Technical Support Bulletin

Issue 1985-1
Issue of 10 May 1985

SYSTEM ADMINISTRATION **1**
1.1. Interpreting Bus Error Message 1
1.2. Panics from transient parity errors 2
1.3. 'No keyboard found' message when booting release 1.3 3
1.4. Color board in 1.2 and 1.3 is at different address 3

DISKS **5**
1.5. How media defects are handled by the SCSI and SMD controllers 5
1.6. 1.4 diag Disk Label Changes 7

UTILITIES **9**
1.7. Tips on using .cshrc and .login 9
1.8. The 'c' command in mail 9
1.9. Inter-record gaps when using tar to write a tape 10

IN DEPTH **11**
1.10. Changing the Size of a Network Disk Server's Partitions 11

Technical Support Bulletins are distributed to customers with software/hardware or software only support contracts. Send comments or corrections to RoseAnn Loughrey at Sun Microsystems, Inc., 2550 Garcia Ave., M/S 2-30, Mountain View, CA 94043 or by uucp mail to sun!rloughrey. Customers who have technical questions about topics in the Bulletin should call Sun Technical Support.

Technical Support Bulletin

Sun Microsystems, Inc.

Issue 1985-1
Issue of 10 May 1985

SYSTEM ADMINISTRATION

1.1) Interpreting Bus Error Message

Applies to: All Sun-2 machines

Bus error messages are divided into two portions, the hexidecimal bus error value, and an English translation of the fields. The hexidecimal value and the fields of the bus error register are defined as follows:

HEX	BIT	NAME	MEANING
0x01	D0	LPARERR	Parity Error Low Byte
0x02	D1	UPARERR	Parity Error Upper Byte
0x04	D2	TIMEOUT	Timeout Error
0x08	D3	PROTERR	Protection Error
0x80 Other bits are reserved.	D7	VALID	1 => Valid Page, 0 => Invalid Page

Bus errors are catagorized into four different types:

Parity errors (LPARERR and UPARERR) can occur only on read cycles from Sun memory boards. Since parity errors are detected quite late in the memory cycle, they actually abort the next cycle that recognizes bus errors. The 68010 address receiving the error is unrelated to the address of the unavailable parity error. Parity errors always indicate a failure of hardware, as opposed to software. See item 1.2 for more information.

TIMEOUT results from a reference that took too long to complete. This usually occurs when accessing non-existent devices on the multibus or VMEbus. This may be caused by an error in a user-written device driver, a malfunctioning device, a faulty CPU board, or a faulty backplane.

Protection error (PROTERR) means that the page protection bits or the PAGEVALID bit did not allow the operation attempted (e.g., writing to a read-only page). This may also be caused by an error in a user-written device driver, a malfunctioning device, a faulty CPU board, or a faulty backplane.

Page invalid (VALID = 0) means the page referenced did not have the PAGEVALID bit set. This may be caused by an error in a user-written device driver, or a malfunctioning CPU board.

The system translates bits into words for the user as shown in the table . The error message for 'bus error reg 0x4e98 <VALID,PROTERR>' is translated into binary as illustrated.

Hex	4	e	9	8
Binary	0100	1110	1001	1000
Bit	high			low

1.2) Panics from transient parity errors

Applies to: All Sun-2 machines

Sometimes a system will panic and die, complaining of a transient parity error in the following way:

Parity Error! Bus Error Reg 4e92<VALID,UPARERR>
Can't find parity error (transient?)
panic: parity error
.
.
.
dumping to dev 501, offset 27552

This message says that the system was just hit with a transient upper byte parity error.

. The number 4e92 is the contents of the bus error register on the cpu board when the parity exception occurred. As explained in item 1.1 above, VALID means the memory was mapped validly, and UPARERR means the bus error was caused by an upper byte parity error. (See item 1.1 for an explanation of the bus error causes).

The message is telling you that the logic on the processor board computed a parity bit for a read cycle which differed from the bit detected for the data from the P2 bus. This can be caused either by a bit really wrong in memory, CPU parity logic which is not working correctly, or a temporary error on the P2 bus possibly caused by a power fluctuation.

When UNIX gets a parity error, it doesn't know the address of the error. It immediately scans (reads) all of memory, looking for the parity error again, to determine whether this error was a hard memory failure, or whether it was transient. In this case, the error was transient *(''Can't find parity error (transient?)'')*, in other words, the error was not caused by ''a bit really wrong in memory''.

The rest of the message *''panic: ... dumping to ... ''* are the kernel's last words before it dies.

Look at the file */usr/adm/shutdownlog* to see how often this happens on your system. Since the system usually reboots, these parity errors often go unnoticed. However, they are logged in */usr/adm/shutdownlog*. If there are quite a few parity errors, and you have eliminated ''dirty power'' (voltage fluctuations and/or spikes) as a cause, you should contact field service for possible hardware replacement.

1.3) 'No keyboard found' message when booting release 1.3

Applies to: 1.3

There is a problem with the 1.3 release which usually manifests itself with the message *"No keyboard found"* when attempting to boot. The problem is not actually with the keyboard, but with the sample kernel configuration files in /sys/conf. Between release 1.2 and 1.3, there was a change in the required format of device lines in the config file. 1.3 includes a new /sys/conf/GENERIC which reflects this change, but the other sample config files for the model 120 (SDST120 and ND120) are not updated in 1.3 and still have the old (now incorrect) syntax.

Here is the 1.2 device line for zs1:

device zs1 at mb0 csr 0xeec000 flags 3 priority 2 # video ports

The device line for zs1 in the 1.3 release onwards is:

device zs1 at mb0 csr all virt 0xeec000 flags 0x103 priority 2 # video

This same problem occurs if a user tries to use an old pre-existing config file from 1.2.

The solution is to make sure that all 1.3 kernel config files are derived from the 1.3 GENERIC kernel config file. The errors in ND120 and SDST120 are fixed in 1.4.

1.4) Color board in 1.2 and 1.3 is at different address

Applies to: Sun releases 1.2 and beyond

Device cgone0, the Sun medium resolution Multibus color board, may not be jumpered to the address found in the /etc/config file. As a result, the device will not be found when the system boots release 1.3.

In device drivers released prior to 1.3, a list of addresses for the board was compiled into the driver code. The driver would first probe its list of addresses, looking for the board . If the probe routine found the board at one of those addresses, it would use that value as the address of the board. If not, it would try the value specified in the kernel configuration file.

The factory has shipped most Sun-1 color boards configured at **0xec000**, but the address specified in the configuration file was **0xe8000**. In releases prior to 1.3, the board was found anyway, because its address was one of those compiled into the cgone device driver.

Drivers released in 1.3 or greater do not have a pre-compiled list of addresses. They look for the board ONLY at the address specified in the configuration file. If your workstation can't find cgone0 at boot time, you can either put the address **0xec000** in your configuration file, or you can reset the board's address switches to reflect the address in the config file.

The following figure shows the jumpers necessary for configuring the Sun II Color Board for addresses **0xe8000** and **0xec000**. The block of jumpers is located on the lower left portion of the board, at position J2. **"In"** means to short-circuit the indicated pair of pins (for example 1 to 2) by inserting a shunt onto those pins. **"Out"** means that the indicated pair of pins should not be shorted. The jumper block maps to the high-order 6 bits of the board address.

Pin No.	11	9	7	5	3	1
Jumper Block	• •	• •	• •	• •	• •	• •
Pin No.	12	10	8	6	4	2
Address 0xec000	in	in	in	out	in	in
Address 0xe8000	in	in	in	out	in	out

Jumper Block J2

DISKS

1.5) How media defects are handled by the SCSI and SMD controllers

Applies to: All systems with disks

This article is intended to serve as a tutorial on how media defects are handled by the two disk interfaces which Sun uses. For a more general discussion of disk drives and file systems, see the *System Administration Manual,* which will start shipping with the Sun 2.0 Release Operating System.

SMD Controller Interface: The Xylogics and Interphase controllers are connected directly to the system bus (Multibus for model 120s and 170s), and communicate with the disk drive through an SMD interface, an interface that is specifically designed for controlling disk drives. The CPU writes commands directly to the SMD control registers, and the controller's processor carries out the specified operations by directly manipulating the SMD interface. This means that the device driver has a lot of knowledge about the specific nature of the disk drive, and the UNIX filesystem takes advantage of this to arrange data on the disk in a way that will provide fast access for reading, writing, and updating files.

SCSI Controller Interface: By contrast, the SCSI disks are insulated from the system. The SCSI board itself is actually a bus adapter; it receives commands on the system bus, and reissues them on the SCSI bus (Small Computer System Interface). The SCSI bus is a general interface for I/O on small computers, and has a very idealized view of devices. Each device is treated as a continuous stream of records, whether it is a disk, a tape, or a printer. The SCSI bus also has a rigid, and very different protocol for transferring information, compared to the system bus.

The Adaptec controller is connected to the SCSI bus, and handles all of the device-specific aspects of the disk. It communicates with the disk itself using the ST-506 interface, which is another interface (like SMD) adapted particularly to disks. The Adaptec converts the requests for absolute block numbers into specific cylinder, head and sector addresses, and keeps track of all the physical details. The result is that all commands from the CPU to the disk controller are addressed to this idealized device, and the filesystem knows nothing about the geometry and layout of the disk.

With that as background, we can discuss how media defects are handled.

SMD Media Defect Handling: In order to map out a sector on an SMD drive, *diag* (the disk diagnostic program) "zaps" the header for the bad sector, and make an entry in the bad block list in the alternate cylinders which says that cylinder/head/sector such-and-such is mapped to cylinder/head/sector so-and-so in the alternate cylinders. There is however, no actual pointer to the alternate sector in the sector which has been marked bad. The xy device driver reads in the bad block list at boot time from the disk, and handles the translations in software.

The defective sectors are "zapped" by writing out an illegal sector header there. When UNIX tries to read a part of the disk which has a mapped sector, it gets an error reported by the Xylogics 450 when it cannot find the mapped sector. Then UNIX looks in the mapping table stored in memory and sees if that sector is mapped. If it is, it starts up a one sector transfer on the alternate sector. When that completes, it will then continue the transfer back at the original location (if needed). Thus it is software driven and not very efficient. However, the actual number of sectors on a disk that are mapped vs. the number of good sectors is extremely small, so the overall inpact on performance is very small.

Sector Slipping on the SMD: There is a second way of dealing with media defects on SMD drives: sector slipping. In order for sector slipping to work, there has to be at least one extra sector at the end of each track. If a media defect is found during either the *format, fix,* or *scan* command, and the drive is configured for an extra sector, then *diag* will slip the bad sector. The sector with the defect is zapped by writing a special code to indicate it has been slipped (much as with mapping), and then the rest of the logical sectors for that track are slipped one sector further.

Sectors can not be slipped across track or cylinder boundaries on the SMD drives. Sectors can be slipped as long as there are spare sectors on the track. Any additional defects have to be mapped. As Sun configures its SMD drives, you can get no more than one extra sector per track, but it is rare to find more than one actual defect per track. If you find more than two, it's probably a cable problem.

When the controller comes to the expected location of a sector which has been slipped and fails to find a legal header there, it will allow up to one more rotation to find the sector it's looking for before issuing an error message. Since the sector is, in fact, immediately following its previous location, it only has to wait for one sector's worth of time. The entire operation is transparent to UNIX, so almost no time is lost.

Some problems with sector slipping:

1) Once it has taken place, it's transparent. Unlike mapping, it doesn't leave a list behind. System administrators will probably want to keep records of slipped sectors, so they can enter this information in *diag* next time the disk is formatted.

2) Even with *diag*, it's difficult to find the slipped sectors from a prior formatting. If, for example, there were a cable problem, then *diag* might slip many sectors which have no actual media defect. You would have to keep an eye on *diag* while it's formatting to see that this had happened, since there's no easy way to tell afterward.

Spurious sector slipping has almost no effect on performance, whereas spurious mapping is a big problem. Overall, the benefits outweigh the problems, so in the future, Sun will be shifting over to ship SMD drives configured for sector slipping. At that time there will be a *Tech Support Bulletin* item on converting SMD drives to support sector slipping.

SCSI Media Defect Handling: For the SCSI disk, a bad block isn't mapped to an alternate cylinder. If cylinder 6 track 3 sector 12 is bad, the controller marks the place on the disk where sector 12 should be as bad and instead lays down the format for sector 12 where sector 13 would be if there had been no bad spot. As defects add up, the sectors can be skewed across track and cylinder boundaries, and unless the controller knows how much slip there has been, it can waste a lot of time finding the actual location of the sector it wants. That's why the Adaptec controller does a quick scan of the disk when it first comes up. The controller divides the disk up into zones, and keeps a table of the total slip for each zone. When it seeks for a particular sector, it adds in the appropriate slip for each zone it has to go through, so that it ends up only a little off. The reason for not keeping a list of all the defect locations is simply one of board real estate. There is no way for the Adaptec to communicate this defect information to the file system, so a seek between "adjacent" sectors may actually cross cylinder boundaries, cylinder groups may actually start in the middle of cylinders. Hence, the UNIX filesystem's efforts to lay data out efficiently provide less of a performance benefit for SCSI disks.

The Adaptec controller can only do this slipping at format time, since it involves moving the location of all succeeding sectors on the disk. The defect list stored at the end of the disk is there solely for the use of *diag*. Once formatting is completed, the current version of the defect list is written out to the end of the disk. The next time you want to reformat the disk, you should start by using the *r* command to read the list into memory. Then use the *a* command to add to the list, or the surface analysis command to scan the disk for additional defects. Surface analysis will

update the version of the defect list in memory. Then format the disk again, and the updated copy will be used during the format process, and will be written at the end of the disk. If the formatting process is allowed to complete normally, the defect list on the disk will always correspond to the state of the disk.

Actually, if the defect list is entered correctly the first time, no additional problems should occur, except for occasional instances where, due to a sudden power problem, or the drive being knocked over, the read/write heads actually scrape the surface of the disk. Such "head kisses" should never take place during normal operation, so if additional defects start turning up on an SCSI disk, there is probably a problem with the controller, cables, or drive electronics.

1.6) 1.4 diag Disk Label Changes

Applies to: 1.4 and later releases

Diag has changed somewhat in the 1.4 release causing difficulties to occur when the Xylogics driver compares the geometries of same-type drives. This problem can arise when two same-type drives are attached to a single Xylogics board, one of which was labelled by a pre-1.4 *diag*, and the other by a *diag* of release 1.4 or later.

To determine the particular geometries, the driver looks for two numbers in the label of each drive. These figures reflect the number of cylinders actually used for data storage, **ncyl**, and the the number of alternate cylinders available, **alt**. In 1.4 *diag*, however, **ncyl** gets larger and **alt** gets smaller. Two disks of the same type, i.e. 0 for the old 65Mb disks, 1 for 130 Mb disks, and 2 for eagles, can therefore appear to be of different sizes. The xy driver gets upset by this and puts out an error message:

xy1: drive type 0. conflict with xy0.

The way to resolve this is to relabel the old disk with 1.4 *diag*. This is done by constructing a label identical to the old one, with the extra cylinders added at the end of certain partitions. The disks will once again be compatible, while the file system and other disk information remain unaffected. At some point, you will have to remake the file system if you wish to take advantage of that extra space.

For example, if you used the default disk label under 1.3 *diag*, you can boot 1.4 *diag* and relabel your disk with the new default label, without harming your file system. If, however, you customized your disk label, you need to add in the extra cylinders to partition **c** and the last partition on your disk. The following is an example of a label created for a disk under 1.3 *diag*:

```
id: <Fujitsu M2284/M2322 cyl 820 alt 3 hd 10 sec 32>
        Partition a: starting cyl=0, # blocks 15884
        Partition b: starting cyl=50, # blocks 33440
        Partition c: starting cyl=0, # blocks 262400
        Partition g: starting cyl=155, # blocks 96000
        Partition h: starting cyl=455, # blocks 116800
```

Now, the same label as it would appear under 1.4 *diag*, taking into account the extra data cylinder on partitions **c** and **h**:


```
id: <Fujitsu M2284/M2322 cyl 821 alt 2 hd 10 sec 32>
        Partition a: starting cyl=0, # blocks 15884
        Partition b: starting cyl=50, # blocks 33440
        Partition c: starting cyl=0, # blocks 262720
        Partition g: starting cyl=155, # blocks 96000
        Partition h: starting cyl=455, # blocks 117120
```

Note that the changes in partitions **c** and **h** in the 1.4 label cause them to have 320 more sectors than in the 1.3 label. This change occurs because 1.4 *diag* gives the disk one extra data cylinder:

1.4 ncyl - 1.3ncyl = 1 cylinder = 10 heads * 32 sectors/track = 320 sectors

UTILITIES

1.7) Tips on using .cshrc and .login

Applies to: All Sun releases

Some commands can cause trouble if they are placed in your .cshrc file. These commands, generally, are those which:

1) set a prompt, set mail variables or produce output,
   like echo and fortune
2) assume stdin or stdout is a tty, like tset, stty, or biff

For example, when programs like *vi*(1), *ftf*(1C), *rsh*(1), or *rcp*(1) need to find out what *.c expands to, they may start up a sub-shell (*sh*(1) or *csh*(1) depending on the user) with the command *"echo *.c"*. Setting a mail variable in .cshrc to notify you when you have new mail *(set mail=(180 /usr/spool/mail/$USER))* and then, from vi, typing *:e* * can result in vi trying to edit files called *"You"*, *"have"*, and *"mail"*.

You should not set your prompt, set the mail variable, or produce output from your .cshrc file **unless the csh is interactive.** You can check this by putting

```
if ($?prompt) then
        ... csh is interactive, set prompt,
        ... set mail variable, etc. here
endif
```

in your .cshrc.

Now, with the point made that "if ($?prompt) then" makes a safe .cshrc file, you should still think about which commands to put in .cshrc and which in .login.

Put things in .login that:

a)You want done only at login, not once for each invocation of csh (e.g. stty, tset)

b)Get carried automatically to forked c shells (e.g., setenv, set term, set path)

Put things in .cshrc that need to be done for each shell:

a)alias commands, set history, prompt, mail, notify, unmask....

Doing a tset or stty in your .cshrc will really cause problems when the csh is talking to a socket instead of a terminal. These commands need only be done when you login, so they should be moved from .cshrc to .login.

1.8) The 'c' command in mail

Applies to: 1.x

- The "c" command in mail is actually an abbreviation for "copy" and not for "change directory". The abbreviation for "change directory" is "cd" just like in the shell.

The copy command is the same as "save" except that the mail message will not be deleted when the "quit" command is given. (This is like setting the *"keepsave"* option; see the Mail User's Guide for more information.)

The Mail User's Guide, the help screen in mail and the *mail*(1) manual page all incorrectly state that "c" is the abbreviation for "change directory".

1.9) Inter-record gaps when using tar to write a tape

Applies to: All Sun releases

When using tar to write a 1/4-inch cartridge tape, there is an "inter-record gap" (IRG), a space separating each block of data and tape, between each write request. A gap occurs each time the tape drive does its "washing machine" act - write data, coast to a halt, back up, halt, start forward, get up to speed, write data. These gaps are fairly long. They are transparent to software, unlike the IRG's on 1/2-inch tape. For this reason you must read a 1/2-inch tape using the same tape blocksize at which it was written, whereas, you do not have this concern with 1/4-inch tape.

However, tar uses a default blocksize of 20. This leads to lots of inter-record gaps, and lots of wasted tape, especially on 1/4-inch cartridges.

To get the maximum amount on these tapes, you should specify a block size of 126 in the tar command. There is a *b* (block size) option in *dump* which should also be specified as 126. Besides the savings in space, the *tar, dump,* and *restore* commands will run about five times faster with *b 126* than if no block size is specified.

For example, if you are creating a tar tape on a SCSI drive of everything found in /usr/tmp, the command should be:

tar cbf 126 /dev/rst0 /usr/tmp

See the *tar*(1) manual page for further details.

Even at 126 blocks (== 63 KB blocks), the most a "20 MB cartridge" holds is about 15-16 MB. The "20 MB" is only possible if you write the entire tape in streaming mode, which a Sun doesn't do. These same basic factors apply to the 45 MB tape as well. Refer to 1.4 Upgrade Manual, section 2.4, page 24 to observe the procedure for this command.

Additionally, make sure that you are using Sun or Sun qualified tape cartridges. The cartridges that many people buy from catalogs come in various lengths shorter that Sun's own. Tape cartridges used by Sun are Scotch DC300XL's and are certified for 8000 BPI operation.

IN DEPTH

1.10) Changing the Size of a Network Disk Server's Partitions

Applies to: 1.x

Some sites want to make an **nd** server's root file system and/or paging area larger or smaller than it currently is. The basic process is summarized as:

- Prepare for repartitioning by backing up the entire system;
- Determine the necessary changes;
- Edit *nd.local* to change the nd partitions;
- Use *diag*(8) to change the label partitions;
- Boot mini-UNIX and restore the root and pub file systems;
- Boot UNIX and restore all affected nd partitions.

1. Prepare for Repartitioning by Backing-up the Entire System

1.1. Halt All Clients and Shutdown to Single-User Mode

Before starting, make sure that all the clients are halted. Take the server to single-user state by using */etc/shutdown*(8). Also stop the nd disk service function with */etc/nd soff* (see *nd*(8)).

1.2. Backup All File Systems

Make tape backups of all file systems on the disk using */etc/dump*(see *dump*(8)). These need to be level 0 dumps. Since this is a server, there are client partitions and the corresponding ndl devices must also be backed up. *If you do not understand the preceding sentence, do not proceed further without getting assistance from Sun Technical Support, or you may risk losing client file systems.*

For safety's sake, we suggest that an additional tape backup be made of critical user files using *tar*(1).

2. Determine the Necessary Changes

2.1. Record the Current Label-Partition Information

Use */etc/dkinfo* to display the current disk label, for example:

```
sunburst# /etc/dkinfo xy0
xy0: Xylogics 450 controller at addr ee40, unit # 0
836 cylinders 20 heads 46 sectors/track
a: 20240 sectors (22 cyls)
   starting cylinder 0
b: 32200 sectors (35 cyls)
   starting cylinder 22
c: 769120 sectors (836 cyls)
   starting cylinder 0
d: 359720 sectors (391 cyls)
   starting cylinder 57
```

```
e: No such device or address
f: No such device or address
g: 216200 sectors (235 cyls)
   starting cylinder 448
h: 140760 sectors (153 cyls)
   starting cylinder 683
sunburst#
```

Write it down or print it out in hardcopy form.

2.2. Record the Current Nd-Partition Information

You also need size and location information for the pub and client ndl partitions. Write down, or print out the contents of the */etc/nd.local* file, for example:

```
sunburst# cat /etc/nd.local
#       nd.local        4.6     84/02/06

#
#nd.local - net disk local initialization file
#
#...
clear
version 1
#
# These lines added by the Sun Setup Program
#
clear
version 1
user 0 0 /dev/xy0g 0 164680 -1
user 0 1 /dev/xy0d 0 359720 -1
user cosmos 0 /dev/xy0g 164680 27600 0
user cosmos 1 /dev/xy0g 192280 23920 -1
user capella 1 /dev/xy0h 0 16540 -1
user capella 0 /dev/xy0h 16540 43240 1
user celestial 1 /dev/xy0h 59780 16540 -1
user celestial 0 /dev/xy0h 76320 22080 2
user realworld 1 /dev/xy0h 98400 21160 -1
user realworld 0 /dev/xy0h 119560 21200 3
ether cosmos 8:0:20:1:1:12
ether capella 8:0:20:ff:ff:ff
ether celestial 8:0:20:1:1:AC
ether realworld 8:0:20:1:1:4f
son
sunburst#
```

The */etc/nd.local* file contains commands for */etc/nd*(8). See *nd*(8) for an explanation of the contents of */etc/nd.local.*

2.3. Study the Partitioning Scheme

Examine the information recorded in the above steps to see how the disk is laid out. Note that label partitions must start on cylinder boundaries. The xy0c partition maps to the entire disk and

should not be modified during this process. A server disk is usually divided into three label partitions:

a - server root
b - server paging
g - nd area

The nd area is further subdivided into pub, client, and/or ndl partitions as defined in */etc/nd.local.* Some sites define additional label partitions **d**, **e**, **f**, and/or **h**, usually for use as local file systems on the server.

In the above example, besides the standard partitions **a**, **b**, and **g**, additional label partitions **d** and **h** are defined. Both of these are used by nd. Partition **d** contains in its entirety the public nd partition ndp1. Partition **h** is divided into client root and paging areas for capella, celestial and realworld. The following table summarizes the relationship of these label partitions with nd partitions.

Label Partition	Nd Partition	Hostname
/dev/xy0d	/dev/ndp1	[public 1]
/dev/xy0g	/dev/ndp0	[public 0]
/dev/xy0g	/dev/ndl0	cosmos
/dev/xy0h	/dev/ndl1	capella
/dev/xy0h	/dev/ndl2	celestial
/dev/xy0h	/dev/ndl3	realworld

2.4. Plan Partition Changes

Decide exactly how the partitions will change, and how the change(s) may affect other partitions. For example, if we were only changing the size of partition **b** (server paging), it would not affect partition **a** (server root) since **a** appears before **b** on the disk, but it would affect all partitions following **b**. Suppose for discussion's sake that there were only partitions **a**, **b**, and **g**. If **b** is made larger, then **g** will have a higher-numbered starting cylinder and will be smaller, requiring one or more soft partitions within **g** to be made smaller. Conversely, if **b** is made smaller, then **g** will start at a lower-numbered cylinder and will be correspondingly larger, permitting one or more soft partitions within **g** to be correspondingly larger.

In this example, let us say that we want to make partition **a** larger by about 2 megabytes, partition **b** smaller by about 4 megabytes, and add the resulting surplus 2 megabytes to client celestial's root file system (/dev/ndl3) in partition **h**. First, we will adjust the starting cylinder number and size of label partitions by creating a new disk label using *diag*'s *partition* command. Then we will increase the size of celestial's nd subpartition within label partition **h** by editing */etc/nd.local.*

2.5. Calculate Label Partition Starting Cylinder and Size Values

Calculate the size and starting cylinder number of affected label partitions. The information in the disk label is expressed in terms of sectors and cylinders. Sectors are 512 bytes each. To find the number of sectors in a cylinder, multiply the number of heads (tracks/cylinder) by the number of sectors/track. The second line of *dkinfo* output provides the necessary information:

```
xy0: Xylogics 450 controller at addr ee40, unit # 0
836 cylinders 20 heads 46 sectors/track
```

This disk, an Eagle, has 920 sectors per cylinder (20 heads * 46 sectors/track).

For simplicity, it is best to make all adjustments to the disk partitions in terms of whole numbers of cylinders. We want to enlarge partition **a** by about two megabytes, or 4096 sectors. Round this off to 4 cylinders, or 3680 sectors (4 cylinders * 920 sectors/cylinder), which increases the size of partition **a** by 1884160 bytes (3680 sectors * 512 bytes/sector). This now gives partition **a** 26 cylinders (22 original + 4 just added), or 23920 sectors. Since partition **a** is the first label partition on this disk, we can have the next partition, **b**, starting at cylinder 26.

We also want to subtract 4 megabytes from the paging area, partition **b**. Round this up to 9 cylinders, or 8280 sectors (9 * 920). This decreases partition **b** by 4239360 bytes (8280 * 512), giving partition **b** 26 cylinders. The next label partition, **d**, can start at cylinder 52 (26 cylinders for **a** + 26 cylinders for **b**).

The difference of 5 cylinders, or 4600 sectors, will be added to celestial's root area in the next step. The starting cylinders for the remaining label partitions, **d**, **g**, and **h**, are simply offset by these 5 cylinders.

Here is what the resulting label will look like. Compare these with the original numbers. This information will be entered into the label using *diag* later (see below).

```
xy0: Xylogics 450 controller at addr ee40, unit # 0
836 cylinders 20 heads 46 sectors/track
a: 23920 sectors (26 cyls)
   starting cylinder 0
b: 23920 sectors (26 cyls)
   starting cylinder 26
c: 769120 sectors (836 cyls)
   starting cylinder 0
d: 359720 sectors (391 cyls)
   starting cylinder 52
e: No such device or address
f: No such device or address
g: 216200 sectors (235 cyls)
   starting cylinder 443
h: 145360 sectors (158 cyls)
   starting cylinder 678
```

Note that the starting cylinder numbers for label partitions **b**, **d**, **g**, and **h** changed according to the change in size of **a**, **b**, and **h**. The sum of the starting cylinder number of a partition and the size of that partition in cylinders must not exceed the starting cylinder number of the following partition.

2.6. Calculate Nd Partition Starting Sector and Size Values

Similarly, adjust the numbers in */etc/nd.local*. Note that starting offsets specified in */etc/nd.local* are given in sectors, not in cylinders as was used when setting up label partitions above. Here is what the resulting */etc/nd.local* will look like. Compare these numbers with those from the original */etc/nd.local*.

```
...
user 0 0 /dev/xy0g 0 164680 -1
user 0 1 /dev/xy0d 0 359720 -1
user cosmos 0 /dev/xy0g 164680 27600 0
user cosmos 1 /dev/xy0g 192280 23920 -1
user capella 1 /dev/xy0h 0 16540 -1
user capella 0 /dev/xy0h 16540 43240 1
```



```
user celestial 1 /dev/xy0h 59780 16540 -1
user celestial 0 /dev/xy0h 76320 26680 2
user realworld 1 /dev/xy0h 103000 21160 -1
user realworld 0 /dev/xy0h 124160 21200 3
...
```

3. Edit nd.local to Change the nd Partitions

These changes to */etc/nd.local* can be edited in at this time before halting the server to run *diag*.

4. Use diag to Change the Label Partitions

4.1. Running *diag*

Halt the server and boot *diag*(8):

```
sunburst# /etc/halt
Syncing disks ... done.
UNIX halted.
> b stand/diag
Boot: xy(0,0,0)stand/diag
Load: xy(0,0,0)boot
Boot: xy(0,0,0)stand/diag
Size: 34816+20480+1160 bytes

Version 2.3 84/09/25
Disk Initialization and Diagnosis

When asked if you are sure, respond with 'y' or 'Y'

Specify controller:
        0 - Interphase SMD-2180
        1 - Xylogics 440 (prom set 926)
        2 - Xylogics 450
        3 - Adaptec ACB 4000 - SCSI
which one? 2
Specify controller address on the Multibus (in hex): ee40
Device address: EE40

Which unit? 0
Specify drive:
        0 - Fujitsu-M2312K
        1 - Fujitsu-M2384/M2322
        2 - Fujitsu-M2351 Eagle
        3 - Fujitsu-M2294
        4 - Other
which one? 2
ncyl 836 acyl 6 nhead 20 nsect 46 interleave 1
diag>
```

4.2. Set-up the Label Partitions

Use *diag*'s *partition* command to specify these label partitions. *Diag* prompts for a starting cylinder number and size in sectors for each possible label partition. Carefully enter the new dkinfo numbers as calculated above. Label partitions that are undefined, partitions **e** and **f** in the example above, use starting cylinder = 0 and size = 0.

```
diag> partition
Do you wish to modify this table? y
Partition a: starting cyl=0, # of blocks 20240
Change this partition? y
starting cylinder? 0
# of blocks? 23920
Partition b: starting cyl=22, # of blocks 32200
Change this partition? y
starting cylinder? 26
# of blocks? 23920
Partition c: starting cyl=0, # of blocks 769120
Change this partition? n
Partition d: starting cyl=57, # of blocks 359720
Change this partition? y
starting cylinder? 52
# of blocks? 359720
Partition e: starting cyl=0, # of blocks 0
Change this partition? n
Partition f: starting cyl=0, # of blocks 0
Change this partition? n
Partition g: starting cyl=448, # of blocks 216200
Change this partition? y
starting cylinder? 443
# of blocks? 216200
Partition h: starting cyl=683, # of blocks 140760
Change this partition? y
starting cylinder? 678
# of blocks? 145360
...
diag>
```

You now have a completed partition map ready for storage on disk.

4.3. Label the Disk

Write out this partition map to disk by using *diag*'s *label* command. When this completes we are done with *diag*. Run *diag*'s *quit* command to return control back to the PROM monitor.

```
diag> label
diag> quit
>
```

5. Boot Mini-UNIX and Restore the Root and Pub File Systems

5.1. Boot Mini-UNIX

Since both root and pub have been modified, we cannot boot UNIX off the disk. Insert your Sun UNIX Release boot tape and follow the procedure in the "*System*Manager's for installing UNIX, skipping the step for booting *diag* to format and label the disk. The following outlines these steps:

- Load the bootstrap program;
- Skip loading *diag* to format and label the disk;
- Load the standalone copy program;
- Copy the mini-UNIX file system;
- Boot mini-UNIX.

5.2. Prepare and Restore the Root and Pub File Systems

Use mini-UNIX to load the root and pub file systems. Use */dev/MAKEDEV*(8) to create the necessary special files. In this example we need special files for *ndl* units *ndl0* through *ndl3*, the *xy0* device, and a tape device, either *mt0*, *st0*, or *ar0*, depending on what tape device you have.

```
# cd /dev
# MAKEDEV xy0 ndl0 ndl1 ndl2 ndl3 {mt0, st0, or ar0}
#
```

Load the root file system by:

- Running */etc/newfs*(8) on /dev/rxy0a;
- Running */etc/restore*(8) using *r* to restore the root file system from the dump tape;
- Running */etc/fsck*(8) on the restored file system.

This process appears as follows:

```
# /etc/newfs /dev/rxy0a
# fsck /dev/rxy0a
# /etc/mount /dev/xy0a /a
# cd /a
# /etc/restore r
# cd /
# /etc/umount /dev/xy0a
# sync
# /etc/fsck /dev/rxy0a
...
#
```

Be sure to have the dump tape mounted and on line before starting the *restore*.

These examples assume a ½" tape. If you have a ¼" tape drive (**ar** or **st**) the *restore* command line looks a little different, e.g., with SCSI tape, the command is:

```
/etc/restore rbf 126 /dev/rst0
```

Reload the pub file system by:

- Running */etc/mkfs*(8) on /dev/rxy0g using the size (164680) found in */etc/nd.local* above; • Running */etc/fsck*(8) on the restored file system;
- Running */etc/restore*(8) to restore the pub file system (/dev/rxy0g) from the dump tape.

These steps appear as:

```
# /etc/mkfs /dev/rxy0g 164680 46 20 4096 1024
# /etc/fsck /dev/rxy0g
# /etc/mount /dev/xy0g /a
# cd /a
# /etc/restore r
# cd /
# /etc/umount /dev/xy0g
# sync
# /etc/fsck /dev/rxy0g
...
#
```

Note that *newfs* **MUST NOT** be used on the *pub* file system because it does not occupy the entire hard partition */dev/rxy0g*.

6. Boot UNIX and Restore All Affected Nd File Systems

6.1. Boot UNIX from Disk

Bring up the server by aborting mini-UNIX and then booting UNIX.

```
# sync
# [escape back to the monitor by typing an 'L1-A' or the appropriate
  abort sequence from your keyboard]
Abort at some address
> b
Boot: xy(0,0,0)vmunix
Load: xy(0,0,0)boot
Boot: xy(0,0,0)vmunix
...
sunburst login:
```

A side effect of rebooting is the initialization and starting of the network disk service as indicated in the */etc/nd.local* file. For this reason, be sure that all clients are halted or, better yet, powered off before booting full UNIX. The clients cannot be booted until their file systems are restored.

6.2. Prepare All Affected Nd File Systems for Restoring

Run */etc/mkfs*(8) for each file system that has moved and/or changed size. First login as root. Then run */etc/mkfs*(8) for each nd partition using the size found in */etc/nd.local*:

```
sunburst login: root
Password: [enter the root password]
...
sunburst# /etc/newfs /dev/rxy0d
sunburst# /etc/mkfs /dev/rndl0 164680 46 20 4096 1024
sunburst# /etc/mkfs /dev/rndl1 43240 46 20 4096 1024
sunburst# /etc/mkfs /dev/rndl2 26680 46 20 4096 1024
sunburst# /etc/mkfs /dev/rndl3 21200 46 20 4096 1024
sunburst#
```

Notice that it was possible to use *newfs* to construct the new file system on */dev/rxy0d*. This is only possible when a file system fills an entire label partition. Then *newfs* can get the size of the

file system from the disk label. If this condition is not met, the longer *mkfs* command must be used. In this case, the *mkfs* would be:

```
/etc/mkfs /dev/rxy0d 359720 46 20 4096 1024
```

6.3. Restore All Affected Nd File Systems

Do full restores of those file systems on which */etc/mkfs*(8) or */etc/newfs*(8) has just been run.. Also run */etc/fsck*(8) on these restored file systems.

```
sunburst# /etc/mount /dev/xy0d /mnt
sunburst# cd /mnt
sunburst# /etc/restore r
sunburst# cd /
sunburst# /etc/umount /dev/xy0d
sunburst# sync
sunburst# /etc/fsck /dev/rxy0d
...
sunburst# /etc/mount /dev/ndl0 /mnt
sunburst# cd /mnt
sunburst# /etc/restore r
sunburst# cd /
sunburst# /etc/umount /dev/ndl0
sunburst# sync
sunburst# /etc/fsck /dev/rndl0
...
sunburst# /etc/mount /dev/ndl1 /mnt
...
```

6.4. Install the Client Bootstrap Programs

Run */usr/mdec/installboot* to install the clients' bootstrap program.

```
sunburst# cd /usr/mdec
sunburst# installboot bootnd /dev/xy0g
sunburst#
```

The server is now fully operational and ready to handle clients.