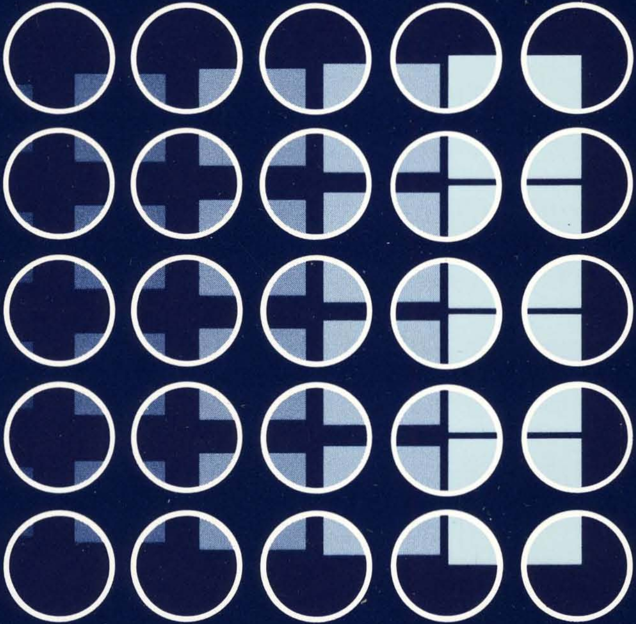
# ULTRIX-32™

## Programmer's Manual

**Building ULTRIX-32™ Systems With Config**

Order No. AA-BG60A-TE



digital

software

# Building ULTRIX-32™ Systems With Config

Order No. AA-BG60A-TE

digital equipment corporation, merrimack, new hampshire

**First printing, May 1984**

This software and documentation is based in part on the Fourth Berkeley Software Distribution under license from The Regents of the University of California. We acknowledge the following individuals for their role in its development:

Eric Allman, Ken Arnold, Ozalp Babaoglu, Scott B. Baden, Jerry Berkman, John Breedlove, Earl T. Cohen, Robert P. Corbett, Mike Curry, Steve Feldman, Tom Ferrin, John Foderaro, Susan L. Graham, Charles Haley, Robert R. Henry, Andy Hertzfeld, Mark Horton, S.C. Johnson, William Joy, Howard Katseff, Peter Kessler, Jim Kleckner, J.E. Kulp, James Larus, Kevin Layer, Mike Lesk, Steve Levine, Jeff Levinsky, Louise Madrid, M. Kirk McKusick, Colin L. McMaster, Mikey Olson, Geoffrey Peck, Ed Pelegri-Llopart, Rob Pike, Dave Presotto, John F. Reiser, Asa Romberger, Bill Rowan, Jeff Schreibman, Eric P. Scott, Greg Shenaut, Eric Shienbrood, Kurt Shoens, Keith Sklower, Helge Skrivervik, Al Stanberger, Ken Thompson, Michael C. Toy, Richard Tuck, Bill Tuthill, Mike Urban, Edward Wang, David Wasley, Joseph Weizenbaum, Jon L. White, Glenn Wichman, Niklaus Wirth.

# Building an ULTRIX-32 System with Config

## Table of Contents

# Building an ULTRIX-32 System with Config

## Overview

This document describes how to configure and create a bootable ULTRIX-32 system image using config(8). Based on a configuration file of tunable parameters and hardware support, config generates a collection of files used to build a copy of the ULTRIX-32 system appropriate to that configuration. config simplifies system maintenance by isolating system dependencies in a single, easy-to-understand file.

This document is divided into three sections:

- Section 1 discusses the system configuration file in which you define the system's tunable parameters and specify hardware support. It is critical that you tailor the configuration file to reflect the state of your system. A major portion of this document discusses the configuration file and how to edit it with ed(1).
- Section 2 outlines the two remaining steps to build your system: running config, and using make(8) to compile and load the system.
- Section 3 contains example configuration files for a VAX-11/780, VAX-11/750, and VAX-11/730 processor. This section also discusses how each file was constructed. You should refer to this section if you need additional information to tailor the configuration file to your system.

The appendixes cover these subjects:

- Appendix A discusses configuration file grammar.
- Appendix B outlines the rules config uses for defaulting system devices.
- Appendix C, geared for sophisticated users, describes data structure sizing rules.

## 1. The Configuration File

These are the basic steps for building an ULTRIX-32 system:

1. Create a configuration file for your system. (subsection 1.1)
2. Make a directory in which you will construct the system. (subsection 2.1)
3. Run config on the configuration file. config generates the files required to compile and load the system image. (subsection 2.2)
4. Construct source code dependency rules for the configured system. (subsection 2.3)
5. Compile and load the system with make(1). (subsection 2.4)

Steps 1 and 2 are usually done only once. If you are reconfiguring your system to add, delete, or switch hardware, you only need to repeat the last three steps.

Section 1 describes how to create a configuration file for a standard configuration. It outlines the contents of the configuration file, and describes the rules for writing the file.

### 1.1. Existing Configuration Files

The first step for building an ULTRIX-32 system is to create a configuration file. It is easier to copy and edit an existing configuration file with ed(1) rather than to create an entirely new file.

The ULTRIX-32 system is distributed with four configuration files you can use as a basis for configuring your system. The files are located in the /sys/conf directory. The files are:

| File Name: | Configured for: |
|---|---|
| MY780 | VAX-11/780 |
| MY750 | VAX-11/750 |
| MY730 | VAX-11/730 |
| GENERIC | any VAX processor |

To create a configuration file for your system, first change directory to /sys/conf. Then copy the appropriate template file to a file that reflects your VAX processor name. By convention, the configuration file name is always in uppercase letters. For example, to copy the MY780 file to a new file called MYVAX, type:

```
        # cd /sys/conf
        # cp MY780 MYVAX
```

Do not move the new file out of the /sys/conf directory.

## 1.2. Contents of the Configuration File

This section describes basic and additional configuration file information.

After making a copy of the appropriate configuration file, modify the new file using the ed(1) editor to reflect your system's configuration. See *The ed Command Summary Sheet*, in *The ULTRIX-32 Installation Guide* for a quick lesson on how to use ed.

Your configuration file must contain this information:

- Machine type
- CPU type
- System identification
- INET and QUOTA options
- Maximum number of users
- Location of the root filesystem
- Available hardware
- Pseudodevices pt, loop, INET and ETHER

A configuration file is broken up into three logical parts:

- Global configuration parameters for all system images specified in the configuration file
- Parameters specific to each system image to be generated
- Device specifications

### 1.2.1. Global Configuration Parameters

Specify each global configuration parameter with a separate line in the configuration file. The following lists and explains how to specify each parameter:

**machine** *type*

This parameter indicates the system is going to run on a VAX-11 computer. To identify the machine *type*, make sure your configuration file contains this line:

```
machine     vax
```

config uses the machine type to locate machine specific data files, and to select rules used in constructing resultant configuration files.

**cpu**

This parmeter indicates the CPUs on which the system will operate. You must enclose *type* in quotation marks. You can specify more than one CPU type, but you should configure your system to run on all the CPUs you list here. Legal CPU types are: VAX780, VAX750, and VAX730.

For example, to configure the system for a VAX-11/780, this line must be in your configuration file:

```
cpu       "VAX780"
```

**ident** *name*

This parameter identifies the system, and is often the name of the machine running the system. The ident name you chose must contain all alphabetic characters, and must be the same name as that in the configuration file you created. For example, the following line identifies your system as MYVAX:

```
ident     MYVAX
```

GENERIC is an identifier for a system that runs on any VAX CPU. Do not use GENERIC as a system identifier unless you want to configure a generic system.

**timezone** *number* [ **dst** [ *x* ] ]

This parameter identifies your timezone, which is measured by the number of hours west of Greenwich Mean Time (GMT). For example, Eastern Standard Time is five hours west of GMT, and Pacific Standard time is eight hours west of GMT. Negative numbers indicate hours east of GMT. If you specify **dst**, the system operates under daylight savings time.

You can include an integer or floating point number (*x*) to request a particular daylight saving time correction algorithm. The default value is 1, indicating the United States. The other values are: 2 (Australia), 3 (Western Europe), 4 (Central Europe), and 5 (Eastern Europe).

For example, to indicate you are in the Eastern time zone and use daylight savings time, a line in your configuration file should read:

```
timezone    5 dst
```

In this example, *x* is the default of 1, indicating the United States.

The timezone parameter supplies the information returned by the gettimeofday(2) system call.

**maxusers** *number*

The maximum number of simultaneously active users is set by your license agreement. Do not alter the maxusers parameter in your configuration file.

**options** *optionlist*

The example configuration files distributed with the ULTRIX-32 system include the INET and QUOTA options. The INET option provides internet communication protocols; the QUOTA option allows disk quotas to be set. You need both options to create a new kernel for your machine. You should leave the options as they appear in the example configuration files (see section 3.0).

There are additional options associated with certain peripheral devices. These are listed under the Synopsis of each device description in Section 4 of the *ULTRIX Programmer's Manual* (for example dz(4))

### 1.2.2. System Image Parameters

The second part of the configuration file pertains to the system image parameters.

You can generate more than one system image with a single configuration file. Using one file allows you to create two kernels that differ only on the location of the root filesystem. The system image parameters are:

- System image name
- Root filesystem location
- Number and location of swapping and paging areas
- Device to use for system dumps
- Device used for argument processing with execve(2)

System image parameters are specific to each system image generated. Specify the system image parameters by editing your configuration file, adding a line of this form:

config *image-name configuration clauses*

The *image-name* field is the name you assign to the loaded system image; name this field vmunix.

In addition to the *image-name* called vmunix, you can specify alternate config lines, provided that each *image-name* is unique. This allows you to configure one of multiple kernels with different root and/or swap devices to be run on the same system.

The *configuration clauses* are:

**root** [ **on** ] *device*
**swap** [ **on** ] *device* [ **and** *device* ] [ **size** *x* ]
**dumps** [ **on** ] *device*
**args** [ **on** ] *device*

**on** is optional. The configuration clauses all appear on the same line, however, you should separate multiple configuration clauses by a blank. Begin each continuation line of a specification covering multiple lines with a tab character.

Further details on the components that make up the *configuration clauses* are:

**root [ on ]**

The system must know the location of the root filesystem at boot time. Follow the root on clause with the name of the device which contains the root filesystem.

**swap [ on ]**

The kernel must know what devices to use for swapping. Follow the swap on clause with the device name and partition that will be used for the paging and swapping areas.

**dumps [ on ]**

To force system dumps on a particular device, follow this clause with the name of the device where the dump should be taken.

**args [ on ]**

You can specify the device on which the system should process arguments during system calls. However, most sites let config determine this device based on the rules for selecting default locations for system devices.

**device**

A complete device specification consists of a device name, unit, and filesystem partition.

You can specify complete device names in the configuration clauses, or you can rely on config's default rules to select unit numbers and filesystem partitions. The rules depend on the overall system configuration. Appendix B contains a complete list of the default rules for selecting system configuration devices.

config translates the device names to major and minor device numbers on a per-machine basis. It uses specifications in the /sys/conf/devices.vax file to map a device name to its major block device number. It calculates the minor device number using standard disk partitioning rules.

You can state explicitly the major/minor device in the configuration clause if the default mapping of device name to major/minor device number is incorrect for your configuration. To do this, substitute:

```
major x minor y
```

for the device name in the configuration clause. For example, here is a device name substitution:

```
config vmunix root on major 99 minor 1
```

**size**

The system sizes the areas configured for swap space at boot time. You can specify a non-standard partition size for one or more swap areas. To do this, append size and the appropriate size in sectors to the device name specification. For example:

```
config vmunix root on hp0 swap on hp0b size 1200
```

In this example, config forces swapping in partition b of hp0, with the swap partition size set to 1200 sectors. A swap area sized larger than the associated disk partition is trimmed to the partition size.

**swap generic**

Specify this clause to create a generic configuration. Any extra clauses will cause an error.

### 1.2.3. Device Specifications

When the system boots it goes through an autoconfiguration phase. During this period, the system searches for all the hardware devices that the system builder indicated might be present. This search requires information such as register addresses and bus interconnects.

You can configure the system's hardware to be very flexible, or to be totally inflexible. Most system managers configure hardware devices into the system for these reasons: 1) the devices are currently present on the machine, 2) the devices will be present shortly, 3) to guard against a hardware failure somewhere else at the site. You should configure in extra disks in case an emergency requires moving one disk off a machine with hardware problems.

Hardware devices specified in the configuration do not need to be present on the machine where the generated system will run. The system only uses the hardware it finds at boot time.

Specification of hardware devices in the configuration file parallels the interconnection hierarchy of the machine to be configured. Thus, the configuration file must indicate which MASSBUS and UNIBUS adapters are present and which NEXUSes they might be connected to.

**NOTE:** Although VAX-11/750s and VAX-11/730s do not actually have NEXUSes, the system treats these systems as having simulated NEXUSes to simplify device configuration.

Devices and controllers must indicate possible connections to one or more adapters.

A device description may either provide a complete or incomplete definition of the possible configuration parameters. For an incomplete definition, the system probes for all the possible values. Leaving a parameter undefined allows a single device configuration list to match many possible physical configurations.

For example, a disk may either be indicated as present at MASSBUS adapter 0, or at any MASSBUS adapter located at boot time. The latter scheme, termed *wildcarding*, allows more flexibility in the physical configuration of a system; if a disk must be moved around for some reason, the system still locates it at the alternate location.

A device specification takes one of these forms:

**controller** *device-name device-info* [ *interrupt-spec* ]

> **controller** is a disk controller, a UNIBUS tape controller, a MASSBUS adapter, or a UNIBUS adapter.

**device** *device-name device-info interrupt-spec*

> **device** is an autonomous device which connects directly to a UNIBUS adapter (as opposed to a disk, for example, which connects through a disk controller).

**disk** *device-name device-info*

> **disk** identifies disk drives connected to a controller or master.

**master** *device-name device-info*

> **master** is a MASSBUS tape controller.

**tape** *device-name device-info*

> **tape** identifies tape drives connected to a controller or master.

The variable *device-name* is one of the standard device names, indicated in Section 4 of the *ULTRIX-32 Programmer's Manual*, followed by the logical unit number assigned the device. The logical unit number may differ from the physical unit number indicated on the front of the device. The logical unit number refers to the

ULTRIX-32 device, not the physical unit number. For example, hp0 is logical unit 0 of a MASSBUS storage device, even though it might be physical unit 3 on MASSBUS adapter 1.

The variable *device-info* specifies how the hardware is connected in the interconnection hierarchy. The UNIBUS and MASSBUS adapters are connected to the internal system bus through a NEXUS. Thus, you should edit your configuration file, and you could use one of these specifications:

```
controller      mba0      at nexus x
controller      uba0      at nexus x
```

To tie a controller to a specific NEXUS, supply *x* as the number of that NEXUS; otherwise specify *x* as **?** and the system will probe all NEXUSes present, looking for the specified controller.

The remaining interconnections are:

- A controller can be connected to another controller (for example, a disk controller attached to a UNIBUS adapter)
- A master is always attached to a controller (for example, a MASSBUS adapter)
- A tape is always attached to a master (for example, MASSBUS tape drives)
- A disk is always attached to a controller
- Devices are always attached to controllers (for example, UNIBUS controllers attached to UNIBUS adapters).

These are examples that you could add to your configuration file:

```
controller      hk0       at uba0 ...
disk            rk1       at hk0 ...
master          ht0       at mba0 ...
tape            tu0       at ht0 ...
device          dz0       at uba0 ...
```

You can wildcard any piece of hardware accross multiple controllers which can be connected to a specific controller except drives on a MASSBUS.

For the system to configure devices, it needs an indication of where or how a device will interrupt. For tapes and disks, simply specify the slave or drive number to locate the control status register for the device. For controllers, you must explicitly state the control status register, as well how many interrupt vectors are used and the names of the routines to which they should be bound. Thus you could complete the example lines given above as:

```
controller      hk0   at uba0 csr 0177440          vector rkintr
disk            rk1   at hk0 drive 1
master          ht0   at mba0 drive 0
tape            tu0   at ht0 slave 0
```

```
device    dz0    at uba0 csr 0160100    flags 0xff vector dzrint dzxint
```

Some device drivers require extra information passed to them at boot time to tailor their operation to the actual hardware present. For example, the line printer driver needs to know how many columns are present on each non-standard line printer (that is, a line printer with other than 80 columns). The drivers for the terminal multiplexors need to know which lines are attached to modem lines so that no one can use them unless a connection is present. For this reason, you can specify one last parameter called the *flags* field. It follows *csr*, and has this syntax:

**flags** *number*

*number* is passed directly to the associated driver. See Section 4 of the *ULTRIX-32 Programmer's Manual* to determine how each driver uses this value (if at all).

Communications interface drivers commonly use the flags to indicate whether modem control signals are in use.

For further information about a specific device, see its synopsis section in Section 4 of the *ULTRIX-32 Programmer's Manual*, for example dmf(4).

## 1.2.4. Required Pseudodevices

Many drivers and software subsystems are treated like device drivers without any associated hardware. This type of driver is called a pseudodevice. The ULTRIX-32 system requires several pseudodevice specifications in the configuration file. The example configuration files distributed with the ULTRIX-32 system include the following pseudodevice specifications:

```
pseudo-device   pty
pseudo-device   loop
pseudo-device   inet
pseudo-device   ether
```

Your configuration file must also contain these specifications.

## 1.2.5. Adding the Network Support

The Internet protocols use the INET *pseudo-device* in addition to the global INET option. You need pseudo terminals to allow users to log in across the network.

You also need to add the DEUNA Ethernet device to your configuration file, if it is appropriate for your system. The specification is:

```
device          de0     at uba? csr 0174510 vector deintr
```

Optional pseudodevice specifications include imp, which is needed if you configure a CSS of ACC imp.

## 2. Using config to Build the System

Section 1 of this document describes the first step in building an ULTRIX-32 system, that is, how to set up and edit a configuration file for your system.

Section 2 describes the remaining four steps for building a bootable system image for your system with the config program. When your configuration file edits are complete, proceed with the steps described in this section.

### 2.1. Making a Directory for the System

The second step for building an ULTRIX-32 system is to build a directory in which you will construct the system.

The configuration file must have the same name as the directory you build. Use uppercase letters for both the configuration file and the directory names. Also, config assumes that both the directory containing the configuration file and the directory where you will construct the system are subdirectories of the same parent directory.

For example, if your configuration file is called /sys/conf/MYVAX, you need to create a directory called /sys/MYVAX in which to construct the system. You should now be in the /sys/conf directory. Create the new directory using this command:

```
        # mkdir ../MYVAX
```

Do not move configuration directories out of /sys. Most of the system code and the files config creates use pathnames of the ../ form.

### 2.2. Running the config Program

The third step for building an ULTRIX-32 system is to run config on the configuration file.

Once you have completed the configuration file and created a configuration directory, you are ready to run the config program. You should still be located in the /sys/conf directory.

To run config on a a configuration file called MYVAX, type:

```
        # config MYVAX
```

Check for any errors that config may report. In general, the error diagnostics are self explanatory. Do not use a system in which config has reported errors; the results are unpredictable.

A successful run of config on your configuration file generates a number of files in the configuration directory. These files are:

- The makefile, which is used by make(1) in compiling and loading the system.

- One file for each possible system image for your machine which describes where swapping, the root filesystem, and other miscellaneous system devices are located.
- A collection of header files, one per possible device the system supports, which define the hardware configured.
- A file containing the i/o configuration tables used by the system during its autoconfiguration phase.
- An assembly language file of interrupt vectors which connect interrupts from your machine's external buses to the main system path for handling interrupts.

## 2.3. Constructing Source Code Dependencies

The fourth step for building an ULTRIX-32 system is to construct the source code dependency rules.

After config has finished generating the files you need to compile and link your system, it reports:

```
        Don't forget to run "make depend"
```

This message reminds you to run the make(1) program. make is a tool for maintaining program groups. You run make to build the rules that recognize dependencies in the system source code. These rules ensure that any changes to a piece of the system source code results in the proper modules being recompiled when you use make to rebuild a bootable system image.

To construct these source code dependencies, first change directory to the configuration directory. For example, if your configuration directory is /sys/MYVAX, change directory with:

```
        # cd ../MYVAX
```

Then type:

```
        # make depend
```

The result is a report on the progress of the make program. The system eventually responds with a # prompt:

```
        # make depend
        grep '^#include' ...
           .
           .
           .
        rm eddep makedep
        #
```

## 2.4. Building the System

The last step for building an ULTRIX-32 system is to compile and load the system.

After you have constructed the source code dependencies, use the make program to build the new system. To do this, type:

```
# make image-name
```

*image-name* is the name you gave the system image in your configuration file. For example, if you named your bootable system image vmunix, then type the following to generate a bootable image named vmunix:

```
# make vmunix
```

The make program prints several lines, reporting its progress. The system eventually responds with a # prompt.

If you have multiple kernels on your system (see subsection 1.2.2), you should do a make on each specific image-name. For example, suppose you configured vmunix with the root filesystem on an hp device, and hkvmunix with the root filesystem on an hk device. In this case, you can generate the binary images for each root filesystem by typing:

```
# make vmunix hkvmunix
```

**NOTE:** vmunix must precede any other *image-name*. Also, the name of a bootable image is different from the system identifier. All bootable images are configured for the same system; but the information about the root filesystem and paging devices differ.

You have now built a new system image. You should make a copy of the new system image, and bring the system up to multi-user mode to make sure it functions properly.

Return to Section 3.3 of the *ULTRIX-32 Installation Guide* for the procedures to do this.

## 3. Sample Configuration Files

Section 3 describes how to configure a sample VAX-11/780 system so the hardware can be reconfigured to guard against various hardware mishaps. It then outlines the configuration for a VAX-11/750 and a VAX-11/730.

Read this section if you need examples on how to edit the configuration file to reflect your system's configuration.

### 3.1. VAX-11/780 System

This subsection outlines the steps for configuring a basic VAX-11/780 system. Table 3.1 lists the hardware to be configured.

Table 3.1 Example VAX-11/780 Hardware Support

| Item | Connection | Name | Reference |
|---|---|---|---|
| cpu | | "VAX780" | |
| MASSBUS adapter | nexus ? | mba0 | |
| disk | mba0 | hp0 | hp(4) |
| disk | mba0 | hp1 | |
| MASSBUS adapter | nexus ? | mba1 | |
| tape controller | mba1 | ht0 | ht(4) |
| tape drive | ht0 | tu0 | |
| UNIBUS adapter | nexus ? | uba0 | |
| line printer | uba0 | lp0 | lp(4) |
| DZ multiplexor | uba0 | dz0 | dz(4) |
| DMF multiplexor | uba0 | dmf0 | dmf(4) |
| DEUNA Ethernet | uba0 | de0 | de(4) |

The rest of this section describes step-by-step how to construct a configuration file.

#### 3.1.1. Filling in Global Configuration Parameters

The first step is to edit your configuration file. Using the ed(1) editor (a summary sheet of how to use ed is included in this binder), fill in the global configuration parameters. The machine is a VAX; therefore, *machine type* is vax. Assume this system runs only on one processor; therefore, *cpu type* is VAX780. For this example, system identifier is EXAMPLE780.

Do not alter the *maxusers* option; it was determined by the license agreement, and it specifies the maximum number of simultaneous users on the system. After editing the configuration file, the beginning of it looks like this:

```
#
# EXAMPLE780 (an example configuration for a VAX-11/780)
#
machine         vax
cpu             VAX780
timezone        5 dst
ident           EXAMPLE780
maxusers        32
options         INET
options         QUOTA
```

### 3.1.2. Specifying System Images

Now add to the configuration file the specifications for three system images.

The first specification is the standard system with the root on hp0 and swapping on the same drive as the root. The second has the root filesystem in the same location, but the swap space is interleaved between hp0 and hp1. The third is a generic system, to allow us to boot off either disk drive. These are the appropriate specification lines:

```
config          vmunix              root on hp0
config          hpvmunix            root on hp0 swap on hp0 and hp1
config          genvmunix           swap generic
```

### 3.1.3. Specifying Hardware

Now add the hardware specifications to the configuration file.

Transcribe the information from Table 3.1. Refer to the Synopsis section for each device description in Section 4 of the *ULTRIX-32 Programmer's Manual* for the device name and other information needed in the configuration file. The device specifications could look like this:

```
controller      mba0   at nexus ?
disk            hp0    at mba0 drive 0
disk            hp1    at mba0 drive 1
controller      mba1   at nexus ?
master          ht0    at mba1 drive 0
tape            tu0    at ht0 slave 0
controller      uba0   at nexus ?
device          lp0    at uba0 csr 0177514    vector lpintr
device          dz0    at uba0 csr 0160100 flags 0x00    vector dzrint dzxint
device          dmf0   at uba0 csr 0160340 flags 0xff
        vector dmfsrint dmfsxint dmfdaint dmfdbint dmfrint dmfxint dmflint
        device          de0     at uba0 csr 0174510    vector deintr
```

This configuration file suffices, but leaves little flexibility. Suppose the first disk controller were to break. You would want to recable the drives on the second controller so that you could still use all the disks without reconfiguring the system. To recable the drives, wildcard the MASSBUS adapter connections and also the slave numbers. The revised device specifications are:

```
controller          mba0        at nexus ?
disk                hp0         at mba? drive ?
disk                hp1         at mba? drive ?
controller          mba1        at nexus ?
master              ht0         at mba1 drive ?
tape                tu0         at ht0 slave 0
controller          uba0        at nexus ?
device              lp0         at uba? csr 0177514    vector lpintr
device              dz0         at uba? csr 0160100 flags 0x00 vector dzrint dzxint
device              dmf0        at uba? csr 0160340 flags 0xff
    vector dmfsrint dmfsxint dmfdaint dmfdbint dmfrint dmfxint dmflint
device              de0         at uba? csr 0174510    vector deintr
```

### 3.1.4. Example Configuration File for VAX-11/780

The following is an example of an edited configuration file for a VAX-11/780 processor:

```
#
# MY780 Configuration file.
#
machine         vax
cpu             "VAX780"
ident           MY780
timezone        5 dst
maxusers        32
options         INET
options         QUOTA

config          vmunix   root on hp0 swap on hp0 and hp1

controller      mba0      at nexus ?
controller      mba1      at nexus ?
controller      mba2      at nexus ?
controller      mba3      at nexus ?
controller      uba0      at nexus ?
controller      uba1      at nexus ?
controller      uba2      at nexus ?
controller      uba3    at nexus ?
disk            hp0     at mba? drive 0
disk            hp1     at mba? drive 1
disk            hp2     at mba? drive 2
disk            hp3     at mba? drive 3
disk            hp4     at mba? drive 4
master          ht0     at mba? drive ?
tape            tu0     at ht0 slave 0
tape            tu1     at ht0 slave 1
master          mt0     at mba? drive ?
tape            mu0     at mt0 slave 0
tape            mu1     at mt0 slave 1
controller      hk0     at uba? csr 0177440   vector rkintr
disk            rk0     at hk0 drive 0
disk            rk1     at hk0 drive 1
device          lp0     at uba? csr 0177514   vector lpintr
device          dz0     at uba? csr 0160100 flags 0x00   vector dzrint dzxint
device          dz1     at uba? csr 0160110 flags 0xff   vector dzrint dzxint
device          dz2     at uba? csr 0160120 flags 0x00   vector dzrint dzxint
device          dz3     at uba? csr 0160130 flags 0xff   vector dzrint dzxint
device          de0     at uba? csr 0174510   vector deintr
pseudo-device   pty
pseudo-device   inet
pseudo-device   loop
pseudo-device   ether
```

## 3.2. VAX-11/750 System

This subsection presents a sample configuration file for a VAX-11/750 with networking support. Table 3.2 lists the hardware to be configured.

Table 3.2 Example VAX-11/750 Hardware Support

| Item | Connection | Name | Reference |
|---|---|---|---|
| cpu | | VAX750 | |
| MASSBUS adapter | nexus ? | mba0 | |
| RM05 disks | mba0 | hp0 | hp(4) |
| UNIBUS adapter | nexus ? | uba0 | |
| UDA50-a controller | uba0 | uda0 | uda(4) |
| RA81 disks | uda0 | ra0 | |
| DZ multiplexer | uba0 | dz0 | dz(4) |
| DEUNA Ethernet | uba0 | de0 | de(4) |

### 3.2.1. Filling in Global Parameters

The machine is a VAX; therefore, *machine type* is vax. Assume this system runs only on one processor; therefore, *cpu type* is VAX750. For this example, the system identifier is EXAMPLE750.

Do not alter the *maxusers* option; it was determined by the license agreement.

You must specify the INET option for networking support. After editing the configuration file, the beginning of it looks like this:

```
#
# EXAMPLE750 (an example VAX-11/750 with networking support)
#
machine         vax
cpu             "VAX750"
ident           EXAMPLE750
timezone        5 dst
maxusers        32
options         INET
options         QUOTA
```

### 3.2.2. Specifying System Images

Now add to the configuration file the specifications for the system image.

For this example, swapping will be on the RM05 (hp0):

```
        config      vmunix      root on hp0
```

## 3.2.3. Specifying Hardware

Now the hardware specifications to the configuration file.

Transcribe the information from Table 3.2, and refer to the Synopsis for each device description in Section 4 of the *ULTRIX-32 Programmer's Manual* for the device name and other information needed in the configuration file. Remember to allow for flexibility.

The configured hardware looks like this:

```
        controller          mba0   at nexus ?
        disk                hp0    at mba? drive 0
        controller          uba0   at nexus ?
        controller          uda0   at uba? csr 0172150 vector udintr
        disk                ra0    at uda0 drive 0
        device              dz0    at uba? csr 0160100 flags 0xff vector dzrint dzxint
        device              de0    at uba? csr 0174510 vector deintr
```

## 3.2.4. Example Configuration File for VAX-11/750

The following is an example of an edited configuration file for a VAX-11/750 processor:

```
#
# MY750
# Configuration file for a VAX-11/750
#
machine         vax
cpu             "VAX750"
ident           MY750
timezone        5 dst
maxusers        32
options         QUOTA
options         INET

config          vmunix      root on hp0a swap on hp0b dumps on hp0b args on hp0b

controller      mba0        at nexus ?
controller      mba1        at nexus ?
controller      uba0        at nexus ?
disk            hp0         at mba? drive 0
disk            hp1         at mba? drive ?
```

```
master          ht0     at mba? drive ?
tape            tu0     at ht0 slave 0
controller      hk0     at uba? csr 0177440   vector rkintr
disk            rk0     at hk0 drive 0
disk            rk1     at hk0 drive 1
device          dz0     at uba? csr 0160100 flags 0xff   vector dzrint dzxint
controller      zs0     at uba? csr 0172520   vector tsintr
device          ts0     at zs0 drive 0
device          dmf0    at uba? csr 0160340 flags 0xff
      vector dmfsrint dmfsxint dmfdaint dmfdbint dmfrint dmfxint dmflint
pseudo-device   pty
pseudo-device   loop
pseudo-device   inet
pseudo-device   ether
device          lp0     at uba? csr 0177514   vector lpintr
device          de0     at uba? csr 0174510   vector deintr
```

## 3.3. VAX-11/730 System

This subsection describes the steps for configuring a basic VAX-11/730 system. The configuration file for a VAX-11/730 with networking support is similar to the VAX-11/750 described in subsection 3.2, but without a MASSBUS. Table 3.3 lists the hardware to be configured.

Table 3.3 Example VAX-11/730 Hardware Support

| Item | Connection | Name | Reference |
|---|---|---|---|
| cpu | | VAX730 | |
| UNIBUS adapter | nexus ? | uba0 | |
| UDA50-a controller | uba0 | uda0 | uda(4) |
| RA80 | uda0 | ra0 | |
| RA81 | uda0 | ra1 | |
| IDC | uba0 | idc0 | |
| RL02 | uda0 | rb0 | |
| tape controller | uba0 | zs0 | ts(4) |
| tape drive | zs0 | ts0 | |
| line printer | uba0 | lp0 | lp(4) |
| DZ multiplexor | uba0 | dz0 | dz(4) |
| DMF multiplexor | uba0 | dmf0 | dmf(4) |
| DEUNA Ethernet Device | uba0 | de0 | de(4) |

### 3.3.1. Filling Global Configuration Parameters

The machine is a VAX; therefore, *machine type* is vax. Assume this system runs only on one processor; therefore, *cpu type* is VAX730. For this example, the system identifier is EXAMPLE730.

Do not alter the *maxusers* option; it was set by the license agreement.

Set the *disk quota* option. For information on this option, see *Disk Quotas in a UNIX Environment* in the *ULTRIX-32 Programmer's Manual*, Volume 2C.

The global variables for this configuration file look like this:

```
#
# EXAMPLE730
#
machine         vax
cpu             "VAX730"
ident           EXAMPLE730
timezone        5 dst
maxusers        16
options         INET
```

```
        options             QUOTA
```

## 3.3.2. Specifying System Images

Now add to the configuration file the specifications for the system image. In this example, swapping and the root filesystem are on ra0. This is the appropriate specification line:

```
        config      vmunix      root on ra0 swap on ra0
```

## 3.3.3. Specifying Hardware

Now add the hardware specifications to the configuration file.

Transcribe the information from Table 3.3 and refer to the Synopsis for each device description in Section 4 of the *ULTRIX-32 Programer's Manual* for the device name and other information needed in the configuration file.

The configured hardware looks like this:

```
        controller          uba0        at nexus ?
        controller          uda0        at uba0 csr 0172150     vector udintr
        disk                ra0         at uda0 drive 0
        disk                ra1         at uda0 drive 1
        controller          idc0        at uba0 csr 0175606     vector idcintr
        disk                rb0         at idc0 drive 0
        controller          zs0         at uba0 csr 0172520     vector tsintr
        device              ts0         at zs0 drive 0
        device              lp0         at uba0 csr 0177514     vector lpintr
        device              dz0         at uba0 csr 0160110 flags 0xff    vector dzrint dzxint
        device              dmf0        at uba0 csr 0160400 flags 0xff
             vector dmfsrint dmfsxint dmfdaint dmfdbint dmfrint dmfxint dmflint
        device              de0         at uba0 csr 0174510      vector deintr
```

## 3.3.4. Example Configuration File for VAX-11/730

The following is an example of an edited configuration file for a VAX-11/730 processor:

```
#
# MY730
# Configuration file for VAX 11/730
#
machine         vax
cpu             "VAX730"
ident           MY730
timezone        5 dst
```

```
maxusers        16
options         INET
options         QUOTA

config          vmunix      root on ra0 swap on ra0

controller      uba0        at nexus ?
controller      hk0         at uba0 csr 0177440    vector rkintr
disk            rk0         at hk0 drive 0
disk            rk1         at hk0 drive 1
controller      uda0        at uba0 csr 0172150    vector udintr
disk            ra0         at uda0 drive 0
disk            ra1         at uda0 drive 1
controller      idc0        at uba0 csr 0175606    vector idcintr
disk            rb0         at idc0 drive 0
disk            rb1         at idc0 drive 1
controller      hl0         at uba0 csr 0174400    vector rlintr
disk            rl0         at hl0 drive 0
controller      zs0         at uba0 csr 0172520    vector tsintr
device          ts0         at zs0 drive 0
device          dmf0        at uba0 csr 0160400 flags 0xff
    vector dmfsrint dmfsxint dmfdaint dmfdbint dmfrint dmfxint dmflint
device          dz0         at uba0 csr 0160110 flags 0xff    vector dzrint dzxint
pseudo-device   pty
pseudo-device   loop
pseudo-device   inet
pseudo-device   pup
pseudo-device   ether
device          de0         at uba0 csr 0174510    vector deintr
```

## APPENDIX A. Configuration File Grammar

The following grammar is a compressed form of the actual yacc(1) grammar config uses to parse configuration files. All uppercase indicates terminal symbols; bold indicates literals; brackets ([ ]) indicate optional clauses; and asterisks (*) indicate zero or more instantiations.

Configuration ::= [ Spec ; ]*

Spec ::= Config spec
  | Device spec
  | **trace**
  | /* lambda */

/* configuration specifications */

Config spec ::= **machine** ID
  | **cpu** ID
  | **options** Opt list
  | **ident** ID
  | System spec
  | **timezone** [ – ] NUMBER [ **dst** [ NUMBER ] ]
  | **timezone** [ – ] FPNUMBER [ **dst** [ NUMBER ] ]
  | **maxusers** NUMBER

/* system configuration specifications */

System spec ::= **config** ID System parameter [ System parameter ]*

System parameter ::= swap spec | root spec | dump spec | arg spec

swap spec ::= **swap** [ **on** ] swap dev [ **and** swap dev ]*

swap dev ::= dev spec [ **size** NUMBER ]

root spec ::= **root** [ **on** ] dev spec

dump spec ::= **dumps** [ **on** ] dev spec

arg spec ::= **args** [ **on** ] dev spec

dev spec ::= dev name | major minor

major minor ::= **major** NUMBER **minor** NUMBER

dev name ::= ID [ NUMBER [ ID ] ]

/* option specifications */

Opt list ::= Option [ , Option ]*

Option ::= ID [ = Opt value ]

Opt value ::= ID | NUMBER

/* device specifications */

Device spec ::= **device** Dev name Dev info Int spec  
  | **master** Dev name Dev info  
  | **disk** Dev name Dev info  
  | **tape** Dev name Dev info  
  | **controller** Dev name Dev info [ Int spec ]  
  | **pseudo-device** Dev [ NUMBER ]

Dev name ::= Dev NUMBER

Dev ::= **uba** | **mba** | ID

Dev info ::= Con info [ Info ]*

Con info ::= **at** Dev NUMBER  
  | **at nexus** NUMBER

Info ::= **csr** NUMBER  
  | **drive** NUMBER  
  | **slave** NUMBER  
  | **flags** NUMBER

Int spec ::= **vector** ID [ ID ]*  
  | **priority** NUMBER

## Lexical Conventions

The terminal symbols are loosely defined as:

ID

One or more alphabetic characters, either uppercase or lowercase, and an underscore ( ).

NUMBER

Approximately the C language specification for an integer number. That is, a leading 0x indicates a hexadecimal value, and a leading 0 indicates an octal value. Otherwise the number is expected to be a decimal value. Hexadecimal numbers may use either uppercase or lowercase alphabetic characters.

FPNUMBER

A floating point number without an exponent. That is, a number of the form *nnn.ddd*, where the fractional component is optional.

In special instances you can substitute a question mark (?) for a NUMBER token. The question mark effects wildcarding in device interconnection specifications.

To indicate comments in a configuration file, use a # character at the beginning of the line; the remainder of the line is not interpreted.

To have a specification interpreted as a continuation of the previous line, make the first character of the line a tab.

## APPENDIX B. Rules for Default System Devices

When config processes a config rule which does not fully specify the location of the root file system, paging area(s), device for system dumps, and device for argument list processing, it applies a set of rules to define those values left unspecified. The following list of rules are used as defaults for system devices:

1) If a root device is not specified, the swap specification indicates building a generic system.
2) If the root device does not specify a unit number, it defaults to unit 0.
3) If the root device does not include a partition specification, it defaults to the a partition.
4) If no swap area is specified, it defaults to the b partition of the root device.
5) If no device is specified for processing argument lists, the first swap partition is selected.
6) If no device is chosen for system dumps, the first swap partition is selected

The following summarizes the default partitions selected when a device specification is incomplete, for example, hp0:

| Type | Partition |
|---|---|
| root | a |
| swap | b |
| args | b |
| dumps | b |

### Multiple Swap/Paging Areas

When multiple swap partitions are specified, the ULTRIX-32 system treats the first partition specified as the primary swap area. The remaining partitions are interleaved into the paging system when a swapon(2) system call is made. Interleaving usually happens at boot time with a call to swapon(8) from the /etc/rc file.

### System Dumps

System dumps automatically happen after a system crash, provided the device driver for the dumps device supports this. The dump contains the contents of memory, but not the swap areas. Normally the dump device is a disk, in which case the information is copied to a location near the back of the partition. The dump is placed there because the primary swap and dump device are commonly the same device and this allows the system to be rebooted without immediately overwriting the saved information.

If a dump occurs, the system variable *dumpsize* is set to a non-zero value indicating the size (in bytes) of the dump. The savecore(8) program then copies the information from the dump partition to a file in a crash directory, and also makes a copy of the system which was running at the time of the crash (usually /vmunix).

The offset to the system dump is defined in the system variable *dumplo* (a sector offset from the front of the dump partition). savecore(8) operates by reading the contents of *dumplo*, *dumpdev*, and *dumpmagic* from /dev/kmem. It then compares the value of *dumpmagic* read from /dev/kmem to that located in the corresponding location in the dump area of the dump partition. If a match is found, savecore(8) assumes a crash occurred and reads *dumpsize* from the dump area of the dump partition. This value is then used in copying the system dump.

*dumplo* is defined as:

> *dumpdev-size* – DUMPDEV

*dumpdev-size* is the size of the disk partition where system dumps are to be placed, and DUMPDEV is 10 megabytes. If the disk partition is not large enough to hold a 10 megabyte dump, *dumplo* is set to 0 (the front of the partition).

## APPENDIX C. VAX Kernel Data Structure Sizing Rules

**NOTE:** Do not follow the procedures described in Appendix C unless you are experienced with the ULTRIX-32 operating system.

Some system data structures are sized at compile time according to the maximum number of simultaneous users expected, while others are calculated at boot time based on the physical resources present; for example, memory.

This appendix lists both sets of rules and also includes some hints on how to change built-in limitations on certain data structures.

### Compile Time Rules

The file `/sys/conf/param.c` contains the definitions of almost all data structures sized at compile time. This file is copied into the directory of each configured system to allow configuration-dependent rules and values to be maintained. The rules implied by its contents are summarized in this appendix.

**NOTE:** MAXUSERS refers to the maximum number of simultaneous users defined in the configuration file. It is set by your license agreement. Do not alter this parameter.

**nproc**
: The maximum number of processes which may be running at any time. It is defined to be 20 + 8 * MAXUSERS and referred is to in other calculations as NPROC.

**ntext**
: The maximum number of active shared text segments. It is defined as 24 + MAXUSERS + NETSLOP, where NETSLOP is 20 when the Internet protocols are configured in the system; otherwise it is 0. The added size for supporting the network is to take into account the numerous server processes which are likely to exist.

**ninode**
: The maximum number of files in the file system which may be active at any time. This includes files in use by users, as well as directory files being read or written by the system, and files associated with bound sockets in the ULTRIX-32 ipc domain. **ninode** is defined as (NPROC + 16 + MAXUSERS) + 32.

**nfile**
: The number of file table structures. One file table structure is used for each open, unshared, file descriptor. Multiple file descriptors may reference a single file table entry when they are created through a `dup(2)` call, or as the result of a

fork(2.) **nfile** is defined to be

```
16 * (NPROC + 16 + MAXUSERS) / 10 + 32 + 2 * NETSLOP
```

NETSLOP is defined as it is for **ntext**.

**ncallout**
: The number of callout structures. One callout structure is used per internal system event handled with a timeout. Timeouts are used for terminal delays, watchdog routines in device drivers, protocol timeout processing, and so forth. **ncallout** is defined as 16 + NPROC.

**nclist**
: The number of c-list structures. C-list structures are used in terminal i/o. **nclist** is defined as 100 + 16 * MAXUSERS.

**nmbclusters**
: The maximum number of pages which may be allocated by the network. This is defined as 256 (a quarter megabyte of memory) in /sys/h/mbuf.h. The network rarely requires this much memory. It starts off by allocating 64 kilobytes of memory, then requests more as required. **nmbclusters** represents an upper bound.

**nquota**
: The number of quota structures allocated. Quota structures are present only when disc quotas are configured in the system. One quota structure is kept per user. **nquota** is defined as (MAXUSERS * 9) / 7 + 3.

**ndquot**
: The number of dquot structures allocated. Dquot structures are present only when disc quotas are configured in the system. One dquot structure is required per user, per active file system quota. That is, when a user manipulates a file on a file system on which quotas are enabled, the information regarding the user's quotas on that file system must be in-core. This information is cached, so that not all information must be present in-core all the time. **ndquot** is defined as (MAXUSERS * NMOUNT) / 4 + NPROC, where NMOUNT is the maximum number of mountable file systems.

In addition to the above values, the system page tables (which map virtual memory in the kernel's address space) are sized at compile time by the SYSPTSIZE definition in the file /sys/vax/param.h. SYSPTSIZE is defined as 20 + MAXUSERS pages of page tables. Its definition affects the size of many data structures allocated at boot time because it constrains the amount of virtual memory which the running system may address. This is often the limiting factor in the size of the buffer cache.

## System Size Limitations

The sum of the virtual sizes of the core-resident processes is limited to 64M bytes. The size of the text, and data segments of a single process are currently limited to 6M bytes each. The stack segment size is limited to 512K bytes as a soft, user-changeable limit, and may be increased to 6M with the setrlimit(2) system call. If these limits are insufficient, they can be increased by changing the constants MAXTSIZ, MAXDSIZ and MAXSSIZ in /sys/vax/vmparam.h, You must also change the definitions in /sys/h/dmap.h and /sys/h/text.h.

Be careful in making the changes; be sure that you have adequate paging space. As normally configured, the system has only 16M bytes per paging area. The best way to get more space is to provide multiple, thereby interleaved, paging areas.

To increase the amount of resident virtual space possible, you can alter the constant USRPTSIZE in /sys/vax/vmparam.h). To allow 128 megabytes of resident virtual space, change the 8 to a 16.

Because the file system block numbers are stored in page table *pg blkno* entries, the maximum size of a file system is limited to 2^19 1024 byte blocks. Thus no file system can be larger than 512M bytes.

There can be no more than 15 mountable file systems. If you have many disks, you should make some of them single file systems, and the paging areas do not count in this total. To increase the total, it is necessary to change the core-map /sys/h/cmap.h since there is a four bit field used here. The size of the core-map will then expand to 16 bytes per 1024 byte page. Remember to change MSWAPX and NMOUNT in /sys/h/param.h.

You can raise the maximum value NOFILE (open files per process limit) to 30 because of a bit field in the page table entry in /sys/machine/pte.h.

# HOW TO ORDER ADDITIONAL DOCUMENTATION

## DIRECT TELEPHONE ORDERS

| | | |
|---|---|---|
| In Continental USA and Puerto Rico call **800–258–1710** | In Canada call **800–267–6146** | In New Hampshire, Alaska or Hawaii call **603–884–6660** |

## DIRECT MAIL ORDERS (U.S. and Puerto Rico*)

DIGITAL EQUIPMENT CORPORATION
P.O. Box CS2008
Nashua, New Hampshire 03061

## DIRECT MAIL ORDERS (Canada)

DIGITAL EQUIPMENT OF CANADA LTD.
940 Belfast Road
Ottawa, Ontario, Canada K1G 4C2
Attn: A&SG Business Manager

## INTERNATIONAL

DIGITAL EQUIPMENT CORPORATION
A&SG Business Manager
c/o Digital's local subsidiary
or approved distributor

Internal orders should be placed through the Software Distribution Center (SDC), Digital Equipment Corporation, Northboro, Massachusetts 01532

*Any prepaid order from Puerto Rico must be placed
with the Local Digital Subsidiary:
809–754–7575

**ULTRIX - 32**
**Programmer's Manual**
AA-BG60A-TE

## Reader's Comments

**Note:** This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement. ______

______

______

______

Did you find errors in this manual? If so, specify the error and the page number.

______

______

______

Please indicate the type of user/reader that you most nearly represent.

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Other (please specify) ______

Name ______ Date ______

Organization ______

Street ______

City ______ State ______ Zip Code or Country ______

------Do Not Tear · Fold Here and Tape ----------------------------------------

digital

No Postage
Necessary
if Mailed in the
United States

**BUSINESS REPLY MAIL**
FIRST CLASS PERMIT NO.33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

Documentation Manager
ULTRIX-32™ Documentation Group
MK02-1/H10
Continental Blvd.
Merrimack, N.H.
03054

------Do Not Tear · Fold Here and Tape ----------------------------------------

Cut Along Dotted Line

## Notes:

## Notes:

digital™