable register.
   (i) Verifies that the system memory line for addr x was unmodified.
   (j) Verifies that the IO cache tag was updated correctly (valid,dirty, address bits).
   (k) Verifies that the IO cache data was written correctly.
   (l) Verifies that the Central Cache tag was updated correctly (invalid).

   Upon error, the test loops through steps a - l with a constant test address. Possible error messages for this test are:

```
error1: Bad system memory after cache valid write hit:
testaddr=0x00000000,memaddr=0x00000000,exp=0x00000000,obs=0x00000000

error2: Bad IOC tag ram after cache valid write hit:
testaddr=0x00000000,tagramaddr=0x00000000,exp=0x00000000,obs=0x00000000

error3: Bad IOC data ram after cache valid write hit:
testaddr=0x00000000,dataramaddr=0x00000000,exp=0x00000000,obs=0x00000000

error4: Bad Central Cache tag ram after cache valid write hit:
testaddr=0x00000000,tagramaddr=0x00000000,exp=0x00000000,obs=0x00000000
```

The LED display is the same as that for the IOC Read Hit Test.

IOC Invalid Write Miss (Central Cache Match,Unmod) Test

This test verifies that a data operand write to system memory with a memory block address that is INVALID in in the IO Cache and VALID in the Central Cache causes a write to IO Cache and not to system memory, and that the entry in the Central Cache is invalidated. The test does the following:

1. Turns off EN_CACHE,EN_IOCACHE,EN_DVMA, and EN_VME_LOOP in the System Enable Register.
2. Clears IOC data, tag RAM.
3. Clears Central Cache tags, data, and the first 64KB of system memory.
4. For each longword aligned address within each of 128 pages (0x0,0x4,0x8,0x10,...0x1000,0x2000,0x2004,...0x3000,...0x000f1000).

   the test does the following:

   (a) Writes the system memory line for addr x with 00-0f data.
   (b) Writes the line in IO cache for addr x with all zero data.
   (c) Writes the address (x), INVALID in IO cache tag ram.
   (d) Writes the address (x), VALID in Central cache tag ram.
   (e) Writes the IO Mapper entry for the test address.
   (f) Turns on Central Cache, IO Cache, DVMA, VME-loopback in the System Enable register.
   (g) Writes to the test address (with DVMA offset) with the inverted address as data.
   (h) Turns off Central Cache, IO Cache, DVMA, VME-loopback in the System Enable register.
   (i) Verifies that the system memory line for addr x was unmodified.
   (j) Verifies that the IO cache tag was updated correctly (valid,dirty,address bits).
   (k) Verifies that IO cache data was written correctly.
   (l) Verifies that Central Cache tag was updated correctly (invalid).

   Upon error, the test loops through steps a - l with a constant test address. Possible error messages for this test are:

```
error1: Bad system memory after cache invalid write miss:
testaddr=0x00000000,memaddr=0x00000000,exp=0x00000000,obs=0x00000000

error2: Bad IOC tag ram after cache invalid write miss:
testaddr=0x00000000,tagramaddr=0x00000000,exp=0x00000000,obs=0x00000000

error3: Bad IOC data ram after cache invalid write miss:
testaddr=0x00000000,dataramaddr=0x00000000,exp=0x00000000,obs=0x00000000

error4: Bad Central Cache tag ram after cache invalid write miss:
testaddr=0x00000000,tagramaddr=0x00000000,exp=0x00000000,obs=0x00000000
```

The LED display is the same as that for the IOC Read Hit Test.

IOC Invalid Read Miss (Central Cache Match,Unmod) Test

This test verifies that a data operand read from system memory with the memory block address that is INVALID in the IO Cache and VALID in the Central Cache causes a read from Central Cache and not from IO cache or system memory. The test does the following:

1. Turns off EN_CACHE,EN_IOCACHE,EN_DVMA, and EN_VME_LOOP in the System Enable Register.
2. Clears IOC data, tag RAM.
3. Clears Central Cache tags, data, and the first 64KB of system memory.
4. For each longword aligned address within each of 128 pages (0x0,0x4,0x8,0x10,...0x1000,0x2000,0x2004,...0x3000,...0x000f1000).

   the test does the following:

   (a) Writes the system memory line for addr x with ff-f0 data.
   (b) Writes the line in IO cache for addr x with all zero data.
   (c) Writes the address (x), INVALID in IO cache tag ram.
   (d) Writes the line in Central cache for addr x with 00-0f data.
   (e) Writes the address (x), VALID in Central cache tag ram.
   (f) Writes the IO Mapper entry for test address.
   (g) Turns on Central Cache, IO Cache, DVMA, VME-loopback in the System Enable register.
   (h) Reads from the test address (with DVMA offset).
   (i) Turns off Central Cache, IO Cache, DVMA in the System Enable register.
   (j) Reads from the test address again (with DVMA offset).
   (k) Turns off VME-loopback in the System Enable register.
   (l) Verifies that the data read in is correct (came from Central cache).
   (m) Verifies that the system memory line for addr x was unmodified.
   (n) Verifies that the IO cache tag was updated correctly (valid, clean, address bits).
   (o) Verifies that the IO cache data was written correctly.
   (p) Verifies that the Central cache tag was unchanged.

Upon error, the test loops through steps a - p with a constant test address. Possible error messages for this test are:

```
error1: Bad read data on cache invalid read:
testaddr=0x00000000,dataramaddr=0x00000000,exp=0x00000000,obs=0x00000000

error2: Bad system memory after cache invalid read miss:
testaddr=0x00000000,memaddr=0x00000000,exp=0x00000000,obs=0x00000000

error3: Bad IOC tag ram after cache invalid read miss:
testaddr=0x00000000,tagramaddr=0x00000000,exp=0x00000000,obs=0x00000000

error4: Bad IOC data ram after cache valid read miss:
testaddr=0x00000000,dataramaddr=0x00000000,exp=0x00000000,obs=0x00000000

error5: Bad Central Cache tag ram after cache invalid read miss:
testaddr=0x00000000,tagramaddr=0x00000000,exp=0x00000000,obs=0x00000000
```

The LED display is the same as that for the IOC Read Hit Test.

IOC Invalid Read Miss (Central Cache Match,Modified) Test

This test verifies that a data operand read from system memory with a memory block address that is INVALID in the IO Cache; VALID in the Central Cache; and MODIFIED causes a read from Central Cache and not from IO cache or system memory. The test does the following:

1. Turns off EN_CACHE,EN_IOCACHE,EN_DVMA, and EN_VME_LOOP in the System Enable Register.
2. Clears IOC data, tag RAM.
3. Clears Central Cache tags, data, and the first 64KB of system memory.
4. For each longword aligned address within each of 128 pages (0x0,0x4,0x8,0x10,...0x1000,0x2000,0x2004,...0x3000,...0x000f1000), the test does the following:

   (a) Writes the system memory line for addr x with ff-f0 data.
   (b) Writes the line in IO cache for addr x with all zero data.
   (c) Writes the address (x), INVALID in IO cache tag ram.
   (d) Writes the line in Central cache for addr x with 00-0f data.
   (e) Writes the address (x), VALID, DIRTY in Central cache tag ram.
   (f) Writes the IO Mapper entry for the test address.
   (g) Turns on Central Cache, IO Cache, DVMA and VME-loopback in the System Enable register.
   (h) Reads from the test address (with DVMA offset).
   (i) Turns off Central Cache, IO Cache, DVMA in the System Enable register.
   (j) Reads from the test address (with DVMA offset).
   (k) Turns off VME-loopback in System Enable register.
   (l) Verifies that the data read in is correct (came from Central cache).
   (m) Verifies that the system memory line for addr x was unmodified.
   (n) Verifies that the IO cache tag was updated correctly (valid,clean, address bits).
   (o) Verifies that IO cache data was written correctly.
   (p) Verifies that the Central cache tag was unchanged.
   (q) Verifies that the Central cache data was unchanged.

   Upon error, the test loops through steps a - q with a constant test address. Possible error messages for this test are:

```
error1: Bad read data on cache invalid read:
testaddr=0x00000000,dataramaddr=0x00000000,exp=0x00000000,obs=0x00000000

error2: Bad system memory after cache invalid read miss:
testaddr=0x00000000,memaddr=0x00000000,exp=0x00000000,obs=0x00000000

error3: Bad IOC tag ram after cache invalid read miss:
testaddr=0x00000000,tagramaddr=0x00000000,exp=0x00000000,obs=0x00000000

error4: Bad IOC data ram after cache valid read miss:
testaddr=0x00000000,dataramaddr=0x00000000,exp=0x00000000,obs=0x00000000

error5: Bad Central Cache tag ram after cache invalid read miss:
testaddr=0x00000000,tagramaddr=0x00000000,exp=0x00000000,obs=0x00000000
```

The LED display is the same as that for the IOC Read Hit Test.

IOC Valid Read Miss (Central Cache Match), Writeback Test

This test verifies that a data operand read from system memory with a memory block address that is VALID,MODIFIED in the IO Cache and another that is VALID in the Central Cache causes a read from Central Cache and not from IO cache or system memory. Also, a writeback of the valid and modified data in the IOC should take place. The test does the following:

1. Turns off EN_CACHE,EN_IOCACHE,EN_DVMA, and EN_VME_LOOP in the System Enable Register.
2. Clears IOC data, tag RAM.
3. Clears Central Cache tags, data, and the first 64KB of system memory.
4. For each longword aligned address within each of 128 pages (0x0,0x4,0x8,0x10,...0x1000,0x2000,0x2004,...0x3000,...0x000f1000), the test does the following:

   (a) Writes the system memory line for addr x with ff-f0 data.
   (b) Writes the system memory line for addr x+0x10 with all zero data.
   (c) Writes the line in IO cache for addr x with 50-5f data.
   (d) Writes the line in Central cache for addr x with 00-0f data.
   (e) Writes the address (x+0x10),VALID,DIRTY in IO Cache tag RAM.
   (f) Writes the address (x), VALID in Central Cache tag RAM.
   (g) Writes the IO Mapper entry for test address.
   (h) Turns on Central Cache, IO Cache, DVMA and VME-loopback in the System Enable register.
   (i) Reads from test address x (with DVMA offset).
   (j) Turns off Central Cache, IO Cache and DVMA in the System Enable register.
   (k) Reads from test address x (with DVMA offset).
   (l) Turns off VME-loopback in the System Enable register.
   (m) Verifies that data read in is correct (came from Central cache).
   (n) Verifies that the system memory line for addr x was unmodified.
   (o) Verifies that a writeback took place to system memory address x+0x10.
   (p) Verifies that the IO cache tag was updated correctly (valid,clean,address bits).
   (q) Verifies that the IO cache data was written correctly.
   (r) Verifies that the Central cache tag was unchanged.
   (s) Verifies that the Central cache data was unchanged.

Upon error, the test loop through steps a - s with a constant test address. Possible error messages for this test are:

```
error1: Bad read data on IOC valid read miss:
testaddr=0x00000000,readaddr=0x00000000,exp=0x00000000,obs=0x00000000

error2: Bad system memory where NO writeback expected:
testaddr=0x00000000,memaddr=0x00000000,exp=0x00000000,obs=0x00000000

error3: Bad system memory where WRITEBACK expected:
testaddr=0x00000000,memaddr=0x00000000,exp=0x00000000,obs=0x00000000

error4: Bad IOC tag ram after IOC valid read miss:
testaddr=0x00000000,tagramaddr=0x00000000,exp=0x00000000,obs=0x00000000

error5: Bad IOC data ram after IOC valid read miss:
testaddr=0x00000000,dataramaddr=0x00000000,exp=0x00000000,obs=0x00000000

error6: Bad Central Cache tag ram after IOC valid read miss:
testaddr=0x00000000,tagramaddr=0x00000000,exp=0x00000000,obs=0x00000000

error7: Bad Central Cache data ram after IOC valid read miss:
testaddr=0x00000000,dataramaddr=0x00000000,exp=0x00000000,obs=0x00000000
```

The LED display is the same as that for the IOC Read Hit Test.

IOC Flush (Valid, Modified) Test

This test verifies that a flush cache block write with valid and dirty data in the IO cache causes the line to be written back to memory and the entry in the IO cache to be invalidated. The test does the following:

1. Turns off EN_CACHE,EN_IOCACHE,EN_DVMA, and EN_VME_LOOP in the System Enable Register.
2. Clears IOC data, tag RAM.
3. Clears Central Cache tags, data, and the first 64KB of system memory.
4. For each line base address within each of 128 pages (0x0,0x4,0x8,0x10,...0x1000,0x2000,0x2004,...0x3000,...0x000f1000), the test does the following:

   (a) Writes the line for memory address x with 00-0f data.
   (b) Writes the line in data cache for addr x with ff-f0 data.
   (c) Writes the address (x), VALID, DIRTY in cache tag RAM.
   (d) Writes IO Mapper entry for test address.
   (e) Turns on IO Cache, DVMA, VME-loopback in the System Enable register.
   (f) Writes to the flush block addr for the test address.
   (g) Turns off IO Cache, DVMA and VME-loopback in the System Enable register.
   (h) Verifies that the line for addr x was written back to memory.
   (i) Verifies that the IOC tag was updated correctly (invalid).
   (j) Verifies that the IOC data RAM remained unchanged.

   Upon error, the test loops through steps a - j with a constant test address. Possible error messages for this test are:

```
error1: Bad system memory where writeback should have occurred:
testaddr=0x00000000,memaddr=0x00000000,exp=0x00000000,obs=0x00000000

error2: Bad IOC tag ram after IOC block flush:
testaddr=0x00000000,tagramaddr=0x00000000,exp=0x00000000,obs=0x00000000

error3: Bad IOC data ram after IOC block flush:
testaddr=0x00000000,dataramaddr=0x00000000,exp=0x00000000,obs=0x00000000
```

The LED display is the same as that for the IOC Read Hit Test.

IOC Flush (Valid, Not Modified) Test

This test verifies that a flush cache block write with valid and clean data in the IO cache causes the line NOT to be written back to memory and the IO cache entry to be invalidated. The test does the following:

1. Turns off EN_CACHE,EN_IOCACHE,EN_DVMA, and EN_VME_LOOP in the System Enable Register.
2. Clears IOC data, tag RAM.
3. Clears Central Cache tags, data, and the first 64KB of system memory.
4. For each line base address within each of 128 pages (0x0,0x4,0x8,0x10,...0x1000,0x2000,0x2004,...0x3000,...0x000f1000), the test does the following:
   (a) Writes the line for memory address x with 00-0f data.
   (b) Writes the line in data cache for addr x with ff-f0 data.
   (c) Writes the address (x), VALID, in cache tag ram.
   (d) Writes the IO Mapper entry for the test address.
   (e) Turns on the IO Cache, DVMA, VME-loopback in the System Enable register.
   (f) Writes to the flush block address for the test address.
   (g) Turns off the IO Cache, DVMA and VME-loopback in the System Enable register.
   (h) Verifies that the line for address x was NOT written back to memory.
   (i) Verifies that the cache tag was updated correctly (invalid).
   (j) Verifies that IOC data RAM remained unchanged.

   Upon error, the test loops through steps a - j with a constant test address. Possible error messages for this test are:

```
error1: Bad system memory where NO writeback should have occurred:
testaddr=0x00000000,memaddr=0x00000000,exp=0x00000000,obs=0x00000000

error2: Bad IOC tag ram after IOC block flush:
testaddr=0x00000000,tagramaddr=0x00000000,exp=0x00000000,obs=0x00000000

error3: Bad IOC data ram after IOC block flush:
testaddr=0x00000000,dataramaddr=0x00000000,exp=0x00000000,obs=0x00000000
```

The LED display is the same as that for the IOC Read Hit Test.

IOC Flush (Not Valid, Not Modified) Test

This test verifies that a flush cache block write with invalid data in the IO cache causes the NOT to be written back to memory and the entry in the IO cache to remain invalidated. The test does the following:

1. Turns off EN_CACHE,EN_IOCACHE,EN_DVMA, and EN_VME_LOOP in the System Enable Register.
2. Clears IOC data, tag RAM.
3. Clears Central Cache tags, data, and the first 64KB of system memory.
4. For each line base address within each of 128 pages (0x0,0x4,0x8,0x10,...0x1000,0x2000,0x2004,...0x3000,...0x000f1000), the test does the following:
   (a) Writes the line for memory address x with 00-0f data.
   (b) Writes the line in data cache for addr x with ff-f0 data.
   (c) Writes the address (x), INVALID, in cache tag ram.
   (d) Writes the IO Mapper entry for the test address.
   (e) Turns on the IO Cache, DVMA and VME-loopback in the System Enable register.
   (f) Writes to the flush block address for the test address.
   (g) Turns off the IO Cache, DVMA and VME-loopback in the System Enable register.
   (h) Verifies that the line for addr x was NOT written back to memory.
   (i) Verifies that the cache tag remained invalid.
   (j) Verifies that IOC data RAM remained unchanged.

   Upon error, the test loops through steps a - j with a constant test address. Possible error messages for this test are:

```
error1: Bad system memory where NO writeback should have occurred:
testaddr=0x00000000,memaddr=0x00000000,exp=0x00000000,obs=0x00000000

error2: Bad IOC tag ram after IOC block flush:
testaddr=0x00000000,tagramaddr=0x00000000,exp=0x00000000,obs=0x00000000

error3: Bad IOC data ram after IOC block flush:
testaddr=0x00000000,dataramaddr=0x00000000,exp=0x00000000,obs=0x00000000
```

The LED display is the same as that for the IOC Read Hit Test.

I/O Mapper Invalid Page (IO.DT) Test

This test verifies that a data operand read from system memory with the memory block address in the IO Cache INVALID and the IO mapper entry for that page set to INVALID causes an access violation, and that there is no data cached from system memory. For each of (IO.DT = 00, 10, 11), the test does the following:

1. Turns off EN_CACHE,EN_IOCACHE,EN_DVMA, and EN_VME_LOOP in the System Enable Register.
2. Clears IOC data, tag RAM.
3. Clears Central Cache tags, data, and the first 64KB of system memory.
4. For each longword aligned address within each of 128 pages (0x0,0x4,0x8,0x10,...0x1000,0x2000,0x2004,...0x3000,...0x000f1000), the test does the following:

   (a) Writes ff-f0 as data into memory for cache line of interest.
   (b) Writes 00-0f as data into cache data RAM for cache line of interest.
   (c) Writes the address, valid bit OFF in cache tag ram.
   (d) Writes the IO Mapper entry for the test address; sets INVALID page.
   (e) Turns on IO Cache, DVMA and VME-loopback in the System Enable register.
   (f) Reads the address, using DVMA offset.
   (g) Turns off IO Cache and DVMA in the System Enable register.
   (h) Reads the address, using DMVA offset.
   (i) Turns off the VME-loopback in System Enable register.
   (j) Verifies that the cache tag is unchanged.
   (k) Verifies that the cache data is unchanged.

   Upon error, the test loops through steps a - k with a constant test address. Possible error messages for this test are:

```
error1: Bad IOC tag ram after cache invalid (page invalid) read:
testaddr=0x00000000,tagramaddr=0x00000000,exp=0x00000000,obs=0x00000000

error2: Bad IOC data ram after cache invalid (page invalid) read:
testaddr=0x00000000,dataramaddr=0x00000000,exp=0x00000000,obs=0x00000000
```

The LED display is the same as that for the IOC Read Hit Test.

IOC Write Miss, Writeback (Write Protect) Test

This test verifies that a data operand write to system memory with a different memory block address in the IO cache valid and dirty and the IO mapper entry for the block's page set to Write Protect (IO.WP = 1) causes NO writeback to system memory. The test does the following:

1. Turns off EN_CACHE,EN_IOCACHE,EN_DVMA, and EN_VME_LOOP in the System Enable Register.
2. Clears IOC data, tag RAM.
3. Clears Central Cache tags, data, and the first 64KB of system memory.
4. For each longword aligned address within each of 128 pages (0x0,0x4,0x8,0x10,...0x1000,0x2000,0x2004,...0x3000,...0x000f1000), the test does the following:

   (a) Turns off IO cache.
   (b) Writes the line for memory address x (writeback addr) with ff-f0 data.
   (c) Writes the line for memory address x+0x10 with 50-5f data.
   (d) Writes the line in data cache for addr x with 00-0f data.
   (e) Writes the address (x), VALID, DIRTY in cache tag RAM.
   (f) Writes the IO Mapper entry for the test address.
   (g) Turns on the IO Cache, DVMA and VME-loopback in the System Enable register.
   (h) Writes the address (with DVMA offset) with inverse address as data.
   (i) Turns off IO Cache, DVMA and VME-loopback in the System Enable register.
   (j) Verifies that the line for addr x was NOT written back to memory.
   (k) Verifies that system memory for location (x+0x10) was unmodified.
   (l) Verifies that the cache tag was NOT updated.
   (m) Verifies that cache data was NOT updated.

   Upon error, the test loops through steps b - m with a constant test address. Possible error messages for this test are:

```
error1: Bad system memory where NO writeback should occur (write protect):
testaddr=0x00000000,memaddr=0x00000000,exp=0x00000000,obs=0x00000000

error2: Bad system memory where no write should have occurred (write protect):
testaddr=0x00000000,memaddr=0x00000000,exp=0x00000000,obs=0x00000000

error3: Bad IOC tag ram after cache write miss (write protect):
testaddr=0x00000000,tagramaddr=0x00000000,exp=0x00000000,obs=0x00000000

error4: Bad IOC data ram after cache write miss (write protect):
testaddr=0x00000000,dataramaddr=0x00000000,exp=0x00000000,obs=0x00000000
```

The LED display is the same as that for the IOC Read Hit Test.

IOC Invalid Read Miss (IO Mapper IO.EN = 0) Test

This test verifies that a data operand read from system memory with an invalid memory block address in the IO Cache *and* the IO.EN bit in the IO Mapper entry for this page set OFF causes a read from system memory and not from the IO Cache, but that nothing gets modified in the IOC data and tags. The test does the following:

1. Turns off EN_CACHE,EN_IOCACHE,EN_DVMA, and EN_VME_LOOP in the System Enable Register.
2. Clears IOC data, tag RAM.
3. Clears Central Cache tags, data, and the first 64KB of system memory.
4. For each longword aligned address within each of 128 pages (0x0,0x4,0x8,0x10,...0x1000,0x2000,0x2004,...0x3000,...0x000f1000), the test does the following:

   (a) Writes 00-0f as data into memory for cache line of interest.
   (b) Writes ff-f0 as data into cache data RAM for cache line of interest.
   (c) Writes the address, valid bit OFF in cache tag RAM.
   (d) Writes the IO Mapper entry for the test address, IO.EN = 0.
   (e) Turns on IO Cache, DVMA and VME-loopback in the System Enable register.
   (f) Reads the address, using DVMA offset.
   (g) Turns off IO Cache and DVMA in the System Enable register.
   (h) Reads the address again, using DMVA offset.
   (i) Turns off VME-loopback in the System Enable register.
   (j) Verifies that the data read is the same as the address, and therefore is not read from IO Cache, but rather from system memory.
   (k) Verifies that the IOC tag is unchanged (still invalid).

   Upon error, the test loops through steps a - k with a constant test address. Possible error messages for this test are:

```
error1: Bad read data on cache invalid read:
testaddr=0x00000000,dataramaddr=0x00000000,exp=0x00000000,obs=0x00000000

error2: Bad IOC tag ram after cache invalid read:
testaddr=0x00000000,tagramaddr=0x00000000,exp=0x00000000,obs=0x00000000
```

The LED display is the same as that for the IOC Read Hit Test.

IOC Write Miss (IO Mapper IO.EN = 0) Test

This test verifies that a data operand write to system memory with an invalid memory block address in the IO cache *and* the IO.EN bit in the IO Mapper entry for this page set OFF causes a write to system memory and not to the IOC. This test does the following:

1. Turns off EN_CACHE,EN_IOCACHE,EN_DVMA, and EN_VME_LOOP in the System Enable Register.
2. Clears IOC data, tag RAM.
3. Clears Central Cache tags, data, and the first 64KB of system memory.
4. For each longword aligned address within each of 128 pages (0x0,0x4,0x8,0x10,...0x1000,0x2000,0x2004,...0x3000,...0x000f1000), the test does the following:

   (a) Write all zeroes into memory line for test address.
   (b) Write 00-0f data into IOC data ram for cache line of interest.
   (c) Write the address, valid bit OFF in cache tag ram.
   (d) Write IO Mapper entry for test address, IO.EN = 0.
   (e) Turn on IO Cache, DVMA, VME-loopback in System Enable reg.
   (f) Write address (with DVMA offset) with inverse address as data.
   (g) Turn off IO Cache, DVMA, VME-loopback in System Enable reg.
   (h) Verifies that system memory location was modified, since the write should have gone to memory (cache disabled for this page).
   (i) Verifies that the cache tag was not modified (still invalid).

   Upon error, the test loops through steps a - i with a constant test address. Possible error messages for this test are:

```
error1: Bad system memory after cache write miss:
testaddr=0x00000000,memaddr=0x00000000,exp=0x00000000,obs=0x00000000

error2: Bad IOC tag ram after cache write miss:
testaddr=0x00000000,tagramaddr=0x00000000,exp=0x00000000,obs=0x00000000
```

The LED display is the same as that for the IOC Read Hit Test.

IOC Random Data Block Write Test

This test does the following:

1. Turns off EN_CACHE,EN_IOCACHE,EN_DVMA, and EN_VME_LOOP in the System Enable Register.
2. Clears IOC data, tag RAM.
3. Clears Central Cache tags, data, and the first 64KB of system memory.
4. For each (x) test address 0x0,0x4,0x8,0x10,0x20,0x40,0x80,...0x100000, the test:
   (a) Clears out all IOC tags, data.
   (b) Clears out the entire IO Mapper.
   (c) Writes eight IO Mapper entries for the 64MB space to be tested (x-->x+64ME with address,IO.DT,IO.EN.
   (d) Turns on IO Cache, DVMA and VME-loopback in the System Enable register.
   (e) Copies the entire 64KB EPROM contents to memory starting at the test address, doing DVMA writes through the IOC, flushing the last line of each 8K block.
   (f) Turns off the IO Cache, DVMA and VME-loopback in the System Enable register.
   (g) Verifies that 64KB of system memory matches EPROM contents.

   Upon error, the test loops through step (g) with a constant test address. Possible error messages for this test are:

```
error1: Bad system memory after block write:
testaddr=0x00000000,memaddr=0x00000000,exp=0x00000000,obs=0x00000000
```

The LED display is the same as that for the IOC Read Hit Test.

IOC Random Data Block Read (Central Cache off) Test

This test does the following:

1. Turns off EN_CACHE,EN_IOCACHE,EN_DVMA, and EN_VME_LOOP in the System Enable Register.
2. Clears IOC data, tag RAM.
3. Clears Central Cache tags, data, and the first 64KB of system memory.
4. For each (x) test address 0x0,0x4,0x8,0x10,0x20,0x40,0x80,...0x100000, the test:

```
(a) Clears out all IOC tags, data.
(b) Clears out entire IO Mapper.
(c) Writes eight IO Mapper entries for 64MB space to be tested (x-->x+64MB),
    with address,IO.DT,IO.EN.
(d) Copies the entire 64KB EPROM contents to memory, starting at the test address.
(e) Turns on IO Cache, DVMA and VME-loopback in the System Enable register.
(f) Reads 64KB of data back from memory by way of DVMA; reads one longword at
    a time and compares with contents of EPROM.
(g) Turns off IO Cache, DVMA and VME-loopback in the System Enable register.
```

Upon error, the test loops through step (f) with a constant test address. Possible error messages for this test are:

```
error1: Bad IOC read data:
testaddr=0x00000000,memaddr=0x00000000,exp=0x00000000,obs=0x00000000
```

The LED display is the same as that for the IOC Read Hit Test.

IOC Random Data Block Read (Central Cache on) Test

This test does the following: IP 1. Turns off EN_CACHE,EN_IOCACHE,EN_DVMA, and EN_VME_LOOP in the System Enable Register.

2. Clears IOC data, tag RAM.
3. Clears Central Cache tags, data, and the first 64KB of system memory.
4. For each (x) test address 0x0,0x4,0x8,0x10,0x20,0x40,0x80,...0x100000, the test:

```
(a) Clears out all IOC tags, data.
(b) Clears out entire IO Mapper.
(c) Writes eight IO Mapper entries for 64MB space to be tested (x-->x+64MB),
    with address,IO.DT,IO.EN.
(d) Copies the entire 64KB EPROM contents to memory, starting at test address.
(e) Turns on IO Cache, DVMA, VME-loopback, Central Cache in SYSENREG.
(f) Reads 64KB of data back from memory, using DVMA, reading one longword at
    a time and comparing with EPROM contents.
(g) Turns off IO Cache, DVMA, VME-loopback and Central Cache in the
    System Enable register.
```

Upon error, the test loops through step (f) with a constant test address. Possible error messages for this test are:

```
error1: Bad IOC read data:
testaddr=0x00000000,memaddr=0x00000000,exp=0x00000000,obs=0x00000000
```

The LED display is the same as that for the IOC Read Hit Test.

P4 Overlay Frame Buffer Write/Write/Read Test

This test will be executed only if a P4 Video RAM board is detected. First, the test probes for a P4 board, with the following possible messages:

```
<P4 High Resolution Monochrome Video RAM Board Detected>

<P4 Low Resolution Monochrome Video RAM Board Detected>

<P4 Low Res Color Video RAM Board Detected>

<ILLEGAL P4 Video RAM Board ID Detected: 0x00000000>

<P4 Video RAM Board NOT Detected>
```

The same test is executed if either the monochrome or color board is detected. The test verifies the address and data paths to the P4 Overlay Frame Buffer, as well as address bit and data bit uniqueness.

For each test address of P4 Overlay Frame Buffer, the test does the following, for each data pattern at each test address (0x0,0x1,0x2,0x4,0x10,0x20,...0x80000000):
(a) Writes test data to test address.
(b) Writes inverted test data to test address + 0x04.
(c) Reads back data from test address and compares.

Upon error, the test loop through steps a - c with constant test data and a constant test address.

This test enters a scope loop on data compare errors, with the following message:

```
addr xxxxxxxx, exp xxxxxxxx, obs xxxxxxxx
```

The LED indications are as follows:

Test Number	*Hexadecimal Value Of LEDs*	*Visual Representation*	*Condition*
21	**0x15**	**bit 7 ooo•o•o• bit 0**	**okay**
21	**0x95**	**•oo•o•o•**	**error**

P4 Overlay Frame Buffer Address Test

Note that this test is only present in the diagnostic PROM code if the `P4VIDEORAM_ADDR_TEST` compile-time flag is on in the Makefile that generates the diagnostic executable code.

The same test is executed if either the monochrome or color board is detected.

This test writes the complete P4 Overlay Frame Buffer address space with the longword address as data, then reads back the entire address space and verifies that no addresses are overwritten. This is a test for addressing uniqueness of the P4 Overlay Frame Buffer RAM.

This test enters a scope loop on data compare errors, with the following message:

```
addr xxxxxxxx, exp xxxxxxxx, obs xxxxxxxx
```

The LED display is the same as that for the P4 Overlay Frame Buffer Write/Write/Read Test.

P4 Overlay Frame Buffer 3-Pattern Test

Note that this test is only present in the diagnostic PROM code if the `P4VIDEORAM_3PATT_TEST` compile-time flag is on in the Makefile that generates the diagnostic executable code.

The same test is executed if either the monochrome or color board is detected. This test writes the complete P4 Overlay Frame Buffer address space with a repeated three-long-word pattern sequence, then reads back the entire address space and verifies the data.

This test enters a scope loop on data compare errors, with the following message:

```
addr xxxxxxxx, exp xxxxxxxx, obs xxxxxxxx
```

The LED display is the same as that for the P4 Overlay Frame Buffer Write/Write/Read Test.

P4 Overlay Frame Buffer March Test

Note that this test is only present in the diagnostic PROM code if the `P4VIDEORAM_MARCH_TEST` compile-time flag is on in the Makefile that generates the diagnostic executable code.

The same test is executed if either the monochrome or color board is detected. A background of longword 0's is written to all of the P4 Overlay Frame Buffer RAM, starting from address 0 and ending at the highest address. Then the longword data is read at the first address and a 0xffffffff is written into this address. The same two-step read/write procedure is continued at each sequential longword until the end of memory is reached. Then each longword is tested and changed back to zero in reverse order until the first address is reached. Finally, the same sequence is repeated using complemented data (i.e. a background of 1's is written into memory).

Upon error, the test loops on a complete write then read of Video RAM. This test enters a scope loop on data compare errors, with the following message:

```
addr xxxxxxxx, exp xxxxxxxx, obs xxxxxxxx
```

The LED display is the same as that for the P4 Overlay Frame Buffer Write/Write/Read Test.

P4 Overlay Frame Buffer Read Byte Alignment Test

The same test is executed if either the monochrome or color board is detected. This test verifies that byte, word, and long word read operations produce the appropriate byte-aligned data.

This test enters a scope loop on data compare errors, with the following message:

```
byte misalignment: addr xxxxxxxx, exp xxxxxxxx, obs xxxxxxxx
```

The LED display is the same as that for the P4 Overlay Frame Buffer Write/Write/Read Test.

P4 Overlay Frame Buffer Write Byte Alignment Test

The same test is executed if either the monochrome or color board is detected. This test verifies that byte, word, and long word read operations produce the appropriate byte-aligned data for various byte-aligned write operations.

This test enters a scope loop on data compare errors, with the following message:

```
byte misalignment: addr xxxxxxxx, exp xxxxxxxx, obs xxxxxxxx
```

The LED display is the same as that for the P4 Overlay Frame Buffer Write/Write/Read Test.

P4 Enable Plane Write/Write/Read Test

This test will be executed only if a P4 Low Res Color Video RAM board is detected. The test verifies the address and data paths to the P4 Enable Plane RAM, as well as address bit and data bit uniqueness. For each test address of P4 Enable Plane
(base+0x0,base+0x01,base+0x02,base+0x04,base+0x08,...,base+ram_size), the test does the following:
For each data pattern at each test address:
(0x0,0x1,0x2,0x4,0x10,0x20,...0x80000000)

(a) Writes test data to the test address.
(b) Writes the inverted test data to the test address + 0x04.
(c) Reads the data back from the test address and compares.

Upon error, the test loops through steps a - c with constant test data and a constant test address. This test enters a scope loop on data compare errors, with the following message:

```
addr xxxxxxxx, exp xxxxxxxx, obs xxxxxxxx
```

The LED display is the same as that for the P4 Overlay Frame Buffer Write/Write/Read Test.

P4 Color Plane Write/Write/Read Test

This test will be executed only if a P4 Low Res Color Video RAM board is detected. The test verifies the address and data paths to the P4 Color Plane RAM, as well as address bit and data bit uniqueness. For each test address of P4 Color Plane
(base+0x0,base+0x01,base+0x02,base+0x04,base+0x08,...,base+ram_size)

the test does the following:

For each data pattern at each test address:
0x0,0x1,0x2,0x4,0x10,0x20,...0x80000000), the test

(a) Writes test data to the test address.
(b) Write inverted test data to the test address + 0x04.
(c) Reads the data back from the test address and compares.

Upon error, the test loops through steps a - c with constant test data and a constant test address. This test enters a scope loop on data compare errors, with the following message:

```
addr xxxxxxxx, exp xxxxxxxx, obs xxxxxxxx
```

The LED display is the same as that for the P4 Overlay Frame Buffer Write/Write/Read Test.

Printer Controller Check — Sun-3/80

The *Printer Controller Check* tests the ability of the processor to access the Parallel Printer Controller registers.

An incrementing pattern is displayed on the data output bits at the parallel printer connector. Each pattern is read back through the internal registers and verified.

If an error is detected, the test will enter a scope-loop, writing and reading the failed pattern. Only the first error will be reported to the serial port.

```
Printer Controller Data Error Expected xxxxxxxx, OBS xxxxxxxx
```

Clock/Calendar Device — 3/80 Only

This test verifies that the Clock/Calendar device is operational. The test is in several phases:

1. It verifies that the battery is not low:

```
WARNING: The Clock/Calendar Battery is low.
```

2. It verifies that the seconds counter is updating (the clock is running). If the seconds counter is not updating, the device is reinitialized and its date and time set to January 1, 1989 00:00:00. Two warning messages may be displayed that indicate an incorrectly initialized device or possibly a failed device.

```
<Clock is not running!, re-starting TOD Clock Module.>

<New TOD Module installed, initializing and starting the C
```

If an error is detected, the test will enter a scope-loop repeating the test sequence that failed. Only the first error will be reported to the serial terminal.

```
Time of Day Clock FAILED, Unit or support logic BAD
```

Configuration Memory Check — 3/80 Only

This test verifies the operation of the Configuration memory.

The Diagnostic Test locations in CMOS are tested and any errors detected are reported.

If an error is detected the test will enter a scope-loop, writing and reading the failed pattern. Only the first error will be reported to the serial port.

```
Data Path error Expected [xxxxxxxx] Found [xxxxxxxx]

Configuration RAM ERROR: Address uniqueness Fault
```

LANCE Ethernet Controller Check — 3/80 only

This test verifies the ability of the processor to access the controller's internal registers.

```
LANCE Controller BAD, unable to access.
```

ESP SCSI Check — 3/80 Only)

The ESP Controller Check includes the ability to access the ESP registers and the operation of the internal state-machine.

If an error is detected the test will enter a scope-loop repeating the test sequence that failed. Only the first error will be reported to the serial terminal.

```
ESP Controller Bad, FLAGS xxxxxxxxx FIFO xxxxxxxxx [xxxxxxx
```

Host System Initialization

After the Sun-3 firmware has executed the power-up sequence, it initializes the host system. Devices initialized at this time are the TIA, TIB, Page Tables, PMMU, keyboard, mouse, serial ports, frame buffer, memory, and other CPU devices. Initialization tasks include memory sizing, interrupt vector setting, trap vector setting, and setting entry points that correspond to support routines.

The Sun-3 PROM Monitor

This section describes the PROM monitor (sometimes called the "system monitor") commands available on Sun-3 workstation. For those with Sun-3/400 series or Sun-3/80 workstation, it is intended to replace Chapter 8 of the *PROM User's Manual.*

Taken as a whole, the monitor commands offer a low-level user interface to the Sun hardware. They control a variety of options, including booting from an alternate device, changing the console output, reading or altering registers or memory locations, and so on.

The effects of most monitor commands disappear when the power is turned off, with the exception of the **q** command, which programs the EEPROM. Parameters entered with the **q** command remain until deliberately altered with another

PROM command. Programming the EEPROM *reconfigures* your workstation. The Boot PROM consults the EEPROM to determine whether or not to poll for a boot device, whether to boot a specified program, which device is the console, and so on. Refer to the **q** command in this chapter, for information on programming the EEPROM. Also refer to the `eeprom` command described in the *SunOS Reference Manual*

Bringing up the PROM Monitor

Read *Chapter 3* of the *PROM User's Manual* for instructions on bringing up the PROM monitor.

Conventions

The paragraphs below describe each of the monitor commands in detail. Each paragraph starts with a line describing the command syntax. If a command has more than one distinct format, each one is shown on a separate line. Characters in **`bold courier`** font mean that you should enter them exactly as shown. Plain `courier` font represents what you should see on the screen or a program or path name. Words in *Roman italic* show the type of information you are to enter (variables), or list document names. *Roman italic* or **boldfaced** font is also used for notes or emphasis within the text. Optional arguments and default values are listed in the descriptions.

Monitor Command Overview

The following example shows the menu that comes up when you enter **h** (help) at the monitor prompt. This section describes the general characteristics that all of the commands have in common. Detailed descriptions of each command are listed in the next section. Some commands are available only for Sun-3/400 series workstations, and those will be identified.

NOTE The **i, j, n,** *and* **y** *commands are only available on systems with on-board cache memory, as indicated in italics.*

Figure 6 *Sun-3 Monitor Help Menu*

```
Boot PROM Monitor Commands
------------------------------------------------------------------
a [digit]                                        |Open CPU Addr Reg (0-7)
b [dev([cntrl],[unit],[part])]                   |Boot a file
c [addr]                                         |Continue program at Addr
d [digit]                                        |Open CPU Data Reg (0-7)
e [addr]                                         |Open Addr as 16 bit word
f beg_addr end_addr pattn [size]                 |Fill Memory
g [addr]                                         |Go to Addr
h or ?                                           |Help Menu
i [c] [i] [addr](Sun-3/470, 3/260)               |Open Central or I/O Cache Data
j [c] [i] [addr](Sun-3/470, 3/260)               |Open Central or I/O Cache Tags
k [number]                                       |Reset (0)CPU, (1)MMU, (2)System
l [addr]                                         |Open Addr as 32 bit long
m [addr]                                         |Open Segment Map
m [A] [B] [addr] (Sun-3/470,3/80)                   |Display TIA or TIB Table Entries
n [i/e/d]  (not for Sun-3/470)                   |Cache Invalidate/Enable/Disable
o [addr]                                         |Open Addr as 8 bit byte
p [addr]                                         |Open Page Map
p [addr]       (for Sun-3/470)                   |Display Page Table (For Page Or I/O Map
q [addr]                                         |Open EEPROM
r                                                |Open CPU/MMU Registers (i.e. PC)
s [digit]                                        |Set/Query Function Code (0-7)
t [y/n/c]                                        |Trace: Yes/No/Continue
u [arg]                                          |Select Console Device
v beg_addr end_addr [size]                       |Display Memory
w [addr] [string]                                |Vector
x                                                |Extended Diag Tests
y [c cxt] [s cxt sg_addr] [p cxt pg_adr]|Flush Cntxt/Seg/Page (not for Sun-3/470 or 3/80)
z [addr]                                                |Set Breakpoint
^a  (Sun-3/80                                    |Display physical/virtual address
^f  (Sun-3/80)                                   |Flush ATC Cache
^r  (Sun-3/80)                                   |Read on-board device registers
------------------------------------------------------------------
```

Executing a Command

In general, to execute a command, you type the appropriate command letter, followed by any required command arguments. For example, if you wanted to execute the hypothetical command "θ" (which we will say needs two arguments `100` and `200`) you would type a line like this:

```
> θ 100 200
```

The command letter can be upper or lower case, and all commands and arguments are separated by white-space (tabs or spaces). Pressing the return key executes the command.

Default Values

Many of the monitor commands have built in, or *default* values, which the command uses if you do not supply arguments. The default values vary from command to command. Check the command descriptions for the default values of interest.

Word Sizes

Word sizes referred to in this chapter are defined as follows: A **byte** is eight bits long; a **word** is 16 bits long; a **long word** is 32 bits long.

The Monitor Commands

This section provides more detailed information on the commands listed in the help menu example. Both the commands and their arguments are described here.

Displaying and Modifying Memory

A number of the commands listed here may be used to display and/or modify the system's memory and registers. Regardless of the type of memory (RAM, EEPROM, etc.) or register, these commands have the same command syntax. This section describes the memory modification syntax used by the monitor commands. The example here references long words (32 bits) of memory, but the syntax is the same for bytes (8 bits) and words (16 bits).

The *address* argument specifies the initial memory location that is to be displayed and/or modified.

The second argument determines whether the current content of a memory location is to be displayed and/or modified. Entering *only* the address of a memory location after the command *displays* the content of that address.

```
>command FFE80000 Return
FFE80000: 12345678 ?
```

At this point, you can respond in one of *three* different ways.

1. Simply pressing the Return key displays the contents of the next memory location (in this case, 0xFFE80004) as shown below.

```
>command FFE80000 Return

FFE80000: 12345678 ?  Return

FFE80004: 00000001 ?
```

   Successively pressing the Return key displays the contents of successive memory locations. Assuming that you pressed Return four times, the contents of memory locations 0xFFE80004, 0xFFE80008, 0xFFE8000C and 0xFFE80010 would be displayed.

2. To exit the command, enter **any** *non-hexadecimal* character (q for quit, for example) before pressing Return Note that you will now return to the monitor's basic command level, with the > prompt.

```
>command FFE80000 Return

FFE80000: 12345678 ? q Return

>
```

3. The *third* response to the display of memory location contents is to *modify* those contents. You enter the *new* hexadecimal value immediately following the question mark ?, BEFORE pressing Return. The following display demonstrates how the value of memory location 0xFFE80000 can be changed from "12345678" to "00ABCDEF".

NOTE *Following the assignment of the new value to memory location 0xFFE80000, the new value will not be displayed; instead, the contents of the next memory location are shown. You will not be returned to the monitor's basic command level.*

```
>command FFE80000 Return

FFE80000: 12345678 ? 00ABCDEF Return

FFE80004: 00000001 ?
```

Entering a memory location's virtual address followed *only* by a non-hexadecimal character before pressing the Return key causes the monitor to *display* the contents of that location then exit to the monitor command level.

```
>command FFE80004 q Return
FFE80004: 00000001
>
```

Entering a non-hexadecimal character *and* a hexadecimal value after an address causes the monitor to display the original value at that virtual address, assign a new value, then display that value. For instance, assume that the original content at address 0xFFE80010 is "00000005". The command

```
>command FFE80010 ? 55555550 Return
```

displays the original value "00000005" then assigns the value "55555550" to address FFE80010 before returning to the monitor prompt:

```
>command FFE80010 ? 55555550 Return
FFE80010: 00000005 -> 55555550
>
```

You may also enter multiple display and/or modify commands for multiple memory locations on the same command line. If you enter this command line:

```
>command FFE80000 ? 00000000 ? ? 22222220 33333330 q Return
```

You will see this on the screen:

```
FFE80000: 12345678 -> 00000000
FFE80004: 00000001
FFE80008: 00000002 -> 22222220
FFE8000C -> 33333330
FFE80010: 00000004
>
```

The first part of the command line,

>*command* FFE80000 ? 00000000

displays the original contents of location FFE80000 before assigning the new value "00000000" to it.

The next question mark directs the monitor to display the contents of FFE80004. The next part of the command line, **? 22222220**, tells the monitor to display the original contents of FFE80008, before assigning the new value "22222220" to it. The **33333330** tells the monitor to assign the value "33333330" to memory location FFE8000C. Finally, the **q** causes the monitor to exit to the command level after the contents of FFE80010 are displayed.

Special Monitor Commands

Address Increment/Decrement Command

By preceding the command with a **+** or **−**, you can cause the address display to increment or decrement to the next location.

While traversing a range of addresses, type a + or − to change direction after the program displays the content and waits for input.
For example:

```
>1 0 Return
00000000 00000000 ? Return
00000004 00000001 ? Return
00000008 00000002 ? - (you enter a minus sign) Return
00000004 00000001 ? Return (contents of previous location are displayed)
00000000 00000000 ? + Return (after decrement, you enter a plus sign)
00000004 00000001 ? Return you increment again
00000008 00000002 ? Return (you increment again)
```

The ^r Command — Sun-3/400 series only

A set of commands that are prefaced with the caret (^) character access the second level of the PROM monitor command menu.

Entering the ^ character and then the r key displays the following MC68030 (Sun-3/400 series) registers:

- The Translation Control Register in the PMMU
- The CPU Root Pointer (both Status Long Word and Table Address)
- The System Enable Register
- The Interrupt Register
- The Bus Error Register

Viewing these registers gives a quick view of the current CPU status as opposed to the monitor r command, which displays status that was stored during an exception error.

The ^t Command

Entering the ^ character and then the **t** key, followed by a virtual address, displays the physical address to which that address is mapped, along with a detailed description of all the bits in the page table entry, the segment and page RAM addresses, and what space they are in.

For example, entering

```
>^t 1000 Return
```

results in this display:

```
Virtual Addr 1000 is mapped to Physical Addr 1000
Context = 0x0, Seg Map = 0x0, Page Map = 0xC0000000.
Page 0 has these attributes:

      Valid bit  = 0
      Permission = 1
      No Cache   = 0
      Type       = 0
      Access bit = 0
      Modify bit = 0
```

Entering **^t** with no arguments provides information with reference to virtual address 0x00.

The **^i** Command

Entering the **^** character and then the **i** key, followed with a command, displays compilation information for the system firmware. It includes the date, host name and build directory path. For example:

```
Compiled at 6/7/87 on hostname in /directory_name
```

The **^c** Command

^c *source destination n*

Entering the **^** character and then the **c** key, followed with the parameters shown, causes a block of *n* length to be copied from *source* to *destination* address, byte by byte. There is enough delay to copy to EEPROM also.

The **!** Command

Entering an exclamation point executes the last monitor command you entered again.

Regular Monitor Commands

Monitor **a** Command

a *register_number action*

The a command provides access to the CPU's address registers. *register_number* may be a value from 0 to 7 inclusive. The default value is 0. Register_number 7 accesses A7, the system stack pointer. To see the user stack pointer, use the **r** command.

It is important to note that it is *not* possible to display the state of the processor at all times. In particular, processor state is only observable after:

- An unexpected trap
- A user program has "dropped" into the monitor (by calling monitor function `abortent`).
- You have manually "broken" into the monitor mode (by typing the `L1-a` sequence on a Sun keyboard or Break on a dumb terminal's keyboard).

Read the section entitled *Displaying and Modifying Memory* for details on how to use this command. Replace the word *command* in the description with the letter **a**.

Monitor **A** Command—Sun-3/400 Series

For the Sun-3/400 series, the uppercase **A** command displays these physical and virtual addresses:

Device	*Virtual Address:*	*Physical Address:*
Keybrd/Mouse	0x:fef00000	0x:62000000
Serial Port	0x:fef02000	0x:62002000
EEPROM	0x:fef04000	0x:64000000
TOD	0x:fef06000	0x:64002000
Mem Err Reg	0x:fef09000	0x:61001000
Int Reg	0x:fef0b400	0x:61001400
P4 DAC	0x:fef0c000	0x:50200000
ie Ethernet	0x:fef08000	0x:65000000
ECC ENA Reg	0x:fef14000	0x:6a1e0000
Sys ENA Reg	0x:fef16000	0x:61000000
Bus ERR Reg	0x:fef18400	0x:61000400
IDPROM	0x:fef1cc00	0x:61000c00
P4 Video Reg	0x:fef1e000	0x:50300000
VIDEOMEM_BASE	0x:fef20000	0x:50400000
P4 Overlay	0x:fef20000	0x:50400000
P4 ENA Plane	0x:fef40000	0x:50600000
IOMAPPER	0x:fef66000	0x:60000000
CACHETAGS	0x:fefc0000	0x:68000000
CACHEDATA	0x:fefd0000	0x:69000000
IO CACHE TAGS	0x:fef6c000	0x:6c000000
IO CACHE DATA	0x:fef6e000	0x:6c002000

Monitor **b** Command

b ? or **b**! *boot_device path argument_list*

The `boot` command loads and executes the SunOS operating system, an EEPROM-specified program, or a user-specified program. The boot program can be loaded from the default device, the device specified in the EEPROM, or the boot device specified in the command argument. A *boot_device* is a secondary storage device (disk, Ethernet or tape) that contains the program to be loaded and executed.

If the diagnostic switch on the back of the system is in the NORM position, the value in EEPROM address 0x18 is *not* equal to 0x12, and the boot command is entered without arguments, the system will boot the SunOS operating system, using the following default boot device polling sequence.

1. Xylogics Disk (xy), (xd), (xt).
2. SCSI Disk (sd), (st).
3. Ethernet (ie), (le).

If the EEPROM value at address 0x18 *is* equal to 0x12, the system will boot the SunOS operating system from an EEPROM-specified device. The boot device is specified in locations 0x19 through 0x1D, inclusive, of the EEPROM. Refer to the `q` command for information on how to open and modify these EEPROM

locations.

When the diagnostic switch is in the DIAG position and command **b** is entered by itself, the system will boot an EEPROM-specified program from an EEPROM-specified device. In this case, the boot path is specified in locations 0x28 through 0x50, inclusive, of the EEPROM; the boot device is specified in locations 0x22 through 0x26, inclusive, of the EEPROM. If the boot attempt fails, the user is returned to the monitor's command level.

In order to boot from a specific device, the **b** command must be followed with a boot device abbreviation, such as those shown below. Enter **b ?** to view the boot device identifier arguments that your PROM monitor will accept.

b *device* (*controller*, *unit*, *partition*) *path argument_list*

You must surround `controller, unit, and partition` with parentheses, and place a comma after each entry. To invoke the default values, simply enter:

b *device* **(,,)**

device may be one of the following:

fd — Floppy Disk
xd — Xylogics 7053 Disk
xy — Xylogics 450/451 disk
sd — SCSI disk
ie — Intel Ethernet
st — SCSI tape
xt — Xylogics 472 Tape
mt — Tape Master 9-Track Tape

controller stands for the Controller Number, referring to the tape or disk controller board. The default is 0.

unit refers to Unit Number, meaning disk number. The default is 0.

partition is the partition number of the boot device. The default is 0.

path is the path and filename of the program to boot.

argument_list is the list of arguments for the boot program. Up to seven optional arguments (which are passed to the boot program) may follow the path argument.

If you do not want the system to be reset prior to booting, you must insert ! before the device identifier argument.

```
b ie(0,0,1)/stand/video.diag -t
```

The boot command shown above would boot the video diagnostic from the `/stand` directory over the Ethernet. Because the ! argument does *not* precede the device identifier argument, the system is reset before the booting process begins. Note that one optional argument (`-t`) is passed on to program `video.diag`.

Monitor **c** Command

c *virtual_address*

The `continue` command resumes execution of an interrupted program. You can specify the virtual address of the instruction to be executed when the program restarts.

By default, the program resumes execution at the address pointed to by the program counter.

NOTE *This command is helpful if you should use the* `L1-A` *sequence and decide you did not want to abort the operating system.*

Monitor **d** Command

d *register_number action*

The `data register` command provides access to the CPU's data registers. *register_number* can be a value from 0 to 7 inclusive. The default value is 0. If you do not specify a register, this command displays the data registers one-at-a-time, in ascending order.

It is important to note that it is *not* possible to display the state of the processor at all times. In particular, processor state is only observable after:

- An unexpected trap
- A user program has "dropped" into the monitor (by calling monitor function `abortent`)
- You have manually "broken" into the monitor (by typing `L1-a` on the console's keyboard or **break** on the "dumb" terminal's keyboard).

Read the previous section, *Displaying and Modifying Memory* for details on how to use this command. Replace the word *command* in the description with the letter **d**.

Monitor **e** Command

e *virtual_address*

The `display/modify memory` command displays and/or modifies the content of one or more virtual addresses in *word* mode (i.e. each virtual address will be treated as a 16-bit unit). That is, the content of one or more words can be *displayed, modified* or, both *displayed* and *modified.*

See the previous section *Displaying and Modifying Memory* for a description of this command's syntax. Use the letter **e** in place of the word *command* in the description.

Monitor **f** Command

f *start_virtual_address end_virtual_address pattern size*

The `block write` command writes the *pattern* you enter into each byte, word or long word in the range of virtual addresses you specify with the *start_virtual_address* and *end_virtual_address* arguments . By default, if the final, memory-cell-size argument is not present, bytes are written. Arguments *start_virtual_address end_virtual_address* and *pattern* are required while *size* is optional. The possible values for *size* are `b` (8-bit byte), `w` (16-bit word) or `l` (32-bit long word).

Monitor **F** Command—Sun-3/400 Series

For the Sun-3/400 series, the **F** command flushes the ATC Cache on the MC68030 CPU.

Monitor **g** Command

g *vector argument*
or
g *virtual_address argument*

When you use the `goto` command, you may go to (jump to) a *pre-determined, user-specified* or *default* routine. If a pre-determined or default routine is invoked, optional arguments `vector` (a hexadecimal number) and `argument` (a string) are passed to the routine to be executed.

In its other form, the argument `virtual_address` is used to indicate the virtual address of a *user-specified* routine, and the optional `argument` is passed along to that routine. If you do not supply the `vector/virtual_address` argument, the value in the Program Counter is used.

In order to set up a *pre-determined* routine, a user program has to set variable `*romp->v_vector_cmd` equal to the virtual address of the routine. Variable `*romp->v_vector_cmd` must be set prior to executing the `g` command. Pre-determined routines may or may not return to the monitor.

The *default* routine, defined by the monitor, simply prints the user-specified `vector` argument according to the user-specified format (given in `argument`) before returning to the monitor. The only allowable formats are `%x` and `%d`. Format `%x` prints argument `vector` as a hexadecimal number while format `%d` prints argument `vector` as a decimal number.

Monitor **h** Command

h *or* **?**

The `help` command brings up a Help display that describes the basic monitor commands and their argument(s). An example of the help menu is shown at the beginning of this chapter.

If the help menu on your system has numbered entries, you may enter the number next to the command, and then Return to obtain further information about that command.

Monitor **i** Command — Sun-3/200 or Sun-3/400 series

i *c icache_data_offset*

The `modify cache data` RAM command displays and/or modifies the contents of one or more of the cache data addresses. For Sun-3/400 series systems, you may add a `c` after entering `i`, to specify Central Cache. Adding a second `i` specifies I/O Cache.

> Read the previous section *Displaying and Modifying Memory* for details on how to use this command. Replace the word *command* in the description with the letter **i**.

Monitor **j** Command — Sun-3/200 or Sun-3/400 Series

j *cache_tag_offset*

The `modify cache tag RAM` command displays and/or modifies the contents of one or more of the cache tag addresses. For Sun-3/400 Series systems, you may add a `c` to specify Central Cache, or an `i` to specify I/O Cache.

Read *Displaying and Modifying Memory* for details on how to use this command. Replace the word *command* in the description with the letter **j**.

Monitor **k** Command

k *reset_level*

The `reset` command performs various levels of system resets. It can also be used to display the system's banner. This command accepts one optional argument. Entering **k 0** resets the VME-bus, interrupt register and video monitor (the Low-Level Reset).

Entering **k 1** invokes a Software Reset.

Entering **k 2** invokes a Power-On Reset (Hard Reset).

Finally, entering **k b** displays the banner on the video monitor. The default value of the argument is `0`.

Monitor **l** Command

l *virtual_address*

The `modify long words of memory` command displays and/or modifies the contents of one or more virtual addresses in *long* word mode. Each virtual address is treated as a 32-bit unit (long word). Read "Displaying and Modifying Memory" for details on how to use this command. Replace the word *command* in the description with the letter **l**.

Monitor **m** Command

m *a b virtual_address*

The `modify segment table` command displays and/or modifies the contents of one or more of the Segment Table entries. For Sun-3/400 series systems, you may specify `a` for translation table TIA, or `b` for translation table TIB. If you do not specify the table, TIA is displayed by default.

Read *Displaying and Modifying Memory* for details on how to use this command. Replace the word *command* in the description with the letter **m**.

Monitor **n** Command — Sun-3/200 and 3/400 series only

n *c i cache_command*

The `control cache` command globally controls the cache. Entering **n d** *disables* the cache (Central Cache for Sun-3/400 series systems). Entering **n e** *enables* the cache (Central Cache for Sun-3/400 series systems). Finally, entering **n i** *invalidates* the cache (Central Cache for Sun-3/400 series systems).

For Sun-3/400 series systems, enter

ni *d, e, or i*

to perform the action on the I/O Cache. To select the Central Cache, enter

nc *d, e, or i*

Monitor **o** Command

o *virtual_address*

The `modify bytes of memory` command displays and/or modifies the content of one or more virtual addresses in *byte* mode. Each virtual address is treated as an 8-bit unit (byte). Read *Displaying and Modifying Memory* for details on how to use this command. Replace the word *command* in the description with the letter **o**.

Monitor **p** Command

p *virtual_address*

The `modify page table` command displays and/or modifies the contents of one or more of the Page Table entries. For Sun-3/400 series systems, the virtual address determines whether the page displayed is from the page table or from the I/O Mapper RAM. A virtual address between 0xff000000 and 0xffffffff searches for the requested page in the I/O Mapper RAM pages.

Read *Displaying and Modifying Memory* for details on how to use this command. Replace the word *command* in the description with the letter **p**.

Monitor **q** Command

q *eeprom_offset* or **q ***

The `modify bytes of EEPROM` command displays and/or modifies the configuration information within the EEPROM in *byte* mode. This command works similarly to the memory commands discussed previously, except that the modified addresses are offset, and the changes you make remain when you power-down the workstation. Read the previous section, *Displaying and Modifying Memory* for details on how to use this command. Replace the word *command* in the description with the letter **q**. Refer to the *Sun-3 EEPROM Layout* chapter of the for information *PROM User's Manual* on what parameters are stored at which EEPROM addresses.

q*—Version 2.4 and later Boot PROM Only

In the Version 2.4 Boot PROM, you may enter an asterisk instead of an EEPROM offset value as an argument, and the **q** command will erase the entire EEPROM.

Monitor **r** Command

r register_symbol

The `miscellaneous register` command displays and/or modifies the contents of the CPU's miscellaneous registers. You may enter any of the symbols shown on the table below to specify registers. For example, if you enter the following, you will display the state of the MC68030 Cache Control register:

```
>r cc
```

Table 2 *Miscellaneous Registers for the 68020*

Symbol	*Name*
IS	Interrupt Stack Pointer
MS	Master Stack Pointer
US	User Stack Pointer
SF	Source Function Code
DF	Destination Function Code
VB	Vector Base
CA	Cache Address Register
CC	Cache Control Register
CX	Context Register
SR	Status Register
PC	Program Counter

Table 3 *Miscellaneous Registers for the 68030*

Symbol	*Name*
SS	System Stack
US	User Stack
VB	Vector Base Register
PC	Program Counter
SR	CPU Status Register
CC	Cache Control Register
CA	Cache Address Register

It is important to note that it is *not* always possible to display the registers. They may be observed only after

- An unexpected trap
- A user program has "dropped" into the monitor (by calling monitor function `abortent`) or
- You have manually "broken" into the monitor (by typing `L1-a` on the Sun keyboard or Break on a "dumb" terminal's keyboard).

Read the previous section *Displaying and Modifying Memory* for details on how to use this command. Replace the word *command* in the description with the letter **r**.

Monitor **R** Command—(Sun-3/400 Series only)

The uppercase **R** command reads the System Enable, Interrupt, Bus Error, I/O Mapper and Memory Error registers in the MC68030. Addresses of those registers are as follows:

Register	*Address*
System Enable	0xa100
Interrupt	0x81
Bus Error	0xe3
I/O Mapper	
Memory Error	0x007feab8

Monitor **s** Command

s *number*

The `modify` command sets or displays the address space to be used by subsequent memory access commands. The processor function codes decode the address spaces. Argument choices represent the function codes:

Table 4 *Function Code Values*

Value	Address Space
0	Reserved (don't use)
1	User Data
2	User Program
3	Control Space (Reserved in Sun-3/400 Series)
4	Reserved (don't use)
5	Supervisor Data
6	Supervisor Program
7	Supervisor CPU Space

If you do not enter a function code number, the current setting (either 1 or 5) is displayed, and entry of the monitor **o** command, for example, would cause the program to look for the specified address in either user or supervisor data space. Conversely, if you reset the function code number, you could query registers in the space represented by the number entered. For example, entering

```
>s 3
```

would allow you to read the bus error or system enable register, which are located in Control Space.

Monitor **T** Command—Sun-3/400 Series

Entering and uppercase **T** for the Sun-3/400 series translates the virtual address entered after the command to its corresponding physical address, similarly to the `^t` command for other Sun-3 systems. If you entered

```
>t 12345678
```

for example, the display might look like this:

```
Virtual Addr 0x12345678 is mapped to Physical Addr 0xfef12345
TIA Entry = 0x7fa400, TIB Entry = 0x7fa60a PTE = 0x59

        Page 0x1a has these attributes:
        No Cache          = 1
        Modify bit        = 1
        Access bit        = 1
        Write Protect bit = 0
        Descriptor Type   = 1
```

Monitor **u** Command

u *port options baud_rate*
or

u *echo*
or

u u*virtual_address*

The *input/output* command configures the input and output devices and their characteristics or displays the current input and output device set-up.

The **u** command requires arguments to specify from which device(s) you want the system to expect input or which device(s) will display output.

If you do not enter an argument after the u command, the program will display the current settings. If no serial port is specified when changing baud rates, the baud rate of the current input device is changed. The default serial port baud rate is 9600 for both ports during a normal boot. During a diagnostic boot, defaults are 9600 baud for Port A and 1200 for Serial Port B.

Upon normal power-up (diag switch is in NORM position), the default console input device is the Sun keyboard, unless the EEPROM has specified another default input device. If the keyboard is unavailable, the system looks to serial port A for for input.

The default console output device is the Sun monitor (subject to change through EEPROM programming). If the workstation has a color monitor and a color board is unavailable, the program will look for a monochrome monitor as an output device.

You may alter the existing I/O settings while you are in the monitor mode, using the commands listed below; however, the default settings will be reinstated when the system is power cycled.

u Command Arguments:

The *port* argument can be one of the following:

Table 5 *Port Arguments*

Port Argument	Device
a	Serial Port A
b	Serial Port B
k	Keyboard
s	Screen

The `options` arguments are:

Table 6 *Option Arguments*

Option Argument	Meaning
i	input
o	output
u	UART
e	input echoed to output
ne	input *not* echoed to output
r	reset specified serial port

The `baud_rate` argument specifies baud rate of the serial port being discussed. The `virtual_address` argument specifies the virtual address of the UART (Universal Asynchronous Receiver/Transmitter).

Following are examples of port and options arguments:

- Enter **u aio** or **u bio** to select serial port A or B as the input and output device.
- Enter **u ai** or **u bi** to select serial port A or B for input only.
- Enter **u ao** or **u bo** to select serial port A or B for output only.
- Enter **u k** to select the keyboard for input.
- Enter **u ki** to select the keyboard for input.
- Enter **u s** to select the screen for output.
- Enter **u so** to select the screen for output.
- Enter **u ks, sk** to select the keyboard for input and the screen for output.
- Enter **u a***baud rate* or **u b***baud rate* to set the serial port speed.
- Enter **u e** to cause the output to echo the input.
- Enter **u ne** to cause the output **not** to echo the input.
- Enter **u** *address* to set the serial port virtual address.

A *previously mapped* UART can also be used for input and/or output. Command **u u***virtual_address*'', where *virtual_address* is the virtual address of a previously mapped UART, changes the virtual address of the UART. Do not enter a space between the second **u** and the virtual address.

Monitor **v** Command

v *start_virtual_address end_virtual_address size*

The `display memory block` The `display memory block` command displays the contents of each byte, word or long word in the range of virtual addresses specified by *start_virtual_address* and *end_virtual_address*. The *word_size* argument is optional; the default is to display memory in byte format. In this format, sixteen consecutive bytes are displayed on each line. In word format, eight consecutive words appear on each line.

If long word format is enabled, four consecutive long words appear on each line. To the far right of each line, the character corresponding to each byte is also shown. All bytes that contain a non-ASCII code are shown as a period (.). Legal values for *size* are **b** (8-bit byte), **w** (16-bit word), or **l** (32-bit long word).

To terminate the **v** command, press the *space bar*. To "freeze" the display, press any key *except* the space bar. To restart the **v** command, press the space bar again.

Monitor **w** Command

w *virtual_address argument*

The `set execution vector` command allows you to vector to a *pre-determined* or *default* routine. Optional arguments *virtual_address* and *argument* are passed along to the to-be-executed routine.

In order to set up a *pre-determined* routine, a user program sets the variable `*romp->v_vector_cmd` equal to the virtual address of the *pre-determined* routine. Variable `*romp->v_vector_cmd` must be set prior to executing the **w** command. *Pre-determined* routines may or may not return to the monitor.

The *default* routine, defined by the monitor, simply prints the user-specified *virtual_address* argument according to the user-specified format (given in *argument*) before returning to the monitor. The only allowable formats are "%x" and "%d". Format "%x" prints argument *virtual_address* as a hexadecimal number; format "%d" prints it as a decimal number.

Monitor **x** Command

x

The `extended test system` command invokes the Extended Test Sequence or the Extended Test System, depending on what Boot PROM revision is on the CPU board. See *Chapter 9* for details.

Monitor **y** Command — Sun-3/200 or -3/400 Series

y *c number*
or

y *cache_section number virtual_address*

The `flush cache` command performs a number of *flush* operations on the cache. Depending on what is to be flushed, two or three arguments are required. In order to flush a context, enter command **y c** *number*, where *number* is a valid context number.

Entering **y s** *number virtual_address* flushes segment *virtual_address* within context *number*. The argument *number* is a valid context number.

Entering **y p** *number virtual_address* flushes page *virtual_address* within context *number*. The argument *number* is a valid context number.

Monitor **z** Command

***z** number breakpoint_virtual_address type len*

Command **z** sets or resets breakpoints for debugging purposes. Optional argument *number* can have values from 0 to 3, corresponding to the processor debug registers DR0 to DR3, respectively. Up to four distinct breakpoints can be specified. If argument *number* is not specified, the monitor will choose a breakpoint number.

The argument *breakpoint_virtual_address* specifies the breakpoint address to set.

The argument *type* can have three values: **x** for Instruction Execution breakpoint, **m** for Data Write only breakpoint, and **r** for Data Reads-and-Writes-only breakpoint. The default value for *type* is **x**.

The argument *len* can also have three values: **b**, **w**, and **l**, corresponding to the breakpoint field length of byte, word, and long-word, respectively. The default value for *len* is `b`. Since the breakpoints are set in the on-chip registers, an instruction breakpoint can be placed in ROM code or in code shared by several tasks.

If the argument *number* is specified but the argument *breakpoint_virtual_address* is not specified, then the corresponding breakpoint will be reset.

This command without any arguments will display all the existing breakpoints.

1. SPARCsystem 330 Self-Tests and Monitor Commands

For the most part, the diagnostic self-test and monitor command descriptions for the SPARCsystem 330 are the same as those in Chapters 11, 12 and 13 of the *PROM User's Manual.* This addendum provides descriptions of commands or tests that differ from those used for Sun-4/110 or Sun-4/200 series systems.

When a terminal is attached to Serial Port A, and the system in is diagnostic mode, the name of each self-test is displayed as the test is executed, as shown on the following pages.

When in normal mode, self-tests for 8 Mbtyes of memory may last 45 seconds; a 128 MB memory may take up to 8 minutes. In diagnostic mode, when all of memory is tested, an 8 MB system may take four minutes to complete self-tests, while a 128 MB system may take 56 minutes to execute the self-tests.

Self-Test Interaction

There are several commands that allow limited interaction with the SPARCsystem self-tests, and they are described below.

b Press the **b** (a mnemonic for *burn-in*) key, prior to the display of the `...Completed` or `Selftest Finished` message, to execute the power-up test sequence indefinitely. This option is useful during the manufacturing burn-in stage.

For the Sun-3/80, when the last selftest is finished the message tests`Automaticallycontinuing` will be displayed and the self-test will be restarted again, using the System Enable Register read test as the first test. The SCC and terminal I/O tests are bypassed in burn-in mode since operator interaction is required. This looping sequence will continue until an ESCAPE is entered, a reset is done, or the "*b* " key pressed again to turn burn-in mode off. The burn-in mode can be toggled by successive pressing of the "*b* " key. Note that this burn-in mode key can be processed during a test that is running normally (no errors), if the test is presently looping on an error encountered, or at the end of selftest when the "Selftest Finished" is displayed and an operator input is requested.

s Press the **s** key prior to the display of the `...Completed` message to *re-start* the power-up test sequence.

Space Bar

If one of the power-up tests fails, it will continue to re-execute forever unless interrupted. Press the space bar to terminate the failed test and execute the next power-up test.

p — Toggle Print Mode

By default, an unsuccessful power-up test prints one error message before entering an infinite scope loop. Once inside of the scope loop, the failing test will *not* print any more messages. If, however, you would like to see test messages while in the scope loop, press the **p** key. Then, to turn the messages back off, press the **p** key a second time. In other words, the `p` command acts like a toggle switch to turn message mode on or off.

Escape Key

Pressing Esc prior to the `Selftest`Completed message skips the remaining power-up tests and warns:

```
<Warning: Selftests aborted by user>
```

Control-L
Holding down the Control key while pressing **L** causes the test that is currently running to re-execute forever. Pressing the Control-L sequence again turns toggle-loop mode off.

Control-Q
Holding down the Control key while pressing **Q** terminates the current test and skips to the next test in the power-up self-test sequence. This feature works on a passing or failing test.

NOTE *If EEPROM location 0x17 is set to 0x12, you may press the User Reset Switch at the back of the system to restart the self-tests. The tests will restart regardless of the position of the Diagnostic Switch.*

Figure 7 *SPARCsystem 330 Diagnostic Boot-Up Display*

```
Boot PROM Selftest
EPROM Checksum Test.
Context Register Test.
Segment Map Write-Write-Read-Read Test.
Segment Map Address Test.
Segment Map 3-Pattern Test.
Page Map Write-Write-Read-Read Test.
Page Map Address Test.
Page Map 3-Pattern Test.
Software Traps Test: levels 0x80 -> 0xFF
Interrupt Register Test
Software Interrupts Test.
TOD Interrupt Test.
<Probing for a P4 Video Board>
(For Color systems, refer Video tests change as shown on tables that follow
Video Memory Write-Write-Read Test. 32-bit (black and white)
Video Memory Address Test. 32-bit (black and white)
Video Memory 3-Pattern Test. 32-bit (black and white)
<Probing for Main Memory>
<Found 0x00000008 MB of Main Memory>
Limited Memory Test: End of Megabyte 0x00000008. Tests Complete.
MMU Read Access/Modified Bits Test.
MMU Write Access/Modify Bit Test.
MMU Write to Write-Protected Page Test.
MMU Read From Write-Protected Invalid Page Test
MMU Read Writable Invalid Page Test.
MMU Write To Write-Protected Invalid Page Test.
MMU Write To Writable Invalid Page Test.
Main Memory Non Existent Physical Address Read Timeout Test.
Main Memory Invalid Physical Address Read Timeout Test.
Main Memory Invalid Physical Address Write Timeout Test.
Control Space Timeout Test.
Range Error Test
Size Error Test.
```

The Diagnostic Display is continued on the next page.

Figure 8 *Diagnostic Boot-Up Display - Continued*

```
Parity No Fault Test.
Parity Error Detection and Trap Test.
Cache Tag RAM Write-Write-Read-Read Test.
Cache Tag RAM Address Test.
Cache Tag RAM 3-Pattern Test.
Cache Data RAM Write-Write-Read-Read Test.
Cache Data RAM Address Test.
Cache Data RAM 3-Pattern Test.
<Exiting boot state>
<Successfully exited boot state>
Cache Read Hit (Addr Match,Valid) Test
Cache Read Hit (Addr Match,Valid,Cntx Diff,Spvsr) Test
Cache Read Miss (Addr Match,Valid,Cntx Diff,User) Test
Cache Read Miss (Addr Match,Invalid,Cacheable) Test
Cache Read Miss (No Addr Match,Valid,Cacheable) Test
Cache Read Miss (Addr Match,Invalid,Non-Cacheable) Test
Cache Write Hit (Addr Match,Valid,M=1) Test
Cache Write Hit (Addr Match,Valid,M=0) Test
Cache Write Hit (Addr Match,Valid,Cntx Diff,Spvsr) Test
Cache Write Miss (Addr Match,Valid,Cntx Diff,User) Test
Cache Write Miss (Addr Match,Invalid,Cacheable) Test
Cache Write Miss (No Addr Match,Valid,Cacheable) Test
Cache Write Miss (Addr Match,Invalid,Non-Cacheable) Test
Cache Flush Context (Cntx Match) Test
Cache Flush Context (Cntx No Match) Test
Cache Read Double Hit (Addr Match,Valid) Test
Cache Read Double Hit (Addr Match,Valid,Cntx Diff,Spvsr) Test
Cache Read Double Miss (Addr Match,Valid,Cntx Diff,User) Test
Cache Read Double Miss (Addr Match,Invalid,Cacheable) Test
Cache Read Double Miss (No Addr Match,Valid,Cacheable) Test
Cache Read Double Miss (Addr Match,Invalid,Non-Cacheable) Test
Cache Write Double Hit (Addr Match,Valid,M=1) Test
Cache Write Double Hit (Addr Match,Valid,M=0) Test
Cache Write Double Hit (Addr Match,Valid,Cntx Diff,Spvsr) Test
Cache Write Double Miss (Addr Match,Valid,Cntx Diff,User) Test
Cache Write Double Miss (Addr Match,Invalid,Cacheable) Test
Cache Write Double Miss (No Addr Match,Valid,Cacheable) Test
Cache Write Double Miss (Addr Match,Invalid,Non-Cacheable) Test
Cache Successive Write Double Hit (Addr Match,Valid,M=1) Test
Cache Successive Write Double Hit (Addr Match,Valid,M=0) Test
```

This display is continued on the following page.

Figure 9 *Diagnostic Boot-Up Display - Continued*

```
  Atomic Load/Store Hit (Addr Match,Valid) Test
  Atomic Load/Store Miss (Addr Match,Invalid) Test
  Atomic Load/Store Hit (Addr Match,Valid,Write Protected Page) Test
  Atomic Load/Store Miss (Addr Match,Invalid,Write Protected Page) Test
  <Re-entering boot state>
  <Successfully re-entered boot state>
  VME Loopback and DVMA Test
  VME Loopback and DVMA Read Timeout Test
  VME Loopback and DVMA Write Timeout Test

  END OF SELFTEST # 0x00000000 (SELFTEST PASSED) .
  #PASSED = 0x00000000,  #FAILED = 0x00000001

  <Sizing Main Memory>
  <Found 0x00000008 MB of Main Memory>
  <Initializing Memory...>
  <Turning Parity Checking ON>
  Main Memory Test: Megabyte 0x00000008. Tests Complete.
  <Found 0x00000008 MB of Main Memory>
  <Initializing Memory...>

  Selftest Completed.

Type a key within 10 seconds to enter Extended Test System (e for echo mode).
```

If the system contains a CG6 board, the self-tests function the same as the Video tests shown previously, and the display looks like this:

```
<Probing for a P4 Video Board>
Video Memory Write-Write-Read-Read Test. 32-bit - Color Plane.
Video Memory Address Test. 32-bit - Color Plane.
Video Memory 3-Pattern Test. 32-bit - Color Plane.
```

If the system contains a CG8 board, the self-tests function the same as the Video tests shown previously, and the display looks like this:

```
<Probing for a P4 Video Board>
Video Memory Write-Write-Read-Read Test. 32-bit - Overlay Plane.
Video Memory Address Test. 32-bit - Overlay Plane.
Video Memory 3-Pattern Test. 32-bit - Overlay Plane.
Video Memory Write-Write-Read-Read Test. 32-bit - Enable Plane.
Video Memory Address Test. 32-bit - Enable Plane.
Video Memory 3-Pattern Test. 32-bit - Enable Plane.
Video Memory Write-Write-Read-Read Test. 24-bit - Color Plane.
Video Memory Address Test. 24-bit - Color Plane.
Video Memory 3-Pattern Test. 24-bit - Color Plane.
P4 Color Map Test.
```

If the Diagnostic Switch is in the *diagnostic* position, the power-up tests executed successfully and the user did *not* enter a character on either of the *terminal* keyboards within the initial ten second period, the following display will appear on the *workstation's* screen. Again, note that you have a *second* ten-second

opportunity to invoke the additional non-power-up tests by pressing *any* key on the workstation's keyboard, except the Esc key. Pressing the Esc key in this situation will terminate the delay and invoke the PROM monitor program. Providing two opportunities to enter the Extended Test System will give the user the choice of interacting with either an RS232 terminal or the workstation's keyboard and video monitor.

If you do not respond to either of the *ten-second* delays, an attempt will be made to boot an EEPROM-specified program from the Diagnostic boot path area of the EEPROM. If this attempt should fail, the monitor program is invoked.

Successful Self-Test Display

The test names actually appear on the screen sequentially as each test is performed. The previous example shows the display on the the terminal (connected to Serial Port A at 9600 baud or Port B at 1200 baud) screen after all the self-tests are successfully completed.

NOTE *If no terminal is connected to the serial port, you will not be able to interact with the self-tests, and you won't see the Diagnostic Boot-Up Display. If self-test is successful, the console display prompts will offer choices described on the following page.*

```
Selftest Completed Successfully

        Sun Workstation, Model Sun-4/300 Series,
        Sun-4 Keyboard
        ROM Rev __, __ MB memory installed, Serial #______
        Ethernet address __:__:__:__:__:__

Type a character within 10 seconds to enter Extended Test System.
```

If you do not press a key on the console keyboard, the Boot PROM program checks parameters stored in the EEPROM that tell it to do one of the following:

- Boot a specific (diagnostic) program from a specific device
- Drop into the PROM monitor mode.

If you press a key, the Extended Test System menu is offered. Refer to the SPARCsystem 330 extended test system addendum at the end of this chapter.

To Read the CPU Board LED Table

The following table provides a brief interpretation of the patterns displayed by the LED indicators on the edge of the CPU board. For vertical installations, the LED depicted on the left in this table is located at the top of the display; the LED on the right is at the bottom of the display.

Table 7 *SPARCsystem CPU Board LED Interpretation*

LED Display ●= ON, o= OFF 7	6	5	4	3	2	1	0	*Self-Test being Performed.*
● ⇒	o ⇒	o ⇒	o	o ⇐	o ⇐	o ⇐	●	LED Loop Test
o	o	o	o	o	o	o	o	SCC WR 12 Write-Read Test
o	o	o	o	o	o	o	●	Initialize SCC UART
o	o	o	o	o	o	●	o	"BOOT PROM Selftest" Message
o	o	o	o	o	o	●	●	EPROM Checksum Test
o	o	o	o	o	●	o	o	Context Register Read-Write Test
o	o	o	o	o	●	o	●	Segment Map Tests
o	o	o	o	o	●	●	o	Page Map Tests
o	o	o	o	o	●	●	●	Software Traps Test
o	o	o	o	●	o	o	o	Interrupt Tests (Software and Register)
o	o	o	o	●	o	o	●	TOD Interrupt Test
o	o	o	o	●	o	●	o	Video Memory Tests
o	o	o	o	●	o	●	●	Limited Main Memory Tests
o	o	o	o	●	●	o	o	MMU Read Access/Modified Bits Test
o	o	o	o	●	●	o	●	MMU Write Access/Modified Bits Test
o	o	o	o	●	●	●	o	MMU Write to Write-Protected Page Test
o	o	o	o	●	●	●	●	MMU Read Not-Writeable Invalid Page Test.sp 2p
o	o	o	●	o	o	o	o	MMU Read Writeable Invalid Page Test
o	o	o	●	o	o	o	●	MMU Write Not-Writeable Invalid Page Test
o	o	o	●	o	o	●	o	MMU Write Writeable Invalid Page Test
o	o	o	●	o	o	●	●	Main Memory Timeout Test
o	o	o	●	o	●	o	o	Control Space Timeout Test
o	o	o	●	o	●	o	●	Range Error Test
o	o	o	●	o	●	●	o	Size Error Test
o	o	o	●	o	●	●	●	Parity Memory Test
o	o	o	●	●	o	o	o	CPU Cache Tag RAM Tests
o	o	o	●	●	o	o	●	CPU Cache Data RAM Tests
o	o	o	●	●	o	●	o	CPU Cache Functional Tests
o	o	o	●	●	o	●	●	VME Loopback Tests
o	o	o	●	●	●	o	●	Main Memory Tests (Parity on)

The SPARCsystem 330 PROM Monitor

The boot PROM stores a program known as the "monitor" that contains self-tests and controls system operation during boot-up until the SunOS kernel takes over. The monitor program is invoked when you use the halt SunOS, and is identified with the ">" prompt.

The PROM monitor program provides commands that perform a variety of tasks, such as booting from alternate devices, changing the console output to a serial port, and so on.

Access the monitor as described in the Manual,*PROM*User's then enter **h** to view a "Help" table of monitor commands that looks something like this:

```
Monitor        REV:1              date              Help Menu

1  b           Boot a program.
2  k           Reset all or part of the machine or display the banner.
3  u           Initialize the input and output devices.
4  c/g/w       Resume or modify program flow.
5  d/r         Display and/or modify the registers.
6  o/e/l       Display and/or modify individual memory locations.
7  ^c/f/v       Copy, display or modify a block of memory.
8  m/p         Display and/or modify segment or page table entries.
9  q           Display and/or modify EEPROM locations.
10 s           Display or modify the Address Space Identifier.
11 i/j         Display and or modify cache data or tag entries.
12 n/y         Disable, enable, invalidate, or flush the cache.
13 ^a/^t       Display virtual and physical address information.
   ^i          Display firmware compilation information.
   ^p          Enable or disable parity circuitry.
    !          Re-execute the previously executed command.
   h/?         Enter the help system for the basic monitor commands.
   x           Enter the Extended Test System.

''option_number''= Additional Help <esc> = Go-To-Previous Menu q= quit

Command == >
```

If you need more help concerning any of the Help Menu options, enter the number that precedes the command, and then RETURN.

The "Monitor (8S)" section of the *Commands Reference Manual* also provides information on PROM monitor commands.

The monitor commands described in Chapter 12 of the *PROM User's Manual* are valid for the SPARCsystem 330, with the minor changes described below.

When using the **i, j** or **n** or **y** commands, you must specify the type of cache on which you want the operation performed. For example, you may enter the monitor **j** command, followed with **cpu** and the central processor cache will be opened for display or modification. If you replaced cpu with **io**, the I/O cache would be displayed or modified:

`j` *cpu cache_offset*

Refer to the "Displaying and Modifying Memory" section of the *PROM User's Manual* for more information on the use of these commands.

Monitor `x` Command

Add the following SPARCsystem 330 registers to the Processor Register table in the *PROM User's Manual* Sun-4 PROM Monitor Command descriptions:

Register Number	*Register Name*
0x80 - 0x87	g0,g1,g2,g3,g4,g5,g6,g7
0x88 - 0x8d	PSR,PC,nPC,WIM,TBR,Y
0x8e - 0xae	FSR,f0,...,f31

Monitor `s` Command

For SPARCsystem 330 CPU boards, you may enter the following Address Space Identifiers (ASI) after the s command. If you do not enter an (ASI) value, the current value will be displayed.

Name	*ASI (hex)*
Control Space	2
Segment Map	3
Page Map	4
Block Copy	5
User Instruction	8
Supervisor Instruction	9
User Data	a
Flush Page	d
Flush Context	e
Flush User	f

Monitor `t` Command

For the SPARCsystem 330, the t command displays or modifies the content of one or more region table entries. Refer to the *PROM User's Manual* for more information on using and modifying memory locations.

Monitor `v` Command

The "view" command, described in the *PROM User's Manual* For the SPARCsystem 330, pressing the space bar terminates the command. To freeze the display before it scrolls off the screen, press any key *other than the space bar*. To restart the `v` command, press any key except the space bar.

If you enter two asterisks (`* *`) at the end of the `v` command string (which includes the low and high virtual address and byte, word or long word specification), the view command will continue to read each location until you press the space bar. If you add only one asterisk, the content of each location will be displayed once.

Monitor ^a Command

For SPARCsystem 330 CPU boards, entering the ^ character followed by an **a** displays a list of devices and their corresponding physical and virtual addresses.

Monitor ^c Command

For SPARCsystem 330 CPU boards, entering the ^ character followed by an **c**, a source and destination virtual address, and the number of bytes, copies the content of those locations to another location.

Monitor ! Command

Entering an exclamation point after the PROM monitor prompt on a SPARCsystem 330 repeats the last executed basic monitor command.

Identifying A Faulty Memory Module

In the event a memory error is detected during the power-on self tests, the faulty SIMM will be identified by its "U Number" designation.

Additional memory tests are available in the PROM monitor Extended Test System. After entering the "Help" table shown at the beginning of the "The PROM Monitor" subsection, select **x** and enter the extended test system. Most of the extended tests are the same as those documented in Chapter 13 of the *PROM User's Manual.* Chapter 13 describes the extended test system user interface. You may enter command lines, or simply make menu selections. You need only enter the letters shown in upper case on the extended test menus.

NOTE *Entering an exclamation point displays the last five command lines you have entered. To repeat one of those command lines, simply enter the appropriate command line number.*

The tests that differ from those described for the Sun-4/200 and Sun-4/110 series of workstations are described in the next section.

SPARCsystem 330 Extended Tests

```
Monitor  REV:some number  Date  Main Menu


All             All Test Sequence.
Default         Default Test Sequence.
Boot            Boot Paths Menu.
Ethernet        Ethernet Menu.
Keyboard        Keyboard and Mouse Menu.
Memory          Memory Menu.
Serial          Serial Ports Menu.
Video           Video Menu.
Color           Color Map Menu.
Options         Set global flags/options.

? = Help <esc> = Return-To-Previous-Menu l = [n] = Repeat-Command-Line-n-Times  q= Quit
Command == >
```

The first Main Menu choice, `All`, brings up the tests shown below. The `Default` selection runs some of, but not all of these tests. Note that the Ethernet loopback connector, as well as the within-channel external loop-back cables for the keyboard, mouse, Serial Port A, and Serial Port B must be installed prior to running the `All` sequence. Contact Sun customer support to obtain a

loopback connector kit. The pin assignments for the various connectors appear in *The PROM User's Manual*, with the exception of the Ethernet loopback connector. The pin assignments for the Ethernet connector are:

Figure 10 *Ethernet Loopback Connector*

From Pin	*To Pin*
3	5
10	12
13	14

These tests are performed when the `All` test sequence is chosen from the Extended Test Main Menu for a black and white system:

```
        All Test Sequence

a. Ethernet Local Loop Back Test
b. Ethernet Encoder Loop Back Test
c. Ethernet External Loop Back Test
d. Keyboard Register 12 Test
e. Mouse Register 12 Test
f. Keyboard Transmit Test
g. Mouse Transmit Test
h. Keyboard Internal Loop Back Test
i. Mouse Internal Loop Back Test
j. Keyboard External Loop Back Test
k. Mouse External Loop Back Test
l. Main Memory Address Test/Byte Mode
m. Main Memory Constant Pattern Test/Word Mode
n. Main Memory Fill Utility/Long Mode
o. Main Memory Check Utility/Long Mode
p. Serial Port A Register 12 Test
q. Serial Port B Register 12 Test
r. Serial Port A Transmit Test
s. Serial Port B Transmit Test
t. Serial Port A Internal Loop Back Test
u. Serial Port B Internal Loop Back Test
v. Serial Port A External Loop Back Test
w. Serial Port B External Loop Back Test
x. Video Address Test/Byte Mode
y. Video Constant Pattern Test/Word Mode
z. Video Fill Utility/Long Mode
aa. Video Check Utility/Long Mode
Cmd=>
```

If the SPARCsystem 330 has a color monitor, these tests are also performed:

```
Overlay Plane Address Test/Byte Mode
Overlay Plane Constant Pattern Test/Word Mode
Overlay Plane Fill Utility/Long Mode
Overlay Plane Check Utility/Long Mode
Enable Plane Address Test/Byte Mode
Enable Plane Constant Pattern Test/Word Mode
Enable Plane Fill Utility/Long Mode
Enable Plane Check Utility/Long Mode
Color Plane Address Test/Byte Mode
Color Plane Constant Pattern Test/Word Mode
Color Plane Fill Utility/Long Mode
Color Plane Check Utility/Long Mode
Color Map Address Test
Color Map Constant Pattern Test
Color Map Fill Utility
Color Map Check Utility
```

Consult the Sun-4 Extended Test System chapter of the *PROM User's Manual* for descriptions of the tests listed above. The Extended Test System user interface is the same for all Sun-4 systems. The color tests function exactly as the Video tests described in Chapter 13 of the *PROM User's Manual*, except that they test the overlay, enable and color planes rather than the video circuitry. Some SPARCsystem extended tests documented in the *PROM User's Manual* may be labeled "for Sun-4/2xx only" or "for Sun-4/110 only". If the test is identical to one of the SPARCsystem 330 tests, please ignore the label.

SPARCsystem 330 Initialization

After error-free completion of the power-up self-tests, but before execution of the default boot sequence, the PROM initializes the system. The initialization sequence is:

1. The system enable register is set to normal state.
2. The context register is set to zero.
3. The processor status register is initialized: the CPU is in supervisor mode, window zero is active, level 14 and 15 interrupts are allowed, and traps are enabled. If the floating point processor probe was successful, the Floating Point Unit is enabled.
4. The virtual address of the trap table is written into the trap base register.
5. The window invalid mask register is initialized to 0x2.
6. All page table entries are initialized.
7. Memory is sized.
8. All available main memory is initialized to zero (an unimplemented instruction.)
9. The initial 32 Megabytes of working memory is mapped in ascending order, starting at location zero.

10. One page of RAM is reserved for a stack area and the stack pointer is initialized.
11. A page of RAM is reserved for the PROM monitor global variables.
12. An additional page of RAM is reserved for the monitor's trap vecor table and font table.
13. Assuming that they exist, these devices are mapped and initialized:

    Parity memory registers

    The EEPROM

    Other PROMs

    Ethernet

    The interrupt register

    The keyboard and mouse

    Serial ports

    Time of day clock

    P4 black and white board

    P4 color board
14. The copy of the monitor's trap vector table, which contains the addresses of the trap and interrupt handling routines, is set up.
15. Central processor cache is initialized.
16. Entry points to all support routines provided by the PROM monitor are set up.

Index

NAME

intro – introduction to commands

DESCRIPTION

This section describes publicly accessible commands in alphabetic order. Commands of general utility, many with enhancements from 4.3 BSD. Wherever possible, we have incorporated System V versions of commands and utilities into our standard UNIX release. Where a command has both a System V and a BSD version and it has been possible to merge them, we have done so. In some cases, where the System V version is compatible with BSD and offers significant added value, SunOS incorporates that version as its standard.

Pages of special interest have been categorized as follows:

1C Commands for communicating with other systems.

1G Commands used primarily for graphics and computer-aided design.

1V Commands that only have System V versions, or that have separate BSD and System V versions.

These commands either depend upon System V functionality, or have incompatibilities with the corresponding BSD version. They are included in the System V Software installation option. Once installed, they can be found in the directory **/usr/5bin**.

SEE ALSO

- Section 8 in this manual for system administration procedures, system maintenance and operation commands, local daemons, and network-services servers.
- Section 7 in this manual for descriptions of publicly available files and macro packages for document preparation.
- Section 6 in this manual for computer games.
- *Getting Started with SunOS: Beginner's Guide*
- *Setting Up Your SunOS Environment: Beginner's Guide*
- *SunView 1 Beginner's Guide*
- *Using the Network: Beginner's Guide*
- *Doing More with SunOS: Beginner's Guide*
- *Programming Utilities and Libraries*

DIAGNOSTICS

Upon termination each command returns two bytes of status, one supplied by the system giving the cause for termination, and (in the case of "normal" termination) one supplied by the program, see **wait** and **exit**(2). The former byte is 0 for normal termination, the latter is customarily 0 for successful execution, nonzero to indicate troubles such as erroneous parameters, bad or inaccessible data, or other inability to cope with the task at hand. It is called variously "exit code," "exit status" or "return code," and is described only where special conventions are involved.

LIST OF COMMANDS

Name	Appears on Page	Description
%	**csh**(1)	C shell built-in commands
@	**csh**(1)	C shell built-in commands
Mail	**mail**(1)	read or send mail messages
adb	**adb**(1)	general-purpose debugger
addbib	**addbib**(1)	create or extend a bibliographic database
adjacentscreens	**adjacentscreens**(1)	connect multiple screens to SunView window driver
admin	**admin**(1)	create and administer SCCS files
aedplot	**plot**(1G)	graphics filters for various plotters
alias	**csh**(1)	C shell built-in commands
align_equals	**textedit_filters**(1)	filters provided with **textedit**(1)
ar	**ar**(1V)	create library archives, and add or extract files
arch	**arch**(1)	display the architecture of the current host
as	**as**(1)	Sun-1, Sun-2 and Sun-3, Sun-4 and Sun386i assemblers
at	**at**(1)	execute a command or script at a specified time
atq	**atq**(1)	display the queue of jobs to be run at specified times
atrm	**atrm**(1)	remove jobs spooled by at or batch
awk	**awk**(1)	pattern scanning and processing language
banner	**banner**(1V)	display a string in large letters
bar	**bar**(1)	create tape archives, and add or extract files
basename	**basename**(1)	display portions of pathnames and filenames
batch	**at**(1)	execute a command or script at a specified time
bc	**bc**(1)	arbitrary-precision arithmetic language
bg	**csh**(1)	C shell built-in commands
bgplot	**plot**(1G)	graphics filters for various plotters
biff	**biff**(1)	give notice of incoming mail messages
binmail	**binmail**(1)	an early program for processing mail messages
break	**csh**(1)	C shell built-in commands
breaksw	**csh**(1)	C shell built-in commands
cal	**cal**(1)	display a calendar
calendar	**calendar**(1)	a simple reminder service
capitalize	**textedit_filters**(1)	filters provided with **textedit**(1)
case	**csh**(1)	C shell built-in commands
cat	**cat**(1V)	concatenate and display
cb	**cb**(1)	a simple C program beautifier
cc	**cc**(1V)	C compiler
cd	**cd**(1)	change working directory
cdc	**cdc**(1)	change the delta commentary of an SCCS delta
cflow	**cflow**(1)	generate a flow graph for a C program
checkeq	**eqn**(1)	typeset mathematics
checknr	**checknr**(1)	check nroff and troff input files; report possible errors
chfn	**passwd**(1)	change password file information
chgrp	**chgrp**(1)	change the group ownership of a file
chkey	**chkey**(1)	change your encryption key
chmod	**chmod**(1V)	change the permissions mode of a file
chsh	**passwd**(1)	change password file information
clear	**clear**(1)	clear the terminal screen
clear_colormap	**clear_colormap**(1)	clear the colormap to make console text visible
clear_functions	**clear_functions**(1)	reset the selection service to clear stuck function keys
click	**click**(1)	enable or disable the keyboard's keystroke click
clock	**clock**(1)	display the time in an icon or window

cluster	**cluster**(1)	find the Sun386i cluster containing a file
cmdtool	**cmdtool**(1)	run a shell (or program) using the SunView text facility
cmp	**cmp**(1)	perform a byte-by-byte comparison of two files
col	**col**(1V)	filter reverse paper motions from nroff for display
colcrt	**colcrt**(1)	filter nroff output for a terminal lacking overstrike
coloredit	**coloredit**(1)	alter color map segment
colrm	**colrm**(1)	remove characters from specified columns within each line
comb	**comb**(1)	combine SCCS deltas
comm	**comm**(1)	display lines in common between two sorted lists
compress	**compress**(1)	compress or expand files, display expanded contents
continue	**csh**(1)	C shell built-in commands
cp	**cp**(1)	copy files
cpio	**cpio**(1)	copy file archives in and out
cpp	**cpp**(1)	the C language preprocessor
crontab	**crontab**(1)	install, edit, remove or list a user's crontab file
crtplot	**plot**(1G)	graphics filters for various plotters
crypt	**crypt**(1)	encode or decode a file
csh	**csh**(1)	a shell with a C-like syntax
csplit	**csplit**(1)	split a file with respect to a given context
ctags	**ctags**(1)	create a tags file for use with vi
ctrace	**ctrace**(1)	generate a C program execution trace
cu	**tip**(1C)	terminal emulator, telephone connection to a remote system
cut	**cut**(1)	remove selected fields from each line of a file
cxref	**cxref**(1)	generate a C program cross-reference
date	**date**(1V)	display or set the date
dbx	**dbx**(1)	source-level debugger
dbxtool	**dbxtool**(1)	SunView interface for the dbx source-level debugger
dc	**dc**(1)	desk calculator
dd	**dd**(1)	convert and copy files with various data formats
default	**csh**(1)	C shell built-in commands
defaults_from_input	**defaultsedit**(1)	create or edit default settings for SunView 1
defaults_from_input	**input_from_defaults**(1)	update the mouse and keyboard from defaults
defaults_to_indentpro	**defaultsedit**(1)	create or edit default settings for SunView 1
defaults_to_mailrc	**defaultsedit**(1)	create or edit default settings for SunView 1
defaultsedit	**defaultsedit**(1)	create or edit default settings for SunView 1
delta	**delta**(1)	make a delta (change) to an SCCS file
deroff	**deroff**(1)	remove nroff, troff, tbl and eqn constructs
des	**des**(1)	encrypt or decrypt data using Data Encryption Standard
df	**df**(1)	report free disk space on file systems
diff3	**diff3**(1V)	display line-by-line differences between 3 files
diff	**diff**(1)	display line-by-line differences between pairs of text files
diffmk	**diffmk**(1)	mark differences between versions of a troff input file
dircmp	**dircmp**(1V)	compare directories
dirname	**basename**(1)	display portions of pathnames and filenames
dirs	**csh**(1)	C shell built-in commands
dis	**dis**(1)	object code disassembler for COFF
domainname	**domainname**(1)	set or display name of the current YP domain
dos2unix	**dos2unix**(1)	convert text file from DOS format to SunOS format
dos	**dos**(1)	SunView window for IBM PC/AT applications
du	**du**(1V)	display the number of disk blocks used per directory or file
dumbplot	**plot**(1G)	graphics filters for various plotters
e	**ex**(1)	line editor
echo	**echo**(1V)	echo arguments to the standard output

ed	**ed**(1)	basic line editor
edit	**ex**(1)	line editor
egrep	**grep**(1V)	search a file for a string or regular expression
eject	**eject**(1)	eject floppy disk from an autoeject floppy drive
else	**csh**(1)	C shell built-in commands
end	**csh**(1)	C shell built-in commands
endif	**csh**(1)	C shell built-in commands
endsw	**csh**(1)	C shell built-in commands
enroll	**xsend**(1)	send or receive secret mail
env	**env**(1)	obtain or alter environment variables
eqn	**eqn**(1)	typeset mathematics
error	**error**(1)	categorize compiler error messages
eval	**csh**(1)	C shell built-in commands
ex	**ex**(1)	line editor
exec	**csh**(1)	C shell built-in commands
exit	**csh**(1)	C shell built-in commands
expand	**expand**(1)	expand TAB characters to SPACE characters
expr	**expr**(1V)	evaluate arguments as an expression
false	**true**(1)	provide truth values
fdformat	**fdformat**(1)	format diskettes for use under SunOS
fg	**csh**(1)	C shell built-in commands
fgrep	**grep**(1V)	search a file for a string or regular expression
file	**file**(1)	determine the type of a file by examining its contents
find	**find**(1)	find files by name, or by other characteristics
finger	**finger**(1)	display information about users
fmt	**fmt**(1)	simple text and mail-message formatters
fmt_mail	**fmt**(1)	simple text and mail-message formatters
fold	**fold**(1)	fold long lines for display
fontedit	**fontedit**(1)	a vfont screen-font editor
foption	**foption**(1)	determine available floating-point code generation options
foreach	**csh**(1)	C shell built-in commands
from	**from**(1)	display the sender and date of newly-arrived mail messages
ftp	**ftp**(1C)	file transfer program
gcore	**gcore**(1)	get core images of running processes
get	**get**(1)	get a version of an SCCS file
get_selection	**get_selection**(1)	copy contents of a selection to the standard output
getopt	**getopt**(1)	parse command options in shell scripts
getoptcvt	**getopts**(1)	parse command options in shell scripts
getopts	**getopts**(1)	parse command options in shell scripts
gfxtool	**gfxtool**(1)	run graphics programs in a SunView window
gigiplot	**plot**(1G)	graphics filters for various plotters
glob	**csh**(1)	C shell built-in commands
goto	**csh**(1)	C shell built-in commands
gprof	**gprof**(1)	display call-graph profile data
graph	**graph**(1G)	draw a graph
grep	**grep**(1V)	search a file for a string or regular expression
groups	**groups**(1)	display a user's group memberships
hashstat	**csh**(1)	C shell built-in commands
head	**head**(1)	display first few lines of specified files
help	**help**(1)	ask for help regarding SCCS errors or warnings
help_viewer	**help_viewer**(1)	SunView help application
history	**csh**(1)	C shell built-in commands
hostid	**hostid**(1)	print the numeric identifier of the current host

hostname	**hostname**(1)	set or print name of current host system
hpplot	**plot**(1G)	graphics filters for various plotters
i386	**machid**(1)	return true if processor is of a given type
iAPX286	**machid**(1)	return true if processor is of a given type
iconedit	**iconedit**(1)	edit images for SunView icons, cursors and panels
id	**id**(1)	print the user name and ID, and group name and ID
if	**csh**(1)	C shell built-in commands
implot	**plot**(1G)	graphics filters for various plotters
indent	**indent**(1)	indent and format a C program source file
indentpro_to_defaults	**defaultsedit**(1)	create or edit default settings for SunView 1
indxbib	**indxbib**(1)	create an inverted index to a bibliographic database
inline	**inline**(1)	in-line procedure call expander
input_from_defaults	**defaultsedit**(1)	create or edit default settings for SunView 1
input_from_defaults	**input_from_defaults**(1)	update the mouse and keyboard from defaults
insert_brackets	**textedit_filters**(1)	filters provided with **textedit**(1)
install	**install**(1)	install files
ipcrm	**ipcrm**(1)	remove message queue, semaphore, shared memory ID
ipcs	**ipcs**(1)	report interprocess communication facilities status
jobs	**csh**(1)	C shell built-in commands
join	**join**(1)	relational database operator
keylogin	**keylogin**(1)	decrypt and store secret key
kill	**kill**(1)	send a signal to a process, or terminate a process
label	**csh**(1)	C shell built-in commands
last	**last**(1)	indicate last logins by user or terminal
lastcomm	**lastcomm**(1)	show the last commands executed, in reverse order
ld.so	**ld**(1)	link editor, dynamic link editor
ld	**ld**(1)	link editor, dynamic link editor
ldd	**ldd**(1)	list dynamic dependencies
leave	**leave**(1)	remind you when you have to leave
lex	**lex**(1)	lexical analysis program generator
limit	**csh**(1)	C shell built-in commands
line	**line**(1)	read one line
lint	**lint**(1V)	a C program verifier
ln	**ln**(1)	make hard or symbolic links to files
load	**load**(1)	load Sun386i clusters
loadc	**load**(1)	load Sun386i clusters
lockscreen	**lockscreen**(1)	maintain SunView context and prevent unauthorized access
lockscreen_default	**defaultsedit**(1)	create or edit default settings for SunView 1
lockscreen_default	**lockscreen**(1)	maintain SunView context and prevent unauthorized access
logger	**logger**(1)	add entries to the system log
login	**login**(1)	log in to the system
logname	**logname**(1)	get the name by which you logged in
logout	**csh**(1)	C shell built-in commands
look	**look**(1)	find words in the system dictionary or lines in a sorted list
lookbib	**lookbib**(1)	find references in a bibliographic database
lorder	**lorder**(1)	find an ordering relation for an object library
lpq	**lpq**(1)	display the queue of printer jobs
lpr	**lpr**(1)	send a job to the printer
lprm	**lprm**(1)	remove jobs from the printer queue
lptest	**lptest**(1)	generate lineprinter ripple pattern
ls	**ls**(1V)	list the contents of a directory
m4	**m4**(1V)	macro language processor
m68k	**machid**(1)	return true if processor is of a given type

mach	**mach**(1)	display the processor type of the current host
machid	**machid**(1)	return true if processor is of a given type
mail	**mail**(1)	read or send mail messages
mailrc_to_defaults	**defaultsedit**(1)	create or edit default settings for SunView 1
mailtool	**mailtool**(1)	SunView interface for the mail program
make	**make**(1)	maintain, update, and regenerate related programs and files
man	**man**(1)	display reference manual pages; find pages by keyword
mesg	**mesg**(1)	permit or deny messages on the terminal
mkdir	**mkdir**(1)	make a directory
mkstr	**mkstr**(1)	create an error message file by massaging C source files
more	**more**(1)	browse or page through a text file
mt	**mt**(1)	magnetic tape control
mv	**mv**(1)	move or rename files
neqn	**eqn**(1)	typeset mathematics
newgrp	**newgrp**(1)	log in to a new group
nice	**nice**(1)	run a command at low priority
nl	**nl**(1)	line numbering filter
nm	**nm**(1)	print name list
nohup	**nohup**(1V)	run a command immune to hangups and quits
notify	**csh**(1)	C shell built-in commands
nroff	**nroff**(1)	format documents for display or line-printer
objdump	**objdump**(1)	dump selected parts of a COFF object file
od	**od**(1V)	octal, decimal, hex, and ascii dump
oldccat	**oldcompact**(1)	compress and uncompress files, and cat them
oldcompact	**oldcompact**(1)	compress and uncompress files, and cat them
oldeyacc	**oldeyacc**(1)	modified yacc allowing much improved error recovery
oldfilemerge	**oldfilemerge**(1)	window-based file comparison and merging program
oldmake	**oldmake**(1)	maintain, update, and regenerate groups of programs
oldprmail	**oldprmail**(1)	display waiting mail
oldpti	**oldpti**(1)	phototypesetter interpreter
oldsetkeys	**oldsetkeys**(1)	modify interpretation of the keyboard
oldsun3cvt	**oldsun3cvt**(1)	convert Sun-2 executables for use on a Sun-3
oldsyslog	**oldsyslog**(1)	make a system log entry
oldtektool	**oldtektool**(1)	SunView Tektronix 4014 terminal-emulator window
olduncompact	**oldcompact**(1)	compress and uncompress files, and cat them
oldvc	**oldvc**(1)	version control
on	**on**(1C)	execute on remote system with local environment
onintr	**csh**(1)	C shell built-in commands
organizer	**organizer**(1)	file and directory manager
overview	**overview**(1)	run a program from SunView that takes over the screen
pack	**pack**(1)	compress and expand files
page	**more**(1)	browse or page through a text file
pagesize	**pagesize**(1)	display the size of a page of memory
passwd	**passwd**(1)	change password file information
paste	**paste**(1)	join lines of several files
pcat	**pack**(1)	compress and expand files
pdp11	**machid**(1)	return true if processor is of a given type
perfmeter	**perfmeter**(1)	display system performance values in a meter or strip chart
pg	**pg**(1V)	page through a file on a soft-copy terminal
plot	**plot**(1G)	graphics filters for various plotters
popd	**csh**(1)	C shell built-in commands
pr	**pr**(1V)	prepare file(s) for printing, perhaps in multiple columns
printenv	**printenv**(1)	display environment variables currently set

prof	**prof**(1)	display profile data
prs	**prs**(1)	display selected portions an SCCS history
prt	**prt**(1)	display the delta and commentary history of an SCCS file
ps	**ps**(1)	display the status of current processes
ptx	**ptx**(1)	generate a permuted index
pushd	**csh**(1)	C shell built-in commands
pwd	**pwd**(1)	display the pathname of the current working directory
quota	**quota**(1)	display a user's disk quota and usage
ranlib	**ranlib**(1)	convert archives to random libraries
rasfilter8to1	**rasfilter8to1**(1)	convert an 8-bit deep rasterfile to a 1-bit deep rasterfile
rastrepl	**rastrepl**(1)	magnify a raster image by a factor of two
rcp	**rcp**(1C)	remote file copy
rdist	**rdist**(1)	remote file distribution program
refer	**refer**(1)	expand and insert references from a bibliographic database
rehash	**csh**(1)	C shell built-in commands
repeat	**csh**(1)	C shell built-in commands
reset	**tset**(1)	establish or restore terminal characteristics
rev	**rev**(1)	reverse the order of characters in each line
rlogin	**rlogin**(1C)	remote login
rm	**rm**(1)	remove (unlink) files or directories
rmdel	**rmdel**(1)	remove a delta from an SCCS file
rmdir	**rm**(1)	remove (unlink) files or directories
roffbib	**roffbib**(1)	format and print a bibliographic database
rpcgen	**rpcgen**(1)	an RPC protocol compiler
rsh	**rsh**(1C)	remote shell
rup	**rup**(1C)	show host status of local machines (RPC version)
ruptime	**ruptime**(1C)	show host status of local machines
rusers	**rusers**(1C)	who's logged in on local machines (RPC version)
rwall	**rwall**(1C)	write to all users over a network
rwho	**rwho**(1C)	who's logged in on local machines
sact	**sact**(1)	print current SCCS file editing activity
sccs	**sccs**(1)	front end for the Source Code Control System (SCCS)
sccsdiff	**sccsdiff**(1)	compare two versions of an SCCS file
screenblank	**screenblank**(1)	turn off the screen when the mouse and keyboard are idle
screendump	**screendump**(1)	dump a frame-buffer image to a file
screenload	**screenload**(1)	load a frame-buffer image from a file
script	**script**(1)	make typescript of a terminal session
scrolldefaults	**defaultsedit**(1)	create or edit default settings for SunView 1
sdiff	**sdiff**(1)	contrast two text files by displaying them side-by-side
sed	**sed**(1V)	stream editor
selection_svc	**selection_svc**(1)	SunView selection service
set	**csh**(1)	C shell built-in commands
setenv	**csh**(1)	C shell built-in commands
sh	**sh**(1)	the standard UNIX system shell
shelltool	**shelltool**(1)	run a shell (or command) in a SunView terminal window
shift	**csh**(1)	C shell built-in commands
shift_lines	**textedit_filters**(1)	filters provided with **textedit**(1)
size	**size**(1)	display the size of an object file
sleep	**sleep**(1)	suspend execution for a specified interval
snap	**snap**(1)	SunView application for system and network administration
soelim	**soelim**(1)	resolve and eliminate .so requests from nroff or troff input
sort	**sort**(1V)	sort and collate lines
sortbib	**sortbib**(1)	sort a bibliographic database

source	**csh**(1)	C shell built-in commands
sparc	**machid**(1)	return true if processor is of a given type
spell	**spell**(1)	report spelling errors
spellin	**spell**(1)	report spelling errors
spellout	**spell**(1)	report spelling errors
spline	**spline**(1G)	interpolate smooth curve
split	**split**(1)	split a file into pieces
stop	**csh**(1)	C shell built-in commands
strings	**strings**(1)	find printable strings in an object file or binary
strip	**strip**(1)	remove symbols and relocation bits from an object file
stty	**stty**(1V)	set or alter the options for a terminal
stty_from_defaults	**defaultsedit**(1)	create or edit default settings for SunView 1
stty_from_defaults	**stty_from_defaults**(1)	set terminal editing characters from the defaults database
su	**su**(1)	super-user, temporarily switch to a new user ID
sum	**sum**(1V)	calculate a checksum for a file
sun	**machid**(1)	return true if processor is of a given type
sunview	**sunview**(1)	the SunView window environment
suspend	**csh**(1)	C shell built-in commands
swin	**swin**(1)	set or get SunView user input options
switch	**csh**(1)	C shell built-in commands
switcher	**switcher**(1)	switch between multiple SunView desktops
symorder	**symorder**(1)	rearrange a list of symbols
sync	**sync**(1)	update the super block; force changed blocks to the disk
sysex	**sysex**(1)	invoke the system exerciser
syswait	**syswait**(1)	execute a command, suspending termination until user input
t300	**plot**(1G)	graphics filters for various plotters
t300s	**plot**(1G)	graphics filters for various plotters
t4013	**plot**(1G)	graphics filters for various plotters
t450	**plot**(1G)	graphics filters for various plotters
tabs	**tabs**(1V)	set tab stops on a terminal
tail	**tail**(1)	display the last part of a file
talk	**talk**(1)	talk to another user
tar	**tar**(1)	create tape archives, and add or extract files
tbl	**tbl**(1)	format tables for nroff or troff
tcopy	**tcopy**(1)	copy a magnetic tape
tcov	**tcov**(1)	construct test coverage analysis and statement profile
tee	**tee**(1)	replicate the standard output
tek	**plot**(1G)	graphics filters for various plotters
telnet	**telnet**(1C)	interface to remote system using TELNET protocol
test	**test**(1V)	return true or false according to a conditional expression
textedit	**textedit**(1)	SunView window- and mouse-based text editor
textedit_filters	**textedit_filters**(1)	filters provided with **textedit**(1)
tftp	**tftp**(1C)	trivial file transfer program
then	**csh**(1)	C shell built-in commands
time	**time**(1V)	time a command
tip	**tip**(1C)	terminal emulator, telephone connection to a remote system
toolplaces	**toolplaces**(1)	display SunView window locations
touch	**touch**(1V)	update the access and modification times of a file
tput	**tput**(1V)	initialize a terminal or query the terminfo database
tr	**tr**(1V)	translate characters
trace	**trace**(1)	trace system calls and signals
traffic	**traffic**(1C)	SunView program to display Ethernet traffic
troff	**troff**(1)	typeset or format documents

true	**true**(1)	provide truth values
tset	**tset**(1)	establish or restore terminal characteristics
tsort	**tsort**(1)	topological sort
tty	**tty**(1)	display the name of the terminal
u3b15	**machid**(1)	return true if processor is of a given type
u3b2	**machid**(1)	return true if processor is of a given type
u3b5	**machid**(1)	return true if processor is of a given type
u3b	**machid**(1)	return true if processor is of a given type
ul	**ul**(1)	do underlining
umask	**csh**(1)	C shell built-in commands
unalias	**csh**(1)	C shell built-in commands
uname	**uname**(1V)	display the name of the current system
uncompress	**compress**(1)	compress or expand files, display expanded contents
unexpand	**expand**(1)	expand TAB characters to SPACE characters
unget	**unget**(1)	undo a previous get of an SCCS file
unhash	**csh**(1)	C shell built-in commands
unifdef	**unifdef**(1)	resolve and remove ifdef'ed lines from cpp input
uniq	**uniq**(1)	remove or report adjacent duplicate lines
units	**units**(1)	conversion program
unix2dos	**unix2dos**(1)	convert text file from SunOS format to DOS format
unlimit	**csh**(1)	C shell built-in commands
unload	**unload**(1)	unload Sun386i clusters
unloadc	**unload**(1)	unload Sun386i clusters
unpack	**pack**(1)	compress and expand files
unset	**csh**(1)	C shell built-in commands
unsetenv	**csh**(1)	C shell built-in commands
uptime	**uptime**(1)	show how long the system has been up
users	**users**(1)	display a compact list of users logged in
uucp	**uucp**(1C)	system to system copy
uudecode	**uuencode**(1C)	encode a binary file, or decode its ASCII representation
uuencode	**uuencode**(1C)	encode a binary file, or decode its ASCII representation
uulog	**uucp**(1C)	system to system copy
uuname	**uucp**(1C)	system to system copy
uusend	**uusend**(1C)	send a file to a remote host
uustat	**uustat**(1C)	uucp status inquiry and job control
uux	**uux**(1C)	remote system command execution
vacation	**vacation**(1)	reply to mail automatically
val	**val**(1)	validate an SCCS file
vax	**machid**(1)	return true if processor is of a given type
vfontinfo	**vfontinfo**(1)	inspect and print out information about fonts
vgrind	**vgrind**(1)	grind nice program listings
vi	**vi**(1)	visual display editor based on **ex**(1)
view	**vi**(1)	visual display editor based on **ex**(1)
vplot	**vplot**(1)	plot graphics for a Versatec printer
vswap	**vswap**(1)	convert a foreign font file
vtroff	**vtroff**(1)	troff to a raster plotter
vwidth	**vwidth**(1)	make a troff width table for a font
w	**w**(1)	who is logged in, and what are they doing
wait	**wait**(1)	wait for a process to finish
wall	**wall**(1)	write to all users logged in
wc	**wc**(1)	display a count of lines, words and characters
what	**what**(1)	identify the version of files under SCCS
whatis	**whatis**(1)	display a one-line summary about a keyword

whereis	**whereis**(1)	locate binary, source, and manual page for a command
which	**which**(1)	locate a command; display its pathname or alias
while	**csh**(1)	C shell built-in commands
who	**who**(1)	who is logged in on the system
whoami	**whoami**(1)	display the effective current username
whois	**whois**(1)	DARPA Internet user name directory service
write	**write**(1)	write a message to another user
xargs	**xargs**(1)	construct the arguments list(s) and execute a command
xget	**xsend**(1)	send or receive secret mail
xsend	**xsend**(1)	send or receive secret mail
xstr	**xstr**(1)	extract strings from C programs to implement shared strings
yacc	**yacc**(1)	yet another compiler-compiler: parsing program generator
yes	**yes**(1)	be repetitively affirmative
ypcat	**ypcat**(1)	print values in a YP data base
ypmatch	**ypmatch**(1)	print the value of one or more keys from a YP map
yppasswd	**yppasswd**(1)	change your network password in the Yellow Pages
ypwhich	**ypwhich**(1)	which host is the YP server or map master?
zcat	**compress**(1)	compress or expand files, display expanded contents

NAME

ar – create library archives, and add or extract files

SYNOPSIS

ar d | m | p | q | r | t | x [[clouv] [abi *position-name* **]]** *archive* **[** *member-file*... **]**

SYSTEM V SYNOPSIS

ar [–] d | m | p | q | r | t | x [[clouvs] [abi *position-name* **]]** *archive* **[** *member-file*... **]**

DESCRIPTION

ar maintains groups of files combined into a single archive file. An archive file comprises a set of member files and header information for each file. The archive header and the headers for the file consist of printable characters (assuming that the names of the files in the archive contain printable characters), and are in a format portable across all machines. This format is described in detail in **ar**(5)). If an archive is composed of printable files, with printable file names, the entire archive is printable.

ar is normally used to create and update library files used by the link editor **ld**(1), but can be used for any similar purpose.

archive is the name of the archive file. *member-file* is a member file contained in the archive. If this argument is omitted, the command applies to all entries in the archive. Member names have a maximum of 15 characters, except on Sun386i systems, where they have a maximum of 14 characters, longer names are truncated.

SYSTEM V DESCRIPTION

ar runs **ranlib** after modifying an archive, so that the symbol table member of the archive is kept up-to-date.

OPTIONS

You must indicate only one of: **d, m, p, q, r, t,** or **x,** which may be followed by one or more **Modifiers.**

d Delete the named files from the archive file.

m Move the named files to the end of the archive.

p Print. If no names are given, all files in the archive are written to the standard output; if no names are given, only those files are written, and they are written in the order that they appear in the archive.

q Quick append. Append the named files to the end of the archive file without searching the archive for duplicate names. Useful only to avoid quadratic behavior when creating a large archive piece-by-piece. If this option is used to add a member to an archive, and a member with the same name as that member already exists in the archive, the old member will not be removed; two members with the same name will exist in the archive.

r Replace the named files in the archive.

t Table of contents. If no names are given, all files in the archive are listed; if names are given, only those files are listed.

x Extract. If no names are given, all files in the archive are extracted into the current directory; if names are given, only those files are extracted. In neither case does **x** alter the archive file.

Modifiers

c Create. Suppress the message that is produced by default when *archive* is created.

l Local. Place temporary files in the current working directory rather than in the default temporary directory, **/tmp**.

o Old date. When files are extracted with the **x** option, set the "last modified" date to the date recorded in the archive.

u Update. Replace only those files that have changed since they were put in the archive. Used with the **r** option.

v Verbose. When used with the **r**, **d**, **m**, or **q** option, give a file-by-file description of the creation of a new archive file from the old archive and the constituent files. When used with **x**, give a file-by-file description of the extraction of archive files. When used with **t**, give a long listing of information about the files. When used with **p**, write each member's name to the standard output before writing the member to the standard output.

a *position-name*
Place new files after *position-name* (*position-name* argument must be present). Applies only to the **r** and **m** options.

b *position-name*
Place new files before *position-name* (*position-name* argument must be present). Applies only to the **r** and **m** options.

i *position-name*
Identical to the **b** modifier.

SYSTEM V OPTIONS

The options may be preceded by '–'.

Modifiers

s Force the regeneration of the archive symbol table even if **ar** is not invoked with a command that will modify the archive contents.

EXAMPLES

Creating a new archive:

```
example% ar rcv archive file.o
a – file.o
```

Adding to an archive:

```
example% ar rav file.o archive next.c
a – next.c
```

Extracting from an archive:

```
example% ar xv archive file.o
x – file.o
example% ls file.o
file.o
```

Seeing the table of contents:

```
example% ar t archive
file.o
next.c
```

FILES

/tmp/v*.	temporary files
/tmp	

SEE ALSO

ld(1), **lorder**(1), **ranlib**(1), **ar**(5)

BUGS

If the same file is mentioned twice in an argument list, it is put in the archive twice.

The ''last-modified'' date of a file will not be altered by the **o** option unless the user is either the owner of the extracted file or the super-user.

NAME

arch – display the architecture of the current host

SYNOPSIS

arch
arch –k
arch *archname*

DESCRIPTION

arch displays the application architecture of the current host system.

Sun systems can be broadly classified by their *architectures*, which define what executables will run on which machines. A distinction can be made between *kernel architecture* and *application architecture* (or, commonly, just "architecture"). Machines which run different kernels due to underlying hardware differences may yet be able to run the same application programs. For example, the MC68020-based Sun-3/160 system and the MC68030-based Sun-3/80 system run different SunOS kernels, but can execute the same applications. One could describe these machines as having "the same application architecture" but "different kernel architectures."

Executing **arch** will display the application architecture of the machine, such as **sun3**, **sun4**, etc. All machines of the same application architecture will execute the same application programs, or user binaries.

Current Architectures

Application architecture	Kernel architecture	Current Sun System Models
sun2	sun2	2/50, 2/120
sun3	sun3	3/50, 3/60, 3/75, 3/110, 3/140, 3/160, 3/180, 3/260, 3/280
sun3	sun3x	3/80, 3/460, 3/470, 3/480
sun4	sun4	4/110, 4/260, 4/280, SPARCsystem 330
sun386	sun386	386*i*/150, 386*i*/250

OPTIONS

–k Display the kernel architecture, such as **Sun3**, **Sun3x**, etc. This defines which specific SunOS kernel will run on the machine, and has implications only for programs that depend on the kernel explicitly (for example, **ps**(1), **vmstat**(1), etc.).

archname Return "true" (exit status 0) if user binaries for *archname* can run on the current host system, otherwise, return "false" (exit status 1). This is the preferred method for installation scripts to determine the environment of the host machine; that is, which architecture of a multi-architecture release to install on this machine. *archname* must be a valid application architecture.

SEE ALSO

mach(1), **machid**(1)

Installing the SunOS 4.0.3

NAME

cpp – the C language preprocessor

SYNOPSIS

/usr/lib/cpp [**–BCHMpPRT**] [**–undef**] [**–D***name*] [**–D***name=def*] [**–I***directory*] [**–U***name*] [**–Y***directory*] [*input-file* [*output-file*]]

DESCRIPTION

cpp is the C language preprocessor. It is invoked as the first pass of any C compilation started with the **cc**(1V) command, and may also used as a first-pass preprocessor for other Sun compilers.

Although **cpp** can be used as a macro processor, this is not normally recommended, as its output is geared toward that which would be acceptable as input to a compiler's second pass. Thus, the preferred way to invoke **cpp** is through the **cc**(1V), or another compilation command. For general-purpose macro-processing, see **m4**(1V), and the *Programming Utilities and Libraries.*

cpp optionally accepts two filenames as arguments. *input-file* and *output-file* are, respectively, the input and output files for the preprocessor. They default to the standard input and the standard output.

OPTIONS

–B Support the C++ comment indicator '//'. With this indicator everything on the line after the // is treated as a comment.

–C Pass all comments (except those that appear on **cpp** directive lines) through the preprocessor. By default, **cpp** strips out C-style comments.

–H Print the pathnames of included files, one per line on the standard error.

–M Generate a list of makefile dependencies and write them to the standard output. This list indicates that the object file which would be generated from the input file depends on the input file as well as the include files referenced.

–p Use only the first eight characters to distinguish preprocessor symbols, and issue a warning if extra tokens appear at the end of a line containing a directive.

–P Preprocess the input without producing the line control information used by the next pass of the C compiler.

–R Allow recursive macros.

–T Use only the first eight characters for distinguishing different preprocessor names. This option is included for backward compatibility with systems which always use only the first eight characters.

–undef Remove initial definitions for all predefined symbols.

–D*name* Define *name* as 1 (one). This is the same as if a **–D***name*=**1** option appeared on the **cpp** command line, or as if a

#define *name* **1**

line appeared in the source file that **cpp** is processing.

–D*name=def* Define *name* as if by a **#define** directive. This is the same as if a

#define *name* **def**

line appeared in the source file that **cpp** is processing. The **–D** option has lower precedence than the **–U** option. That is, if the same name is used in both a **–U** option and a **–D** option, the name will be undefined regardless of the order of the options.

–I*directory* Insert *directory* into the search path for **#include** files with names beginning with '/'. *directory* is inserted ahead of the standard list of "include" directories. Thus, **#include** files with names enclosed in double-quotes (") are searched for first in the directory of the file with the **#include** line, then in directories named with **–I** options, and lastly, in directories from the standard list. For **#include** files with names enclosed in angle-brackets (< >), the directory of the file with the **#include** line is not searched. See **Details** below for exact details of this search order.

–U*name* Remove any initial definition of *name*, where *name* is a symbol that is predefined by a particular preprocessor. Here is a partial list of symbols that may be predefined, depending upon the application architecture of the system:

Operating System:	**ibm, gcos, os, tss** and **unix**
Hardware:	**interdata, pdp11, u370, u3b, u3b2, u3b5, u3b15, u3b20d, vax, mc68000, mc68010, mc68020, ns32000, iAPX286, i386, sparc,** and **sun**
UNIX system variant:	**RES,** and **RT**
The **lint**(1V) command:	**lint**

The symbols **sun** and **unix** are defined for all Sun systems, as is the value returned by the **mach** command. For Sun-4 systems, this value would be **sparc**. In addition, **mc68000** is defined for Sun-2, Sun-3, and Sun-3x systems.

–Y*directory* Use directory *directory* in place of the standard list of directories when searching for **#include** files.

USAGE

Directives

All **cpp** directives start with a hash symbol (#) as the first character on a line. White space (SPACE or TAB characters) can appear after the initial # for proper indentation.

#define *name token-string*

Replace subsequent instances of *name* with *token-string*.

#define *name*(*argument* [, *argument*] ...) *token-string*

There can be no space between *name* and the '('. Replace subsequent instances of *name*, followed by a parenthesized list of arguments, with *token-string*, where each occurrence of an *argument* in the *token-string* is replaced by the corresponding token in the comma-separated list. When a macro with arguments is expanded, the arguments are placed into the expanded *token-string* unchanged. After the entire *token-string* has been expanded, **cpp** re-starts its scan for names to expand at the beginning of the newly created *token-string*.

#undef *name*

Remove any definition for the symbol *name*. No additional tokens are permitted on the directive line after *name*.

#include "*filename* **"**
#include <*filename***>**

Read in the contents of *filename* at this location. This data is processed by **cpp** as if it were part of the current file. When the **<***filename***>** notation is used, *filename* is only searched for in the standard "include" directories. See the **–I** and **–Y** options above for more detail. No additional tokens are permitted on the directive line after the final '"' or '>'.

#line *integer-constant "filename"*

Generate line control information for the next pass of the C compiler. *integer-constant* is interpreted as the line number of the next line and *filename* is interpreted as the file from where it comes. If *"filename"* is not given, the current filename is unchanged. No additional tokens are permitted on the directive line after the optional *filename*.

#if *constant-expression*

Subsequent lines up to the matching **#else**, **#elif** , or **#endif** directive, appear in the output only if *constant-expression* yields a nonzero value. All binary non-assignment C operators, including '&&', '||', and ',', are legal in *constant-expression*. The '**?:**' operator, and the unary '–', '**!**', and '~' operators, are also legal in *constant-expression*.

The precedence of these operators is the same as that for C. In addition, the unary operator **defined**, can be used in *constant-expression* in these two forms: '**defined** (*name*)' or '**defined** *name*'. This allows the effect of **#ifdef** and **#ifndef** directives (described below) in the **#if** directive. Only these operators, integer constants, and names that are known by **cpp** should be used within *constant-expression*. In particular, the **sizeof** operator is not available.

#ifdef *name*

Subsequent lines up to the matching **#else**, **#elif**, or **#endif** appear in the output only if *name* has been defined, either with a **#define** directive or a **–D** option, and in the absence of an intervening **#undef** directive. No additional tokens are permitted on the directive line after *name*.

#ifndef *name*

Subsequent lines up to the matching **#else**, **#elif**, or **#endif** appear in the output only if *name* has *not* been defined, or if its definition has been removed with an **#undef** directive. No additional tokens are permitted on the directive line after *name* .

#elif *constant-expression*

Any number of **#elif** directives may appear between an **#if**, **#ifdef**, or **#ifndef** directive and a matching **#else** or **#endif** directive. The lines following the **#elif** directive appear in the output only if all of the following conditions hold:

- The *constant-expression* in the preceding **#if** directive evaluated to zero, the *name* in the preceding **#ifdef** is not defined, or the *name* in the preceding **#ifndef** directive *was* defined.
- The *constant-expression* in all intervening **#elif** directives evaluated to zero.
- The current *constant-expression* evaluates to non-zero.

If the *constant-expression* evaluates to non-zero, subsequent **#elif** and **#else** directives are ignored up to the matching **#endif**. Any *constant-expression* allowed in an **#if** directive is allowed in an **#elif** directive.

#else This inverts the sense of the conditional directive otherwise in effect. If the preceding conditional would indicate that lines are to be included, then lines between the **#else** and the matching **#endif** are ignored. If the preceding conditional indicates that lines would be ignored, subsequent lines are included in the output. Conditional directives and corresponding **#else** directives can be nested.

#endif End a section of lines begun by one of the conditional directives **#if**, **#ifdef**, or **#ifndef**. Each such directive must have a matching **#endif**.

Macros

Formal parameters for macros are recognized in **#define** directive bodies, even when they occur inside character constants and quoted strings. For instance, the output from:

```
#define abc(a) | \a|
abc(xyz)
```

is the seven characters " |`xyz|" (SPACE, vertical-bar, backquote, x, y, z, vertical-bar). Macro names are not recognized within character constants or quoted strings during the regular scan. Thus:

```
#define abc xyz
printf("abc");
```

does not expand **abc** in the second line, since it is inside a quoted string that is not part of a **#define** macro definition.

Macros are not expanded while processing a **#define** or **#undef**. Thus:

```
#define abc zingo
#define xyz abc
#undef abc
xyz
```

produces **abc**. The token appearing immediately after an **#ifdef** or **#ifndef** is not expanded.

Macros are not expanded during the scan which determines the actual parameters to another macro call. Thus:

```
#define reverse(first,second)second first
#define greeting hello
reverse(greeting,
#define greeting goodbye
)
```

produces " **#define hello goodbye hello**".

Output

Output consists of a copy of the input file, with modifications, plus lines of the form:

#lineno " *filename* " "*level*"

indicating the original source line number and filename of the following output line and whether this is the first such line after an include file has been entered (*level*=**1**), the first such line after an include file has been exited (*level*=**2**), or any other such line (*level* is empty).

Details

Directory Search Order

#include files is:

1. The directory of the file that contains the **#include** request (that is, **#include** is relative to the file being scanned when the request is made).
2. The directories specified by **–I** options, in left-to-right order.
3. The standard directory(s) (**/usr/include** on SunOS systems).

Special Names

Two special names are understood by **cpp**. The name _ _LINE_ _ is defined as the current line number (a decimal integer) as known by **cpp**, and _ _FILE_ _ is defined as the current filename (a C string) as known by **cpp**. They can be used anywhere (including in macros) just as any other defined name.

Newline Characters

A NEWLINE character terminates a character constant or quoted string. An escaped NEWLINE (that is, a backslash immediately followed by a NEWLINE) may be used in the body of a **#define** statement to continue the definition onto the next line. The escaped NEWLINE is not included in the macro value.

Comments

Comments are removed (unless the –C option is used on the command line). Comments are also ignored, except that a comment terminates a token.

FILES

/usr/include	standard directory for **#include** files

SEE ALSO

cc(1V), **m4**(1V)

Programming Utilities and Libraries

DIAGNOSTICS

The error messages produced by **cpp** are intended to be self-explanatory. The line number and filename where the error occurred are printed along with the diagnostic.

NOTES

When NEWLINE characters were found in argument lists for macros to be expanded, some previous versions of **cpp** put out the NEWLINE characters as they were found and expanded. The current version of **cpp** replaces them with SPACE characters.

Because the standard directory for included files may be different in different environments, this form of **#include** directive:

#include <file.h>

should be used, rather than one with an absolute path, like:

#include "/usr/include/file.h"

cpp warns about the use of the absolute pathname.

NAME

crontab – install, edit, remove or list a user's crontab file

SYNOPSIS

crontab [*filename*]
crontab –e [*username*]
crontab –l [*username*]
crontab –r [*username*]

DESCRIPTION

crontab copies the specified file, or the standard input if no file is specified, into a directory that holds all users' **crontab** files. A user's **crontab** file lists commands that are to be executed on behalf of that user at specified times on specified dates; the format of these files is described in **crontab**(5).

If the file **/var/spool/cron/at.allow** exists, only users whose username appears in it can use **crontab**. If that file does *not* exist, however, **crontab** checks the **/var/spool/cron/at.deny** file to determine if the user should be denied the use of **crontab**. If neither file exists, only the super-user is allowed to submit a **crontab** job. If **at.allow** does not exist and **at.deny** exists and is empty, global usage is permitted. The allow/deny files consist of one user name per line.

OPTIONS

–e Make a copy of the current user's **crontab** file, or create an empty file if it does not exist, and edit that file. The **vi**(1) editor will be used unless the environment variable **VISUAL** or **EDITOR** indicates an alternate editor. When editing is complete, install the file as the user's **crontab** file if it was modified. If a *username* is given, the specified user's **crontab** file is edited, rather than the current user's **crontab** file; this may only be done by the super-user.

–l List the user's **crontab** file.

–r Remove the current user's **crontab** file from the **crontab** directory. If a *username* is given, the specified user's **crontab** file is removed, rather than the current user's **crontab** file; this may only be done by the super-user.

FILES

/var/spool/cron	main cron directory
/var/spool/cron/crontabs	spool area
/var/spool/cron/at.allow	list of allowed users
/var/spool/cron/at.deny	list of denied users

SEE ALSO

sh(1), **crontab**(5), **cron**(8)

WARNINGS

If you inadvertently enter the **crontab** command with no argument, do not attempt to get out by typing CTRL-D. This removes all entries in your **crontab** file. Instead, exit by typing your interrupt character (normally CTRL-C).

NAME

ed – basic line editor

SYNOPSIS

ed [–] [**–sx**] [**–p** *string*] [*filename*]

DESCRIPTION

ed is the most basic line editor of the UNIX system. Although superseded by **ex**(1) and **vi**(1) for most purposes, **ed** is still used by various system utilities.

ed operates on a copy of *filename*, called a buffer, and overwrites a file only when you issue the **w** (write) command. **ed** provides line oriented editing commands to display or change lines, to insert and delete lines from the buffer, to move or copy lines within the buffer, or to substitute character strings within lines.

OPTIONS

–

–s Suppress printing of character counts (by **e**, **r**, and **w** commands), diagnostics (by **e** and **q** commands), and the **!** prompt (after a **!** command). Also, suppress printing the **?** diagnostic before overwriting unsaved changes in the buffer.

–x Edit an encrypted file (see **crypt**(1) for details).

–p *string* Use *string* as the editing prompt in command mode.

USAGE

Command Structure

ed commands have a simple and regular structure. They consist of an optional *address*, or two optional *address*es separated by a comma or semicolon, then a single-character *command*, which may be followed by a *parameter* for that *command*:

[*address*[,*address*]] *command* [*parameter*]

If only one *address* is specified, operations are performed on that line. If two *address*es are specified, **ed** performs the operation on the inclusive range of lines. Commands that requires an *address* use certain addresses by default, typically the address of the current line.

For example, **1,10p** means "print (display) lines 1 through 10" (two addresses), **5a** means "append text after line 5" (one address), and **d** means "delete the current line" (no address with the current line used as default). The meaning of *parameter* varies for each operation — for the move (**m**) and transfer (**t**) operations, for instance, it is the line that the addressed lines are to be moved to or transferred after. For reading (**r**) and writing (**w**) a file, *parameter* specifies the name of the file that is to be read or written.

ed is extremely terse in its interaction with the user. Its normal response to most problems is simply a question mark (**?**). This may happen when **ed** cannot find a specified line in the buffer, or if a search for a regular expression fails in a substitute (**s**) command. The **h** command prints a somewhat more complete diagnostic for the most recent error encountered; the **H** command requests that the diagnostic be printed for all errors.

Addresses

Lines can be addressed in several ways:

nnn By line number. Lines in the buffer are numbered relative to the start of the buffer. When displayed, line numbers are not physically present with the text of the file or buffer.

$ The last line of the buffer.

. The current line. **ed** keeps track of the line on which you last performed an operation. This line is called the *current line*. You can address this line by typing a "dot" character.

±*n* By relative line number. Address the line number that is *n* lines higher, or *n* lines lower than the current line.

'*c* Address the line marked with the mark character *c*, which must be a lower-case letter. Lines are marked with the **k** command, described below.

/RE/ An *RE* is a Regular Expression, described under **Regular Expressions** below. When enclosed by slashes, *RE* addresses the first line found by searching for a matching string. The search proceeds forward from the line following the current line, and wraps through the beginning of the buffer to include all preceding lines, as well as the current line.

?RE? An *RE* enclosed in question marks addresses the first line containing a match found by searching backward from the line preceding the current line. The search wraps through the end of the buffer to include all lines following the current line (in reverse order), as well as the current line.

address±n

An address followed by a plus sign (+) or a minus sign (–), followed by a decimal number, specifies that address plus or minus the indicated number of lines. (The plus sign may be omitted.) If the address is omitted, the current line is used as the base. For example, '**31–3**' addresses line 28 in the buffer.

address±

If an address ends with '+' or '–', then 1 is added to or subtracted from the address, respectively. As a consequence of this rule and the previous rule, the address '–' refers to the line preceding the current line. (To maintain compatibility with earlier versions of **ed**, the character '^' is equivalent to '–'.) Trailing '+' and '–' characters have a cumulative effect, so '––' refers to the current line, less 2.

, By itself, a comma stands for the address pair '**1,$**'.

; A semicolon by itself stands for the pair '**.,$**'.

By default for a given command. If you do not specify an address for a command to operate on, a command that requires an address supplies one by default. This is typically the current line.

A pair of addresses separated by a comma signifies an inclusive range of lines, and the current line is not changed unless the command changes it. When addresses are separated by a semicolon, however, the current line is set to the address preceding the semicolon before any subsequent addresses are interpreted. This feature can be used to determine the starting line for forward and backward searches using '/', and '?'.

The second address of any two-address sequence must correspond to a line that occurs later in the buffer than that of the first address.

Regular Expressions

ed supports a limited form of regular-expression notation, which can be used in a line address to specify lines by content. A regular expression (RE) specifies a set of character strings to match against — such as "any string containing digits 5 through 9" or "only lines containing uppercase letters." A member of this set of strings is said to be *matched* by the regular expression. Regular expressions or *patterns* are used to address lines in the buffer (see **Addresses** , above), and also for selecting strings to be replaced using the **s** (substitute) command.

Where multiple matches are present in a line, a regular expression matches the the *longest* of the *leftmost* matching strings.

Regular expressions can be built up from the following "single-character" RE's:

c Any ordinary character not listed below. An ordinary character matches itself.

\ Backslash. When followed by a special character, the RE matches the "quoted" character. A backslash followed by one of <, >, (,), {, or }, represents an *operator* in a regular expression, as described below.

. Dot. Matches any single character except NEWLINE.

^ As the leftmost character, a caret (or circumflex) constrains the RE to match the leftmost portion of a line. A match of this type is called an "anchored match" because it is "anchored" to a specific place in the line. The ^ character loses its special meaning if it appears in any position other than the start of the RE.

$** As the rightmost character, a dollar sign constrains the RE to match the rightmost portion of a line. The **$ character loses its special meaning if it appears in any position other than at the end of the RE.

^*RE***$** The construction ^*RE* **$** constrains the RE to match the entire line.

\< The sequence \< in an RE constrains the one-character RE immediately following it only to match something at the beginning of a "word"; that is, either at the beginning of a line, or just before a letter, digit, or underline and after a character not one of these.

\> The sequence \> in an RE constrains the one-character RE immediately following it only to match something at the end of a "word."

[*c* . . .] A nonempty string of characters, enclosed in square brackets matches any single character in the string. For example, **[abcxyz]** matches any single character from the set **'abcxyz'**. When the first character of the string is a caret (^), then the RE matches any character *except* NEWLINE and those in the remainder of the string. For example, **'[^45678]'** matches any character except **'45678'**. A caret in any other position is interpreted as an ordinary character.

[ ] *c* . . .] The right square bracket does not terminate the enclosed string if it is the first character (after an initial '^', if any), in the bracketed string. In this position it is treated as an ordinary character.

[*l*–*r*] The minus sign, between two characters, indicates a range of consecutive ASCII characters to match. For example, the range **'[0–9]'** is equivalent to the string **'[0123456789]'**. Such a bracketed string of characters is known as a *character class*. The '–' is treated as an ordinary character if it occurs first (or first after an initial ^) or last in the string.

d Delimiter character. The character used to delimit an RE within a command is special for that command (for example, see how / is used in the **g** command, below).

The following rules and special characters allow for constructing RE's from single-character RE's:

A concatenation of RE's matches a concatenation of text strings, each of which is a match for a successive RE in the search pattern.

* A single-character RE, followed by an asterisk (*) matches *zero* or more occurrences of the single-character RE. Such a pattern is called a *closure*. For example, **[a–z][a–z]*** matches any string of one or more lower case letters.

\{m\}
\{m,\}
\{m,n\} A one-character RE followed by \{*m*\}, \{*m*,\}, or \{*m*,*n*\} is an RE that matches a *range* of occurrences of the one-character RE. The values of *m* and *n* must be nonnegative integers less than 256; \{*m*\} matches *exactly m* occurrences; \{*m*,\} matches *at least m* occurrences; \{*m*,*n*\} matches *any number* of occurrences *between m* and *n*, inclusively. Whenever a choice exists, the RE matches as many occurrences as possible.

\(. . .\) An RE enclosed between the character sequences \(and \) matches whatever the unadorned RE matches, but saves the string matched by the enclosed RE in a numbered substring register. There can be up to nine such substrings in an RE, and parenthesis operators can be nested.

n Match the contents of the *n*th substring register from the current RE. This provides a mechanism for extracting matched substrings. For example, the expression **^\(..*\)\1$** matches a line consisting entirely of two adjacent non-null appearances of the same string. When nested parenthesized substrings are present, *n* is determined by counting occurrences of \(starting from the left.

// The null RE (//) is equivalent to the last RE encountered.

Commands

The commands **a** for *append*, **c** for *change*, and **i** for *insert*, allow you to add new text to the buffer. While accepting new text, **ed** is said to be in *input mode*. While in input mode, *no* commands are recognized; all character input is inserted into the buffer. To exit from input mode, enter a dot (**.**) on a line by itself; **ed** then reverts to command mode. Or, you can interrupt **ed** (typically with CTRL-C), in which case it displays

a ? and returns to command mode.

Commands may accept zero, one, or two addresses. Commands that accept no addresses regard the presence of an address as an error. Commands that accept one or two addresses assume default addresses when too few addresses are given; if more addresses are given than such a command requires, only the last ones are used.

In the following list of **ed** commands, the default addresses are shown in parentheses; the parenthesized addresses are *not* part of the command.

It is generally illegal for more than one command to appear on a line. However, any command (except **e**, **f**, **r**, or **w**) may be followed by **l**, **n**, or **p** in which case the current line is either listed, numbered or printed, respectively.

(.)**a**
text
.
Append text. Add lines of *text* into the buffer after the addressed line. The resulting current line is the last line of input, or the addressed line if no text is entered. Address **0** is legal for this command, in which case the *text* is placed at the beginning of the buffer. The maximum number of characters per input line (from a terminal) is 256, including the final NEWLINE.

(.)**c**
text
.
Change lines. Delete the addressed lines, and then accept lines of *text* to replace them. **c** accepts one or two addresses; the default is the current line. The resulting current line is the last line of input, or the line preceding the deleted lines if no text is entered.

(.,.)**d** Delete the addressed lines from the buffer. **d** accepts one or two addresses; the default is the current line. The resulting current line is the line following the last one deleted; if the deleted lines were at the end of the buffer, the new last line is the resulting current line.

e *filename*
Edit a file. Delete the entire contents of the buffer, and then read in the named file. The resulting current line is the last line of the buffer. **e** reports the number of characters read into the buffer, and sets *filename* to be the current file (for use as a default filename in subsequent commands). If no *filename* is given, the current filename, if any, is used (see the **f** command, below). If *filename* is replaced by a shell (**sh**(1)) command prefaced with a '**!**', the shell command is executed and its output is read into the buffer after the current line. Such a shell command is *not* used as the current filename. **e** displays a **?** if the buffer has not been written out since the last change made — another **e** command in response to the **?** forces the command to take effect.

E *filename*
The **E** command is like **e**, except that the editor does not check for changes to the buffer since the last **w** command was performed.

f *filename*
Display or set the current filename. If *filename* is given as an argument, the file (**f**) command changes the current filename to *filename*; otherwise, it prints the current filename.

(1,$) **g**/*RE*/*command-list*
The global (**g**) command performs *command-list* on all lines in the range of addresses that match *RE*. **ed** executes *command-list* for each matching line in succession, setting the current line to each in turn. *command-list* can contain a single command, or it can be continued across input lines, with one **ed** command per line, by escaping all but the last NEWLINE with a \ character. Operations that place **ed** into input mode (**a**, **i**, and **c**), *are* permitted in *command-list*; the final '**.**' terminating text input may be omitted if it is the last line of the *command-list*. **g**, **G**, **v**, and **V** commands, however, are *not* permitted. An empty *command-list* is equivalent to the **p** command.

(1,$) **G**/*RE*/

The interactive **G** (Global) command, selects all lines that match the given *RE*. Then, each selected line is made current, and any *one* command (other than one of the **a, c, i, g, G, v,** and **V** commands) can be performed upon that line. A NEWLINE acts as a null command; an **&** reexecutes the most recent command. Commands entered during execution of the **G** command can address and affect lines other than the current line. The **G** command can be terminated by an interrupt (typically CTRL-D).

h Help. Display a short error message that explains the reason for the most recent **?** diagnostic.

H Automatic printing of help diagnostics. Toggle between printing the **?** diagnostic, or automatically printing diagnostic messages as well.

(.)**i**

text

.

Insert Text. Insert the given *text* into the buffer, above the addressed line. **i** accepts one *address*; the default is the current line. The resulting current line is the last line of input; if no text is input, it is the line just before the addressed line. This command differs from the *a* command only in the placement of the input text; Address 0 is not allowed for this command. The maximum number of characters that may be entered from a terminal is 256 per line (including the NEWLINE character).

(.,.+1)**j** Join Lines. Remove the NEWLINE character from between the two addressed lines. The defaults are the current line and the line following. If exactly one address is given, this command does nothing. The joined line is the resulting current line.

(.)**k***c* Mark the addressed line with the name *c*, a lower-case letter. The address-form, *'c*, addresses the line marked by *c*. **k** accepts one *address*; the default is the current line. The current line is left unchanged.

(.,.)**l** List nonprinting characters. Print the addressed lines in an unambiguous way: a few nonprinting characters, such as TAB and BACKSPACE are represented by visually mnemonic overstrikes. All other nonprinting characters are shown in octal, with long lines folded. **l** accepts one or two addresses; the default is the current line. The resulting current line is the last line printed. An **l** command may be appended to any command other than **e, f, r,** or **w**.

(.,.) **m***address*

Move addressed lines to just after *address*. Address 0 is legal, and moves the addressed line(s) to the beginning of the file. An error results if *address* falls within the range of lines to move. **m** accepts two addresses to specify a range of lines to move; the default is the current line. The resulting current line is the last of the moved lines.

(.,.)**n** Number the displayed lines. Print the addressed lines, preceding each with its line number and a TAB character. **n** accepts one or two addresses; the default is the current line. The resulting current line is the last line printed. The **n** command can be appended to any command other than **e, f, r,** or **w**.

(.,.)**p** Print the addressed lines. **p** accepts one or two addresses; the default is the current line. The resulting current line is the last line printed. The **p** command may be appended to any command other than **e, f, r,** or **w**. For example, **dp** deletes the current line and prints the new current line.

P Toggle prompting on or off. When prompting is in effect, the editor prompts with a * for commands. A subsequent **P** command turns prompting off.

q Quit. Exit from **ed**. Note, however, that the buffer is *not* automatically written out; you must write any changes to be saved with the **w** command; **ed** warns you once if you have not saved your changes (unless the '–' option is in effect). A second **q** forces **ed** to exit regardless, destroying the buffer's contents.

Q Force quit. This is the same as **q**, but you do not get any warning if you have not previously written out the buffer. **ed** simply exits.

($) **r** *filename*

Read in the contents of *filename*, after the addressed line. If *filename* is not given, the current filename, if any, is used (see the **e** and **f** commands). The current filename is *not* altered; if there is no current filename, *filename* becomes the current filename. **r** accepts one *address*; the default is **$**. Address 0 is legal for **r**, in which case the file is read in at the beginning of the buffer. If the read is successful, the number of characters read is typed. The resulting current line is the last line read in from the file. If *filename* is replaced by a shell (**sh**(1)) command prefaced with a **!**, the shell command is executed and its output is read in. Such a shell command is *not* remembered as the current filename.

(.,.) **s**/*RE*/*rs*/
(.,.) **s**/*RE*/*rs*/**g**
(.,.) **s**/*RE*/*rs*/*n*

Substitute. Search each addressed line for the first occurrence of a string matching the specified *RE*, and replace it with *rs*, the replacement string. If **g** (global suffix) is appended to the command, replace *all* (non-overlapped) matching strings in each addressed line with the replacement string *rs*. Note: the **g** *suffix* is distinct from the **g** *command*. If a number *n* is appended, replace only the *n*'th occurrence of the matched string on each addressed line. **s** accepts one or two addresses; the default is the current line. The resulting current line is the last line on which a substitution is made. An error results if *RE* matches no strings in the addressed line or range. Any character (other than SPACE or NEWLINE can be used instead of / to delimit *RE* and *rs*. As with RE's in addresses, you can refer to the entire string matched by *RE* with an '**&**'; you can refer to parenthesized substrings within *RE* using **\1** ... *****n*. When **%** is the only character in *rs*, the *rs* from in the most recent substitute command is used as the current *rs*. The **%** loses its special meaning when it is in a replacement string of more than one character, or if it is preceded by a backslash.

A line may be split by substituting a NEWLINE character into it. The NEWLINE in the *replacement* must be escaped by preceding with an '\'. Such substitutions cannot be done as part of a **g** or **v** command list.

(.,.) **t***address*

Transfer. Transpose a copy of the addressed range of lines to just after the given *address*. **t** (transfer) is like **m** (move), except that it copies of the lines, rather than moving them. **t** accepts two addresses preceding the operation letter, the current address is the default. The resulting current line is the last line copied. Address 0 is allowed.

u Undo. Reverse the effect of the most recent command that modified the buffer. A second **u** undoes the undo operation.

(1,$) **v**/*RE*/*command-list*

This command is the same as the global command **g** except that the *command-list* is executed with '.' initially set to every line that does *not* match the RE.

(1,$) **V**/*RE*

Similar to the **G** command, except that the lines selected are those that do *not* match the *RE*.

(1,$) **w** *filename*

Write the addressed lines to *filename*. If the file does not exist, **ed** creates it. The current filename is *not* altered; if there is no current filename, then *filename* becomes current. If no *filename* is given, the current filename, if any, is used. **w** accepts one or two addresses; the default is all lines in the file. The current line is unchanged. If the command is successful, the number of characters written is displayed. If *filename* is replaced by a shell (**sh**(1)) command prefaced with a '**!**', the shell command is executed with standard input taken from the addressed lines. Such a shell command is *not* remembered as the current filename.

(1,$) **W** *filename*

Like **w**, but append the addressed lines onto the named file.

x Encrypt the file. **ed** prompts for an encryption key from the standard input. Subsequent **e**, **r**, and

w commands encrypt and decrypt the text with this key (see **crypt**(1)). An empty key turns off encryption. Encryption can also be specified on the command line with the **–x** option.

(**$**)**=** Display the line number of the addressed line; the current line remains unchanged.

!*shell-command*

Run a shell command. *shell-command* is a (Bourne shell) command line. **ed** replaces the unescaped character **%** with the current filename; if a **!** appears as the first character of the shell command, it is replaced with the text of the previous shell command. (**!!** repeats the last shell command.) If any such expansion is performed, the expanded line is echoed. The current line is unchanged.

address

NEWLINE

An address, alone on a line, prints the addressed line. A NEWLINE alone is equivalent to '**.+1p**', which is useful for stepping forward through the buffer.

If an interrupt signal (ASCII DEL or BREAK) is sent, **ed** prints a **?** and returns to *its* command level.

File Format Specification Support

ed supports the **fspec**(5) formatting capability for displaying lines. When the first line of a file is a format specification of the form:

<:*ts*[**,***ts*]... **s***max***:>**

where *ts* is the column number of a tab stop and *max* is the maximum line length for display purposes, and with the terminal in '**stty –tabs**' or '**stty tab3**' mode (see **stty**(1V) for details), the indicated tab stops are used in displayed lines. While inserting text, however, tab stops are set to every eighth column.

ENVIRONMENT

TMPDIR If this environment variable is set and is not null, its value is used in place of **/usr/tmp** as the directory in which the temporary file is placed.

FILES

/usr/tmp/e#	temporary; # is the process number
ed.hup	file for saved work if the terminal is hung up

SEE ALSO

crypt(1), **ex**(1), **grep**(1V), **sed**(1V), **sh**(1), **stty**(1V), **vi(1), regexp(3), fspec**(5)

Editing Text Files

LIMITATIONS

The following limitations apply:

512 characters per line.
256 characters per global command-list.
1024 characters per filename.
The limit on the number of lines depends on the amount of user memory:
each line takes 1 word.

When reading a file, **ed** discards ASCII NUL characters and all characters after the last NEWLINE. Files (such as executable images) that contain characters not in the ASCII set (bit 8 on) cannot be edited using **ed**.

If a file is not terminated by a NEWLINE character, **ed** adds one and prints a message saying that it has done so.

If the closing delimiter of an RE or of a replacement string (such as /) would be the last character before a NEWLINE, that delimiter can be omitted, in which case the addressed line is printed. The following pairs of commands are equivalent:

s/s1/s2	**s/s1/s2/p**
g/s1	**g/s1/p**
?s1	**?s1?**

DIAGNOSTICS

? For command errors.

?file*:*error

For an inaccessible file (use the **h** (help) and **H** (Help) commands for detailed explanations).

If changes have been made in the buffer since the last **w** command, **ed** issues a warning **?** when a command is given that would destroy the buffers contents. A second **e** or **q** command at this point will take effect. The '–' and –**s** command-line options inhibit this feature.

CAVEATS AND BUGS

A **!** command cannot be subject to a **g** or a **v** command.

The sequence **\n** in an RE does not match a NEWLINE character.

Files encrypted directly with the **crypt**(1) command with the null key cannot be edited.

The encryption facilities of **ed** are not available on software shipped outside the U.S.

Characters are masked to 7 bits on input.

If the editor input is coming from a command file, the editor exits at the first failure of a command in that file.

NAME

eject – eject floppy disk from an autoeject floppy drive

SYNOPSIS

eject [**–d** | **–f** | **–n** | *device* | *nickname*]

AVAILABILITY

Sun-3/80 systems only.

DESCRIPTION

eject is used for those removable media devices that do not have a manual eject button. The device may be specified by its name or by a nickname; if no device is specified the default device is used.

Only devices that support **eject** under program control respond to this command.

eject can also display its default device and a list of nicknames.

OPTIONS

–d Display the name of the default device to be ejected.

–f Force the device to eject even if it is busy.

–n Display the nickname to device name translation table.

device Specify which device to **eject**, by the name it appears in the directory **/dev**.

nickname
Specify which device to **eject**, by its nickname as known to this command.

FILES

/dev/rfd0a

SEE ALSO

fd(4S)

DIAGNOSTICS

A short help message is printed if an unknown flag is specified. A diagnostic is printed if the device name cannot be opened or does not support **eject**.

BUGS

There ought to be a way to change the default on a per-user basis.

NAME

env – obtain or alter environment variables for command execution

SYNOPSIS

env [–] [*name=value* ...] [*command*]

DESCRIPTION

env obtains the current **environment**, modifies it according to its arguments, and executes the command with the modified environment that results.

If no *command* is specified, the resulting environment is displayed.

OPTIONS

–	Ignore the environment that would otherwise be inherited from the current shell. Restricts the environment for *command* to that specified by the arguments.
name=value	Set the environment variable *filename* to *value* and add it to the environment.

SEE ALSO

sh(1), **execve**(2), **profil**(2), **environ**(5V)

NAME

expr – evaluate arguments as a logical, arithmetic, or string expression

SYNOPSIS

expr *argument...*

DESCRIPTION

expr evaluates expressions as specified by its arguments. After evaluation, the result is written on the standard output. Each token of the expression is a separate argument, so terms of the expression must be separated by blanks. Characters special to the shell must be escaped. Note: **0** is returned to indicate a zero value, rather than the null string. Strings containing blanks or other special characters should be quoted. Integer-valued arguments may be preceded by a unary minus sign. Internally, integers are treated as 32-bit, two's-complement numbers.

The operators and keywords are listed below. Characters that need to be escaped are preceded by '\'. The list is in order of increasing precedence, with equal precedence operators grouped within { } symbols.

expr \| *expr*
Return the first *expr* if it is neither NULL nor **0**, otherwise returns the second *expr*.

expr \& *expr*
Return the first *expr* if neither *expr* is NULL or **0**, otherwise returns **0**.

expr { =, \>, \>= , \<, \<=, != } *expr*
Return the result of an integer comparison if both arguments are integers, otherwise returns the result of a lexical comparison.

expr { +, − } *expr*
Addition or subtraction of integer-valued arguments.

expr { *, /, % } *expr*
Multiplication, division, or remainder of the integer-valued arguments.

string : *regular-expression*
match *string regular-expression*
The two forms of the matching operator above are synonymous. The matching operators : and **match** compare the first argument with the second argument which must be a regular expression. Regular expression syntax is the same as that of **ed**(1), except that all patterns are "anchored" (treated as if they begin with ^) and, therefore, ^ is not a special character, in that context. Normally, the matching operator returns the number of characters matched (**0** on failure). Alternatively, the \(...\) pattern symbols can be used to return a portion of the first argument.

substr *string integer-1 integer-2*
Extract the subtring of *string* starting at position *integer-1* and of length *integer-2* characters. If *integer-1* has a value greater than the length of *string*, **expr** returns a null string. If you try to extract more characters than there are in *string*, **expr** returns all the remaining characters from *string*. Beware of using negative values for either *integer-1* or *integer-2* as **expr** tends to run forever in these cases.

index *string character-list*
Report the first position in *string* at which any one of the characters in *character-list* matches a character in *string*.

length *string*
Return the length (that is, the number of characters) of *string*.

(**expr**) Parentheses may be used for grouping.

SYSTEM V DESCRIPTION

The operators **substr**, **index**, and **length** are not supported.

EXAMPLES

1. **a=‘expr $a + 1‘**

   Adds 1 to the shell variable **a**.

2. **# ’For $a equal to either "/usr/abc/file" or just "file"’**
   expr $a : ’.*/\(.*\)’ \| $a

   Returns the last segment of a path name (that is, the filename part). Watch out for / alone as an argument: *expr* will take it as the division operator (see BUGS below).

3. **# A better representation of example 2.**
   expr //$a : ’.*/\(.*\)’

   The addition of the // characters eliminates any ambiguity about the division operator and simplifies the whole expression.

4. **expr $VAR : ’.*’**

   Returns the number of characters in **$VAR**.

SEE ALSO

ed(1), **sh**(1), **test**(1V)

EXIT CODE

expr returns the following exit codes:

0	if the expression is neither null nor **0**
1	if the expression *is* null or **0**
2	for invalid expressions.

DIAGNOSTICS

syntax error	for operator/operand errors
non-numeric argument	if arithmetic is attempted on such a string
division by zero	if an attempt to divide by zero is made

BUGS

After argument processing by the shell, **expr** cannot tell the difference between an operator and an operand except by the value. If **$a** is an =, the command:

expr $a = ’=’

looks like:

expr = = =

as the arguments are passed to **expr** (and they will all be taken as the = operator). The following works:

expr X$a = X=

Note: the **match**, **substr**, **length**, and **index** operators cannot themselves be used as ordinary strings. That is, the expression:

example% expr index expurgatorious length
syntax error
example%

generates the ‘**syntax error**’ message as shown instead of the value **1** as you might expect.

NAME

fdformat – format diskettes for use under SunOS

SYNOPSIS — Sun386i SYSTEMS

fdformat [**–L**] [**–2**]
fdformat [**–eflLv**] [*device*]

AVAILABLITY

Sun386i and Sun-3/80 systems only.

DESCRIPTION

fdformat is a program for formatting diskettes to use with the SunOS operating system. All new blank diskettes must be formatted before use. **fdformat** formats and verifies each track on the diskette, and terminates if it finds any bad sectors. **fdformat** destroys all existing data on the diskette.

By default, **fdformat** formats a 1.44 megabyte high density diskette. Use the **–L** or **–l** option to format low density diskettes.

On Sun386i systems, use the **–2** option to format diskettes in the optional external 5 1/4" floppy drive.

On Sun-3/80 systems the default device is the internal floppy drive, **/dev/rfd0c**.

To format a diskette for use under MS-DOS, use the MS-DOS format command in a DOS window on the Sun386i system.

OPTIONS

–e Eject the diskette when done. (Sun-3/80 systems only.)

–f Force. Do not ask for confirmation before starting format. (Sun-3/80 systems only.)

–l Format a low density diskette (720 kilobyte) diskette. (Sun-3/80 systems only.)

–L Format a low density diskette (720 kilobyte) diskette.

–v Verify the floppy diskette after formatting. (Sun-3/80 systems only.)

–2 Format a diskette in the optional external 5 1/4" floppy drive. (Sun386i systems only.)

FILES — Sun386i SYSTEMS

/dev/rfd0c
/dev/rfdl0c
/dev/rfd2c
/dev/rfdl2c

SEE ALSO

dos(1), **fd**(4s)

BUGS

The SunOS system currently does not support bad sector mapping on diskettes. Therefore, a diskette is unusable if **fdformat** finds an error (bad sector).

NAME

file – determine the type of a file by examining its contents

SYNOPSIS

file [**–f** *ffile*] [**–cL**] [**–m** *mfile*] *filename*...

DESCRIPTION

file performs a series of tests on each *filename* in an attempt to determine what it contains. If the contents of a file appear to be ASCII text, **file** examines the first 512 bytes and tries to guess its language.

file uses the file **/etc/magic** to identify files that have some sort of *magic number*, that is, any file containing a numeric or string constant that indicates its type.

OPTIONS

–c Check for format errors in the magic number file. For reasons of efficiency, this validation is not normally carried out. No file type-checking is done under **–c**.

–f *ffile* Get a list of filenames to identify from *ffile*.

–L If a file is a symbolic link, test the file the link references rather than the link itself.

–m *mfile*
Use *mfile* as the name of an alternate magic number file.

EXAMPLE

This example illustrates the use of **file** on all the files in a specific user's directory:

```
example% pwd
/usr/blort/misc
example% file *
code:           mc68020 demand paged executable
code.c:         c program text
counts:         ascii text
doc:            roff, nroff, or eqn input text
empty.file:     empty
libz:           archive random library
memos:          directory
project:        symbolic link to /usr/project
script:         executable shell script
titles:         ascii text
s5.stuff:       cpio archive
example%
```

FILES

/etc/magic

SEE ALSO

magic(5)

BUGS

file often makes mistakes. In particular, it often suggests that command files are C programs.

Does not recognize Pascal or LISP.

NAME

make – maintain, update, and regenerate related programs and files

SYNOPSIS

make [**–f** *makefile*] ... [**–d**] [**–dd**] [**–D**] [**–DD**] [**–e**] [**–i**] [**–k**] [**–n**] [**–p**] [**–P**] [**–q**] [**–r**] [**–s**] [**–S**] [**–t**] [**–z**] [*target* ...] [*macro=value* ...]

DESCRIPTION

make executes a list of shell commands associated with each *target*, typically to create or update a file of the same name. *makefile* contains entries that describe how to bring a target up to date with respect to those on which it depends, which are called *dependencies*. Since each dependency is a target, it may have dependencies of its own. Targets, dependencies, and sub-dependencies comprise a tree structure that **make** traces when deciding whether or not to rebuild a *target*.

make recursively checks each *target* against its dependencies, beginning with the first target entry in *makefile* if no *target* argument is supplied on the command line. If, after processing all of its dependencies, a target file is found either to be missing, or to be older than any of its dependencies, **make** rebuilds it. Optionally with this version of **make**, a target can be treated as out-of-date when the commands used to generate it have changed since the last time the target was built.

To build a given target, **make** executes the list of commands, called a *rule*. This rule may be listed explicitly in the target's makefile entry, or it may be supplied implicitly by **make**.

When no *makefile* is specified with a **–f** option:

- If there is a file named **makefile** in the working directory, **make** uses that file. If, however, there is an SCCS history file (**SCCS/s.makefile**) which is newer, **make** attempts to retrieve and use the most recent version.
- In the absence of the above file(s), if a file named **Makefile** is present in the working directory, **make** attempts to use it. If there is an SCCS history file (**SCCS/s.Makefile**) that is newer, **make** attempts to retrieve and use the most recent version.

If no *target* is specified on the command line, **make** uses the first target defined in *makefile*.

If a *target* has no makefile entry, or if its entry has no rule, **make** attempts to derive a rule by each of the following methods, in turn, until a suitable rule is found. (Each method is described under USAGE below.)

- Pattern matching rules.
- Implicit rules, read in from a user-supplied makefile.
- Standard implicit rules (also known as suffix rules), typically read in from the file **/usr/include/make/default.mk**.
- SCCS retrieval. **make** retrieves the most recent version from the SCCS history file (if any). See the description of the **.SCCS_GET:** special-function target for details.
- The rule from the **.DEFAULT:** target entry, if there is such an entry in the makefile.

If there is no makefile entry for a *target*, if no rule can be derived for building it, and if no file by that name is present, **make** issues an error message and halts.

OPTIONS

–f *makefile*

Use the description file *makefile*. A '–' as the *makefile* argument denotes the standard input. The contents of *makefile*, when present, override the standard set of implicit rules and predefined macros. When more than one '**–f** *makefile*' argument pair appears, **make** uses the concatenation of those files, in order of appearance.

–d Display the reasons why **make** chooses to rebuild a target; **make** displays any and all dependencies that are newer. In addition, **make** displays options read in from the **MAKEFLAGS** environment variable.

–dd Display the dependency check and processing in vast detail.

–D Display the text of the makefiles read in.

–DD Display the text of the makefiles, **default.mk** file, the state file, and all hidden-dependency reports.

–e Environment variables override assignments within makefiles.

–i Ignore error codes returned by commands. Equivalent to the special-function target '**.IGNORE:**'.

–k When a nonzero error status is returned by a rule, or when **make** cannot find a rule, abandon work on the current target, but continue with other dependency branches that do not depend on it.

–n No execution mode. Print commands, but do not execute them. Even lines beginning with an **@** are printed. However, if a command line contains a reference to the **$(MAKE)** macro, that line is always executed (see the discussion of **MAKEFLAGS** in **Reading Makefiles and the Environment**).

–p Print out the complete set of macro definitions and target descriptions.

–P Merely report dependencies, rather than building them.

–q Question mode. **make** returns a zero or nonzero status code depending on whether or not the target file is up to date.

–r Do not read in the default makefile, **/usr/include/make/default.mk**.

–s Silent mode. Do not print command lines before executing them. Equivalent to the special-function target '**.SILENT:**'.

–S Undo the effect of the **–k** option. Stop processing when a non-zero exit status is returned by a command.

–t Touch the target files (bringing them up to date) rather than performing their rules. *This can be dangerous when files are maintained by more than one person.* When the **.KEEP_STATE:** target appears in the makefile, this option updates the state file just as if the rules had been performed.

–z Propagate **–d** and **–D** to nested (recursive) commands through the **MAKEFLAGS** macro.

macro=value

Macro definition. This definition overrides any regular definition for the specified macro within the makefile itself, or in the environment. However, this definition can still be overridden by conditional macro assignments.

USAGE

Refer to *Doing More with SunOS: Beginner's Guide* and **make** in *Programming Utilities and Libraries* for tutorial information about **make.**

Reading Makefiles and the Environment

When **make** first starts, it reads the **MAKEFLAGS** environment variable to obtain any the following options specified present in its value: **–e, –i, –k, –l, –n, –p, –q, –r, –s, –S, –t,** or **–z.** (Within the **MAKEFLAGS** value, the leading '**—**' character for the option string is omitted.) **make** then reads the command line for additional options, which also take effect.

Next, **make** reads in a default makefile that typically contains predefined macro definitions, target entries for implicit rules, and additional rules, such as the rule for retrieving SCCS files. If present, **make** uses the file **default.mk** in the current directory; otherwise it reads the file **/usr/include/make/default.mk**, which contains the standard definitions and rules.
Use the directive:

include /usr/include/make/default.mk

in your local **default.mk** file to include them.

Next, **make** imports variables from the environment (unless the **–e** option is in effect), and treats them as defined macros. Because **make** uses the most recent definition it encounters, a macro definition in the makefile normally overrides an environment variable of the same name. When **–e** is in effect, however, environment variables are read in *after* all makefiles have been read. In that case, the environment variables take precedence over definitions in the makefile.

Next, **make** reads the state file, **.make.state** in the local directory if it exists, and then any makefiles you specify with **–f**, or one of **makefile** or **Makefile** as described above.

Next, (after reading the environment if **–e** is in effect), **make** reads in any macro definitions supplied as command line arguments. These override macro definitions in the makefile and the environment both, but only for the **make** command itself.

make exports environment variables, using the most recently defined value. Macro definitions supplied on the command line are not normally exported, unless the macro is also an environment variable.

make does not export macros defined in the makefile. If an environment variable is set, and a macro with the same name is defined on the command line, **make** exports its value as defined on the command line. Unless **–e** is in effect, macro definitions within the makefile take precedence over those imported from the environment.

The macros **MAKEFLAGS, MAKE, KEEP_STATE, SHELL, HOST_ARCH, TARGET_ARCH, HOST_MACH,** and **TARGET_MACH** are special cases. See **Special-Purpose Macros** below, for details.

Makefile Target Entries

A target entry has the following format:

target... [: | ::] [*dependency*] ... [; *command*] ...
[*command*]
...

The first line contains the name of a target, or a space-separated list of target names, terminated with a colon or double colon. If a list of targets is given, this is equivalent to having a separate entry of the same form for each target. The colon(s) may be followed by a *dependency*, or a dependency list. **make** checks this list before building the target. The dependency list may be terminated with a semicolon (;), which in turn can be followed by a single Bourne shell command. Subsequent lines in the target entry begin with a TAB, and contain Bourne shell commands. These commands comprise the rule for building the target.

Shell commands may be continued across input lines by escaping the NEWLINE with a backslash (\). The continuing line must also start with a TAB.

To rebuild a target, **make** expands macros, strips off initial TAB characters and either executes the command directly (if it contains no shell metacharacters), or passes each command line to a Bourne shell for execution.

The first line that does not begin with a TAB or # begins another target or macro definition.

Makefile Special Characters

Global

Start a comment. The comment ends at the next NEWLINE. If the # follows the TAB in a command line, that line is passed to the shell (which also treats # as the start of a comment).

include *filename*

If the word **include** appears as the first seven letters of a line and is followed by a SPACE or TAB, the string that follows is taken as a filename to interpolate at that line. **include** files can be nested to a depth of no more than about 16. If *filename* is a macro reference, it is expanded.

Targets and Dependencies

: Target list terminator. Words following the colon are added to the dependency list for the target or targets. If a target is named in more than one colon-terminated target entry, the dependencies for all its entries are added to form that target's complete dependency list.

:: Target terminator for alternate dependencies. When used in place of a ':' the double-colon allows a target to be checked and updated with respect to alternate dependency lists. When the target is out-of-date with respect to dependencies listed in the first alternate, it is built according to the rule for that entry. When out-of-date with respect to dependencies in another alternate, it is built according the rule in that other entry. Implicit rules do not apply to double-colon targets; you must supply a rule for each entry. If no dependencies are specified, the rule is always performed.

target [+ *target*...] :
Target group. The rule in the target entry builds all the indicated targets as a group. It is normally performed only once per **make** run, but is checked for command dependencies every time a target in the group is encountered in the dependency scan.

% Pattern matching wild card metacharacter. Like the * shell wild card, % matches any string of zero or more characters in a target name or dependency, in the target portion of a conditional macro definition, or within a pattern replacement macro reference. Note: only one % can appear in a target, dependency-name, or pattern-replacement macro reference.

Macros

= Macro definition. The word to the left of this character is the macro name; words to the right comprise its value. Leading and trailing white space characters are stripped from the value. A word break following the = is implied.

$ Macro reference. The following character, or the parenthesized or bracketed string, is interpreted as a macro reference: **make** expands the reference (including the $) by replacing it with the macro's value.

( )
{ } Macro-reference name delimiters. A parenthesized or bracketed word appended to a $ is taken as the name of the macro being referred to. Without the delimiters, **make** recognizes only the first character as the macro name.

$$ A reference to the dollar-sign macro, the value of which is the character '$'. Used to pass variable expressions beginning with $ to the shell, to refer to environment variables which are expanded by the shell, or to delay processing of dynamic macros within the dependency list of a target, until that target is actually processed.

\$ Escaped dollar-sign character. Interpreted as a literal dollar sign within a rule.

+= When used in place of '=', appends a string to a macro definition (must be surrounded by white space, unlike '=').

:= Conditional macro assignment. When preceded by a list of targets with explicit target entries, the macro definition that follows takes effect when processing only those targets, and their dependencies.

Rules

– **make** ignores any nonzero error code returned by a command line for which the first non-TAB character is a '–'. This character is not passed to the shell as part of the command line. **make** normally terminates when a command returns nonzero status, unless the **–i** or **–k** options, or the **.IGNORE:** special-function target is in effect.

@ If the first non-TAB character is a **@**, **make** does not print the command line before executing it. This character is not passed to the shell.

? Escape command-dependency checking. Command lines starting with this character are not subject to command dependency checking.

! Force command-dependency checking. Command-dependency checking is applied to command lines for which it would otherwise be suppressed. This checking is normally suppressed for lines that contain references to the '?' dynamic macro (for example, '$?').

When any combination of '–', '@', '?', or '!' appear as the first characters after the TAB, all that are present apply. None are passed to the shell.

Special-Function Targets

When incorporated in a makefile, the following target names perform special-functions:

.DEFAULT:
If it has an entry in the makefile, the rule for this target is used to process a target when there is no other entry for it, no rule for building it, and no SCCS history file from which to retrieve a current version. **make** ignores any dependencies for this target.

.DONE: If defined in the makefile, **make** processes this target and its dependencies after all other targets are built. This target is also performed when **make** halts with an error, unless the **.FAILED** target is defined.

.FAILED:
This target, along with its dependencies, is performed instead of **.DONE** when defined in the makefile and **make** halts with an error.

.IGNORE:
Ignore errors. When this target appears in the makefile, **make** ignores non-zero error codes returned from commands.

.INIT: If defined in the makefile, this target and its dependencies are built before any other targets are processed.

.KEEP_STATE:
If this target appears in the makefile, **make** updates the state file, **.make.state**, in the current directory. This target also activates command dependencies, and hidden dependency checks.

.MAKE_VERSION:
A target-entry of the form:

.MAKE_VERSION: VERSION–*number*

enables version checking. If the version of **make** differs from the version indicated, **make** issues a warning message.

.PRECIOUS:
List of files not to delete. **make** does not remove any of the files listed as dependencies for this target when interrupted. **make** normally removes the current target when it receives an interrupt.

.SCCS_GET:
This target contains the rule for retrieving the current version of an SCCS file from its history file. To suppress automatic retrieval, add an entry for this target with an empty rule to your makefile.

.SILENT:
Run silently. When this target appears in the makefile, **make** does not echo commands before executing them.

.SUFFIXES:
The suffixes list for selecting implicit rules (see **The Suffixes List**).

Clearing Special Targets

In this version of **make**, you can clear the definition of the following special targets by supplying entries for them with no dependencies and no rule:

.DEFAULT, .DONE, .FAILED, .INIT, .SCCS_GET, and .SUFFIXES

Command Dependencies

When the **.KEEP_STATE:** target appears in the makefile, **make** checks the command for building a target against the state file, **.make.state**. If the command has changed since the last **make** run, **make** rebuilds the target.

Hidden Dependencies

When the **.KEEP_STATE:** target appears in the makefile, **make** reads reports from **cpp**(1) and other compilation processors for any "hidden" files, such as **#include** files. If the target is out of date with respect to any of these files, **make** rebuilds it.

Macros

Entries of the form

macro=value

define macros. *macro* is the name of the macro, and *value*, which consists of all characters up to a comment character or unescaped NEWLINE, is the value. **make** strips both leading and trailing white space in accepting the value.

Subsequent references to the macro, of the forms: $(*name*) or ${*name*} are replaced by *value*. The parentheses or brackets can be omitted in a reference to a macro with a single-character name.

Macro references can contain references to other macros, in which case nested references are expanded first.

Suffix Replacement Macro References

Substitutions within macros can be made as follows:

$(*name*:*string1*=*string2*)

where *string1* is either a suffix, or a word to be replaced in the macro definition, and *string2* is the replacement suffix or word. Words in a macro value are separated by SPACE, TAB, and escaped NEWLINE characters.

Pattern Replacement Macro References

Pattern matching replacements can also be applied to macros, with a reference of the form:

$(*name*: *op*%*os*= *np*%*ns*)

where *op* is the existing (old) prefix and *os* is the existing (old) suffix, *np* and *ns* are the new prefix and new suffix, respectively, and the pattern matched by % (a string of zero or more characters), is carried forward from the value being replaced. For example:

PROGRAM=fabricate
DEBUG= $(PROGRAM:%=tmp/%-g)

sets the value of **DEBUG** to **tmp/fabricate-g**.

Note: pattern replacement macro references cannot be used in the dependency line of a pattern matching rule; the % characters are not evaluated independently.

Appending to a Macro

Words can be appended to macro values as follows:

macro += *word* ...

Special-Purpose Macros

When the **MAKEFLAGS** variable is present in the environment, **make** takes options from it, in combination with options entered on the command line. **make** retains this combined value as the **MAKEFLAGS** macro, and exports it automatically to each command or shell it invokes.

Note: flags passed by way of **MAKEFLAGS** are only displayed when the **–d**, or **–dd** options are in effect.

The **MAKE** macro is another special case. It has the value **make** by default, and temporarily overrides the **–n** option for any line in which it is referred to. This allows nested invocations of **make** written as:

$(MAKE) ...

to run recursively, with the **–n** flag in effect for all commands but **make.** This lets you use '**make –n**' to test an entire hierarchy of makefiles.

For compatibility with the 4.2 BSD **make**, the **MFLAGS** macro is set from the **MAKEFLAGS** variable by prepending a '–'. **MFLAGS** is not exported automatically.

The **SHELL** macro, when set to a single-word value such as **/usr/bin/csh**, indicates the name of an alternate shell to use. The default is **/bin/sh**. Note: **make** executes commands that contain no shell metacharacters itself. Built-in commands, such as **dirs** in the C shell, are not recognized unless the command line includes a metacharacter (for instance, a semicolon). This macro is neither imported from, nor exported to the environment, regardless of **–e**. To be sure it is set properly, you must define this macro within every makefile that requires it.

The **KEEP_STATE** environment variable has the same effect as the **.KEEP_STATE:** special-function target. It enables command dependencies, hidden dependencies and writing of the state file.

The following macros are provided for use with cross-compilation:

HOST_ARCH
: The machine architecture of the host system. By default, this is the output of the **arch**(1) command prepended with '—'. Under normal circumstances, this value should never be altered by the user.

TARGET_ARCH
: The machine architecture of the target system. By default, the output of **arch**, prepended with '—'.

HOST_MACH
: The machine architecture of the host system. By default, this is the output of the **mach**(1), prepended with '—'. Under normal circumstances, this value should never be altered by the user.

TARGET_MACH
: The machine architecture of the target system. By default, the output of **mach**, prepended with '—'.

Dynamic Macros

There are several dynamically maintained macros that are useful as abbreviations within rules. They are shown here as references; if you were to define them, **make** would simply override the definition.

$* The basename of the current target, derived as if selected for use with an implicit rule. In the case of pattern matching rules, the value is the string matched by the '**%**'.

$< The name of a dependency file, derived as if selected for use with an implicit rule.

$@ The name of the current target.

$? The list of dependencies that are newer than the target. Command-dependency checking is automatically suppressed for lines that contain this macro, just as if the command had been prefixed with a '**?**'. See the description of '**?**', under **Makefile Special Tokens**, above. You can force this check with the **!** command-line prefix.

$% The name of the library member being processed. (See **Library Maintenance**, below.)

To refer to a dynamic macro within a dependency list, precede the reference with an additional '$' character (for example, '$$@'). Because **make** assigns $< and $* as it would for implicit rules (according to the suffixes list and the directory contents), they may be unreliable when used within explicit target entries.

These macros can be modified to apply either to the filename part, or the directory part of the strings they stand for, by adding an upper case **F** or **D**, respectively (and enclosing the resulting name in parentheses or braces). Thus, '$(@D)' refers to the directory part of the string '$@'; if there is no directory part, '.' is assigned. $(@F) refers to the filename part.

Conditional Macro Definitions

A macro definition of the form:

target-list := *macro* = *value*

indicates that when processing any of the targets listed *and their dependencies* , *macro* is to be set to the *value* supplied. Note that if a conditional macro is referred to in a dependency list, the **$** must be delayed (use $$ instead). Also, *target-list* may contain a **%** pattern, in which case the macro will be conditionally defined for all targets encountered that match the pattern. A pattern replacement reference can be used within the *value*.

You can temporarily append to a macro's value with a condtional definition of the form:

target-list := *macro* += *value*

Predefined Macros

make supplies the macros shown in the table that follows for compilers and their options, host architectures, and other commands. Unless these macros are read in as environment variables, their values are not exported by **make**. If you run **make** with any of these set in the environment, it is a good idea to add commentary to the makefile to indicate what value each is expected to take. If **–r** is in effect, **make** does not supply these macro definitions.

Table of Predefined Macros		
Use	*Macro*	*Default Value*
Library Archives	AR	ar
	ARFLAGS	rv
Assembler Commands	AS	as
	ASFLAGS	
	COMPILE.s	$(AS) $(ASFLAGS) $(TARGET_MACH)
	COMPILE.S	$(CC) $(ASFLAGS) $(CPPFLAGS) $(TARGET_MACH) -c
C Compiler Commands	CC	cc
	CFLAGS	
	CPPFLAGS	
	COMPILE.c	$(CC) $(CFLAGS) $(CPPFLAGS) $(TARGET_ARCH) -c
	LINK.c	$(CC) $(CFLAGS) $(CPPFLAGS) $(LDFLAGS) $(TARGET_ARCH)
FORTRAN 77 Compiler Commands	FC	f77
	FFLAGS	
	COMPILE.f	$(FC) $(FFLAGS) $(TARGET_ARCH) -c
	LINK.f	$(FC) $(FFLAGS) $(TARGET_ARCH) $(LDFLAGS)
	COMPILE.F	$(FC) $(FFLAGS) $(CPPFLAGS) $(TARGET_ARCH) -c
	LINK.F	$(FC) $(FFLAGS) $(CPPFLAGS) $(LDFLAGS) $(TARGET_ARCH)
Link Editor Command	LD	ld
	LDFLAGS	
lex Command	LEX	lex
	LFLAGS	
	LEX.l	$(LEX) $(LFLAGS) -t
lint Command	LINT	lint
	LINTFLAGS	
	LINT.c	$(LINT) $(LINTFLAGS) $(CPPFLAGS) $(TARGET_ARCH)
Modula 2 Commands	M2C	m2c
	M2FLAGS	
	MODFLAGS	
	DEFFLAGS	
	COMPILE.def	$(M2C) $(M2FLAGS) $(DEFFLAGS) $(TARGET_ARCH)
	COMPILE.mod	$(M2C) $(M2FLAGS) $(MODFLAGS) $(TARGET_ARCH)
Pascal Compiler Commands	PC	pc
	PFLAGS	
	COMPILE.p	$(PC) $(PFLAGS) $(CPPFLAGS) $(TARGET_ARCH) -c
	LINK.p	$(PC) $(PFLAGS) $(CPPFLAGS) $(LDFLAGS) $(TARGET_ARCH)
Ratfor Compilation Commands	RFLAGS	
	COMPILE.r	$(FC) $(FFLAGS) $(RFLAGS) $(TARGET_ARCH) -c
	LINK.r	$(FC) $(FFLAGS) $(RFLAGS) $(TARGET_ARCH) $(LDFLAGS)
rm Command	RM	rm -f
sccs Command	SCCSFLAGS	
	SCCSGETFLAGS	-s
yacc Command	YACC	yacc
	YFLAGS	
	YACC.y	$(YACC) $(YFLAGS)
Suffixes List	SUFFIXES	.o .c .c~ .s .s~ .S .S~ .ln .f .f~ .F .F~ .l .l~ .mod .mod~ .sym .def .def~ .p .p~ .r .r~ .y .y~ .h .h~ .sh .sh~ .cps .cps~

Implicit Rules

When a target has no entry in the makefile, **make** attempts to determine its class (if any) and apply the rule for that class. An implicit rule describes how to build any target of a given class, from an associated dependency file. The class of a target can be determined either by a pattern, or by a suffix; the corresponding dependency file (with the same basename) from which such a target might be built. In addition to a predefined set of implicit rules, make allows you to define your own, either by pattern, or by suffix.

Pattern Matching Rules

A target entry of the form:

tp%ts: *dp%ds*
rule

is a pattern matching rule, in which *tp* is a target prefix, *ts* is a target suffix, *dp* is a dependency prefix, and *ds* is a dependency suffix (any of which may be null). The **%** stands for a basename of zero or more characters that is matched in the target, and is used to construct the name of a dependency. When **make** encounters a match in its search for an implicit rule, it uses the rule in that target entry to build the target from the dependency file. Pattern-matching implicit rules typically make use of the **$@** and **$<** dynamic macros as placeholders for the target and dependency names. The dynamic macro **$*** is set to the string matched by the **%** wild card. Other, regular dependencies may occur in the dependency list. An entry of the form:

tp%ts: [*dependency*...] *dp%ds* [*dependency*...]
rule

is a valid pattern matching rule.

Suffix Rules

When no pattern matching rule applies, **make** checks the target name to see if it ends with a suffix in the known suffixes list. If so, **make** checks for any suffix rules, as well as a dependency file with same root and another recognized suffix, from which to build it.

The target entry for a suffix rule takes the form:

DsTs:
rule

where *Ts* is the suffix of the target, *Ds* is the suffix of the dependency file, and *rule* is the rule for building a target in the class. Both *Ds* and *Ts* must appear in the suffixes list. (A suffix need not begin with a '.' to be recognized.)

A suffix rule with only one suffix describes how to build a target having a null (or no) suffix from a dependency file with the indicated suffix. For instance, the **.c** rule could be used to build an executable program named **file** from a C source file named '**file.c**'.

Table of Standard Implicit (Suffix) Rules		
Use	*Implicit Rule Name*	*Command Line*
Assembly Files	.s.o	$(COMPILE.s) -o $@ $<
	.s.a	$(COMPILE.s) -o $% $< $(AR) $(ARFLAGS) $@ $% $(RM) $%
	.S.o	$(COMPILE.S) -o $@ $<
	.S.a	$(COMPILE.S) -o $% $< $(AR) $(ARFLAGS) $@ $% $(RM) $%
C Files	.c	$(LINK.c) -o $@ $< $(LDLIBS)
	.c.ln	$(LINT.c) $(OUTPUT_OPTION) -i $<
	.c.o	$(COMPILE.c) $(OUTPUT_OPTION) $<
	.c.a	$(COMPILE.c) -o $% $< $(AR) $(ARFLAGS) $@ $% $(RM) $%
FORTRAN 77 Files	.f	$(LINK.f) -o $@ $< $(LDLIBS)
	.f.o	$(COMPILE.f) $(OUTPUT_OPTION) $<
	.f.a	$(COMPILE.f) -o $% $< $(AR) $(ARFLAGS) $@ $% $(RM) $%
	.F	$(LINK.F) -o $@ $< $(LDLIBS)
	.F.o	$(COMPILE.F) $(OUTPUT_OPTION) $<
	.F.a	$(COMPILE.F) -o $% $< $(AR) $(ARFLAGS) $@ $% $(RM) $%
lex Files	.l	$(RM) $*.c $(LEX.l) $< > $*.c $(LINK.c) -o $@ $*.c $(LDLIBS) $(RM) $*.c
	.l.c	$(RM) $@ $(LEX.l) $< > $@
	.l.ln	$(RM) $*.c $(LEX.l) $< > $*.c $(LINT.c) -o $@ -i $*.c $(RM) $*.c
	.l.o	$(RM) $*.c $(LEX.l) $< > $*.c $(COMPILE.c) -o $@ $*.c $(RM) $*.c
Modula 2 Files	.mod .mod.o .def.sym	$(COMPILE.mod) -o $@ -e $@ $< $(COMPILE.mod) -o $@ $< $(COMPILE.def) -o $@ $<
NeWS	.cps.h	cps $*.cps
Pascal Files	.p	$(LINK.p) -o $@ $< $(LDLIBS)
	.p.o	$(COMPILE.p) $(OUTPUT_OPTION) $<
Ratfor Files	.r	$(LINK.r) -o $@ $< $(LDLIBS)
	.r.o	$(COMPILE.r) $(OUTPUT_OPTION) $<
	.r.a	$(COMPILE.r) -o $% $< $(AR) $(ARFLAGS) $@ $% $(RM) $%

Table of Standard Implicit (Suffix) Rules (continued)		
Use	*Implicit Rule Name*	*Command Line*
SCCS Files	**.SCCS_GET**	**sccs $(SCCSFLAGS) get $(SCCSGETFLAGS) $@ –G$@**
Shell Scripts	**.sh**	**cat $< >$@** **chmod +x $@**
yacc Files	**.y**	**$(YACC.y) $<** **$(LINK.c) –o $@ y.tab.c $(LDLIBS)** **$(RM) y.tab.c**
	.y.c	**$(YACC.y) $<** **mv y.tab.c $@**
	.y.ln	**$(YACC.y) $<** **$(LINT.c) –o $@ –i y.tab.c** **$(RM) y.tab.c**
	.y.o	**$(YACC.y) $<** **$(COMPILE.c) –o $@ y.tab.c** **$(RM) y.tab.c**

make reads in the standard set of implicit rules from the file **/usr/include/make/default.mk**, unless **–r** is in effect, or there is a **default.mk** file in the local directory that does not **include** that file.

The Suffixes List

The suffixes list is given as the list of dependencies for the '**.SUFFIXES:**' special-function target. The default list is contained in the **SUFFIXES** macro (See *Table of Predefined Macros* for the standard list of suffixes). You can define additional **.SUFFIXES:** targets; a **.SUFFIXES** target with no dependencies clears the list of suffixes. Order is significant within the list; **make** selects a rule that corresponds to the target's suffix and the first dependency-file suffix found in the list. To place suffixes at the head of the list, clear the list and replace it with the new suffixes, followed by the default list:

.SUFFIXES:
.SUFFIXES: *suffixes* **$(SUFFIXES)**

A tilde (~) indicates that if a dependency file with the indicated suffix (minus the ~) is under SCCS its most recent version should be retrieved, if necessary, before the target is processed.

Library Maintenance

A target name of the form:

lib(*member* ...)

refers to a member, or a space-separated list of members, in an **ar**(1V) library.

The dependency of the library member on the corresponding file must be given as an explicit entry in the makefile. This can be handled by a pattern matching rule of the form:

lib(*%.s*): *%.s*

where *.s* is the suffix of the member; this suffix is typically **.o** for object libraries.

A target name of the form

lib((*symbol*))

refers to the member of a randomized object library (see **ranlib**(1)) that defines the entry point named *symbol*.

Command Execution

Command lines are executed one at a time, *each by its own process or shell*. Shell commands, notably **cd**, are ineffectual across an unescaped NEWLINE in the makefile. A line is printed (after macro expansion) just before being executed. This is suppressed if it starts with a '**@**', if there is a '**.SILENT:**' entry in the makefile, or if **make** is run with the **–s** option. Although the **–n** option specifies printing without execution, lines containing the macro **$(MAKE)** are executed regardless, and lines containing the **@** special

character are printed. The **-t** (touch) option updates the modification date of a file without executing any rules. This can be dangerous when sources are maintained by more than one person.

Bourne Shell Constructs

To use the Bourne shell **if** control structure for branching, use a command line of the form:

```
if expression ; \
then command ; \
        ... ; \
else ; \
        ... ; \
fi
```

Although composed of several input lines, the escaped NEWLINE characters insure that **make** treats them all as one (shell) command line.

To use the Bourne shell **for** control structure for loops, use a command line of the form:

```
for var in list ; \
        do command; \
        ... ;\
done
```

To refer to a shell variable, use an escaped dollar-sign (\$). This prevents expansion of the dollar-sign by **make**.

To incorporate the standard output of a shell command in a macro, use a definition of the form:

MACRO **:sh** =*command*

The command is executed only once, standard error output is discarded, and NEWLINE characters are replaced with SPACEs. If the command has a non-zero exit status, **make** halts with an error.

To capture the output of a shell command in a macro reference, use a reference of the form:

$(*MACRO* **:sh)**

where *MACRO* is the name of a macro containing a valid Bourne shell command line. In this case, the command is executed whenever the reference is evaluated. As with shell command substitutions, the reference is replaced with the standard output of the command. If the command has a non-zero exit status, **make** halts with an error.

Signals

INT and QUIT signals received from the keyboard halt **make** and remove the target file being processed unless that target is in the dependency list for '**.PRECIOUS:**'.

EXAMPLES

This makefile says that **pgm** depends on two files **a.o** and **b.o**, and that they in turn depend on their corresponding source files (**a.c** and **b.c**) along with a common file **incl.h**:

```
pgm: a.o b.o
        $(LINK.c) -o $@ a.o b.o
a.o: incl.h a.c
        cc -c a.c
b.o: incl.h b.c
        cc -c b.c
```

The following makefile uses implicit rules to express the same dependencies:

```
pgm: a.o b.o
        cc a.o b.o -o pgm
a.o b.o: incl.h
```

FILES

makefile	
Makefile	current version(s) of **make** description file
SCCS/s.makefile	
SCCS/s.Makefile	SCCS history files for the above makefile(s)
default.mk	default file for user-defined targets, macros, and implicit rules
/usr/include/make/default.mk	makefile for standard implicit rules and macros (not read if **default.mk** is)
.make.state	state file in the local directory

SEE ALSO

ar(1V), **cc**(1V), **cd**(1), **get**(1), **lex**(1), **ranlib**(1), **passwd**(5),
Doing More with SunOS: Beginner's Guide
Programming Utilities and Libraries

DIAGNOSTICS

make returns an exit status of 1 when it halts as a result of an error. Otherwise it returns an exit status of **0**.

Do not know how to make *target*. **Stop.**
There is no makefile entry for *target*, and none of **make**'s implicit rules apply (there is no dependency file with a suffix in the suffixes list, or the target's suffix is not in the list).

*** *target* **removed.**
make was interrupted while building *target*. Rather than leaving a partially-completed version that is newer than its dependencies, **make** removes the file named *target*.

*** *target* **not removed.**
make was interrupted while building *target* and *target* was not present in the directory.

*** *target* **could not be removed,** *reason*
make was interrupted while building *target*, which was not removed for the indicated reason.

Read of include file '*file***' failed**
The makefile indicated in an **include** directive was not found, or was inaccessible.

Loop detected when expanding macro value '*macro***'**
A reference to the macro being defined was found in the definition.

Could not write state file '*file***'**
You used the .KEEP_STATE: target, but do not have write permission on the state file.

*** **Error code** *n*
The previous shell command returned a nonzero error code.

*** *signal message*
The previous shell command was aborted due to a signal. If '– **core dumped**' appears after the message, a **core** file was created.

Conditional macro conflict encountered
Displayed only when **–d** is in effect, this message indicates that two or more parallel targets currently being processed depend on a target which is built differently for each by virtue of conditional macros. Since the target cannot simultaneously satisfy both dependency relationships, it is conflicted.

BUGS

Some commands return nonzero status inappropriately; to overcome this difficulty, prefix the offending command line in the rule with a '–'.
Filenames with the characters '=', ':', or '@', do not work.
You cannot build **file.o** from **lib(file.o)**.
Options supplied by MAKEFLAGS should be reported for nested **make** commands. Use the **–d** option to find out what options the nested command picks up from MAKEFLAGS.
This version of **make** is incompatible in certain respects with previous versions:

- The **–d** option output is much briefer in this version. **–dd** now produces the equivalent voluminous output.
- **make** attempts to derive values for the dynamic macros '**$***', '**$<**', and '**$?**', while processing explicit targets. It uses the same method as for implicit rules; in some cases this can lead

either to unexpected values, or to an empty value being assigned. (Actually, this was true for earlier versions as well, even though the documentation stated otherwise.)

- **make** no longer searches the current directory for SCCS history files.
- Suffix replacement in macro references are now applied after the macro is expanded.

There is no guarantee that makefiles created for this version of **make** will work with earlier versions.

If there is no **default.mk** file in the current directory, and the file **/usr/include/make/default.mk** is missing, **make** stops before processing any targets. To force **make** to run anyway, create an empty **default.mk** file in the current directory.

Once a dependency is made, **make** assumes the dependency file is present for the remainder of the run. If a rule subsequently removes that file and future targets depend on its existence, unexpected errors may result.

When hidden dependency checking is in effect, the **$?** macro's value includes the names of hidden dependencies. This can lead to improper filename arguments to compiler commands when **$?** is used in a rule.

Pattern replacement macro references cannot be used in the dependency line of a pattern matching rule.

Unlike previous versions, this version of **make** strips a leading '**./**' from the value of the '**$@**' dynamic macro.

With automatic SCCS retrieval, this version of **make** does not support tilde suffix rules.

NAME

mt – magnetic tape control

SYNOPSIS

mt [**–f** *tapename*] *command* [*count*]

DESCRIPTION

mt sends commands to a magnetic tape drive. If *tapename* is not specified, the environment variable **TAPE** is used. If **TAPE** does not exist, **mt** uses the device **/dev/rmt12**. *tapename* must refer to a raw (not block) tape device. By default, **mt** performs the requested operation once; multiple operations may be performed by specifying *count*.

The available commands are listed below. Only as many characters as are required to uniquely identify a command need be specified.

mt returns a 0 exit status when the operation(s) were successful, 1 if the command was unrecognized or if **mt** was unable to open the specified tape drive, and 2 if an operation failed.

OPTIONS

eof, weof Write *count* EOF marks at the current position on the tape.

fsf Forward space *count* files. The tape is positioned on the first block of the file.

fsr Forward space *count* records.

bsf Back space *count* files. The tape is positioned on the first block of the file.

bsr Back space *count* records.

bsfm Back space *count* file marks. The tape is positioned on the beginning–of–tape side of the file mark.

asf Absolute space to *count* file number. This is equivalent to a **rewind** followed by a **fsf** *count*.

For the following commands, *count* is ignored:

eom Space to the end of recorded media on the tape (SCSI only). This is useful for appending files onto previously written tapes.

rewind Rewind the tape.

offline, rewoffl Rewind the tape and, if appropriate, take the drive unit offline by unloading the tape.

status Print status information about the tape unit.

retension Rewind the tape completely, then wind it forward to the end of the reel and back to beginning–of–tape to smooth out tape tension.

erase Erase the entire tape.

FILES

/dev/rmt*	raw magnetic tape interface
/dev/rar*	raw Archive cartridge tape interface
/dev/rst*	raw SCSI tape interface
/dev/rmt*	raw Xylogics tape interface

SEE ALSO

ar(4S), **mtio**(4), **st**(4S), **tm**(4S), **xt**(4S) **environ**(5V)

BUGS

Not all devices support all options, in particular **retension** and **bsfm**. For example, **ar**(4S) currently does not support the **fsr**, **bsf**, or **bsr** options. The half-inch tape driver, **/dev/rmt***, does not support the **retension** option.

NAME

mv – move or rename files

SYNOPSIS

mv [–] [–**fi**] *filename1 filename2*
mv [–] [–**fi**] *directory1 directory2*
mv [–] [–**fi**] *filename ... directory*

DESCRIPTION

mv moves files and directories around in the file system. A side effect of **mv** is to rename a file or directory. The three major forms of **mv** are shown in the synopsis above.

The first form of **mv** moves (changes the name of) *filename1* to *filename2*. If *filename2* already exists, it is removed before *filename1* is moved. If *filename2* has a mode which forbids writing, **mv** prints the mode (see **chmod**(2)) and reads the standard input to obtain a line; if the line begins with **y**, the move takes place, otherwise **mv** exits.

The second form of **mv** moves (changes the name of) *directory1* to *directory2*, *only* if *directory2* does not already exist — if it does, the third form applies.

The third form of **mv** moves one or more *filename*s (may also be directories) with their original names, into the last *directory* in the list.

mv refuses to move a file or directory onto itself.

OPTIONS

– Interpret all the following arguments to **mv** as file names. This allows file names starting with minus.

–**f** Force. Override any mode restrictions and the –**i** option. The –**f** option also suppresses any warning messages about modes which would potentially restrict overwriting.

–**i** Interactive mode. **mv** displays the name of the file or directory followed by a question mark whenever a move would replace an existing file or directory. If you type a line starting with **y**, **mv** moves the specified file or directory, otherwise **mv** does nothing with that file or directory.

SEE ALSO

cp(1), **ln**(1), **chmod**(2), **rename**(2)

DIAGNOSTICS

mv: *pathname*: **rename: Permission denied**
Attempted to move *pathname* into a directory that did not have write permission.

BUGS

If *filename1* and *filename2* are on different file systems, then **mv** must copy the file and delete the original. In this case the owner name becomes that of the copying process and any linking relationship with other files is lost.

Modification times may be different than expected when **mv** must copy the file's data, rather than simply updating a directory entry.

mv will not move a directory from one file system to another. Use **cp**(1) instead.

NAME

intro – introduction to device drivers, protocols, and network interfaces

DESCRIPTION

This section describes device drivers, high-speed network interfaces, and protocols available under SunOS. The system provides drivers for a variety of hardware devices, such as disks, magnetic tapes, serial communication lines, mice and frame buffers, as well as virtual devices such as pseudo-terminals and windows. SunOS provides hardware support and a network interface for the 10-Megabit Ethernet, along with interfaces for the IP protocol family and a STREAMS-based Network Interface Tap (NIT) facility.

In addition to describing device drivers that are supported by the 4.3BSD operating system, this section contains subsections that describe:

- SunOS-specific device drivers, under '4S'.
- Protocol families, under '4F'.
- Protocols and raw interfaces, under '4P'.
- STREAMS modules, under '4M'.
- Network interfaces, under '4N'.

Configuration

The SunOS kernel can be configured to include or omit many of the device drivers described in this section. The CONFIG section of the manual page gives the line(s) to include in the kernel configuration file for each machine architecture on which a device is supported. If no specific architectures are indicated, the configuration syntax applies to all Sun systems.

The GENERIC kernel is the default configuration for SunOS. It contains all of the optional drivers for a given machine architecture. See **config**(8), for details on configuring a new SunOS kernel.

The manual page for a device driver may also include a DIAGNOSTICS section, listing error messages that the driver might produce. Normally, these messages are logged to the appropriate system log using the kernel's standard message-buffering mechanism (see **syslogd**(8)); they may also appear on the system console.

Ioctls

Various special functions, such as querying or altering the operating characteristics of a device, are performed by supplying appropriate parameters to the **ioctl**(2) system call. These parameters are often referred to as "ioctls." Ioctls for a specific device are presented in the manual page for that device. Ioctls that pertain to a class of devices are listed in a manual page with a name that suggests the class of device, and ending in 'io', such as **mtio**(4) for magnetic tape devices, or **dkio**(4S) for disk controllers. In addition, some ioctls operate directly on higher-level objects such as files, terminals, sockets, and streams:

- Ioctls that operate directly on files, file descriptors, and sockets are described in **filio**(4). Note: the **fcntl**(2) system call is the primary method for operating on file descriptors as such, rather than on the underlying files. Also note that the **setsockopt** system call (see **getsockopt**(2)) is the primary method for operating on sockets as such, rather than on the underlying protocol or network interface. Ioctls for a specific network interface are documented in the manual page for that interface.
- Ioctls for terminals, including pseudo-terminals, are described in **termio**(4). This manual page includes information about both the BSD **termios** structure, as well as the System V **termio** structure.
- Ioctls for STREAMS are described in **streamio**(4).

Devices Always Present

Device drivers present in every kernel include:

- The paging device; see **drum**(4).
- Drivers for accessing physical, virtual, and I/O space in memory; see **mem**(4S).
- The data sink; see **null**(4).

Terminals and Serial Communications Devices

Serial communication lines are normally supported by the terminal driver; see **tty**(4). This driver manages serial lines provided by communications drivers, such as those described in **mti**(4S) and **zs**(4S). The terminal driver also handles serial lines provided by virtual terminals, such as the Sun console monitor described in **console**(4S), and true pseudo-terminals, described in **pty**(4).

Disk Devices

Drivers for the following disk controllers provide standard block and raw interfaces under SunOS;

- SCSI controllers, in **sd**(4S),
- Xylogics 450 and 451 SMD controllers, in **xy**(4S),
- Xylogics 7053 SMD controllers, in **xd**(4S).

Ioctls to query or set a disk's geometry and partitioning are described in **dkio**(4S).

Magnetic Tape Devices

Magnetic tape devices supported by SunOS include those described in **ar**(4S), **tm**(4S), **st**(4S), and **xt**(4S). Ioctls for all tape-device drivers are described in **mtio**(4S).

Frame Buffers

Frame buffer devices include color frame buffers described in the **cg***(4S) manual pages, monochrome frame buffers described in the **bw***(4S) manual pages, graphics processor interfaces described in the **gp***(4S) manual pages, and an indirect device for the console frame buffer described in **fb**(4S). Ioctls for all frame-buffer devices are described in **fbio**(4S).

Miscellaneous Devices

Miscellaneous devices include the console keyboard described in **kbd**(4S), the console mouse described in **mouse**(4S), window devices described in **win**(4S), and the DES encryption-chip interface described in **des**(4S).

Network-Interface Devices

SunOS supports the 10-Megabit Ethernet as its primary network interface; see **ec**(4S), **ie**(4S), and **le**(4S) for details. However, a software loopback interface, **lo**(4) is also supported. General properties of these network interfaces are described in **if**(4N), along with the ioctls that operate on them.

Support for network routing is described in **routing**(4N).

Protocols and Protocol Families

SunOS supports both socket-based and STREAMS-based network communications. The Internet protocol family, described in **inet**(4F), is the primary protocol family primary supported by SunOS, although the system can support a number of others. The raw interface provides low-level services, such as packet fragmentation and reassembly, routing, addressing, and basic transport for socket-based implementations. Facilities for communicating using an Internet-family protocol are generally accessed by specifying the AF_INET address family when binding a socket; see **socket**(2) for details.

Major protocols in the Internet family include:

- The Internet Protocol (IP) itself, which supports the universal datagram format, as described in **ip**(4P). This is the default protocol for SOCK_RAW type sockets within the AF_INET domain.
- The Transmission Control Protocol (TCP); see **tcp**(4P). This is the default protocol for SOCK_STREAM type sockets.
- The User Datagram Protocol (UDP); see **udp**(4P). This is the default protocol for SOCK_DGRAM type sockets.
- The Address Resolution Protocol (ARP); see **arp**(4P).
- The Internet Control Message Protocol (ICMP); see **icmp**(4P).

The Network Interface Tap (NIT) protocol, described in **nit**(4P), is a STREAMS-based facility for accessing the network at the link level.

SEE ALSO

fcntl(2), **getsockopt**(2), **ioctl**(2), **socket**(2), **ar**(4S), **arp**(4P), **dkio**(4S), **drum**(4), **ec**(4S), **fb**(4S), **fbio**(4S), **filio**(4), **icmp**(4P), **if**(4N), **inet**(4F), **ip**(4P), **kbd**(4S), **le**(4), **lo**(4), **mbio**(4S), **mem**(4S), **mti**(4), **mtio**(4), **nit**(4P), **null**(4), **pty**(4), **routing**(4N), **sd**(4S), **st**(4S) **streamio**(4), **tcp**(4P), **termio**(4), **tm**(4S), **tty**(4), **udp**(4P), **win**(4S), **xd**(4S), **xy**(4S), **zs**(4S)

LIST OF DEVICES, INTERFACES AND PROTOCOLS

Name	Appears on Page	Description
alm	**mcp**(4S)	Asynchronous Line Multiplexer
ar	**ar**(4S)	Archive 1/4 inch Streaming Tape Drive
arp	**arp**(4P)	Address Resolution Protocol
bk	**bk**(4)	line discipline for machine-machine communication
bwone	**bwone**(4S)	Sun-1 black and white frame buffer
bwtwo	**bwtwo**(4S)	Sun-3/Sun-2 black and white frame buffer
cgeight	**cgeight**(4S)	24-bitcolor memory frame buffer
cgfour	**cgfour**(4S)	Sun-3 color memory frame buffer
cgone	**cgone**(4S)	Sun-1 color graphics interface
cgsix	**cgsix**(4S)	Sun-3, Sun-4, and Sun-3x low-end graphics accelerator
cgthree	**cgthree**(4S)	Sun386i color memory frame buffer
cgtwo	**cgtwo**(4S)	Sun-3/Sun-2 color graphics interface
clone	**clone**(4)	open any minor device on a STREAMS driver
console	**console**(4S)	console driver and terminal emulator for the Sun workstation
des	**des**(4S)	DES encryption chip interface
dkio	**dkio**(4S)	generic disk control operations
drum	**drum**(4)	paging device
ec	**ec**(4S)	3Com 10 Mb/s Ethernet interface
fb	**fb**(4S)	driver for Sun console frame buffer
fbio	**fbio**(4S)	general properties of frame buffers
fd	**fd**(4S)	Disk driver for Floppy Disk Controllers
filio	**filio**(4)	ioctls that operate directly on files, file descriptors, and sockets
fpa	**fpa**(4S)	Sun-3 floating point accelerator
gpone	**gpone**(4S)	Sun-3/Sun-2 graphics processor
icmp	**icmp**(4P)	Internet Control Message Protocol
ie	**ie**(4S)	Intel 10 Mb/s Ethernet interface
if	**if**(4N)	general properties of network interfaces
inet	**inet**(4F)	Internet protocol family
ip	**ip**(4P)	Internet Protocol
kb	**kb**(4M)	Sun keyboard STREAMS module
kbd	**kbd**(4S)	Sun keyboard
kmem	**mem**(4S)	main memory and bus I/O space
ldterm	**ldterm**(4M)	standard terminal STREAMS module
le	**le**(4S)	Sun-3/50, Sun-3/60 10MB Ethernet interface
lo	**lo**(4)	software loopback network interface
lofs	**lofs**(4S)	loopback virtual file system
mbio	**mem**(4S)	main memory and bus I/O space
mbmem	**mem**(4S)	main memory and bus I/O space
mcp	**mcp**(4S)	MCP Multiprotocol Communications Processor
mem	**mem**(4S)	main memory and bus I/O space
mouse	**mouse**(4S)	Sun mouse
ms3	**mouse**(4S)	Sun mouse
ms	**ms**(4M)	Sun mouse STREAMS module

mti	**mti**(4S)	Systech MTI-800/1600 multi-terminal interface
mtio	**mtio**(4)	UNIX system magnetic tape interface
NFS	**nfs**(4P)	network file system
nif_pf	**nit_pf**(4M)	streams NIT packet filtering module
nit	**nit**(4P)	Network Interface Tap facility
nit_buf	**nit_buf**(4M)	streams NIT buffering module
nit_if	**nit_if**(4M)	streams NIT device interface module
null	**null**(4)	data sink
pp	**pp**(4)	Centronics-compatible parallel printer port
pty	**pty**(4)	pseudo terminal driver
root	**root**(4S)	pseudo-driver for Sun root disk
routing	**routing**(4N)	system supporting for local network packet routing
sd	**sd**(4S)	Disk driver for SCSI Disk Controllers
sockio	**sockio**(4)	ioctls that operate directly on sockets
st	**st**(4S)	Sysgen SC 4000 and Emulex MT-02 Tape Controller
streamio	**streamio**(4)	STREAMS ioctl commands
tcp	**tcp**(4P)	Transmission Control Protocol
termio	**termio**(4)	general terminal interface
tm	**tm**(4S)	tapemaster 1/2 inch tape drive
ttcompat	**ttcompat**(4M)	V7/4BSD compatibility STREAMS module
tty	**tty**(4)	controlling terminal interface
udp	**udp**(4P)	User Datagram Protocol
vme16d16	**mem**(4S)	main memory and bus I/O space
vme16d32	**mem**(4S)	main memory and bus I/O space
vme24d16	**mem**(4S)	main memory and bus I/O space
vme24d32	**mem**(4S)	main memory and bus I/O space
vme32d16	**mem**(4S)	main memory and bus I/O space
vme32d32	**mem**(4S)	main memory and bus I/O space
vp	**vp**(4S)	Ikon 10071-5 Versatec parallel printer interface
vpc	**vpc**(4S)	Systech VPC-2200 Versatec plotter and Centronics printer
win	**win**(4S)	Sun window system
xd	**xd**(4S)	Disk driver for Xylogics 7053 SMD Disk Controller
xt	**xt**(4S)	Xylogics 472 1/2 inch tape controller
xy	**xy**(4S)	Disk driver for Xylogics SMD Disk Controllers
zero	**zero**(4S)	source of zeroes
zs	**zs**(4S)	Zilog 8530 SCC serial comunications driver

NAME

bwone – Sun-1 black and white frame buffer

CONFIG — SUN-2 SYSTEM

device bwone0 at mbmem ? csr 0xc0000 priority 3

DESCRIPTION

The **bwone** interface provides access to Sun-1 system black and white graphics controller boards. It supports the ioctls described in **fbio**(4S).

FILES

/dev/bwone[0-9] device files

SEE ALSO

mmap(2), **fb**(4S), **fbio**(4S)

BUGS

Use of vertical-retrace interrupts is not supported.

The video state returned by the **FBIOGVIDEO** ioctl may be incorrect. It is not possible for the driver to determine the state of the hardware video enable bit, so it reports the last state stored by the **FBIOSVIDEO** ioctl. User processes which map the frame buffer can directly enable or disable the video, unknown to the driver.

NAME

bwtwo – Sun-3/Sun-2 black and white frame buffer

CONFIG — SUN-3/SUN-3x SYSTEMS

device bwtwo0 at obmem 1 csr 0xff000000 priority 4
device bwtwo0 at obmem 2 csr 0x100000 priority 4
device bwtwo0 at obmem 3 csr 0xff000000 priority 4
device bwtwo0 at obmem 4 csr 0xff000000
device bwtwo0 at obmem 7 csr 0xff000000 priority 4
device bwtwo0 at obmem ? csr 0x50300000 priority 4

The first synopsis line given above is used to generate a kernel for Sun-3/75, Sun-3/140 or Sun-3/160 systems; the second, for a Sun-3/50 system; the third, for a Sun-3/260 system; the fourth, for a Sun-3/110 system; the fifth, for a Sun-3/60 system; and the sixth for Sun-3/80 and Sun-3/470 systems.

CONFIG — SUN-2 SYSTEM

device bwtwo0 at obmem 1 csr 0x700000 priority 4
device bwtwo0 at obio 2 csr 0x0 priority 4

The first synopsis line given above is used to generate a kernel for a Sun-2/120 or Sun-2/170 system; the second, for a Sun-2/50 or Sun-2/160 system.

CONFIG — Sun386i SYSTEM

device bwtwo0 at obmem ? csr 0xA0200000

DESCRIPTION

The **bwtwo** interface provides access to Sun monochrome memory frame buffers. It supports the ioctls described in **fbio**(4S).

If **flags 0x1** is specified, frame buffer write operations are buffered through regular high-speed RAM. This "copy memory" mode of operation speeds frame buffer accesses, but consumes an extra 128K bytes of memory. Only Sun-2, Sun-3/75, and Sun-3/160 systems support copy memory; on other systems a warning message is printed and the flag is ignored.

Reading or writing to the frame buffer is not allowed — you must use the **mmap**(2) system call to map the board into your address space.

FILES

/dev/bwtwo[0-9] device files

SEE ALSO

mmap(2), **cgfour**(4S), **fb**(4S), **fbio**(4S)

BUGS

Use of vertical-retrace interrupts is not supported.

NAME

cgeight – 24-bit color memory frame buffer

CONFIG — SUN-3 AND SUN-4 SYSTEMS

device cgeight0 at obmem 7 csr 0xff300000 priority 4
device cgeight0 at obio 4 csr 0xfb300000 priority 4

The first synopsis line should be used to generate a kernel for the Sun-3/60; the second synopsis for a Sun-4/110 or Sun-4/150 system.

CONFIG — SUN-3x SYSTEM

device cgeight0 at obio ? csr 0x50300000 priority 4

DESCRIPTION

The **cgeight** is a 24-bit color memory frame buffer with a monochrome overlay plane and an overlay enable plane implemented optionally on the Sun-4/110, Sun-4/150, Sun-3/60, Sun-3/470 and Sun-3/80 system models. It provides the standard frame buffer interface as defined in **fbio**(4S).

In addition to the ioctls described under **fbio**(4S), the **cgeight** interface responds to two **cgeight**-specific colormap ioctls, **FBIOPUTCMAP** and **FBIOGETCMAP**. **FBIOPUTCMAP** returns no information other than success/failure using the ioctl return value. **FBIOGETCMAP** returns its information in the arrays pointed to by the red, green, and blue members of its **fbcmap** structure argument; **fbcmap** is defined in **/usr/include/sun/fbio.h** as:

```
struct fbcmap {
        int             index;          /* first element (0 origin) */
        int             count;          /* number of elements */
        unsigned char   *red;           /* red color map elements */
        unsigned char   *green;         /* green color map elements */
        unsigned char   *blue;          /* blue color map elements */
};
```

The driver uses color board vertical-retrace interrupts to load the colormap.

The systems have an overlay plane colormap, which is accessed by encoding the plane group into the index value with the **PIX_GROUP** macro (see **/usr/include/pixrect/pr_planegroups.h**).

When using the **mmap** system call to map in the **cgeight** frame buffer. The device looks like:

```
DACBASE: 0x200000    -> Brooktree Ramdac        16 bytes
         0x202000    -> P4 Regiter               4 bytes
OVLBASE: 0x210000    -> Overlay Plane           1152x900x1
         0x230000    -> Overlay Enable Planea   1152x900x1
         0x250000    -> 24-bit Frame Buffera    1152x900x32
```

FILES

/dev/cgeight0
/usr/include/sun/fbio.h
/usr/include/pixrect/pr_planegroups.h

SEE ALSO

mmap(2), **fbio**(4S)

NAME

cgfour – Sun-3 color memory frame buffer

CONFIG — SUN-3 SYSTEM

device cgfour0 at obmem 4 csr 0xff000000 priority 4
device cgfour0 at obmem 7 csr 0xff000000 priority 4

The first synopsis line given should be used to generate a kernel for the Sun-3/110 system; and the second, for a Sun-3/60 system.

CONFIG — SUN-3x SYSTEM

device cgfour0 at obmem ? csr 0x50300000 priority 4

DESCRIPTION

The **cgfour** is a color memory frame buffer with a monochrome overlay plane and an overlay enable plane implemented on the Sun-3/110 system and some Sun-3/60 system models. It provides the standard frame buffer interface as defined in **fbio**(4S).

In addition to the ioctls described under **fbio**(4S), the **cgfour** interface responds to two **cgfour**-specific colormap ioctls, **FBIOPUTCMAP** and **FBIOGETCMAP**. **FBIOPUTCMAP** returns no information other than success/failure using the ioctl return value. **FBIOGETCMAP** returns its information in the arrays pointed to by the red, green, and blue members of its **fbcmap** structure argument; **fbcmap** is defined in **<sun/fbio.h>** as:

```
struct fbcmap {
        int             index;          /* first element (0 origin) */
        int             count;          /* number of elements */
        unsigned char   *red;           /* red color map elements */
        unsigned char   *green;         /* green color map elements */
        unsigned char   *blue;          /* blue color map elements */
};
```

The driver uses color board vertical-retrace interrupts to load the colormap.

The Sun-3/60 system has an overlay plane colormap, which is accessed by encoding the plane group into the index value with the **PIX_GROUP** macro (see **<pixrect/pr_planegroups.h>**).

FILES

/dev/cgfour0

SEE ALSO

mmap(2), **fbio**(4S)

NAME

cgone – Sun-1 color graphics interface

CONFIG — SUN-2 SYSTEM

device cgone0 at mbmem ? csr 0xec000 priority 3

DESCRIPTION

The **cgone** interface provides access to the Sun-1 system color graphics controller board, which is normally supplied with a 13'' or 19'' RS170 color monitor. It provides the standard frame buffer interface as defined in **fbio**(4S).

It supports the **FBIOGPIXRECT** ioctl which allows SunView to be run on it; see **fbio**(4S)

The hardware consumes 16 kilobytes of Multibus memory space. The board starts at standard addresses 0xE8000 or 0xEC000. The board must be configured for interrupt level 3.

FILES

/dev/cgone[0-9]

SEE ALSO

mmap(2), **fbio**(4S)

BUGS

Use of color board vertical-retrace interrupts is not supported.

NAME

cgsix – low-end graphics accelerator with 8-bit color frame buffer

CONFIGURATION — SUN-3, SUN-4, and SUN-3x SYSTEMS

device cgsix0 at obmem ? csr 0xff000000 priority 4
device cgsix0 at obmem ? csr 0xfb000000 priority 4
device cgsix0 at obmem ? csr 0x50000000 priority 4

The first synopsis line is used for Sun-3 systems, the second for Sun-4 systems, and the third for Sun-3x systems.

DESCRIPTION

cgsix(4S) is a low-end, P4 graphics accelerator that increases vector and polygon drawing performance. It uses an 8-bit color frame buffer and provides the standard frame buffer interface, as defined in **fbio**(4S).

In addition to the ioctls described under **fbio**, the **cgsix** interface responds to two **cgsix**-specific colormap ioctls, **FBIOPUTCMAP** and **FBIOGETCMAP**. **FBIOPUTCMAP** returns no information other than success/failure using the ioctl return value. **FBIOGETCMAP** returns its information in the arrays pointed to by the red, green, and blue members of its **fbcmap** structure argument; **fbcmap** is defined in **/usr/include/sun/fbio.h** as:

```
struct fbcmap {
        int             index;  /* first element (0 origin) */
        int             count;  /* number of elements       */
        unsigned char   *red;   /* red color map elements   */
        unsigned char   *green; /* green color map elements */
        unsigned char   *blue;  /* blue color map elements  */
};
```

The driver uses color board vertical-retrace interrupts to load the colormap.

cgsix contains memory that may be mapped in through calls to the **cgsixmmap**( ) function. Each portion of this memory is specified by an offset from the base address that is configured into the kernel. The portions that may be mapped are the colormap, the FBC/TEC, the FHC/THC, and the framebuffer. The exact offsets are defined in **/usr/includesundev/cg6reg.h.**

FILES

/dev/cgsix0
/usr/include/sundev/cg6reg.h

SEE ALSO

mmap(2), **fbio**(4S)

NAME

console – console driver and terminal emulator for the Sun workstation

CONFIG

None; included in standard system.

SYNOPSIS

```
#include <fcntl.h>
#include <sys/termios.h>
open("/dev/console", mode);
```

DESCRIPTION

console is an indirect driver for the Sun console terminal. On a Sun workstation, this driver refers to the workstation console driver, which implements a standard UNIX system terminal. On a Sun server without a keyboard or a frame buffer, this driver refers to the CPU serial port driver (**zs**(4S)); a terminal is normally connected to this port.

The workstation console does not support any of the **termio**(4) device control functions specified by flags in the **c_cflag** word of the **termios** structure or by the **IGNBRK, IGNPAR, PARMRK,** or **INPCK** flags in the **c_iflag** word of the **termios** structure, as these functions apply only to asynchronous serial ports. All other **termio**(4) functions must be performed by STREAMS modules pushed atop the driver; when a slave device is opened, the **ldterm**(4M) and **ttcompat**(4M) STREAMS modules are automatically pushed on top of the stream, providing the standard **termio**(4) interface.

The workstation console driver calls the PROM resident monitor to output data to the console frame buffer. Keystrokes from the CPU serial port to which the keyboard is connected are routed through the keyboard STREAMS module (**kb**(4M)) and treated as input.

When the Sun window system **win**(4S) is active, console input is directed through the window system rather than being treated as input by the workstation console driver.

IOCTLS

An ioctl TIOCCONS can be applied to pseudo-terminals (**pty**(4)) to route output that would normally appear on the console to the pseudo-terminal instead. Thus, the window system does a TIOCCONS on a pseudo-terminal so that the system will route console output to the window to which that pseudo-terminal is connected, rather than routing output through the PROM monitor to the screen, since routing output through the PROM monitor destroys the integrity of the screen. Note: when you use TIOCCONS in this way, the console *input* is routed from the pseudo-terminal as well.

If a TIOCCONS is performed on **/dev/console**, or the pseudo-terminal to which console output is being routed is closed, output to the console will again be routed to the workstation console driver.

ANSI STANDARD TERMINAL EMULATION

The Sun Workstation's PROM monitor provides routines that emulates a standard ANSI X3.64 terminal.

Note: the VT100 also follows the ANSI X3.64 standard but both the Sun and the VT100 have nonstandard extensions to the ANSI X3.64 standard. The Sun terminal emulator and the VT100 are *not* compatible in any true sense.

The Sun console displays 34 lines of 80 ASCII characters per line, with scrolling, (x, y) cursor addressability, and a number of other control functions.

The Sun console displays a non-blinking block cursor which marks the current line and character position on the screen. ASCII characters between 0x20 (space) and 0x7E (tilde) inclusive are printing characters — when one is written to the Sun console (and is not part of an escape sequence), it is displayed at the current cursor position and the cursor moves one position to the right on the current line. If the cursor is already at the right edge of the screen, it moves to the first character position on the next line. If the cursor is already at the right edge of the screen on the bottom line, the Line-feed function is performed (see CTRL-J below), which scrolls the screen up by one or more lines or wraps around, before moving the cursor to the first character position on the next line.

Control Sequence Syntax

The Sun console defines a number of control sequences which may occur in its input. When such a sequence is written to the Sun console, it is not displayed on the screen, but effects some control function as described below, for example, moves the cursor or sets a display mode.

Some of the control sequences consist of a single character. The notation

CTRL-*X*

for some character *X* , represents a control character.

Other ANSI control sequences are of the form

ESC [*params*char

Spaces are included only for readability; these characters must occur in the given sequence without the intervening spaces.

ESC represents the ASCII escape character (ESC, CTRL-[, 0x1B).

[The next character is a left square bracket '[' (0x5B).

params are a sequence of zero or more decimal numbers made up of digits between 0 and 9, separated by semicolons.

char represents a function character, which is different for each control sequence.

Some examples of syntactically valid escape sequences are (again, ESC represent the single ASCII character 'Escape'):

ESC[m	*select graphic rendition with default parameter*
ESC[7m	*select graphic rendition with reverse image*
ESC[33;54H	*set cursor position*
ESC[123;456;0;;3;B	*move cursor down*

Syntactically valid ANSI escape sequences which are not currently interpreted by the Sun console are ignored. Control characters which are not currently interpreted by the Sun console are also ignored.

Each control function requires a specified number of parameters, as noted below. If fewer parameters are supplied, the remaining parameters default to 1, except as noted in the descriptions below.

If more than the required number of parameters is supplied, only the last *n* are used, where *n* is the number required by that particular command character. Also, parameters which are omitted or set to zero are reset to the default value of 1 (except as noted below).

Consider, for example, the command character M which requires one parameter. ESC[;M and ESC[0M and ESC[M and ESC[23;15;32;1M are all equivalent to ESC[1M and provide a parameter value of 1. Note: ESC[;5M (interpreted as 'ESC[5M') is *not* equivalent to ESC[5;M (interpreted as 'ESC[5;1M') which is ultimately interpreted as 'ESC[1M').

In the syntax descriptions below, parameters are represented as '**#**' or '**#1;#2**'.

ANSI Control Functions

The following paragraphs specify the ANSI control functions implemented by the Sun console. Each description gives:

- the control sequence syntax
- the hex equivalent of control characters where applicable
- the control function name and ANSI or Sun abbreviation (if any).
- description of parameters required, if any
- description of the control function
- for functions which set a mode, the initial setting of the mode. The initial settings can be restored with the SUNRESET escape sequence.

Control Character Functions

CTRL-G (0x7) Bell (BEL)

The Sun Workstation Model 100 and 100U is not equipped with an audible bell. It 'rings the bell' by flashing the entire screen. The Sun-2 models have an audible bell which beeps. The window system flashes the window.

CTRL-H (0x8) Backspace (BS)

The cursor moves one position to the left on the current line. If it is already at the left edge of the screen, nothing happens.

CTRL-I (0x9) Tab (TAB)

The cursor moves right on the current line to the next tab stop. The tab stops are fixed at every multiple of 8 columns. If the cursor is already at the right edge of the screen, nothing happens; otherwise the cursor moves right a minimum of one and a maximum of eight character positions.

CTRL-J (0xA) Line-feed (LF)

The cursor moves down one line, remaining at the same character position on the line. If the cursor is already at the bottom line, the screen either scrolls up or "wraps around" depending on the setting of an internal variable *S* (initially 1) which can be changed by the ESC[r control sequence. If *S* is greater than zero, the entire screen (including the cursor) is scrolled up by *S* lines before executing the line-feed. The top *S* lines scroll off the screen and are lost. *S* new blank lines scroll onto the bottom of the screen. After scrolling, the line-feed is executed by moving the cursor down one line.

If *S* is zero, 'wrap-around' mode is entered. 'ESC [1 r' exits back to scroll mode. If a line-feed occurs on the bottom line in wrap mode, the cursor goes to the same character position in the top line of the screen. When any line-feed occurs, the line that the cursor moves to is cleared. This means that no scrolling occurs. Wrap-around mode is not implemented in the window system.

The screen scrolls as fast as possible depending on how much data is backed up waiting to be printed. Whenever a scroll must take place and the console is in normal scroll mode ('ESC [1 r'), it scans the rest of the data awaiting printing to see how many line-feeds occur in it. This scan stops when any control character from the set {**VT, FF, SO, SI, DLE, DC1, DC2, DC3, DC4, NAK, SYN, ETB, CAN, EM, SUB, ESC, FS, GS, RS, US**} is found. At that point, the screen is scrolled by N lines ($N \geq 1$) and processing continues. The scanned text is still processed normally to fill in the newly created lines. This results in much faster scrolling with scrolling as long as no escape codes or other control characters are intermixed with the text.

See also the discussion of the 'Set scrolling' (**ESC[r**) control function below.

CTRL-K (0xB) Reverse Line-feed

The cursor moves up one line, remaining at the same character position on the line. If the cursor is already at the top line, nothing happens.

CTRL-L (0xC) Form-feed (FF)

The cursor is positioned to the Home position (upper-left corner) and the entire screen is cleared.

CTRL-M (0xD) Return (CR)

The cursor moves to the leftmost character position on the current line.

Escape Sequence Functions

CTRL-[(0x1B) Escape (ESC)

This is the escape character. Escape initiates a multi-character control sequence.

ESC[#@ Insert Character (ICH)

Takes one parameter, # (default 1). Inserts # spaces at the current cursor position. The tail of the current line starting at the current cursor position inclusive is shifted to the right by # character positions to make room for the spaces. The rightmost # character positions shift off the line and are lost. The position of the cursor is unchanged.

ESC[#A Cursor Up (CUU)

Takes one parameter, # (default 1). Moves the cursor up # lines. If the cursor is fewer than # lines from the top of the screen, moves the cursor to the topmost line on the screen. The character position of the cursor on the line is unchanged.

ESC[#B Cursor Down (CUD)

Takes one parameter, # (default 1). Moves the cursor down # lines. If the cursor is fewer than # lines from the bottom of the screen, move the cursor to the last line on the screen. The character position of the cursor on the line is unchanged.

ESC[#C Cursor Forward (CUF)

Takes one parameter, # (default 1). Moves the cursor to the right by # character positions on the current line. If the cursor is fewer than # positions from the right edge of the screen, moves the cursor to the rightmost position on the current line.

ESC[#D Cursor Backward (CUB)

Takes one parameter, # (default 1). Moves the cursor to the left by # character positions on the current line. If the cursor is fewer than # positions from the left edge of the screen, moves the cursor to the leftmost position on the current line.

ESC[#E Cursor Next Line (CNL)

Takes one parameter, # (default 1). Positions the cursor at the leftmost character position on the #-th line below the current line. If the current line is less than # lines from the bottom of the screen, positions the cursor at the leftmost character position on the bottom line.

ESC[#1;#2f Horizontal And Vertical Position (HVP)

or

ESC[#1;#2H Cursor Position (CUP)

Takes two parameters, #1 and #2 (default 1, 1). Moves the cursor to the #2-th character position on the #1-th line. Character positions are numbered from 1 at the left edge of the screen; line positions are numbered from 1 at the top of the screen. Hence, if both parameters are omitted, the default action moves the cursor to the home position (upper left corner). If only one parameter is supplied, the cursor moves to column 1 of the specified line.

ESC[J Erase in Display (ED)

Takes no parameters. Erases from the current cursor position inclusive to the end of the screen. In other words, erases from the current cursor position inclusive to the end of the current line and all lines below the current line. The cursor position is unchanged.

ESC[K Erase in Line (EL)

Takes no parameters. Erases from the current cursor position inclusive to the end of the current line. The cursor position is unchanged.

ESC[#L Insert Line (IL)

Takes one parameter, # (default 1). Makes room for # new lines starting at the current line by scrolling down by # lines the portion of the screen from the current line inclusive to the bottom. The # new lines at the cursor are filled with spaces; the bottom # lines shift off the bottom of the screen and are lost. The position of the cursor on the screen is unchanged.

ESC[#M Delete Line (DL)

Takes one parameter, # (default 1). Deletes # lines beginning with the current line. The portion of the screen from the current line inclusive to the bottom is scrolled upward by # lines. The # new lines scrolling onto the bottom of the screen are filled with spaces; the # old lines beginning at the cursor line are deleted. The position of the cursor on the screen is unchanged.

ESC[#P Delete Character (DCH)

Takes one parameter, # (default 1). Deletes # characters starting with the current cursor position. Shifts to the left by # character positions the tail of the current line from the current cursor position inclusive to the end of the line. Blanks are shifted into the rightmost # character positions. The position of the cursor on the screen is unchanged.

ESC[#m Select Graphic Rendition (SGR)

Takes one parameter, # (default 0). Note: unlike most escape sequences, the parameter defaults to zero if omitted. Invokes the graphic rendition specified by the parameter. All following printing characters in the data stream are rendered according to the parameter until the next occurrence of this escape sequence in the data stream. Currently only two graphic renditions are defined:

0 Normal rendition.

7 Negative (reverse) image.

Negative image displays characters as white-on-black if the screen mode is currently black-on white, and vice-versa. Any non-zero value of # is currently equivalent to 7 and selects the negative image rendition.

ESC[p Black On White (SUNBOW)

Takes no parameters. Sets the screen mode to black-on-white. If the screen mode is already black-on-white, has no effect. In this mode spaces display as solid white, other characters as black-on-white. The cursor is a solid black block. Characters displayed in negative image rendition (see 'Select Graphic Rendition' above) is white-on-black in this mode. This is the initial setting of the screen mode on reset.

ESC[q White On Black (SUNWOB)

Takes no parameters. Sets the screen mode to white-on-black. If the screen mode is already white-on-black, has no effect. In this mode spaces display as solid black, other characters as white-on-black. The cursor is a solid white block. Characters displayed in negative image rendition (see 'Select Graphic Rendition' above) is black-on-white in this mode. The initial setting of the screen mode on reset is the alternative mode, black on white.

ESC[#r Set scrolling (SUNSCRL)

Takes one parameter, # (default 0). Sets to # an internal register which determines how many lines the screen scrolls up when a line-feed function is performed with the cursor on the bottom line. A parameter of 2 or 3 introduces a small amount of "jump" when a scroll occurs. A parameter of 34 clears the screen rather than scrolling. The initial setting is 1 on reset.

A parameter of zero initiates "wrap mode" instead of scrolling. In wrap mode, if a linefeed occurs on the bottom line, the cursor goes to the same character position in the top line of the screen. When any linefeed occurs, the line that the cursor moves to is cleared. This means that no scrolling ever occurs. 'ESC [1 r' exits back to scroll mode.

For more information, see the description of the Line-feed (CTRL-J) control function above.

ESC[s Reset terminal emulator (SUNRESET)

Takes no parameters. Resets all modes to default, restores current font from PROM. Screen and cursor position are

4014 TERMINAL EMULATION

The PROM monitor for Sun models 100U and 150U provides the Sun Workstation with the capability to emulate a subset of the Tektronix 4014 terminal. This feature does not exist in other Sun PROMs and will be removed from models 100U and 150U in future Sun releases. **tektool**(1) provides Tektronix 4014 terminal emulation and should be used instead of relying on the capabilities of the PROM monitor.

FILES

/dev/console

SEE ALSO

tektool(1) **kb**(4M), **pty**(4), **termio**(4), **ttcompat**(4M), **ldterm**(4M), **win**(4S), **zs**(4S)

ANSI Standard X3.64, "*Additional Controls for Use with ASCII*", Secretariat: CBEMA, 1828 L St., N.W., Washington, D.C. 20036.

BUGS

TIOCCONS should be restricted to the owner of **/dev/console**.

NAME

db – SunDials STREAMS module

CONFIG

pseudo-device*db***n**

SYNOPSIS

```
#include <sys/stream.h>
#include <sundev/vuid_event.h>
#include <sundev/dbio.h>
#include <sys/time.h>
#include <sys/ioctl.h>
open("/dev/dialbox", O_RDWR);
ioctl(fd, I_PUSH, "db");
```

DESCRIPTION

The **db** STREAMS module processes the byte streams generated by the SunDials dial box. This dial box generates a stream of bytes, which encode the identity of the dials and the amount by which they are turned.

Each dial sample in the byte stream consists of three bytes. The first byte identifies which dial was turned and the next two bytes return the delta in signed binary format. An event from a dial is constrained to lie between 0x80 and 0x87. When bound to an application using the window system, *Virtual User Input Device* events are generated.

A stream with **db** pushed into it can emit *firm_events* as specified by the protocol of a VUID. **db** understands the **VUIDSFIRMAT** and **VUIDGFORMAT** ioctls (see reference below), as defined in **/usr/include/sundev/dbio.h** and **/usr/include/sundev/vuid_event.h**. All other *ioctl*s are passed downstream. **db** sets the parameters of a serial port when it is opened. No **termios**(4) *ioctl*s should be performed on a **db** stream, as **db** expects the device parameters to remain as it set them.

IOCTLS

VUIDSFORMAT
VUIDGFORMAT These are standard *Virtual User Input Device ioctl*s. See *SunView 1 System Programmer's Guide* for a description of their operation.

FILES

/usr/include/sundev/dbio.h
/usr/include/sundev/vuid_event.h
/usr/include/sys/ioctl.h
/usr/include/sys/stream.h
/usr/include/sys/time.h

SEE ALSO

termios(4), **dbconfig**(6), **dialtest**(6)

SunView 1 System Programmer's Guide, SunDials Programmers Guide

BUGS

VUIDSADDR and **VUIDGADDR** are not supported.

WARNING

The SunDials dial box must be used with a serial port.

NAME

des – DES encryption chip interface

CONFIG — SUN-3 SYSTEM

device des0 at obio ? csr 0x1c0000

CONFIG — SUN-3x SYSTEM

device des0 at obio ? csr 0x66002000

CONFIG — SUN-2 SYSTEM

des0 at virtual ? csr 0xee1800

SYNOPSIS

#include <sys/des.h>

DESCRIPTION

The **des** driver provides a high level interface to the AmZ8068 Data Ciphering Processor, a hardware implementation of the NBS Data Encryption Standard.

The high level interface provided by this driver is hardware independent and could be shared by future drivers in other systems.

The interface allows access to two modes of the DES algorithm: Electronic Code Book (ECB) and Cipher Block Chaining (CBC). All access to the DES driver is through **ioctl**(2) calls rather than through reads and writes; all encryption is done in-place in the user's buffers.

IOCTLS

The ioctls provided are:

DESIOCBLOCK

This call encrypts/decrypts an entire buffer of data, whose address and length are passed in the '**struct desparams**' addressed by the argument. The length must be a multiple of 8 bytes.

DESIOCQUICK

This call encrypts/decrypts a small amount of data quickly. The data is limited to **DES_QUICKLEN** bytes, and must be a multiple of 8 bytes. Rather than being addresses, the data is passed directly in the '**struct desparams**' argument.

FILES

/dev/des

SEE ALSO

des(1), **des_crypt**(3)

Federal Information Processing Standards Publication 46

AmZ8068 DCP Product Description, Advanced Micro Devices

NAME

fbio – general properties of frame buffers

DESCRIPTION

All of the Sun frame buffers support the same general interface. Each responds to a **FBIOGTYPE** ioctl which returns information in a structure defined in **<sun/fbio.h>**:

```
struct  fbtype {
        int     fb_type;        /* as defined below */
        int     fb_height;      /* in pixels */
        int     fb_width;       /* in pixels */
        int     fb_depth;       /* bits per pixel */
        int     fb_cmsize;      /* size of color map (entries) */
        int     fb_size;        /* total size in bytes */
};

#define FBTYPE_SUN1BW       0
#define FBTYPE_SUN1COLOR    1
#define FBTYPE_SUN2BW       2
#define FBTYPE_SUN2COLOR    3
#define FBTYPE_SUN2GP       4
#define FBTYPE_SUN5COLOR    5
#define FBTYPE_SUN3COLOR    6
#define FBTYPE_MEMCOLOR     7
#define FBTYPE_SUN4COLOR    8
```

Each device has an **FBTYPE** which is used by higher-level software to determine how to perform raster-op and other functions. Each device is used by opening it, doing an **FBIOGTYPE** ioctl to see which frame buffer type is present, and thereby selecting the appropriate device-management routines.

Full-fledged frame buffers (that is, those that run SunView) implement an **FBIOGPIXRECT** ioctl, which returns a pixrect. This call is made only from inside the kernel. The returned pixrect is used by **win**(4S) for cursor tracking and colormap loading.

FBIOSVIDEO and **FBIOGVIDEO** are general-purpose ioctls for controlling possible video features of frame buffers. They are defined in **<sun/fbio.h>**. These ioctls either set or return the value of a flags integer. At this point, only the **FBVIDEO_ON** option is available, controlled by **FBIOSVIDEO**. **FBIOGVIDEO** returns the current video state.

The **FBIOSATTR** and **FBIOGATTR** ioctls allow access to special features of newer frame buffers. They use the following structures as defined in **<sun/fbio.h>**:

```
#define FB_ATTR_NDEVSPECIFIC    8       /* no. of device specific values */
#define FB_ATTR_NEMUTYPES 4     /* no. of emulation types */
struct fbsattr {
        int     flags;                  /* misc flags */
#define FB_ATTR_AUTOINIT        1       /* emulation auto init flag */
#define FB_ATTR_DEVSPECIFIC 2   /* dev. specific stuff valid flag */
        int     emu_type;               /* emulation type (-1 if unused) */
        int     dev_specific[FB_ATTR_NDEVSPECIFIC]; /* catchall */
};
struct fbgattr {
        int     real_type;              /* real device type */
        int     owner;                  /* PID of owner, 0 if myself */
        struct fbtype fbtype;           /* fbtype info for real device */
        struct fbsattr sattr;           /* see above */
        int     emu_types[FB_ATTR_NEMUTYPES];   /* possible emulations */
                                        /* (-1 if unused) */
};
```

SEE ALSO

mmap(2), **bwone**(4S), **bwtwo**(4S), **cgeight**(4S), **cgfour**(4S), **cgone**(4S), **cgtwo**(4S), **fb**(4S), **gpone**(4S), **win**(4S)

BUGS

FBIOSATTR and **FBIOGATTR** are only supported by the **cgfour**(4S) and **cgeight**(4S) frame buffers.

The **FBVIDEO_ON** flag my be incorrect for Sun-1 system black and white frame buffers; see **bwone**(4S).

NAME

fd – Disk driver for Floppy Disk Controllers

CONFIG — Sun386i SYSTEMS

controller fdc0 at atmem ? csr 0x1000 dmachan 2 irq 6 priority 2
disk fd0 at fdc0 drive 0 flags 0

CONFIG — SUN-3x SYSTEMS

controller fdc0 at obio ? csr 0x6e000000 priority 6 vector fdintr 0x5c

This synopsis should be used to generate a kernel for a Sun-3/80 system only.

AVAILABILITY

Sun386i and Sun-3/80 systems only.

DESCRIPTION

The block-files access the disk using the system's normal buffering mechanism and may be read and written without regard to physical disk records. There is also a 'raw' interface that provides for direct transmission between the disk and the user's read or write buffer. A single read or write call usually results in one I/O operation; therefore raw I/O is considerably more efficient when many words are transmitted. The names of the raw files conventionally begin with an extra 'r.'

Disk Support

The **fd0** partition on a floppy disk is normally used for the file system.

FILES

1.44 MB Floppy Disk Drives:

/dev/fd0a	block file
/dev/fd0c	block file **/dev/rfd0a** raw file **/dev/rfd0c** raw file

720 K Floppy Disk Drives:

/dev/fdl0a	block file
/dev/fdl0c	block file
/dev/rfdl0a	raw file
/dev/rfdl0c	raw file

SEE ALSO

dkio(4S)

DIAGNOSTICS

fd drv %d, trk %d: %s

A command such as read or write encountered a format-related error condition. The value of *%s* is derived from the error number given by the controller, indicating the nature of the error. The track number is relative to the beginning of the partition involved.

fd drv %d, blk %d: %s

A command such as read or write encountered an error condition related to I/O. The value of *%s* is derived from the error number returned by the controller and indicates the nature of the error. The block number is relative to the start of the partition involved.

fd controller: %s

An error occurred in the controller. The value of *%s* is derived from the status returned by the controller and specifies the error encountered.

fd(%d):%s please insert

I/O was attempted while the floppy drive door was not latched. The value of *%s* indicates which disk was expected to be in the drive.

NAME

filio – ioctls that operate directly on files, file descriptors, and sockets

SYNOPSIS

#include <sys/filio.h>

DESCRIPTION

The IOCTL's listed in this manual page apply directly to files, file descriptors, and sockets, independent of any underlying device or protocol.

Note: the **fcntl**(2V) system call is the primary method for operating on file descriptors as such, rather than on the underlying files.

IOCTLS for File Descriptors

FIOCLEX The argument is ignored. Set the close-on-exec flag for the file descriptor passed to **ioctl**. This flag is also manipulated by the **F_SETFD** command of **fcntl**(2V).

FIONCLEX The argument is ignored. Clear the close-on-exec flag for the file descriptor passed to **ioctl**.

IOCTLs for Files

FIONREAD The argument is a pointer to a **long**. Set the value of that **long** to the number of immediately readable characters from whatever the descriptor passed to **ioctl** refers to. This works for files, pipes, sockets, and terminals.

FIONBIO The argument is a pointer to an **int**. Set or clear non-blocking I/O. If the value of that **int** is a 1 (one) the descriptor is set for non-blocking I/O. If the value of that **int** is a 0 (zero) the descriptor is cleared for non-blocking I/O.

FIOASYNC The argument is a pointer to an **int**. Set or clear asynchronous I/O. If the value of that **int** is a 1 (one) the descriptor is set for asynchronous I/O. If the value of that **int** is a 0 (zero) the descriptor is cleared for asynchronous I/O.

FIOSETOWN The argument is a pointer to an **int**. Set the process-group ID that will subsequently receive **SIGIO** or **SIGURG** signals for the object referred to by the descriptor passed to **ioctl** to the value of that **int**.

FIOGETOWN The argument is a pointer to an **int**. Set the value of that **int** to the process-group ID that is receiving **SIGIO** or **SIGURG** signals for the object referred to by the descriptor passed to **ioctl**.

SEE ALSO

ioctl(2), **fcntl**(2V), **getsockopt**(2), **sockio**(4)

NAME

fpa – Sun-3/Sun-3x floating-point accelerator

CONFIG — SUN-3/SUN-3X SYSTEM

fpa0 at virtual ? csr 0xe0000000

SYNOPSIS

#include <sundev/fpareg.h>
open("/dev/fpa", flags);

DESCRIPTION

FPA and FPA+ are compatible floating point accelerators available on certain Sun-3 and Sun-3x systems. They provide hardware contexts for simultaneous use by up to 32 processes. The same **fpa** device driver manages either FPA or FPA+ hardware.

Processes access the device using **open**(2) and **close**(2) system calls, and the FPA is automatically mapped into the process' address space by SunOS. This is normally provided transparently at compile time by a compiler option, such as the **–ffpa** option to **cc**(1V).

The valid **ioctl**(2) system calls are used only by diagnostics and by system administration programs, such as **fpa_download**(8).

IOCTLS

FPA_ACCESS_OFF	Clear **FPA_ACCESS_BIT** in FPA state register to disable access to constants RAM using FPA load pointer.
FPA_ACCESS_ON	Set **FPA_ACCESS_BIT** in FPA state register to enable access to constants RAM using FPA load pointer.
FPA_FAIL	Disable the FPA.
FPA_GET_DATAREGS	Return the contents of 8 FPA registers.
FPA_INIT_DONE	Called when downloading is complete. Allows multiple users to access the FPA.
FPA_LOAD_OFF	Set **FPA_LOAD_BIT** in FPA state register to disable access to microstore or map RAM via FPA load pointer.
FPA_LOAD_ON	Set **FPA_LOAD_BIT** in FPA state register to enable access to microstore or map RAM using FPA load pointer.

The following two ioctls are for diagnostic use only. **fpa** must be compiled with **FPA_DIAGNOSTICS_ONLY** defined to enable these two calls.

FPA_WRITE_STATE	Overwrite the FPA state register.
FPA_WRITE_HCP	Write to the hard clear pipe register.

ERRORS

The following error messages are returned by **open** system calls only.

EBUSY	All 32 FPA contexts are being used.
EEXIST	The current process has already opened **/dev/fpa**.
EIO	Downloading has not completed, so only 1 root process can have the FPA open at a time.
ENETDOWN	FPA is disabled.
ENOENT	68881 chip does not exist.
ENXIO	FPA board does not exist.

The following error messages are returned by **ioctl** system calls only.

EINVAL	Invalid ioctl. This may occur if diagnostic only ioctls, **FPA_WRITE_STATE** or **FPA_WRITE_HCP**, are used with a driver which didn't compile in those calls.

EPERM All ioctl calls except for **FPA_GET_DATAREGS** require root execution level.

EPIPE The FPA pipe is not clear.

FILES

/dev/fpa device file for both FPA and FPA+.

SEE ALSO

cc(1V), **close**(2), **ioctl**(2), **open**(2V), **fpa_download**(8), **fparel**(8), **fpaversion**(8)

DIAGNOSTICS

If hardware problems are detected then all processes with **/dev/fpa** open are killed, and future opens of **/dev/fpa** are disabled.

NAME

icmp – Internet Control Message Protocol

SYNOPSIS

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <netinet/ip_icmp.h>

s = socket(AF_INET, SOCK_RAW, proto);
```

DESCRIPTION

ICMP is the error and control message protocol used by the Internet protocol family. It is used by the kernel to handle and report errors in protocol processing. It may also be accessed through a "raw socket" for network monitoring and diagnostic functions. The protocol number for ICMP, used in the *proto* parameter to the socket call, can be obtained from **getprotobyname** (see **getprotoent**(3N)). ICMP sockets are connectionless, and are normally used with the *sendto* and *recvfrom* calls, though the **connect**(2) call may also be used to fix the destination for future packets (in which case the **read**(2V) or **recv**(2) and **write**(2V) or **send**(2) system calls may be used).

Outgoing packets automatically have an Internet Protocol (IP) header prepended to them. Incoming packets are provided to the holder of a raw socket with the IP header and options intact.

ICMP is an unreliable datagram protocol layered above IP. It is used internally by the protcol code for various purposes including routing, fault isolation, and congestion control. Receipt of an ICMP "redirect" message will add a new entry in the routing table, or modify an existing one. ICMP messages are routinely sent by the protocol code. Received ICMP messages may be reflected back to users of higher-level protocols such as TCP or UDP as error returns from system calls. A copy of all ICMP message received by the system is provided using the ICMP raw socket.

ERRORS

A socket operation may fail with one of the following errors returned:

EISCONN	when trying to establish a connection on a socket which already has one, or when trying to send a datagram with the destination address specified and the socket is already connected;
ENOTCONN	when trying to send a datagram, but no destination address is specified, and the socket hasn't been connected;
ENOBUFS	when the system runs out of memory for an internal data structure;
EADDRNOTAVAIL	when an attempt is made to create a socket with a network address for which no network interface exists.

SEE ALSO

connect(2), **read**(2V), **recv**(2), **send**(2), **write**(2V), **getprotoent**(3N), **inet**(4F), **ip**(4P), **routing**(4N)

Postel, Jon, *Internet Control Message Protocol — DARPA Internet Program Protocol Specification*, RFC 792, Network Information Center, SRI International, Menlo Park, Calif., September 1981. (Sun 800-1064-01)

BUGS

Replies to ICMP "echo" messages which are source routed are not sent back using inverted source routes, but rather go back through the normal routing mechanisms.

NAME

ie – Intel 10 Mb/s Ethernet interface

CONFIG — SUN-4 SYSTEM

device ie0 at obio ? csr 0x6000000 priority 3
device ie1 at vme24d16 ? csr 0xe88000 priority 3 vector ieintr 0x75

CONFIG — SUN-3x SYSTEM

device ie0 at obio ? csr 0x65000000 priority 3
device ie1 at vme24d16 ? csr 0xe88000 priority 3 vector ieintr 0x75

CONFIG — SUN-3 SYSTEM

device ie0 at obio ? csr 0xc0000 priority 3
device ie1 at vme24d16 ? csr 0xe88000 priority 3 vector ieintr 0x75
device ie0 at vme24d16 ? csr 0x31ff02 priority 3 vector ieintr 0x74

CONFIG — SUN-2 SYSTEM

device ie0 at obio 2 csr 0x7f0800 priority 3
device ie1 at vme24 ? csr 0xe88000 priority 3 vector ieintr 0x75
device ie0 at mbmem ? csr 0x88000 priority 3
device ie1 at mbmem ? csr 0x8c000 flags 2 priority 3

CONFIG — SUN386i SYSTEM

device ie0 at obmem ? csr 0xD0000000 irq 21 priority 3

DESCRIPTION

The **ie** interface provides access to a 10 Mb/s Ethernet network through the Intel 82586 controller chip. For a general description of network interfaces see **if**(4N).

The first Sun-4 and Sun-3x lines above specify CPU-board-resident Intel Ethernet interfaces; the second Sun-4 and Sun-3x lines specify Multibus Intel Ethernet interfaces for use with VME adapters.

In the Sun-3 lines above, the first line specifies the CPU-board-resident Intel Ethernet interface. The second line specifies a Multibus Intel Ethernet interface for use with a VME adapter. The third line specifies the Intel Ethernet interface present on a Sun-3 Eurocard board.

In the Sun-2 lines above, the first line specifies the CPU-board-resident Intel Ethernet interface on a Sun-2/50 or Sun-2/160 system. The second line specifies a Multibus Intel Ethernet controller for use with a VME adapter on these systems. The third line specifies the first Multibus Intel Ethernet controller for a Sun-2/120 or Sun-2/170 system. The fourth line specifies the second such controller for these systems.

The Sun386i line above specifies the CPU-board-resident Intel Ethernet interface.

SEE ALSO

if(4N)

DIAGNOSTICS

There are too many driver messages to list them all individually here. Some of the more common messages and their meanings follow.

ie%d: Ethernet jammed

Network activity has become so intense that sixteen successive transmission attempts failed, and the 82586 gave up on the current packet. Another possible cause of this message is a noise source somewhere in the network, such as a loose transceiver connection.

ie%d: no carrier

The 82586 has lost input to its carrier detect pin while trying to transmit a packet, causing the packet to be dropped. Possible causes include an open circuit somewhere in the network and noise on the carrier detect line from the transceiver.

ie%d: lost interrupt: resetting

The driver and 82586 chip have lost synchronization with each other. The driver recovers by resetting itself and the chip.

ie%d: iebark reset

The 82586 failed to complete a watchdog timeout command in the allotted time. The driver recovers by resetting itself and the chip.

ie%d: WARNING: requeueing

The driver has run out of resources while getting a packet ready to transmit. The packet is put back on the output queue for retransmission after more resources become available.

ie%d: panic: scb overwritten

The driver has discovered that memory that should remain unchanged after initialization has become corrupted. This error usually is a symptom of a bad 82586 chip.

ie%d: giant packet

Provided that all stations on the Ethernet are operating according to the Ethernet specification, this error "should never happen," since the driver allocates its receive buffers to be large enough to hold packets of the largest permitted size. The most likely cause of this message is that some other station on the net is transmitting packets whose lengths exceed the maximum permitted for Ethernet.

NAME

if – general properties of network interfaces

DESCRIPTION

Each network interface in a system corresponds to a path through which messages may be sent and received. A network interface usually has a hardware device associated with it, though certain interfaces such as the loopback interface, **lo**(4), do not.

At boot time, each interface with underlying hardware support makes itself known to the system during the autoconfiguration process. Once the interface has acquired its address, it is expected to install a routing table entry so that messages can be routed through it. Most interfaces require some part of their address specified with an SIOCSIFADDR IOCTL before they will allow traffic to flow through them. On interfaces where the network-link layer address mapping is static, only the network number is taken from the ioctl; the remainder is found in a hardware specific manner. On interfaces which provide dynamic network-link layer address mapping facilities (for example, 10Mb/s Ethernets using **arp**(4P),), the entire address specified in the ioctl is used.

The following **ioctl** calls may be used to manipulate network interfaces. Unless specified otherwise, the request takes an **ifreq** structure as its parameter. This structure has the form

```
struct  ifreq {
        char    ifr_name[16];           /* name of interface (e.g. "ec0") */
        union {
                struct  sockaddr ifru_addr;
                struct  sockaddr ifru_dstaddr;
                short   ifru_flags;
        } ifr_ifru;
#define ifr_addr                ifr_ifru.ifru_addr      /* address */
#define ifr_dstaddr             ifr_ifru.ifru_dstaddr   /* other end of p-to-p link */
#define ifr_flags               ifr_ifru.ifru_flags     /* flags */
};
```

SIOCSIFADDR Set interface address. Following the address assignment, the "initialization" routine for the interface is called.

SIOCGIFADDR Get interface address.

SIOCSIFDSTADDR Set point to point address for interface.

SIOCGIFDSTADDR Get point to point address for interface.

SIOCSIFFLAGS Set interface flags field. If the interface is marked down, any processes currently routing packets through the interface are notified.

SIOCGIFFLAGS Get interface flags.

SIOCGIFCONF Get interface configuration list. This request takes an **ifconf** structure (see below) as a value-result parameter. The **ifc_len** field should be initially set to the size of the buffer pointed to by **ifc_buf**. On return it will contain the length, in bytes, of the configuration list.

```
/*
* Structure used in SIOCGIFCONF request.
* Used to retrieve interface configuration
* for machine (useful for programs which
* must know all networks accessible).
*/
struct  ifconf {
        int     ifc_len;        /* size of associated buffer */
        union {
                caddr_t ifcu_buf;
                struct  ifreq *ifcu_req;
        } ifc_ifcu;
#define ifc_buf  ifc_ifcu.ifcu_buf /* buffer address */
#define ifc_req  ifc_ifcu.ifcu_req /* array of structures returned */
};
```

SIOCADDMULTI Enable a multicast address for the interface. A maximum of 64 multicast addresses may be enabled for any given interface.

SIOCDELMULTI Disable a previously set multicast address.

SIOCSPROMISC Toggle promiscuous mode.

SEE ALSO

arp(4P), **ec**(4S), **lo**(4)

NAME

lofs – loopback virtual file system

CONFIG

options LOFS

SYNOPSIS

#include <sys/mount.h>
mount(MOUNT_LOFS, virtual, flags, dir);

DESCRIPTION

The loopback file system device allows new, virtual file systems to be created, which provide access to existing files using alternate pathnames. Once the virtual file system is created, other file systems can be mounted within it without affecting the original file system. File systems that are subsequently mounted onto the original file system, however, *are* visible to the virtual file system, unless or until the corresponding mount point in the virtual file system is covered by a file system mounted there.

virtual is the mount point for the virtual file system. *dir* is the pathname of the existing file system. *flags* is either 0 or **M_RDONLY**. The **M_RDONLY** flag forces all accesses in the new name space to be read-only; without it, accesses are the same as for the underlying file system. All other **mount**(2) flags are preserved from the underlying file systems.

A loopback mount of '/' onto **/tmp/newroot** allows the entire file system hierarchy to appear as if it were duplicated under **/tmp/newroot**, including any file systems mounted from remote NFS servers. All files would then be accessible either from a pathname relative to '/', or from a pathname relative to **/tmp/newroot** until such time as a file system is mounted in **/tmp/newroot**, or any of its subdirectories.

Loopback mounts of '/' can be performed in conjunction with the **chroot**(2) system call, to provide a complete virtual file system to a process or family of processes.

Recursive traversal of loopback mount points is not allowed; after the loopback mount of **/tmp/newroot**, the file **/tmp/newroot/tmp/newroot** does not contain yet another file system hierarchy; rather, it appears just as **/tmp/newroot** did before the loopback mount was performed (say, as an empty directory).

The standard RC files perform first **4.2** mounts, then **nfs** mounts, during booting. On Sun386*i* systems, **lo** (loopback) mounts are performed just after **4.2** mounts. **/etc/fstab** files depending on alternate mount orders at boot time will fail to work as expected. Manual modification of **/etc/rc.local** will be needed to make such mount orders work.

WARNINGS

Loopback mounts must be used with care; the potential for confusing users and applications is enormous. A loopback mount entry in **/etc/fstab** must be placed after the mount points of both directories it depends on. This is most easily accomplished by making the loopback mount entry the last in **/etc/fstab**, though see **mount**(8) for further warnings.

SEE ALSO

chroot(2), **mount**(2), **fstab**(5), **mount**(8)

BUGS

Because only directories can be mounted or mounted on, the structure of a virtual file system can only be modified at directories.

NAME

mcp, alm – Sun MCP Multiprotocol Communications Processor/ALM-2 Asynchronous Line Multiplexer

CONFIG — SUN-3 SYSTEM

MCP

device mcp0 at vme32d32 ? csr 0x1000000 flags 0x1ffff priority 4 vector mcpintr 0x8b
device mcp1 at vme32d32 ? csr 0x1010000 flags 0x1ffff priority 4 vector mcpintr 0x8a
device mcp2 at vme32d32 ? csr 0x1020000 flags 0x1ffff priority 4 vector mcpintr 0x89
device mcp3 at vme32d32 ? csr 0x1030000 flags 0x1ffff priority 4 vector mcpintr 0x88

ALM-2

pseudo-device mcpa64

SYNOPSIS

#include <fcntl.h>
#include <sys/termios.h>
open("/dev/tty*xy*", mode);
open("/dev/ttyd*n*", mode);
open("/dev/cua*n*", mode);

DESCRIPTION (MCP)

The Sun MCP (Multiprotocol Communications Processor) supports up to four synchronous serial lines in conjunction with SunLink™ Multiple Communication Protocol products.

DESCRIPTION (ALM-2)

The Sun ALM-2 Asynchronous Line Multiplexer provides 16 asynchronous serial communication lines with modem control and one Centronics-compatible parallel printer port.

Each port supports those **termio**(4) device control functions specified by flags in the **c_cflag** word of the **termios** structure and by the **IGNBRK, IGNPAR, PARMRK,** or **INPCK** flags in the **c_iflag** word of the **termios** structure are performed by the **mcp** driver. All other **termio**(4) functions must be performed by STREAMS modules pushed atop the driver; when a device is opened, the **ldterm**(4M) and **ttcompat**(4M) STREAMS modules are automatically pushed on top of the stream, providing the standard **termio**(4) interface.

Bit *i* of **flags** may be specified to say that a line is not properly connected, and that the line *i* should be treated as hard-wired with carrier always present. Thus specifying **flags 0x0004** in the specification of **mcp0** would treat line **/dev/ttyh2** in this way.

Minor device numbers in the range 0 – 63 correspond directly to the normal tty lines and are named **/dev/tty***XY*, where *X* represents the physical board as one of the characters **h, i, j,** or **k**, and *Y* is the line number on the board as a single hexadecimal digit. (Thus the first line on the first board is **/dev/ttyh0**, and the sixteenth line on the third board is **/dev/ttyjf**.)

To allow a single tty line to be connected to a modem and used for both incoming and outgoing calls, a special feature, controlled by the minor device number, has been added. Minor device numbers in the range 128 – 191 correspond to the same physical lines as those above (that is, the same line as the minor device number minus 128).

A dial-in line has a minor device in the range 0 – 63 and is conventionally renamed **/dev/ttyd***n*, where *n* is a number indicating which dial-in line it is (so that **/dev/ttyd0** is the first dial-in line), and the dial-out line corresponding to that dial-in line has a minor device number 128 greater than the minor device number of the dial-in line and is conventionally named **/dev/cua***n*, where *n* is the number of the dial-in line.

The **/dev/cua***n* lines are special in that they can be opened even when there is no carrier on the line. Once a **/dev/cua***n* line is opened, the corresponding tty line cannot be opened until the **/dev/cua***n* line is closed; a blocking open will wait until the **/dev/cua***n* line is closed (which will drop Data Terminal Ready, after which Carrier Detect will usually drop as well) and carrier is detected again, and a non-blocking open will return an error. Also, if the **/dev/ttyd***n* line has been opened successfully (usually only when carrier is recognized on the modem) the corresponding **/dev/cua***n* line cannot be opened. This allows a modem to be

attached to e.g. **/dev/ttyd0** (renamed from **/dev/ttyh0**) and used for dialin (by enabling the line for login in **/etc/ttytab**) and also used for dialout (by **tip**(1C) or **uucp**(1C)) as **/dev/cua0** when no one is logged in on the line. Note: the bit in the **flags** word in the configuration file (see above) must be zero for this line, which enables hardware carrier detection.

IOCTLS

The standard set of **termio ioctl()** calls are supported by the ALM-2.

If the CRTSCTS flag in the **c_cflag** is set, output will be generated only if CTS is high; if CTS is low, output will be frozen. If the CRTSCTS flag is clear, the state of CTS has no effect. Breaks can be generated by the **TCSBRK, TIOCSBRK,** and **TIOCCBRK ioctl()** calls. The modem control lines **TIOCM_CAR, TIOCM_CTS, TIOCM_RTS,** and **TIOCM_DTR** are provided.

The input and output line speeds may be set to any of the speeds supported by **termio.** The speeds cannot be set independently; when the output speed is set, the input speed is set to the same speed.

ERRORS

An **open()** on a **/dev/tty*** or a **/dev/cu*** device will fail if:

ENXIO	The unit being opened does not exist.
EBUSY	The dial-out device is being opened and the dial-in device is already open, or the dial-in device is being opened with a no-delay open and the dial-out device is already open.
EBUSY	The unit has been marked as exclusive-use by another process with a **TIOCEXCL ioctl()** call.
EINTR	The open was interrupted by the delivery of a signal.

DESCRIPTION (PRINTER PORT)

The printer port is Centronics-compatible and is suitable for most common parallel printers. Devices attached to this interface are normally handled by the line printer spooling system, and should not be accessed directly by the user.

Minor device numbers in the range 64 – 67 access the printer port, and the recommended naming is **/dev/mcpp[0-3]**.

IOCTLS

Various control flags and status bits may be fetched and set on an MCP printer port. The following flags and status bits are supported; they are defined in **sundev/mcpcmd.h**:

MCPRIGNSLCT	0x02	set if interface ignoring SLCT– on open
MCPRDIAG	0x04	set if printer is in self-test mode
MCPRVMEINT	0x08	set if VME bus interrupts enabled
MCPRINTPE	0x10	print message when out of paper
MCPRINTSLCT	0x20	print message when printer offline
MCPRPE	0x40	set if device ready, cleared if device out of paper
MCPRSLCT	0x80	set if device online (Centronics SLCT asserted)

The flags **MCPRINTSLCT, MCPRINTPE,** and **MCPRDIAG** may be changed; the other bits are status bits and may not be changed.

The **ioctl()** calls supported by MCP printer ports are listed below.

MCPIOGPR	The argument is a pointer to an **unsigned char**. The printer flags and status bits are stored in the **unsigned char** pointed to by the argument.
MCPIOSPR	The argument is a pointer to an **unsigned char**. The printer flags are set from the **unsigned char** pointed to by the argument.

ERRORS

Normally, the interface only reports the status of the device when attempting an **open**(2V) call. An **open()** on a **/dev/mcpp*** device will fail if:

ENXIO The unit being opened does not exist.

EIO The device is offline or out of paper.

Bit 17 of the configuration **flags** may be specified to say that the interface should ignore Centronics SLCT– and RDY/PE– when attempting to open the device, but this is normally useful only for configuration and troubleshooting: if the SLCT– and RDY lines are not asserted during an actual data transfer (as with a **write**(2V) call), no data is transferred.

FILES

/dev/mcpp[0-3]	parallel printer port
/dev/tty[h-k][0-9a-f]	hardwired tty lines
/dev/ttyd[0-9a-f]	dialin tty lines
/dev/cua[0-9a-f]	dialout tty lines

SEE ALSO

tip(1C), **uucp**(1C), **mti**(4S), **termio**(4), **ldterm**(4M), **ttcompat**(4M), **zs**(4S)

DIAGNOSTICS

Most of these diagnostics "should never happen;" their occurrence usually indicates problems elsewhere in the system as well.

mcpa*n*: silo overflow.
More than *n* characters (*n* very large) have been received by the **mcp** hardware without being read by the software.

*****port *n* supports RS449 interface*****
Probably an incorrect jumper configuration. Consult the hardware manual.

mcp port *n* receive buffer error
The **mcp** encountered an error concerning the synchronous receive buffer.

Printer on mcpp*n* is out of paper
Printer on mcpp*n* paper ok
Printer on mcpp*n* is offline
Printer on mcpp*n* online
Assorted printer diagnostics, if enabled as discussed above.

BUGS

Note: pin 4 is used for hardware flow control on ALM–2 ports 0 through 3. These two pins should *not* be tied together on the ALM end.

NAME

mti – Systech MTI-800/1600 multi-terminal interface

CONFIG — SUN-3 SYSTEM

device mti0 at vme16d16 ? csr 0x620 flags 0xffff priority 4 vector mtiintr 0x88
device mti1 at vme16d16 ? csr 0x640 flags 0xffff priority 4 vector mtiintr 0x89
device mti2 at vme16d16 ? csr 0x660 flags 0xffff priority 4 vector mtiintr 0x8a
device mti3 at vme16d16 ? csr 0x680 flags 0xffff priority 4 vector mtiintr 0x8b

CONFIG — SUN-2 SYSTEM

device mti0 at mbio ? csr 0x620 flags 0xffff priority 4
device mti1 at mbio ? csr 0x640 flags 0xffff priority 4
device mti2 at mbio ? csr 0x660 flags 0xffff priority 4
device mti3 at mbio ? csr 0x680 flags 0xffff priority 4
device mti0 at vme16 ? csr 0x620 flags 0xffff priority 4 vector mtiintr 0x88
device mti1 at vme16 ? csr 0x640 flags 0xffff priority 4 vector mtiintr 0x89
device mti2 at vme16 ? csr 0x660 flags 0xffff priority 4 vector mtiintr 0x8a
device mti3 at vme16 ? csr 0x680 flags 0xffff priority 4 vector mtiintr 0x8b

SYNOPSIS

#include <fcntl.h>
#include <sys/termios.h>
open("/dev/tty*xy*", mode);
open("/dev/ttyd*n*", mode);
open("/dev/cua*n*", mode);

DESCRIPTION

The Systech MTI card provides 8 (MTI-800) or 16 (MTI-1600) serial communication lines with modem control. Each port supports those **termio**(4) device control functions specified by flags in the **c_cflag** word of the **termios** structure and by the **IGNBRK, IGNPAR, PARMRK,** or **INPCK** flags in the **c_iflag** word of the **termios** structure are performed by the **mti** driver. All other **termio**(4) functions must be performed by STREAMS modules pushed atop the driver; when a device is opened, the **ldterm**(4M) and **ttcompat**(4M) STREAMS modules are automatically pushed on top of the stream, providing the standard **termio**(4) interface.

Bit *i* of **flags** may be specified to say that a line is not properly connected, and that the line *i* should be treated as hard-wired with carrier always present. Thus specifying **flags 0x0004** in the specification of **mti0** would treat line **/dev/tty02** in this way.

Minor device numbers in the range 0 – 63 correspond directly to the normal tty lines and are named **/dev/tty***XY*, where *X* is the physical board number (0 – 3), and *Y* is the line number on the board as a single hexadecimal digit. (Thus the first line on the first board is **/dev/tty00**, and the sixteenth line on the third board is **/dev/tty2f**.)

To allow a single tty line to be connected to a modem and used for both incoming and outgoing calls, a special feature, controlled by the minor device number, has been added. Minor device numbers in the range 128 – 191 correspond to the same physical lines as those above (that is, the same line as the minor device number minus 128).

A dial-in line has a minor device in the range 0 – 63 and is conventionally renamed **/dev/ttyd***n*, where *n* is a number indicating which dial-in line it is (so that **/dev/ttyd0** is the first dial-in line), and the dial-out line corresponding to that dial-in line has a minor device number 128 greater than the minor device number of the dial-in line and is conventionally named **/dev/cua***n*, where *n* is the number of the dial-in line.

The **/dev/cua***n* lines are special in that they can be opened even when there is no carrier on the line. Once a **/dev/cua***n* line is opened, the corresponding tty line can not be opened until the **/dev/cua***n* line is closed; a blocking open will wait until the **/dev/cua***n* line is closed (which will drop Data Terminal Ready, after which Carrier Detect will usually drop as well) and carrier is detected again, and a non-blocking open will return an error. Also, if the **/dev/ttyd***n* line has been opened successfully (usually only when carrier is

recognized on the modem) the corresponding **/dev/cua*n*** line can not be opened. This allows a modem to be attached to e.g. **/dev/ttyd0** (renamed from **/dev/tty00**) and used for dialin (by enabling the line for login in **/etc/ttytab**) and also used for dialout (by **tip**(1C) or **uucp**(1C)) as **/dev/cua0** when no one is logged in on the line. Note: the bit in the **flags** word in the configuration file (see above) must be zero for this line, which enables hardware carrier detection.

WIRING

The Systech requires the CTS modem control signal to operate. If the device does not supply CTS then RTS should be jumpered to CTS at the distribution panel (short pins 4 to 5). Also, the CD (carrier detect) line does not work properly. When connecting a modem, the modem's CD line should be wired to DSR, which the software will treat as carrier detect.

IOCTLS

The standard set of **termio ioctl()** calls are supported by **mti**.

The state of the CRTSCTS flag in the **c_cflag** word has no effect; no output will be generated unless CTS is high. Breaks can be generated by the **TCSBRK**, **TIOCSBRK**, and **TIOCCBRK ioctl()** calls. The modem control lines **TIOCM_CAR**, **TIOCM_CTS**, **TIOCM_RTS**, and **TIOCM_DTR** are provided; however, as described above, the DSR line is treated as CD and the CD line is ignored.

The input and output line speeds may be set to any of the speeds supported by **termio**. The speeds cannot be set independently; when the output speed is set, the input speed is set to the same speed. The baud rates **B200** and **B38400** are not supported by the hardware; **B200** selects 2000 baud, and **B38400** selects 7200 baud.

ERRORS

An **open()** will fail if:

ENXIO	The unit being opened does not exist.
EBUSY	The dial-out device is being opened and the dial-in device is already open, or the dial-in device is being opened with a no-delay open and the dial-out device is already open.
EBUSY	The unit has been marked as exclusive-use by another process with a **TIOCEXCL ioctl()** call.
EINTR	The open was interrupted by the delivery of a signal.

FILES

/dev/tty[0-3][0-9a-f]	hardwired tty lines
/dev/ttyd[0-9a-f]	dialin tty lines
/dev/cua[0-9a-f]	dialout tty lines

SEE ALSO

tip(1C), **uucp**(1C), **mcp**(4S), **termio**(4), **ldterm**(4M), **ttcompat**(4M), **zs**(4S)

DIAGNOSTICS

Most of these diagnostics "should never happen" and their occurrence usually indicates problems elsewhere in the system.

mti*n*, *n*: silo overflow.
More than 512 characters have been received by the mti hardware without being read by the software. Extremely unlikely to occur.

mti*n*: read error code <*n*>. Probable hardware fault
The **mti** returned the indicated error code. See the MTI manual.

mti*n*: DMA output error.
The **mti** encountered an error while trying to do DMA output.

mti*n*: impossible response *n*.
The **mti** returned an error it could not understand.

NAME

mtio – general magnetic tape interface

SYNOPSIS

#include <sys/ioctl.h>
#include <sys/mtio.h>

DESCRIPTION

Both 1/2" and 1/4" magnetic tape drives share the same general interface regardless of the hardware involved. The remainder of this section discusses the common features of that interface.

The "cooked" magnetic tape device files read and write magnetic tape in 2048 byte blocks (the 2048 is actually **BLKDEV_IOSIZE** in **/usr/include/sys/param.h**). The name of such a device file might be **/dev/mt0**. The final component of the device name is the type of device the file refers to, and the unit number of that device.

These files are rewound when closed, except for "no-rewind" versions. The names of no-rewind device files use the letter **n** as the beginning of the final component. The no-rewind version of **/dev/mt0** would be **/dev/nmt0**. When a 1/2" tape file, opened for writing or just written, is closed, two tape marks are written. If the tape is not to be rewound, it is positioned with the head between the two tapemarks.

The files discussed above are useful for making taped files consistent with ordinary files. This interface requires that all blocks be 2048 bytes long, and does not permit special operations (such as spacing the tape forward or backward) to be performed. The "raw" interface is appropriate when reading or writing long records or using foreign tapes. Raw device files are indicated by the letter **r** before the device type in the device name: the raw version of **/dev/mt0** would be **/dev/rmt0**, and the raw version of **/dev/nmt0** would be **/dev/nrmt0**.

read(2V) or **write**(2V) calls read or write the next record on the tape. When the call is to write, the record has the same length as the given buffer. During a read call, the record size is passed back as the number of bytes read, provided it is no greater than the buffer size. In raw tape I/O, seeks are ignored (except **st**). When a tape mark is read, a zero byte count is returned; another read will fetch the first record of the next tape file. Two successive reads returning zero byte counts indicate the end of recorded media.

1/2" Reel Tape

Data bytes are recorded in parallel onto the 9–track tape. The number of bytes in a physical record varies between 1 to 65535 bytes. Files end with one file mark except for the last, which signals the end of recorded media with two file marks. Care should be taken when overwriting records; the erase head is just forward of the write head and any following records will also be erased.

The recording formats available (check specific tape drive) are 800 BPI, 1600 BPI, and 6250 BPI, and data compression. Actual storage capacity is a function of the recording format and the length of the tape reel. For example, using a 2400 foot tape, 20 MB can be stored using 800 BPI, 40 MB using 1600 BPI, 140 MB using 6250 BPI, or up to 700 MB using data compression.

1/4" Cartridge Tape

Data is recorded serially onto 1/4–inch cartridge tape. The number of bytes per record is determined by the physical record size of the device. The I/O request size must be a multiple of the physical record size of the device. For QIC–11, QIC–24, and QIC–150 tape drives the block size is 512 bytes.

The records are recorded on tracks in a serpentine motion. As one track is completed, the drive switches to the next and begins writing in the opposite direction, eliminating the wasted motion of rewinding. Each file, including the last, ends with one file mark.

Files may be written only at the beginning of the tape or after the last written file. This prevents corrupting data by overwriting files.

Storage capacity is based on the number of tracks the drive is capable of recording. For example, 4–track drives can only record 20 MB of data on a 450 foot tape; 9–track drives can record up to 45 MB of data on a tape of the same length.. QIC–11 is the only tape format available for 4–track tape drives. In contrast, 9–track tape drives can use either QIC–24 or QIC–11. Storage capacity is not appreciably affected by using

either format. QIC–24 is preferable to QIC–11 because it records a reference signal to mark the position of the first track on the tape, and each block has a unique block number.

The QIC–150 tape drives require DC–6150 (or equivalent) tape cartridges for writing. However, they can read other tape cartridges in QIC–11, QIC–24, QIC–120, or QIC–150 tape formats.

A number of additional ioctl operations are available on "raw" devices. The following definitions are from **/usr/include/sys/mtio.h**:

```
/*
 * Structures and definitions for mag tape io control commands
 */

/* structure for MTIOCTOP - mag tape op command */
struct  mtop    {
        short   mt_op;          /* operations defined below */
        daddr_t mt_count;       /* how many of them */
};

#define MTWEOF      0       /* write an end-of-file record */
#define MTFSF       1       /* forward space file */
#define MTBSF       2       /* backward space file */
#define MTFSR       3       /* forward space record */
#define MTBSR       4       /* backward space record */
#define MTREW       5       /* rewind */
#define MTOFFL      6       /* rewind and put the drive offline */
#define MTNOP       7       /* no operation, sets status only */
#define MTRETEN     8       /* retension the tape */
#define MTERASE     9       /* erase the entire tape */
#define MTEOM       10      /* position to end of media (SCSI only) */
#define MTBSFM      11      /* backward space file mark */

/* structure for MTIOCGET - mag tape get status command */
struct  mtget {
        short   mt_type;                /* type of magtape device */
/* the following two registers are grossly device dependent */
        short   mt_dsreg;               /* "drive status" register */
        short   mt_erreg;               /* "error" register */
/* optional error info. */
        daddr_t mt_resid;               /* residual count */
        daddr_t mt_fileno;              /* file number of current position */
        daddr_t mt_blkno;               /* block number of current position */
};

/*
 * Constants for mt_type byte
 */
#define MT_ISTS             0x01    /* vax: unibus ts-11 */
#define MT_ISHT             0x02    /* vax: massbus tu77, etc */
#define MT_ISTM             0x03    /* vax: unibus tm-11 */
#define MT_ISMT             0x04    /* vax: massbus tu78 */
#define MT_ISUT             0x05    /* vax: unibus gcr */
#define MT_ISCPC            0x06    /* sun: Multibus tapemaster */
#define MT_ISAR             0x07    /* sun: Multibus archive */
#define MT_ISSC             0x08    /* sun: SCSI archive */
```

```
#define MT_ISXY             0x09    /* sun: Xylogics 472 */

#define MT_ISSYSGEN11       0x10    /* sun: SCSI Sysgen, QIC-11 only */
#define MT_ISSYSGEN         0x11    /* sun: SCSI Sysgen QIC-24/11 */
#define MT_ISDEFAULT        0x12    /* sun: SCSI default CCS */
#define MT_ISCCS3           0x13    /* sun: SCSI generic (unknown) CCS */
#define MT_ISMT02           0x14    /* sun: SCSI Emulex MT02 */
#define MT_ISVIPER1         0x15    /* sun: SCSI Archive QIC-150 Viper */
#define MT_ISWANGTEK1       0x16    /* sun: SCSI Wangtek QIC-150 */
#define MT_ISCCS7           0x17    /* sun: SCSI generic (unknown) CCS */
#define MT_ISCCS8           0x18    /* sun: SCSI generic (unknown) CCS */
#define MT_ISCCS9           0x19    /* sun: SCSI generic (unknown) CCS */
#define MT_ISCCS11          0x1a    /* sun: SCSI generic (unknown) CCS */
#define MT_ISCCS12          0x1b    /* sun: SCSI generic (unknown) CCS */
#define MT_ISCCS13          0x1c    /* sun: SCSI generic (unknown) CCS */
#define MT_ISCCS14          0x1d    /* sun: SCSI generic (unknown) CCS */
#define MT_ISCCS15          0x1e    /* sun: SCSI generic (unknown) CCS */
#define MT_ISCCS16          0x1f    /* sun: SCSI generic (unknown) CCS */
#define MT_ISCDC            0x20    /* sun: SCSI CDC 1/2" cartridge */
#define MT_ISFUJI           0x21    /* sun: SCSI Fujitsu 1/2" cartridge */
#define MT_ISKENNEDY        0x22    /* sun: SCSI Kennedy 1/2" reel */
#define MT_ISHP             0x23    /* sun: SCSI HP 1/2" reel */
#define MT_ISCCS21          0x24    /* sun: SCSI generic (unknown) CCS */
#define MT_ISCCS22          0x25    /* sun: SCSI generic (unknown) CCS */
#define MT_ISCCS23          0x26    /* sun: SCSI generic (unknown) CCS */
#define MT_ISCCS24          0x27    /* sun: SCSI generic (unknown) CCS */
#define MT_ISEXABYTE        0x28    /* sun: SCSI Exabyte 8mm cartridge */
#define MT_ISCCS26          0x29    /* sun: SCSI generic (unknown) CCS */
#define MT_ISCCS27          0x2a    /* sun: SCSI generic (unknown) CCS */
#define MT_ISCCS28          0x2b    /* sun: SCSI generic (unknown) CCS */
#define MT_ISCCS29          0x2c    /* sun: SCSI generic (unknown) CCS */
#define MT_ISCCS30          0x2d    /* sun: SCSI generic (unknown) CCS */
#define MT_ISCCS31          0x2e    /* sun: SCSI generic (unknown) CCS */
#define MT_ISCCS32          0x2f    /* sun: SCSI generic (unknown) CCS */

/*
 * Device table structure and data for looking tape name from
 * tape id number.  Used by mt.c.
 */
struct mt_tape_info {
        short   t_type;         /* type of magtape device */
        char    *t_name;        /* printing name */
        char    *t_dsbits;      /* "drive status" register */
        char    *t_erbits;      /* "error" register */
};

#define MT_TAPE_INFO { \
{ MT_ISCPC,         "TapeMaster",       TMS_BITS, 0 }, \
{ MT_ISXY,          "Xylogics 472",     XTS_BITS, 0 }, \
{ MT_ISAR,          "Archive",          ARCH_CTRL_BITS, ARCH_BITS }, \
{ MT_ISSYSGEN11,    "Sysgen QIC-11",    0, 0 }, \
{ MT_ISSYSGEN,      "Sysgen",           0, 0 }, \
{ MT_ISMT02,        "Emulex MT-02",     0, 0 }, \
```

```
{ MT_ISVIPER1, "Archive QIC-150",      0, 0 }, \
{ MT_ISWANGTEK1,    "Wangtek QIC-150",     0, 0 }, \
{ MT_ISCDC,         "CDC",                 0, 0 }, \
{ MT_ISKENNEDY,     "Kennedy",             0, 0 }, \
{ MT_ISHP,          "HP-88780",            0, 0 }, \
{ MT_ISEXABYTE,     "Exabyte",             0, 0 }, \
{ 0 } \
}


/*
 * Constants for mt_type byte
 */

/*
 * Check if mt_type is one of the SCSI tape devices.
 */
#define MT_TYPE_SCSI(mt_type) \
        ((mt_type >= MT_ISSYSGEN11)  &&  (mt_type <= MT_ISCCS32))

/*
 * Older 1/4-inch cartridge tapes devices.
 * A blocking factor of 126 is recommended for compatibility.
 * A larger blocking factor may be used for improved streaming
 * performance.
 */
#define MT_TYPE_OLD_CARTRIDGE(mt_type) \
        ((mt_type == MT_ISSYSGEN11) || (mt_type == MT_ISSYSGEN) || \
         (mt_type == MT_ISAR))

/*
 * Current 1/4-inch cartridge tape devices.
 * A blocking factor of 40 (to 60) is recommended for
 * optimal streaming performance.
 */
#define MT_TYPE_NEW_CARTRIDGE(mt_type) \
        (mt_type >= MT_ISDEFAULT  &&  mt_type <= MT_ISCCS16)

/*
 * All 1/4-inch cartridge tape devices.
 * A blocking factor of 126 is recommended for compatibility
 * (during writes).  See above for specific recommendations for
 * reading.
 */
#define MT_TYPE_CARTRIDGE(mt_type) \
        ((mt_type >= MT_ISSYSGEN11  &&  mt_type <= MT_ISCCS16) || \
         (mt_type == MT_ISAR))

/*
 * All 1/2-inch reel tape devices.
 * A blocking factor of 20 is recommended for compatibility.
 */
#define MT_TYPE_REEL(mt_type) \
```

```
        ((mt_type >= MT_ISCDC  &&  mt_type <= MT_ISCCS32)  || \
         (mt_type >= MT_ISTS   &&  mt_type <= MT_ISCPC)    || \
         (mt_type == MT_ISXY))

/* mag tape io control commands */
#define  MTIOCTOP       _IOW(m, 1, struct mtop) /* do a mag tape op */
#define  MTIOCGET       _IOR(m, 2, struct mtget) /* get tape status */
#ifndef  KERNEL
#define  DEFTAPE        "/dev/rmt12"
#endif
```

SEE ALSO

mt(1), **tar**(1), **read**(2V), **write**(2V), **ar**(4S), **st**(4S), **tm**(4S), **xt**(4S)

BUGS

"Cooked" mode does not work for all magnetic tape devices (**st**, **tm**, and **xt** for example).

NAME

sd – driver for SCSI disk devices

CONFIG — SUN-3/4 SYSTEMS

controller si0 at vme24d16 ? csr 0x200000 priority 2 vector siintr 0x40
controller si0 at obio ? csr 0x140000 priority 2
disk sd0 at si0 drive 0 flags 0
disk sd1 at si0 drive 1 flags 0
disk sd2 at si0 drive 8 flags 0
disk sd3 at si0 drive 9 flags 0
disk sd4 at si0 drive 16 flags 0
disk sd6 at si0 drive 24 flags 0

controller sc0 at vme24d16 ? csr 0x200000 priority 2 vector scintr 0x40
disk sd0 at sc0 drive 0 flags 0
disk sd1 at sc0 drive 1 flags 0
disk sd2 at sc0 drive 8 flags 0
disk sd3 at sc0 drive 9 flags 0
disk sd4 at sc0 drive 16 flags 0
disk sd6 at sc0 drive 24 flags 0

The first two **controller** lines above specify the first and second SCSI host adapters for Sun-3, Sun-3x, and Sun-4 VME systems. The third **controller** line specifies the first and only SCSI host adapter on Sun-3/50 and Sun-3/60 systems.

The four lines following the **controller** specification lines define the available **disk** devices, sd0 – sd6.

The **flags** field is used to specify the SCSI device type to the host adapter. **flags** must be set to 0 to identify **disk** devices.

The **drive** value is calculated using the formula:

$8 * target + lun$

where *target* is the SCSI target, and *lun* is the SCSI logical unit number.

The next configuration block, following si0 and si1 above, describes the configuration for the older sc0 host adapter. It uses the same configuration description as the si0 host adapter.

CONFIG — SPARCsystem 330 and SUN-3/80 SYSTEMS

controller sm0 at obio ? csr 0xfa000000 priority 2
disk sd0 at sm0 drive 0 flags 0
disk sd1 at sm0 drive 1 flags 0
disk sd2 at sm0 drive 8 flags 0
disk sd3 at sm0 drive 9 flags 0
disk sd4 at sm0 drive 16 flags 0
disk sd6 at sm0 drive 24 flags 0

The SPARCsystem 330 and Sun-3/80 use an on-board SCSI host adapter, sm0. It follows the same rules as described above for the **Sun-3, Sun-3x, Sun-4** section.

CONFIG — SUN-4/110 SYSTEM

controller sw0 at obio 2 csr 0xa000000 priority 2
disk sd0 at sw0 drive 0 flags 0
disk sd1 at sw0 drive 1 flags 0
disk sd2 at sw0 drive 8 flags 0
disk sd3 at sw0 drive 9 flags 0
disk sd4 at sw0 drive 16 flags 0
disk sd6 at sw0 drive 24 flags 0

The Sun-4/110 uses an on-board SCSI host adapter, sw0. It follows the same rules as described above for the Sun-3, Sun-3x, Sun-4 section.

CONFIG — SUN-2 SYSTEM

controller sc0 at mbmem ? csr 0x80000 priority 2
controller sc1 at mbmem ? csr 0x84000 priority 2
controller sc0 at vme24 ? csr 0x200000 priority 2 vector scintr 0x40
disk sd0 at sc0 drive 0 flags 0
disk sd1 at sc0 drive 1 flags 0
disk sd2 at sc0 drive 8 flags 0
disk sd2 at sc1 drive 0 flags 0
disk sd3 at sc1 drive 1 flags 0

The **controller** lines above specify the first and second SCSI host adapters, sc0 and sc1, on Sun-2/120 and Sun-2/170 systems. The third **controller** line specifies the first and only host adapter on a Sun-2/160 system. It follows the same rules as described under the Sun-3, Sun-3x, Sun-4 configuration section above.

CONFIG — SUN-3/E SYSTEM

controller se0 at vme24d16 ? csr 0x300000 priority 2 vector se_intr 0x40
disk sd0 at se0 drive 0 flags 0
disk sd1 at se0 drive 1 flags 0
disk sd2 at se0 drive 8 flags 0
disk sd3 at se0 drive 9 flags 0

The Sun-3/E uses a VME-based SCSI host adapter, se0. It follows the same rules as described above for the Sun-3, Sun-3x, Sun-4 section.

CONFIG — Sun386i

controller wds0 at obmem ? csr 0xFB000000 dmachan 7 irq 16 priority 2
disk sd0 at wds0 drive 0 flags 0
disk sd1 at wds0 drive 8 flags 0
disk sd2 at wds0 drive 16 flags 0

The Sun-386*i* configuration follows the same rules described above under the Sun-3, Sun-3x, Sun-4 configuration section.

DESCRIPTION

Files with minor device numbers 0 through 7 refer to various portions of drive 0. The standard device names begin with "**sd**" followed by the drive number and then a letter a-h for partitions 0-7 respectively. The character **?** stands here for a drive number in the range 0-6.

The block-files access the disk using the system's normal buffering mechanism and are read and written without regard to physical disk records. There is also a "raw" interface that provides for direct transmission between the disk and the user's read or write buffer. A single read or write call usually results in one I/O operation; raw I/O is therefore considerably more efficient when many bytes are transmitted. The names of the raw files conventionally begin with an extra '**r**.'

I/O requests (for example **lseek**(2)) to the SCSI disk must have an offset that is a multiple of 512 bytes (DEV_BSIZE), or the driver returns an EINVAL error. If the transfer length is not a multiple of 512 bytes, the transfer count is rounded up by the driver.

Disk Support

This driver handles the Adaptec ACB-4000 disk controller for ST-506 drives, the Emulex MD21 disk controller for ESDI drives, and embedded, CCS-compatible SCSI disk drives.

On Sun386*i* systems, this driver supports the CDC Wren III half-height, and Wren IV full-height SCSI disk drives.

The type of disk drive is determined using the SCSI inquiry command and reading the volume label stored on block 0 of the drive. The volume label describes the disk geometry and partitioning; it must be present or the disk cannot be mounted by the system.

The **sd?a** partition is normally used for the root file system on a disk, the **sd?b** partition as a paging area (e.g. swap), and the **sd?c** partition for pack-pack copying. **sd?c** normally maps the entire disk and may also be used as the mount point for secondary disks in the system. The rest of the disk is normally the **sd?g** partition. For the primary disk, the user file system is located here.

FILES

/dev/sd[0-6][a-h]	block files
/dev/rsd[0-6][a-h]	raw files

SEE ALSO

dkio(4S), **directory**(3), **lseek**(2), **read**(2V), **write**(2V)

Product Specification for Wren IV SCSI Model 94171
Product Specification for Wren III SCSI Model 94161
Product Specification for Wren III SCSI Model 94211
Emulex MD21 Disk Controller Programmer Reference Manual
Adaptec ACB-4000 Disk Controller OEM Manual

DIAGNOSTICS

sd?: sdtimer: I/O request timeout
A tape I/O operation has taken too long to complete. A device or host adapter failure may have occurred.

sd?: sdtimer: can't abort request
The driver is unable to find the request in the disconnect que to notify the device driver that it has failed.

sd?: no space for inquiry data
sd?: no space for disk label
The driver was unable to get enough space for temporary storage. The driver is unable to open the disk device.

sd?: <%s>
The driver has found a SCSI disk device and opened it for the first time. The disk label is displayed to notify the user.

sd?: SCSI bus failure
A host adapter error was detected. The system may need to be rebooted.

sd?: single sector I/O failed
The driver attempted to recover from a transfer by writing each sector, one at a time, and failed. The disk needs to be reformatted to map out the new defect causing this error.

sd?: retry failed
sd?: rezero failed
A disk operation failed. The driver first tries to recover by retrying the command, if that fails, the driver rezeros the heads to cylinder 0 and repeats the retries. A failure of either the retry or rezero operations results in these warning messages; the error recovery operation continues until the retry count is exhausted. At that time a hard error is posted.

sd?: request sense failed
The driver was attempting to determine the cause of an I/O failure and was unable to get more information. This implies that the disk device may have failed.

sd?: warning, abs. block %d has failed %d times
The driver is warning the user that the specified block has failed repeatedly.

sd?: block %d needs mapping
sd?: reassigning defective abs. block %d
The specified block has failed repeatedly and may soon become an unrecoverable failure. If the driver does not map out the specified block automatically, it is recommend that the user

correct the problem.

sd?: reassign block failed

The driver attempted to map out a block having excessive soft errors and failed. The user needs to run format and repair the disk.

sd?%c: cmd how blk %d (rel. blk %d)

sense key(0x%x): %s, error code(0x%x): %s

An I/O operation (**cmd**), encountered an error condition at absolute block (**blk %d**), partition (**sd?%c:**), or relative block (**rel. block%d**). The error recovery operation (how) indicates whether it *retry*'ed, *restored*, or *failed*. The **sense key** and **error code** of the error are displayed for diagnostic purposes. The absolute **blk** of the the error is used for mapping out the defective block. The **rel. blk** is the block (sector) in error, relative to the beginning of the partition involved. This is useful for using **icheck**(8) to repair a damaged file structure on the disk.

NAME

sockio – ioctls that operate directly on sockets

SYNOPSIS

#include <sys/sockio.h>

DESCRIPTION

The IOCTL's listed in this manual page apply directly to sockets, independent of any underlying protocol. Note: the **setsockopt** system call (see **getsockopt**(2)) is the primary method for operating on sockets as such, rather than on the underlying protocol or network interface. **ioctl**s for a specific network interface or protocol are documented in the manual page for that interface or protocol.

SIOCSPGRP The argument is a pointer to an **int**. Set the process-group ID that will subsequently receive **SIGIO** or **SIGURG** signals for the socket referred to by the descriptor passed to **ioctl** to the value of that **int**.

SIOCGPGRP The argument is a pointer to an **int**. Set the value of that **int** to the process-group ID that is receiving **SIGIO** or **SIGURG** signals for the socket referred to by the descriptor passed to **ioctl**.

SIOCCATMARK The argument is a pointer to an **int**. Set the value of that **int** to 1 if the read pointer for the socket referred to by the descriptor passed to **ioctl** points to a mark in the data stream for an out-of-band message, and to 0 if it does not point to a mark.

SEE ALSO

ioctl(2), **getsockopt**(2), **filio**(4)

NAME

st – driver for SCSI tape devices

CONFIG — SUN-3, SUN-3x, SUN-4 SYSTEMS

controller si0 at vme24d16 ? csr 0x200000 priority 2 vector siintr 0x40
controller si1 at vme24d16 ? csr 0x204000 priority 2 vector siintr 0x41
controller si0 at obio ? csr 0x140000 priority 2
tape st0 at si0 drive 32 flags 1
tape st1 at si0 drive 40 flags 1
tape st2 at si1 drive 32 flags 1
tape st3 at si1 drive 40 flags 1

controller sc0 at vme24d16 ? csr 0x200000 priority 2 vector scintr 0x40
tape st0 at sc0 drive 32 flags 1
tape st1 at sc0 drive 40 flags 1

The first two **controller** lines above specify the first and second SCSI host adapters for Sun-3, Sun-3x, and Sun-4 VME systems. The third **controller** line specifies the first and only SCSI host adapter on Sun-3/50 and Sun-3/60 systems.

Following the **controller** specification lines are four lines which define the available **tape** devices, **st0–st3**. The first two **tape** devices, **st0** and **st1**, are on the first **controller**, **si0**. The next two **tape** devices, **st2** and **st3**, are on the second **controller**, **si1**.

The **flags** field is used to specify the SCSI device type to the host adapter. The **flags** field must be set to 1 to identify **tape** devices.

The **drive** value is calculated using the formula:

$8 * target + lun$

where *target* is the SCSI target, and *lun* is the SCSI logical unit number.

The next configuration block, following **si0** and **si1** above, describes the older **sc0** host adapter configuration. It follows the same configuration description as the **si0** host adapter.

CONFIG — SPARCsystem 330, SUN–3/80 SYSTEMS

controller sm0 at obio ? csr 0xfa000000 priority 2
tape st0 at sm0 drive 32 flags 1
tape st1 at sm0 drive 40 flags 1

The SPARCsystem 330 and Sun-3/80 use an on-board SCSI host adapter, **sm0**, which follows the rules described above in the Sun-3, Sun-3x, Sun-4 section.

CONFIG — SUN–4/110 SYSTEM

controller sw0 at obio 2 csr 0xa000000 priority 2
tape st0 at sw0 drive 32 flags 1
tape st1 at sw0 drive 40 flags 1

The Sun-4/110 uses an on-board SCSI host adapter, **sw0**, which follows the rules described above in the Sun-3, Sun-3x, Sun-4 section.

CONFIG — SUN–3/E SYSTEM

controller se0 at vme24d16 ? csr 0x300000 priority 2 vector se_intr 0x40
tape st0 at se0 drive 32 flags 1
tape st1 at se0 drive 40 flags 1

The Sun-3/E uses a VME-based SCSI host adapter, **se0**, which follows the rules described above for Sun-3, Sun-3x, Sun-4 systems.

CONFIG — Sun386i

controller wds0 at obmem ? csr 0xFB000000 dmachan 7 irq 16 priority 2
tape st0 at wds0 drive 32 flags 1

The Sun386*i* configuration follows the rules described above in the Sun-3, Sun-3x, Sun-4 configuration section.

CONFIG — SUN–2 SYSTEM

controller sc0 at mbmem ? csr 0x80000 priority 2
controller sc1 at mbmem ? csr 0x84000 priority 2
controller sc0 at vme24 ? csr 0x200000 priority 2 vector scintr 0x40
tape st0 at sc0 drive 32 flags 1
tape st1 at sc0 drive 40 flags 1
tape st0 at sc1 drive 32 flags 1
tape st1 at sc1 drive 40 flags 1

The **controller** lines above specify the first and second SCSI host adapters, **sc0** and **sc1**, on Sun-2/120 and Sun-2/170 systems. The third **controller** line specifies the only host adapter on a Sun-2/160. It follows the rules described in the Sun-3, Sun-3x, Sun-4 configuration section above.

DESCRIPTION

The **st** device driver is an interface to various SCSI tape devices. Supported 1/4–inch cartridge devices include the Archive Viper QIC–150 streaming tape drive, the Emulex MT–02 tape controller, and the Sysgen SC4000 tape controller. **st** provides a standard interface to these various devices, see **mtio**(4) for details.

The driver can be opened with either rewind on close (**/dev/rst***) or no rewind on close (**/dev/nrst***) options. A maximum of four tape formats per device are supported (see FILES below). The tape format is specified using the device name. The four rewind on close formats for **st0**, for example, are **/dev/rst0**, **/dev/rst8**, **/dev/rst16**, and **/dev/rst24**.

Read Operation

Fixed-length I/O tape devices require the number of bytes read or written to be a multiple of the physical record size. For example, 1/4–inch cartridge tape devices only read or write multiples of 512 bytes.

Fixed-length tape devices read or write multiple records if the blocking factor is greater than 64512 bytes (minphys limit). These multiple writes are limited to 64512 bytes. For example, if a write request is issued for 65536 bytes using a 1/4–inch cartridge tape, two writes are issued; the first for 64512 bytes and the second for 1024 bytes.

Tape devices, which support variable-length I/O operations, such as 1/2–inch reel tape, may read or write a range of 1 to 65535 bytes. If the record size exceeds 65535 bytes, the driver reads or writes multiple records to satisfy the request. These multiple records are limited to 65534 bytes. As an example, if a write request for 65540 bytes is issued using 1/2–inch reel tape, two records are written; one for 65534 bytes followed by one for 6 bytes.

If the driver is opened for reading in a different format than the tape is written in, the driver overrides the user selected format. For example, if a 1/4–inch cartridge tape is written in QIC-24 format and opened for reading in QIC–11, the driver will detect a read failure on the first read and automatically switch to QIC–24 to recover the data.

Note: If the **/dev/*st[0–3]** format is used, no indication is given that the driver has overridden the user selected format. Other formats issue a warning message to inform the user of an overridden format selection. Some devices automatically perform this function and do not require driver support (1/2–inch reel and QIC–150 tape drives for example).

If a file mark is encountered during reading, no error is reported but the number of bytes transferred is zero. The next read operation reads into the next file.

End of media is indicated by two successive zero transfer counts. No further reading should be performed past the end of recorded media.

If the read request size is 2048 bytes, the tape driver behaves as a disk device and honors seek positioning requests (see **lseek**(2)). If a file mark is crossed during a read operation, this function is disabled.

Write Operation

Writing is allowed at either the beginning of tape or after the last written file on the tape. Writing from the beginning of tape is performed in the user-specified format. The original tape format is used for appending onto previously written tapes. A warning message is issued if the driver has to override the user-specified format.

Care should be used when appending files onto 1/2–inch reel tape devices, since an extra file mark is appended after the last file to mark the end of recorded media. In other words, the last file on the tape ends with two file marks instead of one. This extra file mark must be overwritten to prevent the creation of a null file. To facilitate write append operations, a space to the end of recorded media ioctl is provided to eliminate this problem by having the driver perform the positioning operation.

If the end of tape is encountered during writing, no error is reported but the number of bytes transferred is zero and no further writing is allowed. Trailer records may be written by first writing a file mark followed by the trailer records. It is important that these trailer records be kept as short as possible to prevent data loss.

Close Operation

If data was written, a file mark is automatically written by the driver upon close. If the rewinding device name is used, the tape will be rewound after the file mark is written. If the user wrote a file mark prior to closing, then no file mark is written upon close. If a file positioning ioctl, like rewind, is issued after writing, a file mark is written before repositioning the tape.

Note: For 1/2–inch reel tape devices, two file marks are written to mark the end of recorded media before rewinding or performing a file positioning ioctl. If the user wrote a file mark before closing a 1/2–inch reel tape device, the driver will always write a file mark before closing to insure that the end of recorded media is marked properly.

If no data was written and the driver was opened for WRITE-ONLY access, a file mark is written thus creating a null file.

IOCTLS

The following ioctls are supported: forwardspace record, forwardspace file, backspace record, backspace file, backspace file mark, rewind, write file mark, offline, erase, retension, space to EOM, and get status.

The backspace file and forwardspace file tape operations are inverses. Thus, a forwardspace "–1" file is equivalent to a backspace "1" file. A backspace "0" file is the same as forwardspace "0" file; both position the tape device to the beginning of the current file.

Backspace file mark moves the tape backwards by file marks. The tape position will end on the beginning of tape side of the desired file mark. Devices which do not support this function, such as 1/4–inch cartridge tape, return an ENXIO error.

Backspace record and forwardspace record operations perform much like space file operations, except that they move by records instead of files. Variable-length I/O devices (1/2–inch reel, for example) space actual records; fixed-length I/O devices space physical records (blocks). 1/4–inch cartridge tape, for example, spaces 512 byte physical records. The status ioctl residue count contains the number of files or records not skipped. Record skipping does not go past a file mark; file skipping does not go past the end of recorded media.

Spacing to the end of recorded media positions the tape at a location just after the last file written on the tape. For 1/4–inch cartridge tape, this is after the last file mark on the tape. For 1/2–inch reel tape, this is just after the first file mark but before the second (and last) file mark on the tape. Additional files can then be appended onto the tape from that point.

The offline ioctl rewinds and, if appropriate, takes the device offline by unloading the tape. Tape must be inserted before the tape device can be used again.

The erase ioctl rewinds the tape, erases it completely, and returns to the beginning of tape.

The retension ioctl only applies to 1/4–inch cartridge tape devices. It is used to restore tape tension improving the tape's soft error rate after extensive start-stop operations or long-term storage. Devices which do not support this function, such as 1/2–inch reel tape, return an ENXIO error.

The get status ioctl call returns the drive id (mt_type), sense key error (mt_erreg), file number (mt_fileno), and record number (mt_blkno) of the last error. The residue count (mt_resid) is set to the number of bytes not transferred or files/records not spaced.

Note: The error status is reset by the get status ioctl call or the next read, write, or other ioctl operation. If no error has occurred (sense key is zero), the current file and record position are returned.

ERRORS

EACCES	The driver is opened for write access and the tape is write protected, or an attempt is made to write on a write protected tape. For writing with QIC–150 tape drives, this error is also reported if the wrong tape media is used for writing.
EBUSY	The tape device is already in use.
EIO	During opening, the tape device is not ready because either no tape is in the drive, or the drive is not on-line. Once open, this error is returned if the requested I/O transfer could not be completed.
EINVAL	The number of bytes read or written is not a multiple of the physical record size (fixed-length tape devices only).
ENXIO	During opening, the tape device does not exist. On ioctl functions, this indicates that the tape device does not support the ioctl function.

FILES

For QIC–150 tape devices (Archive Viper):

/dev/rst[0–3]	QIC–150 Format
/dev/rst[8–11]	QIC–150 Format
/dev/rst[16–20]	QIC–150 Format
/dev/rst[24–28]	QIC–150 Format
/dev/nrst[0–3]	non-rewinding QIC–150 Format
/dev/nrst[8–11]	non-rewinding QIC–150 Format
/dev/nrst[16–19]	non-rewinding QIC–150 Format
/dev/nrst[24–27]	non-rewinding QIC–150 Format

For QIC–24 tape devices (Emulex MT–02 and Sysgen SC4000):

/dev/rst[0–3]	QIC–11 Format
/dev/rst[8–11]	QIC–24 Format
/dev/rst[16–20]	QIC–24 Format
/dev/rst[24–28]	QIC–24 Format
/dev/nrst[0–3]	non-rewinding QIC–11 Format
/dev/nrst[8–11]	non-rewinding QIC–24 Format
/dev/nrst[16–19]	non-rewinding QIC–24 Format
/dev/nrst[24–27]	non-rewinding QIC–24 Format

Note: The QIC–24 format is preferred over QIC–11 for Sun-3, Sun-3x, Sun-4, and Sun386*i* systems.

SEE ALSO

mt(1), **tar**(1), **mtio**(4), **dump**(8), **restore**(8)

Archive Viper QIC–150 Tape Drive Product Specification
Emulex MT–02 Intelligent Tape Controller Product Specification
Sysgen SC4000 Intelligent Tape Controller Product Specification

DIAGNOSTICS

st?: sttimer: I/O request timeout
A tape I/O operation has taken too long to complete. A device or host adapter failure may have occurred.

st?: sttimer: can't abort request
The driver is unable to find the request in the disconnect que to notify the device driver that it has failed. A SCSI bus reset is issued to recover from this error.

st?: unknown SCSI device found
The SCSI device is not a tape device; it is some other type of SCSI device.

st?: warning, unknown tape drive found
The driver does not recognize the tape device. Only the default tape density is used; block size is set to the value specified by the tape drive.

st?: tape is write protected
The tape is write protected.

st?: wrong tape media for writing
For QIC–150 tape drives, this indicates that the user is trying to write on a DC–300XL (or equivalent) tape. Only DC–6150 (or equivalent) tapes can be used for writing.
Note: DC–6150 was formerly known as DC–600XTD.

st?: warning, rewinding tape
The driver is rewinding tape in order to set the tape format.

st?: warning, using alternate tape format
The driver is overriding the user-selected tape format and using the previously used format.

st?: warning, tape rewound
For Sysgen tape controllers, the tape may be rewound as a result of getting sense data.

st?: format change failed
The tape drive rejected the mode select command to change the tape format.

st?: file mark write failed
The driver was unable to write a file mark.

st?: warning, The tape may be wearing out or the head may need cleaning.
st?: read retries= %d, file= %d, block= %d
st?: write retries= %d, file= %d, block= %d
The number of allowable soft errors has been exceeded for this tape. Either the tape heads need cleaning or the tape is wearing out. If the tape is wearing out, continued usage of it is not recommended.

st?: illegal command
The SCSI command just issued was illegal. This message can result from issuing an inappropriate command, such as trying to write over previously written files on the tape. On foreign tape devices, this can also be caused by selecting the wrong tape format.

st?: error: sense key(0x%x): %s, error code(0x%x): %s
An error has occurred. The sense key message and error code are displayed for diagnostic purposes.

st?: stread: not modulo %d block size
st?: stwrite: not modulo %d block size
The read or write request size must be a multiple of the **%d** physical block size.

st?: file positioning error
st?: block positioning error
The driver was unable to position the tape to the desired file or block (record). This is probably caused by a damaged tape.

BUGS

Foreign tape devices which do not return a BUSY status during tape loading prevent user commands from being held until the device is ready. The user must delay issuing any tape operations until the tape device is ready. This is not a problem for Sun supplied tape devices.

Foreign tape devices which do not report a blank check error at the end of recorded media cause file positioning operations to fail. Some tape drives for example, mistakenly report media error instead of blank check error.

"Cooked" mode for read and write operations is not supported.

Systems using the older **sc0** host adapter or the Sysgen SC4000 tape controller, prevent disk I/O over the SCSI bus while the tape is in use (during a rewind for example). This problem is caused by the fact that they do not support disconnect/reconnect to free the SCSI bus. Newer tape devices, like the the Emulex MT–02, and host adapters, like **si0**, eliminate this problem.

Some older systems may not support the QIC–24 format, and may complain (or exhibit erratic behavior) when the user attempts to use this format.

NAME

streamio – STREAMS ioctl commands

SYNOPSIS

#include <stropts.h>
int ioctl (fildes, command, arg)
int fildes, command;

DESCRIPTION

STREAMS (see **intro**(2)) ioctl commands are a subset of **ioctl**(2) commands that perform a variety of control functions on STREAMS. The arguments *command* and *arg* are passed to the file designated by *fildes* and are interpreted by the *stream*head. Certain combinations of these arguments may be passed to a module or driver in the stream.

fildes is an open file descriptor that refers to a stream. *command* determines the control function to be performed as described below. *arg* represents additional information that is needed by this command. The type of *arg* depends upon the command, but it is generally an integer or a pointer to a *command*-specific data structure.

Since these STREAMS commands are a subset of *ioctl*, they are subject to the errors described there. In addition to those errors, the call will fail with *errno* set to EINVAL, without processing a control function, if the stream referenced by *fildes* is linked below a multiplexor, or if *command* is not a valid value for a *stream*.

Also, as described in *ioctl*, STREAMS modules and drivers can detect errors. In this case, the module or driver sends an error message to the *stream head* containing an error value. Subsequent system calls will fail with *errno* set to this value.

IOCTLS

The following **ioctl** commands, with error values indicated, are applicable to all STREAMS files:

I_PUSH Pushes the module whose name is pointed to by *arg* onto the top of the current stream, just below the *stream*head. It then calls the open routine of the newly-pushed module.

I_PUSH will fail if one of the following occurs:

EINVAL	The module name is invalid.
EFAULT	*arg* points outside the allocated address space.
ENXIO	The open routine of the new module failed.
ENXIO	A hangup is received on the stream referred to by *fildes*.

I_POP Removes the module just below the *stream head* of the stream pointed to by *fildes*. *arg* should be 0 in an **I_POP** request.

I_POP will fail if one of the following occurs:

EINVAL	No module is present on *stream*.
ENXIO	A hangup is received on the stream referred to by *fildes*.

I_LOOK Retrieves the name of the module just below the *stream head* of the stream pointed to by *fildes*, and places it in a NULL terminated character string pointed at by *arg*. The buffer pointed to by *arg* should be at least **FMNAMESZ+1** bytes long. An '**#include <sys/conf.h>**' declaration is required.

I_LOOK will fail if one of the following occurs:

EFAULT	*arg* points outside the allocated address space of the process.
EINVAL	No module is present on *stream*.

I_FLUSH This request flushes all input and/or output queues, depending on the value of *arg*. Legal *arg* values are:

FLUSHR	Flush read queues.
FLUSHW	Flush write queues.
FLUSHRW	Flush read and write queues.

I_FLUSH will fail if one of the following occurs:

EAGAIN	No buffers could be allocated for the flush message.
EINVAL	The value of *arg* is invalid.
ENXIO	A hangup is received on the stream referred to by *fildes*.

I_SETSIG Informs the *stream head* that the user wishes the kernel to issue the **SIGPOLL** signal (see **sigvec**(2)) when a particular event has occurred on the stream associated with *fildes*. **I_SETSIG** supports an asynchronous processing capability in STREAMS. The value of *arg* is a bitmask that specifies the events for which the user should be signaled. It is the bitwise-OR of any combination of the following constants:

S_INPUT	A non-priority message has arrived on a *stream head* read queue, and no other messages existed on that queue before this message was placed there. This is set even if the message is of zero length.
S_HIPRI	A priority message is present on the *stream head* read queue. This is set even if the message is of zero length.
S_OUTPUT	The write queue just below the *stream head* is no longer full. This notifies the user that there is room on the queue for sending (or writing) data downstream.
S_MSG	A STREAMS signal message that contains the **SIGPOLL** signal has reached the front of the *stream head* read queue.

A user process may choose to be signaled only of priority messages by setting the *arg* bitmask to the value **S_HIPRI**.

Processes that wish to receive **SIGPOLL** signals must explicitly register to receive them using **I_SETSIG**. If several processes register to receive this signal for the same event on the same *stream*, each process will be signaled when the event occurs.

If the value of *arg* is zero, the calling process will be unregistered and will not receive further **SIGPOLL** signals.

I_SETSIG will fail if one of the following occurs:

EINVAL	The value of *arg* is invalid or *arg* is zero and the process is not registered to receive the **SIGPOLL** signal.
EAGAIN	A data structure could not be allocated to store the signal request.

I_GETSIG Returns the events for which the calling process is currently registered to be sent a **SIGPOLL** signal. The events are returned as a bitmask pointed to by *arg*, where the events are those specified in the description of **I_SETSIG** above.

I_GETSIG will fail if one of the following occurs:

EINVAL	The process is not registered to receive the **SIGPOLL** signal.
EFAULT	*arg* points outside the allocated address space of the process.

I_FIND This request compares the names of all modules currently present in the stream to the name pointed to by *arg*, and returns 1 if the named module is present in the stream. It returns 0 if the named module is not present.

I_FIND will fail if one of the following occurs:

EFAULT	*arg* points outside the allocated address space of the process.
EINVAL	*arg* does not point to a valid module name.

I_PEEK This request allows a user to retrieve the information in the first message on the *stream head* read queue without taking the message off the queue. *arg* points to a *strpeek* structure which contains the following members:

```
struct strbuf   ctlbuf;
struct strbuf   databuf;
long            flags;
```

The *maxlen* field in the *ctlbuf* and *databuf strbuf* structures (see **getmsg**(2)) must be set to the number of bytes of control information and/or data information, respectively, to retrieve. If the user sets *flags* to **RS_HIPRI, I_PEEK** will only look for a priority message on the *stream head* read queue.

I_PEEK returns 1 if a message was retrieved, and returns 0 if no message was found on the *stream head* read queue, or if the **RS_HIPRI** flag was set in *flags* and a priority message was not present on the *stream head* read queue. It does not wait for a message to arrive. On return, *ctlbuf* specifies information in the control buffer, *databuf* specifies information in the data buffer, and *flags* contains the value 0 or **RS_HIPRI.**

I_PEEK will fail if one of the following occurs:

EFAULT	*arg* points, or the buffer area specified in *ctlbuf* or *databuf* is, outside the allocated address space of the process.

I_SRDOPT Sets the read mode using the value of the argument *arg*. Legal *arg* values are:

RNORM	Byte-stream mode, the default.
RMSGD	Message-discard mode.
RMSGN	Message-nondiscard mode.

Read modes are described in **read**(2V).

I_SRDOPT will fail if one of the following occurs:

EINVAL	*arg* is not one of the above legal values.

I_GRDOPT Returns the current read mode setting in an *int* pointed to by the argument *arg*. Read modes are described in **read**(2V).

I_GRDOPT will fail if one of the following occurs:

EFAULT	*arg* points outside the allocated address space of the process.

I_NREAD Counts the number of data bytes in data blocks in the first message on the *stream head* read queue, and places this value in the location pointed to by *arg*. The return value for the command is the number of messages on the *stream head* read queue. For example, if zero is returned in *arg*, but the **ioctl** return value is greater than zero, this indicates that a zero-length message is next on the queue.

I_NREAD will fail if one of the following occurs:

EFAULT *arg* points outside the allocated address space of the process.

I_FDINSERT creates a message from user specified buffer(s), adds information about another stream and sends the message downstream. The message contains a control part and an optional data part. The data and control parts to be sent are distinguished by placement in separate buffers, as described below.

arg points to a *strfdinsert* structure which contains the following members:

```
struct strbuf   ctlbuf;
struct strbuf   databuf;
long            flags;
int             fd;
int             offset;
```

The *len* field in the *ctlbuf strbuf* structure (see **putmsg**(2)) must be set to the size of a pointer plus the number of bytes of control information to be sent with the message. *fd* specifies the file descriptor of the other stream and *offset*, which must be word-aligned, specifies the number of bytes beyond the beginning of the control buffer where **I_FDINSERT** will store a pointer to the *fd* stream's driver read queue structure. The *len* field in the *databuf strbuf* structure must be set to the number of bytes of data information to be sent with the message or zero if no data part is to be sent.

flags specifies the type of message to be created. A non-priority message is created if *flags* is set to 0, and a priority message is created if *flags* is set to **RS_HIPRI**. For non-priority messages, **I_FDINSERT** will block if the stream write queue is full due to internal flow control conditions. For priority messages, **I_FDINSERT** does not block on this condition. For non-priority messages, **I_FDINSERT** does not block when the write queue is full and **O_NDELAY** is set. Instead, it fails and sets *errno* to EAGAIN.

I_FDINSERT also blocks, unless prevented by lack of internal resources, waiting for the availability of message blocks in the *stream*, regardless of priority or whether **O_NDELAY** has been specified. No partial message is sent.

I_FDINSERT will fail if one of the following occurs:

EAGAIN A non-priority message was specified, the **O_NDELAY** flag is set, and the stream write queue is full due to internal flow control conditions.

EAGAIN Buffers could not be allocated for the message that was to be created.

EFAULT *arg* points, or the buffer area specified in *ctlbuf* or *databuf* is, outside the allocated address space of the process.

EINVAL *fd* in the *strfdinsert* structure is not a valid, open stream file descriptor; the size of a pointer plus *offset* is greater than the *len* field for the buffer specified through *ctlptr*;

offset does not specify a properly-aligned location in the data buffer; an undefined value is pointed to by *flags*.

ENXIO — A hangup is received on the stream referred to by *fildes*.

ERANGE — The *len* field for the buffer specified through *databuf* does not fall within the range specified by the maximum and minimum packet sizes of the topmost stream module, or the *len* field for the buffer specified through *databuf* is larger than the maximum configured size of the data part of a message, or the *len* field for the buffer specified through *ctlbuf* is larger than the maximum configured size of the control part of a message.

I_STR Constructs an internal STREAMS ioctl message from the data pointed to by *arg*, and sends that message downstream.

This mechanism is provided to permit a process to specify timeouts and variable-sized amounts of data when sending an **ioctl** request to downstream modules and drivers. It allows information to be sent with the *ioctl*, and will return to the user any information sent upstream by the downstream recipient. **I_STR** blocks until the system responds with either a positive or negative acknowledgement message, or until the request "times out" after some period of time. If the request times out, it fails with *errno* set to ETIME.

At most, one **I_STR** can be active on a stream. Further **I_STR** calls will block until the active **I_STR** completes at the *stream head*. The default timeout interval for these requests is 15 seconds. The **O_NDELAY** (see **open**(2V)) flag has no effect on this call.

To send requests downstream, *arg* must point to a *strioctl* structure which contains the following members:

```
int     ic_cmd;     /* downstream command */
int     ic_timout;  /* ACK/NAK timeout */
int     ic_len;     /* length of data arg */
char    *ic_dp;     /* ptr to data arg */
```

ic_cmd is the internal ioctl command intended for a downstream module or driver and *ic_timout* is the number of seconds (−1 = infinite, 0 = use default, >0 = as specified) an **I_STR** request will wait for acknowledgement before timing out. *ic_len* is the number of bytes in the data argument and *ic_dp* is a pointer to the data argument. The *ic_len* field has two uses: on input, it contains the length of the data argument passed in, and on return from the command, it contains the number of bytes being returned to the user (the buffer pointed to by *ic_dp* should be large enough to contain the maximum amount of data that any module or the driver in the stream can return).

The *stream head* will convert the information pointed to by the *strioctl* structure to an internal ioctl command message and send it downstream.

I_STR will fail if one of the following occurs:

EAGAIN — Buffers could not be allocated for the **ioctl** message.

EFAULT — *arg* points, or the buffer area specified by *ic_dp* and *ic_len* (separately for data sent and data returned) is, outside the allocated address space of the process.

EINVAL — *ic_len* is less than 0 or *ic_len* is larger than the maximum

configured size of the data part of a message or *ic_timout* is less than −1.

ENXIO — A hangup is received on the stream referred to by *fildes*.

ETIME — A downstream **ioctl** timed out before acknowledgement was received.

An **I_STR** can also fail while waiting for an acknowledgement if a message indicating an error or a hangup is received at the *stream*head. In addition, an error code can be returned in the positive or negative acknowledgement message, in the event the **ioctl** command sent downstream fails. For these cases, **I_STR** will fail with *errno* set to the value in the message.

I_SENDFD Requests the stream associated with *fildes* to send a message, containing a file pointer, to the *stream head* at the other end of a stream pipe. The file pointer corresponds to *arg*, which must be an integer file descriptor.

I_SENDFD converts *arg* into the corresponding system file pointer. It allocates a message block and inserts the file pointer in the block. The user id and group id associated with the sending process are also inserted. This message is placed directly on the read queue (see **intro**(2)) of the *stream head* at the other end of the stream pipe to which it is connected.

I_SENDFD will fail if one of the following occurs:

EAGAIN — The sending stream is unable to allocate a message block to contain the file pointer.

EAGAIN — The read queue of the receiving *stream head* is full and cannot accept the message sent by **I_SENDFD**.

EBADF — *arg* is not a valid, open file descriptor.

EINVAL — *fildes* is not connected to a stream pipe.

ENXIO — A hangup is received on the stream referred to by *fildes*.

I_RECVFD Retrieves the file descriptor associated with the message sent by an **I_SENDFD** **ioctl** over a stream pipe. *arg* is a pointer to a data buffer large enough to hold an *strrecvfd* data structure containing the following members:

```
int fd;
unsigned short uid;
unsigned short gid;
char fill[8];
```

fd is an integer file descriptor. *uid* and *gid* are the user ID and group ID, respectively, of the sending stream.

If **O_NDELAY** is not set (see **open**(2V)), **I_RECVFD** will block until a message is present at the *stream*head. If **O_NDELAY** is set, **I_RECVFD** will fail with *errno* set to EAGAIN if no message is present at the *stream*head.

If the message at the *stream head* is a message sent by an **I_SENDFD**, a new user file descriptor is allocated for the file pointer contained in the message. The new file descriptor is placed in the *fd* field of the *strrecvfd* structure. The structure is copied into the user data buffer pointed to by *arg*.

I_RECVFD will fail if one of the following occurs:

EAGAIN — A message was not present at the *stream head* read queue, and the **O_NDELAY** flag is set.

EBADMSG	The message at the *stream head* read queue was not a message containing a passed file descriptor.
EFAULT	*arg* points outside the allocated address space of the process.
EMFILE	Too many descriptors are active.
ENXIO	A hangup is received on the stream referred to by *fildes*.

The following four commands are used for connecting and disconnecting multiplexed STREAMS configurations.

I_LINK Connects two streams, where *fildes* is the file descriptor of the stream connected to the multiplexing driver, and *arg* is the file descriptor of the stream connected to another driver. The stream designated by *arg* gets connected below the multiplexing driver. **I_LINK** causes the multiplexing driver to send an acknowledgement message to the *stream head* regarding the linking operation. This call returns a multiplexor ID number (an identifier used to disconnect the multiplexor, see **I_UNLINK**) on success, and a −1 on failure.

I_LINK will fail if one of the following occurs:

ENXIO	A hangup is received on the stream referred to by *fildes*.
ETIME	The **ioctl** timed out before an acknowledgement was received.
EAGAIN	Storage could not be allocated to perform the **I_LINK**.
EBADF	*arg* is not a valid, open file descriptor.
EINVAL	The stream referred to by *fildes* does not support multiplexing.
EINVAL	*arg* is not a stream, or is already linked under a multiplexor.
EINVAL	The specified link operation would cause a "cycle" in the resulting configuration; that is, if a given *stream head* is linked into a multiplexing configuration in more than one place.

An **I_LINK** can also fail while waiting for the multiplexing driver to acknowledge the link request, if a message indicating an error or a hangup is received at the *stream head* of *fildes*. In addition, an error code can be returned in the positive or negative acknowledgement message. For these cases, **I_LINK** will fail with *errno* set to the value in the message.

I_UNLINK Disconnects the two streams specified by *fildes* and *arg*. *fildes* is the file descriptor of the stream connected to the multiplexing driver. *arg* is the multiplexor ID number that was returned by the **ioctl I_LINK** command when a stream was linked below the multiplexing driver. If *arg* is −1, then all streams which were linked to *fildes* are disconnected. As in **I_LINK**, this command requires the multiplexing driver to acknowledge the unlink.

I_UNLINK will fail if one of the following occurs:

ENXIO	A hangup is received on the stream referred to by *fildes*.
ETIME	The **ioctl** timed out before an acknowledgement was received.
EAGAIN	Buffers could not be allocated for the acknowledgement message.

EINVAL The multiplexor ID number was invalid.

An **I_UNLINK** can also fail while waiting for the multiplexing driver to acknowledge the link request, if a message indicating an error or a hangup is received at the *stream head* of *fildes*. In addition, an error code can be returned in the positive or negative acknowledgement message. For these cases, **I_UNLINK** will fail with *errno* set to the value in the message.

SEE ALSO

close(2), **fcntl**(2V), **getmsg**(2), **intro**(2), **ioctl**(2), **open**(2V), **poll**(2), **putmsg**(2), **read**(2V), **sigvec**(2), **write**(2V)

STREAMS Programmer's Guide
STREAMS Primer

NAME

intro – introduction to games and demos

DESCRIPTION

This section describes available games and demos.

LIST OF GAMES AND DEMOS

Name	Appears on Page	Description
adventure	**adventure**(6)	an exploration game
arithmetic	**arithmetic**(6)	provide drill in number facts
backgammon	**backgammon**(6)	the game of backgammon
banner	**banner**(6)	print large banner on printer
battlestar	**battlestar**(6)	a tropical adventure game
bcd	**bcd**(6)	convert to antique media
bj	**bj**(6)	the game of black jack
boggle	**boggle**(6)	play the game of boggle
boggletool	**boggletool**(6)	play a game of boggle
bouncedemo	**graphics_demos**(6)	graphics demonstration programs
canfield	**canfield**(6)	Canfield solitaire card game
canfieldtool	**canfield**(6)	Canfield solitaire card game
canvas_demo	**sunview_demos**(6)	Window-System demonstration programs
cfscores	**canfield**(6)	Canfield solitaire card game
chase	**chase**(6)	try to escape to killer robots
chess	**chess**(6)	the game of chess
chesstool	**chesstool**(6)	window-based front-end to chess program
ching	**ching**(6)	the book of changes and other cookies
craps	**craps**(6)	the game of craps
cribbage	**cribbage**(6)	the card game cribbage
cursor_demo	**sunview_demos**(6)	Window-System demonstration programs
dialtest	**dialtest**(6)	a demonstration and testing program for SunDials
factor	**factor**(6)	factor a number, generate large primes
fish	**fish**(6)	play "Go Fish"
fortune	**fortune**(6)	print a random, hopefully interesting, adage
framedemo	**graphics_demos**(6)	graphics demonstration programs
gammontool	**gammontool**(6)	play a game of backgammon
graphics_demos	**graphics_demos**(6)	graphics demonstration programs
hack	**hack**(6)	replacement for rogue
hangman	**hangman**(6)	computer version of the game hangman
hunt	**hunt**(6)	a multiplayer multiterminal game
jumpdemo	**graphics_demos**(6)	graphics demonstration programs
life	**life**(6)	John Conway's game of life
mille	**mille**(6)	play Mille Bornes
monop	**monop**(6)	Monopoly game
moo	**moo**(6)	guessing game
number	**number**(6)	convert Arabic numerals to English
ppt	**bcd**(6)	convert to antique media
primes	**factor**(6)	factor a number, generate large primes
primes	**primes**(6)	print all primes larger than some given number
quiz	**quiz**(6)	test your knowledge
rain	**rain**(6)	animated raindrops display
random	**random**(6)	select lines randomly from a file
robots	**robots**(6)	fight off villainous robots
snake	**snake**(6)	display chase game
snscore	**snake**(6)	display chase game

spheresdemo	**graphics_demos**(6)	graphics demonstration programs
sunview_demos	**sunview_demos**(6)	Window-System demonstration programs
trek	**trek**(6)	trekkie game
worm	**worm**(6)	play the growing worm game
worms	**worms**(6)	animate worms on a display terminal
wump	**wump**(6)	the game of hunt-the-wumpus

NAME

cribbage – the card game cribbage

SYNOPSIS

/usr/games/cribbage [**–eqr**] *name* ...

DESCRIPTION

cribbage plays the card game cribbage, with **cribbage** playing one hand and the user the other. **cribbage** initially asks the user if the rules of the game are needed – if so, **cribbage** displays the appropriate section from *According to Hoyle* with **more**(1).

OPTIONS

–e Provide an explanation of the correct score when the player makes mistakes scoring his hand or crib. This is especially useful for beginning players.

–q Print a shorter form of all messages – this is only recommended for users who have played the game without specifying this option.

–r Instead of asking the player to cut the deck, **cribbage** will randomly cut the deck.

PLAYING CRIBBAGE

cribbage first asks the player whether he wishes to play a short game (''once around'', to 61) or a long game (''twice around'', to 121). A response of 's' results in a short game, any other response plays a long game.

At the start of the first game, **cribbage** asks the player to cut the deck to determine who gets the first crib. The user should respond with a number between 0 and 51, indicating how many cards down the deck is to be cut. The player who cuts the lower ranked card gets the first crib. If more than one game is played, the loser of the previous game gets the first crib in the current game.

For each hand, **cribbage** first prints the player's hand, whose crib it is, and then asks the player to discard two cards into the crib. The cards are prompted for one per line, and are typed as explained below.

After discarding, **cribbage** cuts the deck (if it is the player's crib) or asks the player to cut the deck (if it's its crib); in the latter case, the appropriate response is a number from 0 to 39 indicating how far down the remaining 40 cards are to be cut.

After cutting the deck, play starts with the non-dealer (the person who doesn't have the crib) leading the first card. Play continues, as per cribbage, until all cards are exhausted. **cribbage** keeps track of the scoring of all points and the total of the cards on the table.

After play, the hands are scored. **cribbage** requests the player to score his hand (and the crib, if it is his) by printing out the appropriate cards (and the cut card enclosed in brackets). Play continues until one player reaches the game limit (61 or 121).

A carriage return when a numeric input is expected is equivalent to typing the lowest legal value; when cutting the deck this is equivalent to choosing the top card.

SPECIFYING CARDS

Cards are specified as *rank* followed by *suit*. The *rank*s may be specified as one of **a, 2, 3, 4, 5, 6, 7, 8, 9, t, j, q,** and **k,** or alternatively, one of **ace, two, three, four, five, six, seven, eight, nine, ten, jack, queen,** and **king**. *Suit*s may be specified as **s, h, d,** and **c,** or alternatively as **spades, hearts, diamonds,** and **clubs**. A card may be specified as *rank suit*, or *rank* of *suit*. If the single letter *rank* and *suit* designations are used, the space separating the *suit* and *rank* may be left out. Also, if only one card of the desired *rank* is playable, typing the *rank* is sufficient. For example, if your hand was **2h, 4d, 5c, 6h, jc, kd** and you wanted to discard the king of diamonds, you could type any of **k, king, kd, k d, k of d, king d, king of d, k diamonds, k of diamonds, king diamonds,** or **king of diamonds,**

FILES

/usr/games/cribbage

SEE ALSO

more(1)

NAME

dialtest – a demonstration and testing program for SunDials

SYNOPSIS

/usr/demo/DIALS/dialtest

DESCRIPTION

dialtest displays a window with eight dials, corresponding to those on SunDials. To determine if the dialbox has been set up correctly, select the **Diagnostic** button on the panel. If the dialbox is correctly interfaced, **Dialbox OK** is displayed, and turning a dial on the box turn a dial on the screen. If **No Response from Dialbox** is displayed, repeat the dialbox install proceedure.

FILES

/usr/demo/DIALS/dialtest

NAME

factor, primes – factor a number, generate large primes

SYNOPSIS

/usr/games/factor [*number*]

/usr/games/primes [*number*]

DESCRIPTION

factor reads lines from its standard input. If it reads a positive number, **factor** will factor the number and print its prime factors, printing each one the proper number of times. **factor** exits when it reads zero, a negative number, or something other than a number. If a *number* is given, **factor** will factor the number, print its prime factors, and exit.

primes reads a number from the standard input and prints all primes larger than the given number and smaller than 2^{32} (about 4.3×10^{9}). If a *number* is given, **primes** will use that number rather than reading one from the standard input.

DIAGNOSTICS

Ouch. Input out of range or for garbage input.

NAME

intro – introduction to system maintenance and operation commands

DESCRIPTION

This section contains information related to system bootstrapping, operation and maintenance. It describes all the server processes and daemons that run on the system, as well as standalone (PROM monitor) programs.

Disk formatting and labeling is done by **format**(8S). Bootstrapping of the system is described in **boot**(8S) and **init**(8). The standard set of commands run by the system when it boots is described in **rc**(8). Related commands include those that check the consistency of file systems, **fsck**(8); those that mount and unmount file systems, **mount**(8); add swap devices, **swapon**(8); force completion of outstanding file system I/O, **sync**(2); shutdown or reboot a running system **shutdown**(8), **halt**(8), and **reboot**(8); and, set the time on a machine from the time on another machine **rdate**(8).

Creation of file systems is discussed in **mkfs**(8) and **newfs**(8). File system performance parameters can be adjusted with **tunefs**(8). File system backups and restores are described in **dump**(8) and **restore**(8).

Procedures for adding new users to a system are described in **adduser**(8), using **vipw**(8) to lock the password file during editing. **crash**(8S) which describes what happens when the system crashes, **savecore**(8) and **analyze**(8), which can be used to analyze system crash dumps. Occasionally useful as adjuncts to the **fsck**(8) file system repair program are **clri**(8), **dcheck**(8), **icheck**(8), and **ncheck**(8).

Configuring a new version of the kernel requires using the program **config**(8); major system bootstraps often require the use of **mkproto**(8). New devices are added to the **/dev** directory (once device drivers are configured into the system) using **makedev**(8) and **mknod**(8). The **installboot**(8S) command can be used to install freshly compiled programs. The **catman**(8) command preformats the on-line manual pages.

Resource accounting is enabled by the **accton** command, and summarized by **sa**(8). Login time accounting is performed by **ac**(8). Disk quotas are managed using **quot**(8), **quotacheck**(8), **quotaon**(8), and **repquota**(8).

A number of servers and daemon processes are described in this section. The **update**(8) daemon forces delayed file system I/O to occur and **cron**(8) runs periodic events (such as removing temporary files from the disk periodically). The **syslogd**(8) daemon maintains the system error log. The **init**(8) process is the initial process created when the system boots. It manages the reboot process and creates the initial login prompts on the various system terminals, using **getty**(8). The Internet super-server **inetd**(8C) invokes all other internet servers as needed. These servers include the remote shell servers **rshd**(8C) and **rexecd**(8C), the remote login server **rlogind**(8C), the FTP and TELNET daemons **ftpd**(8C), and **telnetd**(8C), the TFTP daemon **tftpd**(8C), and the mail arrival notification daemon **comsat**(8C). Other network daemons include the 'load average/who is logged in' daemon **rwhod**(8C), the routing daemon **routed**(8C), and the mail daemon **sendmail**(8).

If network protocols are being debugged, then the protocol debugging trace program **trpt**(8C) is often useful. Remote magnetic tape access is provided by **rsh** and **rmt**(8C). Remote line printer access is provided by **lpd**(8), and control over the various print queues is provided by **lpc**(8). Printer cost-accounting is done through **pac**(8).

Network host tables may be gotten from the ARPA NIC using **gettable**(8C) and converted to UNIX-system-usable format using **htable**(8).

RPC and NFS daemons

RPC and NFS daemons include:

portmap	used by RPC based services.
ypbind	used by the Yellow Pages to locate the Yellow Pages server.
biod	used by NFS clients to read ahead to, and write behind from, network file systems.
nfsd	the NFS server process that responds to NFS requests on NFS server machines.
ypserv	the Yellow Pages server, typically run on each NFS server.
rstatd	the server counterpart of the remote speedometer tools.

mountd the **mount** server that runs on NFS server machines and responds to requests by other machines to mount file systems.

rwalld used for broadcasting messages over the network.

LIST OF MAINTENANCE COMMANDS

Name	Appears on Page	Description
ac	**ac**(8)	login accounting
accton	**sa**(8)	system accounting
adbgen	**adbgen**(8)	generate adb script
adduser	**adduser**(8)	procedure for adding new users
arp	**arp**(8C)	address resolution display and control
audit	**audit**(8)	audit trail maintenance
audit_warn	**audit_warn**(8)	audit space low warning script
auditd	**auditd**(8)	audit daemon
automount	**automount**(8)	automatically mount NFS file systems
biod	**nfsd**(8)	NFS daemons
boot	**boot**(8S)	start the system kernel or a standalone program
bootparamd	**bootparamd**(8)	boot parameter server
captoinfo	**captoinfo**(8V)	convert a termcap description into a terminfo description
catman	**catman**(8)	create the cat files for the manual
chown	**chown**(8)	change owner
chroot	**chroot**(8)	change root directory for a command
client	**client**(8)	add/remove diskless systems
clri	**clri**(8)	clear i-node
comsat	**comsat**(8C)	biff server
config	**config**(8)	build system configuration files
crash	**crash**(8S)	what happens when the system crashes
cron	**cron**(8)	clock daemon
dcheck	**dcheck**(8)	file system directory consistency check
dkinfo	**dkinfo**(8)	report information about a disk's geometry and partitioning
dmesg	**dmesg**(8)	collect system diagnostic messages to form error log
dump	**dump**(8)	incremental file system dump
dumpfs	**dumpfs**(8)	dump file system information
edquota	**edquota**(8)	edit user quotas
eeprom	**eeprom**(8S)	Sun-3 EEPROM display and load utility
etherd	**etherd**(8C)	Ethernet statistics server
etherfind	**etherfind**(8C)	find packets on Ethernet
exportfs	**exportfs**(8)	export and unexport directories to NFS clients
fastboot	**fastboot**(8)	reboot/halt the system without checking the disks
fasthalt	**fastboot**(8)	reboot/halt the system without checking the disks
fingerd	**fingerd**(8C)	remote user information server
format	**format**(8S)	disk partitioning and maintenance utility
fpa_download	**fpa_download**(8)	download to the Floating Point Accelerator (FPA)
fparel	**fparel**(8)	Sun FPA online reliability tests
fpaversion	**fpaversion**(8)	print FPA version
fpurel	**fpurel**(8)	perform tests the Sun Floating Point Co-processor
fpuversion4	**fpuversion4**(8)	print the Sun-4 FPU version
fsck	**fsck**(8)	file system consistency check and interactive repair
fsirand	**fsirand**(8)	install random inode generation numbers
ftpd	**ftpd**(8C)	DARPA Internet File Transfer Protocol server
gettable	**gettable**(8C)	get DoD Internet format host table from a host
getty	**getty**(8)	set terminal mode
gpconfig	**gpconfig**(8)	initialize the Graphics Processor

grpck	**grpck**(8)	check group database entries
halt	**halt**(8)	stop the processor
htable	**htable**(8)	convert DoD Internet format host table
icheck	**icheck**(8)	file system storage consistency check
ifconfig	**ifconfig**(8C)	configure network interface parameters
inetd	**inetd**(8C)	Internet services daemon
infocmp	**infocmp**(8V)	compare or print out terminfo descriptions
init	**init**(8)	process control initialization
installboot	**boot**(8S)	start the system kernel or a standalone program
iostat	**iostat**(8)	report I/O statistics
ipallocd	**ipallocd**(8C)	Ethernet-to-IP address allocator
kadb	**kadb**(8S)	adb-like kernel and standalone-program debugger
keyenvoy	**keyenvoy**(8C)	talk to keyserver
keyserv	**keyserv**(8C)	server for storing public and private keys
kgmon	**kgmon**(8)	generate a dump of the operating system's profile buffers
ldconfig	**ldconfig**(8)	configure cache for **ld.so**
link	**link**(8)	exercise link and unlink system calls
lockd	**lockd**(8C)	network lock daemon
logintool	**logintool**(8)	graphic login interface
lpc	**lpc**(8)	line printer control program
lpd	**lpd**(8)	printer daemon
mailstats	**mailstats**(8)	print statistics collected by sendmail
MAKEDBM	**makedbm**(8)	make a Yellow Pages dbm file
MAKEDEV	**makedev**(8)	make system special files
makekey	**makekey**(8)	generate encryption key
mc68881version	**mc68881version**(8)	print the MC68881 mask number and approximate clock rate
mconnect	**mconnect**(8)	connect to SMTP mail server socket
mkfs	**mkfs**(8)	construct a file system
mknod	**mknod**(8)	build special file
mkproto	**mkproto**(8)	construct a prototype file system
modload	**modload**(8)	load a loadable module
modstat	**modstat**(8)	display status of loadable modules
modunload	**modunload**(8)	unload a loadable module
monitor	**monitor**(8S)	system ROM monitor
mount	**mount**(8)	mount and dismount filesystems
mountd	**mountd**(8C)	NFS mount request server
named	**named**(8C)	Internet domain name server
ncheck	**ncheck**(8)	generate names from i-numbers
ndbootd	**ndbootd**(8C)	ND boot block server
netconfig	**netconfig**(8C)	PNP boot service
netstat	**netstat**(8C)	show network status
newaliases	**newaliases**(8)	rebuild the data base for the mail aliases file
newfs	**newfs**(8)	construct a new file system
newkey	**newkey**(8)	create a new key in the publickey database
nfsd	**nfsd**(8)	NFS daemons
nfsstat	**nfsstat**(8C)	Network File System statistics
pac	**pac**(8)	printer/plotter accounting information
ping	**ping**(8C)	send ICMP ECHO_REQUEST packets to network hosts
pnp.s386	**pnpboot**(8C)	PNP diskless boot service
pnpboot	**pnpboot**(8C)	PNP diskless boot service
pnpd	**pnpd**(8C)	PNP daemon
portmap	**portmap**(8C)	DARPA port to RPC program number mapper
praudit	**praudit**(8)	print contents of an audit trail file

pstat	**pstat**(8)	print system facts
pwck	**pwck**(8)	check password database entries
pwdauthd	**pwdauthd**(8C)	server for authenticating passwords
quot	**quot**(8)	summarize file system ownership
quotacheck	**quotacheck**(8)	file system quota consistency checker
quotaoff	**quotaon**(8)	turn file system quotas on and off
quotaon	**quotaon**(8)	turn file system quotas on and off
rarpd	**rarpd**(8C)	DARPA Reverse Address Resolution Protocol service
rc.boot	**rc**(8)	command scripts for auto-reboot and daemons
rc.local	**rc**(8)	command scripts for auto-reboot and daemons
rc	**rc**(8)	command scripts for auto-reboot and daemons
rdate	**rdate**(8C)	set system date from a remote host
rdump	**dump**(8)	incremental file system dump
reboot	**reboot**(8)	restart the operating system
renice	**renice**(8)	alter priority of running processes
repquota	**repquota**(8)	summarize quotas for a file system
restore	**restore**(8)	incremental file system restore
rexd	**rexd**(8C)	RPC-based remote execution server
rexecd	**rexecd**(8C)	remote execution server
rlogind	**rlogind**(8C)	remote login server
rmail	**rmail**(8C)	handle remote mail received via uucp
rmt	**rmt**(8C)	remote magtape protocol module
route	**route**(8C)	manually manipulate the routing tables
routed	**routed**(8C)	network routing daemon
rpcinfo	**rpcinfo**(8C)	report RPC information
rquotad	**rquotad**(8C)	remote quota server
rrestore	**restore**(8)	incremental file system restore
rshd	**rshd**(8C)	remote shell server
rstatd	**rstatd**(8C)	kernel statistics server
rusersd	**rusersd**(8C)	network username server
rwalld	**rwalld**(8C)	network rwall server
rwhod	**rwhod**(8C)	system status server
sa	**sa**(8)	system accounting
savecore	**savecore**(8)	save a core dump of the operating system
sendmail	**sendmail**(8)	send mail over the internet
setup_client	**setup_client**(8)	create or remove a nfs client on a 4.0 server.
setup_exec	**setup_exec**(8)	install architecture-dependent executable files
showmount	**showmount**(8)	show all remote mounts
shutdown	**shutdown**(8)	close down the system at a given time
spray	**spray**(8C)	spray packets
sprayd	**sprayd**(8C)	spray server
statd	**statd**(8C)	network status monitor
sticky	**sticky**(8)	persistent text and append-only directories
sundiag	**sundiag**(8)	system diagnostics
suninstall	**suninstall**(8)	SunOS software installation program
sunupgrade	**sunupgrade**(8)	upgrade the Sun Operating System
swapon	**swapon**(8)	specify additional device for paging and swapping
sysdiag	**sysdiag**(8)	system diagnostics
syslogd	**syslogd**(8)	log system messages
talkd	**talkd**(8C)	server for talk program
telnetd	**telnetd**(8C)	DARPA TELNET protocol server
tftpd	**tftpd**(8C)	DARPA Trivial File Transfer Protocol server
tic	**tic**(8V)	terminfo compiler

timed	**timed**(8C)	DARPA Time server
tnamed	**tnamed**(8C)	DARPA Trivial name server
trpt	**trpt**(8C)	transliterate protocol trace
tunefs	**tunefs**(8)	tune up an existing file system
umount	**mount**(8)	mount and dismount filesystems
unconfigure	**unconfigure**(8)	reset the network configuration for a system
unlink	**link**(8)	exercise link and unlink system calls
update	**update**(8)	periodically update the super block
uuclean	**uuclean**(8C)	uucp spool directory clean-up
vipw	**vipw**(8)	edit the password file
vmstat	**vmstat**(8)	report virtual memory statistics
ypbind	**ypserv**(8)	Yellow Pages server and binder processes
ypinit	**ypinit**(8)	build and install Yellow Pages database
ypmake	**ypmake**(8)	rebuild Yellow Pages database
yppasswdd	**yppasswdd**(8C)	server for modifying Yellow Pages password file
yppoll	**yppoll**(8)	what version of a YP map is at a YP server host
yppush	**yppush**(8)	force propagation of a changed YP map
ypserv	**ypserv**(8)	Yellow Pages server and binder processes
ypset	**ypset**(8)	point ypbind at a particular server
ypupdated	**ypupdated**(8C)	server for changing yp information
ypwhich	**ypwhich**(8)	what machine is the YP server?
ypxfr	**ypxfr**(8)	transfer YP map from a YP server to here
zdump	**zdump**(8)	time zone dumper
zic	**zic**(8)	time zone compiler

NAME

ac – login accounting

SYNOPSIS

/usr/etc/ac [**–w** *wtmp*] [**–p**] [**–d**] [*username*] ...

DESCRIPTION

ac produces a printout giving connect time for each user who has logged in during the life of the current *wtmp* file. A total is also produced.

The accounting file **/var/adm/wtmp** is maintained by **init**(8) and **login**(1). Neither of these programs creates the file, so if it does not exist no connect-time accounting is done. To start accounting, it should be created with length 0. On the other hand if the file is left undisturbed it will grow without bound, so periodically any information desired should be collected and the file truncated.

OPTIONS

–w *wtmp* Specify an alternate *wtmp* file.

–p Print individual totals; without this option, only totals are printed.

–d Printout for each midnight to midnight period. Any *people* will limit the printout to only the specified login names. If no *wtmp* file is given, **/var/adm/wtmp** is used.

FILES

/var/adm/wtmp

SEE ALSO

login(1), **utmp**(5), **init**(8), **sa**(8)

NAME

cron – clock daemon

SYNOPSIS

/usr/etc/cron

DESCRIPTION

cron executes commands at specified dates and times. Regularly scheduled commands can be specified according to instructions found in **crontab** files in the directory **/var/spool/cron/crontabs**. Users can submit their own **crontab** files using the **crontab**(1) command. Commands that are to be executed only once may be submitted using the **at**(1) command.

cron only examines **crontab** files and **at** command files during process initialization and when a file changes using **crontab** or **at**. This reduces the overhead of checking for new or changed files at regularly scheduled intervals.

Since **cron** never exits, it should only be executed once. This is normally done by running **cron** from the initialization process through the file **/etc/rc**; see **init**(8). **/var/spool/cron/FIFO** is a FIFO file that **crontab** and **at** use to communicate with **cron**; it is also used as a lock file to prevent the execution of more than one **cron**.

FILES

/var/spool/cron	main cron directory
/var/spool/cron/FIFO	FIFO for sending messages to **cron**
/var/spool/cron/crontabs	directory containing **crontab** files

SEE ALSO

at(1), **crontab**(1), **sh**(1), **queuedefs**(5), **init**(8), **syslogd**(8)

DIAGNOSTICS

cron logs various errors to the system log daemon, **syslogd**(8), with a facility code of **cron**. The messages are listed here, grouped by severity level.

Err Severity

Can't create /var/spool/cron/FIFO: *reason*
: **cron** was unable to start up because it could not create **/var/spool/cron/FIFO**.

Can't access /var/spool/cron/FIFO: *reason*
: **cron** was unable to start up because it could not access **/var/spool/cron/FIFO**.

Can't open /var/spool/cron/FIFO: *reason*
: **cron** was unable to start up because it could not open **/var/spool/cron/FIFO**.

Can't start cron - another cron may be running (/var/spool/cron/FIFO exists)
: **cron** found that **/var/spool/cron/FIFO** already existed when it was started; this normally means that **cron** had already been started, but it may mean that an earlier **cron** terminated abnormally without removing **/var/spool/cron/FIFO**.

Can't stat /var/spool/cron/FIFO: *reason*
: **cron** could not get the status of **/var/spool/cron/FIFO**.

Can't change directory to *directory*:*reason*
: **cron** could not change to *directory*.

Can't read *directory*:*reason*
: **cron** could not read *directory*.

error reading message: *reason*
: An error occurred when **cron** tried to read a control message from **/var/spool/cron/FIFO**.

message received — bad format
A message was successfully read by **cron** from **/var/spool/cron/FIFO**, but the message was not of a form recognized by **cron**.

SIGTERM
received **cron** was told to terminate by having a SIGTERM signal sent to it.

cron could not unlink /var/spool/cron/FIFO: *reason*
cron was told to terminate, but it was unable to unlink **/var/spool/cron/FIFO** before it terminated.

********* CRON ABORTED **********
cron terminated, either due to an error or because it was told to.

Can't open queuedefs file *file*:*reason*
cron could not open a *queuedefs* file.

I/O error reading queuedefs file *file*:*reason*
An I/O error occurred while **cron** was reading a *queuedefs* file.

Using default queue definitions
An error occurred while trying to read a *queuedefs* file; the default queue definitions will be used.

Can't allocate *number* **bytes of space**
An internal error occurred in **cron** while trying to allocate memory.

Info Severity

queue **queue max run limit reached**
There were more jobs running or to be run in the queue *queue* than the maximum number specified. **cron** will wait until one of the currently-running jobs completes before starting to run a new one.

MAXRUN (25) procs reached
There were more than 25 jobs running or to be run by **cron**. **cron** will wait until one of the currently-running jobs completes before starting to run a new one.

***** cron started *****
cron started running.

> CMD: *command*
A **cron** job was started. *command* is the command to be run.

> *user pid queue time*
A **cron** job was started for user *user*, in queue *queue*, with process ID *pid*, at the date and time *time*.

< *user pid queue time status*
A **cron** job completed for user *user*, in queue *queue*, with process ID *pid*, at the date and time *time*. If the command terminated with a non-zero exit status or a signal, *status* indicates the exit status or signal.

Notice Severity

Can't fork
An attempt to **fork** (2) to run a new job failed; **cron** will attempt again after a 30-second delay.

Warning Severity

Can't stat queuedefs file *file*:*reason*
cron could not get the status of a *queuedefs* file in order to determine whether it has changed. **cron** will assume it has changed and will reread it.

NAME

dbconfig – initializes the dial box

SYOPNSIS

/usr/etc/dbconfig

DESCRIPTION

dbconfig opens the designated serial port and sets its baud, parity and transmission rates. It also removes all STREAMS modules already pushed upon it (such as **ttcompat**(4M) and **ldterm**(4M)) and pushes the dial box STREAMS module "db" onto the device. **db** then holds the stream open to maintain this configuration. If the device **/dev/dialbox** has not been created and linked to the serial port, **dbconfig** will fail.

FILES

/dev/dialbox

SEE ALSO

db(4M), **ldterm**(4M), **ttcompat**(4M)

NAME

dcheck – file system directory consistency check

SYNOPSIS

/usr/etc/dcheck [**–i** *numbers*] [*filesystem*]

DESCRIPTION

Note: **dcheck** has been superceded for normal consistency checking by **fsck**(8).

dcheck reads the directories in a file system and compares the link-count in each inode with the number of directory entries by which it is referenced. If the file system is not specified, **dcheck** checks a set of default file systems.

dcheck is fastest if the raw version of the special file is used, since the i-list is read in large chunks.

OPTIONS

–i *numbers*

numbers is a list of i-numbers; when one of those i-numbers turns up in a directory, the number, the i-number of the directory, and the name of the entry are reported.

FILES

Default file systems vary with installation.

SEE ALSO

fs(5), **fsck**(8), **clri**(8), **icheck**(8), **ncheck**(8)

DIAGNOSTICS

When a file turns up for which the link-count and the number of directory entries disagree, the relevant facts are reported. Allocated files which have 0 link-count and no entries are also listed. The only dangerous situation occurs when there are more entries than links; if entries are removed, so the link-count drops to 0, the remaining entries point to thin air. They should be removed. When there are more links than entries, or there is an allocated file with neither links nor entries, some disk space may be lost but the situation will not degenerate.

BUGS

Since **dcheck** is inherently two-pass in nature, extraneous diagnostics may be produced if applied to active file systems.

Inode numbers less than 2 are invalid.

NAME

fpa_download – download to the Floating Point Accelerator (FPA)

SYNOPSIS

fpa_download [**–d**] [**–r**] [**–v**] [**–u** *ufile*] [**–m** *mfile*] [**–c** *cfile*]

AVAILABILITY

fpa_download(8) applies to Sun3 and Sun3x systems equipped with either an FPA or FPA+.

DESCRIPTION

fpa_download writes microcode, map, and constants files to FPA and FPA+ boards. FPA requires a map file; FPA+ does not.

Root execution level is required to download (d,u,m and c options). **fpa_download** is called from **/etc/rc.local** when **/dev/fpa** exists.

Given no arguments, **fpa_download** prints whether an FPA, or FPA+ is installed.

OPTIONS

–d	Download microcode, constants, and map files. Enable default file names.
–r	Print microcode and constant revision.
–v	Verbose mode.
–u *ufile*	Download microcode from *ufile*.
–m *mfile*	Download map from *mfile* (FPA only).
–c *cfile*	Download constants from *cfile*.

FILES

/dev/fpa	device file for both FPA and FPA+.
/usr/etc/fpa/fpa_micro_bin	default microcode file (ufile) for FPA.
/usr/etc/fpa/fpa_constants	default constants file (cfile) for FPA
/usr/etc/fpa/fpa_micro_map	default map file (mfile) for FPA
/usr/etc/fpa/fpa_micro_bin+	default microcode file (ufile) for FPA+
/usr/etc/fpa/fpa_constants+	default constants file (cfile) for FPA+

SEE ALSO

fpa(4)

DIAGNOSTICS

The following diagnostics are printed when **fpa_download** encounters a serious error and asks the kernel to disable the FPA. This might occur if the microcode, map, or constants files are corrupted, or if there is an FPA or system hardware problem.

FPA Download Failed - FPA ioctl failed
An ioctl() on **/dev/fpa** failed, possibly due to a hung FPA pipe.

FPA Failed Download - FPA Bus Error
Received a SIGFPE.

FPA Failed Download - Upload mismatch
After each file is written to the FPA/FPA+, **fpa_download** uploads the contents of FPA memory and compares it with the source. They should always match.

NAME

fparel - Sun FPA online reliability tests

SYNOPSIS

fparel [−p*n*] [−v]

AVAILABILITY

Not available on Sun386i systems.

DESCRIPTION

fparel is a command to execute the Sun FPA online confidence and reliability test program. **fparel** tests about 90% of the functions of the FPA board, and tests all FPA contexts not in use by other processes. **fparel** runs without disturbing other processes that may be using the FPA. **fparel** can only be run by the super-user.

After a successful pass, **fparel** writes

time, date: Sun FPA Passed. The contexts tested are: 0, 1, ... 31

to the file **/var/adm/diaglog.**

If a pass fails, **fparel** writes

time, date: Sun FPA failed

along with the test name and context number that failed, to the file **/var/adm/diaglog. fparel** then broadcasts the message

time, date: Sun FPA failed, disabled, service required

to all users of the system. Next, **fparel** causes the kernel to disable the FPA. Once the kernel disables the FPA, the system must be rebooted to make it accessible.

The file **/etc/rc.local** should contain an entry to cause **fparel** to be invoked upon reboot to be sure that the FPA remains unaccessible in cases where rebooting doesn't correct the problem. See **rc**(8).

The **crontab**(5) file for root should contain an entry indicating that **cron**(8) is to run **fparel** daily, such as:

7 2 * * * /usr/etc/fpa/fparel

which causes **fparel** to run at seven minutes past two, every day. See **cron**(8) and **crontab**(5) for details.

OPTIONS

−p*n* Perform *n* passes. Default is *n*=1. −**p0** means perform 2147483647 passes.

−v Run in verbose mode with detailed test results to the standard output.

FILES

/var/adm/diaglog Log of **fparel** diagnostics.
/etc/rc.local
/var/spool/cron/crontabs/root
/usr/etc/fpa/* directory containing FPA microcode, data files, and loader

SEE ALSO

fpaversion(8), **crontab**(5), **cron**(8), **rc**(8)

NAME

fpaversion – print the Floating Point Accelerator (FPA) version

SYNOPSIS

fpaversion [**–hlqv**] [**–t** [**cdhimprstvxCIMS**]]

AVAILABILITY

fpaversion(8) applies to Sun3 and Sun3x systems equipped with either an FPA or FPA+.

DESCRIPTION

fpaversion performs various tests on the FPA or FPA+. When given no arguments, it prints the microcode version number and constants currently installed on **/dev/fpa**. **fpaversion** also performs a quick test to ensure proper operation and reports whether an FPA or an FPA+ is installed.

OPTIONS

–h Help. Print command-line summary.

–l Loop through tests infinitely.

–q Quiet output. Print out only error messages.

–v Verbose output.

–t Specify certain tests:

- **c** Command register format instructions.
- **d** Double precision format instructions.
- **h** Help. Print summary of test specifiers.
- **i** Imask register.
- **m** Mode register.
- **p** Simple pipe sequencing.
- **r** User registers for all contexts.
- **s** Single precision format instructions.
- **t** Status generation.
- **v** Print version number and date of microcode, and constants. Report whether an FPA or FPA+ is installed.
- **x** Extended format instructions.
- **C** Check checksum for microcode, mapping RAM, and constant RAM for the FPA. Check checksum for microcode RAM and constant RAM for the FPA+.
- **I** Allows interactive reads and writes to the FPA.
- **M** Command register format matrix instructions.
- **S** Shadow registers.

FILES

/dev/fpa	physical FPA device
/usr/etc/fpa/fpa_micro_bin	microcode binaries for the FPA
/usr/etc/fpa/fpa_micro_map	microcode map binaries for the FPA
/usr/etc/fpa/fpa_constants	microcode data file for the FPA
/usr/etc/fpa/fpa_micro_bin+	microcode binaries for the FPA+
/usr/etc/fpa/fpa_constants+	microcode data file for the FPA+
/usr/etc/fpa/fpa_download	microcode loader

SEE ALSO

fparel(8), **sysdiag**(8)

DIAGNOSTICS

If a test fails, its name, along with the actual and expected results will be printed.

NAME

fpurel – perform tests the Sun Floating Point Co-processor

SYNOPSIS

fpurel [**–rv**] [**–p**[*count*]]

DESCRIPTION

fpurel performs a series of functional and computational tests for the Sun Floating Point Co-processor to verify that it is operational and accurate. With no options, **fpurel** runs one pass silently in the foreground and only reports errors if any are found.

OPTIONS

–r Disable stop on error. Continue to run if errors are detected. The default is to display the error message and to stop testing when an error is detected.

–v Verbose. Display the name and results of each test on the console. The default is to run silently.

–p [*count*] Pass count. Specify the number of times to run the test suite. The default is to run one pass.

EXAMPLE

This example uses **fpurel** from the **/usr/diag** directory. If no errors are detected, then no information is displayed.

```
% /usr/diag/fpurel
```

NAME

fpuversion4 – print the Sun-4 FPU version

SYNOPSIS

/usr/etc/fpuversion4

AVAILABILITY

Sun-4 systems only.

DESCRIPTION

fpuversion4 reads the **%fsr** register to determine the FPU version installed on a Sun-4. The printed version field contains a value in the range 0-7; by SPARC convention 7 indicates that no FPU is installed, so floating-point instructions are always emulated in the kernel.

NAME

mount, umount – mount and unmount filesystems

SYNOPSIS

/usr/etc/mount [**–p**]
/usr/etc/mount –a[fnv] [**–t** *type*]
/usr/etc/mount [**–fnrv**] [**–t** *type*] [**–o** *options*] *filesystem directory*
/usr/etc/mount [**–vfn**] [**–o** *options*] *filesystem* | *directory*

/usr/etc/umount [**–t** *type*] [**–h** *host*]
/usr/etc/umount –a[v]
/usr/etc/umount [**–v**] *filesystem* | *directory* ...

DESCRIPTION

mount attaches a named *filesystem* to the filesystem hierarchy at the pathname location *directory*, which must already exist. If *directory* has any contents prior to the **mount** operation, these remain hidden until the *filesystem* is once again unmounted. If *filesystem* is of the form *host*:*pathname*, it is assumed to be an NFS filesystem (type **nfs**).

umount unmounts a currently mounted filesystem, which can be specified either as a *directory* or a *filesystem*.

mount and **umount** maintain a table of mounted filesystems in **/etc/mtab**, described in **fstab**(5). If invoked without an argument, **mount** displays the contents of this table. If invoked with either a *filesystem* or *directory* only, mount searches the file **/etc/fstab** for a matching entry, and mounts the filesystem indicated in that entry on the indicated directory.

MOUNT OPTIONS

–p Print the list of mounted filesystems in a format suitable for use in **/etc/fstab**.

–a All. Attempt to mount all the filesystems described in **/etc/fstab**. If a *type* argument is specified with **–t**, mount all filesystems of that type. Using **–a**, **mount** builds a dependency tree of mount points in **/etc/fstab**. **mount** will correctly mount these filesystems regardless of their order in **/etc/fstab** (except loopback mounts; see WARNINGS below).

–f Fake an **/etc/mtab** entry, but do not actually mount any filesystems.

–n Mount the filesystem without making an entry in **/etc/mtab**.

–v Verbose. Display a message indicating each filesystem being mounted.

–t *type* Specify a filesystem type. The accepted types are **4.2**, **nfs**, and **lo**. See **fstab**(5) for a description of **4.2**, and **nfs**; see **lofs**(4S) for a description of **lo**.

–r Mount the specified filesystem read-only, even if the entry in **/etc/fstab** specifies that it is to be mounted read-write.

Physically write-protected and magnetic-tape filesystems must be mounted read-only. Otherwise errors occur when the system attempts to update access times, even if no write operation is attempted.

–o *options*

Specify filesystem *options* —list of comma-separated words from the list below. Some options are valid for all filesystem types, while others apply to a specific type only.

options valid on *all* filesystems:

rw|**ro**	Read/write or read-only.
suid|**nosuid**	Setuid execution allowed or disallowed.
grpid	Create files with BSD semantics for the propagation of the group ID. Under this option, files inherit the GID of the directory in which they are created, regardless of the directory's set-GID bit.

noauto Do not mount this filesystem that is currently mounted read-only. If the filesystem is not currently mounted, an error results.

remount If the file system is currently mounted, and if the entry in **/etc/fstab** specifies that it is to be mounted read-write or **rw** was specified along with **remount**, remount the file system making it read-write. If the entry in **/etc/fstab** specifies that it is to be mounted read-only and **rw** was not specified, the file system is not remounted. If the file system is currently mounted read-write, specifying **ro** along with **remount** results in an error. If the file system is not currently mounted, an error results.

The default is '**rw,suid**'.

options specific to **4.2** filesystems:

quota|noquota Usage limits are enforced, or are not enforced. The default is **noquota**.

options specific to **nfs** (NFS) filesystems:

bg|fg If the first attempt fails, retry in the background, or, in the foreground.
retry=*n* The number of times to retry the mount operation.
rsize=*n* Set the read buffer size to *n* bytes.
wsize=*n* Set the write buffer size to *n* bytes.
timeo=*n* Set the NFS timeout to *n* tenths of a second.
retrans=*n* The number of NFS retransmissions.
port=*n* The server IP port number.
soft|hard Return an error if the server does not respond, or continue the retry request until the server responds.
intr Allow keyboard interrupts on hard mounts.
secure Use a more secure protocol for NFS transactions.
acregmin=*n* Hold cached attributes for at least *n* seconds after file modification.
acregmax=*n* Hold cached attributes for no more than *n* seconds after file modification.
acdirmin=*n* Hold cached attributes for at least *n* seconds after directory update.
acdirmax=*n* Hold cached attributes for no more than *n* seconds after directory update.
actimeo=*n* Set *min* and *max* times for regular files and directories to *n* seconds.
noac Suppress attribute caching.

Regular defaults are:

fg,retry=10000,timeo=7,retrans=3,port=NFS_PORT,hard,
acregmin=3,acregmax=60,acdirmin=30,acdirmax=60

actimeo has no default; it sets **acregmin, acregmax, acdirmin** and **acdirmax**

Defaults for **rsize** and **wsize** are set internally by the system kernel.

UMOUNT OPTIONS

–h *host* Unmount all filesystems listed in **/etc/mtab** that are remote-mounted from *host*.

–t *type* Unmount all filesystems listed in **/etc/mtab** that are of a given *type*.

–a Unmount all filesystems currently mounted (as listed in **/etc/mtab**).

–v Verbose. Display a message indicating each filesystem being unmounted.

NFS FILESYSTEMS

Background vs. Foreground

Filesystems mounted with the **bg** option indicate that **mount** is to retry in the background if the server's mount daemon (**mountd**(8C)) does not respond. **mount** retries the request up to the count specified in the **retry=***n* option. Once the filesystem is mounted, each NFS request made in the kernel waits **timeo=***n* tenths

of a second for a response. If no response arrives, the time-out is multiplied by **2** and the request is retransmitted. When the number of retransmissions has reached the number specified in the **retrans=*n*** option, a filesystem mounted with the **soft** option returns an error on the request; one mounted with the **hard** option prints a warning message and continues to retry the request.

Read-Write vs. Read-Only

Filesystems that are mounted **rw** (read-write) should use the **hard** option.

Interrupting Processes With Pending NFS Requests

The **intr** option allows keyboard interrupts to kill a process that is hung while waiting for a response on a hard-mounted filesystem.

Secure Filesystems

The **secure** option must be given if the server requires secure mounting for the filesystem.

File Attributes

The attribute cache retains file attributes on the client. Attributes for a file are assigned a time to be flushed. If the file is modified before the flush time, then the flush time is extended by the time since the last modification (under the assumption that files that changed recently are likely to change soon). There is a minimum and maximum flush time extension for regular files and for directories. Setting **actimeo=*n*** extends flush time by *n* seconds for both regular files and directories.

SYSTEM V COMPATIBILITY

System V File-Creation Semantics

Ordinarily, when a file is created its GID is set to the effective GID of the calling process. This behavior may be overridden on a per-directory basis, by setting the set-GID bit of the parent directory; in this case, the GID is set to the GID of the parent directory (see **open**(2V) and **mkdir**(2)). Files created on filesystems that are mounted with the **grpid** option will obey BSD semantics; that is, the GID is unconditionally inherited from that of the parent directory.

EXAMPLES

To mount a local disk:	**mount /dev/xy0g /usr**
To fake an entry for **nd** root:	**mount –ft 4.2 /dev/nd0 /**
To mount all 4.2 filesystems:	**mount –at 4.2**
To mount an NFS remote filesystem:	**mount –t nfs serv:/usr/src /usr/src**
To mount an NFS remote filesystem:	**mount serv:/usr/src /usr/src**
To hard mount an NFS remote filesystem:	**mount –o hard serv:/usr/src /usr/src**
To save current mount state:	**mount –p > /etc/fstab**

FILES

/etc/mtab	table of mounted filesystems
/etc/fstab	table of filesystems mounted at boot

WARNINGS

mount does not understand the mount order dependencies involved in loopback mounting. Loopback mounts may be dependent on two mounts having been previously performed, while **nfs** and **4.2** mounts are dependent only on a single previous mount. As a rule of thumb, place loopback mounts at the end of **/etc/fstab**file. See **lofs**(4S) for a complete description.

SEE ALSO

lofs(4S), **mountd**(8C), **nfsd**(8), **mkdir**(2), **mount**(2), **open**(2V), **unmount**(2), **fstab**(5), **mtab**(5),

BUGS

Mounting filesystems full of garbage crashes the system.

If the directory on which a filesystem is to be mounted is a symbolic link, the filesystem is mounted on *the directory to which the symbolic link refers*, rather than being mounted on top of the symbolic link itself.

NAME

mountd, rpc.mountd – NFS mount request server

SYNOPSIS

/usr/etc/rpc.mountd [**–n**]

AVAILABILITY

This program is available with the *Networking Tools and Programs* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

DESCRIPTION

mountd is an RPC server that answers file system mount requests. It reads the file **/etc/xtab**, described in **exports**(5), to determine which file systems are available for mounting by which machines. It also provides information as to what file systems are mounted by which clients. This information can be printed using the **showmount**(8) command.

The **mountd** daemon is normally invoked by **rc**(8).

OPTIONS

–n Do not check that the clients are root users. Though this option makes things slightly less secure, it does allow older versions (pre-3.0) of client NFS to work.

FILES

/etc/xtab

SEE ALSO

exports(5), **rc**(8), **showmount**(8)

NAME

setup_client – create or remove an NFS client

SYNOPSIS

/usr/etc/install/script/setup_client *op clientname yp_type swapsize rootpath swappath homepath execpath kvmpath arch*

DESCRIPTION

setup_client adds an NFS client to a server, or removes one. It can only be run by the super-user. It is also used by **suninstall**(8).

The *op* argument indicates which operation to perform, either **add** or **remove**, to indicate whether to add or remove a client. *clientname* is the hostname of the client. *yp_type* indicates the type of Yellow Pages server or service to provide to the client, if any; it can be one of **master**, **slave**, **client** or **none**. *swapsize* is the number of bytes reserved for client's swap file. *rootpath* is the pathname of the parent directory in which various client root directories reside; *rootpath/clientname* is the pathname of the client's root directory. *swappath* is the pathname of parent directory in which various client swap files reside; *swappath/clientname* is the pathname of the client's swap file. *homepath* is the pathname of the (parent) directory in which the various home directories are to reside; it is the pathname of the directory that the client is to mount as **/home**. *execpath* is the full pathname of the directory in which the executables for the kernel architecture specified by the *arch* argument. This is the directory that the client mounts as **/usr**. *kvmpath* is the full pathname of the directory in which the *kernel-specific* executables for the architecture specified by the *arch* argument reside. This is the directory that the client mounts as **/usr/kvm**. *arch* specifies the client's kernel architecture (for instance, **sun4**, **sun3x**...). See **arch**(1) for further explanation of "kernel architecture" and examples of valid kernel architectures. **setup_client** with no arguments displays a usage message that includes the proper *arch* argument for each supported kernel architecture.

USAGE

Before you add or remove a client, you must first make sure that the Internet and Ethernet addresses for *clientname* are listed in the YP hosts database (if the server is running the YP), or in the server's **/etc/hosts** and **/etc/ethers** databases, respectively (if otherwise). Then, run **setup_client** with the **add** or **remove** operation. When adding a client, you must then bootstrap that client machine.

A server must support a client's specific kernel architecture before it can mount that client. The executable directory for that client's kernel architecture must be present on the server. If this directory is absent, an error results.

setup_client updates the **/etc/bootparams** file. If the server is a YP master, it updates local YP database. It *does not* propagate the local update to other YP servers. To propagate the updates, use the following commands:

```
example# cd /var/yp
example# make
```

If the server is running YP but is not a YP master, **setup_client** issues a warning to indicate that the database is out of date.

When *arch* is given as **sun2**, **suninstall** issues a reminder to run the **/usr/etc/ndbootd** daemon for booting Sun-2 systems.

setup_client creates *swappath/clientname* with the *size*, (number of bytes) you specify. You can append one of **K** or **k** to indicate kilobytes, **M** or **m** to indicate megabytes, or **B** or **b** to indicate 512-byte blocks, to *size*. Otherwise, *size* is taken to indicate an exact byte count.

suninstall updates the **/etc/exports** file to allow root access to each client's root file system. It exports the client's swap and dump partitions only to the client.

Note: The system administrator should verify that the **/etc/exports** file contains correct information, and that file systems are exported to the correct users and groups. Refer to **exportfs**(8) for details on exporting file systems.

EXAMPLES

This example shows how to add a Sun-4 system NFS client to a server.

```
example# setup_client add frodo client 16M /exports/roots /exports/swaps /exports/dumps /home \
/exports/execs/sun4/4.0 sun4
```

To remove this client, you would merely substitute **remove** for **add** in the above example.

FILES

/etc/hosts	hosts database
/etc/ethers	database of hostnames and Ethernet addresses
/usr/etc/ndbootd	ND boot block server
/etc/bootparams	boot parameter database
/etc/exports	database of exported file systems

SEE ALSO

exportfs(8), **setup_exec**(8) **suninstall**(8)

Installing the SunOS

DIAGNOSTICS

incorrect number of arguments
Check number and order of the arguments.

must be run as root (super-user).
setup_client can only be used by root (super-user).

invalid operation type "*xx***".**
Valid operations are **add** and **remove**.

ATTENTION: *xxxxxxxx* –> boot.sun*?* not created.
(Sun-3 systems only.) A symbolic link can not be created because the boot file does not exist.

ATTENTION: *xxxxxxxx*.SUN*?* –> boot.sun*?* not created.
(Other than Sun-3 systems.) A symbolic link can not be created because the boot file does not exist.

ATTENTION: /usr/etc/ndbootd needs to be running on server before bringing up "*client*".
The Sun-2 system boot daemon must be running in order to bootstrap a Sun-2 system.

NAME

setup_exec – install architecture-dependent executables on a heterogeneous file server

SYNOPSIS

/usr/etc/install/setup_exec *arch execpath kvmpath* [**–o**]

DESCRIPTION

setup_exec installs architecture-dependent executables from either a local tape drive or a remote host. It is used to convert a standalone system or homogeneous file server to a heterogeneous file server. **setup_exec** is a forms-based utility that can be invoked directly, but it is also used by **suninstall**(8). It can only be invoked by the super-user.

The *arch* argument specifies the kernel architecture to install (for instance, **sun4, sun3x**...). See **arch**(1) for further explanation of "kernel architecture" and examples of valid kernel architectures. When run with no arguments, **setup_exec** displays a usage line that includes the proper format of the *arch* argument for each supported kernel architecture. *execpath* is the full pathname of the directory in which to install the executables. When **setup_exec** is done, the *execpath* directory is ready to mount as **/usr** by the heterogeneous server's NFS clients of the indicated *arch*. *kvmpath* is the full pathname of the directory in which to install the *kernel-specific* executables (**libkvm**, **pstat**(8), **ps**(1), etc.). When **setup_exec** is done, the *kvmpath* directory is ready to mount as **/usr/kvm** by the heterogeneous server's NFS clients of the indicated *arch*.

setup_exec also updates the **/etc/exports** file (see **exportfs**(8)) to export the executable directories it has installed. A system administrator should verify this file to make sure that the directory has been exported to the correct groups.

OPTIONS

–o Override the software duplication prevention mechanism. Allows the user to reload a software category that is already installed.

EXAMPLE

This example shows how to install a directory of executables for Sun-4 system clients.

```
example# setup_exec  sun4  /export/exec/kvm/sun4
```

FILES

/etc/hosts	hosts database
/etc/ethers	database of hostnames and Ethernet addresses
/etc/exports	database of exported file systems
/usr/etc/install/files/extractlist.*arch*	record of extracted categories for the indicated kernel architecture
/usr/etc/install/save	history of software installation

SEE ALSO

exportfs(8), **setup_client**(8), **suninstall**(8)

Installing the SunOS

DIAGNOSTICS

incorrect number of arguments
Check the number and the order of arguments.

invalid architecture type "*arch***".**
An unsupported value for *arch* was supplied.

invalid tape drive type "*drive***".**
Valid tape drive types are **local** and **remote**.

invalid tape type "*tape***".**
Valid tape types are **ar**, **st**, **mt**, and **xt**.

can't reach tapehost "*tapehost***".**
The IP address of *tapehost* is not in the hosts database, that is, the hosts YP database if the Yellow Pages are running, or the **/etc/hosts** file otherwise.

Load release tape *n*

Mount the release tape specified on the screen and type RETURN to continue.

NAME

showmount – show all remote mounts

SYNOPSIS

/usr/etc/showmount [**–ade**] [host]

AVAILABILITY

This program is available with the *Networking Tools and Programs* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

DESCRIPTION

showmount lists all the clients that have remotely mounted a filesystem from *host*. This information is maintained by the **mountd**(8C) server on *host*, and is saved across crashes in the file **/etc/rmtab**. The default value for *host* is the value returned by **hostname**(1).

OPTIONS

–a Print all remote mounts in the format

hostname:*directory*

where **hostname** is the name of the client, and **directory** is the root of the file system that has been mounted.

–d List directories that have been remotely mounted by clients.

–e Print the list of exported file systems.

FILES

/etc/rmtab

SEE ALSO

hostname(1), **exports**(5), **exports**(5), **mountd**(8C)

BUGS

If a client crashes, its entry will not be removed from the list until it reboots and executes '**umount –a**'.

NAME

sunupgrade – upgrade the Sun Operating System

SYNOPSIS

/usr/etc/sunupgrade [–l] [–d] [–n]

AVAILABILITY

This command is available on sun2, sun3 and sun4 application architectures running SunOS version 4.0 only. Refer to *Installing the SunOS 4.0.3* for more information.

DESCRIPTION

sunupgrade is an interactive utility that is used to upgrade the Sun Operating System (SunOS) to a higher revision level on sun2, sun3, and sun4 application architectures. The current SunOS level must be at least SunOS 4.0.

sunupgrade lets you upgrade any system configuration. The following are the valid configuration types:

- Standalone
- Homogeneous server
- Heterogeneous server
- Dataless clients
- Diskless clients

Both local and remote installation modes are supported.

sunupgrade overlays the newer executable files on top of existing ones. For files in and if the new file names and old file names conflict and the files are not identical in content, then the new file is installed with a trailing suffix that is the release name. The differences between the old and the new versions of the files must be resolved by the user. All such files are logged in the log file **/usr/etc/upgrade/save/special_files** for servers, standalone systems, or in **/usr/etc/upgrade/save/***clientname***.special_files** for diskless clients. For dataless clients, they are shared under **/home/upgrade/special_files .**

After **sunupgrade** completes execution you must come up in single-user mode, inspect all special files and propagate your older administrative files to the newer ones and rename them without the suffixes.

OPTIONS

–l Create log of all files extracted and overlaid. Performance will deteriorate slightly. Log files are saved in **/usr/etc/upgrade/save**. Special files are called

–d Work in debugging mode. Not recommended for normal operation.

–n Switch off "no-rewind" operation. The no-rewind operation available only on systems running SunOS 4.0.2 or 4.0.3.

FILES

/usr/etc/upgrade/EXCLUDELIST
/usr/etc/upgrade/README
/usr/etc/upgrade/checksums
/usr/etc/upgrade/chk_ok
/usr/etc/upgrade/chk_release
/usr/etc/upgrade/chkextract
/usr/etc/upgrade/config_host
/usr/etc/upgrade/extract
/usr/etc/upgrade/extract_client
/usr/etc/upgrade/extract_clntroot
/usr/etc/upgrade/extract_stand
/usr/etc/upgrade/include
/usr/etc/upgrade/includefile
/usr/etc/upgrade/get_arch
/usr/etc/upgrade/get_clientinfo
/usr/etc/upgrade/get_machtype

/usr/etc/upgrade/get_tapeinfo
/usr/etc/upgrade/get_toc
/usr/etc/upgrade/get_upgradeinfo
/usr/etc/upgrade/mop_up
/usr/etc/upgrade/mount_ufs
/usr/etc/upgrade/mount_usr
/usr/etc/upgrade/start_log
/usr/etc/upgrade/setup_kvm
/usr/etc/upgrade/small_kernel_files
/usr/etc/upgrade/sun2_cp_share
/usr/etc/upgrade/sun2_ln_exec
/usr/etc/upgrade/sunupgrade
/usr/etc/upgrade/tar_clntroot
/usr/etc/upgrade/verify_clntpart
/usr/etc/upgrade/xdrtoc

SEE ALSO

suninstall(8)

Installing the SunOS 4.0.3

NAME

syslogd – log system messages

SYNOPSIS

/usr/etc/syslogd [**–d**] [**–f***configfile*] [**–m** *interval*]

DESCRIPTION

syslogd reads and forwards system messages to the appropriate log files and/or users, depending upon the priority of a message and the system facility from which it originates. The configuration file **/etc/syslog.conf** (see **syslog.conf**(5)) controls where messages are forwarded. **syslogd** logs a mark (timestamp) message every *interval* minutes (default 20) at priority **LOG_INFO** to the facility whose name is given as **mark** in the **syslog.conf** file.

A system message consists of a single line of text, which may be prefixed with a priority code number enclosed in angle-brackets (**<>**); priorities are defined in **sys/syslog.h**.

syslogd reads from the **AF_UNIX** address family socket **/dev/log**, from an Internet address family socket specified in **/etc/services**, and from the special device **/dev/klog** (for kernel messages).

syslogd reads the configuration file when it starts up, and again whenever it receives a **HUP** signal, at which time it also closes all files it has open, re-reads its configuration file, and then opens only the log files that are listed in that file. **syslogd** exits when it receives a **TERM** signal.

As it starts up, **syslogd** creates the file **/etc/syslog.pid**, if possible, containing its process ID (PID).

Sun386i DESCRIPTION

syslogd translates messages using the databases specified on an optional line in the **syslog.conf** as indicated with a **translate** entry.

The format of these databases is described in **translate**(5).

OPTIONS

–d	Turn on debugging.
–f*configfile*	Specify an alternate configuration file.
–m *interval*	Specify an interval, in minutes, between mark messages.

FILES

/etc/syslog.conf	configuration file
/etc/syslog.pid	process ID
/dev/log	AF_UNIX address family datagram log socket
/dev/klog	kernel log device
/etc/services	network services database

SEE ALSO

logger(1), **syslog**(3), **syslog.conf**(5), **translate**(5)

NAME

sundiag – system diagnostics

SYNOPSIS

/usr/diag/sundiag/sundiag [**–Cmptv**] [**–h** *remotehost*] [**–o** *saved_options_file*]
[*generic_tool_arguments*]

AVAILABILITY

This program is available with the *User Diagnostics* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

DESCRIPTION

sundiag is a diagnostic facility that tests the functionality of the operating system and reports its findings. It can also be used to report the hardware configuration as detected by the system.

You must be root to use **sundiag**.

When run on the console monitor, **sundiag** takes full advantage of the *SunView 1* windowing environment. There are four subwindows: a control panel for displaying the discovered hardware configuration and manipulating of the numerous test parameters and options, a test status panel which shows the test results, a console window which is used to display messages, and a performance monitor. There are also some popup frames, including a text frame for viewing **sundiag** and system log files.

When executed from a terminal, **sundiag** uses **curses**(3X) to simulate each subwindow on the screen.

sundiag consists of **sundiag**, along with several binary modules and executable files containing the actual test code, all of which reside in **/usr/diag/sundiag**.

OPTIONS

–C Redirect the console output from any existing console window to the **sundiag** console subwindow. This option does not work when running on a terminal or tty emulator.

–m Create a device file for all devices found during the kernel probe. **sundiag** uses the same major/minor device numbers and permissions declared in **/dev/MAKEDEV**.

–p Ignore the kernel probe for devices, especially when running user-defined tests found in the **.usertest** file.

–t Run **sundiag** on a terminal. Incompatible with **–C**.

–v Suppress the **sundiag** start-up messages so that they do not interfere with the display when SunView windows come up. This argument may be used in the **.sunview** file.

–h *remotehost*
Use this option to invoke **sundiag** when using the SunView-based Remote Interface. Refer to the *Sundiag User's Guide* for information on this interface.

–o *saved_options_file*
Use the *saved_options_file* to restore options. The default option file is **.sundiag**. **.sundiag** is used if the **–o** option is not used and if the default file exists.

generic_tool_arguments
Refer to **sunview**(1) for examples of generic tool arguments that may be used with **sundiag**.

FILES

/var/adm/sundiaglog/options/.sundiag	start-up option file
/usr/diag/sundiag/.usertest	user-defined test description file

SEE ALSO

sunview(1), **curses**(3X)

Installing the SunOS
Sundiag User's Guide

NAME

talkd, in.talkd – server for talk program

SYNOPSIS

/usr/etc/in.talkd

DESCRIPTION

talkd is a server used by the **talk**(1) program. It listens at the udp port indicated in the "talk" service description; see **services**(5). The actual conversation takes place on a tcp connection that is established by negotiation between the two machines involved.

SEE ALSO

talk(1), **services**(5), **inetd**(8C)

BUGS

The protocol is architecture dependent, and can not be relied upon to work between Sun systems and other machines.

Notes

Notes

Notes

Notes

Notes

Notes

Systems for Open Computing™

Corporate Headquarters
Sun Microsystems, Inc.
2550 Garcia Avenue
Mountain View, CA 94043
415 960-1300
TLX 37-29639

For U.S. Sales Office locations, call:
800 821-4643
In CA: 800 821-4642

European Headquarters
Sun Microsystems Europe, Inc.
Bagshot Manor, Green Lane
Bagshot, Surrey GU19 5NL
England
0276 51440
TLX 859017

Australia: (02) 413 2666
Canada: 416 477-6745
France: (1) 40 94 80 00
Germany: (089) 95094-0
Hong Kong: 852 5-8651688
Italy: (39) 6056337
Japan: (03) 221-7021
Korea: 2-7802255
Nordic Countries: + 46 (0)8 7647810
PRC: 1-8315568
Singapore: 224 3388
Spain: (1) 2532003
Switzerland: (1) 8289555
The Netherlands: 02155 24888
Taiwan: 2-7213257
UK: 0276 62111

Europe, Middle East, and Africa, call European Headquarters:
0276 51440

Elsewhere in the world, call Corporate Headquarters:
415 960-1300
Intercontinental Sales