sun microsystems

Software Technical Bulletin
June 1987

Software Information Services

Part Number 812-8701-05
Issue 1987 – 5
June 1987

sun microsystems

Software Technical Bulletin

June 1987

Software Information Services

Part Number 812-8701-05
Issue 1987 – 5
June 1987

Software Technical Bulletins are distributed to customers with software/hardware or software only support contracts. Send comments or corrections to ''Software Technical Bulletins'' at Sun Microsystems, Inc., 2550 Garcia Ave., M/S 2-34, Mountain View, CA 94043 or by electronic mail to sun!stb-editor. Customers wh have technical questions about topics in the Bulletin should call Sun Customer Software Services AnswerLine at 800 USA-4-SUN.

Contents

1

NOTES & COMMENTS

NOTES & COMMENTS

Editor's Notes

Editor's Notes

The editor's notes for this June issue of the Software Technical Bulletin (STB) include five new items of interest: making additional STB copies, a survey on STB distribution within your company, the STB cumulative index, a new STB section containing short subjects, and a preview of an upcoming new STB section that will contain a 'configuration of the month' article.

Duplicating the STB

Your company's software support contract includes a monthly issue of the STB, which contains a quarterly, updated Customer Distributed BugsList (CDB). Each month, the copy of your STB is mailed to your company's primary contact person or department. Sites with more than one contract may receive more than one STB copy, depending on how the contracts are set up.

Your primary contact person or department may duplicate this 'master' STB copy for all Sun workstation end-users as well as to those who need the information. So long as you duplicate copies and route them internally to employees working for a company having a Sun software support contract, there are no copyright infringement problems.

This limited permission for duplication is for your convenience only, however, and does not include any duplication for resale, for distribution outside your company, or for distribution to employees of companies not having a Sun software support contract.

If you have any questions, comments, or articles regarding the STB or CDB, please send your ideas and questions to *sun!stb-editor*.

STB Survey

I would be interested to hear from you regarding how your STB distribution is working. I am taking a survey of STB readers to determine how effectively the STB is being routed to Sun customers. Please take a moment to answer one of the two of the questions below. Then email your answers to me at *sun!stb-editor* or mail them to 'Software Technical Bulletins' at Sun Microsystems, Inc., 2550 Garcia Avenue, M/S 2-34, Mountain View, California 94043. Please include your company's name along with your answers.

Contact Point If you are named as the primary contact person or department on the Sun software support contract, have you missed any monthly STBs from Sun?

YES! _____

If yes, which ones did you miss? ____________________

NO!! _____

What date did you receive this June STB issue? ____________________

How many STB copies do you distribute within your company? ____________________

STB End-User When did you receive this issue? ____________________

I will forward the survey results to those at Sun in charge of STB distribution. Look to an upcoming STB issue for the survey results.

STB Cumulative Index

You may have noticed that your May STB issue began with page 39 and that this June issue begins with page 181. This cumulative pagination is being used since we are now including a cumulative STB index with each issue.

The index will be published monthly and will be current through the date of each monthly issue. It starts with the April 1987 issue, as does the cumulative pagination. The cumulative index also includes entries from each CDB, beginning with the May 1987 version. After looking over the index, please email any comments or suggestions for additional index entries to *sun!stb-editor.*

STB Short Subjects

The STB now contains a new section for short subjects. These are short articles on a single subject of general interest to STB readers. Many of these short subjects are those submitted by STB readers for wider circulation. If you have a short subject that you think would be of interest to STB readers, send it in!

Coming Attractions

The SIS staff is now working on a new section to be called 'Configuration of the Month.' This new section will include timely information on new product functions and technical information on how to install Sun products in a typical customer setting.

These articles will complement information appearing in announcements available from your sales representative and in the 'Read Me First' details shipped with each new product.

Thanks in advance for your readership and comments.

The STB Editor

2

ARTICLES

2

ARTICLES

ie0 Interrupts and SunOS 3.2

ie0 Spurious Interrupts and SunOS 3.2

When changing from SunOS 3.0 to SunOS 3.2, `ie0 spurious interrupts` occur when they had not done so previously. No other software or hardware was changed. No difference resulted after installing the release 3.0 ie ethernet driver and rebooting. The scenerio is shown below.

```
ie0 spurious interrupt
ie0 spurious interrupt
ie0 spurious interrupt
ie0: command not accepted
panic: iechkcca
```

The Possible Problem

A review of correct operation reveals that the ethernet driver loads a command to a register on the Intel ethernet chip and toggles a line on the chip when done. The chip responds by checking the command and processing it if the command is good. The chip then clears the register when done. In the meantime, the ethernet driver is in a busy-wait loop, checking for the cleared register. The driver does a `panic: iechkcca` if the busy-wait loop completes and the register is not cleared. The driver believes it has set a good command, but the chip does not think so because it has not cleared the register.

Our initial theory was that the driver sets the command and gets an unwanted ie interrupt before toggling the line. Another possibility was that the driver sets the command, toggles the line, begins the busy-wait loop, and then gets the unwanted interrupt. The driver then becomes confused because multiple driver parts are trying to process multiple tasks. It seemed, then, that either something was overwriting the good command in the chip, or an interrupt was arriving in the middle of the command-setting portion of the driver process. This would cause that task to be completed incorrectly.

The Problem Defined

The most likely cause of an iechkcca panic is some other hardware present that uses too many resources, preventing the 82586 chip from accessing memory within the timeout period. However, this hardware explanation seems unlikely since `ie0` rather than `ie1` is involved.

The best explanation we have for the spurious interrupt messages is that the CPU is enough faster than the 82586 chip that it can catch the chip unprepared. More precisely, suppose that there is an interrupt already pending from the 82586 and the driver presents the 82586 with a new command via the chip's CA line. The chip then responds by removing its interrupt request and clearing out the status bits in the SCB. Previous indications of the cause of the pending interrupt disappear.

The chip now examines the command that the CPU has placed in the SCB command word and starts to process the command. Finally, the chip resets the interrupt status bits in the SCB according to the new chip state, reactivates its interrupt request line, and clears the command word in the SCB. This assumes that at least one interrupt status bit is set.

In the meantime, the driver has finished and the CPU takes the interrupt originally presented by the 82586. When the driver gets control, the chip has already cleared the interrupt status bits, but has not yet set them again. The driver therefore finds no task to process and complains. Note that if there is a task to be done, the chip will reassert its interrupt request line. In this case, the driver's interrupt routine will regain control and will find correctly set interrupt status bits. The race condition and corresponding lost interrupt message from the driver should be harmless, though annoying.

Summary

The spurious interrupt messages result from a race condition between the ie driver and the 82586 chip.

Sun 3/50 EMT Faults and Errors

Sun 3/50 EMT Faults, Segmentation Faults, and Bus Errors

A combination of software bugs and a hardware bug may result in EMT faults, bus errors, or segmentation faults on Sun 3/50 workstations.[2] These bugs cause the process to die and a core dump. You may find it important to discover whether your SunOS release or the Sun 3/50 hardware bug is causing these errors or faults. They are otherwise common occurrences during program development.

Use this article for a history of the software bug fixes, a hardware bug description, how to determine whether you are running into the hardware bug, what to do about it, or whether the problem is caused by user error.

Hardware and Software Problem History

A hardware bug has been found on the 3/50 CPU board, showing up as an EMT fault, bus error, or segmentation fault. The process dies and core is dumped for these large binary failures.

Software bugs also contribute to the problem. The SunOS release 3.0 kernel did not process some rare prefetch fault cases correctly. This also causes programs to dump core. SunOS release 3.2 contained an attempt to correct this kernel bug. The fix significantly reduced the number of bogus bus errors reported, but did not completely fix the problem. The complete fix is found in SunOS release 3.4. However, this fix does not prevent similar bogus bus errors occurring because of the hardware problem.

Hardware and Software Bug Troubleshooting

You can isolate the probable cause of user programs dumping core on a Sun 3/50 by first determining whether your hardware has been upgraded. See the following paragraphs. If not, the hardware could be at fault. However, if your are running a pre-3.4 release SunOS kernel, it could be the kernel mishandling certain fault conditions. Note that this also present but very rare for SunOS releases 3.2 and 3.3.

The easiest way to find out whether or not your are running into the hardware bug is to run the binary on some other Sun-3 machine, either a 3/75 or a 3/160. If the binary fails on the 3/50 and not on some other Sun-3 architecture, then you are likely running into the hardware bug.

You can isolate the problem in two ways if you do not have access to another type of Sun-3 machine.

First Rearrange the program `.o` files load order. The hardware bug is sensitive to certain instruction sequences falling on page boundaries. You put page boundaries in different places by loading the modules

[2] This continuing column is submitted by Chuq Von Rospach, Customer Software Services.

in a different order. This will cause the failure to either occur in a different place or to go away completely, depending on where the page boundaries fall.

Second

Look at the core file with `adb`. Your problem is the hardware bug if the failure is on a page boundary and the failed instruction involves an indirect addressing mode through a register. An example is shown below.

```
% adb prog core
core file = core -- program ''prog''
SIGEMT 7: F-line or A-line op
```

If you dump the registers, you should see something that looks like this:

```
$r
d0  0xefffc6c
d1  0x7ff0a6eb
d2  0x1
d3  0x0
d4  0x0
d5  0x0
d6  0x32
d7  0x19
a0  0x88d60013
a1  0x1
a2  0x0
a3  0xefffcb4
a4  0xefffcbc
a5  0xefffc9c
a6  0xefffb78
sp  0xefffb70
pc  0x13bffe  _foo154+0x28
ps  0x0
    0x0
_foo154+0x28:       jsr    a0@
```

The failed instruction is at 0x13bffe, crossing a page boundary (suspect addresses would be xxxfx), and the failed instruction is a jsr indirect through a0.

The Hardware Fix

There is no software workaround for this hardware bug. You can try to keep it from happening or reduce the error frequency by rearranging the `.o` file load order into the binary as described above. However, this only changes the instructions that are in the page boundary. It does not fix the failure, but may simply move the failure to a less-used part of the program.

Replacing the CPU board is the only fix. If you are under a hardware maintenance contract, please call the Sun Hotline and log a service call on your Sun 3/50. Reference field alert number 88. Hardware technical support will then schedule your machine for the needed upgrade. If you are not under contract, Sun will replace the CPU board at standard charges.

The Software Fix

The surest way to avoid the bogus bus error messages is to run a SunOS release 3.4 kernel. You can otherwise avoid most such errors by running SunOS release 3.2 or 3.3 kernels.

Scratch or Keep Register D2

dbxtool and Child Processes

You may find that debugging programs with `dbx` is difficult when the program `fork()`s and thereby creates child processes. Debugging can be done, but it does not fit into `dbx` nicely. You will have to change the source code during debugging.

Use the steps below and the `dbxtool` to debug programs that create child processes.

1. Insert a `sleep(20)` or a similar call in the child process path of the `fork()`ed code. This delays the child code execution. Other helpful uses include the parent process now being able to print the Process Identification (PID) of the new child process, store it as a variable for `dbx` to print, or display the PID.

2. Link the program using the `-N` flag. This causes the executable to be non-shared. `dbx` will otherwise return a message that it cannot write to process ???? (Already debugging this file) when you execute the `fork()` call.

3. Start `dbx` on the parent process. Put a break point on the parent process code as you need. You may find it helpful to put a break at the `fork()` point to obtain the child process PID.

4. Start another copy of `dbx`, or `dbxtool`, and enter the first part of a command as shown below.

   ```
   debug <executablefilename>

   <DO NOT PRESS RETURN, YET...>
   ```

5. Start parent process code execution. Obtain the child process PID number when the `fork()` call is encountered. We will use 1234 as the PID in this example.

6. Now complete the command as shown below.

   ```
   debug >>filename<< 1234 <RETURN>
   ```

   This command starts a second `dbx` process to debug the child process suspended earlier by the `sleep(20)` command. A `step` command now allows you to debug the child process 20 seconds after the `fork()` call.

7. You may want to trace one of the `exec(2)` calls executed by most child processes. The PID remains the same, but the executable image changes. Use the following commands in this case. Note that the child process will now execute at full speed.

```
detach
debug >>new executable image<< 1234
```

You can now see how useful is is to alter the child process code by adding a `sleep` command to trace both `fork()` and `exec(2)` calls.

Supported Device Drivers

Sun Workstation Device Drivers: What We Support, Common Questions, and Answers

Sun's Customer Service Division (CSD) refers many customers with device-driver questions to the Consulting Services group. Complicated questions that are beyond standard support are most often referred to Consulting Services.

What Consulting Services Can Do

Our Consulting Services group answers questions which fall into the categories shown below.

- Undocumented software and features
- Questions about hardware architecture
- Questions about software sources
- Questions about driver `select` and `mmap` routines
- Complicated memory management issues
- Device driver design issues
- Drivers which require a knowledge of, and access to, source code including
    - block device drivers
    - sophisticated `pixrect` drivers
    - network interface drivers
    - coprocessor drivers
    - serial communications multiplexers

Writing a device driver requires some knowledge of UNIX internals, fluency in the C programming language, and familiarity with Sun hardware and software.

Common Device-Driver Questions and Answers

An overview of customer concerns about device drivers can be gained by looking at the most commonly asked questions and their answers. If you have a question that remains unanswered, call your sales representative or your Answer Center.

- What is the difference between Direct Memory Access (DMA) and Direct Virtual Memory Access (DVMA)?

  DVMA is Sun's term for DMA. The difference between DMA and DVMA is that DVMA goes through the Memory Management Units (MMUs).

- Can a `vme16d16` device do DVMA?

  No. At least 20 bits are needed to perform multibus DVMA, and 24 bits to perform VMEbus DVMA.

- Does Sun conform to the VMEbus standard?

  Sun follows the Motorola VMEbus specification. Many customers are developing drivers for VMEbus devices. A useful document is the *User's Guide to the Sun-3/100 VMEbus*, part number 800-1487, which may be obtained through your Sun sales representative.

- At which bus grant level should a VMEbus device be set?

  Bus grant level 3.

- What about VMEbus address modifiers? How do you tell the Sun to generate the correct VMEbus address modifier?

  From the device end, jumpers are usually set on the board. You can also write the address modifier to the board via software.

  If the driver resides in the kernel, the address modifiers telling the board which addresses to respond to are normally 0x0D (hex) for `vme32` devices, 0x2D (hex) for `vme16` devices, and 0x3D (hex) for `vme24` devices. The Sun workstation automatically generates the correct address modifier since the kernel configuration `/dev` file is already opened. This file tells the system the address space in which the device resides.

  If the driver is a memory-mapped user process, the device must work with Sun CPU-generated user function codes. Appropriate address modifiers are 0x09 (hex), 0x29 (hex), and 0x39 (hex) for `vme32`, `vme16`, and `vme24` devices; respectively.

- Is there significant overhead in calling the kernel support routine `mbsetup()`?

  No. `mbsetup()` sets up the memory map for a single, main-bus DVMA transfer. A common misconception is that two data transfers take place during DVMA: first, from the user address space to the kernel address space; and second, from the kernel address space to the device. This is not the case. `mbsetup()` performs the mapping between the user address space, the kernel address space, and the physical device. Only one transfer takes place at DVMA time. `mbsetup()` itself does not transfer any data.

- What happens if the flags argument to the kernel support routine `mbsetup()` is 0, and there is no DVMA space?

  The requesting process will sleep until the resource becomes available.

- Is there a functional difference between the `MDR_BIODMA` and `MDR_DMA` flags of the `mdr_flags` field element within the `mb_driver` structure?

  No. The autoconfiguration software does not distinguish between the two flags as of SunOS release 3.2.

- What is the last argument `(off)` to the `mmap(2)` system call?

  The last argument to `mmap` is the 'offset' into the device you have mapped. This is usually the physical address jumpered on the board. This is also the address which would be specified in the kernel configuration file as the `csr`, if a kernel device driver were being written for the device.

- What if the device can only be addressed for a hardwired, pre-determined value?

  This addressing limitation may cause difficulty since your board might conflict with other Sun devices. For example, if your board is a `vme24` device whose physical address is hardwired for 0x200000 (hex), there will be a conflict with the Small Computer Systems Interface (SCSI) controller. If your board is a `vme32` device which can only be addressed at 0x400 (hex), there will be a conflict with the DVMA area. On the other hand, you can have a `vme16` device at physical address 0x400 (hex), but it can not do DVMA.

  Chapter 2 of *Writing Device Drivers for the Sun Workstation*, part number 800-1304, for SunOS release 3.2, lists the physical address ranges for each type of multibus and VMEbus device. This information can also be obtained from the `/usr/sys/conf/GENERIC` kernel configuration file, as well as from the *Sun Microsystems, Inc. Configuration Guide, Sun-3 Product Family, October 1986*, no part number.

- What about devices which use two address spaces? For example, how do you handle a device which has its memory in `vme24d16` space and a register in `vme16d16` space?

  A detailed explanation on dual-address space drivers under SunOS release 3.2 is found on page 125 of *Writing Device Drivers for the Sun Workstation*, part number 800-1304.

- What are the hardware addresses of the MMU tables and registers, the segment map, and the context register?

  You should not need this information to develop a device driver for the Sun workstation. However, if you feel that you need this information, it appears in a proprietary architecture manual. Under special circumstances, your sales representative may arrange for you to obtain proprietary manuals on a case-by-case basis. To obtain such manuals, you would need to sign a non-disclosure agreement, which would also be signed by a Sun Microsystems, Inc. Vice President.

- How do you perform an interruptible sleep in a device driver?

  For SunOS release 3.2 and beyond, include the code shown below to set up an interruptable sleep.

  ```
   if (sleep(addr, priority|PCATCH))
   {
  tell hardware to abort any I/O
  ...
  return(EINTR);
   }
  ```

  This code requires that `priority` is greater than `PZERO (25)`. If you do this, ensure that the I/O in process when you caused the interrupt will not eventually complete and unexpectedly start the interrupt routine of your driver.

Splitting PostScript Output

Splitting PostScript pscat Output for psview Processing

This article contains a sample program to create files so that you can preview selected portions of your `troff` output.

Introduction

When working with NeWS/PostScript, NeWS executes PostScript, using `psview` to process the plain PostScript data to NeWS. If `pscat` output is to be directed to `psview`, it must be sent as one page at a time, or `psview` overwrites the pages.

The following program, `pscat split`, can be used to facilitate splitting `pscat` output into separate files, each containing a single page. These individual pages can then be directed to `psview` for processing to NeWS.

This program is useful when previewing selected `troff` output pages. It also allows the user to send single pages to a laser printer without printing the entire file or instructing `troff` to output specific pages.

Usage Notes

The `pscat split` program separates pages by inserting page numbers. The program uses these numbers to name each single-page output file. These numbers do not necessarily match the page numbers that appear on the `troff` output pages. If the `troff` page number register is changed in the file, it is not reflected in the output page numbering.

The `pscat split` program takes the name of the file to be split as a single argument, shown below.

```
split.c <filename>
```

The output files produced by `pscat split` are named as shown below.

```
<filename>.<pagenumber>
```

Thus, if the file named **document** is split into five output pages by `pscat split`, the output pages are named as shown.

document.1
document.2
document.3
document.4
document.5

The pscat split Program

```
echo x - split.c
sed 's/^@//' > "split.c" <<'@//E*O*F split.c//'
#include <stdio.h>
#include <ctype.h>
```

```
#define LINEMAX          256
#define ENDPROLOG  "%%EndProlog"
#define PAGEMARK   "%%Page:"

FILE *prelude;
char linebuff[LINEMAX];
char *tempname;

main(argc, argv)
        int argc;
        char **argv;
{
        FILE *file;

        if (argc != 2) {
                fprintf(stderr, "Usage: %s filename0, *argv);
                exit(1);
        }
        file = fopen(argv[1], "r");
        if (file == NULL) {
                perror(argv[1]);
                exit(2);
        }

        copy_prelude(file);
        while (copy_page(file, argv[1]))
                ;;
        fclose(file);
        unlink(tempname);
}


copy_prelude(file)
        FILE *file;
{
        int EP_len = strlen(ENDPROLOG);
        int PM_len = strlen(PAGEMARK);
        char *mktemp();

        tempname = mktemp("/tmp/preludeXXXXXX");
        prelude = fopen(tempname, "w");
        if (prelude == NULL) {
                perror(tempname);
                exit(3);
        }
        while(fgets(linebuff, LINEMAX, file) != NULL) {
                fputs(linebuff, prelude);
                if (strncmp(linebuff, ENDPROLOG, EP_len) == 0) {
                        break;
                }
        }

        freopen(tempname, "r", prelude);

        if (feof(file)) {
                return;
        }

        while(fgets(linebuff, LINEMAX, file) != NULL) {
                if (strncmp(linebuff, PAGEMARK, PM_len) == 0) {
                        break;
                }
        }
}


copy_page(file, filename)
        FILE *file;
        char *filename;
{
        int PM_len = strlen(PAGEMARK);
        char *pg_no;
        char *pageno();
        FILE *page;
        char pagename[LINEMAX];

        if (feof(file)) {
                return 0;
        }

        pg_no = pageno(linebuff);
        strcpy(pagename, filename);
        strcat(pagename, pg_no);

        page = fopen(pagename, "w");
        if (page == NULL) {
                perror(pagename);
                exit(4);
        }

        rewind(prelude);
        while(fgets(linebuff, LINEMAX, prelude) != NULL) {
                fputs(linebuff, page);
        }

        while(fgets(linebuff, LINEMAX, file) != NULL) {
                if (strncmp(linebuff, PAGEMARK, PM_len) == 0) {
                        break;
                }
                fputs(linebuff, page);
        }

        fclose(page);
        return 1;
}


char *
pageno(buff)
        char *buff;
{
        char *rover = &buff[strlen(buff)-1];

        if (*rover == '0) {
                *rover = '  ';
                rover--;
        }
        while (isdigit(*rover)) {
                rover--;
        }
        *rover = '.';
        return rover;
}
@//E*O*F split.c//
chmod u=rw,g=rw,o=r split.c

exit 0
```

3

STB SHORT SUBJECTS

3

STB SHORT SUBJECTS

Your SunOS Release Level

What SunOS Release are You Running?

Customers may need to know exactly which SunOS release they are running when a problem occurs. The AnswerLine people often need this information to help customers find a solution to a particular problem.

There are a variety of ways to determine your SunOS release. One commonly used method is the command shown below.

```
     cat /etc/motd
```

This command displays the current contents of the message-of-the-day file. This is fine in many cases since the SunOS release level is usually indicated in this file.

However, in cases where the customer has modified the mechanism in `/etc/rc.local` which loads `/etc/motd`, the release level may be inaccurate, or may be missing completely. For this reason, a more reliable way to determine your SunOS release level is to use the following pipeline.

```
     strings /vmunix | grep UNIX
```

The above displays the version string from the disk image of the kernel always booted under normal conditions. Unless you are running a specially named kernel for debugging purposes, the above sequence will indicate the release level for the system you are actually running at the time the command is executed. This is the surest method of knowing what SunOS release you are using.

Make a note of your SunOS release level when you next call your AnswerLine. The information will then be up-to-date and handy when needed.

Submitting Software Bugs: U.S.

Submitting Software Bugs: U.S.

This article contains information on reporting bugs within the U.S., for customers holding and not holding support contracts. International customers should report bugs through their respective local support groups.

Sun's U.S. Answer Center within the Customer Services Division (CSD) accepts software bug reports from Sun users via electronic mail and by phone. The method you use to submit a bug report varies with your needs.

U.S. users holding support contracts can report bugs to the Sun U.S. Answer Center via the **(800) USA-4-SUN** phone hotline. The U.S. Answer Center phone hotline is the fastest way for a customer to find out if a problem is known and if a workaround exists. The status of previously-reported bugs can also be obtained in this way. The list of open software bugs is contained in the Customer Distributed BugsList (CDB) and appears on a quarterly basis as a part of this bulletin.

Customers holding support contracts can also submit bug reports electronically to the address *sun!hotline* (*hotline@sun.COM*). This method generates a service order, and can be used when lines of code or other information difficult to relay over the phone is needed to describe the bug.

Customers who do not hold Sun software support contracts can report bugs via electronic mail to the address *sun!sunbugs* (or *sunbugs@sun.COM*). These reports are reviewed periodically to determine proper disposition. Those reports determined to be from supported customers are forwarded to the U.S. Answer Center for handling. Reports from customers who cannot be verified as holding a support contract are reviewed by Sun's Software Quality Assurance (SQA) personnel. An internal bug report is generated if the reported bug is new and verifiable.

Summary

For U.S. contract customers, **(800) USA-4-SUN** is the best method to report bugs. The electronic mail address *sun!hotline* is also available to report less time-critical bugs, and for submission of materials that are difficult to relay over the phone.

For non-contract customers, the electronic mail address *sun!sunbugs* is available to report bugs.

To help us serve you better, please include the following information with all electronic mail reports:

- Your name

- The name and address of your organization
- Your Sun site code, if available
- Your workstation model and serial number
- The software release(s) you are running
  (See the preceding STB article!)
- A description of the problem that you are experiencing

The 'setlinewidth' Operator

The NeWS PostScript setlinewidth Operator

This article discusses using the NeWS PostScript `setlinewidth` operator.

In the NeWS PostScript documentation, the `setlinequality` operator-extension controls the following line quality attributes:

`line width`	specified in pixels
`line join`	miter/round/bevel
`line endcap`	butt/round/projecting-square

If `currentlinequality` is 0, the fastest and easiest method to use to set all attributes is to specify a single pixel `line width`, a miter `line join`, and a butt `line endcap`. With a single pixel line width, specifying any other option types for `line join` and `line endcap` does not make sense. A `linequality` of 1 offers full use of the above line attributes. The default `linequality` is 0; hence, this should be set to 1 to obtain the full line attributes, if needed.

This operator was implemented for performance reasons, since single-pixel lines can be rendered much faster than lines which implement other line attributes.

'gettytab' and Modem Baud Rates

gettytab Entry for 2400/1200/300 Rotary Dial-up Line

Periodically a 2400 baud modem `gettytab` entry needs to include additional entries for 1200 baud and 300 baud rates.

The following is the correct `gettytab` entry for a 2400/1200/300 rotary dial-up line. The actual entries can be specified in ascending order (300, 1200, and 2400 baud rate) or in descending order (300, 1200, and 2400 baud rate). The **x** entry for 2400 baud rate is then used in the `ttys` file.

The gettytab Entry

```
#
# Fast dial-up terminals, 2400/1200/300 rotary
#
x|X2400|XFast-Dial-2400:     :ap:nx=X1200:tc=2400-baud:
y|X1200|XFast-Dial-300:      :nx=X300:tc=1200-baud:
z|X300|XFast-Dial-300:       :nx=X2400:tc=300-baud:
```

Verifying Dump Tapes

Dump Tape Verification Methods

The following two methods can be used to check dump tapes to verify that each tape is in good shape, and that the data contained on each tape is in a readable format.

The following three-step procedure is the preferred method to use.

1. Perform a dump-to-disk file.
2. Use `dd` to convert and copy the dumpfile to tape.
3. Use `cmp <disk_dumpfile> <tape_dumpfile>` to compare the two files. If `cmp` detects a difference between the files, the byte and line number of the difference is returned.

The following two-step procedure can be used only if filesystems are not mounted, or if the dump is being performed from single-user mode with no `u` dump option.

1. Perform a dump-to-tape.
2. Dump to standard output device and pipe the output to `cmp - <tape_dumpfile>` to compare the two files.

'at' Usage and .cshrc Shells

Structuring the .cshrc Shell Script for 'at' Usage

This articles contains information on environment variables set up by `at` using a Bourne shell.

How 'at' Works with Shell Scripts

When `at` executes a designated shell script at the prescribed time, it first sets up all the environment variables that were defined in the user's environment when the `at` job was submitted, using a Bourne shell. After the environment variables are set up, it passes the script to the designated $SHELL. A specific shell can be selected with arguments to `at`.

If the shell is `.csh`, the first function performed by `.csh` is to run the `.cshrc` file. To increase speed and prevent error messages from shells started with no `tty`, most users protect the contents of the `.cshrc` file as shown here.

```
if ( $?prompt ) then
        .
        .
        .
endif
```

Modifying the .cshrc Shell Script

Keep in mind that the shell started by `at` does have a prompt--the `systemname%`--but does not have a `tty`. To modify the `.cshrc` shell script to be properly executed by `at`, test `$term` before using `stty` or a similar command. A suggested way to structure the `.cshrc` file is shown below.

```
.
.                  # Commands to be run for every
.                  # csh startup are entered here
.
if ( $?prompt ) then
        .
        .          # Commands for interactive shells
        .          # (aliases, path definition, etc.)
        .          # are entered here
        .
        if ( $?term ) then
                .
                . # Commands affecting the
                . # terminal are entered here
                .
```

```
  endif
endif
```

Assigning Port Numbers

How Port Numbers are Assigned

Every machine has one or more internet addresses in the form `xx.xx.xx.xx`. One internet address exists for each network interface. This serves to identify the source/destination interface, but does not identify the source/destination application on the machine owning the interface. The port number performs this role. The port number can therefore be thought of as the application's address on the given machine.

When a socket is created, the means are established to send or receive messages (data) via that socket. The `bind` is used to establish the port number to the socket. In some cases, a 'transient' port number is automatically assigned to the socket.

Standard port numbers are assigned to generic services, such as `ftp`, `telnet`, `rlogin`, and so on. These port numbers are the values included in the `/etc/services` file. Transient port numbers may also be assigned at random for a given instance of an application. `RPC` does this, transparent to the user. `Portmap` itself has a well-known standard port number, 111, but all other services are assigned transient port numbers. `Portmap` keeps track of the dynamic mappings between `RPC` program numbers and port numbers. The output of `rpcinfo -p` shows the port numbers.

'uucp' and Line Speeds

uucp and Different Line Speeds

Different line speeds may be used with `uucp`. Specify the line speed in the fourth field of each entry in the `/dev/L-devices` file for special devices. Check that a device to be used at that speed appears in the `/usr/lib/uucp/L-devices` file.

Also, the fourth field of entries in the `/usr/lib/uucp/L.sys` file is the class, usually the line speed for the call. There is an entry for each system to which you dial out.

See the `uucp` tutorial in the *System Administration for the Sun Workstation*, part number 800-1323. The tutorial appears in appendix c, 'UUCP Implementation Description,' pages 317 - 322.

'asm' Embedding Assembly Code

Using asm to Include Assembly Code in a C Source File

The asm call allows assembly code to be embedded in a C source file. The argument to asm is a string corresponding to an assembler instruction. The following example shows how the asm call can be embedded. In this example, asm achieves the effect intended by the commented-out assignment statement.

```
#include <stdio.h>

main(argc, argv)
int argc;
char **argv;
{
  int i;
  /*
  i=3;
  */
  asm("movl #0x3,a6@(-0x4)");
  printf("%d0,i);
}
```

4

QUESTIONS, ANSWERS, HINTS, AND TIPS

4

QUESTIONS, ANSWERS, HINTS, AND TIPS

Q&A, and Tip of the Month

Hints & Tips #3

This is the third in a continuing series of this column which I have created for two purposes.[3] First, some questions are asked regularly on the AnswerLine. I feel everyone can benefit from distributing discussions of these problems as widely as possible. Second, a large and constantly growing body of information, hints, and tips are not documented anywhere.

I will collect and distribute these information nuggets in this continuing column so that we can all learn from them. I will cover unusual topics, but this column should not be used as an alternative to contacting your support center or using the AnswerLine.

If you have a question that you would like answered in this column, please mail your question to 'Software Technical Bulletins' at Sun Microsystems, Inc., 2550 Garcia Avenue, M/S 2-34, Mountain View, CA 94043. You can also send in your question by electronic mail to *sun!stb-editor*. U. S. customers can call Sun Customer Software Services AnswerLine at 800 USA-4-SUN for technical questions on this column or any other article in this bulletin. I look forward to hearing from you!

What Does 'Hayes-Compatible' Really Mean?

This month I'm going to talk about a growing problem area we're seeing in Customer Software Services (CSS), and then add some enhancements to last month's folklore.

CSS has received a number of service calls regarding `uucp` and the new, high-speed error-correcting modems now used by many customers. Most of these modems are 'Hayes-compatible' according to their manufacturers, yet `uucp` will not work on them in many cases.

[3] This continuing column is submitted by Chuq Von Rospach, Customer Software Services.

Customers need to be aware that compatibility is loosly defined, especially when buying electronic equipment. To many manufacturers, 'Hayes-compatible' is little more than supporting a subset of the standard Hayes command set. Not all modems support all of those commands that the Sun `uucp` driver needs to dial the modem properly. Thus, *let the buyer beware!* Remember that the final arbiter of compatibility is whether the modem works in actual use, not manufacturer or retailer representations.

The Problem

Fortunately, compatibility problems arising from unsupported Hayes commands are rare. More insidious are problems arising in other areas. For example, the major problem with high-speed modems is the connection with the host computer. In most cases this is a 9600 baud connection using software flow control (^S/^Q). Even if you set the modem connection to only 1200 baud, the physical connection between your computer and the modem remains at 9600 baud.

`uucp` accesses the modem in raw mode to transfer binary data. The software flow control is ignored in the case of a port opened in raw mode. Worse, `uucp` reads the software flow control as data. There is no way to tell whether a '^S' or '^Q' comes as data from the `uucp` program on the other end or whether it comes from the modem at your end. This causes the `checksum` on the packet to fail which causes the entire conversation to fail.

Observing the Problem

Is is fairly easy to determine whether your failure is a software flow control problem. In this case, `uucp` is able to log into the remote system and will usually do a protocol negotiation. The failure typically occurs in the first packet of the first file that `uucp` tries to copy. This is the first time that a block of data is sent that is large enough to cause the modem to try to stop the data transfer with a '^S.'

You can see the modem attempt to stop the data transfer by watching `uucico` using the debugging flag `-x9`. You will observe the `login` sequence, `uucp` choosing a protocol, agreeing to send a file, followed by a series of alarms, and ultimately the conversation failure.

The Workaround

There is no practical way to change `uucp` to use the software flow control. The raw, eight-bit data path is too much an integral part of the program design. The only way you can get `uucp` to work with such modems is to disable the software flow control. Use either a form of hardware flow control, or run the modem at a slower speed that does not require the software control. This may require adjustment down to 1200 or 2400 baud.

Tip of the Month (TOM)

This month's tip expands on the alias discussion included in earlier April STB Questions and Answers.

Alias Change Overview

This month I'm including some improvements in the `.cshrc` aliases I published in Hints & Tips #1 in the April 1987 STB. These aliases allow you to put part of your current environment into the window namestripes of your `shelltool` or `cmdtool`.

The specific changes made in the code appearing at the end of this article are summarized below.

`setbar`	`setbar` is changed to let your home directory show simply as . If you change `setbar` to turn off file globbing, it no longer expands to the full pathname, saving space in the namestripe.
window test	The test to determine whether you are in the window system is improved and simplified. The previous test is overly-complex. An environment variable WINDOW_PARENT is set only when you are in the window system, giving you a foolproof check. The previous test gives you bad results in different ways, such as `rloging` onto another machine if you were not in the window system, but were on the bitmap screen.
lengthy `.cshrc`	Some programs such as `at` set `prompt` inappropriately. You need to verify that both `$prompt` and `$term` are set to protect yourself from executing the `.cshrc` commands when inappropriate.
documentation	The aliases are now documented reasonably including comments as shown in the following paragraphs.

Next month, I hope to build on this `.cshrc` file with other interesting and useful commands and aliases, ending with a comprehensive, documented `.cshrc` file.

New and Improved Aliases and Comments

Add the following comments and commands in your `.cshrc` file for improved window system namestripe and test performance.

```
#! /bin/csh

# If we are running a script, do not source .cshrc for speed. $prompt is
# set if we are interactive. $term is set if we have a tty attached.

if (! $?prompt) exit
if (! $?term) exit

# This sets things up so if you are running SunView the hostname
# and directory stack are in the window stripe. If you are not running
# SunView, it then places the current directory and hostname in the prompt.

set tty = 'tty'
set hostname = 'hostname'
set console = '<< CONSOLE >>'
set promptchar = "%"
set lastdir = $cwd
```

```
if ($?WINDOW_PARENT) then        # active if you are in SunView
   if ($?NEWSSERVER) then              # NeWS is running
      alias iaps "telnet 'hostname' 2000    # alias for interactive PS
      goto getout                    # ACK! Gotos!
   end
   alias hdr  'echo -n "]l*
   alias ihdr 'echo -n "]L*
                                # setbar actually does the work
   alias setbar 'hdr "${hostname}: 'dirs'"; ihdr $cwd:t'

# for rlogin -- if the hostname of the window is not the hostname of hostname
# for my machine, then put the hostname in the icon stripe instead of the
# directory. Note that the default hostname is hardcoded, so this fails if
# you run SunView on multiple machines, but I do not know a way around that.

   if (${hostname} != plaid) then
      alias setbar 'hdr "${hostname}: 'dirs'"; ihdr ${hostname}'
   endif
# HACK ALERT. If the window is on ttyp0, assume it is the console, and flag
# it as such.  This assumes (1) that you start your console window first in
# your .suntools file, (2) that nothing else is taking up ttyp0 when you start
# the window system, and (3) that nothing else unusual is going on.  There is,
# unfortunately, no way to find out if a given window is a kernel short of
# looking in kmem, so we have to guess.  This hack works
# most of the time.

   if (${tty} == "/dev/ttyp0" && ${hostname} == plaid) then
      alias setbar 'hdr "${console} 'dirs'"'
   endif
   if (${tty} == "/dev/ttyp0") echo -n "]Lconsole
   set prompt = "${hostname}$promptchar "
else
# if you are not in windows, use the prompt.
   alias setbar      'set prompt = "${hostname}:$cwd$promptchar "'
endif

# All of the above just set things up.  These aliases do the work. '.' resets
# the window stripe if it needs it.  cd, pushd, and popd are the only commands
# that change the working directory and directory stack, so intercept them and
# make sure things get updated. Finally, run setbar to initialize things for
# the first time.

alias .      'dirs;setbar;jobs'
alias cd 'set lastdir = $cwd; chdir * ; setbar'
alias popd 'set lastdir = $cwd; popd * ; setbar'
alias pushd 'set lastdir = $cwd; pushd * ; setbar'
alias c- 'set x=$cwd; cd $lastdir; setbar; set lastdir=$x'
setbar
getout:
```

```
exit 0
```

Q&A Errata Corrections

April TOM: Errata Corrections

An STB reader has brought some needed corrections to the STB Editor's attention. These corrections appear below, and are incorporated into the expanded discussion of the same subject in the following article.

If you wish correct your April STB, please enter the following corrections on page 27 of the April STB.

1. Under the heading "**The Fix,**" change the line that was

   ```
   :%s/.*:.*:.*:.*//
   ```

   to the following line shown below.

   ```
   :%s/.*:.*:.*:.*//   [this assumes magic is set]
   ```

2. Under the heading "**Tip of the Month - (TOM),**" paragraph 2, line 2, change the expression

   ```
   ^v^[
   ```

   to the expression shown below.

   ```
   ^V^[
   ```

3. Same heading, in the sample program code, lines 3 and 4 are changed as shown below, along with additional indentation and another `endif` to match the added `if` coding. The complete, revised code to be shown in the April STB is as follows.

```
set tty = ‘tty‘
set hostname = ‘hostname‘
if ("$tty" != /dev/console && $?term ) then
  if ("$term" == sun || "$term" == sun-cmd) then
    alias hdr  'echo -n "^[]l*^[
    alias ihdr 'echo -n "^[]L*^[
    alias setbar 'hdr "${hostname}: ‘dirs‘"; ihdr $cwd:t'
    set prompt = "${hostname}% "
  endif
else
    alias setbar      'set prompt = "${hostname}:$cwd% "'
endif

alias .       'dirs;setbar;jobs'
alias cd "cd * ; setbar"
alias pushd "pushd * ; setbar"
alias popd "popd * ; setbar"
```

```
setbar
```

Coding Change Details

Problems arise when doing an `rsh` to a Sun with this code in the `.cshrc` file. The code is not C, but `csh`, using the '&&' and '||' syntax, and the semantics are not precisely similar.

In C, if the first part of an '&&' returns *false*, then the rest of that expression is not executed. In `csh`, the *entire* expression has variable substitution performed before *any* of the expression is evaluated. Using the original code, you would get a '*term undefined error*' since the term is undefined at the time the variable substitution takes place (when you `rsh`).

Second, note that the expression `${tty}` is now enclosed in double quotation marks. This avoids the `tty` program returning the string '*not a tty*' syntax error when you have run `rsh`.

5

IN DEPTH

5

IN DEPTH

ND Second Swap Space

Establishing a Second Swap Space for Network Disk Driver (ND) Clients

This article contains an eight-step procedure to create a second swap space for network disk driver (ND) clients. If needed, please refer to the first topic in the short subjects section of this STB issue to determine the SunOS release level you are running.

Introduction

When a network disk driver (ND) client requires additional swap space, the client's existing swap space can be expanded, or a second swap space can be established for the client.

This article describes step-by-step procedures to establish a second swap space. These procedures are appropriate for SunOS releases 3.0 or greater, which are reflected in all examples. Note that this article does not cover how to add a new client.

Overview

In theory, adding a second swap partition is not complicated. In practice, though, the procedure becomes cumbersome. Most of the work involved in establishing a second swap space is done on the server. Along the way, a special kernel is built, which is assumed to be done on the server. The steps are summarized below.

1. Halt the client with `/etc/halt`.
2. Edit `/etc/nd.local` on the server.
3. Run `/etc/nd` using `/etc/nd.local` as input.
4. Build a special kernel and install it in an appropriate place.
5. Enable booting of the special kernel.
6. Edit the client's `/etc/fstab` file.

7. `MAKEDEV nd2` in the client's `/dev` directory.

8. Boot the client.

Step One: Halt the Client

This step must be performed from the client. `halt(8)` usage is discussed in the system documentation such as the *Commands Reference Manual*, part number 800-1295.

Step Two: Edit /etc/nd.local

This is the first of two steps that allocate the extra disk space for the client, and is performed from the server. Note that the client will neither know about the extra disk space nor be able to use it at this point.

The following describes the `/etc.nd.local` editing process. For a complete explanation of the contents and format of `/etc.nd.local`, see 'Sun Network Disk Service,' section 2.2, chapter 2, 'Sun Network Services' in *System Administration for the Sun Workstation*, part number 800-1150, and `nd(8C)`, described in the *Commands Reference Manual*, part number 800-1295.

The additional swap space for an `nd` client must come from somewhere. The simplest ways for getting this space are by reassigning an unused but existing `nd` partition, or by making an unused but existing hard partition into `nd` partitions.

For example, assume there is a partition, `h`, on a second disk which is available for use. An ND client will be added to this partition, as well as a second swap space for one of the clients on the first disk. Three important points to remember when editing the `/etc/nd.local` file are listed below.

- Do not overlap soft partitions.
- They must always begin and should end on cylinder boundaries.
- Soft partitions must stay within the allotted hard partition.

The fifth and sixth arguments on a 'user' line in `/etc/nd.local` define the soft partition. The fifth argument is the beginning sector, and the sixth argument is the length in sectors. The first sector of the hard partition is sector 0. Together, these two arguments should add up to the beginning sector of the next soft partition. If the sum is greater, an overlap problem results. If the sum is less, disk space is wasted.

Use `dkinfo(8)` to determine the sector-to-cylinder ratio. All soft partition boundaries and sizes should be integral multiples of this number.

Convert the starting position of the last soft partition and its size into cylinder units. The sum of these two numbers plus the starting cylinder of the hard partition should be less than or equal to the starting cylinder of the next hard partition. Ideally, these two should be equal. If the sum of these numbers is greater than the beginning cylinder of the next hard partition, an overlap condition exists. Either the size of one of the soft partitions must be reduced, or the next hard partition must be moved or resized or both. Note that the procedure to move or resize the next hard partition does not appear in this article.

The following samples show how the `/etc/nd.local` file appears before and after editing. The changes are shown in **bold**.

In the edited version of `/etc/nd.local`, space has been added for the machine named **taylor**, and a second swap area has been given to the machine named **lombard**. Notice that the device number by which **lombard** will refer to this new swap partition is **2**, the third argument. For consistency, the version number should also be changed.

`/etc/nd.local` before editing:

```
#
# These lines added by the Sun Setup Program
#
clear
version 1
user 0 1 /dev/xy0f 0 10120 -1
user 0 0 /dev/xy0e 0 11040 -1
user columbus 0 /dev/xy0c 70840 16560 0
user columbus 1 /dev/xy0c 87400 34040 -1
user lombard 0 /dev/xy0c 121440 16560 1
user lombard 1 /dev/xy0c 138000 34040 -1
user broadway 0 /dev/xy0c 172040 16560 2
user broadway 1 /dev/xy0c 188600 34040 -1
user grant 0 /dev/xy0c 172040 16560 3
user grant 1 /dev/xy0c 188600 34040 -1
son
#
# End of lines added by the Sun Setup Program
#
```

`/etc/nd.local` after editing:

```
#
# These lines added by the Sun Setup Program
#
clear
version 1
user 0 1 /dev/xy0f 0 10120 -1
user 0 0 /dev/xy0e 0 11040 -1
user columbus 0 /dev/xy0c 70840 16560 0
user columbus 1 /dev/xy0c 87400 34040 -1
user lombard 0 /dev/xy0c 121440 16560 1
user lombard 1 /dev/xy0c 138000 34040 -1
user broadway 0 /dev/xy0c 172040 16560 2
user broadway 1 /dev/xy0c 188600 34040 -1
user grant 0 /dev/xy0c 172040 16560 3
user grant 1 /dev/xy0c 188600 34040 -1
user taylor 0 /dev/xy1h 0     16560  4
user taylor 1 /dev/xy1h 16560 34040  -1
user lombard 2 /dev/xy1h 50600 34040 -1
son
#
# End of lines added by the Sun Setup Program
#
```

Step Three: Run /etc/nd

This step continues client disk space allocation from step two.

Do this step from the server:

```
# /etc/nd < /etc/nd.local
```

Once this has been done, reboot the client. If the client comes up, fine; simply halt it again. If not, check to be sure that the size or location of its original root partition has not been changed without properly dumping and restoring.

Step Four: Build a Special Kernel

This step is performed from the server. See chapter 8, 'Configuring the System Kernel,' in *Installing UNIX on the Sun Workstation*, part number 800-1158, for guidelines and procedures. Install the kernel in the client's root partition or other appropriate place, depending on the system.

It is conceivable that all clients will be given a second swap space, in which case one kernel can be built for all clients of the same architecture. Then put it into the right `/pub` partition. More likely, though, this kernel will be booted specially from the client's root partition. The latter is assumed in this step and in step five.

Edit the `config` file for your client. Change the 'config' line so it appears as shown below.

```
config  vmunix  root on nd0 swap on nd0b and nd0c
```

Change the 'ident' option to something other than `GENERIC`. One possibility is `TURBO_ND`. Build the kernel as usual, then copy it to the client's root file system. Preferably, do this by mounting `/dev/ndl1` at some convenient place, and then by using `mv` or `cp` to relocate it. `ndl1` is used in this case, because the machine used in step three (**lombard**) is listed in `nd.local` as shown below.

```
user lombard 0 /dev/xy0c 121440 16560 1
```

The **1** appearing as the seventh field indicates that **lombard**'s root file system is accessible to the server as `/dev/ndl1`. **grant**'s root file system is referred to as /dev/nd13 on the server. See section 2.2 of *System Administration for the Sun Workstation*, part number 800-1150, for details.

Previous examples of this special config line have used nd1 and nd2 as the swap partitions. Although this is consistent with the client's way of naming these partitions, this usage causes /etc/config to create the file swapvmunix.c inaccurately. Calls to the makdev macro will contain incorrect minor device numbers, resulting in an unusable kernel. config assumes that a disk device has eight hard partitions (partition a through partition h) which are referenced by minor device numbers 0-7 on disk 0, 8-15 on disk 1, 16-23 on disk 2, and so forth. By default, config also assumes that swapping takes place on partition b of any disk. Although these conventions do not hold for ND, config assumes that the actual swap areas will be nd1b and nd2b, which would be referenced as minor devices 9 and 17. The client really needs to reference these areas as minor devices 1 and 2. By naming them as nd0b and nd0c instead of nd1 and nd2, config creates swapvmunix.c correctly, thus saving some editing work.

Step Five: Enable Booting of the Special Kernel

This step is performed from the server. There are two different methods used to enable booting of the special kernel on the Sun2 and the Sun3. Both are described below.

On a Sun2 client:

Assuming for example purposes that /dev/nd11 is mounted on /mnt on the server, copy /pub.MC68010/boot to /mnt/boot. Run the following command to install a boot track in the client's root partition.

```
# /usr/mdec/installboot /usr/mdec/bootnd /dev/nd11
```

This makes it possible to boot a special kernel. The client partition of the server must be unmounted before booting the client. This must be done manually since there is no way to boot this kernel automatically. By default, the machine boots the kernel in /pub. This functions properly, but the extra swap space goes unused. To boot the special kernel, use the string ie(,,40) as the response to the prom monitor prompt, as in the following example:

```
>b ie(,,40)
```

On a Sun3 client:

Reset the client's link in /tftpboot to point to ndboot.sun3.private. For example, assume that **lombard**'s internet number is 192.9.201.9. Enter the following on the server.

```
# rm /tftpboot/C009C909
# ln -s /tftpboot/ndboot.sun3.private /tftpboot/C009C909
```

lombard will now always boot the kernel in its own root partition automatically. There is no need to install a boot track or boot program for Sun3 clients.

Step Six: Edit the Client's /etc/fstab File

This step is done from the server, assuming the client's root partition is mounted on the server's `/mnt` directory. Add the following line to the client's `/etc/fstab` file so that when `swapon(8)` is run in `rc.local`, the extra swap area is made available:

```
/dev/nd2        0       swap    swap        0 0
```

Step Seven: MAKEDEV nd2 in Client's /dev Directory

With the client's root partition still mounted on the server's `/mnt` directory, enter the following in the client's `/dev` directory.

```
# cd /mnt/dev
# MAKEDEV nd2
```

Step Eight: Reboot the Client

Unmount the client's root partition if it is still mounted on the server, using `umount`. Refer to `mount(8)` in the *Commands Reference Manual*, part number 800-1295.

Reboot the client as described below.

On a Sun2 client:

```
>b ie(,,40)
```

On a Sun3 client:

```
>b
```

Run `pstat -s` on the client to check the swap space size. Refer to `pstat(8)` in the *Commands Reference Manual*, part number 800-1295.

Index

Revision History

Revision	*Date*	*Comments*
FINAL	June 1987	Fifth issue of Software Technical Bulletin (Software Information Services).