sun® microsystems

Change Pages and Addenda to the Docubox

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

System Administration Addenda

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

Contents

Tables

Figures

System and Network Administration Addenda

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

□ Additions to SunOS 4.0.3 Affecting System Administration

□ Configuring the SunOS Kernel

□ Using the Automounter

□ The Sun Yellow Pages Service

□ Setting Up Electronic Mail

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

SunOS 4.0.3 Changes That Affect Administration

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

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

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

0.1. File System Changes in Release 4.0.3

4.0.3 Inclusion Instructions

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

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

The /usr/kvm Directory

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

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

The contents of /usr/kvm are described below.

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

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

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

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

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

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

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

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

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

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

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

0.2. Floppy Format for the Sun-3/80

4.0.3 Inclusion Instructions

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

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

0.3. Formatting your Floppy Disk

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

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

Mounting the Floppy Drive

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

Follow these steps to mount the floppy drive:

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

```
novel# cd /dev
```

3. Mount the device.

```
novel# MAKEDEV fd0
```

You can now format your disk.

`fdformat` Syntax

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

The basic syntax of `fdformat` is as follows:

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

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

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

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

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

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

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

Error Message

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

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

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

Example

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

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

0.4. Ejecting the Floppy Disk

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

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

`eject` Syntax

The syntax for `eject` is as follows:

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

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

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

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

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

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

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

0.5. Adding Clients to an Existing 4.0.3 Network

4.0.3 Inclusion Instructions

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

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

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

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

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

Adding a New Kernel Architecture

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

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

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

Procedures for Running setup_exec

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

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

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

setup_exec displays the SOFTWARE FORM also used by suninstall:

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



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








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

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

```
Path where executables reside :
```

type the full pathname for the executables of the *application* architecture of the client to be added. For example, type /export/exec/sun3 for sun3 and sun3x machines, or /export/exec/sun4 for sun4 and sun4c.

6. In response to the prompt

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

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

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

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

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

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

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

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

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

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

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

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

Adding and Removing Clients through setup_client

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

1. Ensure that the client is physically attached to the server's network, via Ethernet or similar media.

2. Become superuser on your NFS server.

3. Type the following:

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

to display the syntax for setup_client:

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

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

4. Specify setup_client using the syntax shown above. setup_client creates the directories necessary for the client to operate, updates /etc/exports on the server, and creates an /etc/fstab file for the client. It then prompts you when it is finished.

5. On a network running YP, you need to update the bootparams database on the YP server. If your network does not run YP, go on to Step 6.

6. Boot the client machine.

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

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

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

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

Adding a Client to a Heterogeneous Server

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

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

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

setup_client displays the following on your screen:

```
Start creating sun3x client "felafel" :

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

Creating root for client "felafel".

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

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

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

Completed creating sun3x client "felafel".
```

7. Become superuser on the network's YP server.

8. Edit the file `/etc/bootparams` and add the following for the new client:

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

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

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

10. Boot the client machine.

Removing a Client

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

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

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

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

Adding a Client to a Homogeneous Server

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

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

2. Become superuser on server dinner.

3. Add the client's Internet address to `/etc/hosts`.

4. Add the client's Ethernet address to `/etc/ethers`.

5. Type the following to set up the new client:

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

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

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

Converting a Standalone System to NFS Server

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

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

9

Reconfiguring the System Kernel

9

Reconfiguring the System Kernel

4.0.3 Inclusion Instructions

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

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

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

This chapter discusses the following subjects:

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

9.1. Why Reconfigure the Kernel?

There are two reasons to reconfigure the kernel:

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

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

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

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

9.2. Parts of the Kernel Configuration File

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

The Reconfiguration Process

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

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

Major Sections of the GENERIC Configuration File

The GENERIC configuration file is divided into three sections:

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

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

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

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

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

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

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

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

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

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

The System Identification Section

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

General System Description Lines

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

machine *type*

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

cpu *type*

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

Legal types for a Sun-2 machine are:

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

Legal types for a Sun-3 machine are:

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

Legal types for a Sun-3x are:

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

Legal types for a Sun-4 machine are:

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

Legal types for a Sun-4c are:

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

ident *name*

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

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

maxusers *number*

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

options *type*

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

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

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

System-Specific Description Lines

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

The line has the following syntax:

```
config kernelname config_clauses
```

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

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

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

This clause specifies the location of the root file system.

```
swap [on] swap_device
```

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

```
dumps [on] dump_device
```

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

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

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

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

```
config vmunix root xy0
```

For diskless clients:

Use the following `config_clause`

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

The Pseudo-Devices Section

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

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

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

The Connections Section

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

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

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

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

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

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

The Devices Section

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

Device Description Lines

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

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

dev_type dev_name `at` *connect_dev more info*

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

dev_type

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

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

dev_name

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

connect_dev

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

`virtual`	Virtual preset
`obmem`	On-board memory
`obio`	On-board I/O
`mbio`	Multibus I/0 (Sun-2 only)
`mbmem`	Multibus Memory (Sun-2 only)
`vme16d16 (vme16)`	VMEbus: 16 bit address/16 bit data (Sun-3 Sun-3x and Sun-4)
`vme24d16 (vme24)`	VMEbus: 24 bit address/16 bit data (Sun-3 Sun-3x and Sun-4)
`vme32d16`	VMEbus: 32 bit address/16 bit data (Sun-3 and Sun-4)
`vme16d32`	VMEbus: 16 bit address/32 bit data (Sun-3 Sun-3x and Sun-4)
`vme24d32`	VMEbus: 24 bit address/32 bit data (Sun-3 Sun-3x and Sun-4)
`vme32d32`	VMEbus: 32 bit address/32 bit data (Sun-3 Sun-3x and Sun-4)

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

more_info

This is a sequence of the following:

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

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

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

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

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

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

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

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

is attached to the Xylogics controller:

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

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

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

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

9.3. Modifying the Kernel Configuration Files

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

Note: Some parameters relating to the System V Inter-Process Communication (IPC) extensions may also be tuned in the configuration file. These parameters do not appear in the GENERIC file but are documented in Chapter 16 of the *System Administration and Networking* manual.

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

Modification Procedures

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

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

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

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

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

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

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

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

```
# cp GENERIC SYS_NAME
```

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

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

```
# chmod +w SYS_NAME
```

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

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

The Sun-2 GENERIC Kernel

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

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

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

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

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

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

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

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


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

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

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

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

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

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

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

The Sun-3 GENERIC Kernel

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

#
# Support for 4 Xylogics 7053 controllers with 4 drives each.
#
controller      xdc0 at vme16d32 ? csr 0xee80 priority 2 vector xdintr 0x44
controller      xdc1 at vme16d32 ? csr 0xee90 priority 2 vector xdintr 0x45
controller      xdc2 at vme16d32 ? csr 0xeea0 priority 2 vector xdintr 0x46
controller      xdc3 at vme16d32 ? csr 0xeeb0 priority 2 vector xdintr 0x47
disk        xd0 at xdc0 drive 0
disk        xd1 at xdc0 drive 1
disk        xd2 at xdc0 drive 2
disk        xd3 at xdc0 drive 3
disk        xd4 at xdc1 drive 0
disk        xd5 at xdc1 drive 1
disk        xd6 at xdc1 drive 2
disk        xd7 at xdc1 drive 3
disk        xd8 at xdc2 drive 0
disk        xd9 at xdc2 drive 1
disk        xd10 at xdc2 drive 2
disk        xd11 at xdc2 drive 3
disk        xd12 at xdc3 drive 0
disk        xd13 at xdc3 drive 1
disk        xd14 at xdc3 drive 2
disk        xd15 at xdc3 drive 3
#
# Support for 2 Xylogics 450/451 controllers with 2 drives each.
#
controller  xyc0 at vme16d16 ? csr 0xee40 priority 2 vector xyintr 0x48
controller  xyc1 at vme16d16 ? csr 0xee48 priority 2 vector xyintr 0x49
disk        xy0 at xyc0 drive 0
disk        xy1 at xyc0 drive 1
disk        xy2 at xyc1 drive 0
disk        xy3 at xyc1 drive 1
#
# Support for the SCSI-2 host adapter with 2 disks and 1 1/4" tape
# on the first SCSI controller, and 2 disks and 1 1/4" tape on the
# second SCSI controller.
#
controller  sc0 at vme24d16 ? csr 0x200000 priority 2 vector scintr 0x40
tape        st0 at sc0 drive 32 flags 1
tape        st1 at sc0 drive 40 flags 1
disk        sf0 at sc0 drive 49 flags 2
disk        sd0 at sc0 drive 0 flags 0
disk        sd1 at sc0 drive 1 flags 0
disk        sd2 at sc0 drive 8 flags 0
disk        sd3 at sc0 drive 9 flags 0
disk        sd4 at sc0 drive 16 flags 0
disk        sd6 at sc0 drive 24 flags 0
#
# Support for the SCSI-3 host adapter and the on-board SCSI controller
# on several machines (e.g. 3/50).  Same device support as above.
#
controller  si0 at vme24d16 ? csr 0x200000 priority 2 vector siintr 0x40
controller  si1 at vme24d16 ? csr 0x204000 priority 2 vector siintr 0x41
controller  si0 at obio ? csr 0x140000 priority 2
tape        st0 at si0 drive 32 flags 1
tape        st1 at si0 drive 40 flags 1
tape        st2 at si1 drive 32 flags 1
tape        st3 at si1 drive 40 flags 1
disk        sf0 at si0 drive 49 flags 2
disk        sd0 at si0 drive 0 flags 0
disk        sd1 at si0 drive 1 flags 0
disk        sd2 at si0 drive 8 flags 0
disk        sd3 at si0 drive 9 flags 0
disk        sd4 at si0 drive 16 flags 0
disk        sd6 at si0 drive 24 flags 0
#
# Support for the SCSI-E host adapter used with the Sun-3/E.
# Same device support as above.
#
controller      se0 at vme24d16 ? csr 0x300000 priority 2 vector se_intr 0x40
tape        st0 at se0 drive 32 flags 1
tape        st1 at se0 drive 40 flags 1
disk        sd0 at se0 drive 0 flags 0
disk        sd1 at se0 drive 1 flags 0
disk        sd2 at se0 drive 8 flags 0
disk        sd3 at se0 drive 9 flags 0
#disk       sd4 at se0 drive 16 flags 0
#disk       sd6 at se0 drive 24 flags 0
#
# Support for the 2 tty lines (ttya, ttyb) on the cpu board.
# Needed when using a terminal for the console device.
# Flags=3 says to supply carrier in software for both lines.
#
device      zs0 at obio ? csr 0x20000 flags 3 priority 3
#
# Support for the keyboard and mouse interface.  Needed when
# using a frame buffer as the console device or with SunView.
# You can remove this line if you don't use the standard Sun
# Workstation keyboard and mouse, but if you leave it in don't
# change it.
#
device      zs1 at obio ? csr 0x00000 flags 0x103 priority 3
#
# Support for 4 ALM's (Systech MTI-800/1600).  Flags set for
# all lines to be local, i.e., carrier supplied by software
# rather than by the device.
#
device      mti0 at vme16d16 ? csr 0x620 flags 0xffff priority 4
    vector mtiintr 0x88
device      mti1 at vme16d16 ? csr 0x640 flags 0xffff priority 4
    vector mtiintr 0x89
device      mti2 at vme16d16 ? csr 0x660 flags 0xffff priority 4
    vector mtiintr 0x8a
device      mti3 at vme16d16 ? csr 0x680 flags 0xffff priority 4
    vector mtiintr 0x8b
#
# Support for 8 MCP boards.
# Note that the first four MCP's use the same vectors as the ALM's and thus
# ALM's cut into the total number of MCP's that can installed.
# Make sure the maxusers line above has at least one added to it for
# each serial port.
#
device      mcp0 at vme32d32 ? csr 0x01000000 flags 0x1ffff priority 4
    vector mcpintr 0x8b
device      mcp1 at vme32d32 ? csr 0x01010000 flags 0x1ffff priority 4
    vector mcpintr 0x8a
```

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

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

#
# End of Network Interface Declarations
#

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

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

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

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

The Sun-3x GENERIC Kernel

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

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

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

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

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

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

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

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

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

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

```
#
# Support for 4 Xylogics 7053 controllers with 4 drives each.
#
controller      xdc0 at vme16d32 ? csr 0xee80 priority 2 vector xdintr 0x44
controller      xdc1 at vme16d32 ? csr 0xee90 priority 2 vector xdintr 0x45
controller      xdc2 at vme16d32 ? csr 0xeea0 priority 2 vector xdintr 0x46
controller      xdc3 at vme16d32 ? csr 0xeeb0 priority 2 vector xdintr 0x47
disk        xd0 at xdc0 drive 0
disk        xd1 at xdc0 drive 1
disk        xd2 at xdc0 drive 2
disk        xd3 at xdc0 drive 3
disk        xd4 at xdc1 drive 0
disk        xd5 at xdc1 drive 1
disk        xd6 at xdc1 drive 2
disk        xd7 at xdc1 drive 3
disk        xd8 at xdc2 drive 0
disk        xd9 at xdc2 drive 1
disk        xd10 at xdc2 drive 2
disk        xd11 at xdc2 drive 3
disk        xd12 at xdc3 drive 0
disk        xd13 at xdc3 drive 1
disk        xd14 at xdc3 drive 2
disk        xd15 at xdc3 drive 3
#
# Support for 2 Xylogics 450/451 controllers with 2 drives each.
#
controller  xyc0 at vme16d16 ? csr 0xee40 priority 2 vector xyintr 0x48
controller  xyc1 at vme16d16 ? csr 0xee48 priority 2 vector xyintr 0x49
disk        xy0 at xyc0 drive 0
disk        xy1 at xyc0 drive 1
disk        xy2 at xyc1 drive 0
disk        xy3 at xyc1 drive 1
#
# Support for the SCSI-2 host adapter with 2 disks and 1 1/4" tape
# on the first SCSI controller, and 2 disks and 1 1/4" tape on the
# second SCSI controller.
#
controller  sc0 at vme24d16 ? csr 0x200000 priority 2 vector scintr 0x40
tape        st0 at sc0 drive 32 flags 1
tape        st1 at sc0 drive 40 flags 1
#disk       sf0 at sc0 drive 49 flags 2
disk        sd0 at sc0 drive 0 flags 0
disk        sd1 at sc0 drive 1 flags 0
disk        sd2 at sc0 drive 8 flags 0
disk        sd3 at sc0 drive 9 flags 0
disk            sd4 at sc0 drive 16 flags 0
disk            sd6 at sc0 drive 24 flags 0
#
# Support for the SCSI-3 host adapter. Same device support as above.
#
controller  si0 at vme24d16 ? csr 0x200000 priority 2 vector siintr 0x40
controller      si1 at vme24d16 ? csr 0x204000 priority 2 vector siintr 0x41
tape        st0 at si0 drive 32 flags 1
tape        st1 at si0 drive 40 flags 1
tape            st2 at si1 drive 32 flags 1
tape            st3 at si1 drive 40 flags 1
#disk       sf0 at si0 drive 49 flags 2
disk        sd0 at si0 drive 0 flags 0
disk        sd1 at si0 drive 1 flags 0
disk        sd2 at si0 drive 8 flags 0
disk        sd3 at si0 drive 9 flags 0
disk            sd4 at si0 drive 16 flags 0
disk            sd6 at si0 drive 24 flags 0
#
# Support for the SCSI-ESP host adapter for 3/80
#
controller  sm0 at obio ? csr 0x66000000 priority 2
tape        st0 at sm0 drive 32 flags 1
tape        st1 at sm0 drive 40 flags 1
#disk       sf0 at sm0 drive 49 flags 2
disk        sd0 at sm0 drive 0 flags 0
disk        sd1 at sm0 drive 1 flags 0
disk        sd2 at sm0 drive 8 flags 0
disk        sd3 at sm0 drive 9 flags 0
disk            sd4 at sm0 drive 16 flags 0
disk            sd6 at sm0 drive 24 flags 0
#
# Support for the 2 tty lines (ttya, ttyb) on the cpu board.
# Needed when using a terminal for the console device.
# Flags=3 says to supply carrier in software for both lines.
#
device      zs0 at obio ? csr 0x62002000 flags 3 priority 3
#
# Support for the keyboard and mouse interface.  Needed when
# using a frame buffer as the console device or with SunView.
# You can remove this line if you don't use the standard Sun
# Workstation keyboard and mouse, but if you leave it in don't
# change it.
#
device      zs1 at obio ? csr 0x62000000 flags 0x103 priority 3
#
# Support for 4 ALM's (Systech MTI-800/1600).  Flags set for
# all lines to be local, i.e., carrier supplied by software
# rather than by the device.
#
device      mti0 at vme16d16 ? csr 0x620 flags 0xffff priority 4
    vector mtiintr 0x88
device      mti1 at vme16d16 ? csr 0x640 flags 0xffff priority 4
    vector mtiintr 0x89
device      mti2 at vme16d16 ? csr 0x660 flags 0xffff priority 4
    vector mtiintr 0x8a
device      mti3 at vme16d16 ? csr 0x680 flags 0xffff priority 4
    vector mtiintr 0x8b
#
# Support for 8 MCP boards.
# Note that the first four MCP's use the same vectors as the ALM's and thus
# ALM's cut into the total number of MCP's that can installed.
# Make sure the maxusers line above has at least one added to it for
# each serial port.
#
device      mcp0 at vme32d32 ? csr 0x01000000 flags 0x1ffff priority 4
    vector mcpintr 0x8b
device      mcp1 at vme32d32 ? csr 0x01010000 flags 0x1ffff priority 4
    vector mcpintr 0x8a
device      mcp2 at vme32d32 ? csr 0x01020000 flags 0x1ffff priority 4
    vector mcpintr 0x89
device      mcp3 at vme32d32 ? csr 0x01030000 flags 0x1ffff priority 4
    vector mcpintr 0x88
device          mcp4 at vme32d32 ? csr 0x01040000 flags 0x1ffff priority 4
        vector mcpintr 0xa0
device          mcp5 at vme32d32 ? csr 0x01050000 flags 0x1ffff priority 4
        vector mcpintr 0xa1
device          mcp6 at vme32d32 ? csr 0x01060000 flags 0x1ffff priority 4
        vector mcpintr 0xa2
device          mcp7 at vme32d32 ? csr 0x01070000 flags 0x1ffff priority 4
        vector mcpintr 0xa3
#
# Support for the on-board Intel 82586 Ethernet chip
#
device      ie0 at obio ? csr 0x65000000 priority 3
#
# Support for a second Intel Ethernet board, either a second
# Sun-3/E board or a Multibus Ethernet board used with a
# Multibus-to-VME adapter.  Used for Ethernet to Ethernet
# gateways.
#
device      ie1 at vme24d16 ? csr 0xe88000 priority 3 vector ieintr 0x75
#
#
# Support for the on-board LANCE Ethernet (AM7990).
#
device      le0 at obio ? csr 0x65002000 priority 3
# Support for 2 Ciprico TapeMaster tape controllers with 1 tape drive each.
#
controller  tm0 at vme16d16 ? csr 0xa0 priority 3 vector tmintr 0x60
controller  tm1 at vme16d16 ? csr 0xa2 priority 3 vector tmintr 0x61
tape        mt0 at tm0 drive 0 flags 1
tape        mt1 at tm1 drive 0 flags 1
#
# Support for 2 Xylogics 472 tape controllers with 1 tape drive each.
#
controller  xtc0 at vme16d16 ? csr 0xee60 priority 3 vector xtintr 0x64
controller  xtc1 at vme16d16 ? csr 0xee68 priority 3 vector xtintr 0x65
tape        xt0 at xtc0 drive 0 flags 1
tape        xt1 at xtc1 drive 0 flags 1
#
# Support for the GP/GP+/GP2 graphics processors.
# Requires cgtwo as well.
#
```

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

The Sun-3/80 GENERIC Kernel

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

The Sun-4 GENERIC Kernel

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

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

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

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

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

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

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

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

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

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

# connections for machine type 2 (SUN4_110)
# NOTE: does not support bus-master devices.
controller  obmem 2 at nexus ?
controller  obio 2 at nexus ?
controller  vme16d16 2 at nexus ?
controller  vme24d16 2 at nexus ?
controller  vme32d16 2 at nexus ?
controller  vme16d32 2 at nexus ?
controller  vme24d32 2 at nexus ?
controller  vme32d32 2 at nexus ?

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

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

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

```
# Support for the SCSI-2 host adapter with 2 disks and 1 1/4" tape
# on the first SCSI controller, and 2 disks and 1 1/4" tape on the
# second SCSI controller.
#
controller    sc0 at vme24d16 ? csr 0x200000 priority 2 vector scintr 0x40
tape          st0 at sc0 drive 32 flags 1
tape          st1 at sc0 drive 40 flags 1
#disk         sf0 at sc0 drive 49 flags 2
disk          sd0 at sc0 drive 0 flags 0
disk          sd1 at sc0 drive 1 flags 0
disk          sd2 at sc0 drive 8 flags 0
disk          sd3 at sc0 drive 9 flags 0
disk          sd4 at sc0 drive 16 flags 0
disk          sd6 at sc0 drive 24 flags 0
#
# Support for the SCSI-3 host adapter.  Same device support as above.
#
controller    si0 at vme24d16 ? csr 0x200000 priority 2 vector siintr 0x40
controller    si1 at vme24d16 ? csr 0x204000 priority 2 vector siintr 0x41
tape          st0 at si0 drive 32 flags 1
tape          st1 at si0 drive 40 flags 1
tape          st2 at si1 drive 32 flags 1
tape          st3 at si1 drive 40 flags 1
#disk         sf0 at si0 drive 49 flags 2
disk          sd0 at si0 drive 0 flags 0
disk          sd1 at si0 drive 1 flags 0
disk          sd2 at si0 drive 8 flags 0
disk          sd3 at si0 drive 9 flags 0
disk          sd4 at si0 drive 16 flags 0
disk          sd6 at si0 drive 24 flags 0
#
# Support for the "SCSI weird" host adapter used with the Sun-4/110.
# Same device support as above.
#
controller    sw0 at obio 2 csr 0xa000000 priority 2
tape          st0 at sw0 drive 32 flags 1
tape          st1 at sw0 drive 40 flags 1
#disk         sf0 at sw0 drive 49 flags 2
disk          sd0 at sw0 drive 0 flags 0
disk          sd1 at sw0 drive 1 flags 0
disk          sd2 at sw0 drive 8 flags 0
disk          sd3 at sw0 drive 9 flags 0
disk          sd4 at sw0 drive 16 flags 0
disk          sd6 at sw0 drive 24 flags 0
#
# Support for the SCSI-ESP host adapter.
#
controller    sm0 at obio ? csr 0xfa000000 priority 2
tape          st0 at sm0 drive 32 flags 1
tape          st1 at sm0 drive 40 flags 1
#disk         sf0 at sm0 drive 49 flags 2
disk          sd0 at sm0 drive 0 flags 0
disk          sd1 at sm0 drive 1 flags 0
disk        sd2 at sm0 drive 8 flags 0
disk        sd3 at sm0 drive 9 flags 0
disk        sd4 at sm0 drive 16 flags 0
disk        sd6 at sm0 drive 24 flags 0
#
# Support for the 2 tty lines (ttya, ttyb) on the cpu board.
# Needed when using a terminal for the console device.
# Flags=3 says to supply carrier in software for both lines.
#
device      zs0 at obio ? csr 0xf1000000 flags 3 priority 3
#
# Support for the keyboard and mouse interface.  Needed when
# using a frame buffer as the console device or with SunView.
# You can remove this line if you don't use the standard Sun
# Workstation keyboard and mouse, but if you leave it in don't
# change it.
#
device      zs1 at obio ? csr 0xf0000000 flags 0x103 priority 3
#
# Support for 2 additional tty lines on board the Sun-4/330.
#
device      zs2 at obio 3 csr 0xe0000000 flags 0x3 priority 3
#
# Support for 4 ALM's (Systech MTI-800/1600).  Flags set for
# all lines to be local, i.e., carrier supplied by software
# rather than by the device.
#
device      mti0 at vme16d16 1 csr 0x620 flags 0xffff priority 4
    vector mtiintr 0x88
device      mti1 at vme16d16 1 csr 0x640 flags 0xffff priority 4
    vector mtiintr 0x89
device      mti2 at vme16d16 1 csr 0x660 flags 0xffff priority 4
    vector mtiintr 0x8a
device      mti3 at vme16d16 1 csr 0x680 flags 0xffff priority 4
    vector mtiintr 0x8b
#
# Support for 8 MCP boards.
# Note that the first four MCP's use the same vectors as the ALM's and thus
# ALM's cut into the total number of MCP's that can installed.
# Make sure the maxusers line above has at least one added to it for
# each serial port.
#
device      mcp0 at vme32d32 ? csr 0x01000000 flags 0x1ffff priority 4
    vector mcpintr 0x8b
device      mcp1 at vme32d32 ? csr 0x01010000 flags 0x1ffff priority 4
    vector mcpintr 0x8a
device      mcp2 at vme32d32 ? csr 0x01020000 flags 0x1ffff priority 4
    vector mcpintr 0x89
device      mcp3 at vme32d32 ? csr 0x01030000 flags 0x1ffff priority 4
    vector mcpintr 0x88
device          mcp4 at vme32d32 ? csr 0x01040000 flags 0x1ffff priority 4
        vector mcpintr 0xa0
device          mcp5 at vme32d32 ? csr 0x01050000 flags 0x1ffff priority 4
```

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

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

9.4. Procedures for Reconfiguring the Kernel

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

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

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

Kernel Reconfiguration for Standalone Systems

For standalone machines, proceed as follows.

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

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

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

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

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

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

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

> b vmunix
```

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

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

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

Kernel Reconfiguration for Servers and Their Clients

For server machines, proceed as follows.

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

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

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

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

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

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

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

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

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

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

[ On the client machine :  ]
>b vmunix
```

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

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

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

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

On the server type:

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

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

9.5. Changing Swap Space

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

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

Procedures for Increasing Swap Space

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

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

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

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

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

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

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

3. Reboot client raks.

Setting Up a Second Swap Partition

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

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

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

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

```
swapon -a
```

13

Addendum: Using the Automounter

13

Addendum: Using the Automounter

4.0.3 Inclusion Instructions

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

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

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

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

How the automounter works

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

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

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

mount is in effect.

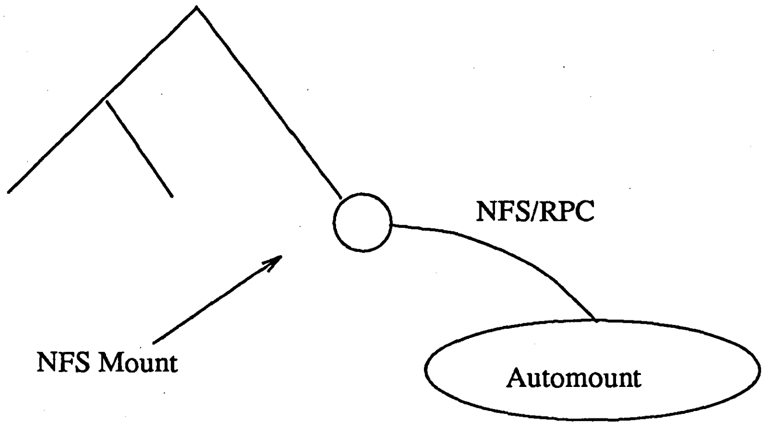
There are two distinct stages in the automounter's actions:

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

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

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

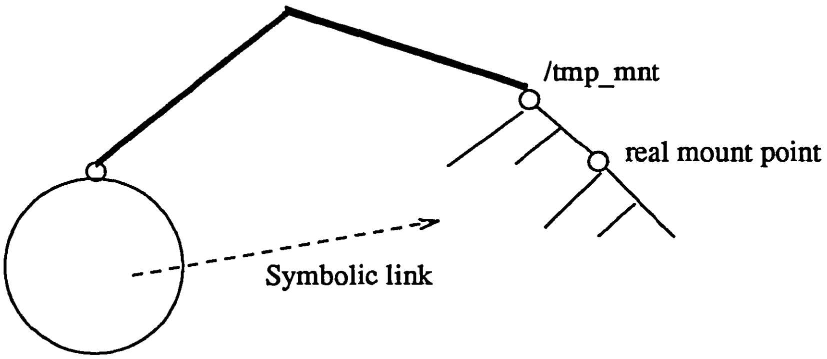
where `&nfs_args` contains the network address for the NFS server. By having the network address in `&nfs_args` refer to the local process (the automount daemon), `automount` in fact deceives the kernel into treating it as if it were an NFS server. Instead, once the parent process completes its calls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



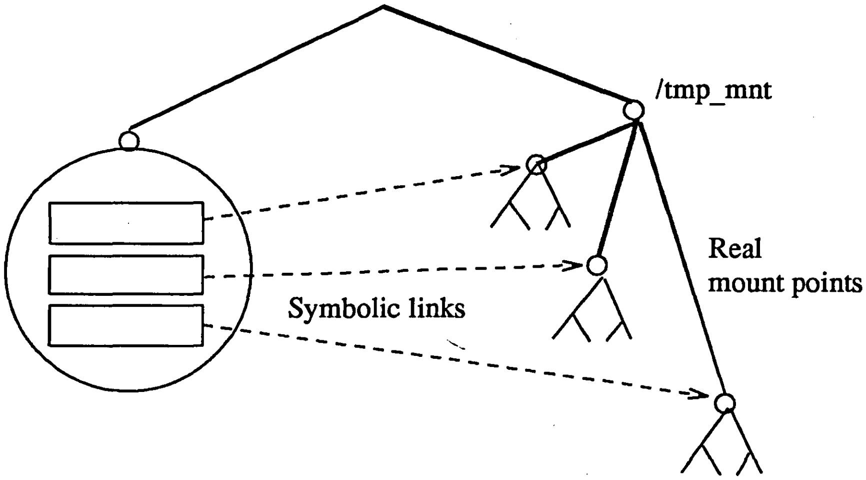
In the second stage, when the user actually requests access to a remote file hierarchy, the daemon intercepts the kernel NFS request and looks up the name in the map associated with the directory. Taking the location (*server:pathname*) of the remote file system from the map, the daemon then mounts the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

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

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



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



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

Summary:

When automount is called from the command line or rc.local, automount forks a daemon to serve each mount point in the maps and makes the kernel believe that the mount has taken place. The daemon sleeps until a request is made to access the corresponding file hierarchy. At that time the daemon does the following:

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

Preparing the Maps

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

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

There are three kinds of automount maps:

- master
- indirect
- direct

They are described below.

The Master Map

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

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

where:

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

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

Direct and Indirect Maps

Lines in direct and indirect maps have the syntax:

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

where

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

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

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

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

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

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

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

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

or specify, in the master map:

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

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

Writing a Master Map

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

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

A typical auto.master file would contain

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

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

Mount point /–

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

Mount point /home

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

Mount point /net

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

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

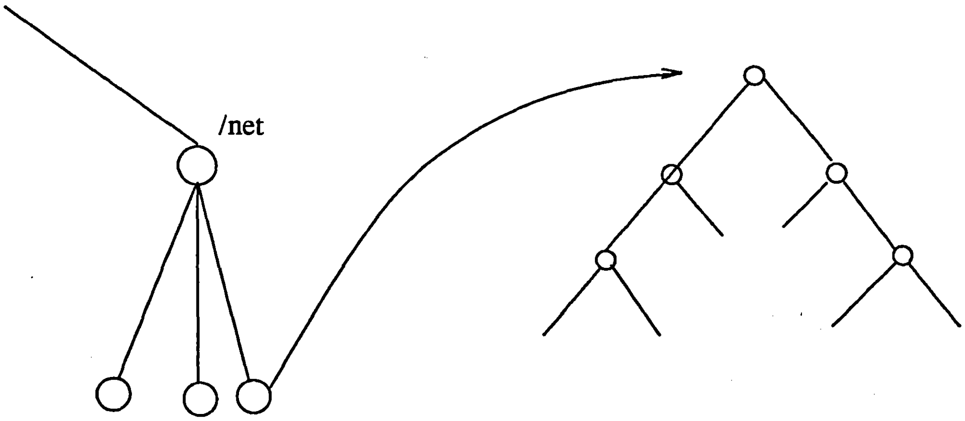
will change directory to the top of the hierarchy of files (i.e., the root file system) of the machine gumbo as long as the machine is in the hosts database. However, the user may not see under /net/gumbo all the files and directories. This is because the automounter can mount only the *shared* file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

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

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

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

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



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

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

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

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

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

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

Writing an indirect map

The syntax for an indirect map is:

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

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

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

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

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

A typical auto.home map might contain:

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

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

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

Now, assume the following conditions occur:

□ User laura's home directory is listed in the password database as /home/willow/laura.

□ Machine willow shares its home hierarchy with the machines mentioned in auto.home.

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

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

Furthermore, now laura can also enter the command

```
% cd ~brent
```

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

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

Writing a Direct Map

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

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

where:

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

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

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

appears in a direct map as:

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

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

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

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

Multiple Mounts

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

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

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

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

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

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

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

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

The following illustration shows a true hierarchical mounting:

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

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

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

Multiple Locations

In the example for a direct map, which was:

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

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

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

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

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

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

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

Specifying Subdirectories

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Substitutions

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

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

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

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

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

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

the use of the ampersand results in:

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

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

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

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

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

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

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

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

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

which is a good candidate to be written as:

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

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

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

because the automounter would interpret it as:

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

Special Characters

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

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

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

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

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

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

Environment Variables

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

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

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

and in willow it would be:

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

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

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

and have the entry in the direct map read:

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

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

Invoking automount

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

The syntax to invoke the automounter is:

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

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

Given the following set of three maps:

- auto.master

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

- auto.home

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

- auto.direct

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

The Temporary Mount Point

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

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

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

The Mount Table

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

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

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

```
% kill -1 PID
```

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

Modifying the Maps

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

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

Error Messages Related to `automount`

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

- `no mount maps specified`

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

- `*mapname*: Not found`

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

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

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

- `*mountpoint*: Not a directory`

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

- `hierarchical mountpoint: *mountpoint*`

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

- `WARNING: *mountpoint* not empty!`

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

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

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

- `hostname:file system already mounted on mountpoint`

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

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

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

- `couldn't create directory: reason`

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

- `bad entry in map mapname "map entry"`
- `map mapname, key map key: bad`

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

- `mapname: yp_err`

  Error in looking up an entry in a YP map.

- `hostname: exports: rpc_err`

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

- `host hostname not responding`
- `hostname:filesystem server not responding`
- `Mount of hostname:filesystem on mountpoint: reason`

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

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

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

- `Cannot create socket for broadcast rpc: rpc_err`
- `Many_cast select problem: rpc_err`
- `Cannot send broadcast packet: rpc_err`
- `Cannot receive reply to many_cast: rpc_err`

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

- `trymany: servers not responding:` *`reason`*

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

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

  Attempted remount after unmount failed. Indicates a server problem.

- `NFS server (pid`*`n`*`@`*`mountpoint`*`) not responding still trying`

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

14

The Sun Yellow Pages Service

14

The Sun Yellow Pages Service

4.0.3 Inclusion Instructions

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

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

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

14.1. The YP Environment

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

The YP Domain

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

YP Machine Types

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

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

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

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

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

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

```
% ypwhich hostname
```

where *hostname* is the name of the client.

YP Maps

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

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

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

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

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

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

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

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

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

```
ypcat mapname
```

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

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

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

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

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

Table 14-2 *YP Map Descriptions*

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

Table 14-2 *YP Map Descriptions— Continued*

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

YP Binding

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

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

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

YP and the Concept of Naming

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

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

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

Commands Used for Maintaining YP

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

`ypserv` — Looks up requested information in a map. `ypserv` is a daemon that runs on YP servers with a complete set of maps. At least one `ypserv` daemon must be present on the network for YP service to function.

`ypbind` — Initiates binding; it is the YP binder daemon. `ypbind` is present on both clients and servers. It initiates binding by trying to find a `ypserv` process that serves maps within the requesting client's domain. `ypserv` must run on each YP server. `ypbind` must run on all clients.

`ypinit` — Automatically creates maps for a YP server from files located in `/etc`. It also constructs the initial maps that are not built from files in `/etc`, such as `ypservers`. Use `ypinit` to set up the master YP server and the slave YP servers for the first time.

make

Updates YP maps. It is a version of the make command that reads the Makefile in /var/yp. You can use make to update all maps based on the files in /etc or to update individual maps. The man page ypmake(8) describes make functionality for YP.

makedbm

Enables you to update maps that are not built from the Makefile in /var/yp. makedbm takes an input file and converts it into dbm .dir and .pag files—valid YP maps. You can also use makedbm to "disassemble" a map, so that you can see the key-value pairs that comprise it.

ypxfr

Moves a YP map from one server to another, using YP itself as the transport medium. You can run ypxfr interactively, or periodically from a crontab file. (See Chapter 8 for information about running crontab.)

yppush

Copies a new version of a YP map from the YP master server to its slaves. You run it on the master YP server.

ypset

Tells a ypbind process to get YP services for a domain from a named YP server. This is not for casual use.

yppoll

Tells which version of a YP map is running on a server that you specify.

ypcat

Displays the contents of a YP map.

ypmatch

Prints the value for one or more specified keys in a YP map. You cannot specify which YP server's version of the map you are seeing.

ypwhich

Shows which YP server a client is using at the moment for YP services, or which YP server is master of a particular map.

ypupdated

Changes YP information. It is a daemon normally started up by inetd.

14.2. Preparing the YP Domain

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

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

Setting the Domain Name and Host Name

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

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

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

1. Log in as superuser.

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

   ```
   /bin/domainname
   ```

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

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

   Then close the file.

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

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

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

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

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

Changing the YP Domain Name

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

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

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

Then run

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

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

Preparing Network Databases for YP Service

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

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

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

Preparing Files on YP Clients

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

Preparing hosts.equiv

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

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

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

Preparing .rhosts

.rhosts also does not have an equivalent YP map. Chapter 13 fully discusses its format and restrictions. Because it controls remote root access to the local machine, unrestricted access to .rhosts is not recommended. Make the list of trusted hosts explicit, or use netgroup names for the same purpose. You can not use secondary hostnames in your .rhosts, hosts.equiv, or netgroup files. You can, however, use secondary hostnames in /etc/hosts. All of the above files are related in that they enable local machines to access remote machines in some fashion.

Preparing hosts

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

Preparing passwd

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

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

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

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

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

Preparing group

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

```
+:
```

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

Preparing Network Databases on the Master Server

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

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

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

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

14.3. Setting Up YP Servers and Clients

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

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

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

Setting Up a Master YP Server

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

Starting YP Service with ypinit

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

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

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

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

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

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

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

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

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

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

Creating the Master Server

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

1. From /var/yp, type the following

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

   to set up the master server.

2. Next, type

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

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

Setting Up a YP Slave Server

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

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

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

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

Now you are ready to create a new slave server.

1. Log in to the slave server as superuser.

2. Change directory to /var/yp.

3. Type the following:

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

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

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

5. When ypinit terminates, make copies of the following files:

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

   For example, you might type:

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

6. Edit the original files (not those with the – or `.old` extension) as described in the previous section, "Preparing Files for YP Service." This ensures that processes on the slave server actually use the YP services, rather than files in the local `/etc`. In this way, you ensure that the YP slave server is also a YP client.

7. Back up copies of the edited files, as well. For instance, you might type:

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

8. Type

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

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

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

Setting Up a YP Client

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

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

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

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

2. Add entries for the client in the following files:

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

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

4. Type

```
% ps -aux
```

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

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

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

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

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

```
% yppasswd login_name
```

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

14.4. Administering YP Maps

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

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

Updating Existing Maps

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

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

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

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

Modifying Maps based on `/etc` Files

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

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

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

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

Creating and Modifying Non-Standard Maps

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

There are two different methods for using `makedbm`:

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

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

Updating Maps Built from Existing ASCII Files

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

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

The mymap map now exists in the directory home_domain.

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

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

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

Updating Maps Built from Standard Input

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

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

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

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

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

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

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

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

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

Whether you use the Makefile in /var/yp or some other procedure Makefile— is one of many possible— the goal is the same: a new pair of well-formed dbm files must end up in the domain directory on the master YP server.

Propagating a YP Map

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

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

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

Using crontab with ypxfr

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

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

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

Using Shell Scripts with ypxfr

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

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

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

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

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

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

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

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

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

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

Directly Invoking `ypxfr`

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

Logging ypxfr's Activities

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

Propagating a YP Map to Another Domain

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

Adding New YP Maps to the Makefile

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

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

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

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

Adding a New YP Server to the Original Set

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

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

1. Go to the YP domain directory by typing:

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

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

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

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

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

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

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

Here makedbm converts ypservers back into dbm format.

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

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

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

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

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

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

Changing a Map's Master Server

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

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

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

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

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

3. Type the following:

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

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

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

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

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

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

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

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

14.5. Administering Users on a YP Network

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

How YP Maps Affect Security

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

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

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

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

```
Not in passwd file.
```

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

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

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

Changing the YP Password

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

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

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

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

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

```
% yppasswd user_name
```

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

How Netgroups Affect Security on a YP Network

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

Various programs use the /etc/netgroup-based YP maps for permission checking during remote mount, login, remote login, and remote shell creation. These programs include: login, mountd, rlogin, and rsh. login consults them for user classifications if it encounters netgroup names in /etc/passwd. The mountd daemon consults them for machine classifications if it encounters netgroup names in /etc/exports. rlogin and rsh consult the netgroup maps for both machine and user classifications if they encounter netgroup names in the /etc/hosts.equiv or /.rhosts file.

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

Adding a New User to a YP Server

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

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

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

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

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

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

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

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

Login name — Should be the same as user name in the local /etc/passwd file, and unique within the network domain.

Encrypted password — This is the password created by the yppasswd command. Have all new users run yppasswd. If a user forgets his or her password, you can make this field empty, enabling them to log in without a password and create a new one with passwd. An asterisk in this field matches no password.

User ID — A number that must be unique within the network domain, which you assign to this user. Failure to keep ID's unique prevents files on different machines from being moved between directories because the system will respond as if the directories are owned by two

	different users. Also, file ownership may become confused when an NFS server exports a directory to an NFS client whose password file contains users with user IDs that match those of different users on the NFS server.
Group ID	IDs which you assigned to groups created in the local `/etc/group` files.
User Information	Same as the local `/etc/passwd` file.
Home Directory	Same as the local `/etc/passwd` file.
Login Shell	Same as the local `/etc/passwd` file.

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

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

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

Making the Home Directory

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

```
   unknown user id:  username
```

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

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

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

14.6. Fixing YP Problems

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

Debugging a YP Client

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

Hanging Commands on the Client

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

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

Sometimes many commands begin to hang, even though the system as a whole seems okay and you can run new commands.

The message above indicates that ypbind on the local machine is unable to communicate with ypserv in the domain *domainname*. This happens when a machine running ypserv has crashed. It may also occur if the network or YP server is so overloaded that ypserv cannot get a response back to the client's ypbind within the timeout period.

Under these circumstances, every client on the network will experience the same or similar problems. The condition is temporary in most cases. The messages will usually go away when the YP server reboots and restarts ypserv, or when the load on the YP server or network itself decreases.

However, commands may hang and require direct action to clear them. The following list describes the causes of such problems and gives suggestions for fixing them:

- The YP client has not set, or has incorrectly set, domainname on the machine. Clients must use a domain name that the YP servers know.

  On the client type domainname to see which domain name is set. Compare that with the actual domain name in /var/yp on the YP master server. If a machine's domainname is not the same as the server's, the machine's domainname entry in rc.local is incorrect. Log in as superuser, edit the client's rc.local, and correct the domainname entry. This assures domain name is correct every time the machine boots. Then set domainname manually by typing:

  ```
      # domainname good_domain_name
  ```

- If your domain name is correct and commands still hang, make sure your local network has at least one YP server machine. Some network domain consist of two or more local networks joined by routers. A client can automatically bind only to a ypserv process on its local network, not on another accessible network. At least one YP server for a client's domain must running on the client's local network. Two or more YP servers improve availability and response characteristics for YP services.

- If your local network has a YP server and commands still hang, make sure the server is up and running. Check other machines on your local network. If several clients also have problems, suspect a server problem. Try to find a client machine behaving normally, and type the ypwhich command on it.

If ypwhich does not respond, kill it and go to a terminal on the YP server. Type:

```
# ps ax | grep yp
```

Look for ypserv and ypbind processes. If the server's ypbind daemon is not running, start it up by typing:

```
# /usr/etc/ypbind
```

If a ypserv process is running, type

```
# ypwhich
```

on the YP server. If ypwhich does not respond, ypserv has probably hung, and you should restart it. While logged in as superuser, type the following:

```
# kill -9 [some pid # from ps]
# /usr/etc/ypserv
```

If ps shows no ypserv process running, start one up.

YP Service is Unavailable

When most machines on the network appear to be okay, but one client cannot receive YP service, that client may experience many different symptoms. For example, some commands appear to operate correctly while others terminate with an error message about the unavailability of YP. Other commands limp along in a backup-strategy mode particular to the program involved. Still other commands or daemons crash with obscure messages or no message at all. Here are messages a client in this situation may receive:

```
samba% ypcat myfile
ypcat: can't bind to YP server for domain <domainname>.
    Reason: can't communicate with ypbind.
```

```
% /usr/etc/yp/yppoll myfile
Sorry, I can't make use of the yellow pages.  I give up.
```

On the problem client, run **ls -l** on a directory containing files owned by many users, including some not in the client's /etc/passwd file—for example /usr. If the resulting display lists file owners not in the local /etc/passwd as numbers, rather than names, this also means that YP service is not working.

These symptoms usually indicate that the client's ypbind process is not running. Run **ps ax** and check for ypbind. If it you do not find it, log in as superuser and start it as follows:

```
# /usr/etc/ypbind
```

YP problems should disappear.

ypbind Crashes

If ypbind crashes almost immediately each time it is started, look for a problem in some other part of the system. Check for the presence of the portmap daemon by typing:

```
% ps ax | grep portmap
```

If it is not running, reboot.

If portmap itself will not stay up or behaves strangely, look for more fundamental problems. Check the network software in the ways suggested in the section on Ethernet debugging in Chapter 12, "The SunOS Network Environment."

You may be able to communicate with portmap on the problematic client from a machine operating normally. From the functioning machine, type:

```
% rpcinfo -p client
```

If portmap on the problematic machine is okay, rpcinfo produces the following output:

```
program vers proto   port
  100007    2   tcp   1024  ypbind
  100007    2   udp   1028  ypbind
  100007    1   tcp   1024  ypbind
  100007    1   udp   1028  ypbind
  100021    1   tcp   1026  nlockmgr
  100024    1   udp   1052  status
         .
         .
         .
```

Your machine will have different port numbers. The four entries for the ypbind process are:

```
  100007    2   tcp   1024  ypbind
  100007    2   udp   1028  ypbind
  100007    1   tcp   1024  ypbind
  100007    1   udp   1028  ypbind
```

If they are not displayed, ypbind has been unable to register its services. Reboot the machine and run rpcinfo again. If the ypbind processes are there and they change each time you try to restart /usr/etc/ypbind, reboot the system, even if the portmap daemon is running. If the situation persists after reboot, call Sun customer support for help.

ypwhich Displays are Inconsistent

When you use ypwhich several times on the same client, the resulting display varies because the YP server changes. This is normal. The binding of YP client to YP server changes over time when the network or the YP servers are busy. Whenever possible, the network stabilizes at a point where all clients get acceptable response time from the YP servers. As long as your client machine gets YP service, it does not matter where the service comes from. For example, one YP server machine gets its own YP services from another YP server on the network.

Debugging a YP Server

Before trying to debug your YP server, read the beginning part of this chapter about the YP environment. Then look in this subsection for the heading that most closely describes the server's problem.

Servers Have Different Versions of a YP Map

Because YP propagates maps among servers, occasionally you find different versions of the same map at YP servers on the network. This version discrepancy is normal if transient, but abnormal otherwise.

Most commonly, normal map propagation is prevented if it occurs when a YP server or router between YP servers is down. When all YP servers and the routers between them are running, ypxfr should succeed.

If a particular slave server has problems updating maps, log in to that server and run ypxfr interactively. If ypxfr fails, it will tell you why it failed, and you can fix the problem. If ypxfr succeeds, but you suspect it has occasionally failed, create a log file to enable logging of messages. As superuser type:

```
ypslave# cd /var/yp
ypslave# touch ypxfr.log
```

This saves all output from The output resembles the output ypxfr displays when run interactively, but each line in the log file is timestamped. (You may see unusual orderings in the timestamps. That is okay— the timestamp tells you when ypxfr started to run. If copies of ypxfr ran simultaneously but their work took differing amounts of time, they may actually write their summary status line to the log files in an order different from that which they were invoked. Any pattern of intermittent failure shows up in the log. When you have fixed the problem, turn off logging by removing the log file. If you forget to remove it, it will grow without limit.

While still logged in to the problem YP slave server, inspect the system crontab file, /var/spool/cron/crontabs/root, and the ypxfr* shell scripts it invokes. Typos in these files cause propagation problems, as do failures to refer to a shell script within /var/spool/cron/crontabs/root, or failures to refer to a map within any shell script.

Also, make sure that the YP slave server is in the map ypservers within the domain. If it is not, it still operates perfectly as a server, but yppush will not tell it when a new copy of a map exists.

If the YP slave server's problem is not obvious, you can work around it while you debug using rcp or tftp to copy a recent version the inconsistent map from any healthy YP server. Be sure not do this remote copy as root, but you can probably do it while logged in as daemon. For instance, here is how you might transfer the map "busted:"

```
ypslave# chmod go+w /var/yp/mydomain
ypslave# su daemon
$ rcp ypmaster:/var/yp/mydomain/busted.\* /var/yp/mydomain
$ ^D
ypslave# chown root /var/yp/mydomain/busted.*
ypslave# chmod go-w /var/yp/mydomain
```

Here the * character has been escaped in the command line, so that it will be expanded on ypmaster, instead of locally on ypslave. Notice that the map files should be owned by root, so you must change ownership of them after the transfer.

ypserv Crashes

When the ypserv process crashes almost immediately, and does not stay up even with repeated activations, the debug process is virtually identical to that previously described in the subsection "ypbind Crashes.". Check for the existence of the portmap daemon as follows:

```
ypserver% ps ax | grep portmap
```

Reboot the server if you do not find the daemon. If it is there, type:

```
% rpcinfo -p yp_server
```

and look for output such as:

```
program vers proto   port
  100004    2   udp    1027   ypserv
  100004    2   tcp    1024   ypserv
  100004    1   udp    1027   ypserv
  100004    1   tcp    1024   ypserv
  100007    2   tcp    1025   ypbind
  100007    2   udp    1035   ypbind
  100007    1   tcp    1025   ypbind
  100007    1   udp    1035   ypbind
  100009    1   udp    1023   yppasswdd
  100003    2   udp    2049   nfs
  100024    1   udp    1074   status
  100024    1   tcp    1031   status
  100021    1   tcp    1032   nlockmgr
  100021    1   udp    1079   nlockmgr
  100020    1   udp    1082   llockmgr
  100020    1   tcp    1033   llockmgr
  100021    2   tcp    1034   nlockmgr
  100012    1   udp    1104   sprayd
  100011    1   udp    1106   rquotad
  100005    1   udp    1108   mountd
  100008    1   udp    1110   walld
  100002    1   udp    1112   rusersd
  100002    2   udp    1112   rusersd
  100001    1   udp    1115   rstatd
  100001    2   udp    1115   rstatd
  100001    3   udp    1115   rstatd
```

Your machine will have different port numbers. The four entries representing the `ypserv` process are:

```
  100004    2   udp    1027   ypserv
  100004    2   tcp    1024   ypserv
  100004    1   udp    1027   ypserv
  100004    1   tcp    1024   ypserv
```

If they are not present, `ypserv` has been unable to register its services. Reboot the machine. If the `ypserv` processes are there, and they change each time you try to restart `/usr/etc/ypserv`, reboot the machine. If the situation persists after reboot, call Sun for assistance.

14.7. Turning Off Yellow Pages Services

If `ypserv` on the master is disabled, you can no longer update any the YP maps. If you choose to turn off YP on a network currently running it, you can disable it by simply renaming the `/usr/etc/ypbind`file to `/usr/etc/ypbind.orig`. `suninstall` does this automatically if you tell it you do not want to run YP. Type the following:

```
% cd /etc
% mv /usr/etc/ypbind /usr/etc/ypbind.orig
```

To disable YP on a particular YP slave or master, type the following on the server in question:

```
% mv /usr/etc/ypserv /usr/etc/ypserv.orig.
```

Again, suninstall does this automatically if you do not select yp.

Refer back to Chapter 13, "The Sun Network File System," for information about the files you need to maintain for a server should you decide to disable YP.

14.8. The publickey Map

This file consists of three fields in the following format:

```
user name    user's public key : user's secret key
```

where *user name* may be the name of a user or of a machine, *user public key* is that key in hexadecimal notation, and *user secret key* is that key also in hexadecimal notation.

Since nobody expects you to be conversant in hexadecimal notation, the program newkey is provided to make things easier. Simply become superuser at the master server and invoke newkey for a given user:

```
# newkey -u username
```

or for the super user in a given host machine:

```
# newkey -h hostname
```

and at the prompt enter the appropriate login password. The program will then create a new public/secret key pair in /etc/publickey, encrypted with the login password of the given user.

Users can later on modify their own entries, or can even create them, by using the program chkey. The user simply types:

```
% chkey
```

and then responds to prompts from the command. A typical chkey session would look like this:

```
willow% chkey
Generating new key for username
Password: user enters password
Sending key change request to server...
Done.
willow%
```

Note that in order for newkey and chkey to be able to run properly, the daemon ypupdated must be running in the master server. If it is not running at this point, enter

```
# /usr/lib/netsvc/yp/ypupdated
```

and also make sure that the appropriate file in /etc/rc?.d contains the lines

```
if [ -f /usr/lib/netsvc/yp/ypupdated -a -d /var/yp/`domainname` ]
then
    /usr/lib/netsvc/yp/ypupdated
    (echo -n ' ypupdated') >/dev/console
fi
```

The ypupdated daemon consults the file /var/yp/updaters for information about which maps should be updated and how to go about it. In the case of the publickey map, changes to /etc/publickey affected through newkey or chkey are mediated by /usr/etc/yp/udpublickey.

Finally, ensure that the master YP server contains the empty file /etc/netid. You do not have to do anything to this file, but it must exist for public key encryption to work.

Automounting with YP

You can use YP along with the automounter to provide a single auto.master file for an entire network. You simply write the automounter files as they would reside in a machine's /etc directory.

A typical auto.master file might contain

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

A typical auto.home map might contain:

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

Here is a typical /etc/auto.direct map:

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

Refer back to the "Using the Automounter" section in Chapter 13, "The Sun Network File System," for full information about these files.

15

Addendum: Setting Up Electronic Mail

15

Addendum: Setting Up Electronic Mail

4.0.3 Inclusion Instructions

This section is a replacement for the first part of Chapter 15, "Electronic Mail and Communications." Replace all pages from the beginning of the chapter to the heading "Dial-up Networks."

SunOS electronic mail is handled by `sendmail`, a general internetwork mail routing service. `sendmail` features aliasing and forwarding, automatic routing to network gateways, and flexible configuration.

This section explains how to install the mail system on your computer. For a more detailed explanation, refer to Chapter 18, "Sendmail Installation and Operation."

The mail system consists of the following commands and files:

`/usr/ucb/mail`	UCB mail program, described in `mail(1)`
`/usr/bin/mailtool`	Window-based interface to
`/usr/lib/sendmail`	Mail routing program
`/usr/lib/sendmail.mx`	Mail routing program linked with the domain name service resolver.
`/etc/sendmail.cf`	Configuration file for mail routing
`/usr/lib/sendmail.main.cf`	Sample configuration file for main machines (see below)
`/usr/lib/sendmail.subsidiary.cf`	Sample configuration file for subsidiary machines (see below)
`/var/spool/mail`	Mail spooling directory for delivered mail
`/var/spool/mqueue`	Spool directory for mail going out over the network

/var/spool/mqueue	Spool directory for mail going out over the network
/var/spool/secretmail	Secure mail directory
/usr/bin/xsend	Secure mail sender
/usr/bin/xget	Secure mail receiver
/usr/bin/enroll	To receive secure mail messages
/etc/aliases	Mail forwarding information
/usr/ucb/newaliases	Symbolic link to /usr/lib/sendmail.
/usr/ucb/biff	Mail notification enabler
/usr/etc/in.comsat	Mail notification daemon
/usr/etc/in.syslog	Error message logger, used by sendmail

Users send mail by using the mail command or mailtool, a user interface fully described in *Mail and Messages: Beginner's Guide*, to edit the messages sent and received. Both pass these messages to sendmail for routing. (Refer to mail(1) and sendmail(8) for detailed information about these programs.)

sendmail processes each piece of mail, using information in the configuration file /etc/sendmail.cf to determine network name syntax, aliasing and forwarding information, and network topology. Local mail is delivered by giving it to the program /bin/mail, which adds it to the mailboxes in the directory /var/spool/mail, using a locking protocol to avoid problems with simultaneous updates. The file /var/spool/mail/*username* normally contains mail for each user name. After the mail is delivered, the local mail notification daemon /usr/etc/in.comsat notifies users who have the command biff y in their .login files, or who use mailtool, that mail has arrived.

Only you can read your mail file. However, anyone with the superuser password can read others' files, including their mail. To send mail that is secure against any possible perusal (except by a code-breaker), use the secret mail facility, which encrypts the mail so that no one can read it. The man page xsend(1) fully describes this facility. Note that xsend does not work over the network.

Domain Names and sendmail

By default, sendmail uses Internet standard domain names, first described in Chapter 12, "The SunOS Network Environment." These standards make it possible for any Internet system in the world to send or receive mail with any other Internet system. This is why each Internet system must have a unique name. As explained in Chapter 12, a domain is an administrative division and has nothing to do with the connectivity of the network. Typically, all machines in given domain are connected to each other, but this is only an administrative convenience.

As you have seen previously, Internet names are divided into domains and subdomains. There are only a small number of top-level domains, for example .COM and .EDU. In some countries, the name of the country is the top level domain. For example, some systems in the United Kingdom use the top level domain `.UK`. In the United States, the top level domain `.US` is used for some personal systems.

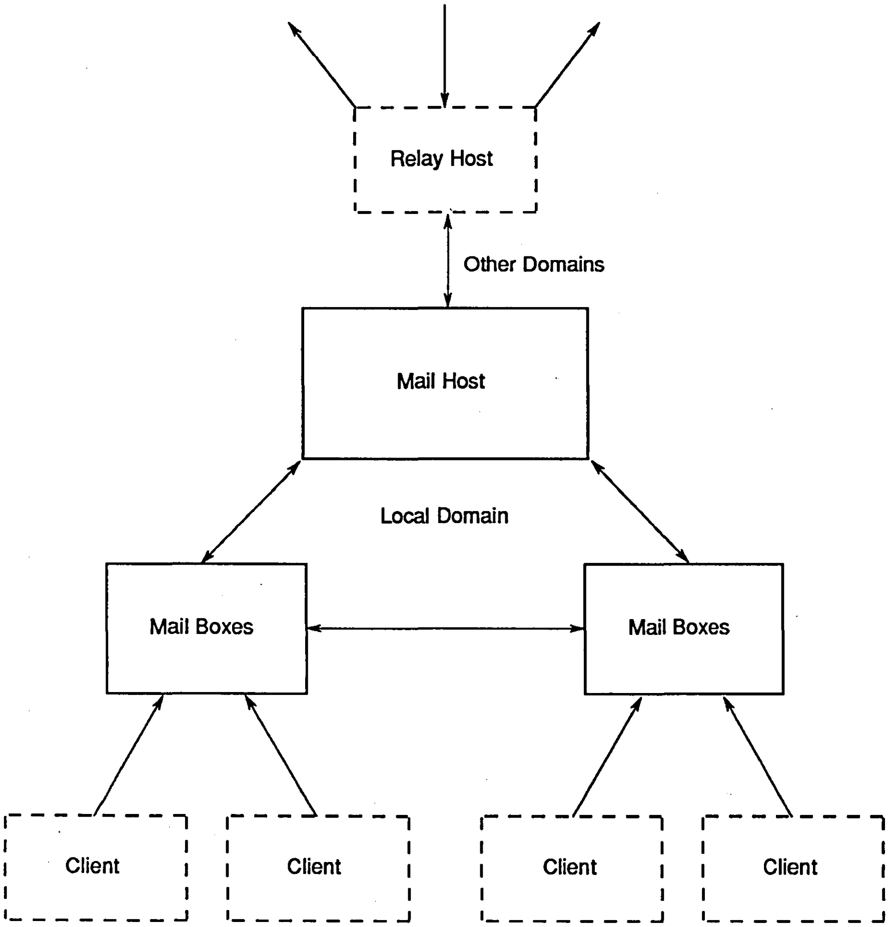
Domains nest inside one another like directories, except that the names go from right to left. Thus `joe.sun.COM` is the name of host joe in subdomain sun, which is in domain `.COM`, in the same way that `/etc/hosts.equiv` is file `hosts.equiv` in directory `/etc`.

If you have not done so already, you should register your domain with the Network Information Center (NIC) at SRI International, as described in Chapter 12.

15.1. Configuring Machines for Mail Operation

From a `sendmail` perspective, three types of machines exist in a network: at least one mailbox server, mail clients, and a mail host. Figure 15-1 illustrates their relationship to one another.

Figure 15-1 *A Typical Electronic Mail Configuration*



This section defines these machines and explains how to configure them.

Setting Up an NFS Mailbox Server and Its Clients

A *mailbox server* is any machine that actually stores mailboxes in the directory `/var/spool/mail`. In Release 4.0, client machines can mount their mailboxes through NFS from a mailbox server. This enables users to log in to any machine, including the server and read their mail. You can designate an NFS server as a mailbox server by having it export `/var/spool/mail`. However, other types of machines, even diskless clients, can operate as mailbox servers.

`/var/spool/mail` holds the individual mailboxes for users on the network. Each file called `/var/spool/mail/`*user_name* contains the individual mailbox for a particular user.

Any NFS server can be a mailbox server, including machines from companies other than Sun or machines running earlier releases of SunOS. To set up a machine as an NFS mailbox server, you need to edit `/etc/exports`, so that the server exports `/var/spool/mail`.

The mailbox server is responsible for sending outgoing mail from a client. When a client sends mail, the mailbox server puts it in a queue for delivery. Thereafter, the client can reboot or even power down, yet its mail will safely reach the recipient—baring other network problems, of course. When the recipient gets the client's mail, the path in the message's "From:" line contains the name of the mailbox server. If the recipient chooses to respond, the response goes to the user's mailbox in `/var/spool/mail` on the server, not directly to the client.

Converting Clients to Use Mailbox Servers

The following instructions explain how to set up the mailbox server's clients:

1. Make sure clients are halted so that no mail is lost during the conversion.
2. Log in as superuser on the mailbox server.
3. If any mailboxes exist on the client, move them to the mailbox server's mailbox directory. Type the following:

```
# mv /export/root/client_name/var/spool/mail/* /var/spool/mail
```

4. Edit `fstab` in each client's `/export/root/`*client_name*`/etc` directory, adding an entry of the form:

```
mailbox_server:/var/spool/mail   /var/spool/mail
```

   where *mailbox_server* is the server's name.

5. When you are finished editing their `/etc/fstab` files reboot each client to have the new `/etc/fstab` file take affect.

Once you have performed these tasks, mail operations can proceed. The mailbox server receives incoming mail from `sendmail`. The `/bin/mail` program running on the server puts the mail in the appropriate mailbox in `/var/spool/mail`. Each client mounts its mailbox through the `/var/spool/mail` entry in its `/etc/fstab` file.

Setting Up the Mailbox Server Alias

The next step in setting up mailbox servers is to assign an alias to the mailbox server. You do this by editing the `/etc/aliases` file. (The later section, "Setting Up the Postmaster Alias" fully describes `/etc/aliases`.)

Suppose machine "ballet" is the mailbox server for a particular organization or group. The existing `/etc/aliases` file on the network might resemble:

```
root: sysadmin@ballet
shamira: shamira@raks
benny: benny@samba
```

where "ballet" is the mailbox server name and "raks" and "samba" are client names.

To set up the mailbox server alias for server ballet, you would first log in to ballet. Then change the client names to the mailbox server names, so that `/etc/aliases` might look like:

```
root: sysadmin@ballet
shamira: shamira@ballet
benny: benny@ballet
```

Be very careful that all aliases in the file resolve to names on the mailbox server, not the clients. In other words, if you leave the entry for user shamira as

```
shamira: shamira@raks
```

and do not change it to

```
shamira: shamira@ballet
```

shamira's outgoing mail will take an extra trip over the network between raks and ballet.

Setting Up the Mailhost and Subsidiary Machines

The next step in mail configuration is to define which machine on your network is the the main mail machine, or *mailhost*, and which machines are mail subsidiaries. Subsidiary mail machines directly distribute mail to recipients in the same domain. However, when a mail subsidiary encounters mail destined for a machine in a different domain, the subsidiary sends it to the mailhost for delivery.

The mailhost must have the file `/usr/lib/sendmail.main.cf` installed as `/etc/sendmail.cf` in its root file system. Subsidiary mail machines have the file `/usr/lib/sendmail.subsidiary.cf` installed as `/etc/sendmail.cf` in their root file systems. The mailhost/subsidiary configuration simplifies administration by reducing the number of machines with custom mail configuration files. In most cases, you will find the default `/usr/lib/sendmail.main.cf` and `/usr/lib/sendmail.subsidiary.cf` appropriate for your network. However, you may want to make a few simple changes to customize the `sendmail.cf` files for your particular network. Should you want to tailor these configuration files for your network, refer to Chapter 18, "Sendmail Installation and Operation," for more information.

A good candidate for mailhost is a machine attached to an Ethernet and to phone lines, or a machine configured as a router to the Internet. Note that if you have a non-networked standalone in a time-sharing configuration, treat it like the mailhost in a one machine network. Similarly, if you have several machines on an Ethernet and none have phones, pick one as the mailhost and leave the others as subsidiaries. You might connect your network to other domains in the future, perhaps by using the Sunlink MHS product.

Configuring Subsidiary Machines

suninstall installs the default subsidiary configuration file, /usr/lib/sendmail.subsidiary.cf on each machine. Thus, you do not have to do anything more to configure a subsidiary machine, unless you want to use a configuration file other than the default. Refer to Chapter 18 for specific instructions. Then put the subsidiary configuration file in /export/root/proto.local/etc/sendmail.cf.

You can make simple changes to the subsidiary configuration file to fit the needs of your particular network. For example, you can change the way in which the network domain name appears in mail messages.

By default, the visible part of the network domain name (as described in Chapter 12) is used for both outgoing and incoming mail. If you want the domain name to appear differently for incoming and outgoing mail, you need to add the appropriate lines to sendmail.cf. In this file, lines beginning with the letters "Dm" set the domain name for outgoing mail; lines beginning with "Cm" set the domain name for incoming mail.

Note: The following instructions also apply when you are configuring the mail host

Suppose your network has the domain name hq.dance.com. Its displayed domain name on mail messages would be dance.com. To change the domain name on outgoing and incoming mail, do the following:

1. Log in as superuser on the subsidiary machine.
2. Edit sendmail.cf.
3. If you want to change the displayed domain name on outgoing mail, add a line beginning with Dm, in this case

   ```
   Dmdance.com
   ```

   Change the name following Dm to the name you want displayed.
4. To change the domain name for incoming mail, add a line beginning with Cm, in this case

   ```
   Cmdance.com
   ```

   Replace the name following Cm to the one you want accepted, for example:

   ```
   Cmdance.uucp
   ```

   You can state a list of acceptible domain name aliases after Cm. Thereafter, sendmail will recognize that incoming mail sent to any of these domain names should in fact be sent to the local domain.

On a subsidiary machine with phone lines, you can edit the /etc/sendmail.cf file so that sendmail routes mail received from uucp to certain hosts via the local phone lines. This is more efficient than having all uucp traffic go through the mailhost. (The second half of this chapter explains how to set up uucp.) Follow these procedures.

1. Log in as superuser on the subsidiary machine with a phone line.
2. Edit sendmail.cf and find the following line:

```
# local UUCP connections -- not forwarded to mailhost
CV
```

3. Put the names of the local uucp sites on the end of the CV line, or create additional CV lines.

   For example, you could do the following:

```
    CV rome prussia georgia
```

Configuring the Mail Host

The first step in creating the mail host is to have the file /usr/lib/sendmail.main.cf become the mail configuration file for that machine. Follow this procedure:

1. Log in as superuser on the machine to become mail host.
2. Type the following:

```
# cp /usr/lib/sendmail.main.cf /etc/sendmail.cf
```

   Thereafter, sendmail will read /etc/sendmail.cf and recognize that this machine is the mail host.

3. Place the mailhost name for your new mail host in the /etc/hosts file on the master YP server, or on all hosts on a network without YP.

   For example, suppose you've selected the machine ballet as the mailhost. Its entry in /etc/hosts should look like:

```
    129.255.99.99   ballet mailhost
```

You can optionally edit /etc/sendmail.cf on the mailhost to suit your particular network. Your network may have a uucp or Ethernet connection with a machine called a *relay host* that will relay mail to you. The relay host runs at least one mail-related protocol called a *mailer*. Each mailer specifies a policy and the mechanics to use when delivering mail. sendmail provides for several different kinds of mailers. The sample sendmail.cf file defines several available mailers, such as smartuucp, ddn, ether, and uucp. You can add others, as described in Chapter 18.

Once you have found a willing relay host, look for the following block of lines in `sendmail.cf`:

```
# major relay mailer
DMsmartuucp

# major relay host
DRddn-gateway
CRddn-gateway
```

Change the line `DMsmartuucp` to the name of the mailer that you connect to the mail host, for example, `uucp` or `ddn`, and the name following the letters `DR` to the name of the relay host.

For example, if your local network includes a machine called "cmu-cs-vlsi" that is on the Internet, you might use the following entry:

```
# major relay mailer
DMddn

#major relay host
DRcmu-cs-vlsi
CRcmu-cs-vlsi
```

On the other hand, your relay host might be `uucp` host ucbvax, which means you might use the following entry:

```
# major relay mailer
DMuucp

#major relay host
DRucbvax
CRucbvax
```

This change enables you to mail to an address such as "charlie@MIT.EDU." Even though your mail host may not be on the Internet, the message will arrive. If you are using Sunlink/MHS, refer to your installation manual for more information.

Setting Up the Postmaster Alias

Someone at your site, possibly yourself, should have the responsibility for handling problems with electronic mail. This person should have the alias *Postmaster*, a title which is recognized throughout the electronic mail community. For example, you can send mail to `postmaster` at other network sites if you have problems with mail originating at those sites.

The `/etc/aliases` File

The `/etc/aliases` file on a local machine contains all names by which a machine or person is known; `sendmail` reads it to determine mailing addresses. It is completely described in the man page `aliases`(5). You can use uppercase letters in names to the left of the colon in `/etc/aliases`. However, it is much safer to only use lowercase letters. `/etc/aliases` has local aliases for individuals and groups, and special aliases. A local alias has the form

```
name: address[, address]
```

where *name* is the person or group's name and *address* is the address of a recipient in the group. A typical local alias might be:

```
benny:benny@samba
```

A special alias has the form:

```
owner-aliasname: address
```

`postmaster` is a special alias. Every local `/etc/aliases` file should have a postmaster entry; here is the default entry:

```
# Following alias is required by the mail protocol, RFC 822
# Set it to the address of a HUMAN who deals with this system's mail problems
postmaster: root
```

To establish the postmaster alias, change "`root`" to the user name of the person who will act as postmaster. Then messages directed to "postmaster" arrive with that person's other mail. If this postmaster also manages a domain with more than one host, add the postmaster alias to `/etc/aliases` on all hosts, or, for networks running YP, in the `mail.aliases` map on the YP master server.

If you manage the mail system for several domains, change `/etc/aliases` on all of them to forward postmaster mail to the machine where the postmaster usually reads mail. Suppose you are user amina, the network postmaster, and you use machine raks. You would use the following entry for postmaster in `/etc/aliases`:

```
postmaster: amina@raks
```

As postmaster, you may not want to have users' mail mixed in with your personal mail. Here are procedures that help you avoid this by redirecting postmaster mail into a separate file on your machine.

1. Log in as superuser on each mail client. (If your network runs YP, you only have to perform Steps 1 and 2 on the master YP server.)

2. Add an alias such as:

   ```
       postmaster: sysadmin@raks
   ```

   to each mail client's `/etc/aliases`. This entry tells `sendmail` to direct mail to the postermaster alias to `sysadmin` on machine raks.

3. Log in as superuser on your own machine.

4. Add your own local mail alias to /etc/aliases. For example, postmaster amina would add the following entry to /etc/aliases on her own machine, raks

```
        sysadmin:/home/amina/mailadm
```

   This entry defines an alias for sysadmin: the file /home/amina/mailadm.

5. Exit superuser and log in with your own user name.
6. Type the following to create the file mailadm:

```
        % touch /home/amina/mailadm
        % chmode og+w
```

7. Type the following to have the file read:

```
        % mail -f mailadm
```

   You can also create aliases for people or groups of people called *mailing lists* when setting up the postmaster alias. The actual /etc/aliases file contains instructions for doing this.

Each time you edit /etc/aliases, you must run the newaliases program to rebuild the local alias database. If your network runs YP, run make in /var/yp on the master YP server after updating domain wide aliases.

Handling Undelivered Mail

By default, any time a message is returned as undeliverable by sendmail, a copy of the message header is sent to the postmaster. You can optionally disable this feature by editing the sendmail.cf file. Look for the following lines:

```
    # CC my postmaster on error replies I generate
    OPPostmaster
```

and change them as shown:

```
    # CC my postmaster on error replies I generate
    OPnobody
```

As shown above, you might want to create a separate file for undelivered mail, instead of mixing it with your personal mail.

Special Considerations for Networks with YP

Networks running YP have additional steps you must take and additional services you can add to sendmail service. Most importantly, after you have finished modifying /etc/hosts and /etc/aliases as described previously, remember to propagate the YP maps associated with these files. Type the following:

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

You can now use mail.aliases for domain-wide mail aliases. Thereafter, you should not have to remember hostnames when sending mail. mail.aliases will usually contain a copy of all the aliases known to a central mail machine.

Using Inverted YP Domain Names

The inversion of the YP domain-wide aliases can be used to simplify mail going outside the current domain. For example, consider the following /etc/aliases entries:

```
benny:benny@samba
gkelly:kelly@jazz
```

e: On networks with YP, you do not have to explicitly state a hostname. sendmail gets this information from the mail.aliases map.

When user kelly sends mail to user benny, the mail header displays the sender name, kelly@jazz. However, if kelly sends a message to a user called placido@song.com, the sender name on the message will be gkelly@dance.com. "dance.com" is the visible part of the domain name.

For incoming mail, sendmail normally replaces the left side of an alias entry with the right side. Thus the left side of the alias gkelly is replaced by the right side of the alias, kelly@jazz.

sendmail performs inverted alias processing on outgoing mail, replacing the right side of the alias with the left. This prevents the specific mailbox servers on a network from being visible outside the local domain. Therefore, if user kelly decides to switch to using host jazz as a mailbox server, user placido@song.com can still reply to gkelly@dance.com. The mail will automatically go to the right mailbox server.

Mail and the Internet Domain Name Server

In a domain running YP, the YP maps hosts.byname and host.byaddr resolve host names and addresses. You may wish to use the Internet domain name service for machines directly connected to the Internet. Consider doing this only if you can route between your machine and one of more of the "official" root servers. This will be the case if you are on the ARPANET, MILNET, or NSFNET.

Even if your are on one of these networks, you should still keep the sendmail.cf files on all clients and mailbox servers with standard configurations. You only need to customize the mail host sendmail.cf to take advantage of domain name service.

Install /usr/lib/sendmail.mx in place of /usr/lib/sendmail on the mail host to use domain name service. Each machine running sendmail.mx must have either /etc/resolv.conf or /etc/named.boot set up properly to allow name resolution or at least a caching server. Refer to Chapter 22 and the resolv.conf(5) man page for more information.

Testing Your Mail Configuration

First, reboot all systems whose configuration files you have changed. Then, send test messages from various machines on the network. To do so, go to a particular machine and type the following:

```
samba% /usr/lib/sendmail -v </dev/null names
```

This command sends a null message to the specified recipient name, and displays messages while it runs. Try the following tests:

- Send mail to yourself or other people on the local machine by addressing the message in the command above to a regular user name, such as `root`.
- If you are on an Ethernet, send mail to someone on another machine, as in `root@jazz`. Try this in three directions: from the main machine to a subsidiary machine, vice versa, and from a subsidiary machine to another subsidiary machine, if more than two are on the network.
- If you have a relay host, send some mail to another domain from the mailhost. This ensures that the relay mailer and host are selected properly.
- If you have set up a `uucp` connection on your phone line to another host, you can send mail to someone at that host and they can send mail back, or call you on the phone if they receive it.

  Try having them send mail to you. For example, you could send to ucbvax!azhar if you have a connection to ucbvax. `sendmail` will not be able to tell you whether the message really got through, since it hands the message to `uucp` for delivery. You have to ask the human at the other end if mail has been received. To get an idea of the message's progress, look in the file `/var/spool/uucp/LOGFILE`. For more information, refer to Chapter 21.
- Send a message to "postmaster" on various machines and make sure that it comes to your postmaster's mailbox, so that when other sites send you mail as postmaster, you will see it.

Diagnosing Problems with Mail Delivery

Here are some tools you can use for diagnosing mail problems.

The `/usr/lib/sendmail` command has a number of options for troubleshooting problems. For example, you can type the following:

```
% /usr/lib/sendmail -v -bv recipient
```

to verify the aliases and the deliverability of a given *recipient*. Here is an example of the resulting display from this command:

```
% /usr/lib/sendmail -v -bv shamira@raks
shamira... aliased to   mwong
mwong... aliased to               shamira@raks
shamira@raks... deliverable
```

sendmail also includes a test mode invoked by the bt option, as in

```
% /usr/lib/sendmail -bt
```

Here are instructions for using test mode.

1. Invoke test mode by typing the following:

```
    % /usr/lib/sendmail -bt
    ADDRESS TEST MODE
    Enter <ruleset> <address>
```

2. Respond to the last prompt by typing a zero, then the mail address you want to test, for example, benny@samba.

```
    > 0 benny@samba
    rewrite: ruleset  3   input: "benny" "@" "samba"
    rewrite: ruleset  6   input: "benny" "<" "@" "samba" ">"
            .
            .
            .
```

   The diagnostic information that sendmail displays is fully described in Chapter 18.

The mconnect program is another diagnostic tool that you can use to open connections to other sendmail systems over the network. mconnect runs interactively, so that you can issue it various diagnostic commands. For example, the VRFY command of SMTP (the protocol used to deliver mail over the network) performs the same operation as the -bv option to sendmail, and VERB invokes verbose mode like the -v option.

Other diagnostic tools include:

- Received lines in the header of the message.

  These trace which systems the message was relayed through. Note that in the uucp network, many sites do not update these lines, and that in the Internet, the lines often get rearranged. You can straighten them out by looking at the date and time in each line. Don't forget to account for different time zones.

- Messages from " MAILER-DAEMON."

  These typically report delivery problems.

- The system log, for delivery problems in your group of workstations.

  sendmail always records what it is doing in the system log, as explained in the next subsection. You might want to modify the crontab file to run a shell script nightly that searches the log for SYSERR messages and mails any that it finds to postmaster. In this way, problems are often fixed before anyone notices them, and the mail system runs more smoothly.

The System Log

The system log is supported by the syslog program, which is fully described in the man page syslog(8). Just as you define a machine called "mailhost" to handle mail relaying, you can define a machine called "loghost" in /etc/hosts to hold all the logs for an entire YP domain.

Format

Each line in the system log consists of a timestamp, the name of the machine that generated it (for logging from several machines over the Ethernet), and a message.

Levels

syslog can log a large amount of information. The log is arranged as succession of levels. At the lowest level, only unusual occurrences are logged. At the highest level, even the most mundane and uninteresting events are recorded for posterity. As a convention, log levels under ten are considered "useful;" log levels above ten are usually for debugging purposes.

Index