sun microsystems

# Software Technical Bulletin
*January 1988*

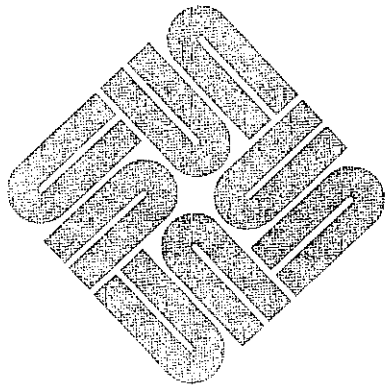
*Software Information Services*

Part Number 812-8801-01
Issue 1988 – 01
January 1988

sun microsystems

# Software Technical Bulletin
*January 1988*

*Software Information Services*



Part Number 812-8801-01
Issue 1988 – 01
January 1988

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 2-312, Mountain View, CA 94043 or by electronic mail to *sun!stb-editor*. U.S customers who have technical questions about topics in the Bulletin should call the Sun Customer Software Services AnswerLine at **800 USA-4-SUN**. Other customers should call the numbers listed in *World Hotlines* appearing in Section 1.

UNIX, UNIX/32V, UNIX System III, and UNIX System V are trademarks of AT&T Bell Laboratories.
DEC, DNA, VAX, VMS, VT100, WPS-PLUS, and Ultrix are registered trademarks of Digital Equipment Corporation.
Courier 2400 is a trademark of U.S. Robotics, Inc.
Hayes is a trademark of Hayes Microcomputer Products, Inc.
Multibus is a trademark of Intel Corporation.
PostScript and TranScript are trademarks of Adobe Systems, Inc.
Ven-Tel is a trademark of Ven-Tel, Inc.
Sun-2, Sun-2/xxx, Sun-3, Deskside, SunStation, Sun Workstation, SunCore, DVMA, SunWindows, NeWS, NFS, SunUNIFY™, SunView™, SunGKS, SunCGI, SunGuide, SunSimplify, SunLink, Sun Microsystems, and the Sun logo are trademarks of Sun Microsystems, Inc.
UNIFY™ is a trademark of Unify Corporation.
ENTER, PAINT, ACCELL, and RPT are trademarks of Unify Corporation.
SQL™ is a trademark of International Business Machines Corporation.
Applix® is a registered trademark of Applix, Inc.
SunAlis™ is a trademark of Sun Microsystems, Inc. and is derived from Alis, a product marketed by Applix, Inc.
SunINGRES™ is a trademark of Sun Microsystems, Inc. and is derived from INGRES, a product marketed by Relational Technology, Inc.

Copyright © 1988 by Sun Microsystems.

This publication is protected by Federal Copyright Law, with all rights reserved. No part of this publication may be reproduced, stored in a retrieval system, translated, transcribed, or transmitted, in any form, or by any means manual, electric, electronic, electro-magnetic, mechanical, chemical, optical, or otherwise, without prior explicit written permission from Sun Microsystems.

# Contents

# 1

# NOTES & COMMENTS

# 1

# NOTES & COMMENTS

## Editor's Notes

### Editor's Notes

The January 1988 Software Technical Bulletin (STB) editor's notes include an introduction to the 'Best of 1987' features, a set of expanded Sun software product and release level tables, notes on world hotlines, and this month's The **Hackers' Corner** in which additions to your `.cshrc` file provide greater user convenience.

### The 'Best of 1987' Features

This month's STB features those short subjects, articles, and in-depth discussions that were most requested, and answered many customer service questions received during 1987.

These articles fall into the general categories described below.

- *Preparing for Your USAC Calls*

  You will find the items in Section 1, Notes and Comments, helpful in having needed information ready when you call **800-USA-4-SUN**. These items include STB duplication permission; a description of the USAC organization; using USAC, *sun!hotline*, and **800-USA-4-SUN**; how to report bugs; and how to determine the SunOS release level you are running.

- *Other Available Information*

  These items include how to use the information in the *Read This First* shipped with each new product or release level, a description of SunOS 'Dot Dot' releases, and information available from the Sun Education email bulletin board.

- *Articles*

  The articles in this issue include helpful information on making changes to your system clock when changing from standard- to daylight-time

according to applicable local law; using the yellow pages and `screendump`; expanding your SunIPC logical hard disk; writing and calling about device drivers; information on disk controllers and disks; and subnetting details.

- *Short Subjects*

  Short items on various topics of interest appear, including information on network packet overload and the limitations of Bridge Box units.

- *In Depth Features*

  Emphasis on networking topics continues with detailed discussions on internet protocols, network transfers, and a description of sockets. Color maps are also discussed in this section.

- *Q&A, and Tip of the Month*

  This month's article repeats the customized `.cshrc` file that adds convenience to your workstation working environment.

**Expanded Current Sun Software Products and Release Level Tables**

This January 1988 Software Technical Bulletin (STB) introduces a set of expanded tables listing available Sun software products and the current release level for each, as of the date shown at the beginning of the tables.

Five tables are now included each month for the categories listed below.

- *Operating Systems*
- *Communications Products*
- *Unbundled Languages*
- *Unbundled Graphics*
- *Unbundled Applications*

Use these tables along with STB articles that appear for a particular product. You can then better determine what your software needs are, what functions are available in a new release, and whether the release you are using is down-level from the most current product release.

**World Hotlines**

For Sun customers served by your local service groups, use the customer service telephone numbers listed in this monthly item. Also, look to this section during the upcoming year for details on your local support call policies and procedures.

**The Hackers' Corner**

This month's **Hackers' Corner** includes code that allows system administrators to conveniently determine the devices attached to their systems.

Again, please note that such applications, scripts, or code are not offered as released Sun products, but as items of interest to enthusiasts wanting to try out something for themselves. They may not not work in all cases, and may not be compatible with future SunOS releases. Please consult your local shell script or programming expert regarding any application, script, or code problems.

Thanks.

The STB Editor

## Software Release Levels

**As of December 18, 1987**

### Operating Systems

| Product Name | Current Release |
|---|---|
| SunOS (Sun-2 and Sun-3 Operating System) | 3.4 |
| Sys4 (Sun-4 Operating System) | 3.2 |

### Communications Products

| Product Name | Current Release |
|---|---|
| SunLink BSC3270 | 3.0 |
| SunLink BSCRJE | 5.0 |
| SunLink Local 3270 | 5.0 |
| SunLink SNA3270 | 5.0 |
| SunLink Peer-to-Peer | 5.0 |
| SunLink IR | 5.0 |
| SunLink DDN | 5.0 |
| SunLink DNI | 5.0 |
| SunLink OSI | 5.0 |
| SunLink MCP | 5.0 |
| SunLink TE100 | 5.0 |
| SunLink X.25 | 5.0 |

## Unbundled Languages

| Product Name | Current Release |
|---|---|
| Sun FORTRAN* (for Sun-2 and Sun-3 systems) | 1.0 |
| Sun FORTRAN* (for Sun-4 systems) | 1.05 |
| SunPro | 2.0 |
| NeWS | 1.0 |
| Sun Common Lisp-D | 2.1 |
| Sun Common Lisp-E | 1.1 |
| Modula-2 | 1.0 |
| Cross Compilers | 2.0 |

* Sun FORTRAN Note

The `f77` compiler is automatically included with SunOS release 3, which includes SunOS releases 3.2, 3.4, and 3.5. Sun FORTRAN release 1.0 (for Sun-2 and Sun-3 systems) and Sun FORTRAN Release 1.05 (for Sun-4 systems) are value-added products that support VMS extensions to the `f77` compiler, and must be purchased separately from the operating system.

## Unbundled Graphics

| Product Name | Current Release |
|---|---|
| SunGKS | 2.1 |

## Unbundled Applications

| Product Name | Current Release |
|---|---|
| SunAlis | 2.1 |
| SunINGRES | 5.0 |
| SunSimplify | 1.0 |
| SunUNIFY | 2.0 |
| Transcript | 2.0 |
| SunIPC | 1.1 |
| PC-NFS | 2.0 |
| SunTrac (for Sun-2 and Sun-3 systems) | 1.0 |
| SunTrac (for Sun-4 systems) | 1.0/3.2 |

## Current Sun Software Products and Release Levels

The preceding tables contain lists of current Sun software products and their respective current release levels.

You will note that the Software Technical Bulletin (STB) contains articles from time to time that detail technical changes in a given software product's next available release.

Please contact your sales representative if you decide that you would like to update the release level of a Sun software product you already use, or wish to purchase another product. Use the tables to determine whether your release is the current release level.

These tables appear monthly in the STB for your convenience.

## World Hotlines

### World Hotlines

Sun Customers throughout the world have service hotlines available for both software and hardware support questions. The service hotlines are shown below. If your country is not shown in the table, please phone your local Sun sales office.

| | | |
|---|---|---|
| Australia | Sun Australia<br>Lionel Singer Group | (011-61-2) 957-2522<br>(011-61-2) 957-2655 |
| Canada | Montreal Branch<br>Ottawa<br>Vancouver Branch<br>Western Branch | (514) 879-1914<br>(613) 748-9617<br>(604) 641-1296<br>(403) 295-0150 |
| France | Paris<br>Sun Microsystems France SA | (33) 1 4630 2324 |
| Germany | Munich<br>Sun Microsystems GmbH | (49) 89/95094-321 |
| Japan | C. Itoh Data Systems<br>Nihon Sun | (011-81-3) 497-4676<br>(011-81-3) 221-7021 |
| The Netherlands | Soest<br>Sun Microsystems Nederland BV | (31) 2155 24888 |
| Switzerland | Zurich<br>Sun Microsystems Schweiz AG | (41) 1 828 9555 |
| United Kingdom | Camberley<br>Sun Microsystems UK Ltd | (44) 276 62111 |
| United States | All,<br>including Puerto Rico | 1-800-USA-4-SUN |
| **Intercon** | All countries outside the<br>USA, Europe, and northern Africa | (415) 691-6775 |

## STB Duplication

### 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. So long as you duplicate copies and route them only internally, 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.

### Direct STB Purchase

The STB is sent to the primary contact person named in all software support contracts. Sun is looking into methods by which customers holding these contracts may purchase extra copies directly.

Look to this column for an announcement regarding the purchase of extra STB copies.

### Further Questions

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

## USAC Organization

**Phone Support Organization**

The Customer Software Services organization is organized to better serve our support contract customer base.

**Product Groupings**

This structure features support groups that are organized around product categories. These are:

- UNIX
- Graphics
- Languages
- Data Communications
- Applications (e.g. Sun Ingres, Alis)
- Personal Answer Line

**U.S. Answer Center**

Collectively, these groups report to the position of U.S. Answer Center manager. The USAC manager reports directly to the Director of Customer Software Services.

**Why the Change**

This organization allows more technical depth in specific areas rather than a generalist approach. It also allows much room for expansion, flexibility and growth as customer requirements change.

This organization fits in with our already announced services (AnswerLine and Personal AnswerLine).

Customer Software Services expects this organization to improve our productivity and, we hope, improve customer satisfaction. Please feel free to send us your comments, and remember that we are convinced this change will bring about a permanent improvement.

## Using USAC

### U.S. Answer Center: Feedback Wanted

Here in the U.S. Answer Center (USAC), we strive to maintain the highest level of commitment to our customers, listening and responding to their needs. In order to help us improve our service, we encourage your feedback. Please send all letters to the address shown below.

```
Sun Microsystems, Inc.
2550 Garcia Ave.
Mt. View, CA  94043
Attn: Marion Brown 2-30
```

## Using *sun!hotline*

### Using your *sun!hotline*

The email address *sun!hotline* is for use by customers holding software support contracts. When sending email to *sun!hotline*, you should always include the information listed below. If any of the information is missing, it may take longer to get the needed support.

- workstation model and serial number
- name
- phone number and area code
- electronic mail address (`sendmail` does not always show the correct address.)
- company or organization name and address
- SunOS release number (See the note *SunOS Release Level* in this issue to find how to determine your SunOS release level.)
- any information which may help diagnose the problem

Information that helps diagnose the problem includes what was running when the problem occurred, a description of symptoms including exact text of any error messages, a small test program that exhibits the problem, your hardware configuration, or any other information that seems appropriate to the problem at hand.

Please avoid sending large files by email.

A service call is logged and the call is assigned to an engineer. After the call is logged, you will receive email giving the service order (SO) number and an engineer's phone number to call. This usually is done the same day the email is received.

Use the phone number when you have not received a response from an engineer within 24 hours. Please note that a response is not necessarily an answer. The engineer may simply send email just to say 'I have it and am working on it.'

Finally, please note that *sun!hotline* is for software-related questions only. For customers holding support contracts, hardware troubleshooting questions may be called in to **800 USA-4-SUN.**

## Reporting Bugs

### 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 note *SunOS Release Level* in this issue!)
- A description of the problem that you are experiencing

## Using USA-4-SUN

### Using 800 USA-4-SUN

All Sun customers may call the **800 USA-4-SUN** phone line for assistance in the use of Sun software, hardware, and network products. This article explains what information you will need when you call, and how your call is routed to the service engineer who helps you. Your call will be routed to different support locations, depending on whether you have a support contract and on what type of product you are using that requires customer support.

When calling the **800 USA-4-SUN** number, you should always have the information listed below ready. If any of the information is not readily available, it may take longer to route your call properly.

- workstation model and serial number
- purchase order (PO) number (for those customers not holding support contracts)
- name
- company or organization name and address
- SunOS release number (See the note *SunOS Release Level* in this issue to find how to determine your SunOS release level.)
- problem description

### Routing Your Calls

Many customers call after talking to their sales representatives. Others call 'cold'. In either case, you are prompted by a prerecorded message. It asks those not holding support contracts to have their PO number handy. The recording then asks you to dial a number, depending on the type of support needed. The current options are listed below.

**Dial 1** for software support

**Dial 2** for hardware support, including returning or exchanging parts

**Dial 3** to schedule the installation of a new system

**Dial 6** for telemarketing, to purchase customer service products or service contracts

After you select a number, a service dispatcher will ask you for the information listed above and for a brief description of your problem. The dispatcher then uses your problem description to route your call to a support engineer who specializes in the product that is the subject of your phone call.

The dispatcher logs your service call and will give you a service (SO) number that you may use as a reference to your call in future calls, mail, or email. Your call is now routed to specialists who answer calls for their particular subject matter area.

You can now expect an engineer to return your call that same day, or during the next normal working day.

## 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.

# The *Read This First*

## Using the *Read This First* (RTF) Document

This article contains a discussion of the purpose and use of the *Read This First* (RTF) document provided with all Sun Microsystems software.

The primary purpose of the RTF is to provide the user with current, pertinent information about the corresponding software product. This includes installation considerations of importance to system administrators when installing a new product or upgrading an existing product. Additionally, details are provided of new or changed features of importance to product users.

Read the RTF thoroughly before beginning the installation or upgrade since much of this information should be kept in mind at that time.

## The RTF Format

The format of the RTF is designed to include the items listed below.

- Software compatibility with Sun system hardware and operating system release levels
- Environmental requirements, such as physical space and minimum swap space needed for proper operation
- Product or release anomalies or both
- How to get help

The RTF is the designated document to include information describing installation and usage problems encountered during the final testing of the product. These descriptions usually include workaround methods. The RTF also describes any errors in the product documentation, as well as the revised form, reflecting the current state of the product.

## Copying and Distributing the RTF to Other Users

In some locations, the individual responsible for installing product software does not actually use the product. In these situations, the person installing the software will want to have copies of the RTF made for internal distribution to the Sun Workstation end-users. This ensures that information affecting product use is provided to developers and users of the corresponding product and dependent applications.

The primary contact person or department may duplicate the 'master' RTF copy for all Sun Workstation end-users, as well as to those who need the information. So long as the copies are duplicated and routed internally to employees working for a company having the product license, there are no copyright infringement problems.

This limited permission is for the convenience of Sun customers only. It does not include any other Sun documentation, nor does it permit any duplication for resale or distribution outside your company.

## 'Dot Dot' Releases

### SunOS 'Dot Dot' Releases

Sun Microsystems is now releasing tapes containing bundled patches every two or three months, between other SunOS releases. These new releases are called 'Dot Dot' releases. Look to this article and future articles in the STB 'Notes and Comments' section that contain announcements of 'Dot Dot' releases, lists of specific fixes, fix reference numbers, and a synopsis of each corrected problem.

### SunOS 'Dot Dot' Release Availability

SunOS 'Dot Dot' releases are available at no charge to Sun customers holding software support contracts, and to all Sun customers under warranty. Other Sun customers wishing to purchase a particular release may do so for $200 USD.

To request or order a release, please call **1-800-USA-4-SUN** and request the release by its 'Dot Dot' number, or by the Order Management and Retrieval (OMAR) number appearing in the next paragraph and in the Customer Support price list. For Sun Europe customers, please call your local support group or sales representative.

The first two releases available at this time are SunOS releases 3.4.1 and 3.4.2. Please note that these two dot dot releases can be installed separately or together on systems currently running SunOS release 3.4.

### 'Dot Dot' Ordering Information

Use the information below to order SunOS releases 3.4.1 or 3.4.2 or both.

A list of release contents appears at the end of this article. Use these two lists to determine whether you need either release.

SunOS Release 3.4.1

| Description | CPU-type | Media Size | OMAR # | Unit Price |
|---|---|---|---|---|
| Docs, & Tape | 68010 | 1/4" | DOT2-01-3.4.1 | $200 |
| Docs, & Tape | 68010 | 1/2" | DOT2-02-3.4.1 | $200 |
| Docs, & Tape | 68020 | 1/4" | DOT3-01-3.4.1 | $200 |
| Docs, & Tape | 68020 | 1/2" | DOT3-02-3.4.1 | $200 |

SunOS Release 3.4.2

| Description | CPU-type | Media Size | OMAR # | Unit Price |
|---|---|---|---|---|
| Docs, & Tape | 68010 | 1/4" | DOT2-01-3.4.2 | $200 |
| Docs, & Tape | 68010 | 1/2" | DOT2-02-3.4.2 | $200 |
| Docs, & Tape | 68020 | 1/4" | DOT3-01-3.4.2 | $200 |
| Docs, & Tape | 68020 | 1/2" | DOT3-02-3.4.2 | $200 |

## SunOS Release 3.4.1

A list of SunOS 3.4.1 fixes, fix reference numbers, and a synopsis for each solved problem appears below.

- `blank`, Reference Number: 1004642

  Synopsis: `screenblank` allows the `-k` and `-m` options while in `suntools`.

- `cgi`, Reference Number: 1003572

  Synopsis: Bad `inquire_cell_array` and `inquire_pixel_array` `name` argument.

- `cgi`, Reference Number: 1003687

  Synopsis: The CGI Mouse cursor is always visible.

- `cgi`, Reference Number: 1004825

  Synopsis: `-lcgi` requires `-lsuntool` to compile a `cgi` program.

- `cgi`, Reference Number: 1005251

  Synopsis: `close_cgi_pw()` fails if no viewsurface is active.

- `cursor`, Reference Number: 1003864

  Synopsis: The crosshair cursor does not work when `CANVAS_FAST_MONO` is used.

- `fpa`, Reference Number: 1004500

  Synopsis: A program compiled using `-ffpa` causes an `FPA KERNEL BUS ERROR` to occur.

- `fsck`, Reference Number: 1003023

  Synopsis: The `fsck: HOLD BAD BLOCK` message is undocumented.

- `gp1`, Reference Number: 1004863

  Synopsis: This is a `GP1_PR_PGON_TEX` problem.

- `gp1`, Reference Number: 1004984

  Synopsis: `GP1_PR_ROP_TEX` semantics are wrong for a 1-bit deep `src`.

- `loopback`, Reference Number: 1005131

  Synopsis: The resolver has the wrong `loopback` address.

- `make`, Reference Number: 1003151

  Synopsis: `make` does not always build the objects that it should.

- `ping`, Reference Number: 1004791

  Synopsis: `ping` says machines are up even when they are not.

- `printer`, Reference Number: 1004074

  Synopsis: `lprm` causes line printer daemon to disappear.

- `rexd`, Reference Number: 1005140

  Synopsis: A `rexd` race condition occurs when mounting in `/tmp`.

- `scsi2`, Reference Number: 1004639

  Synopsis: This is a bug in the Sun-2 SCSI driver.

- `sendmail`, Reference Number: 1005042

  Synopsis: Yellow Page alias must use primary host names.

- `socket`, Reference Number: 1003135

  Synopsis: `panic: mfree` occurs with `AF_UNIX SOCK_STREAM` out-of-band (OOB) data.

- `suncore`, Reference Number: 1000895

  Synopsis: Transformation of text that does not clip.

- `sunpro`, Reference Number: 1004898

  Synopsis: The `install_sunpro` script fails for all configurations.

- `termcap`, Reference Number: 1004731

  Synopsis: `termcap` entry for `TERM=wy` breaks `initscr()`.

SunOS Release 3.4.2

A list of SunOS 3.4.2 fixes, fix reference numbers, and a synopsis for each solved problem appears below.

- `dbx`, Reference Number: 1003647

  Synopsis: Lexically recursive `#includes` confuse `dbx`

- `dbx`, Reference Number: 1004996

  Synopsis: `dbx` shows segmentation violation while `stepiing`

- `diag`, Reference Number: 1005466

  Synopsis: `sysdiag`'s `sptest` fails with `/dev/tty[a,b]`; does not respond

- `disk`, Reference Number: 1005360

  Synopsis: SCSI disk driver hangs when ACB4000 reports write fault

- `disk`, Reference Number: 1005363

  Synopsis: Some SCSI MD21 (141 MB) errors cause system hang

- `ether`, Reference Number: 1006127

  Synopsis: Ethernet problems induced by bad ICMP address mask reply.

- `io`, Reference Number: 1005930

  Synopsis: `physio` bug causes `writev(2V)` failure

- `io`, Reference Number: 1001069

  Synopsis: bug in `physio` breaks `readv`

- `kernel`, Reference Number: 1006165

  Synopsis: `sysdiag`'s `softfp` and mc68881 core dump (illegal instruction)

- `line`, Reference Number: 1004863

  Synopsis: This is a `GP1_PR_PGON_TEX` problem.

- `line`, Reference Number: 1004984

  Synopsis: `GP1_PR_ROP_TEX` semantics are wrong for a 1-bit deep `src`.

- `line`, Reference Number: 1005359

  Synopsis: Problem using `pw_line` and `pw_polyline`

- `lockf`, Reference Number: 1004336

  Synopsis: `lockf()` very slow

- `look`, Reference Number: 1003885

  Synopsis: `look` may dump core on long lines

- `net`, Reference Number: 1004765

  Synopsis: subnet broadcast address computed incorrectly

- `nfs`, Reference Number: 1005489

  Synopsis: NFS attribute cache functions incorrectly

- `rpc`, Reference Number: 1004739

  Synopsis: `rpc.lockd` fails to free, thus using excess memory

- `sccs`, Reference Number: 1003207

  Synopsis: SCCS uses delta times for `diffs`

- `sccs`, Reference Number: 1005438

  Synopsis: SCCS `deledit` duplicates random lines in a file

- `scsi3`, Reference Number: 1005366

  Synopsis: System panics when using `ttya` with SCSI3

- `serial`, Reference Number: 1006154

  Synopsis: system is flooded with `zs` interrupts on `synca/b` transitions

- `sunpro`, Reference Number: 1004598

  Synopsis: `make` does not handle square bracket characters in target filenames

- `sunpro`, seven unnumbered fixes

  Descriptions:

  1) No longer dumps core if the source needed to build a library member does not exist; instead reports "Don't know how to build x".

  2) Fixed the `-k` option so that it works for lists of targets given on the make command line.

  3) Remove the `.make.state` lock file if make is interrupted.

  4) Use the `varargs` mechanism for the error routines.

  5) Fixed bug that caused very long command lines to be read incorrectly.

  6) Fixed bug that caused `$${X}.il` to be read incorrectly when used as a dependency.

  7) Made it possible to undefine default suffix rules from the user's makefile.

- `tape`, Reference Number: 1004559

  Synopsis: UNIX hangs while booting if `xt` controller has on-line drive

- `transfer`, Reference Number: 1006132

  Synopsis: TCP/IP file transfer using `ftp` hangs/stops when using 3.4

# Sun Education

## Sun Educational Services email Bulletin Board

Sun Educational Services, located in Milpitas, California, has set up an 'email Bulletin Board' facility for Sun Microsystems customers.

This facility provides a means for interested customers to learn about new courses and related developments happening within Educational Services. This facility is also a good means for customers to direct questions and general requests for information to Educational Services. These questions and requests might be regarding course outlines, catalogs, how to arrange for a dedicated class at the customer's site, and so on.

## How to Be Added to the email Bulletin Board

Customers who want to be added to the Education Services email Bulletin Board should send their Usenet or DARPA addresses to:

```
customer-training@sun.com
suncustomer-training
```

To ask questions, simply send the question to one of the above email addresses.

## The New Educational Services Course Brochure

Sun Microsystems customers should soon be receiving the new Sun Educational Services course brochure. The brochure contains complete course descriptions as well as the course offerings scheduled from January, 1988 through June, 1988.

If you do not receive this brochure by November 10,1987, please contact your local sales office or Sun Educational Services directly at either of the toll-free numbers listed below.

| | |
|---|---|
| In California: | 800-423-8020 |
| Elsewhere in the continental U.S.: | 800-422-8020 |

Sun customers in the United Kingdom and in Europe should contact their local service center or sales representative.

# 2

# ARTICLES

# 2

# ARTICLES

## Time Changes

### UNIX Kernels and Greenwich Mean Time (GMT)

All UNIX kernels use GMT. However, some systems allow users to set environmental time variables as they like. This allows users to dial in from any time zone and have the date command reflect the correct local user time.

The time zone, TZ, variable is supported by any program built in the System V, release 3.2 environment. This includes all commands in `/usr/5bin`, which contains less than 1% of the commands. Sun supplies S5 versions of commands where there is a significant incompatibility between release 4.2 and S5 versions. Thus, 99% of the commands are located in `/bin`, `/usr/bin`, `/usr/ucb`, and the like. These commands are built with the release 4.2 libraries and do not understand the TZ environment variable.

### Time Zone (TZ) Problems

There are two problems with the TZ environment variable. First, it is difficult to get TZ into every process' environment. Vanilla S5 systems attempt a solution by setting TZ in `/etc/rc` for programs and their children that run from there. TZ is also set in `/etc/profile` for login Bourne shells. This, however, omits any user with non-Bourne login shells. You can fix this for the C-shell by putting TZ in your `.login`, or by making a system-wide `.login` file (`/etc/csh.login`, for example) and placing TZ there.

However, this fix does not solve the TZ problem for some specialized applications where the login shells are not shells in the standard UNIX sense. You can fix TZ here by modifying the S5 login to preserve the TZ value in its environment and having something set TZ before running `getty`. Note that `init` does not run `getty` directly in S5. `init` has a table telling it what programs to run and when. Each line in the table contains a UNIX command to be run. For example, you could run the `env` command with arguments telling it to set TZ and then to run `getty`.

The standard S5 as distributed by AT&T uses Eastern Standard Time (EST) if TZ is not set. Vendors may change the time to their local time zone. Xenix continues to use the V7 `ftime` call, which gets time zone information from the kernel. You are not forced to set TZ in the environment to get a local time zone.

Second, some programs do not use the user time zone but need to know the time zone in which the the computer is located. An example is `uucico`, which makes long-distance calls to other machines after 2300 in the computer's time zone, not that of the user. A user can force the computer to make a long-distance call at any local time by setting TZ as required and then running `uucico`.

**A History of Time**

Prior to the V7 `ftime` call, V6 had neither an environment nor an `ftime` call. It supported the time zone complied into the C library only. If you were not in the same time zone as Dennis Richie's computer, you had to recompile `ctime` and rebuild every command that used the date.

V7, from AT&T and not Berkeley, fixed this by adding an `ftime` call that provided the current clock value, finer time resolution to include milliseconds, the local time zone offset from GMT, and a flag indicating whether Daylight Savings Time (DST) was to be used. You then merely had to reconfigure the kernel. Release 4.1BSD, based on UNIX/32V which is a VAX variation of V7, used `ftime` as well.

Release 4.2BSD improved `ftime` by replacing it with `gettimeofday`. This new call provided the time as `timeval`. It provided higher resolution in a standard, system-wide format, the GMT offset, and the DST flag. The flag now included what type DST was to be used. This accounted for the fact that not all countries go on and off DST at the same time as the United States. Further, the dates for the beginning and end of DST differ as well. Release 4.2BSD provided tables for some locations outside the USA, but those tables were not always correct. Some fixes were made in release 4.3BSD. Sun OS release 3.2 provides additional fixes correct at the time of release.

System III did not include `ftime`. It sets the current time zone from the TZ environment variable. This arrangement has the same problems detailed above and it still does not cover time zones outside the USA. Problems arise when the USA, Australia, and Europe did not go to DST at the same times.

Release 4.2BSD is somewhat more accurate, although it does not understand Australian and European rules completely. Canada is treated as being under the same rules as the USA and the USA DST changes of 1987 are not incorporated. Release 4.3BSD has corrected some of the European and Australian rules, as well as Canadian rules that did not change in the mid 1974-5.

**Time and Time Again: Sun OS Release 3.2**

Sun OS Release 3.2 has rules that should be correct for Europe. The Australian rules have two variants. The first starts on the last Sunday in October and ends on the first Sunday in March. The second does the same thing until 1985 when it ends on the last Sunday before March 21 and in 1986 when it starts on the last Sunday in October and ends on the last Sunday before March 21.

According to the Australian consulate, DST starts on the next-to-the-last Sunday in October and ends on the last Sunday before March 21. Finally, Sun OS Release 3.2 includes USA changes made in 1987.

Sun OS Release 3.2 should also choose the proper time to start and end DST. It starts at 0200 standard time and ends at 0200 DST in the USA and Canada. It starts at 0200 standard time and ends at 0300 DST in Australia. It starts at 0100 standard time and ends at 0200 DST in Great Britain, Eire, and the Western European time zone. It starts at 0200 standard time and ends at 0300 DST in the Central European time zone.

Note that these latest tables are compiled into the `ctime` code. Only applications built with the Release 3.2 library will use them. Applications built with earlier library versions will give the same results under Release 3.2 that they gave under earlier releases.

There is no facility in Release 4.2 to do the same time calculations. However, Arthur Olson at the US National Institutes of Health (NIH), implemented a new version of the time zone code that permits four things:

1. Permits you to set TZ to alter the time zone information you see.
2. Permits programs like `uucico` to see the correct local time zone, regardless of the TZ value.
3. Permits you to set up any time zone conversions in accordance with the wisdom of your local politicians.
4. Permits you to change time zone conversions as political wisdom changes, as it is want to do from time to time.

### The TZ Environment Variable

The TZ environment variable is treated like a filename. The file contains a DST rules table. Given the filename, a routine reads in the table. If the table has not already been read in, `ctime`, `localtime`, and the S5 routine `tzset` call this routine with the value of the TZ environment variable as the filename. The routine looks for the file named `localtime` if a null pointer is passed. You will then get the proper local time if TZ is not set or if the program explicitly calls this routine with a null pointer like `uucico` does.

This will probably appear in Release 4.4BSD (date unknown) and may appear in a future Sun OS release. This would free developers from having to anticipate future DST rules changes.

## Yellow Pages

### The Purpose of the Yellow Pages (YP) Service

The Yellow Pages (YP) Service provides a set of maps or data files common to one or several systems. These maps contain network configuration information about these systems. Workstations and terminals then use these maps instead of having files of their own.

The yellow pages server contains a set of yellow pages maps. YP clients bind to the server to access these maps. Only the server maps need updating since it sets up a domain to which YP slave servers and clients belong. Multiple YP domains may exist on the same network.

The YP server makes its maps from configuration files used in Ethernet networking with other systems. A YP client checks its local file first. Not finding the information locally, it then consults the YP server maps. The configuration files used to create the maps are shown below.

```
/usr/lib/aliases
/etc/ethers
/etc/hosts
/etc/ethers
/etc/group
/etc/netgroup
/etc/networks
/etc/passwd
/etc/protocols
/etc/services
```

Customers get the best use of the yellow pages service at a site including large numbers of computers which are networked together in a common network. Such sites are constantly dynamic, adding or moving systems, and adding or removing users from the networked systems. A number of Sun workstations may be included which network with each other and other computers through utilities including `rlogin`, `rsh`, `telnet`, `ftp` and `tftp`.

System administrators may have difficulty maintaining the several network configuration files for each system when the environment is constantly changing. The yellow pages service helps by having only one set of master maps that is changed by the master YP server for all Suns workstations on the network.

Other workstations use the master maps by binding to the YP server. This bind process runs continually on the YP client, the workstation being served, and is called `ypbind`. The YP server runs `/etc/ypserv` and `/etc/ypbind` to serve its YP clients.

Once a day or week the system administrator updates the YP master server `/etc/hosts`, `/etc/others`, `/etc/passwd`, and other files. The system administrator then updates the yellow pages by remaking the YP maps. This procedure takes only a few minutes. The YP clients then bind to the YP server and use the updated maps to find host- and user-names only after the client determines that its local files do not have the needed information.

A single YP server may not be sufficient to meet client YP requests in a large network environment. Sun yellow pages have YP slave servers configured in such large networks. These YP slave servers access YP maps and clients bind to these slave servers using `/etc/ypserv` and `/etc/ypbind`. To further aid the customer, YP slave server configuration files are updated automatically, receiving the updated YP maps from the YP master server.

More than a single YP master server is found in very large networking environments. You may find multiple domains on these large networks since each YP master server defines a separate YP domain. There is no conflict since each YP clients knows which domain it is in. The YP client's domain name and YP slave server hosts are determined when the YP master server is configured by `setup`. The `/bin/domainname` utility returns the YP domain name to which a YP client or slave server belongs.

## YP Server Installations

The YP master server, slave server, and client system is installed using the steps listed below.

1. The system administrator initially runs `setup` to install the Sun system. The YP master server is configured as such at this time. The names for all possible YP slave servers are also configured on this same machine. The names are also configured on the YP slave servers at this time.

2. Update the YP master server YP maps daily by changing the YP master server configuration file or files (`/etc/hosts`, `/etc/passwd`, and the like), and by running `/etc/ypmake <mapname>` or `/etc/ypinit` for all maps as needed.

3. The YP master server then sends these updated YP maps out to the YP slave servers. They should be running `/etc/ypserv` to automatically receive the updated maps. The system administrator otherwise has to do a `yppush` to the slave server when it is ready.

4. The YP client then can run `/etc/ypbind` which is broadcast to all YP slave servers. The slave server responding first is the one to which the YP client is bound. This process assumes that the YP slave server responding first is the least busy so that the networking load is most evenly distributed. This process continues and the client stays bound to the same slave server until the YP client reboots.

5. The YP client automatically rebinds to another YP server in the event a YP server goes down. The YP client requests a YP service to its YP server via an `rpc` call. The server may or may not respond. The client drops its `ypbind` with this server and `ypbinds` to another server if a second request results in no response.

6. For example, the YP clients runs a request for telnet host56832. First, the YP client's own `/etc/hosts` file is consulted. If no entry for host56832 is found, the YP client requests a service for a YP map lookup using an `rpc` call to the YP server. This request for the internet host56832 address goes to the client's YP slave server or to the master server if no slave server exists. The YP server then responds with the internet address. The YP client continues and performs a telnet request to host56832.

**YP Utility Summary**

A selected list of YP utilities appears below.

1. `/etc/ypserv`

   The process the YP master or slave server runs to serve YP maps.

2. `/etc/ypbind`

   The process any YP master or slave server or client runs to issue or resolve YP requests via `rpc`.

3. `/etc/ypwhich`

   Returns the YP service domain name.

4. `/etc/ypwhich -m`

   Returns which YP master or slave server owns the YP maps. The one which ran `ypmake` is the one who made the new map. Check this if you suspect that your YP maps are incorrect. It may be the wrong server who ran the `ypmake`.

5. `/etc/yppoll`

   To find out the version of the YP maps you are using. This command may be run on any machine, YP server, or client.

6. `/bin/domainname`

   Returns the YP domain name that the system is in. This is set in the /etc/rc.local file on each system whether a YP server or client. This should be checked if the YP maps appear incorrect.

7. `/etc/ypcat`

   Does a cat of a YP map. For example: "/etc/ypcat hosts" should provide a list of the hosts known to YP.

8. `/etc/ypmatch`

   This utility is new to release 3.0 and provides a grep-like feature. For example, "/etc/ypmatch joe passwd" should show a passwd entry known to YP.

# Using `screendump`

## `screendump`: Saving and Printing Images

Customer Support receives many questions on how to save screen images to files for subsequent display or hardcopy output. This article summarizes usage and possible pitfalls of the related utilities on the Sun workstation.

## Saving Images

Type the following command to save a screen image to a file named `rasterfile`.

```
% screendump rasterfile
```

If you now run *file(1)* on `rasterfile` you will see a response something like the example shown below.

```
% file rasterfile
rasterfile: rasterfile, 1152x900x1 standard format image
```

*file(1)* is reporting the width, height and depth of the raster image in pixels, respectively. If this image had been created on a color screen, the depth would have been listed as '8'. For those interested in the details of rasterfile format, see the *Pixrect Reference Manual*, part number 800-1254, and `/usr/include/rasterfile.h`.[6]

## Printing Saved Images

Once an image has been saved, it is available for redisplay or printing. To display the image on the screen, simply use *screenload(1)* as shown below.[7]

```
% screenload rasterfile
```

To print `rasterfile` on the Sun Laserwriter, a special TRANSCRIPT[8] filter has been created for use by `lpr`. Thus, the command shown below will access this filter and print `rasterfile`.

```
% lpr -v rasterfile
```

Printing can be a problem with color images, however. Since the Laserwriter represents only monochrome images, you must first convert a color rasterfile before sending it to the printer. A handy filter for this purpose is *rasfilter8to1(1)*.

---

[6] In particular, SunView and Pixrect programmers will want to know that `screendump` creates rasterfiles with a colormap size of 256. SunCore programmers should be aware that the `screendump` colormap takes color intensity values from zero to 255 rather than from zero to one as in SunCore. SunCore programs need to convert from one colormap type to the other to interface with `screendump`.

[7] The SunView, Pixrect, and SunCore packages all have internal facilities for loading a rasterfile as well. See *pr_load()* in the Pixrect Reference Manual, and *file_to_raster()* in the SunCore Reference Manual, part number 800-1257, for details.

[8] TRANSCRIPT is a trademark of Adobe Systems, Inc.

Use the command shown below to print an image created on a color monitor.

```
% rasfilter8to1 < rasterfile | lpr -v
```

In addition, `screendump` on color monitors can fail, causing a corrupted image, if anything on the screen is moving during the dump. Examples include moving the mouse cursor, a ticking clock, a perfmeter (performance meter), or even a blinking caret. So be sure to exit or cover any tool which is updating the screen while `screendump` is active.

Another special case is the Sun high-resolution, monochrome monitor. If a screendump made on this monitor is printed, part of the image will be cut off. The solution is to access the TRANSCRIPT filter explicitly, telling it to scale the image so as to fit all on one page. Try the command shown below to scale the image to fit on an eight-inch page.

```
% screendump | pssun -S 8 | lpr
```

Note that since the filter `pssun` is being called explicitly, the `-v` option to `lpr` should be omitted. Calling the filter explicitly provides several additional capabilities in the form of options to `pssun`, including rotation and replication. See the *pssun(1)*[9] manual page for more information.

## Current Frame Buffer Details

Details follow describing how `screendump` knows to dump the current frame buffer. `screendump` has a default notion of the frame buffer, `/dev/fb`. However, there are cases in which one wants to dump images from a frame buffer other than `/dev/fb`, for example, on a machine with two frame buffers. For this purpose, `screendump` has a `-f` option which, in the example shown below, will store the image currently displayed on the color frame buffer, even if that frame buffer is not represented by `/dev/fb`. Note that `screenload` has a parallel `-f` option.

```
% screendump -f /dev/cgtwo0 rasterfile
```

Another reason to specify the frame buffer name is the unique frame buffer configuration of the Sun-3/110. This machine has two frame buffers, one monochrome (`/dev/bwtwo0`) and one color (`/dev/cgfour0`), plus an overlay plane. The *switcher(1)* man page describes how to invoke `suntools` twice, once on each frame buffer. When starting `suntools` in this way, be sure to use the `-f` option of `screenload` or `screendump` to access the desired frame buffer.

Some users like to start `suntools` on the 'merged' frame buffer, that is, combining the color and monochrome frame buffers and using the overlay plane to indicate which frame buffer is being used on a per-pixel basis. Note that starting `suntools` without any options means running in the merged frame

[9] This manual page must be loaded from the TRANSCRIPT installation tape.

buffer. Color windows, e.g. color images from a graphics package, reside in `cgfour0` and monochrome windows, e.g. `shelltools`, `cmdtools`, `textedit` windows, and the like, reside in `bwtwo0`.

As a result, it is not currently possible to screendump the entire screen in this particular case. Running `screendump` on `cgfour0` saves the color images while specifying `bwtwo0` stores the monochrome. In either case, the parts of the screen defined by the other frame buffer are left blank in the resulting rasterfile. If no frame buffer is specified, `/dev/fb` is used and this corresponds to `/dev/cgfour0`.

For Further Information

UNIX manual pages used as references in preparing this article are listed below.

- *screendump(1)*
- *screenload(1)*
- *rasfilter8to1(1)*
- *pssun(1)*
- *rastrepl(1)*
- *file(1)*
- *lpr(1)*
- *switcher(1)*
- *suntools(1)*

Sun manuals used as references are shown below.

- *Pixrect Reference Manual*, part number 800-1254
- *SunView Programmer's Guide*, part number 800-1345
- *SunCore Reference Manual*, part number 800-1257
- *Release 3.2 Manual for the Sun Workstation*, part number 800-1364
- *Release 3.4 Manual for the Sun Workstation*, part number 800-1614

The following `include` file is used for reference.

- `/usr/include/rasterfile.h`

# Expanded SunIPC Disk

## Creating a 30 Mbyte SunIPC Logical Hard Disk

Use the procedures shown in the *Sun IPC™ User's Guide*, part number 814-1002, chapter 4, 'Using Disks' to create logical hard disks up to 20 Mbytes in size.

Use the procedures contained in this article to create a 30 Mbyte SunIPC logical hard disk.

### Background and Requirements

You may wish to increase the size of your SunIPC logical hard disk as your disk needs increase. Initially, your SunIPC logical hard disk occupies about 1 Mbyte of storage space. The name of this file(s) is `/usr/pctool/drive_C.pc0` through `/usr/pctool/drive_C.pc3`, depending on your having up to four SunIPC boards installed in your system. In this article the case of a single SunIPC board and file `/usr/pctool/drive_C.pc0` is considered.

The logical hard disk grows to approximately 10 Mbytes by default as users store more files or PC applications or both. The maximum disk size upper limit may be reset, allowing additional disk storage.

You *must have access* to a SunIPC floppy disk subsystem to change the SunIPC logical hard disk size. You will create a `boot`able floppy before beginning the procedures in this article. This is required since the existing drive C is destroyed when changing the logical hard disk size. Note that you *cannot* backup your logical hard disk to an NFS server since it is *not possible* to boot SunIPC from a network device.

Also note that it is best to change the logical hard disk size when you first receive the SunIPC board. Backup time at a later date may be greatly increased by your having many PC application programs stored on the logical disk.

## Two Procedures

Use one of the two procedures shown in the following paragraphs, depending on whether you have an IBM AT Diagnostics diskette available. Use Procedure I if you have the disk, otherwise use Procedure II.

### Procedure I: IBM AT Diagnostics Diskette Available

Use this procedure in the case that you have a copy of the *DOS User's Manual* and an IBM AT Diagnostics diskette.

1. Backup the SunIPC logical hard disk contents onto floppy disks. Use either the MS-DOS `copy` or `backup` command. See the *DOS User's Manual* for command definitions if needed. The logical disk, drive C, contains the MS-DOS, NFS, GWBASIC, and system utility files included with the SunIPC board, plus any user files.

   Note that additional backup procedures may be required, depending on your application programs. Some application programs create 'hidden'

files that may not be copied unless you use a special backup procedure. This is part of some application programs' software protection schemes. Refer to your application program user manual for any special backup procedures.

2. Make a new system floppy disk. Insert a blank floppy disk in drive A. Move to directory `c:\msdos` and enter the command shown below.

```
c:\msdos> format a:/s
```

   This command causes MS-DOS to copy the necessary system files from the SunIPC logical hard disk to the new system floppy disk.

3. Use the MS-DOS `copy` command to transfer the files listed below from drive C to the new system floppy disk in drive A. Note that you need to copy the `Restore.Com` file only if you used the `backup` command in step 1.

```
COMMAND.COM
FDISK.COM
FORMAT.EXE
RESTORE.COM
```

4. Remove the new system floppy disk from the SunIPC floppy disk drive A. Insert the IBM AT Diagnostics diskette and reboot the `PCTOOL`.

5. From the menu that appears, select option four, `setup`, and press <Return>.

6. When prompted, verify the correct date and time. Change the date and time as required.

7. When prompted with *The following options have been set:.... Are these options correct (Y/N)?*, press <N> and then press <Return>.

8. When prompted with *Are diskette drive types correct (Y/N)?*, press <Y> and then press <Return>. Do not change the floppy disk options.

9. When prompted with *Your fixed disk drive types are set to the following:.... Is this correct (Y/N)?*, press <N> and then press <Return>.

10. When prompted with *How many fixed disks are installed?*, press <1> and then press <Return>.

11. When prompted with *Enter fixed disk type (1-15) for fixed disk drive C.*, press <8> which signifies a 30 Mbyte hard disk.

12. Check the next screen to ensure that you have entered the correct disk type and then press <Y> if correct. Press <N> if incorrect and then repeat steps 10 through 12.

13. Do not change any subsequent options.

14. The final screen prompts you with the selected options and asks for verification that the options are correct. Check that the *Fixed Disks Drive C - Type* is type 8 for the 30 Mbyte hard disk. Also check that no other options were changed. Press <Y> if the options are correct and then press <Return>. Press <N> if the options are not correct, press <Return>, and then repeat steps 8 through 14.

15. Remove the IBM AT Diagnostics diskette from the SunIPC floppy disk drive A. Insert the new system floppy disk you created in steps 2 and 3.

16. Press <Return> or use the mouse to reset the `PCTOOL`.

17. `Reboot` the SunIPC from the new system floppy disk. You must `reboot` since you cannot change the disk size at the same time you are running SunIPC from that disk.

18. Run the MS-DOS `fdisk` utility from the new system floppy disk. This modifies the existing drive C to enlarge the logical hard disk.

19. Refer to the `fdisk` utility documentation in the *DOS User's Manual.*

20. First, select the third menu item, *Delete DOS Partition.* Second, select the first menu item, *Create DOS Partition.* Third, select the second menu item, *Changing the Active Partition.*

21. The `fdisk` utility forces you to `reboot` SunIPC again from the new system floppy disk once the utility has finished changing the logical hard disk partition.

22. Format the logical hard disk by entering the command shown below.

```
> format c:/s/v
```

23. Copy the files from the backup floppy disk(s) you created in step 1 onto the new, 30 Mbyte SunIPC logical hard disk. Use either the MS-DOS `copy` or the `restore` command, depending on whether you used the `copy` or the `restore` command to create the backup floppy disk(s).

24. `Reboot` the SunIPC from the logical hard disk on drive C.

The procedure is completed. You are now ready to use the SunIPC as usual.

**Procedure II: IBM AT Diagnostics Diskette not Available**

Use this procedure in the case that you *do not* have a copy of the *DOS User's Manual* and an IBM AT Diagnostics diskette. You will use the MS-DOS `debug` command to enlarge the size of the SunIPC logical hard disk.

Again, note that up to four SunIPC logical hard disks may be installed on your system. They use files `/usr/pctool/cmos_ram.pc0` through `/usr/pctool/cmos_ram.pc3`, respectively. In this article the case of a single SunIPC board and file `/usr/pctool/cmos_ram.pc0` is considered.

1. Backup the SunIPC logical hard disk contents onto floppy disks. Use either the MS-DOS `copy` or `backup` command. See the *DOS User's Manual* for command definitions if needed. The logical disk, drive C, contains the MS-DOS, NFS, GWBASIC, and system utility files included with the SunIPC board, plus any user files.

   Note that additional backup procedures may be required, depending on your application programs. Some application programs create 'hidden' files that may not be copied unless you use a special backup procedure. This is part of some application programs' software protection schemes. Refer to your application program user manual for any special backup procedures.

2. Make a new system floppy disk. Insert a blank floppy disk in drive A. Move to directory `c:\msdos` and enter the command shown below.

   ```
   c:\msdos> format a:/s
   ```

   This command causes MS-DOS to copy the necessary system files from the SunIPC logical hard disk to the new system floppy disk.

3. Use the MS-DOS `copy` command to transfer the files listed below from drive C to the new system floppy disk in drive A. Note that you need to copy the `Restore.Com` file only if you used the `backup` command in step 1.

   ```
   COMMAND.COM
   FDISK.COM
   FORMAT.EXE
   RESTORE.COM
   ```

4. From a UNIX window, copy file `/usr/pctool/cmos_ram.pc0` to a file named `cmos-tmp` in a directory that is both accessible and mountable via PC-NFS.

5. From a PCTOOL or a PC running PC-NFS on your network, continue with this procedure and perform the following steps.

6. Use the PC-NFS `NET USE` command to mount the UNIX directory containing the `cmos-tmp` file you made in step 4. An example is shown below.

   ```
   NET USE <?>: \\<host>\<dir>...
   ```

7. Change the current hard disk to the PC-NFS volume by issuing a `?:` where `?:` is the drive designation you used in the `NET USE` command example shown in step 6.

8. Type the DOS command `debug cmos-tmp` and then press <Return>.

9. You now see the `debug` prompt, a dash, on the left side of the screen. The next 16 steps (steps 10 through 25) are done from the `debug` prompt. Note that <sp> signifies typing a space using the space bar, and <Return> signifies pressing the <Return> key. Type each command exactly as shown in steps 10 through 25.

10. e <sp> 100 <Return>

11. 26 <Return>

12. e <sp> 102 <Return>

13. 16 <Return>

14. e <sp> 112 <Return>

15. 80 <Return>

16. e <sp> 114 <Return>

17. 33 <Return>

18. e <sp> 12F <Return>

19. 55 <Return>

20. e <sp> 142 <Return>

21. 45 <Return>

22. e <sp> 143 <Return>

23. 4A <Return>

24. w <Return>

25. q <Return>

26. Type the DOS command `debug cmos-tmp` and then press <Return>. You will again see the `debug` prompt, a dash.

27. d <sp> cs:100 <sp> L44 <Return>

28. Check that the screen obtained from step 27 contains the new values you entered in steps 10 through 23. A sample screen is shown below with the actual changes highlighted with asterisks (**).

```
-d cs:100 L44

33CC:0100  26 00 16 00 15 00 06 04-03 87 26 02 50 80 00 00   &.........&.P...
           **    **

33CC:0110  20 00 80 00 33 80 02 00-00 00 00 00 00 00 00 00    ...3...........
              **    **

33CC:0120  00 00 00 00 00 00 00 00-00 00 00 00 00 00 01 55   ...............U
                                                        **

33CC:0130  00 00 19 80 00 00 00 00-00 00 00 00 00 00 00 00   ................

33CC:0140  20 4C 45 4A
              ** **
```

29. If your screen obtained from step 27 matches the screen shown above, go to step 32, skipping steps 30 and 31.

30. If your screen obtained from step 27 *does not match* the screen shown above, from the `debug` dash prompt, type the `debug` command `q` and then press <Return>.

31. Type the MS-DOS command `del cmos-tmp` and then press <Return>. Go to step 4, and repeat this procedure by repeating steps 4 and 5. Then *skip* step 6, and repeat steps 7 through 29.

32. From the `debug` dash prompt, type the `debug` command `q` and then press <Return>.

33. If steps 6 through 32 were issued from a `PCTOOL`, use the right mouse button to 'quit' the `PCTOOL`.

34. Begin working from a UNIX window on a Sun workstation.

35. Copy the `cmos-tmp` file edited in this procedure to file `/usr/pctool/cmos_ram.pc0`. Note again that this procedure assumes that only one SunIPC logical hard disk is on your system. Up to four IPC boards may be installed using files `/usr/pctool/cmos_ram.pc0` through `/usr/pctool/cmos_ram.pc3`, respectively.

36. Insert the new system floppy disk you created in steps 1 through 3 into drive A.

37. Open a SunIPC window by entering `pctool` and pressing <Return> which boots from the new system floppy disk.

38. Run the MS-DOS `fdisk` utility from the new system floppy disk. This modifies the existing drive C to enlarge the logical hard disk.

39. Refer to the `fdisk` utility documentation in the *DOS User's Manual.*

40. First, select the third menu item, *Delete DOS Partition.* Second, select the first menu item, *Create DOS Partition.* Third, select the second menu item, *Changing the Active Partition.*

41. The `fdisk` utility forces you to reboot SunIPC again from the new system floppy disk once the utility has finished changing the logical hard disk partition.

42. Format the logical hard disk by entering the command shown below.

```
> format c:/s/v
```

43. Copy the files from the backup floppy disk(s) you created in step 1 onto the new, 30 Mbyte SunIPC logical hard disk. Use either the MS-DOS `copy` or the `restore` command, depending on whether you used the `copy` or the `restore` command to create the backup floppy disk(s).

44. Reboot the SunIPC from the logical hard disk on drive C.

For Further Information

Regardless of whether you used procedure I or II, the resulting file `cmos_ram.pc0` (for one SunIPC board) will now expand to a maximum of 30 Mbytes.

See the *SunIPC™ User's Guide*, part number 814-1002, chapter 4, 'Using Disks' for a discussion that includes the additional topics appearing below.

- differences between logical and physical hard disks
- creating a board-independent `autoexec.bat` file

- installing PC applications
- using disk drives D through V to work with NFS files
- reducing the logical hard disk size
- changing the logical hard disk location

## 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 interruptible 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 progress when you caused the interrupt will not eventually complete and unexpectedly start the interrupt routine of your driver.

## Device Driver Calls

### Device Driver Calls: Customer Preparation

Those Sun customers wishing to write a device driver may find information in this article helpful in getting the job done. Your first approach includes reading the manual *Writing Device Drivers for the Sun Workstation*, part number 800-1304.

After reading the manual, you may find that additional articles or manuals are needed. Many of those relating to device drivers used most often by the Customer Service Division (CSD) in both the United States Answer Center (USAC) and CSD Consulting, are listed at the end of this article.

You may also find that some or all of your device-driver questions may be answered by calling the USAC at **1-800-USA-4-SUN**. The USAC customer service engineer returning your call may be able to answer your questions, or will be able to refer you to CSD Consulting. Details of both the USAC and CSD Consulting support services follow.

In both cases, please have the information described in this article ready when you call. You will then have the needed answers to questions you will be asked by the USAC customer service engineer. This information will reduce the time needed to define the issues to be solved, and to identify the available solutions.

### Your Initial Call: 1-800-USA-4-SUN

Your first step is to call the Customer Service AnswerLine at **1-800-USA-4-SUN**. Sun customers outside the United States should contact their local support group and follow the local procedures.

Your call is then logged and dispatched to one of the USAC support groups. Please see the article 'Using 800-USA-4-SUN' on page 567 of the September 1987 STB issue for details on how your call is processed and forwarded.

### Device Driver Questions

If your call is about writing a device driver, you can expect to be asked the questions described in these paragraphs. You may find that the USAC can answer your questions, or you may be referred to CSD Consulting to purchase an existing CSD Consulting Special or to explore purchasing a new, customized Special. This referral is in the case that your question is beyond the scope of USAC phone support. The CSD Specials are provided on a Time and Materials (T&M) basis.

1. *What type of device driver is it?*
   This question is asked to determine if the device driver is for an unsupported Special, or for some other unsupported configuration. Unsupported Specials and configurations that may be referred to CSD Consulting on a T&M basis include are listed below.

- drivers requiring knowledge of and access to source code
- block or structured drivers for devices upon which file systems can be mounted
- sophisticated `pixrect` drivers
- network interface drivers (e.g. Ethernet or X.25)
- coprocessor drivers tightly coupled with the CPU and integrated below the kernel driver level
- serial communications multiplexors
- disk and tape drivers
- SCSI drivers
- low-level window system kernel code
- hardware architecture issues (e.g. hardware addresses of Memory Management Unit (MMU) tables and registers, segment map issues, and context register issues)

2. *Do you want to use undocumented kernel support routines?*

   You will be referred to CSD Consulting in this case since the area of interest is beyond Sun's released and documented products.

3. *Do you expect SunOS to be a real-time system?*

   SunOS is a time-sharing system. Refer to the Dennis Ritchie article entitled 'UNIX Time-Sharing System: A Retrospective'. Also, look to an upcoming STB issue for a discussion of differences between SunOS time-sharing processing and real-time processing.

4. *Do you need to lock the process text or data pages into physical memory?*

   A CSD Consulting Special is available to lock the pages into physical memory. Ask for `CONSULT-PLOCK`.

5. *Do you need high-speed disciplines?*

   Another CSD Consulting Special is available. Ask for `CONSULT-HSPEED`.

6. *Is your question about a device driver you purchased from Sun Consulting?*

   You will be referred to CSD Consulting since a Special is *not* a released product and is therefore not supported by the USAC. Specials are unique, case-by-case solutions to customer problems. Unless arranged and included at the time of purchase, Specials do not include USAC phone support or automatic fixes of reported bugs. CSD Consulting provides enhancement and modification services to Specials on a T&M basis.

7. *Do you have a driver written by a third party?*

   You will be referred to the third party in the case that your questions are about a third party driver.

8. *What model Sun workstation are your using?*

   This includes the exact model number within a product line, for example a Sun 3/50. Please see the Tip of the Month in this month's STB Questions and Answers section. This month's tip covers use of the *hostid(1)* command to determine your exact model and product line numbers.

9. *What SunOS release level are you using?*

   The SunOS release level could be very important in determining the nature of the problem you are having.

   Please see the short subject on page 205 of the June 1987 STB issue for details on how to determine the SunOS release level you are using.

10. *What other devices are on your system? In which cardcage slots?*

    The answer to this question includes both Sun products as well as those from third parties. The hardware configuration including cardcage slot locations is needed to help define the problem.

11. *What type of device is it?*

    The complete answer to this question includes the type of bus on your system (Multibus, VMEbus, SCSI). In the case of a VMEbus device, how many address bits and data bits does the device use? (vme24d16, vme16d16, vme32d16, vme32d16, and so forth).

12. *In which slot is the device for the device driver located?*

    In the case of a Sun-3 workstation, you should check the *Configuration Guide Sun-3 Product Family* for supported configurations if you are in doubt.

13. *Does this device do Direct Memory Access (DMA)? If so, does it do DMA between the device memory and the kernel address space? Between the device memory and user address space? Both?*

    Direct Virtual Memory Access (DVMA) is Sun's term for DMA. DVMA processing goes through the hardware Memory Management Units (MMUs). At least 20 bits are required to perform Multibus DVMA, and 24 bits to perform VMEbus DVMA.

14. *Have you verified that the device works by probing it with the PROM Monitor?*

    This procedure is described in Appendix A, *Writing Device Drivers for the Sun Workstation*, part number 800-1304.

15. *Are you writing a kernel device driver or a memory-mapped user process?*

    A quick way to get a device up and running is to write a user program first, a memory-mapped device driver. However, this is not possible for the device in the cases listed below.

    - interrupt-driven
    - does DMA
    - responds only to supervisor function codes as generated by the CPU
    - requires special processing such as done by `ioctl`

16. *What is the device entry in your* `/usr/sys/conf/<filename>` *kernel configuration file?*

    The term '<filename>' signifies the name you used for your kernel configuration file.

17. *If the device is a VME bus device, what is the address modifier?*

    Jumpers are set either on the board or by the driver when it writes the address modifier to a device register. The most commonly used address modifiers are listed in the *VMEbus Specification* and in the *User's Guide to the Sun3/100 VMEbus*, part number 800-1487.

    Use the address modifiers shown below when writing kernel device drivers (supervisor data space).

    ```
    0x0D    vme32
    0x2D    vme16
    0x3D    vme24
    ```

    Use the address modifiers shown below when writing memory-mapped device drivers (user processes and user data space).

    ```
    0x09    vme32
    0x29    vme16
    0x39    vme24
    ```

18. *What is the interrupt level set on the board?*

19. *What is the interrupt vector?*

20. *What is the bus grant level?*

    All Sun VMEbus devices should have the bus grant level set at 3, usually using jumpers on the board.

21. *What is the hardware address of the device on the bus?*

    This depends on the device type, Multibus or vme24, 32 or 16. The addresses are listed in tables in the manual *Writing Device Drivers for the Sun Workstation*, part number 800-1304.

22. *Have you checked the backplane jumpering?*

    Refer to the hardware installation manual shipped with your system.

23. *Are you designing your own board?*

    Refer to the *User's Guide to the Sun-3/100 VMEbus*, part number 800-1487. Sun products are compatible with the *Motorola VMEbus Specification, Revision B*. Portions of revision C are incompatible with Sun products. For example, in the area of unaligned data under revision C, the 68020 processor allows 32-bit transfers at odd addresses. The Sun architecture does not.

24. *Are you fluent in the C programming language? UNIX internals? Sun hardware? Sun software? Have you ever written a device driver for a UNIX system?*

    Unless you answer yes to all of these, you may be referred to CSD Consulting for expertise and support in these subjects.

25. *What is the EXACT error message that you are receiving, including case and punctuation?*

26. *Are you saving kernel core dumps?*

    Kernel core dumps are needed to analyze the conditions at the time the error occurred. A dump may be requested from you by the USAC and may be needed in determining the actual problem.

    Refer to the procedure in the Periodic Maintenance chapter in the *System Administration for the Sun Workstation*, part number 800-1323.

## Additional Device Driver Resources

Additional resources available to Sun customers include the following three-day course available through Customer Service Division (CSD) Education.

- Writing Device Drivers for the Sun Workstation

  *Objective* — This course prepares system programmers to write device drivers for Sun workstations.

  *Prerequisites* — This course requires a strong knowledge of the C programming language, particularly structures, pointers, and typecasting; and familiarity with UNIX and kernel services including system calls and system library functions. Previous experience with device drivers is helpful.

  *Agenda* — The course agenda includes the UNIX I/O system, UNIX device drivers, I/O devices (memory-mapped, DMA/DVMA, and programmed I/O), device driver services, and adding device drivers.

Additional resources also include calling your USAC AnswerLine as your first step to obtain information on purchasing a customized device driver.

- CSD Consulting Specials

  *USAC AnswerLine* — Please call using the USAC AnswerLine at **1-800-USA-4-SUN**, for proper dispatching of your call to the United States Answer Center (USAC). The USAC personnel will then forward your inquiry to CSD Consulting if needed, and contact you at a

later time with information on when a consultant will call. USAC may again follow up on your inquiry or service needs at a later time, as part of their support activity.

*Outside the USA* For those Sun customers outside the USA, please contact your local support group and follow the local procedures.

References for Further Information

Please refer to the publications listed below for further, detailed information on device drivers, the SunOS, and the UNIX system.

- *Writing Device Drivers for the Sun Workstation*, part number 800-1304
- *Configuration Guide Sun-3 Product Family*, no part number
- *User's Guide to the Sun-3/100 VMEbus*, part number 800-1487
- *Cardcage Slot Assignments and Backplane Configuration Procedures*, part number 813-2004
- *UNIX Interface Reference Manual*, part number 800-1303, Section 2, for details on system calls used in writing memory-mapped user programs
- *Software Technical Bulletin*, June 1987, part number 812-8701-05
- *Sun-3 Architecture: A Sun Technical Report*, no part number
- *The UNIX System: A Sun Technical Report*, no part number
- *Motorola VMEbus Specification, Revision B*, VMEbus Manufacturers Group, August 1982; for hardware engineers and designers
- *Pixrect Reference Manual*, part number 800-1254, Appendix A, 'Writing a Pixrect Driver'
- *Operating Systems: the Xinu Approach*, Doug Comer, Bell Labs, Prentice-Hall, 1984
- *The Design of the UNIX Operating System*, Maurice J. Bach, AT&T, Prentice Hall, 1986

- *The Bell System Technical Journal*, Volume 57, Number 6, July-August 1978

  - *UNIX Time-Sharing System: A Retrospective*, article, D. M. Ritchie

  - *UNIX Time-Sharing System: UNIX Implementation*, article, K. Thompson

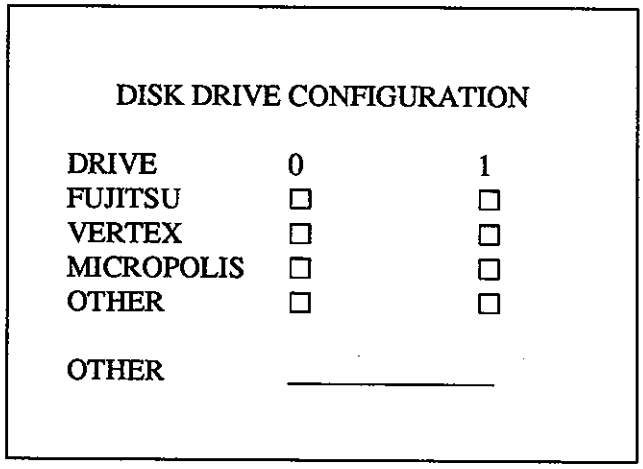
## Disks and Controllers

### Identifying Controller and Disk Configurations

It is often necessary for users to determine the controller/disk configuration of their Sun hardware. Use the following guidelines to determine the existing hardware configuration on different systems.

### Deskside Pedestal Label Information

Deskside pedestals have the following information included on a label located on the front cover of the pedestal. This label is visible after removing the gray faceplate.



DISK DRIVE CONFIGURATION

| DRIVE | 0 | 1 |
|---|---|---|
| FUJITSU | ☐ | ☐ |
| VERTEX | ☐ | ☐ |
| MICROPOLIS | ☐ | ☐ |
| OTHER | ☐ | ☐ |

OTHER _____

The appropriate disk(s) contained in the pedestal will be marked on this label.

The pedestal label does not specify the disk information listed below.

- Disk interface type -- Small Computer System Interface (SCSI), or Storage Module Disk (SMD)
- Disk capacity
- Disk model designation

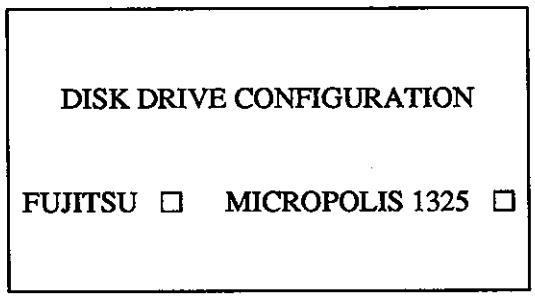
The disk interface type can be ascertained as shown below.

- Disks contained within the CPU pedestal are SCSI-type disks.
- Disks within expansion pedestals are SMD-type disks.

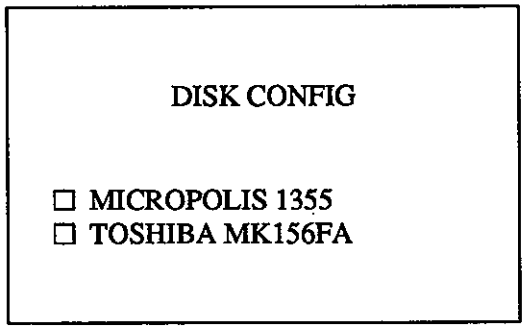
The disk capacity and model designation or both can be usually ascertained from the following information.

## Sun-3 'Shoebox' Disk Subsystem Label Information

Newer Sun-3 disk subsystems (commonly referred to as 'shoeboxes') have a small label affixed to the rear. For 71MB disk subsystems, the label appears as shown below.




For 141MB disk subsystems, the label appears as shown below.




All Sun-3 disk subsystems utilize the SCSI interface.

## Sun-3 System Controller and Disk Combinations

The following lists the controller and disk combinations used in Sun-3 systems.

| | |
|---|---|
| **Sun-3 'Shoebox' Disk Subsystems:** | |
| 71MB | Adaptec controller. Disks are primarily Micropolis 1325 and Fujitsu M2243AS. |
| 141MB | Emulex controller. Disks are Micropolis 1355 and Toshiba MK156F. |
| **Sun-3/160 with SCSI Disk(s) in the CPU Pedestal** | |
| 71 MB | Adaptec controller. Disks are primarily Micropolis 1325 and Fujitsu M2243AS. |
| 141MB | Emulex controller. Disks are Micropolis 1355 and Toshiba MK156F. |
| **Sun-3/160 and 3/260 with SMD Disks in an Expansion Pedestal:** | |
| 280MB | Xylogics 451 controller. Disk is the Fujitsu M2333. |
| **Rack-mount SMD Disks:** | |
| 575MB | Xylogics 451 controller. Disk is the Fujitsu M2361 Eagle XT (also known as the 'Super Eagle'). |

## Sun-2 System Controller and Disk Combinations

The following lists the controller and disk combinations used in Sun-2 systems.

| | |
|---|---|
| **Sun-2 'Shoebox' Disk Subsystems:** | |
| 71MB | Adaptec controller. Disks are primarily Micropolis 1325 and Fujitsu M2243AS. |
| **100U with SMD Disk(s):** | |
| '84MB' | Xylogics 450 controller. Fujitsu M2312K disk. This combination is also referred to as a 'FAT' box, for Fujitsu-disk and Archive Tape. |
| **Sun-2/120 with SCSI Disk(s) in the CPU Pedestal:** | |
| 42MB | Adaptec controller. Disks are Micropolis 1325 and Maxtor XT-1050. |
| 71MB | Adaptec controller. Disks are primarily Micropolis 1325 and Fujitsu M2243AS. |
| **Sun-2/130 and Sun-2/160 with SCSI Disk(s) in the CPU Pedestal:** | |
| 71MB | Adaptec controller. Disks are primarily Micropolis 1325 and Fujitsu M2243AS. |
| **Sun-2/120, Sun-2/130 and Sun-2/160 with SMD Disk(s) in an Expansion Pedestal:** | |
| 130MB | Xylogics 450 controller. Disk is the Fujitsu M2322. |
| **Rack-Mount SMD Disks:** | |
| '169MB' | Xylogics 450 controller. Disk is the Fujitsu M2284. This is only found in 150U and Sun-2/170 systems. |
| 380MB | Xylogics 450 controller. Disk is the Fujitsu M2351 Eagle. |

# SunOS Installation Aid

## System and SunOS Installation Aids

You may find the table shown below useful during system and UNIX installation. It summarizes controller and tape drive information needed when booting from the PROM monitor and then when using `diag`. The first four controllers are used for disk drives, and the last three controllers are used for cartridge and reel-to-reel tape drives.

This information appears in different tables in the references listed at the end of this article. For your convenience, use the single table below which brings together much of the information you will need to boot and to answer the prompts you will see when running `diag`. Please note that this table includes only those products currently available and does not include controllers or devices that are no longer available.

The controllers are described by model and by UNIX device name. Use the UNIX device name appropriate for your hardware to boot the general bootstrap program from the SunOS tape.

`diag` then prompts you for hardware information by asking what controller(s) and drive(s) your are installing on your system. Use the hexadecimal addresses shown to allow `diag` to configure itself to your hardware.

| Available Controllers and Drives | | | | | |
|---|---|---|---|---|---|
| Controller and UNIX Device Name | Address (Hex) | | | Drives | Remarks |
| | Machine Type | First Controller | Second Controller | | |
| Adaptec<br><br>sd | 3/50<br>Multibus<br>VMEbus | 140000<br>80000<br>200000 | NA<br>84000<br>NA | Fujitsu 2243<br>Micropolis 1304<br>Micropolis 1325<br>Vertex V185 | 71Mb, 5-1/4", SCSI<br>42Mb, 5-1/4", SCSI<br>71Mb, 5-1/4", SCSI<br>71Mb, 5-1/4", SCSI |
| Emulex MD21<br>sd | 3/50<br>Multibus<br>VMEbus | 140000<br>80000<br>200000 | NA<br>84000<br>NA | Micropolis 1355<br>Toshiba MK156F | 140Mb, 5-1/4", SCSI<br>140Mb, 5-1/4", SCSI |
| Xylogics 450<br><br>xy | all | ee40 | ee48 | CDC 9720<br>Fujitsu 2322<br>Fujitsu 2333<br>Fujitsu 2351 | 280Mb, 10-1/2", SMD<br>130Mb, 8", SMD<br>280Mb, 10-1/2", SMD<br>374Mb, 10-1/2", SMD, Eagle |
| Xylogics 451<br>xy | all | ee40 | ee48 | Fujitsu 2361 | 694Mb, 10-1/2", SMD, Super Eagle |
| Sysgen<br>st | NA | NA | NA | Archive<br>Cypher<br>Wangtech | 20Mb, 1/4" cartridge<br>20Mb, 1/4" cartridge<br>45Mb, 1/4" cartridge |
| TapeMaster<br>mt | NA | NA | NA | CDC | 30Mb, 1600bpi, 1/2" reel, 2400 feet |
| Xylogics 472<br>xt | NA | NA | NA | Fujitsu | 110Mb, 6250bpi, 1/2" reel, 2400 feet |

References for Further Information

Refer to the publications or courses listed below for details and further information on installing UNIX on your workstation using `diag` and `setup`, disk labeling and formatting, and kernel configuration.

- *Installing UNIX on the Sun Workstation*, part number 800-1521
- *Sun Installation and Networking, Training Manual*; Sun Microsystems, Inc.; 1987. Available with the installation and networking course shown below.
- *System Administration and Installation, 3.2 Version*; Sun Microsystems, Inc.; 1986. Available with the system administration course shown below.

Sun Educational Services

Two courses are available through Sun Educational Services that offer lectures, laboratory exercises, and the training manuals shown above.

- *Sun Workstation Installation and Networking*, 5 days, covering the topics listed below.

  - product overview and system configurations
  - hardware installation
  - introducing the Sun operating system
  - disk initialization and partitioning
  - UNIX installation
  - kernel configuration
  - networking requirements
  - remote tape installation
  - gateway installation

- *System Administration*, 5 days, covering the topics listed below.

  - product line overview
  - system structure and configuration
  - understanding disk drives
  - partitioning disks
  - system diagnostic utilities
  - installing UNIX
  - setting up and maintaining the network
  - ND (Network Disk) fileserver/client relationships
  - NFS (Network File System)
  - YP (Yellow Pages)
  - UNIX filesystem
  - dumps and file restoration
  - installing the mail system
  - adding peripheral devices

# Subnetting

## SunOS Release 3.3, Subnetting, and Restrictions

SunOS release 3.3 contains network subnetting. This article is intended to familiarize you with what subnetting is, in a general sense, and to introduce you to how it will be handled, with certain restrictions, in SunOS release 3.3.

The information presented in this in-depth article is taken from the Defense Advanced Research Projects Agency (DARPA) Internet Standard Subnetting Procedures specification and SunOS 3.3 release notes.

## Subnetting: Basic Questions and Answers

Before getting into the details, we can look at subnetting by answering commonly asked questions.

- What are subnets?

  Subnets are logically visible, sub-sections of a single Internet network. Many customers have chosen to divide one Internet network into several subnets for administrative or technical reasons.

- Why would I want to use a subnet?

  Many customers find that they need more network addresses than otherwise can be provided. They then divide the net address into subnet addresses and fit many more nodes into the same network address as originally provided by ARPA or the Department of Defense (DoD).

  Another reason to use a subnet is to reduce routing of redundant information. Many Internet sites have a large group of Internet Protocol (IP) networks and a direct connection to the ARPAnet or MILnet backbone. The gateway host has to export *all* internal network numbers to the Internet, usually using the Exterior Gateway Protocol (EGP). This is needed so that hosts on the internal net can route IP datagrams to other sites on the backbone via the gateway. However, this routing turns out to be useless information since internal networks must be reached from the backbone via the gateway machine anyway.

  Using subnets allows the gateway machine in this example to export a *single* IP network to the rest of the Internet, making itself the route by which other sites access that network. When the incoming IP datagram arrives at the gateway host, it is then treated as a subnet address and routed normally through the internal network.

- What addressing and subnetting scheme could I use?

  Many customers choose Internet addresses from ARPA/DoD, instead of just choosing addresses randomly. The subnetting scheme that the ARPA community and Sun has chosen is the 'address mask' approach.

- What security advantages does subnetting provide?

  There is no security difference. The main advantage is that you get more hosts online with fewer net addresses.

- Will subnetting function like running with the `routed -q` option on a network gateway system? Will users be forced to `rlogin` to the gateway system to get to machines 'on the other side' of the gateway?

  No. You should be able to `rlogin` to any machine, just like running `routed` normally. No multiple `rlogins` are required.

**The Address Mask Approach: Class A and Class B**

Two address classes are used in the address mask approach to subnetting. Class A and B addresses are described below.

Class A Compare the normal, non-subnetted class A address with the subnetted class A address shown below.

36.0.0.62 (0x24.00.00.3e)Non-Subnetted
36.40.0.62 (0x24.28.00.3e)Subnetted

The address mask for the address field portion constituting the network address for the subnet case is shown below.

255.255.0.0 (0xFF FF 00 00)

This means that the net part of the number and host number are each 16-bits wide.

Class B Compare the normal, non-subnetted class B address with the subnetted class B address shown below.

128.99.0.123 (0x80.63.00.7b)Non-Subnetted
128.99.4.123 (0x80.63.04.7b)Subnetted

The address mask for the address field portion constituting the network address for the subnet case is shown below.

255.255.252.0 (or 0xFF FF FC 00)

This means that the net part of the number is 22-bits wide and that the host part of the number is 10-bits wide.

The subnet mask must be the same on all subnets having the same IP network number. Take, for example, the IP network number x80.9b.0.0 or 128.155.0.0 as it is usually given in decimal. The default class B network mask would be 255.255.0.0, but with subnets the mask would be something like 255.255.255.0. Thus 128.155.24.x and 128.155.25.y are two different subnets of the same network.

Compare this example to those of 128.32.1.1 and 128.32.2.1 (Berkeley's numbers) given in the SunOS release 3.3 manual. This *is supported* in SunOS release 3.3, since it is only one non-standard network mask, 128.155 with 255.255.255.0. You can also have interfaces to any number of other non-subnetted (i.e. default netmask) networks.

One case not allowed in release 3.3 is two different subnetted networks interfaced to the same machine. That is, 128.32.1.2 and 128.155.24.1 on the same machine will not work in release 3.3.

You cannot, however, always know if an address is subnetted by simple inspection of the middle bytes. You can have subnet addresses containing middle bytes of zero (36.0.0.62, subnetted). You can also have non-subnetted addresses containing non-zero middle bytes (36.40.0.62, non-subnetted). Note that a zero subnet number is outside the DARPA Internet specification, but most implementations do not check for this. You can tell if a net is subnetted for sure only by inspecting the mask since subnetting is transparent outside of that network.

**SunOS Release 3.3 Subnetting Limitations**

The default address masks are shown below. Note that net masks must only be explicitly specified when they are "wider" (that is, have more one-bits) than the default values.

| | |
|---|---|
| 0xFF000000 (255.0.0.0) | Class A |
| 0xFFFF0000 (255.255.0.0) | Class B |
| 0xFFFFFF00 (255.255.255.0) | Class C |

It is important to note that all interfaces with non-default subnet masks must be on the same IP network. They may, however, be on different subnets. In other words, there cannot be more than one subnetted network interfaced to any machine. Usually, a workstation will be on only one subnet; a server will be a gateway between subnets of the same net, and possibly other non-subnetted networks. For instance, SunOS release 3.3 will gateway between two or more subnets of the same network, and will gateway between any number of subnets of the same network, as well as between any number of non-subnetted networks.

The only case not supported in SunOS release 3.3 is when you have more than one non-standard network mask, which would mean interfaces to subnets of two *different* IP networks. Interfaces to different subnets of the same network, or non-subnetted networks, may be freely intermixed in release 3.3.

Examples for class A and B networks are shown below.

128.32.0.0 class B network (subnetted)
netmask 255.255.255.0

36.0.0.0 class A network (subnetted)
netmask 255.255.0.0

10.0.0.0 class A network (non-subnetted)
netmask 255.0.0.0

128.32.1.1 and 128.32.2.1 and 10.0.0.78
are legal for SunOS Release 3.3
(two subnets of same net
+ non-subnetted network)

36.8.0.8 and 36.10.0.1 and 10.0.0.11
are legal for SunOS Release 3.3
(two subnets of same net
+ non-subnetted network)

128.32.1.1 and 128.32.2.1 and 36.8.0.8
are NOT legal for SunOS Release 3.3
(two subnets of same net
+ another subnetted network)

SunOS release 3.3 also fixes the broadcast problem. Some IP implementations send broadcasts with a normal, or subnet, network field. They send a host field for all-ones, however. This is correct and within the IP specification. It may, however, cause early Sun software to essentially bring the net down by broadcasting ARP requests for host 255.

Note that all-one host numbers, x.y.z.255 for class C nets for one example, work in previous SunOS releases, even though they are outside the IP specification. All-one host numbers are treated properly, as broadcast, starting with SunOS release 3.3.

In summary, do not use all-ones in the host portion of any host addresses. For example, host 255 is illegal in a class C network.

## Enabling Subnets

The kernel modules required to support subnets have been changed in SunOS release 3.3. `/etc/ifconfig` and `/etc/in.routed` have also changed. `/etc/in.routed` now manages the new routing tables. `/etc/ifconfig` now has a new option to set the network mask.

For example, let us say you have a class B network 128.32 that has an eight-bit wide subnet field, therefore an eight-bit wide host field, and a server that is both host 1 on subnet 37 and host 100 on subnet 3. The lines appearing in your `/etc/rc.boot` would then be as shown below.

```
/etc/ifconfig ie0 128.32.37.1 netmask 255.255.255.0 -trailers up
/etc/ifconfig ie1 128.32.3.100 netmask 255.255.255.0 -trailers up
```

Note that symbolic names defined in `/etc/hosts` can be used in lieu of the 128. ... numbers shown above.

# 3

# STB SHORT SUBJECTS

# 3

# STB SHORT SUBJECTS

## Using `boot`

### Using the `boot` Command from the PROM Monitor Prompt

Use the `boot` command to list the contents of the root directory when in the PROM monitor. This is particularly useful when `/vmunix` is corrupted or missing and you are looking for a backup version of the kernel.

An example of using `boot` from the PROM monitor prompt is shown below.

```
>b *
```

This will list all of the contents of the root partition. The sample result shown below is a typical listing for a Sun-3 server.

```
>b *
bin
boot
dev
etc
kadb
lib
lost+found
mnt
private
private.MC68020
pub
pub.MC68020
stand
sys
tftpboot
tmp
usr
usr.MC68020
vmunix
vmunix.gen
```

In this example, the backup version of the kernel is `vmunix.gen`.

# 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.

## Booting Kernels

### Booting a Specific, Customized Kernel

Some customer installations find that only a few clients of a particular server have different kernel requirements than the other clients. Instead of forcing the other clients into using a kernel containing features they do not need, or that might conflict with their operation, clients with different kernel requirements can boot their own 'private', customized kernel. The remaining clients can continue booting a common kernel. This short subject describes how to use this facility.

In the normal case, a client links to either `ndboot.sun2.pub0` or `ndboot.sun3.pub1` in the server's `/tftpboot` directory, depending on whether the client machine is a Sun2 or Sun3 and from which `pub` partition this client is booting.

The Sun3 diskless client booting process uses this '`/tftpboot` link' on its server for identifying whether to boot from `pub0`, `pub1`, or the client's private nd (network disk) partition. Currently, only booting a Sun3 diskless client uses this feature.

For example, a Sun3 client machine with an internet number of 192.9.4.5 would have the link in `/tftpboot` on its server as shown below.

```
C0090405 -> ndboot.sun3.pub1
```

The 'C0090405' link name is the client's Internet number in hexadecimal notation. Note that `pub1` is the default Sun3 public partition. This client boots the kernel `/pub.MC68020/vmunix` from its server.

In the case that you want the client to boot a specific kernel, change the link as shown below.

```
C0090405 -> ndboot.sun3.private
```

This can be done by either removing the existing link or renaming it. Next make the new link using *ln(1)*. For example, the command for the client in the above discussion is shown below.

```
ln -s ndboot.sun3.private C0090405
```

Now place the customized kernel into the client's root file system (its nd partition). This can be done from the server after halting this client and then mounting the client's root partition, or on the client machine itself. The next time this client boots, it will be using the new, private kernel.

# Power Interrupts

## Diskless Workstations, Power Failures, and Open Files

Some customers have uninterruptable power supplies to power their servers to ensure continuous operation. The question then arises 'what can get lost' from the diskless workstations when the power goes down.

Of importance to users working as the power goes down is the fate of open files. Some disk caching is done by the diskless clients. Of interest, then, is how much work can be lost in this case.

The `/etc/update` standard daemon syncs the file system every 30 seconds. When a `sync` command is issued on a diskless client, the contents of open files in the buffer cache are forced out to the file server.

The open-file contents are forced to the server's disk immediately upon arrival from the diskless client since no write caching is done by either ND or NFS servers.

However, if you are using the standard I/O library, you will lose any data still in the standard I/O buffers. See *flush(3)*.

## Using `history`

### Convenient `history` Command Usage

You may find a large value for `$history` helpful to have many of your recently-entered commands available for viewing. However, seeing several screenfuls of commands may be inconvenient at other times. Use the tip in this short subject to display either a large or a small list of recently-entered commands, depending on your needs at the time.

### Using a `history` Alias

The `csh` allows up to two arguments to the `history` command via the `-h` switch. In your `.cshrc` file, find the line that sets the number of recently-entered commands to be displayed when you use the `history` command. Then insert a new line as shown in the example below.

```
set history = 60
alias his   "history \!* 20"
```

The first line causes the last 60 recently-used commands to be displayed when you enter '*history*' on your command line and then press <return>.

The second line causes only the last 20 recently-used commands to be displayed when you enter '*his*' and then press <return>. Finally, if you enter '*his <n>*' only the last *n* number of commands will be printed, since the second option (20) to your alias is ignored in this case.

Please note that '*his -h <file>*' does not work, since there are now three arguments given to the `history` command.

## Packet Overload

### Back-to-Back Ethernet Packets

'Back-to-back' Ethernet packets are sent over the network to minimize dead-air time on the cable. This packet processing does not involve any actual negotiation, except in the case of a collision. Ignoring input issues, the logic involved is shown below.

**1** Is an output packet command available?

If no,
go to **1**.

**2** Is an ether available, no incoming packet in progress?

If no,
go to **2**.

**3** Start packet transmission and listen for a collision-detect.

**4** Is a collision detected?

If yes,
jam the Ethernet to force a collision-detect on all nodes,
run the backoff timer,
go to **2**.

**5** Go to **1**.

Software can queue several packet transmit commands to the controller chip. The packets so transmitted in steps 1 - 3 above are within the Ethernet specification. The time interval between packets can be as little as a few hundred nanoseconds at a typical 10 MHz chip clock rate.

Back-to-back packet transmission, though within the Ethernet specification on the transmit-side, can cause problems on the receive-side when the hardware buffer size is too small or controller processing speed too slow. For example, a typical Network File System (NFS) response is frequently a file system block, 8 Kb plus protocol, which becomes six Ethernet packets. This can make back-to-back packet processing common on Sun networks. Such packets may originate from the same or from two different nodes, all transparent to network users.

Problems arise on the receive-side when Ethernet controller implementations cannot process back-to-back packets without eventually dropping one. This occasionally occurs due to a slow controller. More often it is due to inadequate buffer size to store the next packet. When a packet is dropped, a protocol layer then has to timeout and retransmit.

For example, an Ethernet controller could have buffer space sufficient for two incoming packets only. A third packet is then dropped if it arrives before the first buffer has been emptied and released. In the case of NFS servers transmitting six consecutive packets and recovery occurring at the NFS level, all six packets have to be resent. This has resulted in new `mount` options to reduce the transmission rate in both directions.

Sun hardware includes sufficient buffer space to make this problem unlikely. Bridge vendors, however, can increase the chances of running into this problem if either buffer memory or processing power for the retransmission are inadequate. For example, 'throughput' rates of 4-6 Mbits/second across the bridge are typical. However, groups of six back-to-back, mostly-full packets represent a peak data rate approaching 10 Mbits/second.

## Bridge Box Limits

### Bridge Box Processing Power

Some customers have observed problems between two machines connected to each other through a Bridge box. With one machine mounted to the second machine's file system and all `fstab` entries correct, the error message shown below may result. This occurs with large files or during an attempt to run an executable. It does not occur when working with small files.

```
NFS Read Error for Server B
RPC timeout
```

### The Workaround

Many Ethernet Bridge boxes do not have the processing power necessary to keep up with a busy Ethernet, especially Sun workstations sending back-to-back packets on the net. It is possible to tune the NFS using parameters to the `mount` command to avoid placing too much processing demand on the Bridge box.

Use the parameters shown below for use over Sun Internetwork Routers.

```
rsize=512, wsize=512, timeo=100
```

These should cause the amount of back-to-back traffic to remain within the power of the Bridge box, at the price of reduced NFS throughput.

Refer to the preceding short subject *Packet Overload* for additional details on Bridge box processing problems associated with sending large files across busy networks.

# 4

# IN DEPTH

# 4

# IN DEPTH

## Internet Protocols

### Introduction to the Internet Protocols

This is an introduction to the Internet networking protocols (TCP/IP). It includes a summary of the facilities available and brief descriptions of the major protocols in the family.[10]

This document is an introduction to the transmission control protocol (TCP) and the Internet protocol (IP), followed by advice on what to read for more information. This is not intended to be a complete description. It can give you a reasonable idea of the capabilities of the protocols. Throughout the text, you will find references to the standards, in the form of request for comment (RFC) or IEN numbers. These are document numbers. The final section of this document tells you how to get copies of those standards.

### 1: What is TCP/IP?

TCP/IP is a set of protocols developed to allow cooperating computers to share resources across a network. It was developed by a community of researchers centered around the ARPAnet. Certainly the ARPAnet is the best-known TCP/IP network. However as of June 1987, at least 130 different vendors had products that support TCP/IP, and thousands of networks of all kinds use it.

First some basic definitions. The most accurate name for the set of protocols we are describing is the *Internet protocol suite*. TCP and IP are two of the protocols in this suite. Because TCP and IP are the best known of the protocols, it has become common to use the term TCP/IP or IP/TCP to refer to the whole family. However this can lead to some oddities. For example, one can talk about NFS as being based on TCP/IP, even though it does not use TCP at all. It does use IP. But it uses an alternative protocol, UDP, instead of TCP.

---

[10] Copyright (C) 1987, Charles L. Hedrick. Anyone may reproduce this document, in whole or in part, provided that: (1) any copy or republication of the entire document must show Rutgers University as the source, and must include this notice; and (2) any other use of this material must reference this manual and Rutgers University, and the fact that the material is copyrighted by Charles Hedrick and is used by permission.

The Internet is a collection of networks, including the ARPAnet, NSFnet, regional networks such as NYsernet, local networks at a number of university and research institutions, and a number of military networks. The term *Internet* applies to this entire set of networks. The subset of them that is managed by the Department of Defense is referred to as the Defense Data Network (DDN). This includes some research-oriented networks, such as the ARPAnet, as well as more strictly military ones. Because much of the funding for Internet protocol developments is done via the DDN organization, the terms Internet and DDN can sometimes seem equivalent.

All of these networks are connected to each other. Users can send messages from any of them to any other, except where there are security or other policy restrictions on access. Officially speaking, the Internet protocol documents are simply standards adopted by the Internet community for its own use. More recently, the Department of Defense issued a MILSPEC definition of TCP/IP. This was intended to be a more formal definition, appropriate for use in purchasing specifications. However, most of the TCP/IP community continues to use the Internet standards. The MILSPEC version is intended to be consistent with it.

Thus, TCP/IP is a family of protocols. A few provide 'low-level' functions needed for many applications. These include IP, TCP, and the user datagram protocol (UDP). Others are protocols for doing specific tasks, e.g. transferring files between computers, sending mail, or finding out who is logged in on another computer. Initially TCP/IP was used mostly between minicomputers or mainframes. These machines had their own disks, and generally were self-contained. Thus the most important 'traditional' TCP/IP services are described below.

File Transfer

The file transfer protocol (FTP) allows a user on any computer to get files from another computer, or to send files to another computer. Security concerns require the user to specify a user name and password for the other computer. Provisions are made for processing file transfers between machines with different character sets, end-of-line conventions, and the like. This is not quite the same thing as more recent *network file system* or *netBIOS* protocols, which will be described below. Rather, FTP is a utility that you run any time you want to access a file on another system. You use it to copy the file to your own system. You then work with the local copy. See RFC 959 for specifications for FTP.

Remote Login

The network terminal protocol (TELNET) allows a user to log in on any other computer on the network. You start a remote session by specifying a computer to connect to. From that time until you finish the session, anything you type is sent to the other computer. Note that you are really still talking to your own computer. But the `telnet` program effectively makes your computer invisible while it is running.

Every character you type is sent directly to the other system. Generally, the connection to the remote computer behaves much like a dialup connection. That is, the remote system will ask you to log in and give a password, in whatever manner it would normally ask a user who had just dialed it up. When you log off of the other computer, the `telnet` program exits, and you will find yourself talking to your own computer. Microcomputer implementations of `telnet` generally include a terminal emulator for some common types of terminals. See RFCs 854 and 855 for specifications for `telnet`. Note that the `telnet` protocol should not be confused with Telenet, a vendor of commercial network services.

Computer Mail

This allows you to send messages to users on other computers. Originally, people tended to use only one or two specific computers. They would maintain mail files on those machines. The computer mail system is simply a way for you to add a message to another user's mail file. There are some problems with this in an environment where microcomputers are used. The most serious is that a microcomputer is not well suited to receive computer mail.

When you send mail, the `mail` software expects to be able to open a connection to the addressee's computer, in order to send the mail. If this is a microcomputer, it may be turned off, or it may be running an application other than the mail system. For this reason, mail is normally processed by a larger system, where it is practical to have a mail server running all the time. Microcomputer mail software then becomes a user interface that retrieves mail from the mail server.

See RFCs 821 and 822 for specifications for computer mail. See RFC 937 for a protocol designed for microcomputers to use in reading mail from a mail server.

These services should be present in any implementation of TCP/IP, except that micro-oriented implementations may not support computer mail. These traditional applications still play a very important role in TCP/IP-based networks. However, more recently, the way in which networks are used has been changing. The older model of a number of large, self-sufficient computers is beginning to change. Now many installations have several kinds of computers, including microcomputers, workstations, minicomputers, and mainframes. These computers are likely to be configured to perform specialized tasks. Although people are still likely to work with one specific computer, that computer will call on other systems on the net for specialized services.

This has led to the *server/client* model of network services. A server is a system that provides a specific service for the rest of the network. A client is another system that uses that service. Note that the server and

client need not be on different computers. They could be different programs running on the same computer.

The kinds of servers typically present in a modern computer setup are described below. Note that these computer services can all be provided within the framework of TCP/IP.

Network File Systems

This allows a system to access files on another computer in a somewhat more closely integrated fashion than FTP. A network file system provides the illusion that disks or other devices from one system are directly connected to other systems. There is no need to use a special network utility to access a file on another system. Your computer simply thinks it has some extra disk drives. These extra 'virtual' drives refer to the other systems' disks. This capability is useful for several different purposes. It lets you put large disks on a few computers, but still give others access to the disk space.

Aside from the obvious economic benefits, this allows people working on several computers to share common files. It makes system maintenance and backup easier, because you do not have to worry about updating and backing-up copies many different machines. A number of vendors now offer high-performance, diskless computers. These computers have no disk drives at all. They are entirely dependent upon disks attached to common *file servers.*

See RFCs 1001 and 1002 for a description of PC-oriented NetBIOS over TCP. In the workstation and minicomputer area, Sun Microsystem's Network File System (NFS) is more likely to be used. Protocol specifications for it are available from Sun Microsystems, Inc.

Remote Printing

This allows you to access printers on other computers as if they were directly attached to yours. The most commonly used protocol is the remote lineprinter protocol from Berkeley UNIX. Unfortunately, there is no protocol document for this. However, the C code is easily obtained from Berkeley, so implementations are common.

Remote Execution

This allows you to request that a particular program be run on a different computer. This is useful when you can do most of your work on a small computer, but a few tasks require the resources of a larger system. There are a number of different kinds of remote execution. Some operate on a command-by-command basis. That is, you request that a specific command or set of commands should run on some specific computer. More sophisticated versions will choose a system that happens to be free. However, there are also *remote procedure call* systems that allow a program to call a subroutine that will run on another computer.

There are many protocols of this sort. Berkeley UNIX contains two servers to execute commands remotely: `rsh` and `rexec`. The man pages describe the protocols that they use. The user-contributed software with Berkeley 4.3 contains a *distributed shell* that distributes tasks among a set of systems, depending upon load. Remote procedure call mechanisms have been a topic for research for a number of years, so many organizations have implementations of such facilities. The most widespread, commercially-supported remote procedure call protocols (RPCs) seem to be Xerox's Courier and Sun Microsystem's RPC. Protocol documents are available from Xerox and Sun Microsystems. There is a public implementation of Courier over TCP as part of the user-contributed software with Berkeley 4.3. An implementation of RPC was posted to Usenet by Sun Microsystems, and also appears as part of the user-contributed software with Berkeley 4.3.

Name Servers

In large installations, there are a number of different collections of names that have to be managed. This includes users and their passwords, names and network addresses for computers, and accounts. It becomes tedious to keep this data up-to-date on all of the computers. Thus the databases are kept on a small number of systems. Other systems access the data over the network.

RFCs 822 and 823 describe the name server protocol used to keep track of host names and Internet addresses on the Internet. This is now a required part of any TCP/IP implementation. IEN 116 describes an older name server protocol that is used by a few terminal servers and other products to look up host names. Sun Microsystem's Yellow Pages system is designed as a general mechanism to process user names, file sharing groups, and other databases commonly used by UNIX systems. It is widely available commercially. Its protocol definition is available from Sun Microsystems.

Terminal Servers

Many installations no longer connect terminals directly to computers. Instead they connect them to terminal servers. A terminal server is simply a small computer that only knows how to run `telnet` or some other protocol to do remote login. If your terminal is connected to one of these, you simply type the name of a computer, and you are connected to it. Generally it is possible to have active connections to more than one computer at the same time. The terminal server will have provisions to switch between connections rapidly, and to notify you when output is waiting for another connection. Terminal servers use the `telnet` protocol, already mentioned. However, any real terminal server will also have to support name service and a number of other protocols.

Network-Oriented Window Systems

Until recently, high-performance graphics programs had to execute on a computer that had a bit-mapped graphics screen directly attached to it. Network window systems allow a program to use a display on a different computer. Full-scale network window systems provide an interface that lets you distribute tasks to the systems that are best suited to process them, but still give you a single graphically-based user interface. The most widely-implemented window system is X. A protocol description is available from MIT's Project Athena. A reference implementation is publically available from MIT. A number of vendors are also supporting NeWS, a window system defined by Sun Microsystems. Both of these systems are designed to use TCP/IP.

Note that some of the protocols described above were designed by Berkeley, Sun Microsystems, or other organizations. Thus they are not officially part of the Internet protocol suite. However, they are implemented using TCP/IP, just as normal TCP/IP application protocols are. Since the protocol definitions are not considered proprietary, and since commercially-support implementations are widely available, it is reasonable to think of these protocols as being effectively part of the Internet suite. Note that the list above is simply a sample of the sort of services available through TCP/IP. However, it does contain the majority of the 'major' applications. The other commonly-used protocols tend to be specialized facilities for getting information of various kinds, such as who is logged in, the time of day, and so forth. However, if you need a facility that is not listed here, look through the current edition of 'Internet Protocols', currently RFC 1011. It lists all of the available protocols, and also to look at some of the major TCP/IP implementations to see what various vendors have added.

## 2: General Description of the TCP/IP Protocols

TCP/IP is a layered set of protocols. In order to understand what this means, it is useful to look at an example. A typical situation is sending mail. First, there is a protocol for `mail`. This defines a set of commands which one machine sends to another, e.g. commands to specify who the sender of the message is, who it is being sent to, and then the text of the message. However, this protocol assumes that there is a way to communicate reliably between the two computers. `mail`, like other application protocols, simply defines a set of commands and messages to be sent. It is designed to be used together with TCP and IP.

TCP is responsible for making sure that the commands get through to the other end. It keeps track of what is sent, and retransmits anything that did not get through. If any message is too large for one datagram, e.g. the text of the mail, TCP will split it up into several datagrams, and make sure that they all arrive correctly. Since these functions are needed for many applications, they are put together into a separate protocol, rather than being part of the specifications for sending mail. You can think of TCP as forming a library of routines that applications can use when they need reliable network communications with another computer. Similarly, TCP calls on the services of IP.

Although the services that TCP supplies are needed by many applications, there are still some kinds of applications that do not need them. However, there are some services that every application needs. So these services are put together into IP. As with TCP, you can think of IP as a library of routines that TCP calls on, but which is also available to applications that do not use TCP. This strategy of building several levels of protocol is called *layering*. We think of the applications programs such as mail, TCP, and IP, as being separate *layers*, each of which calls on the services of the layer below it. Generally, TCP/IP applications use the four layers described below.

- an application protocol such as `mail`
- a protocol such as TCP that provides services needed by many applications
- IP, which provides the basic service of getting datagrams to their destinations
- the protocols needed to manage a specific physical medium, such as Ethernet or a point-to-point line

TCP/IP is based on the *catenet model*. This is described in more detail in IEN 48. This model assumes that there are a large number of independent networks connected together by gateways. The user should be able to access computers or other resources on any of these networks. Datagrams will often pass through a dozen different networks before getting to their final destinations. The routing needed to accomplish this should be completely invisible to the user.

As far as the user is concerned, all she or he needs to know in order to access another system is an Internet address. This is an address that looks like 128.6.4.194. It is actually a 32-bit number. However, it is normally written as four decimal numbers, each representing eight bits of the address.

The term *octet* is used by Internet documentation for such 8-bit sections. The term 'byte' is not used, because TCP/IP is supported by some computers that have byte sizes other than eight bits. Generally, the structure of the address gives you some information about how to get to the system. For example, 128.6 is a network number assigned by a central authority to Rutgers University. Rutgers uses the next octet to indicate which of the campus Ethernets is involved. 128.6.4 is an Ethernet used by the Computer Science Department. The last octet allows for up to 254 systems on each Ethernet. It is 254 because 0 and 255 are not allowed, for reasons that will be discussed later. Note that 128.6.4.194 and 128.6.5.194 would be different systems. The structure of an Internet address is described in more detail later.

People normally refer to systems by name, rather than by their Internet addresses. When we specify a name, the network software looks it up in a database, and finds the corresponding Internet address. Most of the network software deals strictly in terms of the address. RFC 882 describes the name server technology used to process this lookup.

TCP/IP is built on *connectionless* technology. Information is transferred as a sequence of *datagrams*. A datagram is a collection of data that is sent as a single message. Each of these datagrams is sent through the network individually. There are provisions to open connections, i.e. to start a conversation that will continue for some time. However, at some level, information from those connections is broken-up into datagrams, and those datagrams are treated by the network as completely separate.

For example, suppose you want to transfer a 15,000-octet file. Most networks can not process a 15,000-octet datagram. So the protocols will break this up into something like 30 separate, 500-octet datagrams. Each of these datagrams will be sent to the other end. At that point, they will be put back together into the 15,000-octet file. However, while those datagrams are in transit, the network does not know that there is any connection between them. It is possible that datagram 14 will actually arrive before datagram 13. It is also possible that somewhere in the network, an error will occur, and some datagram will not get through at all. In that case, that datagram has to be sent again.

Note that the terms *datagram* and *packet* often seem to be nearly interchangeable. Technically, datagram is correct to use when describing TCP/IP. A datagram is a unit of data, which is what the protocols process. A packet is a physical object, appearing on an Ethernet or some wire. In most cases a packet simply contains a datagram, so there is very little difference. However, they can differ. When TCP/IP is used on top of X.25, the X.25 interface breaks-up the datagrams into 128-byte packets. This is transparent to IP, because the packets are put back together into a single datagram at the other end before being processed by TCP/IP. So in this case, one IP datagram would be carried by several packets. However, with most media, there are efficiency advantages to sending one datagram per packet, and so the distinction tends to vanish.

## 2.1: The TCP Level

Two separate protocols are involved in processing TCP/IP datagrams. TCP is responsible for breaking-up the message into datagrams, reassembling them at the other end, resending anything that gets lost, and putting things back in the right order. IP is responsible for routing individual datagrams. It may seem like TCP is doing all the work. And in small networks that is true. However, in the Internet, simply getting a datagram to its destination can be a complex task.

For example, connection may require the datagram to go through several networks at Rutgers, a serial line to the John von Neuman Supercomputer Center, a couple of Ethernets there, a series of 56 Kbaud phone lines to another NSFnet site, and more Ethernets on another campus. Keeping track of the routes to all of the destinations and processing incompatibilities among different transport media turns out to be a complex task. Note that the interface between TCP and IP is

fairly simple. TCP simply hands IP a datagram with a destination. IP does not know how this datagram relates to any datagram before it or after it.

Clearly it is not enough to get a datagram to the right destination. TCP has to know which connection this datagram is part of. This task is referred to as *demultiplexing*. In fact, there are several levels of demultiplexing found in TCP/IP.

The information needed to do this demultiplexing is contained in a series of headers. A header is a few extra octets added to the beginning of a datagram by some protocol in order to keep track of it. It is a lot like putting a letter into an envelope and putting an address on the outside of the envelope. Except with modern networks it happens several times. It is like you put the letter into a little envelope, your administrator puts that into a somewhat bigger envelope, the campus mail center puts that envelope into a still bigger one, and so forth.

## Header Overview

An overview of the headers that get added to a message that passes through a typical TCP/IP network follows. We start with a single data stream, say a file you are trying to send to some other computer as shown below.

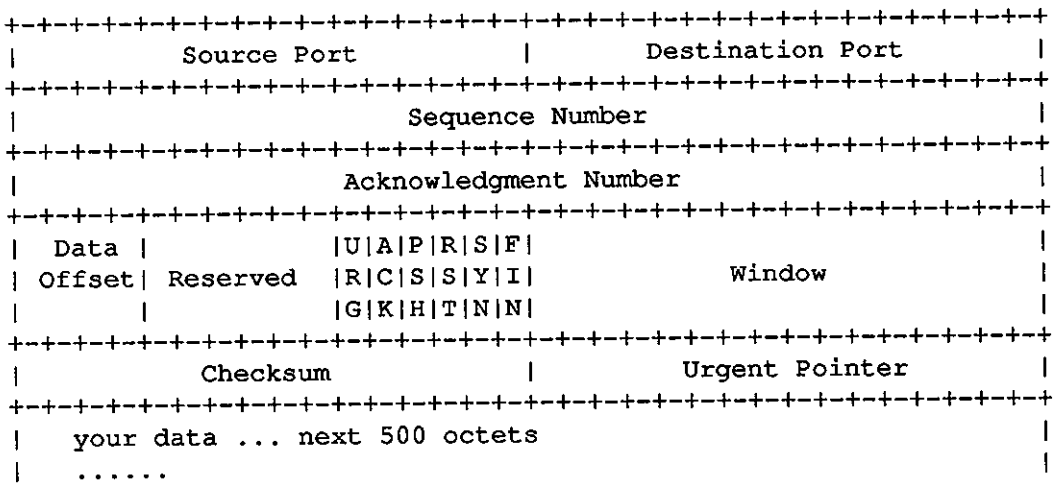
TCP breaks it up into manageable units. In order to do this, TCP has to know how large a datagram your network can process. Actually, the TCPs at each end say how large a datagram they can process, and then they pick the smallest size.

TCP puts a header at the front of each datagram. This header contains at least 20 octets, but the most important ones are a source and destination *port number* and a *sequence number*. The port numbers are used to keep track of different conversations. Suppose three different people are transferring files. Your TCP might allocate port numbers 1000, 1001, and 1002 to these transfers. When you are sending a datagram, this becomes the 'source' port number, since you are the source of the datagram. Of course, the TCP at the other end has assigned a port number of its own for the conversation.

Your TCP has to know the port number used by the other end as well. It finds out when the connection starts, as we will explain below. It puts this in the 'destination' port field. Of course, if the other end sends a datagram back to you, the source and destination port numbers will be reversed, since then it will be the source and you will be the destination.

Each datagram has a sequence number. This is used so that the other end can make sure that it gets the datagrams in the right order, and that it has not missed any. See the TCP specification for details. TCP does not number the datagrams, but the octets. So if there are 500 octets of data in each datagram, the first datagram might be numbered 0, the second 500, the next 1000, the next 1500, and so forth.

Checksum is a number that is computed by adding up all the octets in the datagram. See the TCP specification for details. The result is put in the header. TCP at the other end computes the checksum again. If they disagree, then something bad happened to the datagram in transmission, and it is discarded. The datagram now appears as shown below.




If we abbreviate the TCP header as 'T', then the whole file now looks as shown below.

```
T....   T....   T....   T....   T....   T....   T....
```

Note that there are items in the header not described above. They are generally involved with managing the connection. In order to make sure the datagram has arrived at its destination, the recipient has to send back an *acknowledgement*. This is a datagram whose 'acknowledgement number' field is filled in.

For example, sending a packet with an acknowledgement of 1500 indicates that you have received all the data up to octet number 1500. If the sender does not get an acknowledgement within a reasonable amount of time, it sends the data again. The window is used to control how much data can be in transit at any one time. It is not practical to wait for each datagram to be acknowledged before sending the next one. That would slow processing too much. On the other hand, you can not just keep sending, or a fast computer might overrun the capacity of a slow one to absorb data. Thus each end indicates how much new data it is currently prepared to absorb by putting the number of octets in its *window* field.

As the computer receives data, the amount of space left in its window decreases. When it goes to zero, the sender has to stop. As the receiver processes the data, it increases its window, indicating that it is ready to accept more data. Often the same datagram can be used to acknowledge receipt of a set of data and to give permission for additional new data, by an updated window. The *urgent* field

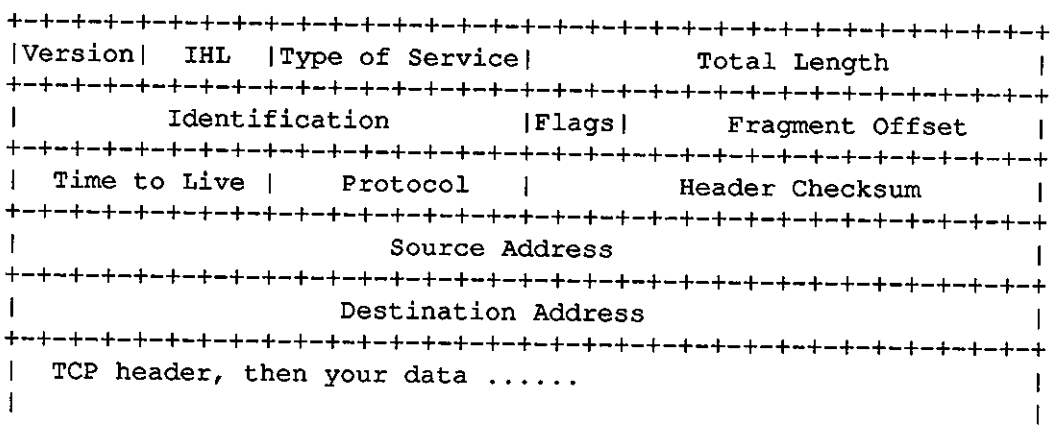
allows one end to tell the other to skip ahead in its processing to a particular octet. This is often useful for handling asynchronous events, for example when you type a control character or other command that interrupts output. The other fields are beyond the scope of this document.

## 2.2: The IP Level

TCP sends each of these datagrams to IP. Of course, it has to tell IP the Internet address of the computer at the other end. Note that this is the only IP concern. It does not care about what is in the datagram, or even in the TCP header. The IP task is to find a route for the datagram and get it to the other end. In order to allow gateways or other intermediate systems to forward the datagram, it adds its own header.

The main items in this header are the source and destination Internet address (32-bit addresses, like 128.6.4.194), the protocol number, and another checksum. The source Internet address is the address of your machine. This is necessary so the other end knows where the datagram came from. The destination Internet address is the address of the other machine. This is necessary so any gateways in the middle know where you want the datagram to go.

The protocol number tells the IP at the other end to send the datagram to TCP. Although most IP traffic uses TCP, there are other protocols that can use IP, so you have to tell IP which protocol to send the datagram to. Finally, the checksum allows IP at the other end to verify that the header was not damaged in transit. Note that TCP and IP have separate checksums. IP needs to be able to verify that the header did not get damaged in transit, or it could send a message to the wrong place. For reasons beyond the scope of this document, it is both more efficient and safer to have TCP compute a separate checksum for the TCP header and data. Once IP has added its header, the message appears as shown below.




If we represent the IP header by an 'I', your file now appears as shown below.

```
IT....   IT....   IT....   IT....   IT....   IT....   IT....
```

Again, the header contains some additional fields that have not been discussed.

Most of them are beyond the scope of this document. The flags and fragment offset are used to keep track of the pieces when a datagram has to be split up. This can happen when datagrams are forwarded through a network for which they are too large. The *time-to-live* is a number that is decremented when the datagram passes through a system. When it goes to zero, the datagram is discarded. This is done in case a loop develops in the system. Of course, this should be impossible, but well-designed networks are built to cope with 'impossible' conditions.

At this point, it is possible that no more headers are needed. If your computer happens to have a direct phone line connecting it to the destination computer, or to a gateway, it may simply send the datagrams out on the line. However, it is more likely that a synchronous protocol such as HDLC would be used, and it would add at least a few octets at the beginning and end.

## 2.3: The Ethernet Level

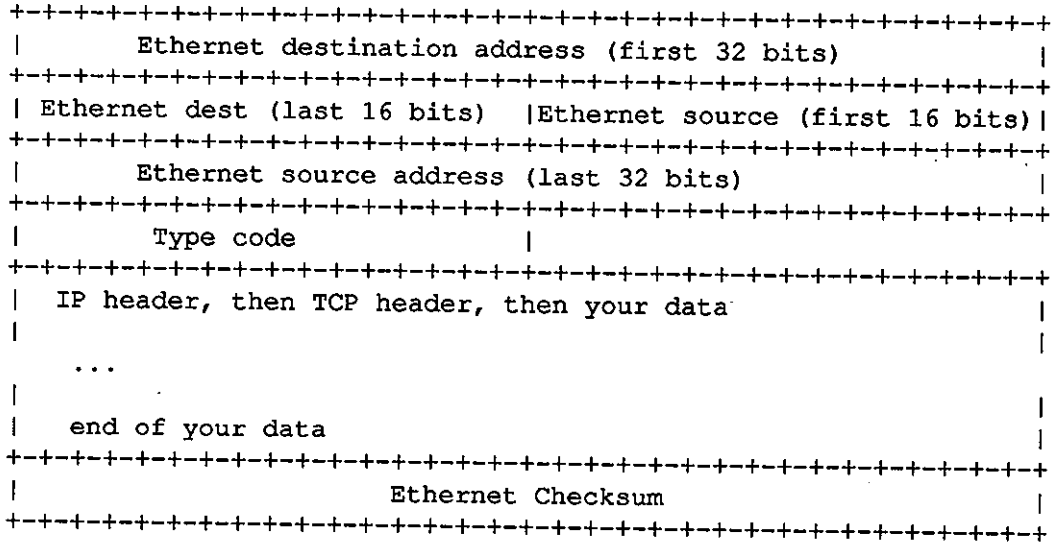
Most networks use Ethernet. Ethernet has its own headers and addresses. The Ethernet designers wanted to make sure that no two machines would have the same Ethernet address. Furthermore, they did not want the user to be concerned with assigning addresses. So each Ethernet controller comes with an address built-in from the factory.

In order to make sure that they would never have to reuse addresses, the Ethernet designers allocated 48 bits for the Ethernet address. Ethernet equipment manufacturers have to register with a central authority, to make sure that the numbers they assign do not overlap any other manufacturer. Ethernet is a 'broadcast medium'. That is, it is in effect shared usage, like an old 'party line' telephone. When you send a packet out on the Ethernet, every machine on the network sees the packet. So something is needed to make sure that the right machine gets it.

This involves the Ethernet header. Every Ethernet packet has a 14-octet header that includes the source and destination Ethernet address, and a type code. Each machine is supposed to pay attention only to packets with its own Ethernet address in the destination field. It is possible to cheat, which is one reason that Ethernet communications are not secure.

Note that there is no connection between the Ethernet address and the Internet address. Each machine has to have a table of which Ethernet address corresponds to which Internet address. In addition to the addresses, the header contains a *type code*. The type code is to allow for several different protocol families to be used on the same network. So you can use TCP/IP, DECnet, Xerox NS, and so forth, at the same time. Each of them will put a different value in the type field.

Finally, there is a *checksum*. The Ethernet controller computes a checksum of the entire packet. When the other end receives the packet, it recomputes the checksum, and throws the packet away if the answer disagrees with the original. The checksum is put on the end of the packet, not in the header. The final result is such that your message appears as shown below.




If we represent the Ethernet header with 'E', and the Ethernet checksum with 'C', your file now is as shown below.

```
EIT....C   EIT....C   EIT....C   EIT....C   EIT....C
```

When these packets are received by the other end, the headers are removed. The Ethernet interface removes the Ethernet header and the checksum. It looks at the type code. Since the type code is the one assigned to IP, the Ethernet device driver passes the datagram up to IP. IP removes the IP header. It looks at the IP protocol field. Since the protocol type is TCP, it passes the datagram up to TCP. TCP now looks at the sequence number. It uses the sequence numbers and other information to combine all the datagrams into the original file.

For detailed descriptions of the items discussed here, see RFC 793 for TCP, RFC 791 for IP, and RFCs 894 and 826 for sending IP over Ethernet.

### 3: Well-Known Sockets and the Applications Layer

There needs to be a way for you to open a connection to a specified computer, log into it, tell it what file you want, and control the transmission of the file. If you have a different application in mind, e.g. computer mail, some analogous protocol is needed. This is done by *application protocols*. The application protocols run 'on top of' TCP/IP. That is, when they want to send a message, they give the message to TCP. TCP makes sure it gets delivered to the other end. Because TCP and IP take care of all the networking details, the applications protocols can treat a network connection as if it were a simple byte stream, like a terminal or phone line.

Finding an application is a complex process. Suppose you want to send a file to a computer whose Internet address is 128.6.4.7. To start the process, you need more than just the Internet address. You have to connect to the FTP server at the other end. In general, network programs are specialized for a specific set of tasks. Most systems have separate programs to process file transfers, remote terminal logins, mail, and the like.

When you connect to 128.6.4.7, you have to specify that you want to talk to the FTP server. This is done by having *well-known sockets* for each server. Recall that TCP uses port numbers to keep track of individual conversations. User programs normally use random port numbers. However, specific port numbers are assigned to the programs that sit waiting for requests.

For example, if you want to send a file, you will start a program called `ftp`. It will open a connection using some random number, for example, 1234, for the port number on its end. However it will specify port number 21 for the other end. This is the official port number for the FTP server. Note that there are two different programs involved. You run `ftp` on your side. This is a program designed to accept commands from your terminal and pass them on to the other end. The program that you talk to on the other machine is the FTP server. It is designed to accept commands from the network connection, rather than from an interactive terminal. There is no need for your program to use a well-known socket number for itself. Nobody is trying to find it. However, the servers have to have well-known numbers, so that people can open connections to them and start sending them commands. The official port numbers for each program are given in 'Assigned Numbers', currently RFC 1010.

Note that a connection is actually described by a set of four numbers, the Internet address and the TCP port number at each end. Every datagram has all four of these numbers in it. The Internet addresses are in the IP header, and the TCP port numbers are in the TCP header.

No two connections can have the same set of numbers. However, it is enough for any one number to be different. For example, it is possible for two different users on a machine to be sending files to the same other machine. This could result in connections with parameters as shown below.

```
                  Internet addresses          TCP ports

connection 1  128.6.4.194, 128.6.4.7      1234, 21
connection 2  128.6.4.194, 128.6.4.7      1235, 21
```

Since the same machines are involved, the Internet addresses are the same. Since they are both doing file transfers, one end of the connection involves the well-known port number for FTP. The only item that differs is the port number for the program that the users are running. That single difference is sufficient. Generally, at least one end of the connection asks the network software to assign it a port number that is guaranteed to be unique. Normally, it is the user's end, since the server has to use a well-known number.

Once TCP has opened a connection, we have something that could be a simple wire. All the complex processing is performed by TCP and IP. However we still need some agreement regarding what we send over this connection. In effect, this is an agreement on what set of commands the application will understand, and the format in which they are to be sent.

Generally, what is sent is a combination of commands and data. They use context to differentiate. For example, the `mail` protocol works as follows. `mail` opens a connection to the mail server at the other end. Your program gives it your machine's name, the sender of the message, and the recipients you want it sent to. It then sends a command saying that it is starting the message. At this point, the other end stops treating what it sees as commands, and starts accepting the message. Your end then starts sending the text of the message. At the end of the message, a special mark is sent (a dot in the first column). After that, both ends understand that your program is again sending commands. This is the simplest method, and the one that most applications use.

File transfer is somewhat more complex. The file transfer protocol involves two different connections. It begins like `mail`. The user's program sends commands like 'log me in as this user', 'here is my password', and 'send me the file with this name'. However once the command to send data is sent, a second connection is opened for the data itself. It would be possible to send the data on the same connection, as `mail` does. However file transfers often take a long time. The designers of the file transfer protocol wanted to allow the user to continue issuing commands while the transfer being processed. For example, the user might make an inquiry, or she or he might abort the transfer. Thus the designers used a separate connection for the data and leave the original connection for commands. It is also possible to open command connections to two different computers, and tell them to send a file from one to the other. In that case, the data could not go over the command connection.

Remote terminal connections use a different mechanism. For remote logins, there is only one connection. It normally sends data. When it is necessary to send a command (for examples, to set the terminal type or to change a mode), a special character is used to indicate that the next character is a command. If the user happens to type that special character as data, two of them are sent.

A detailed description of the application protocols is beyond the scope of this document. Two common conventions used by applications are described here. First is the common network representation. TCP/IP is intended to be usable on any computer. Unfortunately, not all computers agree on how data is represented. There are differences in character codes (ASCII vs. EBCDIC), in end-of-line conventions (carriage return, line feed, or a representation using counts), and in whether terminals expect characters to be sent individually or a line-at-a-time. In order to allow computers of different kinds to communicate, each applications protocol defines a standard representation.

Note that TCP and IP do not care about the representation. TCP simply sends octets. However the programs at both ends have to agree on how the octets are to

be interpreted. The RFC for each application specifies the standard representation for that application. Normally it is 'net ASCII'. This uses ASCII characters, with end-of-line denoted by a carriage return followed by a line feed.

Second is the convention defining a 'standard terminal' for remote login. This is a half-duplex terminal with echoing happening on the local machine. Most applications also make provisions for the two computers to agree on other representations that they may find more convenient. For example, PDP-10s have 36-bit words. There is a way that two PDP-10s can agree to send a 36-bit binary file. Similarly, two systems that prefer full-duplex terminal conversations can agree on that. However, each application has a standard representation, which every machine must support.

### 3.1: An SMTP Application Example

An example of a simple mail transfer protocol (SMTP) follows. This is the mail protocol. Assume that a computer named TOPAZ.RUTGERS.EDU wants to send the following message.

```
Date: Sat, 27 Jun 87 13:26:31 EDT
From: hedrick@topaz.rutgers.edu
To: levy@red.rutgers.edu
Subject: meeting

Let's get together Monday at 1pm.
```

The format of the message itself is described by an Internet standard, RFC 822. The standard specifies that the message must be transmitted as net ASCII, i.e. it must be ASCII, with carriage return/linefeed to delimit lines. It also describes the general structure, as a group of header lines, then a blank line, and then the body of the message. Finally, it describes the syntax of the header lines in detail. Generally they consist of a keyword and then a value.

The addressee is indicated as 'LEVY@RED.RUTGERS.EDU'. Initially, addresses were simply 'person @ machine'. However, recent standards are more flexible. There are now provisions for systems to process other systems' mail. This allows automatic forwarding on behalf of computers not connected to the Internet. It can be used to direct mail for a number of systems to one central mail server. There is no requirement that an actual computer by the name of RED.RUTGERS.EDU even exist.

The name servers could be set up so that you mail to department names, and each department's mail is routed automatically to an appropriate computer. It is also possible that the part before the '@' is something other than a user name. It is possible for programs to be set up to process mail. There are also provisions to process mailing lists, and generic names such as 'postmaster' or 'operator'.

The way the message is to be sent to another system is described by RFCs 821 and 974. The program that is going to be doing the sending asks the name server several queries to determine where to route the message. The first query is to find out which machines process mail for the name RED.RUTGERS.EDU. In this case, the server replies that RED.RUTGERS.EDU processes its own mail.

The program then asks for the address of RED.RUTGERS.EDU, which is 128.6.4.2. Then the mail program opens a TCP connection to port 25 on 128.6.4.2. Port 25 is the well-known socket used for receiving mail. Once this connection is established, the mail program starts sending commands. A typical conversation appears below. Each line is labeled whether it is from TOPAZ or RED. Note that TOPAZ initiates the connection.

```
RED     220 RED.RUTGERS.EDU SMTP Service at 29 Jun 87 05:17:18 EDT
TOPAZ   HELO topaz.rutgers.edu
RED     250 RED.RUTGERS.EDU - Hello, TOPAZ.RUTGERS.EDU
TOPAZ   MAIL From:<hedrick@topaz.rutgers.edu>
RED     250 MAIL accepted
TOPAZ   RCPT To:<levy@red.rutgers.edu>
RED     250 Recipient accepted
TOPAZ   DATA
RED     354 Start mail input; end with <CRLF>.<CRLF>
TOPAZ   Date: Sat, 27 Jun 87 13:26:31 EDT
TOPAZ   From: hedrick@topaz.rutgers.edu
TOPAZ   To: levy@red.rutgers.edu
TOPAZ   Subject: meeting
TOPAZ
TOPAZ   Let's get together Monday at 1pm.
TOPAZ   .
RED     250 OK
TOPAZ   QUIT
RED     221 RED.RUTGERS.EDU Service closing transmission channel
```

First, note that the commands all use normal text. This is typical of the Internet standards. Many protocols use standard ASCII commands. This makes it simple to monitor and to diagnose problems. For example, the `mail` program keeps a log of each conversation. If something goes wrong, the log file can be mailed to the postmaster. Since it is normal text, she or he can determine what has occurred. It also allows a human to interact directly with the mail server, for testing.

Some newer protocols are complex enough that this is not practical. The commands would need a syntax requiring a significant parser. Thus there is a tendency for newer protocols to use binary formats. Generally they are structured like C or Pascal record structures.

Second, note that the responses all begin with numbers. This is also typical of Internet protocols. The allowable responses are defined in the protocol. The numbers allow the user program to respond unambiguously. The rest of the response is text, which is normally for use by any human who may be watching

or looking at a log. It has no effect on the operation of the programs. Note, however, there is one point at which the protocol uses part of the text of the response.

The commands themselves allow the `mail` program on one end to tell the mail server the information it needs to know in order to deliver the message. In this case, the mail server could get the information by looking at the message itself. But for more complex cases, that would not be safe. Every session must begin with a HELO, which gives the name of the system that initiated the connection. Then the sender and recipients are specified. There can be more than one RCPT command, if there are several recipients.

Finally, the data itself is sent. Note that the text of the message is terminated by a line containing a period. If such a line appears in the message, the period is doubled. After the message is accepted, the sender can send another message, or terminate the session as in the example above.

Generally, there is a pattern to the response numbers. The protocol defines the specific set of responses that can be sent as answers to any given command. However programs that do not want to analyze them in detail can look at the first digit only. Typically, responses that begin with a '2' indicate success. Those that begin with '3' indicate further action is needed, as shown above. Responses of '4' and '5' indicate errors. A '4' is a 'temporary' error, such as a disk filling. The message should be saved, and tried again later. A '5' is a permanent error, such as a non-existent recipient. The message should be returned to the sender with an error message.

For more details about the protocols mentioned in this section, see RFCs 821 and 822 for `mail`, RFC 959 for file transfer, and RFCs 854 and 855 for remote logins. For the well-known port numbers, see the current edition of Assigned Numbers, and possibly RFC 814.

## 4: UDP and ICMP Protocols

The discussion has included only connections that use TCP thus far. TCP is responsible for breaking-up messages into datagrams, and reassembling them properly. However, in many applications messages will fit into a single datagram. An example is name lookup. When a user attempts to make a connection to another system, she or he will generally specify the system by name, rather than by Internet address. The user's system has to translate that name to an address before it can do anything.

Generally, only a few systems have the database used to translate names to addresses. So the user's system will want to send a query to one of the systems that has the database. This query is going to be very short. It will certainly fit into one datagram, as will the answer. Thus it is not necessary to use TCP. Of course, TCP does more than just break messages up into datagrams. It also makes sure that the data arrives, resending datagrams where necessary. But for a question that fits in a single datagram, we do not need all the complexity of TCP to do this. If we do not get an answer after a few seconds, we can just ask again. For applications like this, there are alternatives to TCP.

The most common alternative is the user datagram protocol (UDP). UDP is designed for applications where you do not need to put sequences of datagrams together. It fits into the system much like TCP. There is a UDP header. The network software puts the UDP header on the front of your data, just as it would put a TCP header on the front of your data. Then UDP sends the data to IP, which adds the IP header, putting the UDP protocol number in the protocol field instead of the TCP protocol number.

However UDP does not do as much as TCP does. It does not split data into multiple datagrams. It does not keep track of what it has sent so it can resend if necessary. UDP provides port numbers, so that several programs can use UDP at once. UDP port numbers are used just like TCP port numbers. There are well-known port numbers for servers that use UDP. Note that the UDP header is shorter than a TCP header. It still has source and destination port numbers, and a checksum. No sequence number is present, since it is not needed. UDP is used by the protocols that process name lookups and a number of similar protocols. See IEN 116, RFC 882, and RFC 883.

Another alternative protocol is the Internet control message protocol (ICMP). ICMP is used for error messages, and other messages intended for the TCP/IP software itself, rather than by any particular user program. For example, if you attempt to connect to a host, your system may get back an ICMP message saying *host unreachable*. ICMP can also be used to find information about the network. See RFC 792 for details of ICMP. ICMP is similar to UDP in that it processes messages that fit in one datagram. However, it is even simpler than UDP. It does not have port numbers in its header. Since all ICMP messages are interpreted by the network software itself, no port numbers are needed to say where a ICMP message is supposed to go.

### 5: The Domain System: Keeping Track of Names and Information

The network software generally needs a 32-bit Internet address to open a connection or to send a datagram. However, users prefer use computer names rather than numbers. Thus, there is a database that allows the software to look up a name and find the corresponding number.

When the Internet was small, this was easy. Each system had a file that listed all of the other systems, giving both their name and number. There are now too many computers for this approach to be practical. Thus these files have been replaced by a set of name servers that keep track of host names and the corresponding Internet addresses. These servers are somewhat more general, this being just one kind of information stored in the domain system.

Note that a set of interlocking servers is used, rather than a single central one. There are now so many institutions connected to the Internet that it would be impractical for them to notify a central authority whenever they installed or moved a computer. Thus naming authority is delegated to individual institutions. The name servers form a tree, corresponding to institutional structure. The names themselves follow a similar structure. A typical example is the name 'BORAX.LCS.MIT.EDU'. This is a computer at the Laboratory for Computer Science (LCS) at MIT. To find its Internet address, you might have to consult four servers.

First, you would ask a central server, called the root, where the EDU server is. EDU is a server that keeps track of educational institutions. The root server would give you the names and Internet addresses of several servers for EDU. There are several servers at each level, to allow for the possibly that one might be down. You would then ask EDU where the server for MIT is. Again, it would give you names and Internet addresses of several servers for MIT. Generally, not all of those servers would be at MIT, to allow for the possibility of a general power failure at MIT.

Then you would ask MIT where the server for LCS is, and finally you would ask one of the LCS servers about BORAX. The final result would be the Internet address for BORAX.LCS.MIT.EDU. Each of these levels is referred to as a *domain*. The entire name, BORAX.LCS.MIT.EDU, is called a *domain name*. So are the names of the higher-level domains, such as LCS.MIT.EDU, MIT.EDU, and EDU.

You do not have to go do this most of the time. First, the root name servers also are the name servers for the top-level domains such as EDU. Thus, a single query to a root server will get you to MIT. Second, software generally remembers answers that it got before. So once we look up a name at LCS.MIT.EDU, our software remembers where to find servers for LCS.MIT.EDU, MIT.EDU, and EDU. It also remembers the translation of BORAX.LCS.MIT.EDU.

Each of these pieces of information has a *time-to-live* associated with it. Typically this is a few days. After that, the information expires and has to be looked up again. This allows institutions to make changes.

The domain system is not limited to finding Internet addresses. Each domain name is a node in a database. The node can have records that define a number of properties. Examples are Internet address, computer type, and a list of services provided by a computer. A program can ask for a specific piece of information, or all information about a given name. It is possible for a node in the database to be marked as an alias or nickname for another node. It is also possible to use the domain system to store information about users, mailing lists, or other objects.

There is an Internet standard defining the operation of these databases, as well as the protocols used to make queries of them. Every network utility has to be able to make such queries, since this is now the official way to evaluate host names.

Generally, utilities will talk to a server on their own system. This server will take care of contacting the other servers for them. This reduces the amount of code that has to be in each application program.

The domain system is particularly important for processing computer mail. There are entry types to define what computer processes mail for a given name, to specify where an individual is to receive mail, and to define mailing lists.

See RFCs 882, 883, and 973 for specifications of the domain system. RFC 974 defines the use of the domain system in sending mail.

### 6: Routing

The IP implementation is responsible for getting datagrams to the destination indicated by the destination address. The task of finding how to get a datagram to its destination is referred to as *routing*. In fact, many of the details depend on the particular implementation. However, some general statements may be made.

First, it is necessary to understand the model on which IP is based. IP assumes that a system is attached to some local network. We assume that the system can send datagrams to any other system on its own network. In the case of Ethernet, it simply finds the Ethernet address of the destination system, and puts the datagram out on the Ethernet. The problem comes when a system is asked to send a datagram to a system on a different network. This problem is processed by gateways.

A gateway is a system that connects a network with one or more other networks. Gateways are often normal computers that happen to have more than one network interface. For example, we have a UNIX machine that has two different Ethernet interfaces. Thus, it is connected to networks 128.6.4 and 128.6.3. This machine can act as a gateway between those two networks. The software on that machine must be set up so that it will forward datagrams from one network to the other.

If a machine on network 128.6.4 sends a datagram to the gateway, and the datagram is addressed to a machine on network 128.6.3, the gateway will forward the datagram to the destination. Major communications centers often have gateways that connect a number of different networks. In many cases, special-purpose gateway systems provide better performance or reliability than general-purpose systems acting as gateways. A number of vendors sell such systems.

Routing in IP is based upon the network number of the destination address. Each computer has a table of network numbers. For each network number, a gateway is listed. This is the gateway to use to get to that network. Note that the gateway does not have to connect directly to the network. It just has to be the best place to go to get there.

For example, at Rutgers our interface to NSFnet is at the John von Neuman Supercomputer Center (JvNC). Our connection to JvNC is via a high-speed, serial line connected to a gateway whose address is 128.6.3.12. Systems on net

128.6.3 will list 128.6.3.12 as the gateway for many off-campus networks. However, systems on net 128.6.4 will list 128.6.4.1 as the gateway to those same off-campus networks. Address 128.6.4.1 is the gateway between networks 128.6.4 and 128.6.3, so it is the first step in getting to JvNC.

When a computer wants to send a datagram, it first checks to see if the destination address is on the system's own local network. If so, the datagram can be sent directly. Otherwise, the system expects to find an entry for the network that the destination address is on. The datagram is sent to the gateway listed in that entry. This table can get quite long. For example, the Internet now includes several hundred individual networks. Thus, various strategies have been developed to reduce the size of the routing table. One strategy is to depend upon *default routes*. Often, there is only one gateway out of a network.

This single gateway might connect a local Ethernet to a campus-wide backbone network. In that case, we do not need to have a separate entry for every network in the world. We simply define that gateway as a *default*. When no specific route is found for a datagram, the datagram is sent to the default gateway. A default gateway can be used when there are several gateways on a network. There are provisions for gateways to send a message saying 'I am not the best gateway -- use this one instead'. The message is sent via ICMP. See RFC 792.

Most network software is designed to use these messages to add entries to their routing tables. Suppose network 128.6.4 has two gateways, 128.6.4.59 and 128.6.4.1. Address 128.6.4.59 leads to several other internal Rutgers networks. Address 128.6.4.1 leads indirectly to the NSFnet. Suppose we set 128.6.4.59 as a default gateway, and have no other routing table entries. Now what happens when we need to send a datagram to MIT?

MIT is network 18. Since we have no entry for network 18, the datagram will be sent to the default, 128.6.4.59. As it happens, this gateway is the wrong one. So it will forward the datagram to 128.6.4.1. But it will also send back an error saying in effect that 'To get to network 18, use 128.6.4.1.' Our software will then add an entry to the routing table. Any future datagrams to MIT will then go directly to 128.6.4.1. The error message is sent using the ICMP protocol. The message type is called *ICMP redirect.*

Most IP experts recommend that individual computers should not try to keep track of the entire network. Instead, they should start with default gateways, and let the gateways tell them the routes. However, this does not say how the gateways should find out about the routes. The gateways can not depend on this strategy. They require fairly complete routing tables. For this, a routing protocol is needed.

A routing protocol is a technique for the gateways to find each other, and to keep up-to-date about the best way to get to every network. RFC 1009 contains a review of gateway design and routing. `rip.doc` is an introduction to the subject. It contains some tutorial material, and a detailed description of the most commonly-used routing protocol.

## 7: Subnets and Broadcasting -- Internet Address Details

Internet addresses are 32-bit numbers, normally written as four octets (in decimal), e.g. 128.6.4.7. There are actually three types of address. The address has to indicate both the network and the host within the network. It was felt that eventually there would be numerous networks. Many of them would be small, but probably 24 bits would be needed to represent all IP networks. It was also felt that some very large networks might need 24 bits to represent all of their hosts. This would seem to lead to 48- bit addresses. But the designers wanted to use 32-bit addresses.

They adopted a compromise. The assumption is that most of the networks will be small. So they set up three ranges of address. Addresses beginning with one to 126 use only the first octet for the network number. The other three octets are available for the host number. Thus 24 bits are available for hosts. These numbers are used for large networks. But there can only be 126 of these very large networks. The ARPAnet is one, and there are a few large commercial networks.

Few normal organizations get one of these 'class A' addresses. For normal large organizations, 'class B' addresses are used. Class B addresses use the first two octets for the network number. Thus, network numbers are 128.1 through 191.254. We avoid zero and 255, for reasons described below. We also avoid addresses beginning with 127, because that is used by some systems for special purposes. The last two octets are available for host addresses, giving 16 bits of host address. This allows for 64,516 computers, which should be enough for most organizations. It is possible to get more than one class B address, if necessary.

Finally, class C addresses use three octets, in the range 192.1.1 to 223.254.254. These allow only 254 hosts on each network, but there can be many of these networks. Addresses above 223 are reserved for future use, as class D and E, which are currently not defined.

Many large organizations find it convenient to divide their network number into *subnets*. For example, Rutgers has been assigned a class B address, 128.6. We find it convenient to use the third octet of the address to indicate which Ethernet a host is on. This division has no significance outside of Rutgers. A computer at another institution would treat all datagrams addressed to 128.6 the same way. They would not look at the third octet of the address.

Thus, computers outside Rutgers would not have different routes for 128.6.4 or 128.6.5. But inside Rutgers, we treat 128.6.4 and 128.6.5 as separate networks. In effect, gateways inside Rutgers have separate entries for each Rutgers subnet, whereas gateways outside Rutgers have but one entry for 128.6. Note that we could do the same by using a separate class C address for each Ethernet. As far as Rutgers is concerned, it would be just as convenient for us to have a number of class C addresses. However using class C addresses would be inconvenient for the rest of the world.

Every institution that wanted to talk to us would have to have a separate entry for each one of our networks. If every institution did this, there would be far too many networks for any reasonable gateway to monitor. By subdividing a class B network, we hide our internal structure from everyone else, and save them the trouble. This subnet strategy requires special provisions in the network software. It is described in RFC 950.

Zero and 255 have special meanings. Zero is reserved for machines that do not know their address. In certain circumstances, it is possible for a machine not to know the number of the network it is on, or even its own host address. For example, 0.0.0.23 would be a machine that knew it was host number 23, but did not know on what network.

Address 255 is used for *broadcast*. A broadcast is a message that you want every system on the network to see. Broadcasts are used in some situations where you do not know who to talk to. For example, suppose you need to look up a host name and get its Internet address. Sometimes you do not know the address of the nearest name server. In that case, you might send the request as a broadcast. There are also cases where a number of systems are interested in information. It is then less expensive to send a single broadcast than to send datagrams individually to each host that is interested in the information.

In order to send a broadcast, you use an address that is made by using your network address, with all ones (1's) in the part of the address used for the host number. For example, if you are on network 128.6.4, you would use 128.6.4.255 for broadcasts. How this is actually implemented depends upon the medium. It is not possible to send broadcasts on the ARPAnet, or on point-to-point lines. However, it is possible on an Ethernet. If you use an Ethernet address with all ones (1's), every machine on the Ethernet is supposed to look at that datagram.

Although the official broadcast address for network 128.6.4 is now 128.6.4.255, there are some other addresses that may be treated as broadcasts by certain implementations. For convenience, the standard also allows 255.255.255.255 to be used. This refers to all hosts on the local network. It is often simpler to use 255.255.255.255 instead of finding the network number for the local network and forming a broadcast address such as 128.6.4.255. In addition, certain older implementations may use zero instead of 255 to form the broadcast address. Such implementations would use 128.6.4.0 instead of 128.6.4.255 as the broadcast address on network 128.6.4.

Finally, certain older implementations may not understand about subnets. Thus, they consider the network number to be 128.6. In that case, they will assume a broadcast address of 128.6.255.255 or 128.6.0.0. Until support for broadcasts is implemented properly, it can be a somewhat dangerous feature to use.

Because zero and 255 are used for unknown and broadcast addresses, normal hosts should never be given addresses containing zero or 255. Addresses should never begin with zero, 127, or any number above 223.

### 8: Datagram Fragmentation and Reassembly

TCP/IP is designed for use with many kinds of networks. Unfortunately, network designers do not agree on how large packets can be. Ethernet packets can be 1,500 octets long. ARPAnet packets have a maximum of approximately 1,000 octets. Some very fast networks have much larger packet sizes.

IP cannot simply settle on the smallest possible size. This would cause serious performance problems. When transferring large files, large packets are far more efficient than small ones. So we want to be able to use the largest packet size possible. But we also want to be able to communicate with networks using small packet limits.

There are two provisions for this. First, TCP has the ability to 'negotiate' datagram size. When a TCP connection first opens, both ends can send the maximum datagram size they process. The smaller of these limits is used for the rest of the connection. This allows two implementations that can process large datagrams to use them, but also lets them talk to implementations that cannot process them. However, this does not completely solve the problem. The most serious problem is that the two ends do not necessarily know about all of the steps in between.

For example, when sending data between Rutgers and Berkeley, it is likely that both computers will be on Ethernets. Thus they will both be prepared to process 1,500-octet datagrams. However the connection will at some point end up going over the ARPAnet. It can not process packets of that size. For this reason, there are provisions to split datagrams up into pieces. This process is referred to as *fragmentation.*

The IP header contains fields indicating that a datagram has been split, and enough information to let the pieces be put back together. If a gateway connects an Ethernet to the ARPAnet, it must be prepared to take 1,500-octet Ethernet packets and split them into pieces that will fit on the ARPAnet. Furthermore, every host implementation of TCP/IP must be prepared to accept pieces and put them back together. This is referred to as *reassembly.*

TCP/IP implementations differ in the approach they take to deciding on datagram size. It is fairly common for implementations to use 576-byte datagrams whenever they can not verify that the entire path is able to process larger packets. This rather conservative strategy is used because of the number of implementations with bugs in the code to reassemble fragments. Implementors often try to avoid ever having fragmentation occur. Different implementors take different approaches to deciding when it is safe to use large datagrams. Some use them only for the local network. Others will use them for any network on the same campus. A 'safe' size is 576 bytes, which every implementation must support.

### 9: ARP -- Ethernet Encapsulation

This discussion details how to determine which Ethernet address to use when you want to talk to a given Internet address. In fact, there is a separate protocol for this, called the address resolution protocol (ARP).

ARP is not an IP protocol. That is, the ARP datagrams do not have IP headers. Suppose you are on system 128.6.4.194 and you want to connect to system 128.6.4.7. Your system will first verify that 128.6.4.7 is on the same network, so it can talk directly via Ethernet. Then it will look up 128.6.4.7 in its ARP table, to see if it already knows the Ethernet address. If so, it will add an Ethernet header, and send the packet.

But suppose this system is not in the ARP table. There is no way to send the packet, because you need the Ethernet address. So it uses the ARP protocol to send an ARP request. Essentially an ARP request says 'I need the Ethernet address for 128.6.4.7.' Every system listens to ARP requests. When a system sees an ARP request for itself, it is required to respond. So 128.6.4.7 will see the request, and will respond with an ARP reply saying in effect '128.6.4.7 is 8:0:20:1:56:34.'

Recall that Ethernet addresses are 48 bits. This is six octets. Ethernet addresses are conventionally shown in hex, using the punctuation shown. Your system will save this information in its ARP table, so future packets will go directly. Most systems treat the ARP table as a cache, and clear entries in it if they have not been used in a certain period of time.

Note that ARP requests must be sent as broadcasts. There is no way that an ARP request can be sent directly to the right system. After all, the whole reason for sending an ARP request is that you do not know the Ethernet address. So an Ethernet address of all ones (1's) is used, i.e. ff:ff:ff:ff:ff:ff. By convention, every machine on the Ethernet is required to pay attention to packets with this as an address. So every system sees every ARP requests. They all look to see whether the request is for their own address. If so, they respond. If not, they could just ignore it. Some hosts will use ARP requests to update their knowledge about other hosts on the network, even if the request is not for them. Note that packets whose IP address indicates broadcast (e.g. 255.255.255.255 or 128.6.4.255) are also sent with an Ethernet address that is all ones (1's).

## 10: Getting More Information

The references for more information contained in the following paragraphs include some of the many documents describing the major protocols. Internet standards are called request for comments (RFCs). A proposed standard is initially issued as a proposal, and given an RFC number. When it is finally accepted, it is added to 'Official Internet Protocols', but it is still referred to by the RFC number.

We have also included two IENs, which used to be a separate classification for more informal documents. This classification no longer exists. RFCs are now used for all official Internet documents, and a mailing list is used for more informal reports. The convention is that whenever an RFC is revised, the revised version gets a new number. This is fine for most purposes, but it causes problems with two documents, *Assigned Numbers* and *Official Internet Protocols*. These documents are being revised all the time, so the RFC number keeps changing. You will have to look in `rfc-index.txt` to find the number of the latest edition. See RFC 791 which describes IP.

RFC 1009 is also useful. It is a specification for gateways to be used by NSFnet. As such, it contains an overview of a lot of the TCP/IP technology. Read the description of at least one of the application protocols. `mail` is a good one, RFCs 821 and 822. TCP 793 is of course a very basic specification. However, the specification is fairly complex.

## 10.1: Helpful General Documents

A number of helpful documents are described below.

| | |
|---|---|
| `rfc-index` | list of all RFCs |
| `rfc1012` | somewhat fuller list of all RFCs |
| `rfc1011` | Official Protocols. It is useful to scan this to see which tasks for which the protocols have been built. This defines which RFCs are actual standards and which are requests for comments. |
| `rfc1010` | Assigned Numbers. If you are working with TCP/IP, you will probably want a hardcopy of this as a reference. It lists all the officially defined well-known ports and other topics. |
| `rfc1009` | NSFnet gateway specifications. A good overview of IP routing and gateway technology. |
| `rfc1001/2` | netBIOS: networking for PCs |
| `rfc973` | update on domains |
| `rfc959` | FTP (file transfer) |
| `rfc950` | subnets |
| `rfc937` | POP2: protocol for reading mail on PCs |
| `rfc894` | how IP is to be put on Ethernet. See also rfc825. |
| `rfc882/3` | domains, the database used to go from hostnames to Internet address and back, also used to process UUCP. See also `rfc973`. |
| `rfc854/5` | `telnet`, a protocol for remote logins |
| `rfc826` | ARP, a protocol for finding Ethernet addresses |
| `rfc821/2` | `mail` |
| `rfc814` | names and ports, general concepts behind well-known ports |

| | |
|---|---|
| `rfc793` | TCP |
| `rfc792` | ICMP |
| `rfc791` | IP |
| `rfc768` | UDP |
| `rip.doc` | details of the most commonly-used routing protocol |
| `ien-116` | old name server, needed by several kinds of systems |
| `ien-48` | the Catenet model, general description of the philosophy behind TCP/IP |

## 10.2: Helpful Specialized Documents

The following documents are somewhat more specialized.

| | |
|---|---|
| `rfc813` | window and acknowledgement strategies in TCP |
| `rfc815` | datagram reassembly techniques |
| `rfc816` | fault isolation and resolution techniques |
| `rfc817` | modularity and efficiency in implementation |
| `rfc879` | the maximum segment size option in TCP |
| `rfc896` | congestion control |
| `rfc827,888,904,975,985` | EGP and related issues |

The most important RFCs have been collected into a three-volume set, the *DDN Protocol Handbook.* It is available from the DDN Network Information Center, SRI International, 333 Ravenswood Avenue, Menlo Park, California 94025, telephone (800) 235-3155. You should be able to get them via anonymous FTP from *sri-nic.arpa.* File names are shown below.

```
RFCs:   rfc:rfc-index.txt
        rfc:rfcxxx.txt

IENs:   ien:ien-index.txt
        ien:ien-xxx.txt
```

`rip.doc` is available by anonymous FTP from *topaz.rutgers.edu,* as `/pub/tcp-ip-docs/rip.doc`.

Sites with access to UUCP but not FTP may be able to retrieve them via UUCP from UUCP host rutgers. The file names would be as shown below.

```
RFCs:   /topaz/pub/pub/tcp-ip-docs/rfc-index.txt
        /topaz/pub/pub/tcp-ip-docs/rfcxxx.txt

IENs:   /topaz/pub/pub/tcp-ip-docs/ien-index.txt
        /topaz/pub/pub/tcp-ip-docs/ien-xxx.txt
        /topaz/pub/pub/tcp-ip-docs/rip.doc
```

Note that SRI-NIC has the entire set of RFCs and IENs, but `rutgers` and `topaz` have only those specifically mentioned above.

# Network Transfers

## Networking: Transfer of Information

This article contains an overview of several network-related topics. Most of the topics are software aspects of networking, with some hardware topics describing the physical network layout.

The major topics in this article are listed below.

- Ethernet theory of operation
- Network analysis and troubleshooting hints
- Network performance
- Subnet addressing
- Avoiding physical network problems
- Thin Ethernet (Cheapernet) specifications
- Level 1 and Level 2 equipment differences
- Frequently asked questions and answers

## Ethernet Theory of Operation

Ethernet theory of operation includes a definition of CSMA/CD and a description of how the single network channel (Ether) is used.

### CSMA/CD Definition

Ethernet activity is Carrier Sense Multiple Access with Collision Detection (CSMA/CD). CSMA/CD technology allows many devices access to the same network, in the absence of any central controller that manages channel access. Further, there are neither pre-allocated time slots as in token ring technology, nor fixed sharing of frequency bands.

Any device wanting to transmit onto the shared channel *contends* for channel (Ether) use until the channel is *acquired*. The device then transmits a packet onto the acquired channel. In this process, each device senses the channel carrier level and looks for collisions on the network.

### Ether Use

Each device wanting to transmit a packet of information onto the Ether first senses whether or not a carrier signal level is present on the Ether. One of two voltages is sensed, depending on whether another device on the network is transmitting a message at that moment in time. This is Carrier Sense (CS) and your device considers the net busy if a carrier is sensed.

Your device *defers transmitting* your message until it senses that the Ether is quiet. This is when no carrier voltage level is sensed and the net is then considered *not busy*. Once a quiet net is detected, the deferring device immediately begins transmitting the packet containing your message. This is Multiple Access (MA) since each device connected to the Ether shares the channel in real time, in some ways similar to a 'party-line' telephone service.

Your device listens for a collision during the time it takes to transmit the packet containing your message. This is the Collision Detection (CD) that lets each device know that its messages have been transmitted without any other device trying to 'break into' your 'conversation'. A collision is defined as any two devices transmitting to the same network at the same time. Unlike the old-fashioned, 'party line' telephone line, all messages are *retransmitted* if a collision is detected.

All devices look for collisions when transmitting since there is a short time interval just after a transmission begins when another device on the Ether might also begin to transmit its message. This occurs since the second device may not have sensed the carrier voltage that the first device places on the channel to transmit the packet containing the message.

This time interval is called the *collision window* or *collision interval*, after which all devices on the network will detect your carrier and defer their transmissions until yours is completed. This interval is a result of the end-to-end propagation delay on the net.

Your device continues to transmit the packet if no collision is detected during the collision window. However, each device continues to look for any collisions during the entire transmission process, in the case that a malfunctioning device is connected to the network. For example, a machine can lose the ability to detect a carrier on the network and therefore cannot sense collisions. This is known as *babble mode.*

If your station detects a collision, the current transmission is *aborted.* The network is briefly jammed by any connected station what detects a collision by invoking a collision consensus enforcement procedure. Basically, a given packet is fully transmitted only when no device on the net senses a collision, either during or after the collision window.

Each machine involved in the collisions then schedules its packet for *retransmission* at a later time. The two stations sharing the collision use a random delay period so that they each retransmit at a different time. This delay reduces the possibility that each station will retransmit at the same time and cause a second collision.

## Network Analysis and Troubleshooting Hints

This discussion of network analysis and troubleshooting includes running the proper SunOS release level with a proper netmask and diagnosing network routing problems.

SunOS Releases 3.3 and 3.4; and Subnetting

Most customer calls on networking result from trying to use subnets on SunOS release that do not support subnetting. You need to run SunOS release 3.3 or subsequent releases for subnetting. Further, you need to use the proper netmask. Subnet addressing and netmasks are discussed later in this article.

Routing Problem Troubleshooting

Several commands may be issued from your command line to diagnose network routing problems. Commands discussed in this topic are shown below. Please note that the following examples are provided for illustrative purposes only and do not represent actual networks.

- Using `netstat -r`
- Using `ifconfig`
- Using `/etc/networks`

Using `netstat -r`

When `ping` or `rlogin` report *Network Unreachable*, you will need to look at the network routing tables. This allows you to see what gateway the machines think they should use to access other networks. An example of output from the `network` command using the `-r` option is shown below.

```
machine# netstat -r

Routing tables
Destination     Gateway         Flags    Refcnt Use        Interface
sunpie-ptp      sunsnow         UGH      0      0          le0
dish-ptp        sunsnow         UGH      0      0          le0
backbone        sunsnow         UG       1      249        le0
gsd-localnet    sunsnow         UG       0      0          le0
supnet          machine         U        9      16631      le0
loopback        localhost       U        2      1363       lo0

machine#
```

Your device and your internal, loopback network are designated as being UP in the 'Flags' column by `U`. `G` denotes a machine that is a gateway to the network named in the 'Destination' column. `H` signifies the network host, the gateway to be used to gain access to the named network.

Using the `-r` option gives similar information with networks identified by the Internet subnet addresses. An example is shown below.

```
machine# netstat -n -r

Routing tables
Destination      Gateway          Flags    Refcnt Use        Interface
192.9.22.1       192.8.5.44       UGH      0      0          le0
192.9.21.1       192.8.5.44       UGH      0      0          le0
192.9.3          192.8.5.44       UG       1      249        le0
192.9.133        192.8.5.44       UG       0      0          le0
192.9.9          192.9.2.14       U        8      16949      le0
127.0.1          127.0.0.1        U        3      1396       lo0

machine#
```

### Using ifconfig

When ping or rlogin report *Connection Timed Out*, you will need to determine whether the Internet address has changed, or whether the gateway between the machines is down. The problem can also be lack of physical connection between machines.

Use the ifconfig command to determine the status of a network interface, its hosthame, and network address.

```
machine# ifconfig le0

le0: 192.9.1.14 flags=63<UP,BROADCAST,NOTRAILERS,RUNNING>

machine# ifconfig le0 netmask 0xffffff00
machine# ifconfig le0

le0: 192.9.1.14 netmask 255.255.255.0 flags=63<UP,BROADCAST,NOTRAILERS,RUNNING>
```

Note that ifconfig reports the netmask only if it has been set. If it has not been set, the appropriate default netmask shown below is in use.

| Network Address Class | Default Netmask (Hexadecimal) | Default Netmask (Decimal) |
|---|---|---|
| Class A | 0xff 00 00 00 | 255 00 00 00 |
| Class B | 0xff ff 00 00 | 255 255 00 00 |
| Class C | 0xff ff ff 00 | 255 255 255 00 |

### Using /etc/networks

The networks file is located in either the /etc or yp domain_name

directory. This file is needed for correct propagation of routing tables. This file allows you to more conveniently refer to networks by name rather than by Internet number. A sample `networks` file is shown below.

```
machine# more /etc/networks

#
# Sun customer networks
#
loopback    127
sun-ether   192.8.193   sunether ethernet localnet
#
# Internet networks
#
arpanet     10      arpa
#
# Local networks
#
ptp-net     192.2.300   my-net her-net
gymnet      128.8.3     # Home Ec subnet
solnet      192.8.21    # Solidarity Lab Network
nflsubnet   192.8.14.145    # NFL Subnet
lasernet    192.8.54    # Laser lab net
#
# Internetwork Routers
#
spdinr7     192.8.13    # Rond Road
spdinr8     192.8.14    # Rond Road
spdinr9     192.8.15    # Calistoga
```

**Network Performance**

Use the `netstat` and `nfsstat` commands to determine whether your network error, collisions, and nfs and nd statistics are within acceptable limits.

## Using netstat

Calculate your collision rate by first dividing the amount shown under 'Collis' by the amount shown under 'Opkts' and then multiplying the quotient by 100; (Collis/Opkts)*100=collision rate. Collision rate acceptability is great for a 1-2% range, fair for 3-5%, and poor for over 5%. Note that errors are never acceptable.

An example of `netstat` output using the `-i` option is shown below. This example shows a bad collision rate of 9.3%. This was calculated as 16477 / 177024 x 100 = 9.3%, a very bad collision rate.

```
machine# netstat -i

Name  Mtu   Net/Dest    Address      Ipkts   Ierrs Opkts   Oerrs Collis Queue
le0   1500  supnet      snowflake    832513  0     177024  1     16477  0
lo0   1536  loopback    localhost    1961    0     1961    0     0      0
```

A second use of `netstat` using the `-i` and `5` options is shown below. This shows the packets and errors for the input, output, and total input and output, for the time since the machine was booted, and every five seconds.

```
machine# netstat -i 5

    input   (le0)     output           input   (Total)    output
packets errs  packets errs  colls  packets errs  packets errs  colls
832857  0     177306  1     16486  834838  0     179287  1     16486
16      0     65      0     0      16      0     65      0     0
13      0     0       0     0      13      0     0       0     0
7       0     0       0     0      7       0     0       0     0
```

The final example shows `netstat` using the `-m` option. This option reports the buffers allocated to network processes.

```
machine# netstat -m

302/448 mbufs in use:
        65 mbufs allocated to socket structures
        83 mbufs allocated to protocol control blocks
       154 mbufs allocated to routing table entries
0/16 mapped pages in use
184 Kbytes allocated to network (20% in use)
0 requests for memory denied
```

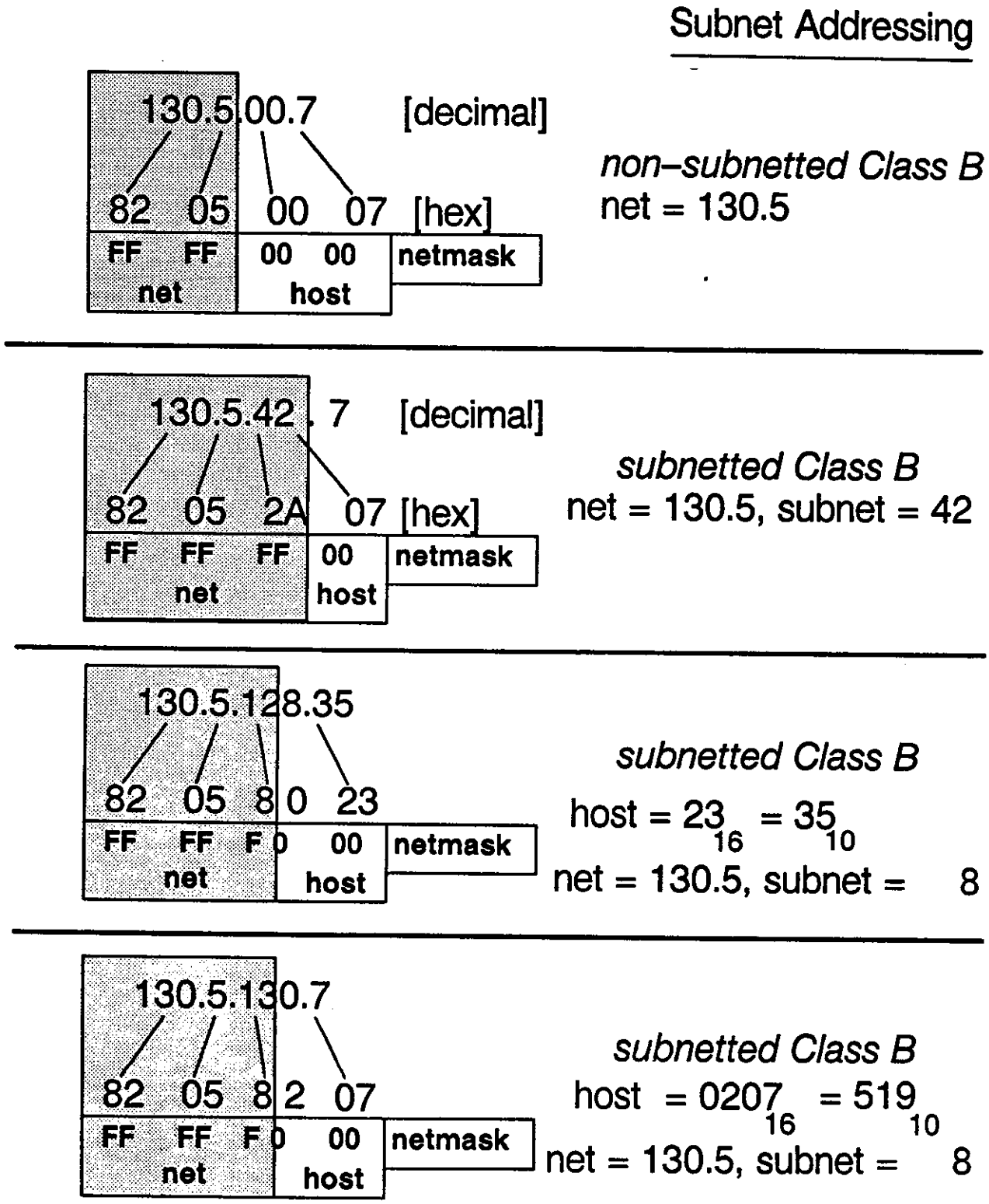
## Using nfsstat

Use the `nfsstat` command for checking nfs and nd statistics, particularly retransmission figures. These figures give you an idea of how good the network connection is. Look for timeouts and any entry labeled *bad....* A sample usage of `nfsstat` is shown below.

```
machine# /usr/etc/nfsstat

Network Disk:
rcv 188971  snd 161467  retrans 66  (0.04%)
notuser 0  noumatch 107  nobuf 0  lbusy 0  operrs 0
rseq 41  wseq 0  badreq 0  stimo 0  utimo 0  iseq 0

Server rpc:
calls      badcalls   nullrecv   badlen     xdrcall
0          0          0          0          0

Server nfs:
calls      badcalls
0          0
null       getattr    setattr    root       lookup     readlink   read
0  0%      0  0%      0  0%      0  0%      0  0%      0  0%      0  0%
wrcache    write      create     remove     rename     link       symlink
0  0%      0  0%      0  0%      0  0%      0  0%      0  0%      0  0%
mkdir      rmdir      readdir    fsstat
0  0%      0  0%      0  0%      0  0%

Client rpc:
calls      badcalls   retrans    badxid     timeout    wait       newcred
13048      0          3          1          3          0          0

Client nfs:
calls      badcalls   nclget     nclsleep
13048      0          13048      0
null       getattr    setattr    root       lookup     readlink   read
0   0%     4346 33%   10   0%    0   0%     2021 15%   4656 35%   1325 10%
wrcache    write      create     remove     rename     link       symlink
0   0%     93   0%    52   0%    1   0%     6   0%     0   0%     0   0%
mkdir      rmdir      readdir    fsstat
0   0%     0   0%     532  4%    6   0%
```

## Subnet Addressing

You cannot determine the meaning of a class B subnet address by inspection. The host address and net address share the address digits in one of several ways. See the following figure for one non-subnetted class B example, and three subnetted class B address examples.



## Avoiding Physical Network Problems

You can avoid problems with the network hardware by observing the following points.

- Avoid flourescent lights and power conduits when running network coaxial cables through the ceiling.
- Ground an Ethernet segment in a *single* place only. Multiple grounding points cause ground loops which may be misinterpreted by machines connected to the network as transmission carriers.
- Have 100 or fewer nodes on the net, not up to 1024 as indicated in the network specification.
- Place nodes only every 2.5m on the thick Ethernet coax. Any other placement causes standing waves on the cable which may be misinterpreted by machines. High error and collision rates may result.
- Place only one or two Ethernet repeaters between *any* two nodes.
- Place only one or two thick-to-thin Ethernet converters on a network.
- Have only a single network number on any one physical network. Multicasting is not supported in SunOS releases 3.x.
- Ensure that the bore is cleaned out using a swab when installing a vampire-type transceiver.
- Define only a single Ethernet address for a gateway machine. A gateway has only one Ethernet address, but two Internet addresses, and two hostnames. The Ethernet address originates from the CPU HostID chip. Note that an Ethernet address needs to be unique for any one network only. The same Ethernet address may be used on *different* networks.

## Thin Ethernet (Cheapernet) Specifications

The trunk cable (thin enet cable) is of constant impedance, coaxial construction. It is terminated at each of the two ends by a terminator of 50 ohms, plus or minus 1%, measured from 0 - 20 MHz. The terminator power rating shall be 0.5 watts or greater.[11]

The cable parameters are normally met by cable types RG58 A/U or RG 58 C/U. The center conductor shall be stranded, tinned copper with an overall diameter of 0.89mm, plus or minus 0.05mm. Devices attached to the trunk require a BNC 'T' connector of 50-ohm impedance. Note that this connector must be connected *directly* to the device, *without* any additional connecting cable.

[11] The information contained in this discussion is taken from *IEEE Draft Standard 802.3*, Section 10, 'Medium Attachment Unit and Baseband Medium Specifications', Type 10BASE2, dated March 1985.

The attenuation of a cable of the maximum 600 foot (185m) length shall not exceed 8.5dB measured at 10MHz, or 6.0dB measured at 5MHz. Also, the maximum length of a segment is 600 feet, with a maximum of 30 devices or Medium Attachment Units (MAUs) connected to the segment.

The maximum end-to-end propagation delay for a coaxial segment is 950ns.

The maximum transmission path permitted between any two MAUs is limited by the maximum of four repeater sets that can be connected in series. This shall consist of no more than three tapped coaxial segments; the remainder shall be link segments.

The minimum distance between any two nodes is 1.6 feet (0.5m). No more than 30 nodes shall be on a single network segment. *No cables are allowed between the BNC T-connector and the device!* Finally, the network may neither loop nor branch.

### Level 1 and Level 2 Equipment Differences

Ethernet hardware equipment is available in two versions or levels. These two versions are also known as DIX 1 and DIX 2 (DEC, Intel, Xerox). The two levels are compatible on the coaxial cable. However, a level 1 Ethernet board should be connected to a level 1 transceiver, and similarly for level 2 boards and transceivers. There are usually no problems communicating between level 1 and 2 systems across the coaxial cable, unless the network has grown extremely large.

Two features distinguish level 2 transceivers from level 1 equipment. The first feature is *jabber* which allows the transceiver to watch the packet data stream during transmission. If the data stream is longer than the maximum legal packet size, the level 2 transceiver shuts down. Note that this feature is not universally available on all level 2 transceivers. The jabber feature prevents a malfunctioning CPU from rendering the network unusable.

The second unique, level 2 feature is *heartbeat* which is a signal that occurs each time the transceiver receives a transmission from the attached device. The heartbeat signal is transmitted back to the device using a specific line in the transceiver cable.

The IEEE 802.3 standard is an Ethernet-like protocol, and is hardware-compatible with level 2 Ethernet equipment. However, the packet structure differs such that the two protocols do not work together on the software level. From the hardware perspective, IEEE 802.2 packets, IEEE 802.3 packets, and Ethernet packets *can* coexist on the same physical network. The Ethernet controllers read all three types of packets.

The driver software, however, must treat some of the packet fields differently. Therefore, IEEE 802.3 and Ethernet machines can talk to like-machines only on the same physical network, with unlike-machines being transparent.

Ethernet level 1 and level 2 differences are found in Ethernet controller and transceiver interaction only. The differences effect neither the electrical interface nor packets sent between transceivers on the coaxial cable. Note that the Sun Ethernet board can be configured as either level 1 or level 2.

**Frequently Asked Questions and Answers**

1. *What is the longest length of Ethernet cable in a network?*

   The longest length possible is 1640.5 feet (500m). This length can be composed of one full, continuous piece of cable. It can also be composed of segments of 23.4m, 70.2m, or 117m lengths.

   If odd-length cable sections must be used, choose the length so that the resulting reflected signals on the cable are not out of phase with the actual signal. This is usually accomplished by using lengths which are odd multiples of the half-wave length at 5Mhz. This length corresponds to the recommended segment lengths of 23.4m 70.2m, and 117m.

   Transceiver drop cables have a maximum length of 164 feet (50m).

2. *What is the maximum time between recognition of the collision and repeating of the collision?*

   The maximum time interval between collision recognition and a repeat collision excluding the carrier sense random retiming delays is 200ns.

3. *Should I ground the device or system?*

   Yes! The sheath conductor of the transceiver shall be connected to an earth ground or chassis, but at *one and only one* point per segment. Any second or additional ground points will only introduce degenerative ground loops, which are seen as a higher voltage level on the coaxial cable. This can be misinterpreted and sensed as a transmission carrier signal voltage.

4. *What are the proper lengths for adding transceivers to the network cable?*

   Transceivers may be added at a minimum of 2.5m from each other, or at multiples of 2.5m. Proper intervals are printed on the outside plastic sheath of the Ethernet coaxial cable.

5. *How many transceivers may be placed on the network?*

   A maximum of 100 nodes may be placed on the network.

6. *What are the timing constraints between a detected collision and carrier sense?*

   The channel logic must assert the collision-detect signal within 200ns following the collision. The channel must then deassert the collision-detect signal within 160ns after the loss of the collision-occurring signal. The carrier-sense is asserted when two stations are transmitting at the same time

must be asserted within 200ns. The channel then as 160ns to deassert carrier-sense after a carrier is no longer present on the network.

7. *What do I need to know about repeaters?*

   A repeater is a device used to extend the cable length beyond the single coaxial segment length of 500m. Repeaters require a transceiver at each of the segments for which it is repeating signals. A maximum of two repeaters may be in the signal path between any two transceivers.

   The repeater implements the carrier-sense and the collision-detect/repeat function for each cable segment it connects. All signals are retimed and amplified to allow for propagation to the other connected cable segment. With carrier-detect, the propagation delay through the repeater cannot exceed 800ns. With collision-detect and retransmission, the delay cannot exceed 200ns.

   The repeater ensures that the signal retransmitted from one cable segment to the other has the same amplification as it would when leaving a station's transceiver and entering the Ethernet coaxial cable.

8. *How do I know that a packet preamble is correct?*

   The preamble should appear as shown below. This 64-bit preamble may be viewed with proper software or a waveform analyzer. The preamble will appear on an oscilloscope as a periodic waveform of 5Mhz frequency.

```
10101010 10101010 10101010 10101010 10101010 10101010 10101010 10101011
```

## Sockets

### Socket Programming Explanations and Examples

This article hopefully will clarify using `SIGIO` and `SIGURG` with sockets and provides a couple of socket programming examples. Sun Software Information Services has been getting a lot of customer calls on this topic. Therefore, we will provide ongoing STB articles relating to sockets, dealing with one or two topics at a time. We will cover two topics in this article -- Sockets: Asynchronous I/O and Out-of-Band Data, and Asynchronous I/O and Internet Domain Stream Sockets.

### Sockets: Asynchronous I/O and Out-of-Band Data

The term *asynchronous I/O* used in connection with a socket refers to notifying the process or the process group associated with the socket when I/O is ready. This is done asynchronously and via a `SIGIO` signal. This facility is useful when you do not want to poll for pending I/O. An example is when the process wants to perform other functions while waiting for I/O ready.

There are several ways to request asynchronous notification of I/O on a socket descriptor by using `ioctl(2)` or `fcntl(2)` or both.

First, either the process or the process group associated with the socket must be set to receive the `SIGIO` signal. Use one of the system calls shown below to set the process group to receive the `SIGIO` signal. Note that the process group can be set to receive `SIGURG` signals using the same system calls.

```
int pid;

pid = -getpid();
if (fcntl(sock, F_SETOWN, pid) < 0)
    perror("fcntl: F_SETOWN");
if (ioctl(sock, FIOSETOWN, (char *)&pid) < 0)
    perror("ioctl: FIOSETOWN");
if (ioctl(sock, SIOCSPGRP, (char *)&pid) < 0)
    perror("ioctl: SIOCSPGRP");
```

Second, you need to set up the socket to receive asynchronous notification by using one of the system calls shown below.

```
int val;

if (fcntl(sock, F_SETFL, FASYNC) < 0)
    perror("fcntl: F_SETFL FASYNC");
val = 1;
if (ioctl(sock, FIOASYNC, (char *)&val) < 0)
    perror("ioctl: FIOASYNC");
```

**Example Server and Client Programs**

The following server and client programs illustrate using asynchronous notification of I/O under the Internet domain stream socket abstraction. The server first creates an Internet domain stream socket. The system selects `TCP` as an appropriate protocol. The server binds a name to the created socket, lets the system select a port number, and then obtains the port number.

The server listens in its main loop for a client to connect to the named socket. A new socket is created upon accepting a connection. The server then writes a message to the client on the new socket at two-second intervals. When the server receives a `SIGPIPE` signal it knows that the client has disconnected. This signal results from writing on the socket without a client to read it. The server then begins listening again for another client connection. This procedure is then repeated.

The server is terminated by a `SIGQUIT` signal, at which time the full-duplex connections on the two sockets are shut down and the socket descriptors are closed.

```
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/signal.h>
#include <netinet/in.h>

int Sig = 0;

char Buf[] = "Eat some chocolate!\n";
int Sock = 0;
int Msgsock = 0;

/* The Server */
main()
{
    int length;
    struct sockaddr_in server;
    int sigpipe(), sigquit();

        /* Create Internet domain stream socket,
         * letting the system select an appropriate protocol
         */

    if ((Sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        perror("creating Internet domain stream socket");
        exit(2);
    }

        /* Assign name to socket, letting system pick port */

    server.sin_family = AF_INET;
    server.sin_addr.s_addr = INADDR_ANY;
    server.sin_port = 0;
    if (bind(Sock, &server, sizeof(server))) {
        perror("binding Internet stream socket");
        close(Sock);
        exit(2);
    }

        /* Get assigned port number */

    length = sizeof(server);
    if (getsockname(Sock, &server, &length)) {
        perror("getting socket name");
        close(Sock);
        exit(2);
    }
    printf("Socket has port #%d\n", ntohs(server.sin_port));

        /* Handle writing to a non-existent client */

    signal(SIGPIPE, sigpipe);
```

```
        /* Terminate server on a SIGQUIT signal */

    signal(SIGQUIT, sigquit);

        /* Continue to listen for connections */

    for (;;) {
        if (listen(Sock, 5)) {
            perror("listen");
            close(Sock);
            exit(2);
        }
        if ((Msgsock = accept(Sock, 0, 0)) < 0) {
            perror("accept");
            close(Sock);
            exit(2);
        }
        do {
            if (write(Msgsock, Buf, strlen(Buf)) != strlen(Buf))
                perror("writing stream message");
            sleep(2);
        } while (!Sig);
        close(Msgsock);
        Msgsock = 0;
        Sig = 0;
    }
}

/* We'll receive a SIGPIPE signal if the client is killed.
 * Handle it so we can accept future connections from new clients.
 */
sigpipe (s, code, scp)
int s, code;
struct sigcontext *scp;
{
    printf("received SIGPIPE\n");
    Sig++;
}

sigquit (s, code, scp)
int s, code;
struct sigcontext *scp;
{
    if (shutdown(Sock, 2))
        perror("shutdown Sock");
    close(Sock);
    if (Msgsock) {
        if (shutdown(Msgsock, 2))
            perror("shutdown Msgsock");
        close(Msgsock);
    }
    exit(0);
}
```

**Asynchronous I/O and Internet Domain Stream Sockets**

The client creates an Internet domain stream socket and the system selects `TCP` as an appropriate protocol. The client then initiates a connection by supplying the destination host name and port number obtained by the server. The client then sets up the process group associated with the socket to receive asynchronous notification of input on the socket. Upon receipt of a `SIGIO` signal, the client prints the server message. Finally, the client is terminated by a `SIGQUIT` signal, at which time the full-duplex connection on the socket is shut down and the socket descriptor closed.

**More Example Programs**

See the following pages for additional example programs.

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <signal.h>
#include <sys/ioctl.h>
#include <fcntl.h>
#include <netdb.h>

#define BUFSIZE 512

int Sock;
int Sigio = 0;
int nSigio = 0;

/* The Client */
main (argc, argv)
int argc;
char **argv;
{
    struct sockaddr_in server;
    struct hostent *hp, *gethostbyname();
    int sigio(), sigquit();
    int n, val, pid;
    char buf[BUFSIZE];

    if (argc != 3)
    {
        printf("Usage: %s hostname portnumber\n", argv[0]);
        exit(2);
    }
        /* Create Internet domain stream socket,
         * letting the system select an appropriate protocol
         */

    Sock = socket(AF_INET, SOCK_STREAM, 0);
    if (Sock < 0) {
        perror("creating Internet domain stream socket");
        exit(2);
    }

        /* Connect socket using host, port specified on command line. */

    server.sin_family = AF_INET;
    hp = gethostbyname(argv[1]);
    bcopy(hp->h_addr, &(server.sin_addr.s_addr), hp->h_length);
    server.sin_port = htons(atoi(argv[2]));

    if (connect(Sock, &server, sizeof(server)) < 0 ) {
        close(Sock);
        perror("connecting stream socket");
        exit(2);
```

```
    };

        /* Set process group to receive SIGIO signals
         * when there is input on socket Sock
         */

    val = 1;
    if (ioctl(Sock, FIOASYNC, (char *)&val) < 0)
        perror("ioctl: FIOASYNC");

    pid = -getpid();
    if (ioctl(Sock, SIOCSPGRP, (char *)&pid) < 0)
        perror("ioctl: SIOCSPGRP");

    signal(SIGIO, sigio);
    signal(SIGQUIT, sigquit);

    for (;;)
    {
        /* Wait for a SIGIO signal indicating input pending */
        /* (Actually client would be doing some useful work here
         *  rather than waiting)
         */
        sigpause(SIGIO);
        if (Sigio) {
            if ((n = recv(Sock, buf, BUFSIZE, 0)) > 0) {
                buf[n] = '\0';
                printf("%s", buf);
            }
            Sigio = 0;
        }
    }
}

sigio ()
{
    Sigio++;
    nSigio++;
}

sigquit ()
{
    printf("Quitting, %d SIGIOs\n", nSigio);
    if (shutdown(Sock, 2))
        perror("shutdown");
    close(Sock);
    exit(0);
}
```

An easy way to run this example is to execute the server and client in two separate windows. Start up the server first in one window and the server in another window. Supply the host name and port number printed by the server as

arguments to the client. To stop the client, type `control-\`, or kill it with `kill -3`.

Under the Internet domain stream socket abstraction, a process group can be set to send or receive out-of-band data on the same pair of connected stream socket descriptors as normal data. Out-of-band data transmission is limited under Release 3.2 to one character.

Both normal and out-of-band data are received via the same descriptor if the process is set up to receive asynchronous notification of I/O on a socket descriptor. `SIGIO` signals will be received on the OOB data as well as `SIGURG` signals.

In the following examples, the server sends the client both normal and out-of-band data. These programs can be executed in the same manner as the `SIGIO` example above.

```
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/signal.h>
#include <netinet/in.h>

int Sig = 0;

char Buf[] = "Eat some chocolate!\n";
int Sock = 0;
int Msgsock = 0;

/* The Server */
main()
{
    int length;
    char oobmsg;
    struct sockaddr_in server;
    int sigpipe(), sigquit();

        /* Create Internet domain stream socket,
         * letting the system select an appropriate protocol
         */

    if ((Sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        perror("creating Internet domain stream socket");
        exit(2);
    }

        /* Assign name to socket, letting system pick port */

    server.sin_family = AF_INET;
    server.sin_addr.s_addr = INADDR_ANY;
    server.sin_port = 0;
    if (bind(Sock, &server, sizeof(server))) {
        perror("binding Internet stream socket");
        close(Sock);
        exit(2);
    }

        /* Get assigned port number */

    length = sizeof(server);
    if (getsockname(Sock, &server, &length)) {
        perror("getting socket name");
        close(Sock);
        exit(2);
    }
    printf("Socket has port #%d\n", ntohs(server.sin_port));

        /* Handle writing to a non-existent client */
```

```
    signal(SIGPIPE, sigpipe);

        /* Terminate server on a SIGQUIT signal */

    signal(SIGQUIT, sigquit);

        /* Continue to listen for connections */

    for (;;) {
        int i = 0;

        if (listen(Sock, 5)) {
            perror("listen");
            close(Sock);
            exit(2);
        }
        if ((Msgsock = accept(Sock, 0, 0)) < 0) {
            perror("accept");
            close(Sock);
            exit(2);
        }
        do {
                /* write normal data to client */

            if (write(Msgsock, Buf, strlen(Buf)) != strlen(Buf))
                perror("writing stream message");
            sleep(3);

                /* send Out-of-Band data to client */

            oobmsg = ('a' + i) &0177;
            printf("sending OOBMSG %c\n", oobmsg);
            if ((send(Msgsock, &oobmsg, 1, MSG_OOB)) < 0)
                perror("sending OOB data");
            i++;
            sleep(3);
        } while (!Sig);
        close(Msgsock);
        Msgsock = 0;
        Sig = 0;
    }
}

/* We'll receive a SIGPIPE signal if the client is killed.
 * Handle it so we can accept future connections from new clients.
 */
sigpipe (s, code, scp)
int s, code;
struct sigcontext *scp;
{
    printf("received SIGPIPE\n");
    Sig++;
}
```

```
sigquit (s, code, scp)
int s, code;
struct sigcontext *scp;
{
    if (shutdown(Sock, 2))
        perror("shutdown Sock");
    close(Sock);
    if (Msgsock) {
        if (shutdown(Msgsock, 2))
            perror("shutdown Msgsock");
        close(Msgsock);
    }
    exit(0);
}

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <signal.h>
#include <sys/ioctl.h>
#include <fcntl.h>
#include <netdb.h>

#define BUFSIZE 512

int Sock;
int Sigurg = 0;
int nSigurg = 0;

/* The Client */
main (argc, argv)
int argc;
char **argv;
{
    struct sockaddr_in server;
    struct hostent *hp, *gethostbyname();
    int oobdata(), sigquit();
    int n, pid;
    char buf[BUFSIZE];

    if (argc != 3)
    {
        printf("Usage: %s hostname portnumber\n", argv[0]);
        exit(2);
    }
        /* Create Internet domain stream socket,
         * letting the system select an appropriate protocol
         */

    Sock = socket(AF_INET, SOCK_STREAM, 0);
    if (Sock < 0) {
        perror("creating Internet domain stream socket");
        exit(2);
```

```
    }

        /* Connect socket using host, port specified on command line. */

    server.sin_family = AF_INET;
    hp = gethostbyname(argv[1]);
    bcopy(hp->h_addr, &(server.sin_addr.s_addr), hp->h_length);
    server.sin_port = htons(atoi(argv[2]));

    if (connect(Sock, &server, sizeof(server)) < 0 ) {
        close(Sock);
        perror("connecting stream socket");
        exit(2);
    };

        /* Set process group to receive SIGURG signals
         * when there is Out-of-Band data on socket Sock
         */

    pid = -getpid();
    if (ioctl(Sock, SIOCSPGRP, (char *)&pid) < 0)
        perror("ioctl: SIOCSPGRP");

    signal(SIGURG, oobdata);
    signal(SIGQUIT, sigquit);

    for (;;)
    {
        char mark;

        /* read normal data */
        if ((n = recv(Sock, buf, BUFSIZE, 0)) > 0) {
            buf[n] = '\0';
            printf("%s", buf);
        }
        if (Sigurg) {
            /* receive Out-of-Band data */
            recv(Sock, &mark, 1, MSG_OOB);
            printf("urgent message %c\n", mark);
            Sigurg = 0;
        }
    }
}

oobdata ()
{
    Sigurg++;
    nSigurg++;
}

sigquit ()
{
    printf("Quitting, %d sigurgs\n", nSigurg);
```

```
    if (shutdown(Sock, 2))
        perror("shutdown");
    close(Sock);
    exit(0);
}
```

See also the following topics found in the *UNIX Interface Reference Manual, Inter-Process Communication Primer*, and *Tutorial Examples of Interprocess Communication in Berkeley UNIX 4.2 BSD* by Stuart Sechrest, Computer Science Research Group, Computer Science Division, Department of Electrical Engineering and Computer Science, University of California, Berkeley.

*accept(2)*
*bind(2)*
*close(2)*
*connect(2)*
*fcntl(2)*
*intro(2)*
*ioctl(2)*
*listen(2)*
*recv(2)*
*shutdown(2)*
*socket(2)*
*write(2)*
*signal(3)*
*intro(4)*
*inet(4F)*
*tcp(4P)*
*fcntl(5)*
*protocols(5)*

# Color Maps

## Sun Color Applications

In this in-depth article, we discuss the use of color on the Sun Workstation.

- Overview to Color
- Hardware Frame Buffers
- Using `suntools`
- SunView
- Sun CGI
- SunCore
- Prism System - Sun Model 3/110

## An Overview to Color

The Sun windows and graphics packages support the use of colors. The two Sun windows packages are `suntools` and SunView. Two Sun graphics packages are SunCGI and SunCore. The graphics packages may use colors whether they are running inside or outside the window system.

Customer applications may each create their own colormap, or share a colormap created by a previous application. The colormap is created by defining red, green, and blue color arrays. A user sets the intensity of the color in each array as desired. The colormap index is defined as the index into the red, green, and blue color arrays. For example, colormap color index 3 displays the color defined by the combined contents of red[3], green[3], and blue[3]. The user application changes the colormaps on the screen, by changing the contents of the red, green, and blue color arrays; *not* by changing the index.

A total of 256 ( $2^8$ ) colors may be chosen from a palette containing 16 million ( $256^3$ ) colors. This limitation is imposed by the color frame buffer hardware. The memory allocated for the color frame buffer has as many colormaps loaded as possible, not to exceed the 256 total colors restriction. If the user applications that are running exceed 256 colors, some colormaps will be swapped out. The system determines which colormaps are loaded, by the user placement of the mouse.

If you run a combination of applications whose colormaps cannot all be held in the frame buffer at the same time, the colormap for the window containing the mouse is placed in the frame buffer so that it fits into a contiguous number of positions. The other windows on your screen may then be displayed in spurious colors, or may be black.

## Hardware Color Frame Buffers

The color frame buffers that Sun supports are described below.

`/dev/cgone0`
: for Sun 2 color, multibus systems

`/dev/cgtwo0`
: for Sun 2 systems with the VME bus, and Sun 3 systems

`/dev/cgfour0`
: the Prism frame buffer for Sun model 3/110 systems

`/dev/gpone`
: the graphics processor, used with `/dev/cgtwo0` on the Sun model 3/160C and 3/260C systems when this optional hardware board is installed.

## Viewing Surfaces

The Sun graphics packages distinguish between the inside and outside of the window system. While running under `suntools`, the graphics packages use different window devices as the view surfaces.

The color view surfaces for the window system are the 'color graphics pixwins,' called 'cgpixwindds.' The graphics packages while running inside the window system use 'cgpixwindds' or 'gppixwindds.' While running outside the window system, the graphics packages use the raw frame buffer. These devices are known as `cg1dd`, `cg2dd`, `cg4dd`, and `gp1dd`. These devices are described below.

`CG1DD`
: the Sun 1 color frame buffer device; and for Sun 2 multibus systems when running outside `suntools`, in 'console mode'

`CG2DD`
: the Sun 2 and Sun 3 color frame buffer device when running outside `suntools`, in 'console mode'

`CG4DD`
: the Sun 3/110 color frame buffer device when running outside `suntools`, in 'console mode'

`CGPIXWINDD`
: a color window, when running the application in a `suntools` window; or in a SunView canvas subwindow

`GP1DD`
: the graphics processor when running the application outside `suntools`, in 'console mode' (SunCore only)

`GP1PIXWINDD`
: the graphics processor when running the application in a `suntools` window, or in a SunView canvas subwindow (SunCore only)

See the *SunCore Reference Manual*, part number 800-1257; and the *SunCGI Reference Manual*, part number 800-1256, for further information. The 'cg'

devices run on the Sun 2 and 3 frame buffers. The 'gp' devices run on the graphics processor in conjunction with the color frame buffer.

The graphics processor is Sun's hardware graphics accelerator. This is a single-board option. An additional graphics buffer board option may also be added at a later time.

**Sharing Colormaps**

The Unix kernel manages the window system and its colormaps. It gives a name to and then stores a colormap when defined. Any particular colormap may be shared among applications by using the same colormap name. This is true across Sun products such as SunView, SunCore, and SunCGI.

Applications may share colormaps by calling `pw_setcmsname()` with the arguments being the `pixwin` for that window, the colormap name already defined, followed by `pw_putcolormap()`. `pw_putcolormap()` sets the colormap size and points to the colormap's red, green, and blue arrays. When sharing colormaps, the application which defines the colormap must be running before the windows for other applications may attach to that colormap.

The colors of a window may be changed 'on the fly' by modifying the red, green, and blue color arrays; and then by calling `pw_putcolormap()` to recolor the window. The `pw_putcolormap()` call changes the values in the colormap table. The CRT then redisplays the same pixel values. The color index is the same; however, a different color is projected on the screen. The pixels on the screen contain the index number into the colormap. The color which the index represents may be changed by modifying what the colormap represents. This is done by changing the contents of the red, green, and blue arrays. It is faster to change the colormap than to redisplay the screen.

`suntools`

The sun windowing system which is invoked by calling the command `suntools` with the `-f` and `-b` options, sets the default frame buffer colors. These colors are inherited by tools and other applications running inside the window system unless otherwise specified. Without these options, all applications receive the default colormap name 'monochrome' which defines white (red = 255, green = 255, blue = 255) as the background color, and black (red = 0, green = 0, blue = 0) as the foreground color.

The user may invoke tools such as `shelltool`, `textedit`, `cmdtool`, and `gfxtool`, with different foreground and background colors by specifying the `-Wf` and `-Wb` options. For example, the following `shelltool` is displayed with a red border and blue namestripe. The inside of the tool remains black and white.

```
shelltool -Wf 0 0 255 -Wb 255 0 0
```

Following the `-Wf` and `-Wb` options are the red, green, and blue color intensities. For example, the following tool has the foreground color purple and a light blue background, for not only the frame border and namestripe, but also for the window inside the tool. This is accomplished by adding the `-Wg` option as the example below shows.

```
shelltool -Wf 185 000 184 -Wb 102 250 247 -Wg
```

SunView

SunView applications may define their own colormaps or share colormaps up to the maximum 256 frame buffer colors. First, define the colormap size for each application. Second, set up the red, green, and blue arrays. Color number 0 is the color defined by red[0], green[0], and blue[0]; color number 1 is the color defined by red[1], green[1], and blue[1]; and so forth. The size of each colormap must be a power of 2, i.e. {2,4,8,16,32,64,128, or 256}.

A minimum value of 0 is used for no intensity of that color. A maximum value of 255 is used for full intensity of that color. For example, red[15] = 200, sets the 16th element of the red array with a very strong red. Blue[10] = 15, sets the 11th element of the blue array with a light intensity of blue. The intensity determines how intensely the monitor Cathode Ray Tube (CRT) displays the color.

For a given `pixwin`, use `pw_setcmsname()` to set the colormap name. The arguments are the `pixwin` for the window, and a character string. To bind the colormap to the `pixwin`, use `pw_putcolormap()`. The arguments are the `pixwin` for the window; the starting entry into the colormap; the number of colors; and the red, green, and blue color arrays. Again, the number of colors must be a power of 2.

A SunView example follows.

```
/*
 *  SunView color example
 *  Draws color lines in a canvas
 */

#include <suntool/sunview.h>
#include <suntool/canvas.h>

#define  NCOLORS        4

Frame                   frame;
Canvas                  canvas;
Pixwin                  *pw;

main()
{

    int             i;

    u_char  red[NCOLORS], green[NCOLORS], blue[NCOLORS];

    frame = window_create( NULL, FRAME,
    0);
    canvas = window_create( frame, CANVAS,
    0);

    /*
     *      Set up the red[], green[], and blue[] arrays.
     */

    red[0] = 255;    green[0] = 0;      blue[0] = 0;   /*red  */
    red[1] = 0;      green[1] = 255;    blue[1] = 0;   /*green*/
    red[2] = 0;      green[2] = 0;      blue[2] = 255; /*blue */
    red[3] = 208;    green[3] = 173;    blue[3] = 203; /*pink */

    /*
     *      Get the canvas pixwin, initialize the colormap,
     *      and put the colormap into the canvas window.
     */

    pw = canvas_pixwin( canvas);
    pw_setcmsname( pw, "Four Colors"); /*kernel now has this name*/
    pw_putcolormap( pw, 0, NCOLORS, red, green, blue);

    /*
     *      Draw lines in the canvas.
     */

    for (i=1; i < NCOLORS; i++) {
```

```
    /*      The PIX_SRC|PIX_COLOR raster op adds the color index
     *      to the source pixel value for display.
     */

          pw_rop(pw, 10, i*20, 300, i*20, PIX_SRC|PIX_COLOR(i),0);
        }

    window_main_loop(frame);
}
```

SunCGI

SunCGI must define its own colormap by creating a new colormap or using a shared colormap whether the SunCGI application is running inside or outside the window system.

The SunCGI color intensity scheme is the same as SunView, ranging from 0 to 255.

In SunCGI, first set the `dd` element of the view surface structure to be the frame buffer type {`CG1DD, CG2DD, CG4DD, GP1DD, or CGPIXWINDD`}. In the window environment, the graphics processor is accessed through `CGPIXWINDD`. If the graphics processor is available, the `CGPIXWINDD` uses that device for transformation calculations. Within the view surface structure, set the `cmapsize` element to the colormap size, and the `cmapname` element to the string that names the colormap. When the view surface is opened, and the red, green, and blue color arrays are initialized; the colormap array of type `Ccentry` points to the red, green, and blue color arrays.

A SunCGI example follows.

```
/*
 *      A SunCGI "C" program
 *      Draws colored lines
 *      Run in a gfxtool
 */

#include <cgidefs.h>
#include <stdio.h>

#define NCOLORS 64
#define MIN     0
#define MAX     10000

static  Ccoor   vpll    = { MIN, MIN }; /* lower left corner */
static  Ccoor   vpur    = { MAX, MAX }; /* upper right corner */

main()
{
        int     name;           /* view surface name */
        Cvwsurf device;         /* view surface device */
        Ccoorlist line;         /* line coordinate list */
        Ccoor   points[2];      /* point list */
        int     i;              /* position counter */
        Ccentry clist;          /* color map list */
        u_char  red[NCOLORS];   /* red color map */
        u_char  green[NCOLORS]; /* green color map */
        u_char  blue[NCOLORS];  /* blue color map */

        /* start cgi */
        device.dd = CGPIXWINDD;         /* select output device */
        open_cgi();                     /* initilize cgi */
        open_vws(&name,&device);        /* open view surface */
        vdc_extent(&vpll,&vpur);        /* reset vdc space */

        /* set the line attributes */
        line_width_specification_mode(ABSOLUTE);
        line_width(1.0);

        /* set up the color map */
        red[0] = 255; green[0] = 000; blue[0] = 000;    /* red    */
        red[1] = 000; green[1] = 255; blue[1] = 000;    /* green  */
        red[2] = 000; green[2] = 000; blue[2] = 255;    /* blue   */
        red[3] = 255; green[3] = 255; blue[3] = 000;    /* yellow */
        red[4] = 255; green[4] = 000; blue[4] = 255;    /* purple */
        red[5] = 150; green[5] = 150; blue[5] = 150;    /* gray   */
        red[6] = 120; green[6] = 090; blue[6] = 000;    /* brown  */
        red[7] = 000; green[7] = 000; blue[7] = 000;    /* black  */

        clist.n = NCOLORS;
        clist.ra = red;
        clist.ga = green;
```

```
        clist.ba = blue;
        color_table(0,&clist);

        /* draw colored lines */
        line.n = 2;
        line.ptlist = points;

        for (i = 0; i < NCOLORS; i++)   {
                line_color(i);
                points[0].y = MIN;
                points[0].x = (i*1000);
                points[1].y = MAX;
                points[1].x = (i*1000);
                polyline(&line);
        }
        sleep(5);

        /* end cgi */
        close_vws(name);
        close_cgi();
}
```

SunCore

SunCore applications running on the console window, in a `gfxtool`, a `shelltool`, or in a SunView canvas window must define their own colormaps. SunCore color devices are `CG1DD`, `CG2DD`, `CG4DD`, `CGPIXWINDD`, `GP1DD`, and `GP1PIXWINDD`.

SunCore colors do not range from 0 to 255. Instead they range as 256 possible values between 0 and 0.99. Most users familiar with the SunView model may assign their colors as shown in the example below.

```
float red[256];

for (i=0; i<256; i++)  {
  red  [i] = (float)i *  ( (float)1 / (float)256 );
  }
```

In SunCore, the view surface `cmapsize` must be set to the size of the color table, and `cmapname` must be set to the colormap name. These must be set before the `initialize_view_surface()` call. After the viewport and window are set up, you may define the color indices for text, line, and fill operations using the `define_color_indices()` call.

A SunCore C-language program example follows.

```
/*
 *  SunCore example written in "C"
 *  Writes a string in color
 *  Run this in a gfxtool
 */
#include        <usercore.h>

#define NCOLORS  8

int              cgpixwindd();
struct  vwsurf   vwsurf = DEFAULT_VWSURF(cgpixwindd);

main()
{

    float   red[NCOLORS], green[NCOLORS], blue[NCOLORS];
    float   x, y;
    int     i;

    vwsurf.cmapsize = NCOLORS;
    strcpy (vwsurf.cmapname , "Colormap");

    red  [0] = 0.99;
    green[0] = 0.99;
    blue [0] = 0.99;

    for ( i=1; i < NCOLORS; i++) {
            red  [i] = (float)i * (1.0 / (float)NCOLORS);
            green[i] = (float)i * (1.0 / (float)NCOLORS);
            blue [i] = (float)i * (1.0 / (float)NCOLORS);
    }

    if (initialize_core(BASIC,NOINPUT,TWOD))
            exit(1);
    if (initialize_view_surface(&vwsurf,FALSE))
            exit(2);
    if (select_view_surface(&vwsurf))
            exit(3);
    set_viewport_2 (0.0,1.0,0.0,.75);
    set_window (-100.0,100.0,-100.0,100.0);

    /*
     *  SunCore - You pass NCOLORS-1 since SunCore wants to
     *  to know where the last value is, not the colormap size.
     */

    define_color_indices(&vwsurf,0,NCOLORS-1,red,green,blue);
    create_temporary_segment();

    x = -100.0;
    y = 90.0;
```

```
    for (i = 1; i < NCOLORS; i++ )  {

            /*
             *  Set line index = color index.
             */

            set_line_index( i);
            move_abs_2(x, y);
            y = y - 20.0;
            line_abs_2( x, y);
            x = x + 20.0;
    }

    sleep( 5);
    close_temporary_segment();
    deselect_view_surface(&vwsurf);
    terminate_core();
}
```

When writing SunCore programs in FORTRAN, the programmer must set up the view surface structure array elements as shown in the example that follows. The rest of the code is similar to the C-language example above; in setting up the red, green, and blue arrays; assigning colors with intensities from 0 to 0.99; and applying the colors to lines, text, and fill operations.

A SunCore FORTRAN program example follows.

```
c
c   SunCore program in FORTRAN
c   Draws two lines
c   Run in a gfxtool
c
        include '/usr/include/f77/usercore77.h'

        integer vsurf(VWSURFSIZE)
c
c       Initialization of view surface structure.
c
        character *20 screenname, windowname, cmapname
        integer cmapsize
        equivalence (vsurf(1),  screenname)
        equivalence (vsurf(6),  windowname)
        equivalence (vsurf(14), cmapsize)
        equivalence (vsurf(15), cmapname)

c
c       Declarations of all color devices.
c
        integer  cg1dd, cg2dd, cgpixwindd
        external cg1dd, cg2dd, cgpixwindd
c
c       Create color arrays.
c
        real red(4), green(4), blue(4)

        integer InitializeCore, InitializeVwsurf, SelectVwsurf
c
c
        data vsurf /VWSURFSIZE*0/

        vsurf(DDINDEX) = loc(cgpixwindd)
        if (InitializeCore(BASIC, NOINPUT, TWOD) .ne. 0) call exit(1)
c
c       Display current vsurf information.
c
        print *, 'initializecore:'
c
c       Initialize colormap.
c
        cmapsize = 4

        red(1)          = 0.0
        green(1)        = 0.5
        blue(1)         = 0.0

        red(2)          = 1.0
        green(2)        = 0.5
        blue(2)         = 0.0
```

```
      red(3)          = 0.0
      green(3)        = 1.0
      blue(3)         = 0.5

      red(4)          = 1.0
      green(4)        = 0.0
      blue(4)         = 0.5

      print *, 'red:   ',     red(1),     red(2),      red(3),     red(4)
      print *, 'green: ',     green(1),     green(2),      green(3),     green(4)
      print *, 'blue:  ',     blue(0),     blue(1),      blue(2),     blue(4)
c
c     Initialize view surface and window.
c
      if (InitializeVwsurf(vsurf, FALSE) .ne. 0) call exit(2)
      if (SelectVwsurf(vsurf) .ne. 0) call exit(3)
      call SetViewPort2(0.125, 0.875, 0.125, 0.75)
      call SetWindow(0.0, 10.0, 0.0, 1.0)
      print *, 'initialization done'
c
c     First line is drawn on screen after setting colors.
c
      call DefColorIndices(vsurf, 1, 4, red, green, blue)
      call CreateTempSeg()
      print *, 'temporary segment created'
      print *, ' '
      print *, 'You will not see the first line'
      print *, 'because its color is the same'
      print *, 'as the background color.'

      do 200 i = 1, cmapsize
        reali = i
        call SetLineIndex(i)
        call MoveAbs2(reali, 0.0)
        call LineAbs2(reali, 9.0)
        print *,'line', i,'created'
  200 continue

      call sleep(5)

c
c     Close down SunCore.
c
      call CloseTempSeg()
      call sleep(5)
      call DeselectVwsurf(vsurf)
      call TerminateCore()
      end
```

The model Sun 3/110 systems, known as 'Prism' systems, have frame buffer architecture that is unique among Sun workstations. Such systems have a 10-bit frame buffer that emulates both a 'monochrome' and a color frame buffer. This frame buffer is called `/dev/cgfour0`.

The architecture for the 10-bit deep frame buffer is shown below.

```
1 plane     - enable plane
1 plane     - overlay plane group, monochrome, (/dev/bwtwo0)
8 planes    - color plane group, color, (/dev/cgfour0)

where the enable plane bit at a pixel location is set to
0 = color, 1 = monochrome.  This is the plane group visible at
that pixel location.
```

The overlay plane group is the black and white plane, the other plane group for color. On the Sun model 3/110 as on all other Sun 2 and 3 color machines, a maximum of 256 colors are visible.

When `suntools` is invoked with no options, all monochrome tools or applications appear from the overlay plane. All color tools or applications appear from the color planes. Tools that are invoked with color options appear from the color plane, and the other tools appear from the overlay plane.

The most common usage of `suntools` on Sun model 3/110 systems is to then run different applications in the different planes. This allows the user to use the model 3/110 as if there were two monitors, like `adjacentscreens` allows. The user invokes `suntools` in the color planes first. Then, from a `shelltool`, invoke a separate `suntools` in the overlay plane. When `suntools` is invoked this way, a `shelltool` with no `-Wf` or `-Wb` options is running in the overlay or monochrome planes belonging to the `suntools` from which it was invoked. From both the color and overlay planes, `switcher` is run, so that you may toggle between the color and the overlay planes.

An example follows, taken from the *switcher(1)* man page.

```
>From the console.
   "suntools -8bit_color_only -toggle_enable"

>You are now in the color planes.  From a shelltool.
   "suntools -d /dev/bwtwo0 -toggle_enable -n &"
    <suntools has been started in the overlay plane>

   "switcher -d /dev/bwtwo0 -s i &"
    <a switcher icon should appear to allow you to switch to the overlay
     plane>

>Click on the switcher icon, you are now in the overlay plane.
>From a shelltool.
   "switcher -s o &"
    <a switcher icon should appear to allow you to switch to the color
     planes>
```

On Sun model 3/110 systems, the default frame buffer, `/dev/fb`, refers to the color portion of the frame buffer. Applications such as *screendump(1)* and *screenload(1)* access `/dev/fb` by default. Thus `screendump` is equivalent to `screendump -f /dev/fb`, which is equivalent to `screendump -f /dev/cgfour0`.

For more information on Sun model 3/110 systems, see the *Release 3.2 Manual*, part number 800-1364.

# 5

# QUESTIONS, ANSWERS, HINTS, AND TIPS

# 5

# QUESTIONS, ANSWERS, HINTS, AND TIPS

## Q&A, and Tip of the Month

### Hints & Tips #4

This is the fourth in a continuing series of this column which I have created for two purposes.[12] 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!

### Yellow Pages and Mail Aliases

This month we are going to look at the Yellow Pages and how they relate to mail aliases on your system, and then look at the rest of the `.cshrc` aliases I promised last month.

### Mail Aliasing

When you send electronic mail, the system uses a set of alias files to allow people to set up mailing lists and make sure that mail gets to the right person. The three places where aliases can be set up are listed below.

---

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

`user/.mailrc`
: This file is read by the `/usr/ucb/Mail` program, and allows a user to set up private aliases or mailing lists without administration intervention.

`/usr/lib/aliases`
: This file is read by `sendmail`, and can be used to set up mailing lists or aliases global to the entire machine.

`mail.aliases`
: This is a yellow pages map found in SunOS releases 3.0 and subsequent releases. This allows an administrator to set up aliases in one location . The aliases are then accessible to all machines within the domain.

Two processes need to be clearly understood: when these files are accessed, and in what order. When you send a piece of mail, the mail system processes it as described below.

Every address is checked against your list of aliases in your `.mailrc` file. If there is a match, the alias is replaced with the list of entries in the `.mailrc` file. One example is shown below.

```
alias friend chuq rx          [.mailrc file]

% mail friend
```

This becomes translated to the entry that follows.

```
% mail chuq rx
```

In this example, the aliasing is quite limited, simply a straightforward string substitution that is space delimited.

The message is then sent to `sendmail`, which inspects it and checks each mailing address in the order shown below.

```
local aliases
yellow page aliases
```

Note that a match on the local alias does *not* keep the system from trying to rematch on the global alias. If the local alias file changes '*rx*' to '*rx@suntoo*' then the global alias file will not change it again to '*rx@ray.*' However, if you alias something locally that matches something in the yellow page alias, it will get aliased again.

After this aliasing, the mail is sent to the machine that does local delivery. On the local machine, a check is made for a `.forward` file. If it exists, the current mail address is replaced with the address in the `.forward` file. `sendmail` then aliases the address again. This process applies for multiple mail addresses as well. The `.mailrc` is not checked after the first round, though. When both rounds of aliasing are completed, `sendmail` drops the mail in your mailbox and all is done.

Pitfalls to Avoid

There are a few points to remember to ensure proper mail processing. First, it is possible to set up degenerative alias loops using the `mail.alias` file. The most common problem is to set up your aliases using the 'bang' format (as in *machine!user*) instead of using the 'at' format (as in *user@machine*). The 'bang' format worked prior to Sun OS release 3.0. However, with the implementation the yellow pages map, mail then transfers between a client and mailhost until it fails due to too many transfers. 'Bang' addressing should be used only for machines that are outside the local network, and then only when using UUCP for communication.

Second, it is possible to set up forwarding loops with a `.forward` file that will cause mail to be destroyed. In general, you should avoid using the `.forward` file and put your aliases into the yellow pages map instead.

**Tip of the Month (TOM)**

This month completes the lengthy `.cshrc`. This file includes a lot of miscellaneous aliases, shorthand command names, and other things to make working with Unix a little easier, nicer, and hopefully give you some ideas on customizing your environment so it works best for you. If you have some favorite aliases you want added to the next generation of the monster `.cshrc`, mail them to *sun!stb-editor*.

The script appears on the following pages.

```
#! /bin/csh
# Monster cshrc -- everything you might ever want to do when you
# use 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 (needed for at)
if (! $?prompt) exit
if (! $?term) exit

# Note that echo is built in and therefore much faster than
# calling pwd as a normal program.  This does not work correctly
# across symlinks.
alias pwd 'echo $cwd'

# set up general variables
set history=99  # nice round number

# These aliases let you bring a job to the foreground simply by
# typing the job number.  Very convenient.
alias 1 %1
alias 2 %2
alias 3 %3
alias 4 %4
alias 5 %5
alias 6 %6
alias 7 %7
alias 8 %8
alias 9 %9

# Quick pushd/popd -- pushd should really be +,
# but that requires shift.
#   is shorthand for pushd $HOME, which tends to happen often.
alias - popd
alias = pushd
alias   pushd

# Back up the directory tree quickly.
alias .. "cd .."

# Quick sunview compiling.
alias ccsv "cc \!* -lsuntool -lsunwindow -lpixrect"
alias ccsvg "cc \!* -g -lsuntool -lsunwindow -lpixrect"

# lint alias.  Lets you look at the libc lint library and check
# a call's parameters. For example, 'check read.'
alias check "grep \!* /usr/lib/lint/llib-lc"

# Convenient shorthands.
alias pe printenv
alias h history
```

```
alias m more
alias clean 'rm *.o core a.out'
alias psa "ps axu | sort -f +0 +1n | more"

# Safety hatches.
alias cp cp -i
alias mv mv -i
alias rm rm -i

# fg brings job into foreground, bg brings job into background,
# and v restarts a stopped vi.
# j lists jobs
# k kills jobs; 'k 1' kills job 1, 'k' kills the most recent job,
# shown with a '+' in the jobs list.
alias v %vi
alias j jobs -l
alias k 'kill %\!*'

# History editor.
# Dump the history into a file, edit it, and then source it back.
# Useful if you have a long, tedious command you do not want
# to retype.
set $hed /tmp/hed$$
alias hed history -h \!* > $hed; vi + $hed; source -h $hed
```

# 6

# THE HACKERS' CORNER

# 6

# THE HACKERS' CORNER

## Devices Present

**The Hackers' Corner: Determining Devices Present**

There are times when it might be helpful to know what devices are present and connected to your system. Again, you might not want to have to intelligently parse through a file to reinvent the wheel.

This article contains two programs, written by different programmers. Each approaches the effort to determine a machine's configuration differently. Use these programs carefully since investigating what the kernel knows is more of an arcane art than some kind of supported interface. UNIX systems export few, if any, internal kernel interfaces or data structures.

Professional Interest

The script or code contained in this article may be of interest to professinals, enthusiasts, or anyone having the time to key the script or code onto their system. If you email the STB editor a request for the script or code at *sun!stb-editor*, we will mail you an online copy. Please include the article name with your request.

Also, please consult your local shell script or programming expert regarding any script or code problems. The script or code is not offered as a supported Sun product, but as an item of interest to enthusiasts wanting to try out something for themselves.

Program 0

This is the first of two programs that determine what devices are present on your system. This program appears on the following pages.

```
#ifndef lint
static  char *sccsid = "%Z%%M% %I% %E% SMI";
#endif

/*
 * Print system hardware configuration
 */

#include <stdio.h>
#include <sys/param.h>
#include <sys/fcntl.h>
#include <nlist.h>

#include <sys/buf.h>
#include <sundev/mbvar.h>
#include <sun/autoconf.h>
#include <machine/mmu.h>
#include <machine/cpu.h>

static char *kmemf = "/dev/kmem";
static char *nlistf = "/vmunix";
static int  kvm_des;

static struct nlist nl[] = {
#define X_MBDINIT   0
    { "_mbdinit" },
#define X_CPUTYPE   1
    { "_cpu" },
#define X_PHYSMEM   2
    { "_physmem" },
    { "" }
};

static int  allflg;
static void usage();
static void printconf();
static int  kvmread();

extern long lseek();

int
main(argc, argv)
    int argc;
    char **argv;
{
    register char *argp;

    argc--, argv++;
    while (argc > 0 && **argv == '-') {
        argp = *argv++;
        argp++;
```

```
        argc--;
        while (*argp++)
        switch (argp[-1]) {

        case 'a':
            allflg++;
            break;

        default:
            usage();
            exit(1);
        }
    }
    if (argc > 1) {
        nlistf = argv[1];
        argv++;
        argc--;
    }
    if (argc > 1) {
        kmemf = argv[1];
        argv++;
        argc--;
    }
    if ((kvm_des = open(kmemf, O_RDONLY)) < 0) {
        (void) fprintf(stderr, "showconfig: Can't open ");
        perror(kmemf);
        exit(1);
    }
    if (nlist(nlistf, nl) < 0) {
        (void) fprintf(stderr, \
            "showconfig: Can't get at kernel namelist\n");
            /* XXX - need better error message */
        exit(1);
    }
    printconf();
    return (0);
}

static void
usage()
{

    (void) fprintf(stderr, "usage: showconfig -a [system] [core]\n");
    exit(1);
}

static void
printconf()
{
    unsigned long mbdptr;       /* address of mb_device tbl */
    unsigned long drvaddr = 0;  /* address of mb_driver */
    unsigned long ctlraddr = 0; /* address of mb_ctlr */
    struct mb_driver driver;
```

```
    struct mb_ctlr ctlr;
    struct mb_device device;
    int cpu;
    int physmem;
    char dname[12];
    char cname[12];
    char *space;
    caddr_t addr;
    int intpri;
    unsigned long intvec;
    struct vec vecs[2];
    register int i;
    register char *c;

    if (kvmread(kvm_des, (unsigned long) nl[X_CPUTYPE].n_value,
        (char *)&cpu, sizeof cpu) < 0) {
        perror("showconfig: Can't read cpu type");
        exit(1);
    }
    /*      12345678901234567890123456789012345678901234567890 */
    switch (cpu) {

#if defined(SUN2_ARCH)
    case CPU_SUN2_120:
        (void) printf("Multibus Sun-2\n");
        break;

    case CPU_SUN2_50:
        (void) printf("VMEbus Sun-2 or Sun-2/50\n");
        break;

    default:
        (void) printf("Sun-2, unknown type %#4.4x\n", cpu);
        break;
#endif
#if defined(SUN3_ARCH)
    case CPU_SUN3_160:
        (void) printf("Sun-3/75, Sun-3/160, or Sun-3/180\n");
        break;

    case CPU_SUN3_50:
        (void) printf("Sun-3/50 or Sun-3/52\n");
        break;

    case CPU_SUN3_260:
        (void) printf("Sun-3/260 or Sun-3/280\n");
        break;

    case CPU_SUN3_110:
        (void) printf("Sun-3/110\n");
        break;

    default:
```

```
        (void) printf("Sun-3, unknown type %#4.4x\n", cpu);
        break;
#endif
    }

    if (kvmread(kvm_des, (unsigned long) nl[X_PHYSMEM].n_value,
        (char *)&physmem, sizeof physmem) < 0) {
        perror("showconfig: Can't read amount of physical memory");
        exit(1);
    }

    (void) printf("Physical memory = %dK\n", ctob(physmem)/1024);

    mbdptr = nl[X_MBDINIT].n_value;
    (void) printf(
    "    DEVICE      SPACE  HEX ADDRESS RANGE      CTRLR  SLV PRI   VEC\n");
    while (1) {          /* a 'for' would be a mess here */
        /*
         * get the next mb_device entry
         */
        if (kvmread(kvm_des, mbdptr, \
            (char *)&device, sizeof device) < 0) {
            perror("showconfig: Can't read device entry");
            exit(1);
        }
        if (device.md_driver == 0)       /* end of table */
            break;
        mbdptr += sizeof device;
        /*
         * get the mb_ctlr and mb_driver, if not current
         */
        if (drvaddr != (unsigned long) device.md_driver) {
            drvaddr = (unsigned long) device.md_driver;
            if (kvmread(kvm_des, drvaddr, (char *)&driver,
                sizeof driver) < 0) {
                perror("showconfig: Can't read driver entry");
                exit(1);
            }
            if (kvmread(kvm_des, (unsigned long)driver.mdr_dname,
                (char *)dname, sizeof dname) < 0) {
                perror("showconfig: Can't read device name");
                exit(1);
            }
            if (device.md_mc != 0) {
                if (kvmread(kvm_des, \
                (unsigned long)driver.mdr_cname,
                    (char *)cname, sizeof cname) < 0) {
                    perror("showconfig: \
                    Can't read controller name");
                    exit(1);
                }
            }
        }
```

```
    if (device.md_mc != 0 && ctlraddr != \
    (unsigned long)device.md_mc) {
        ctlraddr = (unsigned long)device.md_mc;
        if (kvmread(kvm_des, ctlraddr, (char *)&ctlr,
            sizeof ctlr) < 0) {
            perror("showconfig: \
            Can't read controller entry");
            exit(1);
        }
    }

    if (!allflg && !device.md_alive)
        continue;       /* unconfigured device */

    /*
     * Consistency checking
     */
    if (device.md_mc != 0) {
        if (device.md_ctlr == -1) {
            (void) printf(
"%s%-2d - Bad controller number:  md_ctlr: -1\n",
                dname, device.md_unit);
        }
        if (device.md_slave == -1) {
            (void) printf(
"%s%-2d - Bad slave number:  md_slave: -1\n",
                dname, device.md_unit);
        }
    }

    if (device.md_alive && (device.md_mc == 0)) {
        if (device.md_ctlr != -1) {
            (void) printf(
"%s%-2d - Controller number for unspecified controller:  md_ctlr: %d\n",
                dname, device.md_unit,
                device.md_ctlr);
        }
        if (device.md_slave != -1) {
            (void) printf(
"%s%-2d - Slave number for unspecified controller:  md_slave: %d\n",
                dname, device.md_unit,
                device.md_slave);
        }
    }

    if (device.md_mc != 0) {
        if (device.md_driver != ctlr.mc_driver) {
            (void) printf(
"%s%-2d - Driver pointer mismatch:  md_driver: %x  mc_driver: %x\n",
                dname, device.md_unit,
                device.md_driver, ctlr.mc_driver);
        }
        if (device.md_ctlr != ctlr.mc_ctlr) {
```

```
            (void) printf(
"%s%-2d - Controller number mismatch:  md_ctlr: %d  mc_ctlr: %d\n",
                dname, device.md_unit,
                device.md_ctlr, ctlr.mc_ctlr);
        }
    }

    if (device.md_mc != 0 && device.md_alive && !ctlr.mc_alive) {
        (void) printf(
"%s%-2d - Controller not marked alive:  %s%-2d\n",
            dname, device.md_unit,
            cname, device.md_ctlr);
    }

    /*
     * Figure out the address space in which the device is mapped
     * Also, get the interrupt priority and vector
     */
    if (device.md_mc != 0) {
        i = SP_BUSMASK & ctlr.mc_space;
        c = "mc_";
        addr = ctlr.mc_addr;
        intpri = ctlr.mc_intpri;
        intvec = (unsigned long)ctlr.mc_intr;
    } else {
        i = SP_BUSMASK & device.md_space;
        c = "md_";
        addr = device.md_addr;
        intpri = device.md_intpri;
        intvec = (unsigned long)device.md_intr;
    }

    /* Read the first two vectors in */
    if (intvec != 0) {
        if (kvmread(kvm_des, intvec, (char *)vecs,
            sizeof vecs) < 0) {
            perror("showconfig: \
            Can't read interrupt vectors");
            exit(1);
        }
    }

    /*
     * More consistency checking
     */
    if (intpri < 0 || intpri >= 7)
        (void) printf(
            "%s%-2d - Illegal priority:  %sintpri: %d\n",
            dname, device.md_unit, c, intpri);

    if (intvec != 0 && vecs[0].v_func == NULL)
        (void) printf(
            "%s%-2d - Null interrupt handler:  %sintr\n",
```

```
                dname, device.md_unit, c);

        switch (i) {
        case SP_VIRTUAL:
            space = "virtual";
            break;
        case SP_OBMEM:
            space = "obmem";
            break;
        case SP_OBIO:
            space = "obio";
            break;
        case SP_MBMEM:
            space = "mbmem";
            break;
        case SP_MBIO:
            space = "mbio";
            break;
        case SP_VME16D16:
#if defined(SUN2_ARCH)
            space = "vme16";
#else
            space = "vme16d16";
#endif
            break;
        case SP_VME24D16:
#if defined(SUN2_ARCH)
            space = "vme24";
#else
            space = "vme24d16";
#endif
            break;
        case SP_VME32D16:
            space = "vme32d16";
            break;
        case SP_VME16D32:
            space = "vme16d32";
            break;
        case SP_VME24D32:
            space = "vme24d32";
            break;
        case SP_VME32D32:
            space = "vme32d32";
            break;
        default:
            (void) printf(
                "%s%-2d - Unknown address space: %sspace: %d\n",
                dname, device.md_unit, c, i);
        case 0:
            space = "????";
        }
```

```
        /*
         * Now (finally) print out the information
         */
        (void) printf("%s",  (device.md_alive ? " " : "*"));
        (void) printf("%8.8s%-2d ", dname, device.md_unit);
        (void) printf("%8.8s  ", space);
        if (addr != 0) {
            (void) printf("%08x-%08x ",
                addr, (addr + driver.mdr_size - 1));
        } else
            (void) printf("    size:%#-6x ", driver.mdr_size);
        if (device.md_mc != 0)
            (void) printf("%8.8s%-2d %3d ",
                cname, device.md_ctlr, device.md_slave);
        else
            (void) printf("                ");

        if (intpri != 0)
            (void) printf("%3d ", intpri);
        else
            (void) printf("    ");

        if (intvec != 0 && vecs[0].v_func != NULL) {
            for (c = ""; ; ) {
                (void) printf("%s %#x", c, vecs[0].v_vec);
                if (vecs[1].v_func == NULL)
                    break;
                intvec + = sizeof (struct vec);
                if (kvmread(kvm_des, intvec, (char *)vecs,
                    sizeof vecs) < 0) {
                    perror("showconfig: \
                    Can't read interrupt vectors");
                    exit(1);
                }
                c = ",";
            }
        } else
            (void) printf("    ");

        (void) printf("\n");
    }
```

```
}
static int
kvmread(fd, addr, value_ptr, value_size)
    int fd;
    unsigned long addr;
    char *value_ptr;
    int value_size;
{
    if (lseek(fd, (long) addr, 0) == -1L
        || read(fd, value_ptr, value_size) != value_size)
        return (-1);
    return (0);

}
```

Program 1

A second program named `probe.c` looks into the kernel to determine what devices are present. It produces an output similar to that shown below.

```
astra% probe
si0 at obio 0x140000 pri 2
st0 at si0 slave 0
st0 at si0 slave 32
zs0 at obio 0x20000 pri 3
zs1 at obio 0x0 pri 3
le0 at obio 0x120000 pri 3
bwtwo0 at obmem 0x100000 pri 4
des0 at obio 0x1c0000 not attached
astra%
```

Again, use this program on an experimental basis. Consult your local experts to see what they have to say.... Good luck!

```
/*
 * lists all devices defined for a configuration
 * and indicates if a device has not been attached
 *
 * usage: probe [vmunix]
 *
 * mark opperman
 * sun europe
 * 20 aug 86
 */

#include <stdio.h>
#include <ctype.h>
#include <nlist.h>
#include <sys/param.h>
#include <sys/file.h>
#include <sys/types.h>
#include <sys/buf.h>
#include <sys/vmmac.h>
#include <sys/dkbad.h>
#include <machine/param.h>
#include <machine/pte.h>
#include <sun/dklabel.h>
#include <sun/dkio.h>
#include <sundev/mbvar.h>
#include <sundev/screg.h>
#include <sundev/sireg.h>
#include <sundev/scsi.h>

struct nlist nl[] = {
    { "_Sysmap", 0, 0, 0, 0 },
#define NL_SYSMAP   0
    { "_mbcinit", 0, 0, 0, 0 },
#define NL_MBCINIT  1
    { "_mbdinit", 0, 0, 0, 0 },
#define NL_MBDINIT  2
    { "_scdriver", 0, 0, 0, 0 },
#define NL_SCDRIVER 3
    { "_scsi_ntype", 0, 0, 0, 0 },
#define NL_SCSINTYPE    4
    { "_scsi_unit_subr", 0, 0, 0, 0 },
#define NL_SCSIUNITSUBR 5
    { "" },
#define NL_LAST     6
};

#define physaddr(addr) (addr - KERNELBASE)

#define MAX_SCSI_DEV_NAME_LENGTH 3

int mem;
```

```
struct pte *Sysmap;
struct mb_ctlr *mbcinit;
struct mb_device *mbdinit;
struct mb_driver *scdriver;
int scsi_ntype;
struct scsi_unit_subr *scsi_unit_subr;
char *vmunix = "/vmunix";

struct pte getpte();
extern char *malloc();
extern char *calloc();

main(argc, argv)
    int argc;
    char    **argv;
{
    if (argc > 2) {
        fprintf(stderr, "usage: %s [vmunix]\n", argv[0]);
        exit(1);
    }

    if (argc == 2)
        vmunix = argv[1];

    if ((mem = open("/dev/mem", O_RDONLY)) < 0) {
        perror("can't open /dev/mem");
        exit(1);
    }

    getkvars();

    process();
}


getkvars()
{
    register char *devname;
    register i;

    nlist(vmunix, nl);
    if (nl[NL_SYSMAP].n_type == 0 || nl[NL_MBCINIT].n_type == 0 ||
                        nl[NL_MBDINIT].n_type == 0 ) {
        fprintf(stderr, "no namelist\n");
        exit(1);
    }

    for (i=0; i<NL_LAST; i++) {
        if (i == NL_SCDRIVER)    /* need virtual address */
            continue;
        if (nl[i].n_value >= KERNELBASE)
            nl[i].n_value = physaddr(nl[i].n_value);
    }
```

```
    Sysmap = (struct pte *) nl[NL_SYSMAP].n_value;
    if (scdriver = (struct mb_driver *) nl[NL_SCDRIVER].n_value) {
        kread(mem, nl[NL_SCSINTYPE].n_value, &scsi_ntype,
                    sizeof(scsi_ntype));
        scsi_unit_subr = (struct scsi_unit_subr *)
            calloc(scsi_ntype, sizeof(struct scsi_unit_subr));
        kread(mem, nl[NL_SCSIUNITSUBR].n_value, scsi_unit_subr,
            scsi_ntype * sizeof(struct scsi_unit_subr));
        for (i=0; i<scsi_ntype; i++) {
            devname = malloc(MAX_SCSI_DEV_NAME_LENGTH);
            kread(mem, scsi_unit_subr[i].ss_devname, devname,
                        MAX_SCSI_DEV_NAME_LENGTH);
            scsi_unit_subr[i].ss_devname = devname;
        }
    }
}


process()
{
    struct mb_ctlr mb_ctlr;
    struct mb_device mb_device;
    u_int addr;
    int n;

    for (addr = nl[NL_MBCINIT].n_value, n=0; ; n++) {
        kread(mem, addr, (caddr_t) &mb_ctlr,
                    sizeof(struct mb_ctlr));
        if (!mb_ctlr.mc_driver)
            break;
        addr += sizeof(struct mb_ctlr);
    }
    /*
     * Allocate one more controller than really exists and
     * mark its driver as zero to indicate the end
     * of the controllers.
     */
    mbcinit = (struct mb_ctlr *) calloc(n+1, sizeof(struct mb_ctlr));
    kread(mem, nl[NL_MBCINIT].n_value, (caddr_t) mbcinit,
                    n * sizeof(struct mb_ctlr));
    mbcinit[n].mc_driver = (struct mb_driver *) 0;

    for (addr = nl[NL_MBDINIT].n_value, n=0; ;n++) {
        kread(mem, addr, (caddr_t) &mb_device,
                    sizeof(struct mb_device));
        if (!mb_device.md_driver)
            break;
        addr += sizeof(struct mb_device);
    }
    /*
     * Allocate one more device than really exists and
     * mark its driver as zero to indicate the end
     * of the devices.
```

```
     */
    mbdinit = (struct mb_device *) calloc(n+1, sizeof(struct mb_device));
    kread(mem, nl[NL_MBDINIT].n_value, (caddr_t) mbdinit,
                    n * sizeof(struct mb_device));
    mbdinit[n].md_driver = (struct mb_driver *) 0;

    init_ctlr_drivers();
    init_device_drivers();
    display();
}


/*
 * Emulate mapping in this process' address space.
 */
update_drivers(old, new)
register struct mb_driver *old;
register struct mb_driver *new;
{
    register struct mb_ctlr *mc;
    register struct mb_device *md;

    if (scdriver == old)
        scdriver = new;

    for (mc=mbcinit; mc->mc_driver; mc++)
        if (mc->mc_driver == old) {
            mc->mc_driver = new;
        }
    for (md=mbdinit; md->md_driver; md++)
        if (md->md_driver == old) {
            md->md_driver = new;
        }
}


/*
 * Read in structures referenced by the mb_ctlr struct
 * and change pointers.
 */
init_ctlr_drivers()
{
    struct mb_ctlr *mc;
    struct mb_driver *mdr;
    struct vec *vec;
    char buf[16];

    for (mc=mbcinit; mdr=mc->mc_driver; mc++) {
        if (mdr < (struct mb_driver *) KERNELBASE)
            goto intr;

        mdr = (struct mb_driver *) malloc(sizeof(struct mb_driver));
        kread(mem, mc->mc_driver, mdr, sizeof(struct mb_driver));
```

```
        update_drivers(mc->mc_driver, mdr);
        if (mdr->mdr_dname) {
            kread(mem, mdr->mdr_dname, buf, sizeof(buf));
            mdr->mdr_dname = malloc(strlen(buf)+1);
            strcpy(mdr->mdr_dname, buf);
        }
        if (mdr->mdr_cname) {
            kread(mem, mdr->mdr_cname, buf, sizeof(buf));
            mdr->mdr_cname = malloc(strlen(buf)+1);
            strcpy(mdr->mdr_cname, buf);
        }
intr:
        if (mc->mc_intr) {
            vec = (struct vec *) malloc(sizeof(struct vec));
            kread(mem, mc->mc_intr, vec, sizeof(struct vec));
            mc->mc_intr = vec;
        }
    }
}


/*
 * Read in structures referenced by the mb_device struct
 * and change pointers.
 */
init_device_drivers()
{
    struct mb_device *md;
    struct mb_driver *mdr;
    struct vec *vec;
    char buf[16];

    for (md=mbdinit; mdr=md->md_driver; md++) {
        if (mdr < (struct mb_driver *) KERNELBASE)
            goto intr;

        mdr = (struct mb_driver *) malloc(sizeof(struct mb_driver));
        kread(mem, md->md_driver, mdr, sizeof(struct mb_driver));
        update_drivers(md->md_driver, mdr);
        if (mdr->mdr_dname) {
            kread(mem, mdr->mdr_dname, buf, sizeof(buf));
            mdr->mdr_dname = malloc(strlen(buf)+1);
            strcpy(mdr->mdr_dname, buf);
        }
        if (mdr->mdr_cname) {
            kread(mem, mdr->mdr_cname, buf, sizeof(buf));
            mdr->mdr_cname = malloc(strlen(buf)+1);
            strcpy(mdr->mdr_cname, buf);
        }
intr:
        if (md->md_intr) {
            vec = (struct vec *) malloc(sizeof(struct vec));
            kread(mem, md->md_intr, vec, sizeof(struct vec));
```

```
            md->md_intr = vec;
        }
    }
}


/*
 * Display all controllers/devices alive or not on the system.
 * Uses info in mbcinit and mbdinit.
 */
display()
{
    register struct mb_ctlr *mc;
    register struct mb_device *md;
    register struct mb_driver *mdr;
    register char *name;

    for (mc=mbcinit; mdr=mc->mc_driver; mc++) {
        doprobe(mc->mc_addr, mc->mc_space, mdr->mdr_cname,
            mc->mc_ctlr, mc->mc_alive, mc->mc_intpri,
            mc->mc_intr);

        /*
         * Now look for devices attached to this controller
         * (even if it's not attached).
         */
        for (md=mbdinit; md->md_driver; md++) {
            if (md->md_driver != mdr ||
                md->md_driver == (struct mb_driver *) -1 ||
                md->md_ctlr != mc->mc_ctlr)
                continue;
            /*
             * SCSI devices kludge...
             */
            if (md->md_driver == scdriver) {
                md->md_driver->mdr_dname =
                scsi_unit_subr[TYPE(md->md_flags)].ss_devname;
            }
            printf("%s%d at %s%d slave %d ",
                mdr->mdr_dname, md->md_unit,
                mdr->mdr_cname, mc->mc_ctlr, md->md_slave);
            if (!md->md_alive)
                printf("not attached");
            putchar('\n');
            /*
             * -1 indicates that info has already been displayed
             * so not redisplayed below.
             */
            md->md_driver = (struct mb_driver *) -1;
        }
    }

    for (md=mbdinit; mdr=md->md_driver; md++) {
```

```
        if (mdr == (struct mb_driver *) -1)
            continue;

        doprobe(md->md_addr, md->md_space, mdr->mdr_dname,
            md->md_unit, md->md_alive, md->md_intpri,
            md->md_intr);
    }
}


doprobe(addr, space, name, num, alive, intpri, intr)
    caddr_t addr;
    u_int space;
    char *name;
    short num;
    short alive;
    int intpri;
    struct vec *intr;
{
    char *addrspace;
    struct pte pte;

    if (alive) {
        pte = getpte(addr);
        addr = (caddr_t) ((u_int) ptob(pte.pg_pfnum) |
                          ((u_int) addr & PGOFSET));
    }

#define SP_BUSMASK      0x0000FFFF      /* mask for bus type */
#define SP_VIRTUAL      0x00000001
#define SP_OBMEM        0x00000002
#define SP_OBIO         0x00000004
#define SP_VME16D16     0x00000100
#define SP_VME24D16     0x00000200
#define SP_VME32D16     0x00000400
#define SP_VME16D32     0x00001000
#define SP_VME24D32     0x00002000
#define SP_VME32D32     0x00004000

    switch (space & SP_BUSMASK) {
    case SP_VIRTUAL:
        addrspace = "virtual";
        break;
    case SP_OBMEM:
        addrspace = "obmem";
        break;
    case SP_OBIO:
        addrspace = "obio";
        break;
    case SP_VME16D16:
        addrspace = "vme16d16";
        if (alive)
            addr = (caddr_t) ((int) addr &  0xffff0000);
```

```
            break;
        case SP_VME24D16:
            addrspace = "vme24d16";
            if (alive)
                addr = (caddr_t) ((int) addr &  0xff000000);
            break;
        case SP_VME32D16:
            addrspace = "vme32d16";
            break;
        case SP_VME16D32:
            addrspace = "vme16d32";
            if (alive)
                addr = (caddr_t) ((int) addr &  0xffff0000);
            break;
        case SP_VME24D32:
            addrspace = "vme24d32";
            if (alive)
                addr = (caddr_t) ((int) addr &  0xff000000);
            break;
        case SP_VME32D32:
            addrspace = "vme32d32";
            break;
        default:
            addrspace = "unknown";
            break;
        }

        printf("%s%d at %s 0x%x ", name, num, addrspace, addr);

        if (alive) {
            if (intpri) {
                if (intr == (struct vec *) 0)
                    printf("pri %d ", intpri);
                else
                    printf("vec 0x%x ", intr->v_vec);
            }
        }
        else {
            printf("not attached");
        }
        putchar('\n');
    }


    kread(fd, off, into, size)
        int fd;
        long off;
        caddr_t into;
        u_int size;
    {
        if (off >= KERNELBASE)
            off = physaddr(off);
```

```
    lseek(fd, off, 0);
    if (read(fd, into, size) != size) {
        perror("kread: read failed");
        exit(1);
    }
}


struct pte
getpte(a)
u_int a;
{
    u_int v;
    struct pte pte;

    v = btop(physaddr(a));
    kread(mem, (long) (Sysmap + v), &pte, sizeof(struct pte));
    return(pte);
}
```

# 7

# CUMULATIVE INDEX: 1988

# 7

# CUMULATIVE INDEX: 1988

# Index

## T



## U



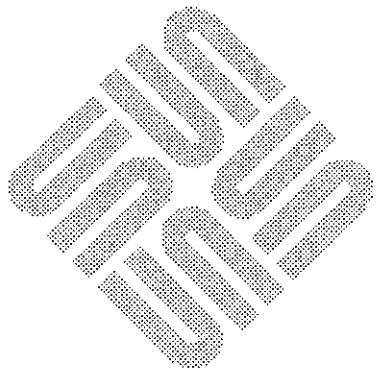
## W



## Y

# Revision History

| *Revision* | *Date* | *Comments* |
|---|---|---|
| **FINAL** | January 1988 | First issue of the 1988 Software Technical Bulletin, developed by Software Information Services (SIS), Customer Services Division (CSD). |



Bulk Rate
U.S. Postage
PAID
Permit No. 515
Mountain View, CA

**Corporate Headquarters**
Sun Microsystems, Inc.
2550 Garcia Avenue
Mountain View, CA 94043
415 960-1300
TLX 287815

**For U.S. Sales Office locations, call:**
800 821-4643
In CA: 800 821-4642

**European Headquarters**
Sun Microsystems Europe, Inc.
Sun House
31-41 Pembroke Broadway
Camberley
Surrey GU15 3XD
England
0276 62111
TLX 859017

**Australia:** 61-2-436-4699
**Canada:** 416 477-6745
**France:** (1) 46 30 23 24
**Germany:** (089) 95094-0
**Japan:** (03) 221-7021
**The Netherlands:** 02155 24888
**UK:** 0276 62111

**Europe, Middle East, and Africa, call European Headquarters:**
0276 62111

**Elsewhere in the world, call Corporate Headquarters:**
415 960-1300
Intercontinental Sales