Solaris™ 10 for Experienced System Administrators SA-225-S10

Student Guide

Sun Microsystems, Inc.
UBRM05-104
500 Eldorado Blvd.
Broomfield, CO 80021
U.S.A.

Revision A

Table of Contents

About This Course

Course Goals

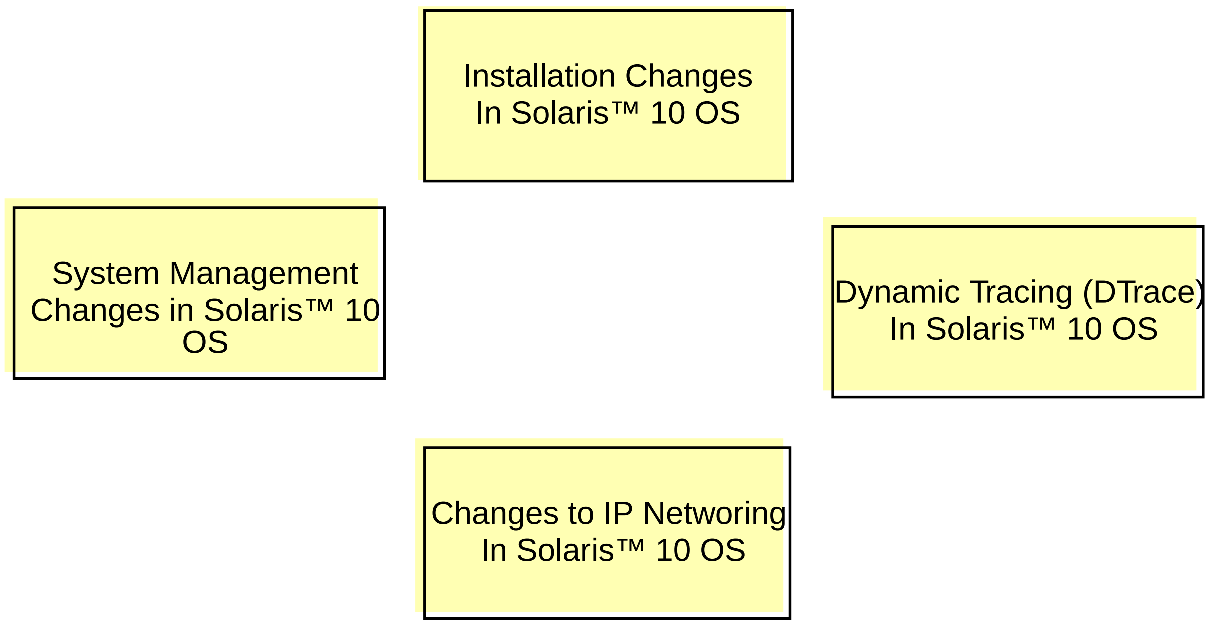
The course is divided into four sections with a total of 11 modules. A student workbook will provide you the opportunity to practice some of the key tasks and examine and test the operating system changes and enhancements.

Upon completion of this course, you should be able to identify changes and enhancements to the following features of the Solaris™ 10 Operating System:

- Identify the changes to the installation
- Identify the new features and changes to system management
- Investigate the functional capabilities of the Dynamic Tracing performance and troubleshooting software tool.
- Investigate the new features and functionality improvements made to IP networking protocol stack

Course Map

The following course map enables you to see what you have accomplished and where you are going in reference to the course goals.


Installation Changes In Solaris™ 10 OS
System Management Changes in Solaris™ 10 OS
Dynamic Tracing (DTrace) In Solaris™ 10 OS
Changes to IP Networking In Solaris™ 10 OS

Topics Not Covered

This course does not cover the following topics. Many of these topics are covered in other courses offered by Sun Educational Services:

- Concepts covering desktop changes
- Concepts covering Enterprise Volume Manager

Refer to the Sun Educational Services catalog for specific information and registration.

How Prepared Are You?

To be sure you are prepared to take this course, can you answer yes to the following questions?

- Can you support Solaris 9 OS?
- Can you create and edit text files using a text editor?
- Can you use a World Wide Web (WWW) browser?

Introductions

Now that you have been introduced to the course, introduce yourself to the other students and the instructor, addressing the following items:

- Name
- Company affiliation
- Title, function, and job responsibility
- Experience related to topics presented in this course
- Reasons for enrolling in this course
- Expectations for this course

How to Use Course Materials

To enable you to succeed in this course, these course materials contain a learning module that is composed of the following components:

- Goals – You should be able to accomplish the goals after finishing this course and meeting all of its objectives.
- Objectives – You should be able to accomplish the objectives after completing a portion of instructional content. Objectives support goals and can support other higher-level objectives.
- Lecture – The instructor presents information specific to the objective of the module. This information helps you learn the knowledge and skills necessary to succeed with the activities.
- Activities – The activities take oemailsn various forms, such as an exercise, self-check, discussion, and demonstration. Activities help you facilitate the mastery of an objective.
- Visual aids – The instructor might use several visual aids to convey a concept, such as a process, in a visual form. Visual aids commonly contain graphics, animation, and video.

Conventions

The following conventions are used in this course to represent various training elements and alternative learning resources.

Icons

Note – Indicates additional information that can help students but is not crucial to their understanding of the concept being described. Students should be able to understand the concept or complete the task without this information. Examples of notational information include keyword shortcuts and minor system adjustments.

Caution – Indicates that there is a risk of personal injury from a nonelectrical hazard, or risk of irreversible damage to data, software, or the operating system. A caution indicates that the possibility of a hazard (as opposed to certainty) might happen, depending on the action of the user.

Typographical Conventions

`Courier` is used for the names of commands, files, directories, programming code, and on-screen computer output; for example:

> Use `ls -al` to list all files.
> `system% You have mail`.

`Courier` is also used to indicate programming constructs, such as class names, methods, and keywords; for example:

> The `getServletInfo` method is used to get author information.
> The `java.awt.Dialog` class contains `Dialog` constructor.

`Courier bold` is used for characters and numbers that you type; for example:

> To list the files in this directory, type:
> `#` **`ls`**

`Courier bold` is also used for each line of programming code that is referenced in a textual description; for example:

```
1 import java.io.*;
2 import javax.servlet.*;
3 import javax.servlet.http.*;
```

Notice the `javax.servlet` interface is imported to allow access to its life cycle methods (Line 2).

`Courier italics` is used for variables and command-line placeholders that are replaced with a real name or value; for example:

To delete a file, use the `rm` *`filename`* command.

`Courier italic bold` is used to represent variables whose values are to be entered by the student as part of an activity; for example:

Type **`chmod a+rwx`** ***`filename`*** to grant read, write, and execute rights for *`filename`* to world, group, and users.

Palatino italics is used for book titles, new words or terms, or words that you want to emphasize; for example:

Read Chapter 6 in the *User's Guide*.
These are called *class* options.

Section I: Solaris™ 10 Operating System Installation

Objectives

Upon completion of this section, you should be able to:

- Understand the changes to the installation procedures and install the operating system.

Module 1

Solaris™ 10 Operating System Installation

Objectives

This module is an overview of the new Solaris™ 10 Operating System (Solaris 10 OS) installation capabilities. Upon completion of this module, you should be able to:

- Describe changes to the Solaris JumpStart™ software and configure, enable, and troubleshoot JumpStart
- Describe changes to Flash archives
- Describe changes to the Solaris Live Upgrade boot environment and configure, enable, and troubleshoot the Live Upgrade boot capability
- Describe the WAN Boot utility and configure, enable, and troubleshoot the WAN Boot utility
- Describe package and patch web installation and configure, enable, and troubleshoot package and patch web installation
- Describe the installation license panel

JumpStart Changes

The JumpStart process is accomplished through multiple tools and multiple configuration files. Changes to the way information is passed from server to client, additional configuration keywords, and a new metacluster, increase flexibility and control of the JumpStart process.

New JumpStart Features

JumpStart can work with a new DHCP vendor option that allows for the specification of the `tftpboot` server as a Universal Resource Identifier (URI). The URI can specify either a *tftp* or an *http* resource. *Tftp* can be used for traditional `inetboot` JumpStart installations and the new WAN Boot installations, while *http* can only be used with WAN Boot.

The new metacluster `SUNWCrnet` creates a more secure installation option. `SUNWCrnet` or the Reduced Networking Metacluster, as it is called, is between `SUNWCreq` and `SUNWCmreq` in functionality. `SUNWCreq` has several network services available and `SUNWCmreq` has no network services available. `SUNWCrnet` has only name service and Kerberos support.

New JumpStart profile keywords enable the creation of Solaris Volume Manager mirrored metadevices and metastate databases as part of the JumpStart process. Previously these devices had to be created with finish scripts or they were created manually after the JumpStart procedure had completed. These new keywords increase the simplicity and speed with which a JumpStart installation, utilizing Solaris Volume Manager mirrored metadevices, can be accomplished. Additional keywords or enhancements to keywords allow for the installation of packages that are not part of the installation media. These keywords also allow for the installation of patches during the JumpStart procedure and allow for the installation of additional packages during a Flash archive JumpStart installation.

DHCP Changes

DHCP allows a system to request an IP address and other information from a DHCP server. The behavior and syntax on the client side has not changed but the changes to DHCP involve a new vendor option on the DHCP server that allows the boot server and boot file to be specified together as a Universal Resource Identifier (URI). The new option is supported by all Solaris OBP 4.x systems, and some 3.x desktops and servers. The functionality is being delivered as FCODE with no platform dependencies. The `/etc/dhcp/inittab` file lists the new vendor options as numbers 16 and 17. The new vendor options are as follows:

```
SbootURI Symbol Vendor=SUNW.Sun-Blade-100 <other
architectures>,16,ASCII,1,0
SHTTPproxy Symbol Vendor=SUNW.Sun-Blade-100 <other
architectures>,17,ASCII,1,0
```

Previously the BOOTP/DHCP packet header used the *`siaddr`* (server IP address) field to specify the server that was to be contacted using `tftp` and the *`file`* field to specify the file that was to be retrieved by that `tftp` request. The *`file`* field now only accepts URIs. The new syntax for the *`file`* field looks like the following examples:

```
bootURI="tftp://172.21.0.88/tftpboot/C00A0A04.sun4u"
```

or

```
bootURI="http://172.21.0.88/tftpboot/C00A0A04.sun4u"
```

Metacluster Changes

The new metacluster (`SUNWCrnet`) was created to provide an installation footprint that would allow for identification via network naming services and for the installation of additional software. `SUNWCrnet` is designed to provide the core operating system functionality including *libm* and generic device driver modules, install and patch utilities, together with utilities commonly used in packaging scripts including Perl and gzip, system localization and keyboard configuration tables, elements needed to support *sysidcfg* options such as name services and Kerberos commands, and nothing else. A system installed with `SUNWCrnet` will have its network interfaces plumbed but no ports will be active.

The existing metaclusters are:

- `SUNWCreq` – Core System Support Software Group

- `SUNWCmreq` – Minimal Core System Support Software Group
- `SUNWCuser` – End User Software Group
- `SUNWCprog` – Developer Software Group
- `SUNWCall` – Entire Solaris Software Group
- `SUNWCXall` – Entire Solaris Software Group Plus OEM Support

Profile Keyword Changes

The JumpStart class file specifies what profile will be used for each type of system to be JumpStarted. The JumpStart profile files also specify the unique configuration of each separate type of system to be JumpStarted. There are several significant changes to the syntax of the JumpStart profile files.

The first change is the addition of the `mirror` qualifier to the `filesys` keyword. The `mirror` qualifier instructs the system to construct a mirror using the slices and sizes listed on the command line. You can specify the mirror name or the system will assign one.

The profile `filesys` keyword syntax is as follows:

```
filesys [mirror[:name]] slice slice size file_system [mount_options]
```

An example of the syntax is:

```
filesys mirror:d10 c0t0d0s0 c1t0d0s0 10240 /
```

This line in a JumpStart profile file creates a mirror named `d10` which consists of two components `c0t0d0s0` and `c1t0d0s0`. The mirror is 10 gigabytes (Gbytes) in size, and the `/` filesystem will be housed on it.

The second change is the addition of the `metadb` keyword. The `metadb` keyword allows the system administrator to specify the size and number of metastate databases that will reside on a mirrored slice.

The `metadb` keyword syntax is as follows:

```
metadb slice [size size] [count count]
```

An example of the syntax is:

```
metadb c1t0d0s0 size 8192 count 3
```

The `metadb` keyword defaults to 1 metastate database and uses a default size of 8192 blocks.

Metastate databases can only be put on slices that will be part of a mirror or the JumpStart will fail. The `mirror` qualifier automatically puts one metastate database on each slice. Use the `metadb` keyword when the system administrator wants more than one metastate database per mirror slice.

The next change is an enhancement to the `package` keyword. Previously, the `package` keyword only allowed deletion of individual packages from a cluster during installation or addition of packages that were not part of a cluster to the installation using the `add` or `delete` keywords. These packages had to be listed one at a time in the profile file. If `add` or `delete` were not explicitly specified, the keyword `add` was assumed. The package to be added had to be part of the installation media or the package installation would fail. The enhancement allows for package additions that are not part of the installation media. If you use the `package` keyword during a flash installation, you must specify the location of the package. You can now obtain the packages from the following locations: an NFS server, HTTP server, local device, or local file. The syntax for each location type differs slightly and is as follows.

A package on an NFS server can use either of the following syntax styles:

```
package SUNWname add nfs server_name:/path [retry n]
```

or

```
package SUNWname add nfs://server_name/pkg_directory_path [retry n]
```

The keyword `retry` is an optional keyword whose argument *n* is the maximum number of times the install process will attempt to mount the directory.

An example of each style entry in the profile file is:

```
package SUNWnewpkg add nfs WANBootserv:/var/spool/pkg/solaris10
```

or

```
package SUNWnewpkg add nfs://WANBootserv/var/spool/pkg/solaris10 retry 5
```

A package stored on an HTTP server must be in package stream format and use the following syntax:

```
package SUNWname add http://server_name:[port] path optional_keywords
```

The keyword `timeout` is an optional keyword that allows the system administrator to specify the maximum length of time in minutes that is allowed to pass without receipt of data from the HTTP server. If the `timeout` value is exceeded, the connection closes, reopens, and resumes. If the value is set to `0`, the connection is not reopened. A time-out reconnection causes `pkgadd` to restart from the beginning and data that was received prior to the timeout is discarded.

The keyword `proxy` is an optional keyword that allows specification of a proxy host and proxy port. A proxy host retrieves a Solaris package from beyond a firewall. When you use the `proxy` keyword, you must specify a proxy port. If you do not use the `proxy` keyword and do not specify a port, port 80 is used.

The following examples show the optional keywords:

```
package SUNWnewpkg http://WANBootserv/solaris10 timeout 5
```

or

```
package SUNWnewpkg add http://WANBootserv/solaris10 proxy localserv:8080
```

You can install a package from a local device such as a diskette or a CD-ROM. You must specify the full device pathname. If the full pathname is not specified, `/dev/dsk` is added to the pathname. If you do not specify the filesystem type, ufs is tried first and hsfs is tried second. The path is relative to the `/` (root) of the device specified. The following example shows the proper syntax:

```
package SUNWname add local_device device path file_system_type
```

For example, a package installation from the local CD-ROM with an HSFS file system uses the following entry in the JumpStart profile file:

```
package SUNWnewpkg add local_device c0t6d0s0 /solaris10/pkg
```

A package can be installed if it is part of the miniroot. The miniroot is a CD, DVD, or NFS mounted directory from which the system is booted. You can access any file that is part of the miniroot during the JumpStart installation. The following example shows the proper syntax for adding packages from a local file:

```
package SUNWname add local_file path
```

For example:

```
package SUNWnewpkg add local_file /solaris10/pkg
```

If you specify a location for a package in the profile file and do not specify a specific location for those packages following the package listed with the location, the subsequent packages are assumed to be in the same location as the first package. For example:

```
package SUNWnewpkg1 http://WANBootserv/var/spool/pkg timeout 5
package SUNWnewpkg2
package SUNWnewpkg3 http://WANBootserv1/var/spool/pkg
package SUNWnewpkg4
package SUNWnewpkg5
package SUNWnewpkg6 nfs://WANBootserv2/export
```

The use of keyword `all` in place of a package name indicates that all packages in the specified location should be added. For example:

```
package all http://WANBootserv/var/spool/pkg
```

The last change is the addition of the keyword `patch`. Before the introduction of this keyword, patches required installation with a finish script, or manual installation after the installation was completed. The patch keyword supports a list of comma separated patches or a file containing a list of patches as arguments. You can obtain patches from the following locations: an NFS server, HTTP server, local device, or local file. The syntax for each location type differs slightly and is as follows.

A patch on an NFS server should be in standard patch directory format and can use either of the following syntax styles:

```
patch <list of patch_ids or patch_order_file>
nfs://server_name/patch_directory [retry n]
```

or

```
patch <list of patch_ids or patch_order_file> nfs
server_name:/patch_directory [retry n]
```

The keyword `retry` is an optional keyword whose argument *n* represents the maximum number of times the install process will attempt to mount the directory.

An example of each style entry in the profile file is:

```
patch patch-list-file nfs://WANBootserv/solaris10/patches retry 5
```

or

```
patch 112467-01,112765-02 nfs WANBootserv:/solaris10/patches
```

A patch stored on an HTTP server must be in JAR format and use the following syntax:

```
patch <list of patch_ids or patch_order_file>
http://server_name:[port]/patch_directory [optional_http_keywords]
```

The keyword `timeout` is an optional keyword that allows the system administrator to specify the maximum length of time in minutes that is allowed to pass without receipt of data from the HTTP server. If the `timeout` value is exceeded, the connection closes, reopens, and resumes. If the value is set to `0`, the connection does not reopen. A timeout reconnection causes the patch to restart from the beginning and data that was received prior to the timeout is discarded.

The keyword `proxy` is an optional keyword to specify a proxy host and proxy port. You use a proxy host to retrieve a Solaris patch from beyond a firewall. Yuu must specify a proxy port when using the `proxy` keyword. If you do not use the `proxy` keyword and specify a port, port 80 is used. The following examples show the optional keywords:

```
patch 117428-01,113172-09 http://WANBootserv/solaris10/patches timeout 5
```

or

```
patch patch-list-file http://WANBootserv/solaris10/patches timeout 5
```

You can install a patch from a local device such as a diskette or a CD-ROM. You must specify the full device pathname. If you do not specify the full pathname, `/dev/dsk` is added to the pathname. If you do not specify the filesystem type, ufs is tried first and `hsfs` is tried second. The path is relative to the `/` (root) of the device specified. The following example shows the proper syntax:

```
patch <list of patch_ids or patch_order_file> local_device device path
file_system_type
```

For example, a patch installation from the local CD-ROM with an HSFS file system uses either of the following entries in the JumpStart profile file:

```
patch 117428-01,113172-09 local_device c0t6d0s0 /solaris10/Patches/
```

or

```
patch patch-list-file local_device c0t6d0s0 /solaris10/Patches/
```

You can install a patch if it is part of the miniroot. The miniroot is a CD, DVD or NFS mounted directory from which the system is booted. You can access any file that is part of the miniroot during the JumpStart installation. The following example shows the proper syntax for adding patches from a local file:

```
patch <list of patch_ids or patch_order_file> local_file patch_directory
```

For example:

```
patch 117428-01,113172-09 local_file /solaris10/Patches
```

or

```
patch patch_order local_file /solaris10/Patches/
```

Flash Archive Changes

The `flarcreate` command creates an archive image that you can use to reinstall the target system where the archive was created; you can also use the image to install a clone system. Archive images are commonly used with the Solaris JumpStart software. They reduce the need to write complex finish scripts to customize the configuration of the system being installed.

The `-x` option of the `flarcreate` command was originally used to exclude from the archive one directory that existed on the target system. The `-x` option was limited to excluding only one directory from the archive. Enhancements to the `-x` option and the introduction of the `-X`, `-y`, and `-z` options increase the flexibility and granularity with which archives can be created.

You can now use the `flarcreate` command to create differential archives. You must give the `flarcreate` command a source master image, which is the original flash configuration. The `flarcreate` command compares that source master image to the target master image (normally the current running system on which `flarcreate` is being executed). You can specify an alternate target master image in place of the current running environment. The source master image can come from one of the following sources:

- A Solaris Live Upgrade boot environment mounted on a directory using the `lumount`(1M) command
- A clone system mounted over NFS with root permissions
- A full Flash archive expanded into a local directory

The `flarcreate` command compares the old and the new to create the differential archive. The default behavior is to make a list of new or changed files and add these to the archive. The `flarcreate` command can also add to the list all of the files that no longer exist on the master, and which will be deleted from the clone when the differential archive is deployed. This combined list is called the *manifest*. If the clone has been manually changed after it was flash-installed from the source master image, the deployment of the differential archive fails.

Features

The `flarcreate` command has the following new options:

- `-x` *`/directory/or/file`*

  Exclude the file or directory from the archive. If you include the parent directory of a file excluded with the `-x` option, then only the file or directory specified by the exclude is excluded. Conversely, if you specify for exclusion the parent directory of an included file, then only the file specified by the include is included.

  For example, if you specify the following, all of `/dir1` except for `/dir1/file-subdirectory` is excluded:

  ```
  -x /dir1 -y /dir1/file-subdirectory
  ```

  If you specify the following, all of `/dir1` except for `/dir1/file-subdirectory` is included:

  ```
  -y /dir1 -x /dir1/file-subdirectory
  ```

- `-y` *`include`*`...`

  Include the file or directory included in the archive. See the description of the `-x` option, above, for a description of the interaction of the `-x` and `-y` options.

- `-X /dir2/filelist`

  The contents of a file named `/dir2/filelist` is a list of files to exclude from the archive. If the argument is a `–`, the list is taken from standard input.

- `-z` *`/dir2/filelist`*

  The *`filelist`* file is a list of files prefixed with a plus (`+`) or minus (`-`). A plus indicates that a file should be included in the archive; the minus indicates exclusion. If *`filelist`* is `–`, the list is taken from standard input.

- `-A /source/master/image`

  The `-A` option specifies where the source master image is located. This can be a Solaris Live Upgrade boot environment mounted on a directory using the `lumount`(1M) command, a clone system mounted over NFS with root permissions, or a full Flash archive expanded into a local directory.

- `-R /target/master/image`

The `-R` option specifies where the target master image is located if the current running system will not be used. This can be a Solaris Live Upgrade boot environment mounted on a directory using the `lumount`(1M) command, a clone system mounted over NFS with root permissions, or a full Flash archive expanded into a local directory.

- `-M`

  The `-M` option specifies that a manifest is not to be created. This enables a differential archive to be deployed without checking the contents of its manifest against the contents of the clone. This could create minor or major inconsistencies, leading to fatal OS problems.

Flash Archive Creation

To use the `flarcreate` command, you must specify the `-n` option followed by the name to be given to the archive. The `flarcreate` command syntax ends with the path name to the archive being created, as shown in the following example:

```
# flar create -n WANBootserv-archive -x /dir1 \
-y /dir1/file-subdirectory /dir3/archive-1
```

This script creates an archive of the machine where the command is run that includes the entire directory structure, with the exception of most of `/dir1`. The file or subdirectory specified with the `-y` option is included in the archive, while the rest of the directory specified with the `-x` option is not included in the archive. The file location of the archive is `/dir3/archive-1`, and the archive name is `WANBootserv-archive`.

The following example uses the contents of the `/archives/original_master` directory as the source master image, and compares that to the current system to create the differential archive:

```
# flar create -n sol_10diff -A /archives/original_master \
/export/archives/sol10_diff.flar
```

The following example uses the contents of the `/archive/original_master` directory as the source master image and compares that to the updated image `/archives/updated_master`:

```
# flar create -n sol_10diff -A /archives/original_master \
-R /archives/updated_master /export/archives/sol10_diff.flar
```

Live Upgrade Enhancements

The lucreate command is used to create a new boot environment while the system continues to run on the existing boot environment. The system can then be upgraded to the new operating system boot environment with just a reboot. Changes to the lucreate command allows you to work with a limited set of operations on logical storage devices, including creating mirrors, removing devices from mirrors, attaching devices to mirrors, and so on. These enhancements do not provide complete SDS control but provide a reasonable subset of the full features available to SDS and SVM.

Upgrade Features

The lucreate command has been modified to support enhanced storage device options for performing a live upgrade. Previous versions only worked with Solaris slice devices.

The JumpStart interface has also been enhanced to support the creation of empty boot environments during installation that are immediately available for use. The Live Upgrade startup scripts work with the new bootenv keyword that can be specified in JumpStart profiles to create these empty boot environments. The keyword bootenv is the only JumpStart profile keyword that affects Live Upgrade.

Configuration Procedure

To create a new boot environment with a metadevice, type the following:

```
# lucreate -n newbootenv -m /:/dev/md/dsk/d10:mirror -m \
/:/dev/dsk/c0t0d0s0:detach,attach,preserve
```

The -n option specifies the name of the new boot environment and the -m option specifies the device or devices to be used to hold the new boot environment. The previous command preserves the data on /dev/dsk/c0t0d0s0, detaches it from whatever mirror it is currently attached to, and then attaches it to /dev/md/dsk/d10 which is the new mirrored root device. The single filesystems component that was specified with the -m option can now be replaced with multiple devices that are configured into an SDS or SVM device.

The new lucreate keywords are: preserve, mirror, attach, and detach.

The `-z` option uses the file specified after it as a list to add or remove files and/or directories from the new boot environment. Lines in the file that start with a `+` are added to the boot filesystem and lines that start with `-` are removed from the boot filesystem.

The `-z` option also specifies which items on the command line are to be added or removed based on the `+` or `-` that is in front of the item name. The `-z` option can be used multiple times in the same command.

The following is a partial example of a JumpStart profile file that shows the use of the `bootenv createbe` JumpStart keyword sequence. This example will create a new boot environment named `newbootenv` with "`/`" on `/dev/dsk/c0t1d0s0`, `swap` on `/dev/dsk/c0t1d0s1`, and share the existing `/export` file system with the OS that was installed:

```
bootenv createbe bename newbootenv
filesystem /:/dev/dsk/c0t1d0s0:ufs
filesystem -:/dev/dsk/c0t1d0s1:swap
filesystem /export:shared:ufsh
```

The new boot environment enhancements do not take effect until the new system is first booted. The existing Live Upgrade system startup script (`/etc/init.d/lu`) is enhanced to recognize keywords passed down from the JumpStart automated Solaris OS installation procedure. If any keywords are passed down, the startup script will execute appropriate operations to satisfy the requests.

The Live Upgrade 2.0 packages are part of the Solaris OS. They are installed only for the developer and larger clusters, not for the core or end-user clusters. The system administrator is free to remove the Live Upgrade packages as there are no Solaris dependencies on the Live Upgrade packages.

WAN Boot Features

The WAN Boot procedure is an automatic installation process much like the JumpStart installation process. It provides a mechanism for automatically installing the Solaris 10 OS on multiple systems simultaneously across a wide area network.

WAN Boot includes the ability to configure each client independently, and uses unique identifying characteristics to select the correct configuration files for each client system. WAN Boot utilizes some of the existing JumpStart framework but contains enhancements to security and scalability that traditional JumpStart protocols, such as NFS, could not provide. WAN Boot supports SPARC® platform or x86 platform servers and SPARC clients. The x86 clients are not supported.

Advantages of the WAN Boot Procedure

System administrators who need to install multiple systems connected by a wide area network such as the Internet can use the WAN Boot procedure to automate the installation process. The WAN Boot process eliminates both the need for operator intervention during the installation process and the need for a JumpStart server on the same local network as the client.

The advantages of using the WAN Boot procedure include some of the same advantages as using a traditional JumpStart for installations. Advantages provided by WAN Boot include the following:

- Simplifies installations by avoiding the lengthy question-and-answer session that is part of the interactive installation process.
- Faster than interactive installations – It lets system administrators install different types of systems simultaneously.
- It allows automatic installation of the Solaris 10 OS and unbundled software.

The specific advantages of WAN Boot include:

- JumpStart boot services are not required to be on the same subnet as the installation client.
- WAN Boot client and server can authenticate using SHA hash algorithms.

- Client download of the Solaris 10 OS can be performed using HTTPS WAN Boot provides a secure, scalable process for the automated installation of systems anywhere the client and server can connect to the Internet or other WAN.

Features

WAN Boot is part of the Solaris 10 OS but works with a minimum of OpenBoot PROM (programmable read-only memory) firmware version 4.14, to support new requirements on the client. If a minimum of OpenBoot PROM revision 4.14 or later is not available, WAN Boot may be performed with a CDROM-based installation. The new firmware supports TCP/HTTP connections, SHA-1 authentication, 3DES or AES encryption, SSL v3 certificates, and several new values and command-line arguments to support these new features. These new features allow the client to contact the WAN Boot server and request the download of the new boot binary `wanboot`.

The `wanboot` download can be authenticated with an SHA-1 signature verification and encrypted with either 3DES or AES encryption. The `wanboot` program contains the information necessary to download the root file system. This information may include certificates and private keys for secure HTTP connections. New DHCP options provide support for WAN Boot clients. All WAN Boot communication occurs with HTTP or HTTPS. NFS is not used.

New features specific to the client for WAN Boot are key management, signature verification, and new OBP arguments.

WAN Boot Changes

Previously JumpStart functioned with RARP, TFTP, and NFS protocols, which do not scale for WAN use. These protocols also do not have the ability to secure the installation process.

WAN Boot utilizes advanced OBP or CDROM capabilities to scale and secure the installation process. In addition, WAN boot uses standard HTTP or HTTPS protocols, SHA-1 signatures, and 3DES or AES encryption to scale and secure the installation process in all scales of network environments including the Internet.

By using HTTP/HTTPS protocols WAN Boot requires a webserver to respond to WAN Boot client requests. Due to the nature of HTTP/HTTPS requests, Flash archives must be available to the web server. Traditional JumpStart images which performed a `pkgadd` style install over an NFS connection do not work over WAN Boot – Flash archives are the only format supported.

The new client-side `obp-tftp` package arguments are `file`, `host-ip`, `router-ip`, `subnet-mask`, `client-id`, `hostname`, `httpproxy`, `tftp-retries`, and `dhcp-retries`. The arguments are specified on the command line or listed in the `network-boot-arguments` NVRAM variable. Figure 1-1 illustrates the WAN Boot sequence and the actions taken in each step. Refer to "The WAN Boot Process" on page 1-18 for more detail about each step.

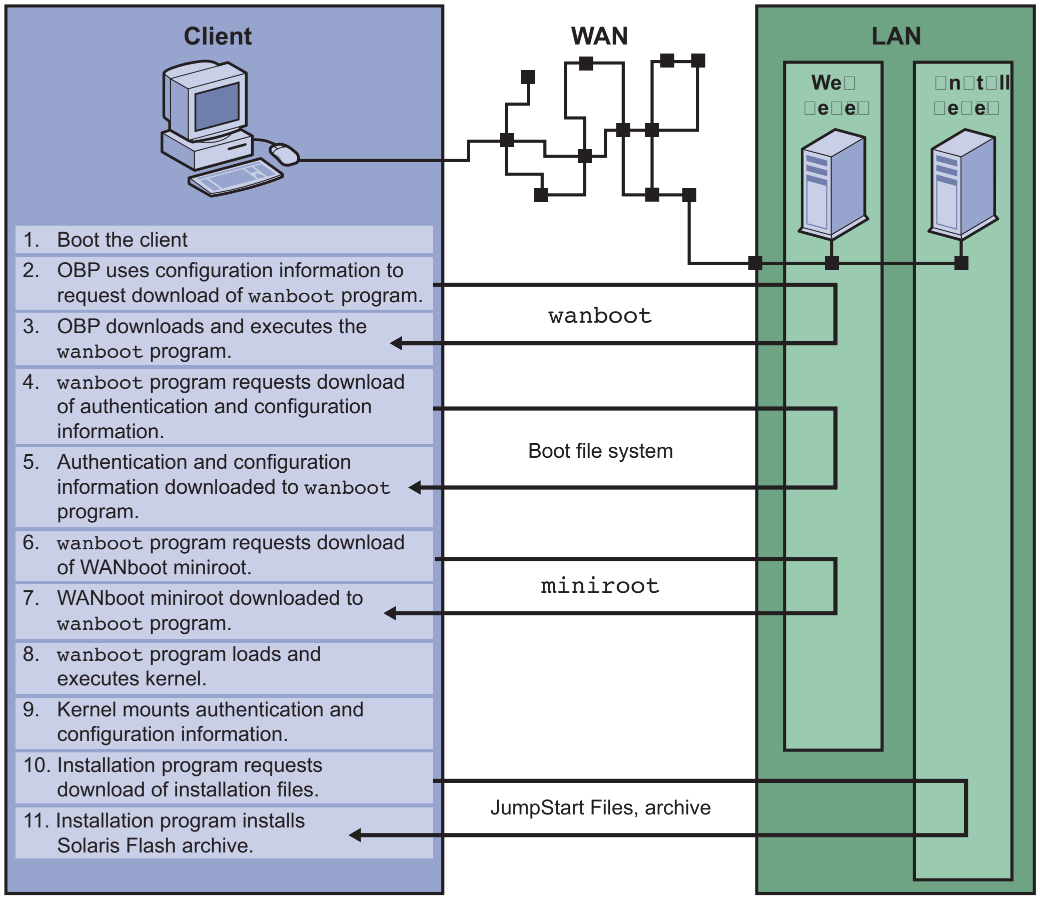

Client
WAN
LAN
1. Boot the client
2. OBP uses configuration information to request download of wanboot program.
3. OBP downloads and executes the wanboot program.
4. wanboot program requests download of authentication and configuration information.
5. Authentication and configuration information downloaded to wanboot program.
6. wanboot program requests download of WANboot miniroot.
7. WANboot miniroot downloaded to wanboot program.
8. wanboot program loads and executes kernel.
9. Kernel mounts authentication and configuration information.
10. Installation program requests download of installation files.
11. Installation program installs Solaris Flash archive.
wanboot
Boot file system
miniroot
JumpStart Files, archive

Figure 1-1 The WAN Boot Sequence

The WAN Boot Process

1. Boot the client.

   ```
   ok boot net - install
   ```

2. OBP uses configuration information to request download of `wanboot` program.

   The client's Internet protocol (IP) address and client ID are included with the request to facilitate possible client-specific downloads. The client ID is computed from the client's Media Access Control (MAC) address and is configurable.

   The download of `wanboot` may be accompanied by a Hashed Message Authentication Code (HMAC) SHA-1 signature for `wanboot` and Secure Sockets Layer (SSL) certificates for HTTP over SSL (HTTPS). Any client-specific information or security keys are obtained from the appropriate global, network, or client-specific directories under `/etc/netboot`.

   Secure Hash Algorithm 1 (SHA-1) signature keys and Triple Data Encryption Standard (3DES) or Advanced Encryption Standard (AES) encryption keys may be created and stored on the WAN Boot server for use with the client.

   The following syntax generates the keys:

```
# wanbootutil keygen -m
The master HMAC/SHA1 key has been generated
# wanbootutil keygen -c -o net=129.156.198.0,cid=010003BA152A42,type=sha1
A new client HMAC/SHA1 key has been generated
# wanbootutil keygen -c -o net=129.156.198.0,cid=010003BA152A42,type=3des
A new client 3DES key has been generated
```

The following syntax displays the keys:

```
#wanbootutil keygen -d -c -o net=129.156.198.0,cid=010003BA152A42,
type=sha1
7fb0895141ecfdff4b7425d0c9f9cf9626b395c8
# wanbootutil keygen -d -c -o net=129.156.198.0,cid=010003BA152A42,
type=3des
07df5e1907ef8a49a2b3c2cb9149fd62fb0b4cb3f440ba68
```

The keys will exist somewhere under the `/etc/netboot` directory. The `/etc/netboot` directory is hierarchical.

The global configuration data resides in `/etc/netboot` and is shared with all WAN Boot clients. Any network-specific data resides in `/etc/netboot/`*`a.b.c.d`* and is shared with all WAN Boot clients on the *`a.b.c.d`* subnet.

Any client-specific data resides in `/etc/netboot/`*`a.b.c.d/clientid`* and only applies to the client with the *`clientid`* on the *`a.b.c.d`* subnet. Client-specific files take precedence over network-specific files which take precedence over global files.

The following syntax shows an example of what might be found in the `/etc/netboot` directory:

```
# find /etc/netboot -print
/etc/netboot
/etc/netboot/keystore
/etc/netboot/129.156.198.0
/etc/netboot/129.156.198.0/keystore
/etc/netboot/129.156.198.0/010003BA152A42
/etc/netboot/129.156.198.0/010003BA152A42/keystore
```

These keys may then be stored in the client's OpenBoot PROM (OBP) or entered on the OBP command line.

The following syntax installs the keys on the client's OBP:

```
ok set-security-key wanboot-hmac-sha1
7fb0895141ecfdff4b7425d0c9f9cf9626b395c8
ok set-security-key wanboot-3des
07df5e1907ef8a49a2b3c2cb9149fd62fb0b4cb3f440ba68
```

The client is booted from the network with interface settings obtained from the OBP, the command line, Dynamic Host Configuration Protocol (DHCP), or the CDROM. Arguments specified on the command line take precedence over the OBP variable. A URL value in the file argument means OBP should execute WAN boot.

The following syntax shows setting the network parameters in the OBP:

```
ok setenv network-boot-arguments
host-ip=129.156.198.25,router-ip=129.156.198.1,subnet-mask=255.255.255.0,
hostname=WANBootclient1,file=http://145.168.198.2/cgi-bin/wanboot-cgi
```

3. OBP downloads and executes the `wanboot` program.

   The client contacts the `wanboot-cgi` program on the WAN Boot server to download the wide area network boot program, `wanboot`, from the server using Hyper Text Transfer Protocol (HTTP). The `wanboot` program is the boot filesystem. The `wanboot` binary must exist in a location under the web server's documents directory. For example:

   `/var/apache/htdocs/wanboot10/wanboot`

The client creates a virtual disk in random access memory (RAM) and writes `wanboot` to the ramdisk as it is received. If an SHA-1 signature is used, the hash is computed as data is received and if encryption is used, the client decrypts the data and rewrites it to the ramdisk.

When the download is complete, the client reads the trailing hash signature and compares it to the computed hash. The signature is all zeros if no hash has been created for `wanboot`. If the downloaded hash and the computed hash are the same, the download is assumed to be uncompromised and the `wanboot` process continues. The client then mounts the boot filesystem.

4. The `wanboot` program requests download of authentication and configuration information.

   The `wanboot` binary then parses the `wanboot.conf` file in the correct location under `/etc/netboot` to retrieve the `rootserver` and `rootpath` values. The `wanboot` program uses these values to create the HTTP/HTTPS URL for requesting the root filesystem called `miniroot`. The `wanboot` program uses the URL to request the client's root file system metadata from the `wanboot-cgi` program on the WAN Boot server.

5. Authentication and configuration information is downloaded to the `wanboot` program.

   The metadata consists of the miniroot size and hash signature. The download may be HMAC SHA-1 signed and 3DES or AES encrypted.

6. The `wanboot` program requests download of the WANBoot `miniroot.`

   The `wanboot` program uses the URL to request the client's root file system from the `wanboot-cgi` program on the WAN Boot server.

7. WANBoot `miniroot` is downloaded to the `wanboot` program.

   The `wanboot` process downloads `miniroot` from the WAN Boot server and writes it to a ramdisk. If an SHA-1 signature is used, the hash is computed as data is received. If encryption is used, the client decrypts the data and rewrites it to the ramdisk.

   When the download is complete, the client reads the trailing hash signature and compares it to the computed hash. The signature is all zeros if no hash has been created for the root filesystem. If the downloaded hash and the computed hash are the same, the download is assumed to be uncompromised and the `wanboot` process continues.

8. The `wanboot` program loads and executes the kernel.

The `wanboot` unmounts the boot filesystem and mounts the `miniroot` filesystem. The kernel from `miniroot` is then loaded into RAM and executed.

9. The installation program requests download of the installation files.

   The `system.conf` file in the appropriate location under `/etc/netboot` is included with the `miniroot` and has the locations of the JumpStart configuration files. The following example shows the entries in `system.conf`:

```
SsysidCF=https://WANBootserv/bootfiles/config
SjumpsCF=https://WANBootserv/bootfiles/config
```

   The JumpStart profile file specifies where to get the Flash archive to install on the client. The following syntax shows part of the contents of the JumpStart profile file:

```
archive_location https://WANBootserv/flashdir/solaris.flar
```

10. The installation program installs the Solaris Flash archive.

    The Flash archive is downloaded and installed on the client.

WAN Boot Server Configuration

Configuring a WAN Boot server involves three components. The components are the web server, the DHCP server, and the JumpStart server. WAN Boot requires a Solaris 10 SPARC or x86 server platform with a web server supporting at least HTTP 1.1 and also supporting HTTPS if digital certificates are used. Apache and iPlanet servers have been tested.

If HTTPS is used, the SSL must be configured. WAN Boot requires access to `wanboot`, `miniroot`, custom JumpStart files, and the Flash archive(s). These are typically stored in the web servers document root directory. It also requires access to `wanboot-cgi` and `bootlog-cgi` programs to serve CGI requests from WAN boot clients. These are typically stored in the web server's `cgi-bin` directory.

Configuring these components involves two significant problems that are beyond Sun's control and outside the scope of this module. The first problem is that even in an all-Sun installation, the administrative tools used to configure the various parts of the WAN Boot server do not communicate with each other. For example, `add_install_client` does not add macro definitions for a given client to the `dhcp_inittab`(4) file but instead creates information that the administrator must manually incorporate. A second and more difficult problem to control is the fact that heterogeneous customer environments (wherein the three services might be supplied by three or more different vendors) are very common.

Thus one finds administrative scripts that, when used, ask the administrator to perform a second action on a (possibly) different machine.

Although the steps to configure a WAN Boot server are different than setting up a JumpStart server, anyone who has configured a JumpStart server should be able to configure a WAN boot server. Reference the following URL:

`http://docs.sun.com/db/doc/817-5504`

To configure the WAN Boot server:

1. Set up the WAN Boot server as a web server with HTTP 1.1 support. Use the following URLs for information:
   - Sun Java™ System web server information:

     `http://docs.sun.com`

     `http://docs.sun.com/source/816-5683-10/contents.htm`
   - Apache web server configuration information:

     `http://httpd.apache.org/docs-project/`
2. Optionally, configure the WAN Boot server as a DHCP server. Two new vendor options support WAN Boot:
   - `SbootURI Symbol Vendor=SUNW.Sun-Blade-100 <other architectures>,16,ASCII,1,0`
   - `SHTTPproxy Symbol Vendor=SUNW.Sun-Blade-100 <other architectures>,17,ASCII,1,0`

   WAN Boot install clients are named using a network number-client ID combination that is designed to be unique (client IDs are required to be unique per network). DHCP originally used this naming scheme and it works well with the framework of WAN Boot.

3. Configure the WAN Boot server as a JumpStart server. Use the following URL:

   `http://docs.sun.com/db/doc/817-5506`

   The `wanboot` program must be copied from install media to a location under the web server's documents directory:

```
# cp /cdrom/cdrom0/s0/Solaris_10/Tools/Boot/platform/sun4u/wanboot \
/var/apache/htdocs/wanboot10/wanboot
```

   The WAN Boot miniroot filesystem must be created in a location under the web server's documents directory:

```
# /cdrom/cdrom0/s0/Solaris_10/Tools/setup_install_server -w `pwd`/wpath \
`pwd`/ipath; cp `pwd`/wpath/miniroot
/var/apache/htdocs/wanboot10/miniroot
```

   The URL paths to the `sysidcfg` file, `rules.ok` file, `profile` file, and begin and finish scripts are specified by the `SsysidCF` and `SjumpsCF` parameters in the `system.conf` file on the miniroot:

```
SsysidCF=https://WANBootserv/bootfiles/config

SjumpsCF=https://WANBootserv/bootfiles/config
```

Alternatively, you can use DHCP with the new vendor options `SbootURL` and `SHTTPproxy`. Use the `SbootURL` option to specify the location of the `wanboot-cgi` script. This option is preferable to using the standard `BootFile` option. Use the `SHTTPproxy` option to define the HTTP or HTTPS proxy if one is to be used. The `wanboot` and miniroot filesystems must each be small enough to fit into the client's RAM. WAN Boot requires the same JumpStart files needed for an NFS install, including a Solaris Flash archive, a `sysidcfg` file, a `rules.ok` file, and a `profile` file. The JumpStart files (Solaris Flash archive, `sysidcfg`, `rules.ok`, and `profile`) must be accessible to the web server. Copy these files to a location under the web server's documents directory:

```
# cp /export/config /var/apache/htdocs/wanboot10/config
```

The `archive_location` keyword in the profile should contain the URL to the Flash archive:

```
archive_location https://WANBootserv/flashdir/solaris.flar
```

The `wanboot.conf` file must be created and put in the appropriate subdirectory under `/etc/netboot`:

- the subdirectory `/etc/netboot/wanboot.conf` is global
- the subdirectory `/etc/netboot/`*`a.b.c.d`*`/wanboot.conf` is network specific

- the subdirectory `/etc/netboot/`*`a.b.c.d/clientid/`*`wanboot.conf` is client specific
- the subdirectory `/etc/inet/wanboot.conf.sample` is an example file
- the subdirectory `/usr/sbin/bootconfchk` is used to check the integrity of the `wanboot.conf` file

The `/etc/netboot` directory contains configuration information, keys, certificates, `wanboot.conf`, and `system.conf` which is used by `wanboot-cgi` to create the boot filesystem. The `/etc/netboot` directory must be created and populated by the system administrator and needs to be owned or at least readable by the web server user. The `/etc/netboot` directory is hierarchical.

The global configuration data resides in `/etc/netboot` and is shared with all WAN Boot clients. Network-specific data resides in `/etc/netboot/`*`a.b.c.d`* and is shared with all WAN Boot clients on the *`a.b.c.d`* subnet.

Client-specific data resides in `/etc/netboot/`*`a.b.c.d/clientid`* and only applies to the client with the *`clientid`* on the *`a.b.c.d`* subnet.

All of the directories can contain the following files:

- `wanboot.conf` – the client configuration file for WAN Boot installation
- `system.conf` – the configuration file specifying the location of the client's `sysidcfg` file and custom JumpStart files
- `keystore` – the file containing client SHA-1 hashing key, 3DES or AES-128 encryption key, and an optional SSL private key
- `truststore` – the file containing the digital certificates of certificate signing authorities that the client can trust
- `certstore` – the file containing the client's digital certificate. Client-specific files take precedence over network-specific files which take precedence over global files.

An example directory structure would look like the following:

```
/etc/netboot
/etc/netboot/129.156.198.0
/etc/netboot/129.156.198.0/010003BA152A42
/etc/netboot/129.156.198.0/010003BA152A42/keystore
/etc/netboot/129.156.198.0/010003BA152A42/truststore
/etc/netboot/129.156.198.0/010003BA152A42/certstore
```

```
/etc/netboot/129.156.198.0/010003BA152A42/system.conf
/etc/netboot/129.156.198.0/010003BA152A42/wanboot.conf
/etc/netboot/keystore
/etc/netboot/truststore
/etc/netboot/system.conf
/etc/netboot/wanboot.conf
```

The `wanboot.conf` file contains information used to drive the WAN Boot process. The CGI program `wanboot-cgi` uses information contained in these files to determine file paths, encryption, signing policies, and other characteristics of the operating environment. The following is a sample available at `/etc/inet/wanboot.conf.sample`:

```
#
# Copyright 2004 Sun Microsystems, Inc. All rights reserved.
# Use is subject to license terms.
#
# ident"@(#)wanboot.conf.sample1.204/01/30 SMI"
####################################################################
# wanboot.conf(4): boot configuration file.
#
# Please consult wanboot.conf(4) for further information. Note that
# this interface is "Evolving" as defined by attributes(5).
#
# Anything after a '#' is comment. Values may be quoted (e.g. "val").
#
# <empty> means there is no value, i.e. null. The absence of any
# parameter implies that it takes a default value (<empty> unless
# otherwise specified).
#
# <url> is of the form http://... or https://...
####################################################################
# The path of the bootstrap file (within htdocs) which is served up
# by wanboot-cgi(bootfile).
#
boot_file=/bootfiles/wanboot# <absolute pathname>
# These are used by wanboot-cgi(bootfile|bootfs|rootfs) to determine
# whether boot_file or the bootfs is to be sent encrypted/signed, or
# root_file is to be sent signed; the client must be setup with the
# corresponding encryption/signature key(s) (which cannot be auto-
# matically verified).
#
# If an encryption_type is specified then a signature_type must also
# be specified.
#
encryption_type=3des# 3des | aes | <empty>
```

```
signature_type=sha1# sha1 | <empty>
# This is used by wanboot-cgi(bootfs) and wanboot to determine whether
# server authentication should be requested during SSL connection
# setup.
#
server_authentication=yes# yes | no
# This is used by wanboot-cgi(bootfs) and wanboot to determine whether
# client authentication should be requested during SSL connection
# setup. If client_authentication is "yes", then server_authentication
# must also be "yes".
#
client_authentication=yes# yes | no
# wanboot-cgi(bootfs) will construct a hosts file which resolves any
# hostnames specified in any of the URLs in the wanboot.conf file,
# plus those found in certificates, etc. The following parameter
# may be used to add additional mappings to the hosts file.
#
resolve_hosts=# <hostname>[,<hostname>*] | <empty>
# This is used to specify the URL of wanboot-cgi on the server on which
# the root_file exists, and used by wanboot to obtain the root server's
# URL; wanboot substitutes root_file for the pathname part of the URL.
# If the schema is http://... then the root_file will be signed if there
# is a non-empty signature_type. If server_authentication is "yes", the
# schema must be https://...; otherwise it must be http://...
#
root_server=https://host:port/cgi-bin/wanboot-cgi# <url> | <empty>
# This is used by wanboot-cgi(rootfs) to locate the path of the
# rootfs image (within htdocs) on the root_server.
#
root_file=/rootimages/miniroot# <absolute pathname> | <empty>
# This is used by wanboot to determine the URL of the bootserver
# (and whether bootlog traffic should be sent using http or https),
# or whether it should simply be sent to the console.
#
boot_logger=# <url> | <empty>
# This is used by the system startup scripts. If set, it should
# point to a file that contains name value pairs to be used at
# start up time. For example, this file may be used to provide
# install the values for sysidcfg and jumpscfg.
#
system_conf=system.conf
```

The following keywords are supported in `wanboot.conf`:

- `boot_file` – specifies the relative web server path to the `wanboot` binary.

- `root_server` – specifies the location of the CGI program that will serve up the information about the root filesystem that will be transmitted to the client.
- `root_file` – specifies the relative web server path to the WAN Boot miniroot.
- `signature_type` – specifies the signing algorithm to be used if a signature is used when transmitting components to the client. WAN Boot currently only supports SHA-1 hash signatures.
- `encryption_type` – specifies the algorithm to use when encrypting components to be transmitted to the client. WAN Boot currently only supports 3DES and AES encryption.
- `server_authentication` – specifies whether server authentication should be requested during the SSL connection setup. If `server_authentication=yes`, then a truststore must exist.
- `client_authentication` – specifies whether client authentication should be requested during the SSL connection setup. If `client_authentication=yes`, then a certstore must exist.
- `boot_logger` – specifies the URL (if any) of a system to which logging messages will be sent.
- `system_conf` – specifies the name of a file in the `/etc/netboot` hierarchy that will be incorporated into the boot filesystem and which is intended for use by the system startup scripts. This file may be used to provide the install values for `sysidcfg` and `jumpscfg`.

To verify the integrity of `wanboot.conf,` use the `/usr/sbin/bootconfchk` command:

```
# bootconfchk /etc/netboot/129.156.198.0/010003BA152A42/wanboot.conf
```

The CGI program `/usr/lib/inet/wanboot/wanboot-cgi` fulfills client download requests for `wanboot` and the root filesystem. The `wanboot-cgi` file must be copied to the web server `cgi-bin` directory.

The CGI program `/usr/lib/inet/wanboot/bootlog-cgi` fulfills client requests for logging WAN Boot messages. It must be copied to the web server `cgi-bin` directory.

The driver `/usr/sbin/wanbootutil` serves as driver for `wanboot_keygen`(1M), `wanboot_keymgmt`(1M), and `wanboot_p12split`(1M). It is executed by the web server "owner."

The wanbootutil utility uses /usr/lib/inet/wanboot/keygen as a keyword to create and display encryption and hashing keys anywhere in the /etc/netboot hierarchy. It is only needed if the keywords encryption_type or signature_type are set to a non-NULL value in wanboot.conf. The -d option displays a key. The -m option creates a master key. The -c option creates and stores a per-client key. The supported keynames for WAN Boot are wanboot-hmac-sha1 and wanboot-3des or wanboot-aes.

Signature verification uses a HMAC SHA-1 keyed hash with matching keys on the server and client. The signature is generated if there is a nonempty value for wanboot-hmac-sha1. WAN Boot aborts if there is a signature mismatch.

The /usr/lib/inet/wanboot/keymgmt keyword is used by the wanbootutil to insert and extract raw keys directly into and from a specific keystore. Its main purpose is to insert a client's private key into a client's keystore when client authentication is configured.

The keyword wanboot_keygen is a better choice for SHA-1, 3DES, and AES keytypes. The -i option works with the -k option to insert a key into a keystore and the -x option removes it. The -s option specifies a repository in which a key will be inserted or from which a key will be extracted.

The wanbootutil utility uses /usr/lib/inet/wanboot/p12split to split PKCS #12 files into separate key and certificate entries. It creates truststore, certstore, and client private keys in the /etc/netboot hierarchy. The extracted client key must be inserted into a keystore using keymgmt.

The wanbootcgi program uses the /usr/lib/inet/wanboot/encr program to encrypt the .boot filesystem before sending it to the client.

The wanbootcgi program uses the /usr/lib/inet/wanboot/hmac program to generate HMAC SHA-1 hash signatures of components transmitted to the client.

The WAN Boot web server CGI programs must be copied to the webs erver cgi-bin directory:

```
# cp /usr/lib/inet/wanboot/*-cgi /webhome/cgi-bin/*-cgi
```

The /usr/sbin/wanbootutil binary with its specific keywords creates and maintains the SHA-1 signature and/or 3DES or AES encryption keys:

```
# wanbootutil keygen -m
The master HMAC/SHA1 key has been generated
# wanbootutil keygen -c -o net=129.156.198.0,cid=010003BA152A42,type=sha1
A new client HMAC/SHA1 key has been generated
# wanbootutil keygen -c -o net=129.156.198.0,cid=010003BA152A42,type=3des
A new client 3DES key has been generated
# find /etc/netboot -print
/etc/netboot
/etc/netboot/keystore
/etc/netboot/129.156.198.0
/etc/netboot/129.156.198.0/010003BA152A42
/etc/netboot/129.156.198.0/010003BA152A42/keystore
# wanbootutil keygen -d -c -o net=129.156.198.0,cid=010003BA152A42,
type=sha1
7fb0895141ecfdff4b7425d0c9f9cf9626b395c8
# wanbootutil keygen -d -c -o net=129.156.198.0,cid=010003BA152A42,
type=3des
07df5e1907ef8a49a2b3c2cb9149fd62fb0b4cb3f440ba68
# wanbootutil keymgmt -i -k pkey -s \
/etc/netboot/129.156.198.0/010003BA152A42/keystore -o type=rsa
The client's RSA key has been set
# wanbootutil keymgmt -x -f rsafile -s \
etc/netboot/129.156.198.0/010003BA152A42/keystore -o type=rsa
# wanbootutil p12split -i p12file -t \
/etc/netboot/129.156.198.0/010003BA152A42/truststore
# chmod 600 /etc/netboot/129.156.198.0/010003BA152A42/truststore
# wanbootutil p12split -i p12file -c \
/etc/netboot/129.156.198.0/010003BA152A42/certstore -k pkey
# chmod 600 /etc/netboot/129.156.198.0/010003BA152A42/certstore
# wanbootutil keymgmt -i -k pkey -s \
/etc/netboot/129.156.198.0/010003BA152A42/keystore -o type=rsa
```

The `wanboot-cgi` uses the `encr` program to encrypt the boot filesystem before sending it to the client:

Usage: `encr -o type=<3des|aes> -k` *`key_file`*

The `wanboot-cgi` uses the `hmac` program to generate HMAC SHA-1 hash signatures of components transmitted to the client:

Usage: `hmac [-i` *`input_file`*`] -k` *`key_file`*

WAN Boot Troubleshooting

- No OBP support for platform

  Is the `network-boot-arguments` NVRAM variable defined?
- OpenBoot PROM cannot download the boot program

  Is the `boot_file` value a URI to the CGI program?

  Did you check the web server logs?
- Boot program cannot create ramdisk

  Does the client have 256 Mbytes of RAM?
- Boot program cannot download component

  Are the values in `wanboot.conf` correct?

  Did you run `bootconfchk` on `wanboot.conf`?
- Hash mismatch reported

  Is the HMAC SHA-1 key installed on client?

  Does the client key match the client's key on the server?
- Boot filesystem (miniroot) does not execute correctly

  Is the encryption key installed on the client?

  Have you installed both 3DES and AES keys on server and client?

  Does the client key match the client's key on the server?
- Secure connection cannot be made

  Are the values in `wanboot.conf` correct?

  Did you run `bootconfchk` on `wanboot.conf`?

  Are you picking up the correct certificate(s)?

  Are the hostnames in the certificates resolvable?

Web Installation for Packages and Patches

The Web Installation for Packages and Patches feature provides the ability to sign, download, verify, and install Solaris packages and patches over the Web.

The feature allows users to deploy Solaris packages and patches through HTTP/HTTPS protocol. It also gives users the ability to digitally sign packages and patches, and verify signatures during or after download. This guarantees the authenticity and integrity of signed packages and patches.

The Web Installation for Packages and Patches feature is turned off by default. Use the new command line switches to invoke it. You can retain the previous functionality of the `patchadd` and `pkgadd` commands by not specifying the new command line switches or by explicitly using the `-n` option, which ignores signatures on packages and patches.

Configuration Procedure

Before you can add a patch or a package with the `http/https` protocol, you must set up your system with a package keystore and specify a proxy server if your system is behind a firewall.

Setting up a Package Keystore

The following example shows how to import a trusted certificate into your package keystore

1. Become the `root` user, or switch to a privileged role.
2. Export the Root CA certificate from the Java keystore into a temporary file:

```
# keytool -export -storepass changeit -alias gtecybertrustca -keystore
/usr/j2se/jre/lib/security/cacerts -file /tmp/root.crt
Certificate stored in file </tmp/root.crt>
#
```

The following table summarizes the options to the `keytool` command used in step 2.

Table 1-1 The `keytool` Command Options

Option	Description
`-export`	Exports the trusted certificate.
`-storepass` ***`storepass`***	Specifies the password that protects the integrity of the Java keystore.
`-alias` ***`gtecybertrustca`***	Identifies the alias of the trusted certificate.
`-keystore` ***`certfile`***	Specifies the name and location of the keystore file.
`-file` ***`filename`***	Identifies the file to hold the exported certificate.

3. Import the Root CA certificate into the package keystore from the temporary file.

```
# pkgadm addcert -t -f der /tmp/root.crt
    Keystore Alias: GTE CyberTrust Root
       Common Name: GTE CyberTrust Root
  Certificate Type: Trusted Certificate
Issuer Common Name: GTE CyberTrust Root
Validity Dates: <Feb 23 23:01:00 1996 GMT> - <Feb 23 23:59:00 2006 GMT>
MD5 Fingerprint: C4:D7:F0:B2:A3:C5:7D:61:67:F0:04:CD:43:D3:BA:58
SHA1 Fingerprint:
90:DE:DE:9E:4C:4E:9F:6F:D8:86:17:57:9D:D3:91:BC:65:A6:89:64

Are you sure you want to trust this certificate? y
Trusting certificate <GTE CyberTrust Root>
Type a Keystore protection Password.
Press ENTER for no protection password (not recommended): mypass
For Verification: Type a Keystore protection Password.
Press ENTER for no protection password (not recommended): mypass
Certificate(s) from </tmp/root.crt> are now trusted
#
```

The following table summarizes the options to the `pkgadm addcert` command used in step 3.

Table 1-2 The `pkgadm addcert` Command Options

Option	Description
`-t`	Indicates that the certificate is a trusted CA certificate. The command output includes the details of the certificate, which the user is asked to verify.
`-f` ***`format`***	Specifies the format of the certificates or private key. When importing a certificate, it must be encoded by using either the PEM (`pem`) or binary DER (`der`) format.
`certfile`	Specifies the file that contains the certificate.

4. Display the certificate information (optional).

```
# pkgadm listcert -P pass:mypass
Keystore Alias: GTE CyberTrust Root
Common Name: GTE CyberTrust Root
Certificate Type: Trusted Certificate
Issuer Common Name: GTE CyberTrust Root
Validity Dates: <Feb 23 23:01:00 1996 GMT> - <Feb 23 23:59:00 2006 GMT>
MD5 Fingerprint: C4:D7:F0:B2:A3:C5:7D:61:67:F0:04:CD:43:D3:BA:58
SHA1 Fingerprint:
90:DE:DE:9E:4C:4E:9F:6F:D8:86:17:57:9D:D3:91:BC:65:A6:89:64
#
```

Specifying a Proxy Server

Now that you have successfully set up your system with a package keystore, you need to specify a proxy server (if your system is behind a firewall). There are two methods for handling this: with environmental variables, or by specifying the proxy server on the command line. The following example presents both methods:

1. Specify the proxy server by using the `http_proxy`, `HTTPPROXY`, or `HTTPPROXYPORT` environment variable.

   a. Using the Korn shell (`ksh`):

```
HTTPPROXY=webcache1.central.sun.com
HTTPPROXYPORT=8080
export HTTPPROXY HTTPPROXYPORT
```

b. Using the C shell (csh):

```
setenv http_proxy http://webcache1.central.sun.com:8080
```

2. Specify the proxy server on the patchadd command line.

```
# patchadd -x mycache.domain:8080 -M
http://www.sun.com/solaris/patches/latest patch-id
#
```

Automatically Download and Add a Signed Solaris Patch

Use this procedure when you want to automatically download and add a signed Solaris OS patch in one step. This procedure assumes you have set up the package keystore and specified a proxy server with an environmental variable. To download and add a signed Solaris patch automatically, perform the following steps:

1. Identify the SunSolve℠ program URL for the patch you wish to download.
   a. Open a web browser and go to the SunSolve Online Web site:
      `http://sunsolve.Sun.COM/pub-cgi/`
      `show.pl?target=patches/patch-access`
   b. Enter the patch number you wish to download, for example, 113277-12, and press Find Patch.
   c. Place your cursor over the HTTPS link at the top of the patch page, but do not click on the link. The URL for the patch displays in the browser status line at the bottom of the screen.
2. Download and add the signed patch from the SunSolve Online Web site.

```
# patchadd "https://sunsolve.sun.com/pub-
cgi/patchDownload.pl?target=113277-12&method=hs"
(some output omitted for brevity)

Downloading patch from <https://sunsolve.sun.com/pub-
cgi/patchDownload.pl?target=113277-12&method=hs>...
..............25%.............50%.............75%.............100%

## Downloading...
## Download Complete

Verifying signed patch <113277-12>...
Verifying digital signature for signer <es-signature>
```

```
Digital signature for signer <es-signature> verified.
Verifying contents of signed patch </tmp/patchadd-dwnld/113277-12.jar>
Contents of signed patch </tmp/patchadd-dwnld/113277-12.jar> verified.
Signature on signed patch <113277-12> has been verified.
Extracting patch contents...
Checking installed patches...
Enter keystore password: mypass
Patch number 113277-12 has been successfully installed.
See /var/sadm/patch/113277-12/log for details

Patch packages installed:
  SUNWcarx
  SUNWcsr
  SUNWhea
  SUNWssad
  SUNWssadx
```

Providing Automation With the admin File

The admin file is an installation defaults file. It allows administrators to define how to proceed when the package being installed already exists on the system.

The new values added to the admin file are highlighted in the following example:

```
# cat /var/sadm/install/admin/default
(output omitted for brevity)
#
mail=
instance=unique
partial=ask
runlevel=ask
idepend=ask
rdepend=ask
space=ask
setuid=ask
conflict=ask
action=ask
basedir=default
networktimeout=60
networkretries=3
authentication=quit
keystore=/var/sadm/security
proxy=
```

Refer to the `admin`(4) man page for details and examples of possible values for the new entries.

Changes to Packages and Files

The Web Installation for Packages and Patches feature modifies some existing package and patch-related commands. The `pkgadd`, `pkgtrans`, and `patchadd` commands have been modified to allow for the new functionality and for the addition of a new command, `/usr/bin/pkgadm`.

The following table summarizes the new or modified package files, header files, and data structures implementing the Web Installation for Packages and Patches feature.

Table 1-3 Affected Packages, Files, and Data Structures

Package, File, or Structure	Filename
`SUNWcsr`	`/var/sadm/security`
`SUNWcsu`	`/usr/bin/pkgadm`
`SUNWcsu`	`/usr/sbin/pkgadd`
`SUNWcsu`	`/usr/bin/pkgtrans`
`SUNWswmt`	`/usr/sbin/patchadd`
`SUNWswmt`	`/usr/lib/patch/patchutil`

Click-Through License Panel

The Solaris Operating System has always relied on printed licenses, and *opening the box* has been acceptance of the license. Most major software vendors force the interactive acceptance of a license before proceeding with the installation of any software.

The S10 GUI SDK-based installer uses the Webstart Wizards SDK 3.0.2 license panel. The S10 CLI mode curses installer uses a new license screen.

During the installation process, you first select the installation media and verify the operating environment to be installed. When the operating environment has been initialized, the license panel appears.

You cannot complete the installation if you do not accept the license. If you decline the license, a panel gives you the option of exiting the installation. This is the only option if a license is declined.

The license panel does not appear in a Jumpstart environment. Flash archives are images of a system that has already accepted a license, therefore no license panel is necessary when performing a flash installation on a clone.

New Click-Through License Panel Features

The license text files are specific to each release of the Solaris Operating System. Within each release there is a license for each localization. All licenses are located in their own locale subdirectory in the `/webstart/os/5.10.X/license` directory on the DVD or CD miniroot. For example, the license for the Japanese locale would be in the `/webstart/os/5.10.X/license/jp` directory. The English license in the `C` subdirectory is the default if a locale does not contain its own license copy.

Figure 1-2 details the directory structure:

```
             /webstart/os
             |
---------------
             |
          5.10.X
             |
------------------------------------------------
|            |              |          |       |
meta_clusters licenses   packages   patches scripts
             |
----------------------------------------
|            |           |          |      |    |    |   |   |     |
C           fr         de       es      sv  ja   zh ko  it  zh_TW
|            |           |          |      |    |    |   |   |     |
license  license   license   license   license
```

Figure 1-2 License Directory Structure

Section II: Solaris™ 10 System Management

Objectives

Upon completion of this section, you should be able to:

- Describe the use of zones in the operating system (OS)
- Describe the authentification features in the OS
- Describe the filesystem features in the OS
- Describe the fault management feature in the OS

Module 2

Zones

Objectives

This module introduces the zones software partitioning technology, a new feature included in the Solaris™ 10 Operating System (Solaris 10 OS).

Upon completion of this module, you should be able to:

- Identify the different zones features
- Understand how and why zone partitioning is used
- Configure zones
- Install zones
- Boot zones
- Configure resource pools
- Configure CPU fair resource scheduling
- Cap physical memory resource

Solaris Zones

Solaris zones technology enables software partitioning of a Solaris 10 OS to support multiple independent operating systems with independent process space, allocated resources, and users. Zones are ideal for environments that consolidate a number of applications on a single server. The cost and complexity of managing numerous machines makes it advantageous to consolidate several applications on larger, more scalable servers. When planning to consolidate servers, there are many solutions in the marketplace. Consumers can choose from three categories of server consolidation solutions:

- Domains and Partitions - These are consolidation schemes based on hardware solutions. This includes Sun Fire™ Domains and IBM LPARs.
- Virtual Machine - This is an application-level consolidation solutions. This includes IBM VM and VMware.
- Operating System Partitions - This is an operating system-level solution. This includes FreeBSD Jails and Linux Vservers

Solaris Zones are in the Operating System Partitioning category.

Zones provide virtual operating system services that look like different Solaris instances to users and applications. This architecture isolates processes, hides the underlying platform, and enables the global administrator to allow the use of system resources on a granular level. This separation can create a more secure environment, where multiple applications that previously had to run on different physical systems can coexist, in different zones, on one machine.

Zones allow the root user of the global zone to dedicate system resources to individual zones. Each zone maintains their own root password and user information, separate from other zones and the global system. Each zone exists with separate process and file system space, and can only monitor and interact with local processes. A single processor and single disk system can support several zones, each with separate resources, users, and process space as shown in Figure 2-1.

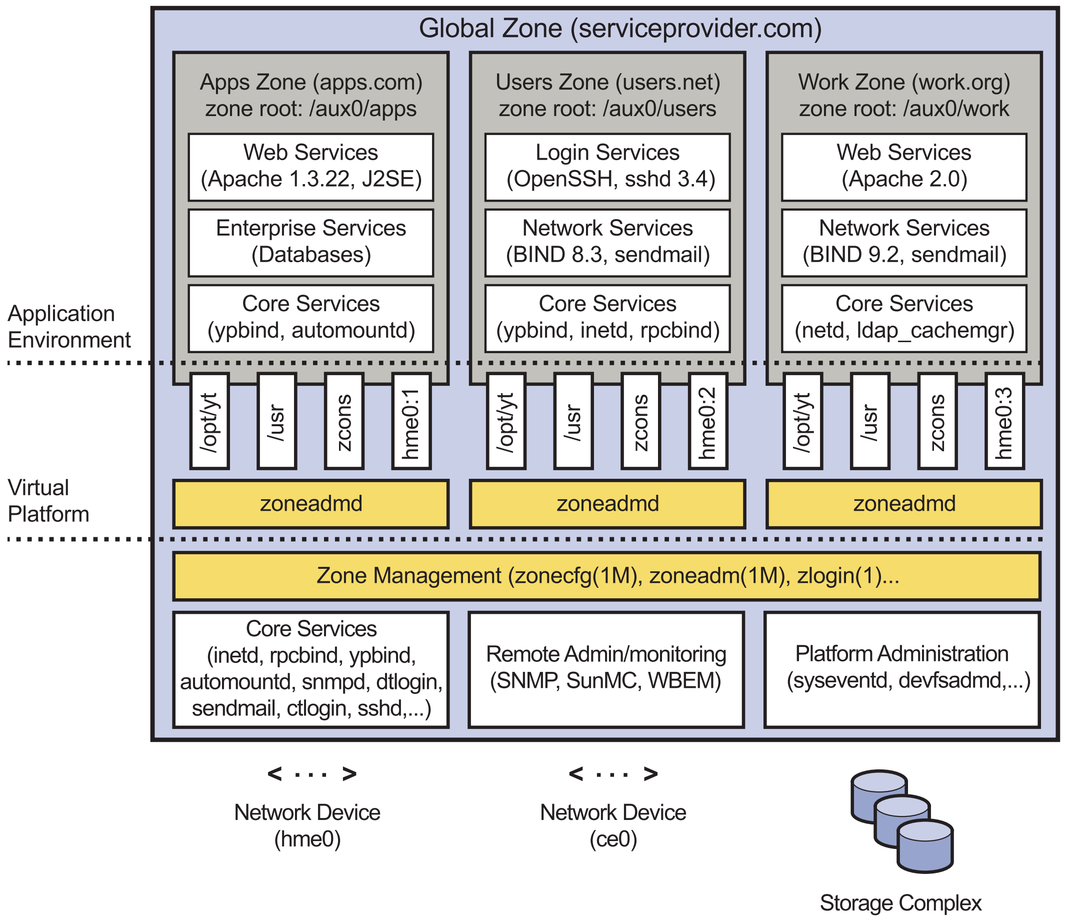

Global Zone (serviceprovider.com)
Apps Zone (apps.com) zone root: /aux0/apps
Web Services (Apache 1.3.22, J2SE)
Enterprise Services (Databases)
Core Services (ypbind, automountd)
Users Zone (users.net) zone root: /aux0/users
Login Services (OpenSSH, sshd 3.4)
Network Services (BIND 8.3, sendmail)
Core Services (ypbind, inetd, rpcbind)
Work Zone (work.org) zone root: /aux0/work
Web Services (Apache 2.0)
Network Services (BIND 9.2, sendmail)
Core Services (netd, ldap_cachemgr)
Application Environment
/opt/yt
/usr
zcons
hme0:1
hme0:2
hme0:3
Virtual Platform
zoneadmd
Zone Management (zonecfg(1M), zoneadm(1M), zlogin(1)...
Core Services (inetd, rpcbind, ypbind, automountd, snmpd, dtlogin, sendmail, ctlogin, sshd,...)
Remote Admin/monitoring (SNMP, SunMC, WBEM)
Platform Administration (syseventd, devfsadmd,...)
Network Device (hme0)
Network Device (ce0)
Storage Complex

Figure 2-1 Typical Zones Environment

Zones allow multiple Solaris instances to operate at the same time on a single hardware platform. File systems, processors, and network interfaces can be shared by multiple zones. Allotment of physical resources to more than one instance allows scaling and sharing of available resources on an as-needed basis. Individual zones can gain files and configurations from the global zone.

Zone Features

- **Security** – Network services can be run in a zone, limiting the potential damage in the event of a security violation. Processes running within a zone, even one with superuser credentials, cannot affect activity in other zones. Certain activities, such as rebooting or shutting down the system as a whole, are only permitted in the global zone. An administrator logged into the global zone can monitor the activity of applications running in other zones and control the system as a whole. The global (default) zone is perpetual.
- **Isolation** – Zones allow the deployment of multiple applications on the same machine, even if the applications operate in different trust domains, require exclusive use of a global resource, or present difficulties with global configurations. Individual zones have their own set of users and their own root password. When rebooted, any other zones running on the system are unaffected.
- **Virtualization** – Zones provide an artificial environment that can hide such details as physical devices, the system's primary internet protocol (IP) address, and host name from the application. Since the same environment can be maintained on different physical machines, this can be useful in supporting rapid deployment and redeployment of applications.
- **Granularity** – Zones can provide isolation at arbitrary granularity. A zone does not require a dedicated central processing unit (CPU), physical device, or chunk of physical memory. These resources can be multiplexed across a number of zones running within a single system, or allocated on a per-zone basis, using resource management features available in the OS.
- **Transparency** – Except when necessary to achieve security and isolation, zones avoid changing the environment in which applications execute. Zones do not present a new API or application binary interface (ABI) to which applications must be ported. Instead, they provide the standard Solaris interfaces and application environment, with some restrictions on applications attempting to perform privileged operations.

Zone Concepts

You must understand the following concepts to understand the Solaris Zones software partitioning technology:

- Zone types
- Zone daemons
- Zone file systems
- Zone networking
- Zone command scope
- Zone states

Zone Types

The Solaris Operating System supports two types of zones:

- Global zone
- Non-global zone

Global Zones

Every Solaris system contains a global zone (Figure 2-1 on page 2-3). The global zone has two functions. The global zone is both the default zone for the system and the zone used for system-wide administrative control. The global zone is the only zone from which a non-global zone can be configured, installed, managed, or uninstalled. All processes run in the global zone if no non-global zones are created by the global administrator.

Only the global zone is bootable from the system hardware. Administration of the system infrastructure, such as physical devices, routing, or dynamic reconfiguration (DR), is only possible in the global zone. Additionally, the global zone contains a complete installation of the Solaris system software packages. It can contain additional software not installed through packages.

The global zone is the only zone from which a non-global zone can be configured, installed, managed, or uninstalled. Appropriately privileged processes running in the global zone can access objects associated with other zones. Unprivileged processes in the global zone might be able to perform operations not allowed to privileged processes in a non-global zone. For example, users in the global zone can view information about every process in the system. If this capability presents a problem for your site, you can restrict access to the global zone.

The global zone provides a complete database containing information about all installed components. It holds configuration information specific to the global zone only, such as the global zone host name and file system table. The global zone is the only zone that is aware of all devices and all file systems.

Each zone, including the global zone, is assigned a zone name. The global zone always has the name *global*. Each zone is also given a unique numeric identifier, which is assigned by the system when the zone is booted. The global zone is always mapped to zone ID 0.

Non-Global Zone

The non-global zones (Figure 2-1 on page 2-3) contain an installed subset of the complete Solaris Operating System software packages. They can also contain Solaris software packages shared from the global zone and additional installed software packages not shared from the global zone. Non-global zones can contain additional software created on the non-global zone that are not installed through packages or shared from the global zone.

The non-global zones share operation under the Solaris kernel booted from the global zone. They are assigned a non-zero zone ID by the system when the zone is booted and must have a user defined name.

The non-global zone is not aware of the existence of any other zones. It cannot install, manage, or uninstall itself or any other zones.

Zone Daemons

The system uses two daemons to control zone operation: `zoneadmd` and `zsched`.

The `zoneadmd` daemon is the primary process for managing the zone's virtual platform. There is one `zoneadmd` process running for each active (ready, running, or shutting down) zone on the system.

The `zoneadmd` daemon is responsible for:

- Managing zone booting and shutting down
- Allocating the zone ID and starting the zsched system process.
- Setting zone-wide resource controls
- Preparing the zone's devices as specified in the zone configuration
- Plumbing virtual network interfaces
- Mounting loopback and conventional file systems

Unless the `zoneadmd` daemon is already running, it is automatically started by the `zoneadm` command.

Every active zone has an associated kernel process, `zsched`. The `zsched` process enables the zones subsystem to keep track of per-zone kernel threads. Kernel threads doing work on behalf of the zone are owned by `zsched`.

Zone File Systems

The Sparse Root Model installs a minimal number of files from the global zone when zones are first initialized. In this model, only certain *root* packages are installed in the non-global zone. These include a subset of the required root packages that are normally installed in the global zone, as well as any additional root packages that the global administrator might have selected. In this way, an administrator could have different versions of an operating system running concurrently on one physical system. Any files that need to be shared between a zone and the global zone can be mounted through the NFS as read-only file systems.

By default, the directories `/lib`, `/platform`, `/sbin`, and `/usr` are mounted in this manner. An example of shared file systems is shown in Figure 2-2.

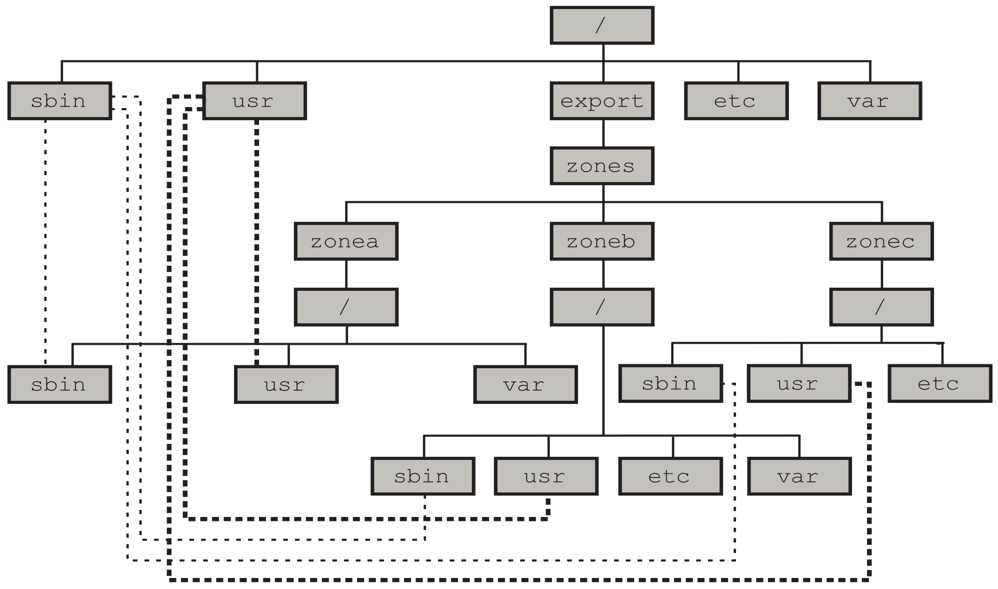


Figure 2-2 Shared File System Example

Once a zone is installed it is no longer dependent on the global zone unless a file system is mounted using NFS. If a critical file is removed from a zone, only that zone is affected. If a critical file is removed from the global zone, and the global zone operating system fails, then each zone would also fail. If the global operating system did not fail, and the zone was not in need of that removed file, the zones would be unaffected.

For files that are mounted using NFS, the removal of a critical file from the global zone would be the same as if it were in a typical client-server situation. The zone's dependence on the file would determine the effect of its removal on the zone.

Note – Currently, a non-global zone cannot be an NFS server.

After it is installed a zone behaves as if it were separate from the global zone. If an application or patch is to be added to the global zone it does not become available to zones, unless the file system the application is added too is mounted by the zones. Even if the file system is mounted, there might be libraries, header files and other necessary installed files that are not available to the zone. If an application or patch is added to a zone a similar issue may take place. A zone is not able to write back to mounted file systems, which may be necessary for the proper installation of an application or patch. Currently, an application or patch being added to the global zone requires each zone to be reinstalled.

Zone Networking

Each non-global zone that requires network connectivity has one or more dedicated IP addresses. These addresses are associated with logical network interfaces that can be placed in a zone by using the `ifconfig` command. For example, if the primary network interface in the global zone is `ce0`, then the non-global's logical network interface is `ce0:1`.

Zone interfaces configured by `zonecfg` will automatically be plumbed and placed in the zone when it is booted. The `ifconfig` command can be used to add or remove logical interfaces when the zone is running. Only the global zone administrator can modify the interface configuration and the network routes.

You can configure IPMP in the global zone, then extend the functionality to non-global zones. The functionality is extended by placing the zones IP address in an IPMP group when you configure the zone. Then, if one of the interfaces in the global zone fails, the non-global zone addresses will migrate to another network interface card.

Zone Command Scope

A single global zone acts as the underlying support for each deployed zone. Configuration changes can affect the global system, a zone, or a resource type within a zone. The level which a command affects is referred to as its *scope*. The global scope affects every zone; the resource scope only affects the zone or parameter with which you are working. For example, the `zonecfg` command prompts change to represent the current scope of a command or subcommand.

Zone States

To understand the operability of a zone we need to understand its state. Zones behave like typical Solaris 10 OS installations, but do not have resources such as power-on self-test (POST) or OpenBoot Programmable Read-only Memory (OBP). Instead, theses settings and tests are managed by the global zone. As a zone is configured, enabled, and used, its status field in the `zoneadm` ("Using the `zoneadm` Command" on page 30) command output changes. Figure 2-3 shows the zone states.

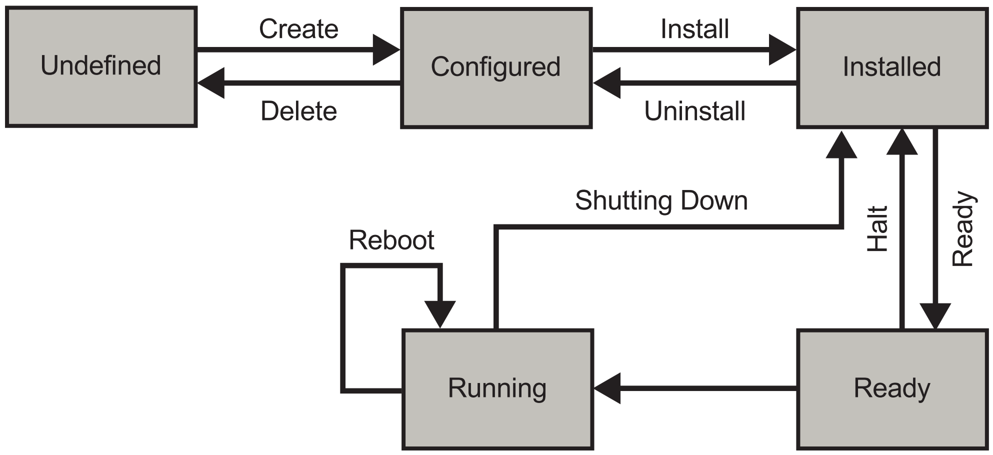


Figure 2-3 Zone States

The possible zone states are defines as:

- Undefined – In this state, the zone's configuration has not been completed and committed to stable storage. This state also occurs when a zone's configuration has been deleted.
- Configured – In this state, the zone's configuration is complete and committed to stable storage. However, those elements of the zone's application environment that must be specified after initial boot are not yet present.
- Incomplete – This is a transitional state. During an install or uninstall operation, `zoneadm` sets the state of the target zone to incomplete. Upon successful completion of the operation, the state is set to the correct state. However, a zone that is unable to complete the install process will stop in this state.

- Installed – During this state, the zones configuration is instantiated on the system. The `zoneadm` command is used to verify that the configuration can be successfully used on the designated Solaris system. Packages are installed under the zones root path. In this state, the zone has no associated virtual platform.
- Ready – In this state, the virtual platform for the zone is established. The kernel creates the `zsched` process, network interfaces are plumbed, file systems are mounted, and devices are configured. A unique zone ID is assigned by the system. At this stage, no processes associated with the zone have been started.
- Running – In this state, the user processes associated with the zone application environment are running. The zone enters the running state as soon as the first user process associated with the application environment (`init`) is created.
- Shutting – down and Down - These states are transitional states that are visible while the zone is being halted. However, a zone that is unable to shut down for any reason will stop in one of these states.

Configuring Zones

To configure a zone, you must perform these tasks:

- Identify the components that will make up the zone.
- Configure the Zone's resources.
- Configure the zone.
- Verify and commit the configured zone.

Identifying Zone Components

When planning zones for your environment, you must consider the components that make up each zones configuration. These components include:

- A zone name
- A zone path to the zone's root
- The zone network interfaces
- The file systems mounted in zones
- The configured devices in zones

Configuring Zone Resources

System-wide resources, such as CPUs, physical memory, and file systems, are shared by the zones. Each zone has unique demands on these resources. Some zones will monopolize resources starving other zones. You can stop a zone's excessive resource consumption by:

- Creating dynamic resource pools
- Establishing zone-wide CPU fair share scheduling
- Capping physical memory
- Use lofs-mounted file systems

Dynamic Resource Pools

Dynamic Resource Pools allow common systems to be placed into pools of resources. These resources can then be resized automatically by a daemon, `poold`, in response to changing conditions of resource availability and consumption.

The `poold` daemon resizes partitionable resources within constraints specified in the pools configuration file in order to achieve this goal. These resources currently are processors, but might soon include memory, disk storage, and network bandwidth.

The `poold` daemon is implemented to process arbitrary resource consumption and load statistics. The daemon responds to changing conditions by modifying the system's pool configuration. Initially, the consumption and load statistics are likely to be associated with operating system components, but the architecture supports application-published consumption and load statistics, which are part of Service Management Facility. The reassignment of resources is based on one or more documented optimization methods; these might be as simple as minimizing the weighted differences between pool loads, or as complex as a multi-resource history-based predictor model.

As more resources become available, for example in response to a dynamic reconfiguration event, then this new resource is apportioned to different resource partitions. As load on a given partition increases, resources are acquired from less-heavily loaded partitions. If none of the global objectives can be satisfied with the currently available system hardware, `poold` advertises this state to any higher-level management frameworks (if any) present by way of an API. These frameworks have the choice of modifying the objectives, removing application load, making additional hardware resources available, or doing nothing.

Resource Pools in Zones

A "one zone, one pool" rule applies to non-global zones. Processes in the global zone, however, can be bound by a sufficiently privileged process to any pool. The resource controller `poold` only runs in the global zone, where there is more than one pool for it to operate on. The `poolstat` utility run in a non-global zone displays only information about the pool associated with the zone. The `pooladm` command run without arguments in a non-global zone displays only information about the pool associated with the zone.

Configuring Resource Pools

The `poold` daemon is necessary for this feature to function in the global zone. The `poold` daemon is started during startup or by the `/etc/init.d/poold` script, a still-e*volving* interface.

In order to manipulate resource pools, you must first run the `pooladm -e` command to enable the pool facility.

Using the `poolcfg` Command

The `poolcfg` command provides configuration operations on pools and sets. These operations are performed upon an existing configuration and take the form of modifications to the specified configuration file. If you use the `-d` option, the modifications occur to the kernel state. Actual activation of the resulting configuration is achieved by way of the `pooladm` command. Table 2-1 lists the `poolcfg` options and arguments.

Table 2-1 `poolcfg` Options and Arguments

Option	Argument	Comment
`-c`	`info [entity-name]`	Display configuration (or specified portion) in human readable form to standard output. If no entity is specified, system information is displayed.
	`destroy entity-name`	Remove the specified entity.
	`modify entity-name [property-list]`	Change the listed properties on the named entity.
	`associate pool-name [resource-list]`	Connect one or more resources to a pool, or replace one or more existing connections.
	`transfer to [resourcetype] name [component-list]`	Transfer one or more discrete components to a resource.
	`transfer [quantity] from [resourcetype] [src] to [tgt]`	Transfer a resource quantity from `src` to `tgt`.
	`transfer [quantity] to [resourcetype] [tgt] from [src]`	Transfer a resource quantity to `tgt` from `src`.

Table 2-1 `poolcfg` Options and Arguments (Continued)

Option	Argument	Comment
	`discover`	Create a system entity, with one pool entity and resources to match current system configuration. All discovered resources of each resource type are recorded in the file, with the single pool referring to the default resource for each resource type.
	`rename entity-name to new-name`	Change the name of an entity on the system to its new name.
`-d`	N/A	Operate directly on the kernel state. No filename is allowed.
`-f`	`command_file`	Take the commands from command_file. Command_file consists of editing commands, one per line.
`-h`	N/A	Display extended information about the syntax of editing commands.

poolcfg Command Examples

- To view the current resource pool configuration sourcing the /etc/pooladm.conf file:

```
# poolcfg -c 'info' /etc/pooladm.conf
system O-zone
        string  system.comment Zones tester
        int     system.version 1
        boolean system.bind-default true
        int     system.poold.pid 44690

        pool pool_default
                int     pool.sys_id 0
                boolean pool.active true
                boolean pool.default true
                int     pool.importance 1
                string  pool.comment The default pool
                pset    pset_default

        pset pset_default
                int     pset.sys_id -1
                boolean pset.default true
                uint    pset.min 1
                uint    pset.max 65536
                string  pset.units population
                uint    pset.load 10
                uint    pset.size 2
                string  pset.comment 2 CPUs!
                boolean testnullchanged true

                cpu
                        int     cpu.sys_id 1
                        string  cpu.comment
                        string  cpu.status on-line

                cpu
                        int     cpu.sys_id 0
                        string  cpu.comment
                        string  cpu.status on-line
```

Table 2-2 lists the properties for this example.

Table 2-2 Pool Configuration File Properties

System Property	Value	Comment
`system.comment`	`Zones tester`	Comment string describing the system.
`system.version`	`1`	`libpool` version required to manipulate this configuration.
`system.bind-default`	`true`	If specified pool not found, bind to pool with `pool.default` property set to true. See the `pool.default` property.
`system.poold.pid`	`44690`	System-assigned process ID
`pool.sys_id`	`0`	System-assigned pool ID.
`pool.active`	`true`	Mark this pool as active, if true.
`pool.default`	`true`	Mark this pool as the default pool, if true. See `system.bind-default` property.
`pool.importance`	`1`	Relative importance of this pool; for possible resource dispute resolution.
`pool.comment`	`The default pool`	Comment string describing the resource pool.
`pset.sys_id`	`-1`	System-assigned processor set ID.
`pset.default`	`true`	Marks default processor set.
`pset.min`	`1`	Minimum number of CPUs permitted in this processor set.
`pset.max`	`65536`	Maximum number of CPUs permitted in this processor set.
`pset.units`	`population`	Identifies meaning of size-related properties; value for all processor sets is population.
`pset.load`	`10`	The load for this processor set.
`pset.size`	`2`	Current number of CPUs in this processor set.
`pset.comment`	`2 CPUs!`	User description of resource.

Table 2-2 Pool Configuration File Properties (Continued)

System Property	Value	Comment
`cpu.sys_id 0`	`0`	System-assigned CPU ID.
`cpu.comment`	`null`	User description of CPU.
`cpu.status`	`on-line`	CPU status.

- To create a processor set consisting of a maximum of 2 CPUs in the kernel:

```
# poolcfg -dc 'create pset my-1st-pset ( uint pset.min = 1 ; uint
pset.max = 2)'
```

- To make a resource pool visible to the kernel:

```
# poolcfg -dc 'create pool my-1st-pool'
```

Using the `pooladm` Command

The pooladm command provides administrative operations on pools and sets. pooladm reads the specified filename and attempts to activate the pool configuration contained in it. Before updating the current pool run-time configuration, pooladm validates the configuration for correctness. Without options, pooladm prints out the current running pools configuration.

- To enable the pool facility:

```
# pooladm -e
```

- To instantiate the configuration at `/etc/pooladm.conf`, type the following:

```
# pooladm -c
```

- To back up this configuration:

```
# pooladm -s /dev/rmt/0
```

- To remove the current configuration:

```
# pooladm -x
```

Zone-Wide CPU Fair Share Scheduling (FSS)

Currently, there is only one zone-wide resource control, `zone.cpu-shares`. Zone-wide resource controls are set through the `zonecfg` utility. The `zone.cpu-shares` resource control sets a limit on the number of FSS CPU shares for a zone. The CPU shares are first allocated to the zone, and then further subdivided among projects within the zone in accordance to the `zone.cpu-shares` parameters. The `rctl` resource type defines the `zone.cpu-shares` parameters. The parameter names are `limit`, `priv`, and `action`.

The `limit` value is local zone's percentage of CPU allocation relative to all the additional non-global zone's defined on the system.

Each `limit` value on a resource control must be associated with a privilege level (`priv`). The privilege levels are:

- `basic` – The `basic` privilege can be modified by the owner of the calling process.
- `privileged` – This privilege can be modified only by privileged (superuser) callers.
- `system` – The `system` privilege is fixed for the duration of the operating system instance.

Additionally, each limit value on a resource control must be associated with one or more actions (`action`). The actions are:

- `deny` – The `deny` action denies resource requests for the amount that is greater than the limit.
- `signal` – The `signal` action sends a specified signal to a process using resources which exceed the limit.
- `none` – The `none` action causes no action to be taken.

Note – Use the following formula to calculate a zone's actual percentage of CPU allocation:

```
CPU_allocation_% = local_zone_limit/(local_zone_limit +
other_zone_limit...) * 100
```

You can check a zone's resource controls by running the `prctl -n zone.cpu-shares -i zone` *`zone_name`* command in the global zone.

Resource Capping

The Resource Capping provides a mechanism for physical memory resource cap enforcement and administration. A resource cap is an upper bound placed on the consumption of a resource, such as physical memory. Physical memory capping is not directly supported by the `zonecfg` utility. To cap the memory resource, you must first establish a *project*.

Like the resource control, the resource cap is defined by using attributes of project entries in the project database. However, while resource controls are synchronously enforced by the kernel, resource caps are asynchronously enforced at the user level by the resource capping daemon. With asynchronous enforcement, a small delay occurs as a result of the sampling interval used by the daemon.

The system determines the existence of a project using the following sequence:

1. If the end user has an entry with a project attribute defined in the `/etc/user_attr` extended user database, then the value of the project attribute is the default project. Each entry has the form: `user:qualifier:res1:res2:`*`project_name`*

   `project_name` is the name of the project defined in the `/etc/project` file.
2. If a `project` file entry with the name `user.`*`user-id`* is present in the project database, then that project is the default project.
3. If a `project` file entry with the name `group.`*`group-id`* is present in the project database, then that project is the default project.
4. If any other project is present in the `project` file then that is the default project.

Configuring a Physical Memory Cap

To define a physical memory resource cap for a project in a non-global zone, you establish a resident set size (RSS) cap by adding this attribute to the project database entry `rcap.max-rss`.

For example, to create a new project database entry in the non-global zone with the attributes:

- Project name = regtool
- Project ID = 20

- Project group = registry
- Comment = Regtool example
- RSS cap = 1 Gbyte

You use the `projadd` and `projmod` commands:

```
# projadd -c Regtool example -G registry -p 20 regtool
# projmod -s -K rcap.max-rss=1GB regtool
```

The resulting `/etc/project` file entry for the project will be:

```
regtool:20:Regtool example::registry:recap.max-rss=1024000
```

After a project is defined, you use the `rcapadm -E` command to limit the physical memory allocated to a non-global zone.

After the project memory has been capped, you can use the `rcapstat` command to determine if the resource control is effective.

For example:

```
$ rcapstat
id  project nproc vm      rss  cap     at  avgat pg avgpg
20  regtool 3     4408K   792K 1000K   0K  0K    0K  0K
...
```

In this `rcapstat` example, the key fields to observe are:

- The `vm` field is the total virtual memory size of the project's processes, including all mapped files and devices.
- The `rss` field is the total memory resident set size of the project's processes.
- The `cap` field indicates the memory limit for the project.
- The `at`, `avgat`, `pg`, and `avgpg` fields are project disk swapping indicators.

The value of `rss` (792K) is less than the value of `cap` (1000K) and the swapping indicators show no swapping (0K). This indicates the memory cap is effective.

Allocating File System Space

There are no limits on how much disk space can be consumed by a zone. The global zone administrator is responsible for space restriction. Even a small uniprocessor system can support a number of zones running simultaneously. The nature of the packages installed in the global zone affects the space requirements of the non-global zones that are created. The number of packages and space requirements are factors.

- As a general guideline, about 100 megabytes of free disk space per non-global zone is required when the global zone has been installed with all of the standard Solaris packages.
- By default, any additional packages installed in the global zone also populate the non-global zones. The amount of disk space required must be increased accordingly. The directory location in the non-global zone for these additional packages is specified through the `inherit-pkg-dir` resource.

You can place the zone on a lofs-mounted partition. This action will limit the amount of space consumed by the zone to that of the file used by lofs. You can also use soft partitions to divide disk slices or logical volumes into partitions. You can use these partitions as zone roots, and thus limit per-zone disk consumption. The soft partition limit is 8192 partitions.

An additional 40 megabytes of RAM per zone are suggested, but not required on a machine with sufficient swap space.

Using the `zonecfg` Command

The `zonecfg` command is used to configure a zone. The `zonecfg` command can be used in interactive mode, in command-line mode, or in command-file mode. The following operations can be performed using this command:

- You can create or delete a zone configuration.
- You can add resources to a particular configuration.
- You can set properties for resources added to a configuration.
- You can remove resources from a particular configuration.
- You can query or verify a configuration.
- You can commit to a configuration.
- You can revert to a previous configuration.
- You can exit from a `zonecfg` session.

There are several subcommands to configure and provision zones within the `zonecfg` utility, as shown in Table 2-1. Several subcommands affect the environment, depending on the current scope. The `zonecfg` prompt indicates if the scope is global or resource scope. Many of the subcommands also allow the `-f`, or force, flag. If this flag is given, the subcommand does not use interactive questioning safeguards.

Table 2-3 The `zonecfg` Subcommands

Command	**Description**
`add`	Add a resource to the zone.
`cancel`	Exits from resources scope back to global. Partially specified resources are abandoned.
`commit`	Verifies settings and commits proper settings from memory to disk. The `revert` subcommand will return to this point.
`create`	Create an in-memory configuration for the specified zone.
`delete`	Delete the configuration from memory.
`end`	Verify that parameters have been assigned and return to the global scope.

Table 2-3 The `zonecfg` Subcommands (Continued)

Command	Description
`export`	Print the configuration to `stdout`, or to the output file specified, in form that can be used in a command file.
`info`	Display current configuration for resource settings or global `zonepath`, `autoboot`, or `pool`.
`remove`	Removes one or more resource depending on scope.
`select`	Find a resource whose parameters are matched within the curly braces and change to its scope.
`set`	Set an in-memory value for a parameter.
`verify`	Verify the in-memory configuration. All resources have required properties specified and the `zonepath` is specified for the zone.
`revert`	Discard any in-memory configurations and return to the last time a `commit` was performed.
`exit`	Commit current in-memory settings and exit the `zonecfg` utility. This command will automatically commit the configuration information to stable storage.

The zonecfg Resources Parameters

Resource types within the `zonecfg` utility include the following:

- `zone name` – Defines the zone name and identifies the zone to the configuration utility.
- `zonepath` – Defines the zone path resource and is the path to the zone root.

- `fs` – Assigns resource parameters for file systems. Use of the `special` parameter allows the local zone to mount global system resources under separate directories. Table 2-4 shows parameters associated with the `fs` resource.

Table 2-4 The `fs` Resource Parameters

`dir`	File system to mount from global zone
`special`	Where to make the global file system available on the zone
`type`	How zone kernel interacts with the file system
`options`	Allow parameters similar to those found with the `mount` command

- `Inherit-pkg-dir` – Gives access to software packages from the global system. The contents of software packages in the `inherit-pkg-dir` directory are inherited by the non-global zone in a read-only mode. The default `inherit-pkg-dir` resources are: `/lib`, `/platform`, `/sbin`, and `/usr`.
- `net` – Provisions logical interfaces of the global systems interfaces to non-global zones. The network interfaces are plumbed when the zone transitions from the installed state to the ready state.
- `device` – References devices for the `select`, `add`, or `remove` commands. Each zone can have devices that should be configured when the zone transitions from the installed state to the ready state.
- `rctl` – Configures those with privileges to modify and assign a resource within a zone, depending on what the project's resource allocation might be. Three possible properties are:
  - `priv`
  - `limit`
  - `action`

  Refer to the `prctl` and `rctladm` `man` pages for more information.
- `attr` – Enables the global administrator to assign generic-attribute settings, such as name type and value. The type must be `int`, `uint` (unsigned), `Boolean` or `string`.

Zone Configuration Walk-Through

To create a zone, you must log into the global system as `root` or role based access control (RBAC)-allowed user. The following shows an example of configuring a zone named `work-zone`:

```
1   global# zonecfg -z work-zone
2   zonecfg:work-zone> create
3   zonecfg:work-zone> set zonepath=/export/work-zone
4   zonecfg:work-zone> set autoboot=true
5   zonecfg:work-zone> set pool=pool_default
6   zonecfg:work-zone> add fs
7   zonecfg:work-zone:fs> set dir=/mnt
8   zonecfg:work-zone:fs> set special=/dev/dsk/c0t0d0s7
9   zonecfg:work-zone:fs> set raw=/dev/rdsk/c0t0d0s2
10  zonecfg:work-zone:fs> set type=ufs
11  zonecfg:work-zone:fs> add options [logging]
12  zonecfg:work-zone:fs> end
13  zonecfg:work-zone> add inherit-pkg-dir
14  zonecfg:work-zone:inherit-pkg-dir> set dir=/opt/sfw
15  zonecfg:work-zone:inherit-pkg-dir> end
16  zonecfg:work-zone> add net
17  zonecfg:work-zone:net> set physical=ce0
18  zonecfg:work-zone:net> set address=192.168.0.1
19  zonecfg:work-zone:net> end
20  zonecfg:work-zone> add device
21  zonecfg:work-zone:device> set match=/dev/sound/*
22  zonecfg:work-zone:device> end
23  zonecfg:work-zone> add rctl
24  zonecfg:work-zone:rctl> set name=zone.cpu-shares
25  zonecfg:work-zone:rctl> add value
    (priv=privileged,limit=20,action=none)
26  zonecfg:work-zone:rctl> end
27  zonecfg:work-zone> add attr
28  zonecfg:work-zone:attr> set name=comment
29  zonecfg:work-zone:attr> set type=string
30  zonecfg:work-zone:attr> set value="The work zone."
31  zonecfg:work-zone:attr> end
32  zonecfg:work-zone> verify
33  zonecfg:work-zone> commit
34  zonecfg:work-zone> exit
```

Line 1 - This line starts the `zonecfg` utility in interactive mode. The zone is called `work-zone`.

Line 2 - This line begins the in-memory configuration.

Line 3 - The zone path resource, `/export/work-zone` in this example, is the path to the zone root. Each zone has a path to its root directory that is relative to the global zone's root directory. This path must exist at installation time. The global zone directory is required to have restricted visibility. It must be owned by `root` with the mode `700`.

Line 4 - This boolean indicates that a zone should be booted automatically at system boot.

Line 5 - This is the name of the resource pool that this zone must be bound to when booted.

Line 6 - This line begins the file system configuration section in this procedure.

Line 7 - Set the mount point for the file system, `/mnt` in this example.

Line 8 - Specify that `/dev/dsk/c0t0d0s7` blocked special file in the global zone is to be mounted as `/mnt` in the work-zone.

Line 9 - Specify that `/dev/rdsk/c0t0d0s7` raw special file. The `zoneadmd` daemon automatically runs the `fsck` command in non-interactive check only mode on this device before it mounts the file system.

Line 10 - This line specifies that the file system type is UFS.

Line 11 - This line specifies the file system-specific option, enable file system logging in this procedure.

Line 12 - This line ends the file system configuration section in this procedure.

Line 13 - This line begins the configuration of a shared file system that is loopback-mounted from the global zone.

Line 14 - This line specifies that `/opt/sfw` is to be loopback mounted from the global zone.

Line 15 - This line ends the mount loopback section in this procedure.

Line 16 - This line begins the network configuration section in this procedure.

Line 17 - This line specifies the physical network interface to be used by this zone is a GigaSwift.

Line 18 - This line specifies the IP address for the network interface, 192.168.0.1 in this procedure.

Line 19 - This line ends the network configuration section in this procedure.

Line 20 - This line begins the device configuration section in this procedure.

Line 21 - This line sets the device match, `/dev/sound/*` in this procedure.

Line 22 - This line ends the device configuration section in this procedure.

Line 23 - This line begins the resource control configuration section in this procedure.

Line 24 - This line sets the name of the resource control, `zone.cpu-shares` in this procedure.

Line 25 - This line sets the resource control parameter values to: privileged calls, a limit of 20 CPU shares, and no action to be taken when that threshold is reached.

Line 26 - This line ends the resource control configuration section in this procedure.

Line 27 - This line begins the attribute configuration section in this procedure.

Line 28 - This line sets the name of the name of the attribute, comment in this procedure.

Line 29 - This line sets the type of attribute as a string of characters.

Line 30 - This line assigns a value to the string of characters, "The work zone." in this procedure.

Line 31 - This line ends the attribute configuration section in this procedure.

Line 32 - This line verifies the current configuration for correctness. It ensure that all resources have all of their required properties specified.

Line 33 - This line commits the current configuration from memory to stable storage. Until the in-memory configuration is committed, changes can be removed with the `revert` subcommand. A configuration must be committed to be used by the `zoneadm` command. This operation is attempted automatically when you complete a `zonecfg` session. Because only a correct configuration can be committed, the commit operation automatically does a verify.

Line 34 - This line exits the `zonecfg` session. You can use the `-F` (force) option with exit.

The zone is now ready to install, boot, and use.

Viewing the Zone Configuration File

When you commit the zone configuration to stable storage, the file is stored in the `/etc/zones` directory in XML format. For example:

```
# more /etc/zones/work-zone.xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE zone PUBLIC "-//Sun Microsystems Inc//DTD Zones//EN"
"file:///usr/share/lib/xml/dtd/zo
necfg.dtd.1">
<zone name="work-zone" zonepath="/export/work-zone" autoboot="true">
....
```

Using the zoneadm Command

The `zoneadm` command is the primary tool used to install and administer non-global zones. Operations using the `zoneadm` command must be run from the global zone. The following tasks can be performed using the `zoneadm` command:

- Verify a zone's configuration
- Install a zone
- Boot a zone
- Reboot a zone
- Display information about a running zone
- Uninstall a zone
- Remove a zone using the `zonecfg` command

Verifying a Configured Zone

You can verify a zone prior to installing it. If you skip this procedure, the verification is performed automatically when you install the zone. You must be the global administrator in the global zone to perform this procedure.

You use the `zoneadm -z` *`zone_name`* `verify` command to verify a zone's configuration. For example:

```
global# zoneadm -z work-zone verify
Warning: /export/work-zone does not exist, so it cannot be verified. When
zoneadm install is run, install will try to create /export/work-zone, and
verify will be tried again, but the verify may fail if: the parent
directory of /export/work-zone is group- or other-writable or
/export/work-zone overlaps with any other installed zones.
```

In this example, a message is displayed warning the administrator that the `zonepath` does not exist. This illustrates the type of messages output by the `zoneadm` command.

If no error messages are displayed, you can install the zone.

Installing a Configured Zone

You use the `zoneadm -z` *`zone_name`* `install` command to perform installation tasks for a non-global zone. You must be the global administrator to perform the zone installation. For example:

```
global# zoneadm -z work-zone install
```

You use the `zoneadm list -iv` command to list the installed zones and verify the status:

```
global# zoneadm list -iv
ID  NAME       STATE     PATH
0   global     running   /
-   work-zone  installed /export/work-zone
```

In this example, the `work-zone` has reached the installed state. The zone ID will be assigned during the zone boot process.

Booting a Zone

Booting a zone places the zone in the running state. If you set the `autoboot` resource property in a zones configuration to `true`, that zone is automatically booted when the global zone is booted. The default setting is `false`.

A zone can be manually booted from the ready state or from the installed state. You use the `zoneadm -z` *`zone_name`* `boot` command to boot a zone:

```
global# zoneadm -z work-zone boot
global# zoneadm list -v
ID  NAME      STATE    PATH
0   global    running  /
1   work-zone running  /export/work-zone
```

In this example, the `work-zone` has reached the running state. The zone ID 1 has been assigned during the zone boot process.

Halting a Zone

The `zoneadm halt` command is used to remove both the application environment and the virtual platform for a zone. The zone is then brought back to the installed state. All processes are killed, devices are unconfigured, network interfaces are unplumbed, file systems are unmounted, and the kernel data structures are destroyed.

```
global# zoneadm -z work-zone halt
global# zoneadm list -v
ID  NAME      STATE      PATH
0   global    running    /
-   work-zone installed  /export/work-zone
```

The `halt` command does not run any shutdown scripts within the zone.

Rebooting a Zone

The `zoneadm reboot` command is used to reboot a zone. The zone is halted and then booted again.

```
global# zoneadm -z work-zone reboot
global# zoneadm list -v
ID  NAME      STATE     PATH
0   global    running   /
2   work-zone running   /export/work-zone
```

In this example, before rebooting the zone ID is set to 1. After the zone is rebooted, the zone ID has changed to 2.

Logging Into and Working With the Zone

Use the `zlogin` command to log in to and access the deployed zone from the global zone. Be aware that `root` users are not allowed to log in by default. To log into the zone as if you were on its console use the `-C` option.

```
#  zlogin -C work-zone
[Connected to zone 'work-zone' console]
```

You are asked to provide a terminal type, host name, time zone, and `root` password. After you enter the appropriate information, you see the following output:

```
System identification is completed.
rebooting system due to change(s) in /etc/default/init
[NOTICE: zone rebooting]
SunOS Release 5.10 Version s10_54 64-bit
Copyright 1983-2004 Sun Microsystems, Inc.  All rights reserved.
Use is subject to license terms.
Hostname: twilight
The system is coming up.  Please wait.
starting rpc services: rpcbind done.
syslog service starting.
Creating new rsa public/private host key pair
Creating new dsa public/private host key pair
The system is ready.
twilight console login: root
Password:
Apr 30 12:37:07 twilight login: ROOT LOGIN /dev/console
Sun Microsystems Inc.   SunOS 5.10      s10_54  May 2004
```

After using the console interface to log into the zone, take a look at how the operating system views its resources.

```
twilight# hostname
twilight
twilight# uname -a
SunOS twilight 5.10 s10_54 sun4u sparc SUNW,Netra-T12
twilight# df -k
File system           kbytes    used   avail capacity  Mounted on
/                     678457   69941  547455    12%    /
/dev                  678457   69941  547455    12%    /dev
/lib                33265565 1893804 31039106   6%    /lib
/platform           33265565 1893804 31039106   6%    /platform
/sbin               33265565 1893804 31039106   6%    /sbin
/usr                33265565 1893804 31039106   6%    /usr
proc                       0       0       0     0%    /proc
mnttab                     0       0       0     0%    /etc/mnttab
fd                         0       0       0     0%    /dev/fd
swap                 7949040      32 7949008     1%    /var/run
swap                 7949008       0 7949008     0%    /tmp
twilight# ps -ef |grep z
     UID   PID  PPID   C    STIME TTY         TIME CMD
    root  6965  6965   0 12:35:38 ?           0:00 zsched
```

```
twilight# ifconfig -a
lo0:1: flags=1000849<UP,LOOPBACK,RUNNING,MULTICAST,IPv4> mtu 8232 index 1
inet 127.0.0.1 netmask ff000000
ce0:1: flags=1000843<UP,BROADCAST,RUNNING,MULTICAST,IPv4> mtu 1500 index
2 inet 192.168.0.1 netmask ffffff00 broadcast 192.168.0.255

twilight# ~.
[Connection to zone 'work-zone' console closed]
```

Note – The zone is now up and running. If you add (or delete) resources to the running zone using the `zonecfg` command, you must restart the zone for the changes to take effect.

To remove a resource from a domain, run the `zonecfg` command and choose the `remove` subcommand with a reference to the device and parameters.

```
#  zonecfg –z work-zone
zonecfg:work-zone> remove net physical=ce0
zonecfg:work-zone> commit
zonecfg:work-zone> exit
```

Deleting a Zone

When deleting a zone, be sure to back up any files that you want to keep. The first stage in deleting a zone is halting the Solaris 10 OS and freeing the system memory.

In the following example, the zone is removed from the global system:

Caution – This operation is not a graceful or controlled shutdown of the zone. Data loss is possible to processes running in the zone.

```
#  zoneadm list -cp
0:global:running:/
3:work-zone:running:/export/work-zone
#  zoneadm -z work-zone halt
#  zoneadm list -cp
0:global:running:/
-:work-zone:installed:/zones/work-zone
```

At this point, the zone is not using system resources other than file system space. Uninstall the zone to remove the zone's file usage.

```
#  zoneadm -z work-zone uninstall
Are you sure you want to uninstall zone work-zone (y/[n])? y
#  zoneadm list -cp
0:global:running:/
-:work-zone:configured:/export/work-zone
```

The final step is to delete the configuration of the zone from the global system with the `delete` subcommand.

```
#  zonecfg -z work-zone delete
Are you sure you want to delete zone work-zone (y/[n])? y
#  zoneadm list -cp
0:global:running:/
```

Module 3

Authentication Changes

Objectives

This module is an overview of the authentication feature changes included in the Solaris™ 10 Operating System (Solaris 10 OS). Upon completion of this module, you should be able to:

- Identify changes to Password Checking
- Identify features of Least Privilege for Solaris
- Identify changes to Kerberos
- Identify changes to Sun Java™ System Web Server 6.1 2004Q2 reserved UID/GID
- Identify changes to nobody account usage

Password Checking

Two updates have been made to how Solaris 10 OS checks passwords. The first is to support prior password history checking. The second is to change the maximum number of characters in a password from eight to a possible 256.

Features

The updated password checking features have the following new features:

Password History – Accounts defined in name services which support password history checking (presently files only), have a password history maintained of up to 26 prior changed passwords. Each user's 26 prior passwords are kept so that an administrator may increase the number to check at any time up to 26, and have the new policy immediately enforced. The default is zero. Setting the `HISTORY` value to zero causes all users' password history to be discarded at the next password change by any user.

If `HISTORY` is non-zero in the `/etc/default/passwd` file `/etc/security/passhistory` is created to record the user's previous 26 encrypted passwords, regardless of the value of `HISTORY`. Should `HISTORY` be set to zero in `/etc/default/passwd` this file is removed.

The prior passwords retained are the `crypt`(3C) form as they would be in the `/etc/shadow` file. The `pam_authtok_check`(5) command has been modified to check the user's password history for a reuse of a prior password. Root is still exempt from password quality checks. The update is synchronized by the existing global locking done during every local account password update.

Maximum Number of Characters Allowed – The need to have greater password flexibility has led to the changing of the `PASS_MAX` variable from eight to 256. This variable change takes place in the `limits.h` header file. The `getconf` and `getpass` commands have been updated to interact with this change.

Changed Packages and Files

The following is a list of files that have changed for password history:

- `/etc/default/passwd`
- `/etc/security/passhistory`
- `pam_authtok_check`

The `/etc/default/passwd` file has been updated to allow name checking and history. The updates to the file follow:

```
# NAMECHECK enables/disables login name checking.
# The default is to do login name checking.
# Specifying a value of "NO" will disable login name checking.

#NAMECHECK=NO

# HISTORY sets the number of prior password changes to keep and
# check for a user when changing passwords. Setting the HISTORY
# value to zero (0), or removing/commenting out the flag will
# cause all users' prior password history to be discarded at the
# next password change by any user. No password history will
# be checked if the flag is not present or has zero value.
# The maximum value of HISTORY is 26.
#
# This flag is only enforced for user accounts defined in the
# local passwd(4)/shadow(4) files.
#HISTORY=0
```

The `/etc/security/passhistory` is an auto-generated, colon-delimited file. It has the following syntax:

`user``:`*`cryptedpw1`*`:`*`cryptedpw2`*`: ….`

The `getconf` command has been updated to remain consistent with XPG. The following has been added to the command:

- `/usr/xpg4/bin/getconf [-v` *`specification`*`]` *`system_var`*
- `/usr/xpg4/bin/getconf [-v` *`specification`*`]` *`path_var pathname`*
- `/usr/xpg4/bin/getconf -a`

The `limits.h` header file has been updated to reflect a maximum number of characters and now has this updated line:

```
PASS_MAX 256 /* max # of characters in a pass-word */
```

Least Privilege Feature

Root user privilege allows a user or process to act as root in every area, even when that access is not necessary. To limit the misuse of privilege the Solaris 10 OS allows for the implementation of Least Privilege. Least Privilege restricts processes to the privileges required to perform the necessary task and no more.

Least Privilege adds support for fine-grained process privileges and allows processes to perform certain privileged operations, such as binding to a reserved port, without having full root privileges. It also allows the powers of root processes to be restricted to a subset of those normally available. Previously, access to privileged operations through devices was restricted by the permission bits on the device node.

The similarities between privileges and the authorizations introduced by the Role Based Access Control project [2] prompted some discussion. The project team explained that privileges represent basic system privileges enforced by the kernel, while authorizations represent higher-level capabilities defined by applications.

Taken together, all defined privileges with the exception of the basic privileges compose the set of privileges that are traditionally associated with the root user. The basic privileges are a subset of the root user privileges.

Features

The traditional UNIX® privilege model associates all privileges with the effective UID 0. The basic flaw of that model is its all-or-nothing approach. An application which needs a single special privilege, such as a web server binding to the reserved port 80, a program running in the real-time scheduling class, a server to keep the clock synchronized, the NFS server, all need to run or start as root. This traditional approach has a number of shortcomings:

- It is not possible to restrict a process to a limited set of privileged operations.
- Each privileged process has complete reign of the system; all vulnerable privileged processes can be leveraged to full access to the system.

- It is not possible to extend an ordinary user's capabilities with a restricted set of privileges.
- It is often unclear exactly what privileged functionality a process requires access to.

Privileges and Privilege Sets

The various privileged operations inside the kernel are grouped under appropriate privileges.

The process model is extended with privilege sets each containing zero or more privileges. The following are allowed privilege sets:

- Effective set which contains the privileges that are currently in effect.
- Permitted set which contains privileges that can be made effective.
- Inheritable set which is made effective at `exec`(2).
- Limit set which is an upper bound on all future effective sets for a process and its offspring.

System-wide Implementation

Least Privilege is a system-wide implementation with new system and library calls to examine and change the privilege sets. The existing user `attr`(4), `prof attr`(4) and `exec attr`(4) and associated utilities have been extended to support privileges for RBAC profiles and users. New utilities are provided to inspect and manipulate process privileges, assign privileges required to open devices, and define additional privileges.

Least Privilege delivers the kernel framework for maintaining and propagating privileges throughout the kernel. Those sections of the Solaris kernel where privilege checks are performed (`suser`(), `drv_priv`(), `cr_uid == 0`) are changed and new checks, against the effective privilege set, are performed instead. The Least Privilege feature adds a number of privilege sets to the Solaris kernel credential structure. All the kernel's security policy decisions based on checks against uid 0 are changed to checks against the relevant privileges.

Process privileges get their own entry under `/proc/pid`, as the `/proc/pid/cred` format does not allow for extensions.

The `user_attr` database changes to allow users to log in with certain privileges and assume them through the `su` command. The `exec_attr` database is extended to allow privileges to be associated with executables in profiles. Some new privileges are defined for operations that are currently unprivileged. This allows for a more restricted type of user.

Compatibility is maintained by awarding effective uid 0 processes all privileges except when a process or one of its ancestors changes it to be *privilege aware*. Processes without an effective uid of 0 always honor their possibly non-empty effective privilege sets.

There are certain defined privileges, basic or otherwise. Table 3-1 is a listing of the defined privileges and their definitions.

Table 3-1 Defined Privileges

Privilege	**Definition**
`PRIV_FILE_CHOWN`	Allow a process to change a file's owner user ID. Allow a process to change a file's group ID to one other than the effective group ID or one of the supplemental group IDs of the process.
`PRIV_FILE_CHOWN_SELF`	Allow a process to give away its files. A process with this privilege will run as if {`_POSIX_CHOWN_RESTRICTED`} is not in effect.
`PRIV_FILE_DAC_EXECUTE`	Allow a process to execute an executable file whose permission bits or ACL would otherwise disallow the process execute permission.
`PRIV_FILE_DAC_READ`	Allow a process to read a file or directory whose permission bits or ACL would otherwise disallow the process read permission.
`PRIV_FILE_DAC_SEARCH`	Allow a process to search a directory whose permission bits or ACL would not otherwise allow the process search permission.
`PRIV_FILE_DAC_WRITE`	Allow a process to write a file or directory whose permission bits or ACL do not allow the process write permission. All privileges are required to write files owned by UID 0 in the absence of an effective UID of 0.
`PRIV_FILE_LINK_ANY`	Allow a process to create hard links to files owned by a UID different from the process' effective UID.

Table 3-1 Defined Privileges (Continued)

Privilege	**Definition**
`PRIV_FILE_OWNER`	Allow a process that is not the owner of a file to modify that file's access and modification times. Allow a process that is not the owner of a directory to modify that directory's access and modification times. Allow a process that is not the owner of a file or directory to remove or rename a file or directory whose parent directory has the “save text image after execution” (sticky) bit set. Allow a process that is not the owner of a file to mount a namefs upon that file. (Does not apply to setting access permission bits or ACLs.)
`PRIV_FILE_SETDAC`	Allow a process that is not the owner of a file or directory to modify that file's or directory's permission bits or ACL.
`PRIV_FILE_SETID`	Allow a process to change the ownership of a file or write to a file without the set-user-ID and set-group-ID bits being cleared. Allow a process to set the set-group-ID bit on a file or directory whose group is not the process' effective group or one of the process' supplemental groups. Allow a process to set the set-user-ID bit on a file with different ownership in the presence of `PRIV_FILE_SETDAC`. Additional restrictions apply when creating or modifying a setuid 0 file.
`PRIV_IPC_DAC_READ`	Allow a process to read a System V IPC Message Queue, Semaphore Set, or Shared Memory Segment whose permission bits would not otherwise allow the process read permission.
`PRIV_IPC_DAC_WRITE`	Allow a process to write a System V IPC Message Queue, Semaphore Set, or Shared Memory Segment whose permission bits would not otherwise allow the process write permission.
`PRIV_IPC_OWNER`	Allow a process that is not the owner of a System V IPC Message Queue, Semaphore Set, or Shared Memory Segment to remove, change ownership of, or change permission bits of the Message Queue, Semaphore Set, or Shared Memory Segment.
`PRIV_NET_ICMPACCESS`	Allow a process to send and receive ICMP packets.

Table 3-1 Defined Privileges (Continued)

Privilege	**Definition**
`PRIV_NET_PRIVADDR`	Allow a process to bind to a privileged port number. The privilege port numbers are 1-1023 (the traditional UNIX privileged ports) as well as those ports marked as `udp/tcp_extra_priv_ports` with the exception of the ports reserved for use by NFS.
`PRIV_NET_RAWACCESS`	Allow a process to have direct access to the network layer.
`PRIV_PROC_CHROOT`	Allow a process to change its root directory.
`PRIV_PROC_CLOCK_HIGHRES`	Allow a process to use high resolution timers.
`PRIV_PROC_AUDIT`	Allow a process to generate audit records. Allow a process to get its own audit preselection information.
`PRIV_PROC_EXEC`	Allow a process to call `execve`(2).
`PRIV_PROC_FORK`	Allow a process to call `fork`(2), `fork1`(2), or `vfork`(2).
`PRIV_PROC_LOCK_MEMORY`	Allow a process to lock pages in physical memory.
`PRIV_PROC_OWNER`	Allow a process to send signals to other processes and inspect and modify the process state in other processes, regardless of ownership. When modifying another process, additional restrictions apply: the effective privilege set of the attaching process must be a superset of the target process' effective, permitted, and inheritable sets; the limit set must be a superset of the target's limit set; if the target process has any UID set to 0, all privilege must be asserted unless the effective UID is 0. Allow a process to bind arbitrary processes to CPUs.
`PRIV_PROC_PRIOCNTL`	Allow a process to elevate its priority above its current level. Allow a process to change its scheduling class to any scheduling class, including the RT class.
`PRIV_PROC_SESSION`	Allow a process to send signals or trace processes outside its session.
`PRIV_PROC_SETID`	Allow a process to set its UIDs at will, assuming that UID 0 requires all privileges to be asserted.
`PRIV_PROC_TASKID`	Allow a process to assign a new task ID to the calling process.

Table 3-1 Defined Privileges (Continued)

Privilege	**Definition**
`PRIV_PROC_ZONE`	Allow a process to trace or send signals to processes in other zones. See `zones`(5).
`PRIV_SYS_ACCT`	Allow a process to enable and disable and manage accounting through `acct`(2).
`PRIV_SYS_ADMIN`	Allow a process to perform system administration tasks such as setting node and domain name and specifying `coreadm`(1M) and `nscd`(1M) settings
`PRIV_SYS_AUDIT`	Allow a process to start the (kernel) audit daemon. Allow a process to view and set audit state (audit user ID, audit terminal ID, audit sessions ID, audit preselection mask). Allow a process to turn off and on auditing. Allow a process to configure the audit parameters (cache and queue sizes, event to class mappings, and policy options).
`PRIV_SYS_CONFIG`	Allow a process to perform various system configuration tasks. Allow file system-specific administrative procedures, such as file system configuration `ioctls`, creation and deletion of snapshots, and manipulating the PCFS boot sector.
`PRIV_SYS_DEVICES`	Allow a process to create device special files. Allow a process to open the real console device directly.
`PRIV_SYS_IPC_CONFIG`	Allow a process to increase the size of a System V IPC Message Queue buffer.
`PRIV_SYS_LINKDIR`	Allow a process to link and unlink directories.
`PRIV_SYS_MOUNT`	Allow a process to mount and unmount file systems that would otherwise be restricted (that is, most file systems except namefs). Allow a process to add and remove swap devices.
`PRIV_SYS_NET_CONFIG`	Allow a process to configure a system's network interfaces and routes. Allow a process to configure network parameters using ndd. Allow a process access to otherwise restricted information using `ndd`.
`PRIV_SYS_NFS`	Allow a process to provide NFS service: start NFS kernel threads, perform NFS locking operations, bind to NFS reserved ports: ports 2049 (`nfs`) and port 4045 (`lockd`).

Table 3-1 Defined Privileges (Continued)

Privilege	Definition
PRIV_SYS_RES_CONFIG	Allow a process to create and delete processor sets, assign CPUs to processor sets and override the PSET_NOESCAPE property. Allow a process to change the operational status of CPUs in the system using p_online(2). Allow a process to configure file system quotas. Allow a process to configure resource pools and bind to them.
PRIV_SYS_RESOURCE	Allow a process to exceeds the resource limits imposed on it by setrlimit(2) and setrctl(2).
PRIV_SYS_SUSER_COMPAT	Allow a process to successfully call a third party loadable module that calls the kernel suser() function to check for allowed access. This privilege exists only for third party loadable module compatibility and is not used by Solaris proper.
PRIV_SYS_TIME	Allow a process to manipulate system time using any of the appropriate system calls: stime(2), adjtime(2), and ntp_adjtime(2).

Of the privileges listed above the following are considered *basic* privileges. These privileges were formerly always available to unprivileged processes. By default, processes still have the following basic privileges:

- PRIV_FILE_LINK_ANY
- PRIV_PROC_SESSION
- PRIV_PROC_FORK
- PRIV_PROC_EXEC

The privileges PRIV_PROC_SETID and PRIV_PROC_AUDIT must be present in the Limit set of a process in order for setuid root execs to be successful, that is, get an effective UID of 0 and additional privileges

Configuration Procedures

There is no global setting to enable or disable this feature. Each process may have a unique manner to become privilege aware. Developers can use system calls to set or change privileges for programs. To interact with privileges use the `ppriv` command.

The `ppriv` command can inspect or modify process privilege sets and attributes. The options and flags to `ppriv` are listed below with a definition of each.

```
/usr/bin/ppriv -e [-D| -N] [-s spec] command [arg…]
```

This shows the `ppriv` command along with the specified privilege sets and flags modified according to the arguments on the command line.

```
/usr/bin/ppriv [-v] [-D| -N] [-s spec] [pid | core]…
```

This examines or changes the privilege state of running process and core files.

```
/usr/bin/ppriv -l [-v] [privilege…]
```

This lists the privileges defined and information about specified privileges.

```
/usr/bin/ppriv -l zone
```

This lists the privileges available in the current zone. When run in the global zone, all defined privileges are listed.

The following options are supported:

- `-D` – Turns on privilege debugging for the processes or command supplied.
- `-e` – Interprets the remainder of the arguments as a command line and runs the command line with specified privilege attributes and sets.
- `-l` – Lists all currently defined privileges on stdout.
- `-N` – Turns off privilege debugging for the processes or command supplied.
- `-s` *`spec`* – Modifies a process' privilege sets according to spec, a specification with the format `[AEILP][+-=]`*`privsetspec`*, containing no spaces, where:

    - AEILP – Indicates one or more letters indicating which privilege sets to change. These are case insensitive; for example, either a or A indicates all privilege sets, E is effective, I is inheritable, L is limit and P is the permitted privilege set.
    - +-= – Indicates a modifier to respectively add (+), remove (-), or assign (=) the listed privileges to the specified set(s) in privsetspec.
    - privsetspec – Indicates a comma-separated privilege set specification (priv1,priv2, and so on), as described in priv_str_to_set(3C). Modifying the same set with multiple -s options is possible as long as there is either precisely one assignment to an individual set or any number of additions and removals. That is, assignment and addition or removal for one set are mutually exclusive.
- -v – Verbose. Reports privilege sets using privilege names.
- -z – Lists all privileges available in the current zone on stdout.

For example, to view current privilege sets run the ppriv command and pass it the variable *$$*, which represents the current shell. In the output below we see the effective, inheritable and permitted privilege sets are set to basic and the limit set is currently at all, which allows for any privilege.

```
$ ppriv $$
1157:bash
flags = 0x0
E: basic
I: basic
P: basic
L: all
```

Now remove the ability for the shell to send signals or trace processes outside of this session by removing the E, effective and I, inheritable privileges from the PROC_SESSION privilege set. In the example below the privilege set is preceded by a bang symbol which shows the privilege is not allowed.

```
$ ppriv -s EI-proc_session $$
$ ppriv $$
1157:bash
flags = 0x0
E: basic,!proc_session
I: basic,!proc_session
P: basic
L: all
```

In order to see how the loss of this privilege affects the shell, we show what happens when we try to add back this privilege set. In the example below we see that we are unable to add back the privilege set we just removed.

```
$ ppriv -s EI+proc_session $$
ppriv: cannot examine 1157: permission denied
```

Kerberos Feature

This feature adds Kerberos functionality to the existing Solaris remote application clients and servers. The existing remote services to be affected are: `rlogin/rlogind`, `telnet/telnetd`, `ftp/ftpd`, and `rsh/rcp/rdist/rshd`. Currently, these services are offered as part of Solaris without any Kerberos functionality. Kerberos-enabled versions of these applications can be added by installing the unbundled Solaris Enterprise Authentication Mechanism (SEAM) package from sun.com.

The new features to Kerberos in Solaris 10 include updates to remote access commands, scripts for initializing Kerberos clients and updating client and DNS servers to provide Kerberos-realm passwords and information.

Features

This project involves adding Kerberos authentication and privacy support to the following programs and modules:

- `/usr/bin/telnet`
- `/usr/sbin/in.telnetd`
- `/usr/kernel/strmod/$(ARCH)/telmod`
- `/usr/bin/rlogin`
- `/usr/sbin/in.rlogind`
- `/usr/kernel/strmod/$(ARCH)/rlmod`
- `/usr/bin/ftp`
- `/usr/sbin/in.ftpd`
- `/usr/bin/rsh`
- `/usr/bin/rcp`
- `/usr/bin/rdist`
- `/usr/sbin/in.rshd`
- `/usr/bin/login`

Exposed Interfaces

Table 3-2 show the exported interfaces with this change:

Table 3-2 Kerberos Exported Interfaces

Interface	Classification	Comments
`/usr/sbin/in.ftpd -K`	Evolving	new cmd line flag
`/usr/sbin/in.rlogind –keMSci`	Evolving	new cmd line flags
`/usr/sbin/in.rshd -keci`	Evolving	new cmd line flags
`/usr/sbin/in.telnetd -aeX`	Evolving	new cmd line flags
`/usr/bin/ftp -m`	Evolving	new cmd line flag
`/usr/bin/ftp cmds`	Evolving	new client commands
`/usr/bin/rcp -xk`	Evolving	new cmd line flags
`/usr/bin/rlogin -fFxk`	Evolving	new cmd line flags
`/usr/bin/rsh -fFxk`	Evolving	new cmd line flags
`/usr/bin/telnet -akfFKxX`	Evolving	new cmd line args
`/usr/bin/telnet cmds`	Evolving	new client commands
`/usr/bin/login`	Consolidation Private	new cmd line flags
`/usr/bin/rdist –xk`	Evolving	new cmd line flags
`$HOME/.k5login`	Evolving	per user access control file
`/usr/kernel/strmod/crmod/ strmod/sparcv9/crmod`	Consolidation Private	Generic streams crypto modules

Table 3-3 show the imported interfaces with this change:

Table 3-3 Kerberos Imported Interfaces

Interface	Classification	Comments
`mech_krb5.so.1`	Consolidation Private	SARC/1996/060
Kerberos Security Protocol	Standard	RFC 1510

Table 3-3 Kerberos Imported Interfaces (Continued)

Interface	**Classification**	**Comments**
ftp GSS-API security protocol	Standard	RFC 2743
ftp security extensions	Standard	RFC 2228
r* KRB5 security	Standard	RFC 1510
telnet authentication option	Standard	RFC 2941
telnet authentication (KRB5)	Standard	RFC 2942
telnet data encryption option	Standard	RFC 2946
telnet encryption – DES CFB	Standard	RFC 2952
telnet encryption – DES OFB	Standard	RFC 2953
Kernel Crypto. Subsystem	Evolving	PSARC 2001/553
PAM REPOSITORY pam_item	Consolidation Private	PSARC 2001/255

Configuration Procedures

Each of the interfaces may have a new flag to take advantage of Kerberos functionality. Many of these commands are evolving. Refer to the command man page for current option usage. Below is a table of updated command options and their descriptions.

The `telnet` Command

The following options are new to the `telnet` command:

- `-a` – Force attempt of automatic login.
- `-k` *`realm`* – Kerberos authentication. If Kerberos authentication is used, request that `telnet` obtain tickets for the remote host in realm instead of the remote host's.
- `-F` – Forward a copy of the local credentials (KRB5 TGT) to the remote system.
- `-f` – Forward a copy of the local credentials (Kerberos TGT) to the remote system.
- `-K` – Specify NO automatic login.
- `-x` – Turn on encryption of the data stream.

- `-X` *`atype`* – Disable *`atype`* of authentication.
- `-S` *`tos`* – Set the IP TOS option.

New `telnet` commands:

- `auth` *`argument`* – The `auth` command manipulates the information sent through the TELNET AUTHENTICATE option. Valid arguments for the `auth` command are as follows:
- `disable` *`type`* – Disables the specified type of authentication. To obtain a list of available types, use the `auth disable ?` command.
- `enable type` – Enables the specified type of authentication. To obtain a list of available types, use the `auth enable ?` command.
- `status` – Displays status of available authentication types. Note that all the available authentication types may not be currently negotiated.
- `encrypt argument` – The `encrypt` command manipulates the information sent through the `TELNET ENCRYPT` option. Valid arguments for the `encrypt` command are as follows:
  - `disable` *`type`* `[input|output]` – Disables the specified type of encryption. If you omit the `input` and `output` parameters, both input and output are disabled. To obtain a list of available types, use the `encrypt disable ?` command.
  - `enable` *`type`* `[input|output]` – Enables the specified type of encryption. If you omit the `input` and `output` parameters, both input and output are enabled. To obtain a list of available types, use the `encrypt enable ?` command.
  - `input` – Same as the `encrypt start input` command.
  - `-input` – Same as the `encrypt stop input` command.
  - `output` – Same as the `encrypt start output` command.
  - `-output` – Same as the `encrypt stop output` command.
  - `start [input|output]` – Attempts to start encryption. If you omit `input` and `output`, both input and output are enabled. To obtain a list of available types, use the `encrypt enable ?` command.
- `status` – Lists the current status of encryption.
- `stop [input|output]` – Stops encryption. If you omit `input` and `output`, encryption is stopped on both input and output.
- `type` *`type`* – Sets the default type of encryption to be used with later `encrypt start` and `encrypt stop` commands.

- `toggle` – Toggle between `TRUE` and `FALSE` the various flags that control how `telnet` responds to events. The `toggle` command is a currently supported `telnet` command, but this project introduces additional parameters that may be toggled with this command.

  New `toggle` arguments:

  - `authdebug` – Turns on debugging information for the authentication code.
  - `autoencrypt`, `autodecrypt` – When the `TELNET ENCRYPT` option is negotiated, by default the actual encryption (decryption) of the data stream does not start automatically. The `autoencrypt (autodecrypt)` command states that encryption of the output (input) stream should be enabled as soon as possible.
  - `autologin` – This command is the same as specifying the `-a` option on the `open` command. By default, autologin is configured to be `TRUE` in the `krb5.conf` file.
  - `Encdebug` – Turns on debugging information for the encryption code.
  - `verbose_encrypt` – When the `verbose_encrypt` flag is `TRUE`, `telnet` prints out a message each time encryption is enabled or disabled. The initial value for this toggle is `FALSE`.

The `rlogin` Command

The following options are new to the `rlogin` command:

- `-f,F,x,k` *`realm`* – Same as for `telnet` (see above).
- `-A` – Require Kerberos V5 authentication. Without this option, `rlogin` operates in normal mode. When used, the remote server must accept Kerberos credentials or the connection is terminated.
- `-D` *`port`* – Connect to an alternate *`port`* on the remote server.

The `rsh` Command

The following options are new to the `rsh` command:

- `-f,F, a,-k` *`realm`* – same as for `telnet` (see above)
- `-D` *`port`* – Connect to alternate *`port`* on remote

The `rcp` Command

The following options are new to the `rcp` command:

- `-x`, `-k` *`realm`* – Same as for `telnet` (see above)
- `-a` – Same as for `rlogin` (see above)
- `-D` *`port`* – Connect to alternate *`port`* on remote

The `rdist` Command

The following options are new to the `rdist` command:

- `-x`, `-k` *`realm`* – Same as for `telnet` (see above)
- `-a` – Same as for `rlogin` (see above)
- `-d` *`port`* – Connect to alternate *`port`* on remote

The `ftp` Command

The following options are new to the `ftp` command:

- `-m` mech – Specify the GSS-API mechanism to use (defaults to "kerberos_v5), as listed in the `/etc/gss/mech` table.
- `-f` – Forward credentials to remote host
- `-a` – Use GSSAPI authentication ONLY, if this fails, close the connection.
- `-x` – Attempt to negotiate encryption and set initial protection level to `private`
- `-k` – Realm same as for `telnet` (see above)

New `ftp` commands:

- `mechanism` **`name_of_mechanism`** – Specify the GSS-API mechanism to use (for example, `kerberos_v5`).
- `protect` *`protection_level`* – Set the protection level on data transfers to the desired protection level. The valid protection levels are:
  - `clear` – For unprotected data transmission. If no ADAT command succeeded, then the only possible level is `clear`. If no level is specified, the current level is printed. The default protection level is `clear`.
  - `safe` – For data transmission integrity protected by cryptographic checksum.
  - `private` – For data transmission confidentiality and integrity protected by encryption.

The `login` Command

The following variables are new to the `login` command:

- `-u` *`identity`* – The `rlogind` and `telnetd` commands fork logins using this argument to tell login the krb5 principal name that was authenticated. The `login` command then uses it to set the `PAM_REPOSITORY` item for the `pam_krb5` module to check. This prevents `pam_krb5` from prompting for a password and re-authenticating the user who has already presented valid Kerberos V5 credentials to the service daemon.
- `-s` *`service`* – Indicates the service name that should be used when calling `pam_start`.
- `-R` *`repository`* – Name of the `PAM_REPOSITORY` to use to insert the *`identity`* information. This is only valid with the `-u` option .

Daemon Configuration Changes

Following are configuration changes to daemons in order to interact with these new features. Below is a listing of the daemons and the possible configuration changes to take advantage of these new features.

The `in.telnetd` Daemon

The following configuration changes are integrated into the `in.telnetd` daemon:

- `-a` *`mode`* – Specifies one of the following modes to be used for authentication:
  - `debug` – Turns on authentication debug
  - `user` – Only allow connections when the remote user provides valid authentication information, and allows access to the specified account without providing a password.
  - `valid` – Only allow connections when the remote user can provide valid authentication information
  - `none` – The default state; authentication information not required
  - `off` – Authentication code is disabled; all authentication occurs through the `pam`(3) framework
- `-e` – Enable encryption debugging support
- `-X` – Disable authentication negotiation

- `-E` – Disable encryption negotiation
- `-U` – Refuse connections that cannot be mapped to a name using 'gethostbyname'
- `-S` *`tos`* – Set the IP TOS option
- `-h` – Disable displaying host specific info before login has been completed
- `-t` *`keytab`* – Set the keytab file to use
- `-M` *`realm`* – Use the indicated Kerberos realm
- `-R` *`realm`* – Same as `-M` (for MIT compatibility)

The `in.rshd` Daemon

The following configuration changes are integrated into the `in.rshd` daemon:

- `-k` – Require Kerberos authentication.
- `-5` – Same as `-k` (for MIT compatibility).
- `-e` – Require an encrypted session.
- `-c` – Checksum. Require Kerberos clients to present a cryptographic checksum of initial connection information (such as the name of the user that the client is trying to access in the initial authenticator).
- `-I` – Ignore authenticator checksums.
- `-A` – Do not allocate a reserved port for the `stderr` connection.
- `-U` – Refuse connections that cannot be mapped to a name using `gethostbyname.`
- `-L` *`env_var`* – Environment variables to save.
- `-S` *`keytab`* – Set the keytab file to use.
- `-s` *`tos`* – Set the IP TOS option.
- `-M` *`realm`* – Use the indicated Kerberos realm.

The `in.rlogind Daemon`

The following configuration changes are integrated into the `in.rlogind` daemon:

- `-k` – Require Kerberos authentication.
- `-5` – Same as `-k` (for MIT compatibility).
- `-e` – Allow an encrypted session.

- `-x` – Allow an encrypted session (MIT compatible).
- `-E` – Allow an encrypted session (MIT compatible).
- `-X` – Allow an encrypted session (MIT compatible+).
- `-c` – Checksum. Require Kerberos clients to present a cryptographic checksum of initial connection information like the name of the user that the client is trying to access in the initial authenticator.
- `-I` – Ignore authenticator checksums.
- `-P` – Prompt for a password in addition to other authentication checks.
- `-p` – Prompt for a password only if the other authentication checks fail.
- `-s` *`tos`* – Set the IP TOS option.
- `-S` *`keytab`* – Set the keytab file to use.
- `-M` *`realm`* – Use the indicated Kerberos realm.
- `-D` *`port`* – Listen on indicated port (used for debugging in standalone mode).
- `-f` – fork daemon when new connection is accepted (useful for debugging in standalone mode).

The `in.ftpd` Daemon

The following configuration changes are integrated into the `in.ftpd` daemon:

- `-K` – Only permit GSSAPI authenticated connections.
- `-C` – Non-anonymous users must have credentials prompt for passwd if credentials were not forwarded in the authentication exchange.

All of the affected clients and servers (except `ftp` and `ftpd`) depend on the `crmod` streams module interface to perform cryptographic functions. This module is delivered as part of this project. The `crmod` module depends heavily on having an in-kernel cryptographic API which supports the required DES crypto modes needed by these applications. If the kernel crypto API project is delayed or not delivered, this project will need to include support for these DES modes.

Setting Up a Client Kerberos Machine

Kerberos can be configured during installation, with the `kclient` script and also through DNS. To change these setting or configure Kerberos after installation use the `kclient` command. The `kclient` command configures a machine as a Kerberos client for a realm, adding the Kerberos host principal to the local keytab file. Use a Kerberos-configured NFS or retrieve a master `krb5.conf` file.

```
/usr/sbin/kclient [ -n ] [ -R realm ] [ -k kdc ] [ -a
adminuser ] [ -c filepath ] [ -p profile ]
```

Mode of Operation

The `kclient` command works in interactive or non-interactive mode. In the non-interactive mode, the user feeds in the required inputs by means of a profile and/or command line options. Should a value not be found in non-interactive mode the user is prompted for required parameter values such as *`realm`*, *`kdc`* and *`adminuser`*. The interactive mode is invoked when the utility is run without any command line arguments. This command always adds the host/Fully Qualified Domain Name (FQDN) entry to the local host's `keytab` file.

Each mode also requires the user to enter the password for the administrative user requested. This obtains the Kerberos Ticket Granting Ticket (TGT) for the adminuser. The host/FQDN, NFS/FQDN and root/FQDN principals are added first to the Key Distribution Center (KDC) database, if not already present, before their addition to the local host's keytab.

The `kclient` utility assumes that the local host has been set up for DNS and requires the presence of a valid `resolv.conf` in order to work properly.

The non-interactive mode supports the following operands:

- -n – Set up the machine for Kerberos-configured NFS. This involves making changes to `nfssec.conf`(4) and addition of the NFS/FQDN and root/FQDN entries to the local host’s keytab file.
- `-R` *`realm`* – Specifies the Kerberos realm.
- `-k` *`kdc`* – Specifies the machine to be used as the Kerberos KDC.

- `-a` *`adminuser`* – Specifies the Kerberos administrative user to be used for addition of principals to the Kerberos database and/or default keytab.
- `-c` *`filepath`* – Specifies the pathname to the krb5.conf(4) master file, to be copied over to the local host.
- `-d` *`dnsarg`* – Specifies the DNS lookup option to be used and specified in `/etc/krb/krb5.conf`. Valid entries are:
  - `none`
  - `dns_lookup_kdc`
  - `dns_lookup_realm`
  - `dns_fallback`
- `-p` *`profile`* – Specifies the profile to be used, in order to read in the values of all the parameters required for setting up the machine as a Kerberos client.

  The profile should have entries in the format:

  `PARAM` *`value`*

  Valid `PARAM` entries are:
  - `REALM`
  - `KDC`
  - `ADMIN`
  - `FILEPATH`
  - `NFS`

  These profile entries correspond to the `-R` *`realm`*, `-k` *`kdc`*, `-a` *`adminuser`*, `-c` *`filepath`* and `-n` command line options respectively. Any other `PARAM` entry is considered invalid and is ignored.

  The NFS profile entry can have a value of 0 (do nothing) or 1 (operation is requested). Any other value is considered invalid and is ignored. It should be noted that the command line options override the `PARAM` values listed in the profile.

Example 1 – Setting up a Kerberos Client Using Command Line Options

To set up a Kerberos client using the clntconfig/admin administrative principal for realm ABC.COM, KDC example1.com and which also performs Kerberos-configured NFS.

```
# /usr/sbin/kclient -n -R ABC.COM -k example1.com -a clntconfig
```

Note – The krb5 administrative principal used by the administrator only needs to have add, inquire, change-pwd and modify privileges (for the principals in the KDC database) in order, for the `kclient` utility to run.

The file `kadm5.acl` entry would be:

```
clntconfig/admin@ABC.COM acmi
```

Example 2 – Setting up a Kerberos Client Using the Profile Option

To set up a Kerberos client using the clntconfig/admin administrative principal for realm ABC.COM, KDC remote_server and which also copies over the master `krb5.conf` from a specified location.

```
# /usr/sbin/kclient -p /net/remote_server/export/profile.krb5
```

Contents of profile.krb5 file would be:

```
REALM ABC.COM
KDC example1
ADMIN clntconfig
FILEPATH /net/remote_server/export/krb5.conf
NFS 0
```

Automated Privilege

A user may create a file called `.k5login` in the home directory. This file contains a list of Kerberos principals that are allowed to access that user's account (with proper credentials) without being prompted for a password.

For example, if user Tim, with a home directory of `/export/home/tim` wanted to allow Joe access to his account without Tim's password he would create the following file and entry:

```
/export/home/tim/.k5login
Joe@SUN.COM
```

This allows Joe to assume another Tim's identity without having to know his password. The impersonating user must have valid credentials issued from his realm. For security reasons, the remote user should own this file.

Changed Packages and Files

Table 3-4 lists the packages affected by the Kerberos change:

Table 3-4 Kerberos Changed Packages and Files

Program	Packages
/bin/login	SUNWcsu
/usr/bin/telnet	SUNWtnetc
/usr/sbin/in.telnetd	SUNWtnetd
/usr/bin/rlogin	SUNWrcmdc
/usr/sbin/in.rlogind	SUNWrcmds
/usr/bin/ftp	SUNWcsu
/usr/sbin/in.ftpd	SUNWcsu
/usr/bin/rsh	SUNWrcmdc
/usr/bin/rcp	SUNWrcmdc
/usr/bin/rdist	SUNWrcmdc
/usr/bin/in.rshd	SUNWrcmds
/usr/lib/security/pam_krb5.so	SUNWkrbu[x]
/usr/kernel/strmod/crmod	SUNWk5pk
/usr/kernel/strmod/sparcv9/crmod	SUNWk5pkx
/usr/kernel/strmod/telmod	SUNWcsu
/usr/kernel/strmod/sparcv9/telmod	SUNWcsxu
/usr/kernel/strmod/rlmod	SUNWcsu

Table 3-4 Kerberos Changed Packages and Files (Continued)

Program	**Packages**
`/usr/kernel/strmod/sparcv9/rlmod`	`SUNWcsxu`

The following files under the `/etc/krb5` directory are also affected:

- `/etc/krb5/kadm5.acl` – Kerberos access control list (ACL) file
- `/etc/krb5/krb5.conf` – Default location for the local host's configuration file
- `/etc/krb5/krb5.keytab` – Default location for the local host's keytab file
- `/etc/nfssec.conf` – File listing NFS security modes
- `/etc/resolv.conf` – DNS resolver configuration file

Frequently Asked Questions

Question: Does it use a naming service such as NIS, NIS+ or LDAP?

Answer: *The server applications such as* `telnetd`, `rlogind`, `ftpd` *and* `rshd` *may use a naming service if the system is so configured.*

Question: What are the ongoing maintenance requirements of these new features (that is, keeping global tables up-to-date, and trimming files)?

Answer: *In order to support the Kerberos functionality being addressed by this project, a Kerberos infrastructure (KDCs and various configuration files) must be in place.*

Question: Does this new interface make access or privilege decisions that are not audited? Does it introduce removable media support that is not managed by the allocated subsystem? Does it provide administration mechanisms that are not audited?

Answer: *The Kerberos authentication mechanism is not currently audited. There is a plan in the current MRP to add auditing to Kerberos. This project will not make any changes to the auditing features currently in place in the applications that this project is modifying.*

Module 4

File System Features

Objectives

This module is an overview of the file system features included in the Solaris™ 10 Operating System (Solaris 10 OS). Upon completion of this module, you should be able to:

- Identify features of the ZFS project
- Identify changes to UFS

ZFS Project

ZFS, the Zetabyte File System, is a new production file system under development at Sun Microsystems. Existing local file systems and volume managers still suffer from many problems, such as the inability to detect data corruption, hit-or-miss on-disk consistency, complex administration, and limited scalability. ZFS addresses these problems using a new file system architecture, which includes virtually-addressed pooled storage, checksums of all disk data, integration of the volume manager into the file system, and an object-based copy-on-write transaction model.

Administrators no longer need to partition disks or create volumes; disk corruption is automatically detected and corrected; file systems are as easy to create as directories; on disk consistency is a provable property of the design; and ZFS is designed to scale to zetabytes of storage and beyond. A zetabyte = 70-bit (a billion Tbytes). ZFS is fully POSIX-compliant; existing applications receive the full benefit of ZFS technology.

What Is it?

ZFS is a file system for the future, with numerous features to handle today's storage needs and allow for future needs. The ZFS architecture is based on three organizing principles:

- All storage is pooled. Pooled storage means that disk space is shared among many file systems. In addition to reducing fragmentation, this also simplifies administration because the data's logical structure is not constrained by physical device boundaries. In the same way that directories can be freely created and deleted within a traditional file system, ZFS file systems can be freely created and deleted within a storage pool. An example of pooled storage can be found in `tmpfs`. All `tmpfs` mounts consume space from the operating system's global swap pool. There is no "raw device" for a `tmpfs` mount; rather, `tmpfs` file systems are named only by their mount points, that is, their logical names.

- ZFS is object-based, modular, and extensible. The ZFS POSIX Layer (ZPL) translates incoming vnode ops into operations that read or write a storage object, which is a linearly-addressable set of bytes. The ZPL then sends these object read/write requests to the Data Management Unit (DMU), which translates <object, offset> directly to physical data location. There is no need for a volume manager, and all the overhead of the logical block layer is eliminated. ZFS stores everything in DMU objects: user data, znodes, directories, DMU, and SPA metadata. The entire storage pool can be viewed as a giant tree with leaf nodes containing data and interior nodes containing metadata. The root of the tree is a single disk block called the uberblock. To commit a transaction group, the DMU first writes out all the modified leaf nodes, then the interior nodes that point to them, and so on up the tree. Finally the SPA issues a single write that atomically changes the uberblock to refer to the new tree. All data and all metadata are stored in DMU objects, and all DMU objects are subordinate to the uberblock, so the entire storage pool is always self-consistent.
- All operations are copy-on-write, transactional, and have a checksum. The SPA provides a powerful checksum system to ensure data integrity. Each interior node in the tree described above contains an array of block pointers that describes its children. Each block pointer contains the child's Data Virtual Address (DVA), birthday (the transaction group in which it was born), and 64-bit checksum.

ZFS Features

The ZFS file system is made up of a series of components which work together to provide its advancements and features.

- ZPL – Provides standard POSIX semantics, for example permission checking, file modes, time stamps, and so on. The ZPL translates incoming `vnode` and `vfs ops` into operations that read or write a storage object, which is a linearly addressable set of bytes (also known as a sparse array or flat file).
- ZFS Directory Service (ZDS) – Provides concurrent, constant-time directory services (create, delete, lookup) for the ZPL.
- DMU – Provides transactions, data caching and object translation (described in Section 4.3).
- Storage Pool Allocator (SPA) – Provides space allocation, replication, checksums, compression, encryption, resource controls, and fault management.

Snapshots

The copy-on-write (COW) transaction model enables constant-time snapshots. ZFS can snapshot the entire storage pool by simply making a copy of the uberblock. All subsequent writes cause new blocks to be allocated (because everything is COW), so the blocks that comprise the snapshot are never modified. Similarly, ZFS can take a snapshot of any subtree of the storage pool (for example, a single file system) by making a copy of its root block.

Checksums and Data Integrity

The SPA detects and corrects any of the five major classes of error:

1. Bit rot (media errors) – When the checksum fails on read, the SPA reads from another replica and repairs the damaged one.
2. Misdirected read (disk firmware reads the wrong block) – The SPA behaves as in (1), but the first replica is not actually damaged, so the "repair" step is unnecessary (but harmless).

3. Phantom write (disk claims it wrote the data, but did not) – Unless all devices silently fail in unison, at least one replica has a good copy of the data. When the SPA reads from any replica that was not updated due to a previous phantom write, the checksum fails, (1) applies, and the damage is repaired.
4. Misdirected write (disk wrote the data, but to the wrong block) – This is equivalent to a phantom write (the intended block was not written) and bit rot (another block was written erroneously). Both cases are handled as described in (1) and (3) above.
5. User error – If the system administrator accidentally uses an active ZFS device for some other purpose, the SPA detects it as a storm of checksum errors. As long as the device is mirrored, ZFS can survive this without any loss of data.

Hot Space

The SPA does not require dedicated hot spares. Instead, it spreads hot space across all devices to serve the same purpose. This approach increases the available I/O bandwidth because all devices can be used. Figure 4-1 is an example of how a traditional model reserves an entire disk as a spare. ZFS uses a hot space model to improve utilization and failure prediction.

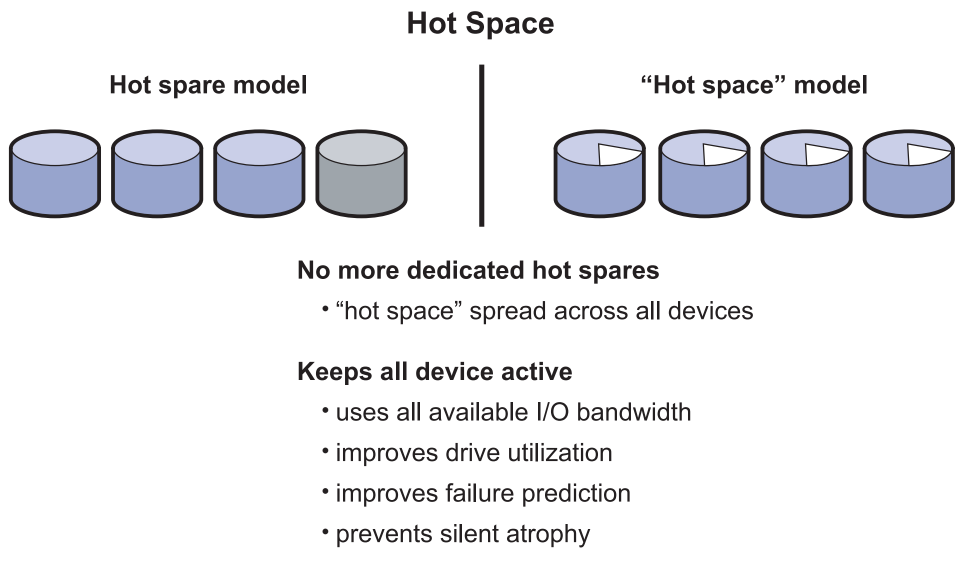

Hot Space
Hot spare model
“Hot space” model
No more dedicated hot spares
• “hot space” spread across all devices
Keeps all device active
• uses all available I/O bandwidth
• improves drive utilization
• improves failure prediction
• prevents silent atrophy

Figure 4-1 ZFS Hot Space Model

Real-Time Remote Replication

The DMU supports real-time remote replication (RTRR). Unlike traditional daily backups or batch executed remote replication, RTRR ensures that the remote copy of the data is always consistent and never more than a few seconds out of date.

User Undo

The most common cause of data loss is user error. In addition to the snapshot facility described earlier, ZFS provides *User Undo*. This allows end users to quickly and easily recover recently-deleted or overwritten files without system administrator intervention.

Capacity

ZFS file systems and storage pools are 128-bit capable, which means that they can support up to 340 undecillion bytes (about 3 x 10^{26} Tbyte). Individual files are also 128-bit capable, but are currently limited to 64-bit access (16EB) because neither the operating system nor file-level protocols like network file system (NFS) support larger files yet.

Performance

Each of the UFS/SVM operations (create a volume, `newfs`, `growfs`) is linear-time: the bigger the disks are, the longer it takes to complete. All ZFS operations are constant-time, regardless of disk size. All writes are sequential.

Dynamic Striping

Traditional striping improves bandwidth by spreading data across multiple disks at some fixed "stripe width." Dynamic striping extends this capability: If there are 20 disks paired into 10 mirrors, the SPA can spread writes across all 10 mirrors. If more disks are added, the SPA can immediately use them. The location of all previously written data is specified by its DMU translations, not by arithmetic based on the number of disks, so there is no stripe geometry to change.

Encryption and Data Security

ZFS supports pluggable encryption modules. Like compression, encryption is a block-level operation. ZFS encryption works with any symmetric block cipher; examples include DES, AES, IDEA, RC6, Blowfish, SEAL, and OCB. The encryption keys can be per-object (to protect a single file), per-file system (to protect one person's or project's data), or per-storage pool (to protect all corporate data).

Volumes and Storage Pools

ZFS makes file system creation and administration easier. The use of pooled storage replaces partitions and volumes. With common, shared space, file systems can grow and shrink automatically. There are no longer raw device names to remember, no more editing `/etc/vfstab` and all administration online. ZFS and its pooled storage also means that `fsck` is no longer needed, which may improve boot time. Figure 4-2 is a comparison chart of volumes compared with storage pools.

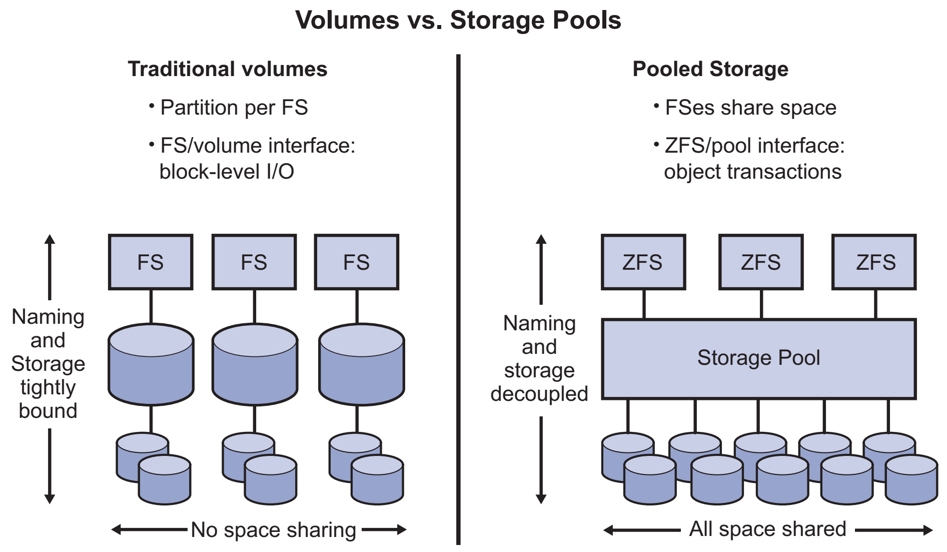

Volumes vs. Storage Pools
Traditional volumes
• Partition per FS
• FS/volume interface: block-level I/O
FS
FS
FS
Naming and Storage tightly bound
No space sharing
Pooled Storage
• FSes share space
• ZFS/pool interface: object transactions
ZFS
ZFS
ZFS
Storage Pool
Naming and storage decoupled
All space shared

Figure 4-2 ZFS Volume and Storage Pool Comparison

Configuration Procedures

To configure and enable ZFS, you must perform the following tasks:

- Create a storage pool and several ZFS file systems to demonstrate the flexibility of this feature.
- Copy files to a ZFS file system and list them.
- Add more storage to the pool dynamically and show that the file systems start using the new space immediately.
- Deliberately corrupt one side of a mirror while it is in active use.
- We will then see that ZFS automatically detects and corrects the disk corruption we caused.
- Bring over, compile, and run the "date" command.

Task Example

The following example demonstrates the configuration procedures

1. Create a storage pool named "home" using a mirror of two disks:

```
# zpool create home mirror /dev/dsk/c3t0d0s0 /dev/dsk/c5t0d0s0
```

Examine the pool we just created:

```
# zpool ls home
Pool                   size   used  avail capacity
home                   8.4G   409K   8.4G     1%
```

2. Create and mount several file systems, one for each user's home directory:

```
# zfs mount -c home/user1 /export/home/user1
# zfs mount -c home/user2 /export/home/user2
# zfs mount -c home/user3 /export/home/user3
```

3. Verify that the file systems were created and mounted:

```
# df -h -F zfs
file system             size   used  avail capacity  Mounted on
home/user1             8.4G     4K   8.4G     1%    /export/home/user1
home/user2             8.4G     4K   8.4G     1%    /export/home/user2
home/user3             8.4G     4K   8.4G     1%    /export/home/user3
```

We created several file systems without making partitions or running the `newfs` command. Notice in the output of the `df` command that all the file systems report 8.4 Gbyte of space available. This is the total amount of space in the storage pool. Potentially, each file system could use all the space in the storage pool, similar to `tempfs`. We can implement quotas to limit how much space each file system can use. Use reservations to reserve space for each file system.

4. Copy some files to `user1`'s file system:

```
# cp /usr/bin/v* /export/home/user1
# ls -l /export/home/user1
total 110
-r-xr-xr-x   1 root     other      72904 Oct 31 00:32 vacation
-r-xr-xr-x   1 root     other       9880 Oct 31 00:32 vax
-r-xr-xr-x   1 root     other     231896 Oct 31 00:32 vedit
-r-xr-xr-x   1 root     other       5447 Oct 31 00:32 vgrind
-r-xr-xr-x   1 root     other     231896 Oct 31 00:32 vi
-r-xr-xr-x   1 root     other     231896 Oct 31 00:32 view
-r-xr-xr-x   1 root     other      28088 Oct 31 00:32 vmstat
-r-sr-xr-x   1 root     other       9972 Oct 31 00:32 volcheck
-r-sr-xr-x   1 root     other      18652 Oct 31 00:32 volrmmount
-rwxr-xr-x   1 root     other      10960 Oct 31 00:32 vsig
```

Note that permissions, ownership, and timestamps are all correct.

5. Add some new disks to the storage pool:

```
# df -h -F zfs
file system             size   used  avail capacity  Mounted on
home/user1             8.4G   901K   8.4G     1%    /export/home/user1
home/user2             8.4G     4K   8.4G     1%    /export/home/user2
home/user3             8.4G     4K   8.4G     1%    /export/home/user3
# zpool add home mirror /dev/dsk/c3t1d0s0 /dev/dsk/c5t1d0s0
# df -h -F zfs
file system             size   used  avail capacity  Mounted on
home/user1              17G   901K    17G     1%    /export/home/user1
home/user2              17G     4K    17G     1%    /export/home/user2
home/user3              17G     4K    17G     1%    /export/home/user3
```

New space is now available to all of the file systems, without the use of the `growfs` command.

6. Next, we simulate disk corruption by writing random information to one of the disks:

```
# dd if=/dev/urandom of=/dev/rdsk/c3t0d0s0 count=10000
10000+0 records in
10000+0 records out
```

7. To verify that the files are unchanged run the `diff` command on all the files with originals in `/usr/bin`:

```
# cd /usr/bin
# diff /export/home/user1/vacation /usr/bin/vacation
# diff /export/home/user1/vax /usr/bin/vax
# diff /export/home/user1/vedit /usr/bin/vedit
# diff /export/home/user1/vgrind /usr/bin/vgrind
# diff /export/home/user1/vi /usr/bin/vi
# diff /export/home/user1/view /usr/bin/view
# diff /export/home/user1/vmstat /usr/bin/vmstat
# diff /export/home/user1/volcheck /usr/bin/volcheck
# diff /export/home/user1/volrmmount /usr/bin/volrmmount
# diff /export/home/user1/vsig /usr/bin/vsig
```

From the lack of returned output from the `diff` commands, we see that no corruption took place. ZFS has 64-bit checksums on every block, which detect data corruption with *nineteen nines* certainty.

8. Look at some statistics to see how many errors ZFS detected and repaired:

```
# zpool ls -v home
                                space      I/O per sec    I/O errors
 vdev   description           used  avail   read  write   found  fixed
    1   mirror(2,3)           1.5M   8.4G      0      0      82     82
    2     /dev/dsk/c3t0d0s0   ----   ----      0      0      82      0
    3     /dev/dsk/c5t0d0s0   ----   ----      0      0       0      0
    4   mirror(5,6)           143K   8.4G      0      0       0      0
    5     /dev/dsk/c3t1d0s0   ----   ----      0      0       0      0
    6     /dev/dsk/c5t1d0s0   ----   ----      0      0       0      0
```

The first two rows show 82 errors on the disk we tried to corrupt. When the mirror device detected that one copy of the data was bad, it tried to read the data from the other copy, on `/dev/dsk/c5t0d0s0`. The checksum on that copy of the data was correct, so it returned that data to the user and wrote the good data back to the corrupted disk. This is why the statistics show that the mirror device found and fixed 82 errors.

ZFS immediately noticed the corruption and automatically corrected the errors and logged them for the administrator. With most file systems, we would not discover disk corruption until file system metadata became corrupted and caused a kernel panic, without detecting user data corruption at all.

9. Watch the pool statistics while we create 5000 directories and copy a large amount of data to all three file systems:

```
# [Start a bunch of disk activity]
# zpool ls -v -i 1 home
                                   space      I/O per sec    I/O errors
 vdev   description             used  avail   read  write   found  fixed
    1   mirror(2,3)             1.5M   8.4G      0   2.4M      82     82
    2     /dev/dsk/c3t0d0s0     ----   ----      0   2.4M      82      0
    3     /dev/dsk/c5t0d0s0     ----   ----      0   2.9M       0      0
    4   mirror(5,6)             146K   8.4G      0   1.6M       0      0
    5     /dev/dsk/c3t1d0s0     ----   ----      0   1.6M       0      0
    6     /dev/dsk/c5t1d0s0     ----   ----      0   2.2M       0      0
                                   space      I/O per sec    I/O errors
 vdev   description             used  avail   read  write   found  fixed
    1   mirror(2,3)             2.6M   8.4G      0   3.1M      82     82
    2     /dev/dsk/c3t0d0s0     ----   ----      0   3.1M      82      0
    3     /dev/dsk/c5t0d0s0     ----   ----      0   2.6M       0      0
    4   mirror(5,6)             918K   8.4G      0   2.3M       0      0
    5     /dev/dsk/c3t1d0s0     ----   ----      0   2.3M       0      0
    6     /dev/dsk/c5t1d0s0     ----   ----      0   1.7M       0      0
 [...]
```

Note – Readers familiar with traditional mirroring may be wondering why the I/O rate for each disk isn't exactly the same as its parent. ZFS does not need to keep its mirrors synchronized except at transaction group boundaries. ZFS-mirrored writes are asynchronous and independent.

10. To see how ZFS automatically handles new disk space, we add more capacity to the pool while all the previous disk activity is going on:

```
# zpool add home mirror /dev/dsk/c3t2d0s0 /dev/dsk/c5t2d0s0
# zpool info -v -i 1 home
                                  space      I/O per sec    I/O errors
vdev   description             used  avail   read  write   found  fixed
   1   mirror(2,3)             7.5M   8.4G      0   2.4M      82     82
   2     /dev/dsk/c3t0d0s0     ----   ----      0   2.5M      82      0
   3     /dev/dsk/c5t0d0s0     ----   ----      0   2.4M       0      0
   4   mirror(5,6)             5.8M   8.4G      0   3.9M       0      0
   5     /dev/dsk/c3t1d0s0     ----   ----      0   3.9M       0      0
   6     /dev/dsk/c5t1d0s0     ----   ----      0   3.8M       0      0
   7   mirror(8,9)             3.4M   8.4G      0   2.2M       0      0
   8     /dev/dsk/c3t2d0s0     ----   ----      0   2.2M       0      0
   9     /dev/dsk/c5t2d0s0     ----   ----      0   2.3M       0      0
[...]
```

The writes are immediately spread out across the new devices, as well as the old ones. With ZFS, the more devices you add, the more disk bandwidth you have, automatically.

Bring over the source for the date command, build it, and execute it.

Common Tasks

To show the simplicity of working with ZFS we examine three routine administrative tasks. Without ZFS, these tasks would take many more commands and far more time to complete. The following examples demonstrate how to:

- Define a storage pool to contain home directories
- Add user Bob's home directory to the pool
- Add more space to the pool

Task Examples

The following examples demonstrate the ease of common tasks.

1. Create a storage pool called `home` with two mirrored disks:

```
# zpool create home mirror disk1 disk2
```

2. Create and mount a ZFS file system called “bob” for Bob's home directory:

```
# zfs mount -c home/bob /export/home/bob
```

3. Add space to the `home` storage pool. The space immediately becomes available to all file systems in the `home` pool, not just Bob’s.

```
# zpool add home mirror disk3 disk4
```

UFS File System

The UFS file system incorporates several new features to make it more efficient, resolve previous issues, and become compliant with Solaris 10. This module covers these new features:

- Improved performance of the existing UFS logging feature
- New directory placement policy
- UFS support of EFI disk labels in support of file systems greater than one terabyte (Tbyte)
- Creation and use of UFS file systems of greater than 1 terabyte
- Tape format for `ufsdump` and `ufsrestore` to permit backups of greater than 2 Tbytes
- Extended the meaning of the new MTB_UFS_MAGIC magic number in the UFS superblock to imply two additional format changes

Improved Logging Features

UFS changes all relate to performance. The main emphasis is improved NFS performance when using UFS logging. Performance improvement for non-logging features is also supported.

The UFS log must be rolled to move the metadata changes to their master on-disk locations. Multiple changes are cached, then written in a batch. The speed of this code is critical to the performance of logging. This code performs a read/modify/write cycle:

1. Read the on-disk metadata block (8K).
2. Overlay this buffer with the log deltas. Up to 50% of the time in testing the log data is not in memory and must be read from disk.
3. Write out the modified sectors.

UFS does not require the read/modify/write cycle. At Moby transaction completion, a buffer exists or is passed into the logging system. Only the pertinent minimally changed metadata is stored on the log, but the aligned sector and sized buffer represents the exact state of the transaction which can be applied to the disk. This buffer can be copied and attached to the map entry describing the delta. If a cached roll buffer (CRB) exists when rolling a delta, it can be used instead to write out the metadata changes.

Cylinder Group Summary Information Management

UFS keeps two sets of cylinder group summary information. The cylinder group header containing individual summaries and a management summary information array is also written. For example, an array of all the individual cylinder group summaries is kept in a contiguous set of sectors following the super block. This copy is maintained to speed the location of space, but it does represent a drain on performance. Whenever an allocation change occurs, the bitmap is logged (delta type DT_CG) and the summary information change logged (DT_SI).

The updates to UFS remove the management summary information array. The data can be easily reconstructed if needed.

Upon mounting a logging UFS file system, a new field in the super block is marked as summary information invalid, and flushed to disk. When the file system is unmounted, the summary information is written and the super block field set to rolled. Should the system crash, the kernel UFS mount code reads all the cylinder groups to reconstruct the summary information (if the super block field is marked invalid). For a nine-way striped 200 Gbyte file system, this reconstruction takes about two seconds. A tunable (`ufs_ncg_log`) is defined to specify when to start logging (in terms of number of file system cylinder groups).

Log Accounting

Transaction logging allocates space for each transaction. The amount assumes that each transaction may need as much space as possible and is often over-reserved. When the log space is over-reserved, a null-sized synchronous transaction is started, which commits the asynchronous actions to disk. Further transactions are blocked until the Moby transaction has committed. The log, which is full from over-reserved transactions, causes an inefficient bottleneck and unnecessary disk log writes. The new UFS feature adds up transaction changes as they are made, on a per-thread basis, and when the transaction is completed the reserved log space is adjusted to the amount actually used.

Determine if Write Lock Is Needed

The `ufs_iupdat` routine extracts the `inode` and pushes any changes to the log. A writer lock on the `inode` calls this function. The writer lock is called in case there are updates to modification times or other values. Often the `inode` needs no other modifications, but the writer lock is still called. Other transactions often read the same file while modifying other parts, which can hold up threads requiring the writer lock. The new feature checks whether the writer lock is needed, because there are no updates to the `inode` before calling for the lock. Not having a write lock gains considerable performance during the critical transaction commit time. This change also improves non-logging performance.

New Directory Placement Policy

A change to the directory placement policy has been proposed. Currently directories are spread across the disk by cylinder groups, regardless of the number of blocks used in each cylinder group. The spreading of the directory entries across the disk leads to long seek times. This configuration was an advantage when disk space was expensive and performance was less of a concern. The new UFS feature favors the current cylinder group rather than the beginning of the disk. When a disk has over 75% utilization, UFS reverts to the previous policy, which affects both logging and non-logging performance.

UFS Support for EFI Disk Labels

The UFS file system needs to be able to interact properly with the growing size of files, file systems, and disks. Previously, UFS did not support devices of greater than 1 Tbyte, which meant that UFS did not need to interact with an EFI disk label. In Solaris 10, EFI labels can appear on any size disk and this feature allows UFS to be usable on smaller disks. The `mkfs` and `newfs` commands also must be able to read an EFI label when creating UFS file systems.

The `mkfs` and `newfs` commands have been modified to use the `efi_alloc_and_read`() library function instead of the `read_vtoc`() function to get the slice size. For disks with EFI labels, `newfs` also needs to be changed to omit the `DKIOCGGEOM` `ioctl`. Instead, `newfs` chooses values for `nsect`, `ntrack`, `cpg`, to enable `mkfs` to create a UFS file system.

Two system files also need to change to reflect the changes to `mkfs` and `newfs`:

- `usr/src/cmd/fs.d/ufs/mkfs/mkfs.c`
- `usr/src/cmd/fs.d/ufs/newfs/newfs.c`

Troubleshooting

These features are enabled as an option to the `mount` command. Issues that may have to do with the file system being mounted and the support that file system has for UFS options.

- How is this feature used? How is the feature turned on?

  ```
  # mount -o logging device mount_point
  ```

- How is the feature turned off?

  ```
  # mount -o nologging device mount_point
  ```

UFS File Systems Greater Than 1 Tbyte

The on-disk format of UFS already supports multi-terabyte file systems. The code that interprets the on-disk format, the kernel UFS module, and various UFS utilities (such as the `mkfs` command) does not. UFS code does not support multi-terabyte file systems in these ways:

- Code that specifically prohibits the creation of UFS file systems of greater than a terabyte, in the mkfs command for example.
- Code that assumes a file system sector offset can be expressed as a 31-bit quantity (UFS kernel code and various UFS utilities).

Several issues also restrict UFS from creating and using file systems greater than 1Tbyte. Following we see some of these features explained.

Older UFS Code

It is necessary to prevent older versions of the UFS code from accessing these multi-terabyte file systems. The old UFS code corrupts the file systems by not using the correct sector offsets. To solve this problem, UFS file systems of greater than one terabyte have a magic number in their superblocks that differs from the historical UFS magic number. Versions of the UFS code that do not have the fix to use 64-bit sector offsets will not recognize the file system as UFS and will not operate on it. Existing UFS kernel modules (that is, those without the fixes to replace use of 32-bit sector offsets with 64-bit offsets) will mount multi-terabyte UFS file systems, which become corrupted when they encounter fragment offsets from beyond the terabyte boundary and shift them into 32-bit quantities, thereby destroying data in the first terabyte of the file system.

Small File Size and Scaling

UFS does not scale well in environments where file systems are used to hold many small and medium files, where the average file size is less than 1 MB. The solution is to restrict the configurations of multi-terabyte UFS file systems allowed to enable the storage of relatively small numbers of very large files and prevent the storage of many small files. This solution requires `mkfs` and `newfs`, the tools used to create file systems, to be modified.

A new `-T` option added to `newfs` specifies that the file system be set up for eventual growth to over a terabyte. When this option is specified, the number-of-bytes-per-inode (nbpi) value for the file system is set by default to 1 Mbyte and the frag size is set to be equal to the file system block size.

Whenever `newfs` is used to create a file of greater than 100 gigabytes, and the `-T` option is *not* specified, the following notice is displayed:

```
The file system parameters used to create this file system
are acceptable for file systems up to 1 terabyte. If the
size of this file system will ever be increased to over 1
terabyte, re-issue this command with the -T option, which
causes the file system to be configured for eventual growth
to over a terabyte.
```

File System Maintenance

The time required to perform a file system check with the `fsck` process is prohibitive on large file systems. Logging reduces the likelihood of having to use `fsck`, but it does not eliminate it. File systems larger than a terabyte, if created with what are now the default characteristics used by `newfs/mkfs`, which is a `nbpi` of 2k and a frag size of 1k, cannot run the `fsck` process. For such large file systems, the failure to run `fsck` is due to the large amount of memory required. Should the `fsck` process acquire the necessary memory, the `fsck` could take weeks. Currently, running the `fsck` process on a 1 Tbyte file system filled with relatively small files can take between four days and a week. With file systems greater than 1Tbyte this time required will grow with the amount of data to check.

64-bit Kernel Support

Multi-terabyte UFS is supported on 64-bit kernels only. An attempt to mount a multi-terabyte UFS file system fails, with an error message, if attempted by a system running a 32-bit kernel.

Logging Enabled

Logging is turned on by default for file systems of greater than one terabyte, which can be overridden by specifying the `nologging` option on the mount. Logging enabled by default is planned for all file systems in Solaris 10 eventually.

`newfs` and `mkfs` Command Changes

The `newfs` and `mkfs` commands need to change to allow for these features. Listed below are the changes to the `man` pages.

The `newfs` `man` page should specify the `-T` option as one of the options passed to `mkfs`. The `-T` option sets the parameters of the file system to allow eventual growth to over a terabyte in total file system size. This option sets `fragsize` to be the same as `bsize`, and sets nbpi to 1 megabyte (unless the `-i` option is used to make it even larger). If the `-f` or `-i` options are used to specify a frag size or nbpi that is incompatible with this option, an error is printed and the file system is not created. The `-T` option to `newfs`(1M) and the `mtb=y` switch to `mkfs`(1M) still set up a file system for eventual growth to over a terabyte. This means that the file created has the two attributes required for online growth to over a terabyte. The required multi-terabyte characteristics are:

- nbpi >= 1MB
- frag size = block size
- A magic number of `MTB_UFS_MAGIC` means that it has the new MTBUFS format

The `mkfs_ufs`(1M) `man` page should be modified to include the `mtb=y` option. This option sets the parameters of the file system to allow eventual growth to over a terabyte in total file system size. This option sets `fragsize` to be the same as `bsize`, and sets `nbpi` to 1 megabyte (unless the `-i` option is used to make it even larger). If the `fragsize` or `nbpi` parameters are explicitly set on the CLI to values incompatible with this option, an error is printed and no file system is created.

Both the `mkfs` and `newfs` `man` pages should be modified to change the description of the `nbpi` and `fragsize` parameters to specify the default settings for file systems of greater than one terabyte.

Installation, Upgrading, and Migration

There are several migration issues that affect this feature:

- Migration from a Solaris version of that doesn't support multi-terabyte UFS (MTBUFS) to one that does, and vice versa
- Solaris OEs that support MTBUFS include the following:
  - Solaris S10 or later, if running a 64-bit kernel.\
  - Solaris 9 Update 4 (assuming that this is integrated into U4), if running a 64-bit kernel.\
  - Solaris 9 Update 3 with an MTBUFS patch (this patch may or may not exist), if running a 64-bit kernel

File System Upgrade or Downgrade

An OS upgrade allows file systems existing on a non-MTBUFS-supporting version of Solaris to continue to be accessible, if the system is upgraded, installed, or patched to a version that does support MTBUFS. If a downgrade is necessary, when the MTBUFS patch was backed out for example, or if a system installed with S10 or S9U4 were re-installed with an earlier version of Solaris, there may be an issue. If file systems of greater than 1 Tbyte had been created on the system, they would no longer be accessible. The mount and `statvfs`(2) system call would fail with `EINVAL` error code. If the system were re-installed later with a version of Solaris that supported MTBUFS, those file systems would become accessible again.

Kernel Upgrade/Downgrade

The kernel itself is a migration issue. Migration from 32-bit kernel to 64-bit kernel and vice versa causes UFS file systems of greater than 1 Tbyte to be in accessible when a system is running a 32-bit kernel. A warning is logged by the 32-bit version of UFS when an attempt is made to mount an MTBUFS file system to clarify why a formerly-available file system is no longer accessible.

Sun Cluster Rolling Upgrade

Sun Cluster software is for data availability. A global file system has to be available to all nodes of the cluster in a failure situation. If every node is incapable of MTBUFS support, the file system may not be available. During a rolling upgrade, nodes of clusters are at different patch and OS levels. Data availability may not continue if a failure occurs in the middle of a rolling upgrade. It is recommended that no multi-terabyte UFS file systems be created on clusters until all nodes in the cluster have been upgraded to a version of Solaris that supports MTBUFS.

Tape Format for `ufsdump` and `ufsrestore`

The tape format currently used by `ufsdump` and `ufsrestore` uses a 32-bit field to represent tape block offsets, where tape blocks are defined to be 1024 bytes. This is not a configurable setting. This limits the amount of data that can be stored in a backup to 2 terabytes.

To combat this issue a new tape format feature is available. The new format has a field in the tape header which contains the tape block size. Tape block offsets are in units of the tape block size, a value defined in the header. By setting the tape block size appropriately, a UFS file system of any size, even one bigger than the current 16 Tbyte maximum file system size, can be backed up. The new tape format is indicated by a new magic number in the tape header. This prevents older versions of `ufsrestore` from attempting to restore data from new-format tapes. If an attempt is made to use an older version of `ufsrestore` to extract data from a backup with the new magic number, `ufsrestore` will not recognize the backup medium as a valid dump. For more flexibility the new format is only be used when the size of the backup to the tape exceeds 2 terabytes. Smaller backups still use the old format and therefore are restorable on older Solaris systems.

UFS Superblock

Testing of the multi-terabyte file system has revealed that the UFS on-disk format specifies a signed 32-bit field to hold the offset to the log, which is specified in disk blocks. The number of disk blocks in a multi-terabyte file system may exceed 2^31, causing the field to overflow and the log to be placed outside the first terabyte of the device, which should not occur.

The meaning of the `MTB_UFS_MAGIC` number is changed to resolve this issue. Now, a file system with a magic number of `MTB_UFS_MAGIC` has the following characteristics:

- File system *may* be greater than 1 Tbyte, but need not be.
- Logging is turned on by default.
- Log offset fields (`fs_logbno` in the superblock and `pbno` in the `extent_t` struct) are in units of frags, not sectors. Frag offsets are guaranteed not to exceed 31 bits.
- Superblock has a new field, `fs_version`, which is the `MTBUFS` disk format version.

The result of this change is that the new UFS format, by itself, has been made largely independent of the matter of multi-terabyte support. The new format is required for multi-terabyte file systems, but may have other uses. This feature is also supported on file systems of less than 1 terabyte.

Changes Related to the Feature

The migration of file systems from less than a terabyte to more than a terabyte requires changes to values stored in the superblock.

File System Characteristics

A file system of less than 1Tbyte which has both the required multi-terabyte characteristics (`nbpi` >= 1MB, frag size = block size), and the magic number of `MTB_UFS_MAGIC` (meaning that it has the new MTBUFS format) can be grown online, which means growing without an unmount, to over a terabyte.

A file system of less than a terabyte with the required multi-terabyte characteristics), but a magic number of `FS_MAGIC` can be grown to over a terabyte, but it first must be unmounted.

A file system of less than one terabyte that does not have the required characteristics of `nbpi` >= 1 Mbyte and frag size equal to block size, cannot be grown to over a terabyte, regardless of its magic number.

Migrations from less than 1 Tbyte to greater than1 Tbyte can take place only in 64-bit mode.

A file system with a magic number of `MTB_UFS_MAGIC` can be mounted on a system running a 32-bit kernel, but only if it's less than a terabyte in size.

The kernel and all utilities recognize a file system as a valid UFS file system if the following is true:

- The magic number in the superblock is `FS_MAGIC`

  or

- The magic number in the superblock is `MTB_UFS_MAGIC`

  and

- The version number in the superblock is less than or equal to the highest version number recognized by the code (currently 1).

A version change, therefore, indicates the existence of a change to the UFS on-disk format, making it incompatible with previous interpreters of the UFS on-disk format.

The `-T` option to `newfs`(1M) and the change from `mtb=y` to `mkfs`(1M) still set up a file system for eventual growth to over a terabyte, which means that the file created has the two attributes required for online growth to over a terabyte.

Module 5

Fault and Service Management

Objectives

This module is an overview of the fault and service management features included in the Solaris™ 10 Operating System (Solaris 10 OS). Upon completion of this module, you should be able to:

- Identify features of the Fault Management Architecture
- Identify features of the Service Management Facility

Fault Management Architecture

A *fault manager* is a software component that acts as a multiplexor between error reports produced by system components and companion software that is designed to diagnose and respond to those error reports to facilitate self-healing and improve availability. The fault manager's clients are a set of components called *diagnosis engines* that can automatically diagnose problems using the error reports and another set of components called *agents* that automatically respond to the problem diagnoses by performing actions such as disabling faulty components, issuing messages to alert human administrators, and providing information to higher-level management software and remote services.

Features

Fault Management Architecture (FMA) introduces a new software architecture and methodology for fault management across Sun's product line. The Solaris FMA model provides for three activities with which fault management code must concern itself:

- Error handling
- Fault diagnosing
- Response

Error handling begins upon detection of an error. Error handlers (frequently drivers) are expected to capture sufficient data to diagnose the underlying problem, isolate the effects of the error and generate the appropriate error report to describe the error to other software such as a diagnosis engine. Fault diagnosis is the asynchronous analysis of a problem or defect in the system. Typically, a diagnosis is inferred from a string of error events. Once a diagnosis of the problem has been determined, system fault management software autonomically responds to perform isolation and self-healing tasks. For example, a hardware component called out in a diagnosis can be automatically reconfigured or disabled until it is repaired and able to be integrated back into the system.

A flowchart of fault management activities is shown in Figure 5-1. An *ereport event* denotes the asynchronous transmission of an error report to a piece of software responsible for its diagnosis. A *fault event* denotes the asynchronous transmission of a fault diagnosis to agent software responsible for taking some action in response to the diagnosis.

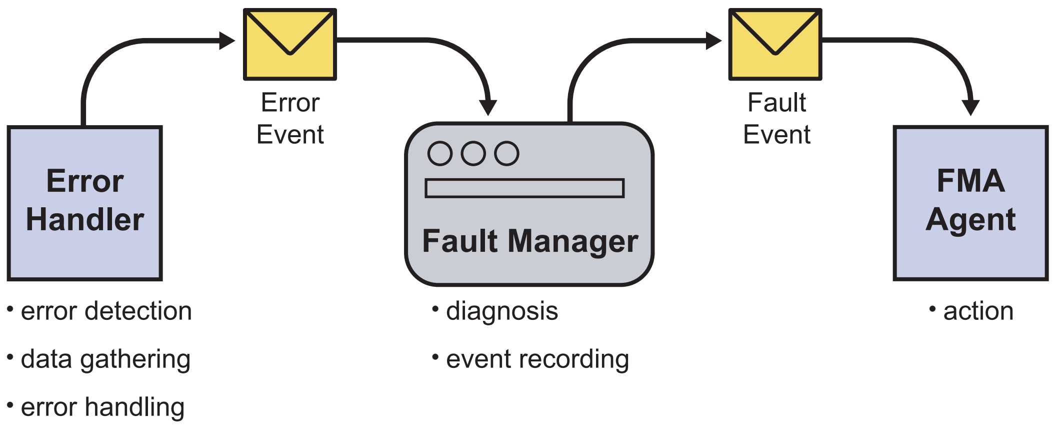


Figure 5-1 Control Flow for Fault Management Activities

Event Naming Schemes

Part of the fault management architecture is an event naming scheme. The naming schemes are hierarchical trees. The three root schemes currently defined for events are:

- `ereport` – Events related to error events
- `fault` – Events related to faults
- `list` – Events related to lists

The subcategories for the `ereport` and `fault` schemes are:

- `asic` – Application specific integrated circuits (ASICs) events
- `cpu` – CPU and memory subsystem events
- `io` – I/O device events
- `solaris` – Events generated for Solaris system software

The list scheme has only one event; the `list.suspect` event. This event is used to list all fault suspects for a set of related events (a *case*).

Events are what are passed among the various components of the fault management architecture. Examples of events are:

```
ereport.cpu.ultraSPARC-III.UE
fault.asic.shizo.*
list.suspect
```

Each event may be associated with a list of name-value pairs. One of the entities included in this is the fault managed resource identifier (FMRI). An FMRI identifies resources that detected an error, are affected by an error or have had a change of state following fault diagnosis. FMRI specifications describe the *detector* in error reports and the indicted Automated System Reconfiguration Unit (ASRU), Field Replaceable Unit (FRU), and *resource* in fault reports. An FMRI may be represented as a list of name-value pairs or a hierarchical text string.

The syntax for describing an FMRI text string is based upon Uniform Resource Identifiers (URI): Generic Syntax (RFC_2396). A URI is a string of characters used to identify an abstract or physical resource. Typically, the resource is one that can be located on the Web where a URI is a superclass of the familiar Universal Resource Locator (URL).

The string-based FMRI introduces a new set of schemes for the fault management namespace. FMRI schemes are hierarchical and describe resources such as CPUs, memory, device paths, services, and network interfaces. the general syntax of the FMRI is dependent upon the scheme. The path may represent either a logical or physical path to the fault managed resource. All FMRIs are composed of three main elements:

`fmri-scheme://[authority]/path`

where:

- *`fmri-scheme`* – Specifies the format of the text string and owner of the resource repository
- *`authority`* – Provides a means by which resource may be identified
- *`path`* – Specifies the resource

The currently defined FMRI protocol schemes are:

- `hc` – Hardware component managed by FMA
- `diag-engine` – Diagnosis engine which is part of FMA
- `dev` – Solaris device path status and properties
- `svc` – Application service managed by the Service Management Facility
- `netif` – Network interface managed by FMA

Some example FMRIs are:

```
hc:///product-id=SunFire15000,chassis-id=138A2036,domain-
id=A/chassis/system-board=0/cpu-module=2/cpu=8

diag-engine://product-id=E450,chassis-
id=238B2036/de=eversholt/:version=1.0

dev:///devices/pci@8,700000/scsi@6/sd@6,0:g/:devid=1234
```

Administrator Commands

The fault management architecture has the following user-level commands:

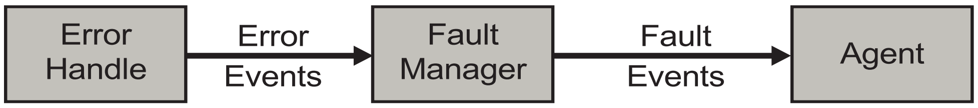
- `fmd` (1M) – The daemon which receives telemetry information relating to problems, diagnoses these problems, and initiates proactive self-healing activities.
- `fmadm` (1M) – A utility to view and modify system configuration parameters maintained by the `fmd` daemon.
- `fmstat` (1M) – A utility to report statistics associated with the `fmd` daemon and its associated modules.
- `fmdump` (1M) – A utility to display the contents of the fault and error logs.

Solaris Fault Management Daemon

The fault management daemon (`fmd`(1M)) contains the following components:

- Diagnosis engines – Software which subscribes to error reports and diagnoses the fault
- Agents – Software which is responsible for repairing a fault
- Schemes – Software which is responsible for translating information from one protocol to another

The components of `fmd` are implemented as plug-in modules (see Figure 5-2). A plug-in module can be any of the above components. The daemon's interactions with the plug-ins is through subscriptions to events.



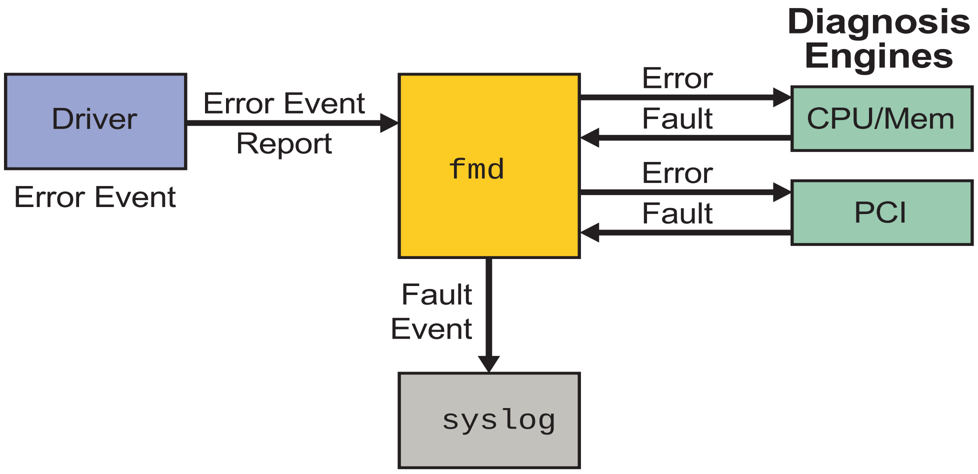


Figure 5-2 Solaris Fault Management Architecture

Plug-ins are searched for in the following directories (in order):

- `/usr/platform/'uname -i'/lib/fm/fmd/plugins`
- `/usr/platform/'uname -m'/lib/fm/fmd/plugins`
- `/usr/lib/fm/fmd /plugins`

This search order allows platform specific modules to be loaded prior to more general modules.

The fault manager provides interfaces for diagnosis engines to create and update Soft Error Rate Discrimination (SERD) engines that are automatically check pointed and restored on behalf of the module. SERD engines are in effect ring buffers of events that can be used to determine whether N events in some time T have been seen, indicating an anomaly or discriminating expected soft upsets from faults.

Modules that implement diagnosis algorithms are expected to associate incoming data and eventual diagnosis results with one or more *cases*, which act as a kind of metaphorical folder used to organize information relevant to a particular problem. Every case is named by a Universal Unique Identifier (UUID) which is recorded in any events related to the case. The fault manger considers each case to be in one of three states at any given time:

- Unsolved – The case has no `list.suspect` event associated with it and the diagnosis engine has not yet indicated that the case is solved.
- Solved – The case has been *solved* as a result of the module adding one or more suspect faults to it and then indicating the case is solved. When a case is solved, a `list.suspect` event for the case is published through the event dispatch mechanism.
- Closed – The case has been closed either because it was unnecessary and never solved, or because it was solved and an agent has acted on the resulting suspect list by disabling the affected system Automated System Reconfiguration Units (ASRUs).

Looking at FMA Data

You can use the `fmadm` utility to view and modify system configuration parameters maintained by FMA. The following are the subcommands to the `fmadm` command:

- `config` – Show fault manager configuration
- `faulty` – Show list of faulty resources
- `flush` *`fmri`* – Flush the cached state for the specified *`fmri`*
- `load` *`path`* – Load the specified plug-in *`module`*
- `repair` *`fmri`* – Record a repair to the specified resource
- `reset [-s` *`serd`*`]` *`module`* – Reset the parameters of the specified *`module`* or its SERD engine
- `rotate` *`logname`* – Rotate the specified log file
- `unload` *`module`* – Unloaded the specified *`module`*

The following example shows the output of the `fmadm config` command. This command shows all modules which are currently loaded.

```
# fmadm config
MODULE                          VERSION DESCRIPTION
cpumem-retire                   1.0     CPU/Memory Retire Agent
fmd-self-diagnosis              1.0     Fault Manager Self-Diagnosis
syslog-msgs                     1.0     Syslog Messaging Agent
```

The following example shows the use of the `fmadm faulty` command. This displays the FMRI and Universal Unique Identifier (UUID) for all faulty components in the system. The UUID is an identifier assigned to each case. All events and diagnoses corresponding to a case will have the same UUID.

```
# fmadm faulty
   STATE RESOURCE / UUID
-------------------------------------------------------------------------
 faulted mem:///component=Slot%20B%3A%20J3000
         dbdc7f15-848c-cbdc-b47f-deb9d9fff5c9
-------------------------------------------------------------------------
```

FMA keeps a resource database which contains the state of any resource which has occurred in an event. The possible states of a resource are:

- `ok` – The resource is present and in use and FMA detects no known problems.
- `unknown` – The resource is not present or not usable but has no known problems. This might indicate the resource has been disabled or unconfigured by an administrator.
- `degraded` – The resource is present and usable, but FMA has diagnosed one or more problems in the resource.
- `faulted` – The resource is present but is not usable because one or more problems have been diagnosed by FMA. The resource has been disabled to prevent further damage to the system.

The `fmadm repair` command is used to specify that a faulty resource has been repaired. The use of this command is only necessary if the component cannot be automatically removed from use by the system.

```
# fmadm repair mem:///component=Slot%20B%3A%20J3000
fmadm: recorded repair to mem:///component=Slot%20B%3A%20J3000
```

Once a resource has been specified as repaired, the resource no longer shows in the faulty resources list.

```
# fmadm faulty
   STATE RESOURCE / UUID
-------------------------------------------------------------------------
```

The `fmstat` command shows statistics about each module active in FMA. The columns of data are:

- `ev_recv` – The number of events received by the module
- `ev_acpt` – The number of events accepted by the module as relevant to a diagnosis
- `wait` – The average number of events waiting to be examined by the module
- `svc_t` – The average service time in milliseconds for events received by the module
- `%w` – The percentage of time that there were events waiting to be examined by the module
- `%b` – The percentage of time that the module was busy processing events
- `open` – The number of active cases owned by the module
- `solve` – The total number of cases solved by this module since it was loaded
- `memsz` – The amount of dynamic memory currently allocated by this module
- `bufsz` – The amount of persistent buffer space currently allocated by this module

```
# fmstat
module              ev_recv ev_acpt wait  svc_t  %w  %b  open solve  memsz
bufsz
cpumem-diagnosis        4       4  0.0  262.8   0   0     1     1   308b
356b
cpumem-retire          1       0  0.0   66.6   0   0     0     0      0
0
fmd-self-diagnosis      0       0  0.0    0.0   0   0     0     0      0
0
syslog-msgs            1       0  0.0    0.1   0   0     0     0    32b
0
```

When a module is specified to the `fmstat` command, all statistics for that module are shown.

```
# fmstat -m cpumem-retire
                NAME VALUE            DESCRIPTION
           auto_flts 0                auto-close faults received
            bad_flts 0                invalid fault events received
         cpu_blfails 0                failed cpu blacklists
          cpu_blsupp 0                cpu blacklists suppressed
           cpu_fails 0                cpu faults unresolveable
            cpu_flts 0                cpu faults resolved
            cpu_supp 0                cpu offlines suppressed
            nop_flts 0                inapplicable fault events received
          page_fails 0                page faults unresolveable
           page_flts 0                page faults resolved
         page_nonent 0                retires for non-existent fmris
           page_supp 0                page retires suppressed
```

Using the `-s` option to the `fmstat` command specifies to show the SERD engine information for the specified module. The columns of output are:

- NAME – The name of the SERD engine
- >N – The number of events it takes to trigger the SERD engine
- T – The time in which the events must take place to trigger the engine
- CNT – The number of events seen by the SERD engine
- DELTA – Nanoseconds from the oldest event to the current time
- STAT – The current status of the SERD engine

```
# fmstat -s -m cpumem-diagnosis
NAME                                  >N     T CNT
DELTA STAT
0f353373-67f0-6585-bd01-a405f6d9cdec  >2    3d   1
43597245900ns pend
```

The following example shows the SERD engine with a status of `fire` after the number of events exceeded the configured count (>N).

```
# fmstat -s -m cpumem-diagnosis
NAME                                  >N     T CNT
DELTA STAT
0f353373-67f0-6585-bd01-a405f6d9cdec  >2    3d   3
200606494100ns fire
```

The `fmdump` command is used to show information from the error and the fault logs kept by FMA. These logs are kept in the `/var/fm/fmd` directory. However, these are binary files and can only be looked at using the `fmdump` command.

When an error event is received by the `fmd` command, the event is logged to the error log prior to acknowledging receipt of the event to the transport. Initially the event is logged with a header that means it has not yet been processed by a module. When a module accepts an error event, that header is changed to indicate the event does not need to be replayed in the case of a failure.

The following example uses the `-e` option to display the contents of the error log. The output of this command is considered contract private. Scripts should not be written that depend on the format of this output.

```
# fmdump -e
TIME                 CLASS
Jul 26 12:49:27.6137 ereport.cpu.ultraSPARC-IIIplus.ce
```

The `-V` option of the `fmdump` command specifies to show all the information available about the event. In the following example, complete data about the detector of the error and the resource in which the error occurred is given.

```
# fmdump -Ve
TIME                           CLASS
Jul 26 12:49:27.613793750 ereport.cpu.ultraSPARC-IIIplus.ce
nvlist version: 0
        class = ereport.cpu.ultraSPARC-IIIplus.ce
        ena = 0x1b130dc18000001
        detector = (embedded nvlist)
        nvlist version: 0
                version = 0x0
                scheme = cpu
                cpuid = 0x0
                cpumask = 0xb1
                serial = 0x12e066b4103
        (end detector)

        afsr = 0x20000003e
        afar-status = 0x1
        afar = 0xb1f1b9c000
        pc = 0x7b6029cc
        tl = 0x0
        tt = 0x63
        privileged = 1
        multiple = 0
        syndrome-status = 0x1
        syndrome = 0x3e
        error-type = Persistent
        l2-cache-ways = 0x1
        l2-cache-data = 0xec0106f1a6 0x39c000 0x0 0x2c7c6000002 0xb9
0xeccf00df1b9c000 0xeccf00df1b9c000 0xeccf00df1b9c000 0xeccf00df1b9c000
0x2c361 0xeccf00df1b9c000 0xeccf00df1b9c000 0xeccf00df1b9c000
0xeccf00df1b9c000 0x2c361
        dcache-ways = 0x0
        icache-ways = 0x0
        resource = (embedded nvlist)
        nvlist version: 0
                version = 0x0
                scheme = mem
                unum = Slot B: J3000
        (end resource)
```

The following example shows that allow there is an event in the error log, no fault has as yet been diagnosed.

```
# fmdump
TIME                 UUID                                 SUNW-MSG-ID
fmdump: /var/fm/fmd/fltlog is empty
```

Once the correctable memory error occurs enough times to cause the SERD engine to fire, a fault is created. Showing the fault log shows the time of the fault, the UUID for the fault and a message identifier. This message identifier can be used on the `http://www.sun.com/msg` site to find additional information about the message. There is also an internal prototype Web Site at `http://msg.central/msg`.

```
# fmdump
TIME                 UUID                                 SUNW-MSG-ID
Jul 26 12:52:10.6786 dbdc7f15-848c-cbdc-b47f-deb9d9fff5c9 SUN4U-8000-1A
```

The `-v` option of `fmdump` provides extra information about the fault.

```
# fmdump -v -u dbdc7f15-848c-cbdc-b47f-deb9d9fff5c9
TIME                 UUID                                 SUNW-MSG-ID
Jul 26 12:52:10.6786 dbdc7f15-848c-cbdc-b47f-deb9d9fff5c9 SUN4U-8000-1A
  100%  fault.memory.page
         FRU: mem:///component=Slot B: J3000
        rsrc: mem:///component=Slot B: J3000
```

The following is a message recorded by syslog about an error that required a system reboot. Notice the individual `ereport` information given as a safety for the information not getting replayed properly.

```
SUNW-MSG-ID: SUNOS-8000-0G, TYPE: Error, VER: 1, SEVERITY: Major
EVENT-TIME: 0x40c5f5b8.0x1017d044 (0x69e2a9b6e4)
PLATFORM: SUNW,Sun-Fire-880, CSN: -, HOSTNAME: s12y-lab
SOURCE: SunOS, REV: 5.10 s10_56
DESC: Errors have been detected that require a reboot to ensure system
integrity.  See http://www.sun.com/msg/SUNOS-8000-0G for more
information.
AUTO-RESPONSE: Solaris will attempt to save and diagnose the error
telemetry
IMPACT: The system will sync files, save a crash dump if needed, and
reboot
REC-ACTION: Save the error summary below in case telemetry cannot be
saved


ereport.io.pci.master-abort ena=69e2a5c19c00005 detector=[ version=0
scheme=

"dev" device-path="/pci@9,600000" ] pci-status=22a0 pci-command=146 pci-
pa=0


ereport.io.pci.master-abort ena=69e2a5c19c00009 detector=[ version=0
scheme=

 "dev" device-path="/pci@8,700000" ] pci-status=2280 pci-command=146 pci-
pa=0


ereport.io.pci.master-abort ena=69e2a5c19c0000d detector=[ version=0
scheme=

 "dev" device-path="/pci@9,700000" ] pci-status=2280 pci-command=146 pci-
pa=0


ereport.cpu.ultraSPARC-IIIplus.ue ena=69e2a5c19c00001 detector=[
version=0

 scheme="cpu" cpuid=0 cpumask=b1 serial=12e1df824e1 ] afsr=1000040000000a
 afar-status=1 afar=a1fe4a4020 pc=7b602a00 tl=0 tt=32 privileged=1
multiple=0

 syndrome-status=1 syndrome=a error-type="Unknown" l2-cache-ways=1

 l2-cache-data=[...] dcache-ways=0 icache-ways=0 resource=[ version=0
scheme=

 "mem" unum="Slot A: J2900 J2901 J3001 J3000" ]
```

The following message shows the retirement of a page to correct a memory error.

```
SUNW-MSG-ID: SUN4U-8000-35, TYPE: Fault, VER: 1, SEVERITY: Minor
EVENT-TIME: Tue Jun  8 13:24:12 EDT 2004
PLATFORM: SUNW,Sun-Fire-880, CSN: -, HOSTNAME: s12y-lab
SOURCE: cpumem-diagnosis, REV: 1.0
EVENT-ID: 7a67763c-08f0-672e-ceee-ad6ab7065113
DESC: The number of errors associated with this memory module has
exceeded acceptable levels.  Refer to http://sun.com/msg/SUN4U-8000-35
for more information.
AUTO-RESPONSE: Pages of memory associated with this memory module are
being removed from service as errors are reported.
IMPACT: Total system memory capacity will be reduced as pages are
retired.
REC-ACTION: Schedule a repair procedure to replace the affected memory
module.  Use fmdump -v -u <EVENT_ID> to identify the module.
```

Troubleshooting

If you believe that `fmd` is non-functional, send it a `SIGABRT`. This forces it to dump core, which can than be examined to determine the cause.

A module has been written for the Modular Debugger (MDB) to simplify the debugging of the `fmd` command and its plug-in modules.The debugging module can also be used to learn more about low-level details. To use this module, launch the ``mdb -p `pgrep -x fmd` `` command. The `fmd` debugging support module loads automatically. Use `::dmods -l fmd` to see a list of available commands.

Changed Packages and Files

Table 5-1 summarizes the new or modified package files, header files, and data structures implementing the FMA.

Table 5-1 Affected Packages, Files, and Data Structures

Package, File, and Structure	**Filename**
SUNWfmd	`fmd` (1M)
	`fmadm` (1M)
	`fmstat` (1M)
	`fmdump` (1M)
	`/usr/lib/fm`
	`/var/fm`

The Service Management Facility

The Service Management Facility (SMF) delivers a unified Solaris service configuration infrastructure capable of accurately modeling any Solaris service and its interaction with Solaris and other services. Rather than the problematic use of `rc` scripts, SMF starts services in parallel according to dependencies, which allows the system to boot faster, and reduces dependency conflicts.

Features

An SMF infrastructure consisting of a service configuration repository, process re-starter, and administrative CLI utilities along with supporting kernel functionality is available, enabling Solaris services to express:

- Restart requirements
- Requirements for the presence of prerequisite services and system facilities (such as networking)
- Requirements for identity and privileges for various tasks
- Configuration settings per instance

Solaris services are modeled by describing them in terms of an SMF schema and associated service methods. For existing services converted to SMF services, compatibility or conversion of legacy configuration files is handled on a service-by-service basis. Once service descriptions are bootstrapped into SMF, instances of such services can be created, started, stopped, and status collected by the infrastructure. This saves time and system administration effort.

The SMF Architecture

The service management facility is a mechanism for providing service start and restart contracts. The goals of the project are:

- Supply a mechanism to formalize relationships between services
- Provide a unified repository for configuration of service startup behavior
- Allow Solaris to start and restart services automatically over the lifetime of a Solaris instance

Figure 5-3 shows the main components of SMF.

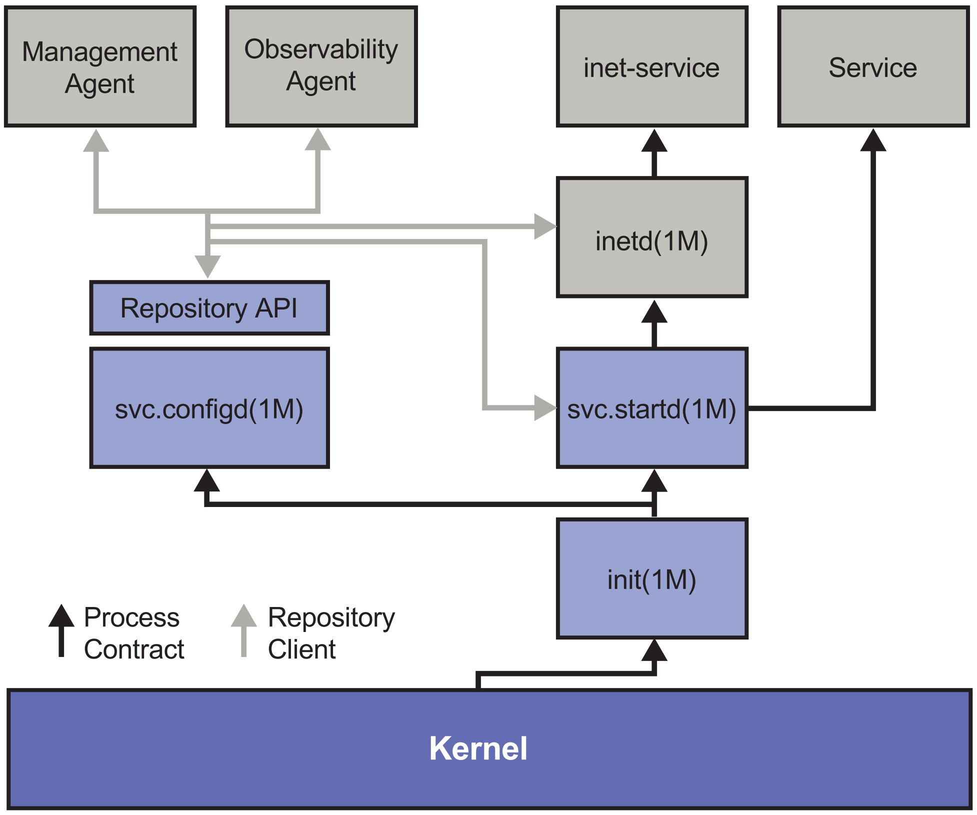


Figure 5-3 The SMF Components

The main components of SMF are:

- Service abstraction
- Repository of service information
- Daemon to access the repository (`svc.configd`)
- APIs for access to the repository
- Master restarter daemon (`svc.startd`)
- Delegated restarters (e.g. `inetd`)
- Command line tools

Services

The fundamental unit of administration in SMF is the service. Generically, a service is an entity which provides a known list of capabilities to other local and remote services. The categories of services are:

- `milestone` – Synthetic services for clean dependency statements
- `device` – General device services
- `system` – Services concerned with host-centric, non networked capabilities
- `system/security` – Low-level host-centric services implementing security facilities
- `network` – Services concerned with host-centric, network infrastructure capabilities
- `application` – General software services
- `application/management` – Services implementing management facilities
- `application/security` – Services implementing high-level security facilities
- `site` – Services implementing site-specific software
- `platform` – Services implementing platform-specific software

The milestone service is special in that there is no software to run in connection with the service. A milestone corresponds to the system arriving at a defined set of capabilities. The milestones are used to replace the run levels used with the `init` command and the `rc*.d` scripts. The current milestones are:

- `milestone/name-services:default` – A milestone for use by services who can not run until a name service is running.
- `milestone/devices:default` – A milestone for use by services that have a dependency on local devices being available.
- `milestone/single-user:default` – A milestone roughly equivalent to single-user mode or init run level one.
- `milestone/multi-user:default` – A milestone roughly equivalent to init run level two.
- `milestone/multi-user-server:default` – A milestone roughly equivalent to init run level three.

Information about services and their state is kept in the repository. This information can be accessed using the `svcs` command.

```
sys-01# svcs
STATE          STIME    FMRI
legacy_run      9:17:58 lrc:/etc/rcS_d/S10pfil
legacy_run      9:17:58 lrc:/etc/rcS_d/S29wrsmcfg
legacy_run      9:17:58 lrc:/etc/rcS_d/S35cacheos_sh
legacy_run      9:17:58 lrc:/etc/rcS_d/S41cachefs_root
legacy_run      9:17:58 lrc:/etc/rcS_d/S55fdevattach
legacy_run      9:18:09 lrc:/etc/rc2_d/S10lu
legacy_run      9:18:09 lrc:/etc/rc2_d/S20sysetup
legacy_run      9:18:09 lrc:/etc/rc2_d/S40llc2
legacy_run      9:18:09 lrc:/etc/rc2_d/S42ncakmod
legacy_run      9:18:09 lrc:/etc/rc2_d/S47pppd
legacy_run      9:18:10 lrc:/etc/rc2_d/S65ipfboot
legacy_run      9:18:10 lrc:/etc/rc2_d/S70sckm
legacy_run      9:18:10 lrc:/etc/rc2_d/S70uucp
. . .
online          9:16:08 svc:/system/svc/restarter:default
online          9:17:12 svc:/milestone/name-services:default
online          9:17:28 svc:/network/loopback:default
online          9:17:29 svc:/network/initial:default
online          9:17:29 svc:/network/physical:default
online          9:17:30 svc:/network/service:default
online          9:17:44 svc:/network/ssh:default
online          9:17:46 svc:/milestone/devices:default
online          9:17:46 svc:/system/device/local:default
online          9:17:55 svc:/system/filesystem/minimal:default
online          9:17:56 svc:/network/rpc/bind:default
online          9:17:56 svc:/network/rpc/keyserv:default
. . .
online          9:55:48 svc:/system/console-login:default
online         13:19:00 svc:/network/telnet:default
offline         9:16:11 svc:/application/print/ipp-listener:default
```

Solaris uses a URI string called a Fault Managed Resource Identifier (FMRI) to identify system objects for which advanced fault and resource management capabilities are provided. Services managed by SMF are assigned FMRI strings prefixed with the scheme name `svc` or `lrc`. The `svc` scheme is the type used for services that are SMF aware. The `lrc` scheme is used to support legacy services which have not been migrated to SMF.

A service provides a known list of capabilities. There are times when it is helpful to run multiple instances of a service (for example a Web Server serving multiple ports). SMF provides for service instances. The first instance of a service is normally tagged the default instance. For example, `svc:/network/rpc/bind:default` identifies the default instance of the `/network/rpc/bind` service.

The following is an example of a service with multiple instances:

```
sys-01# svcs "svc:/system/sysidtool*"
STATE          STIME    FMRI
online          9:17:56 svc:/system/sysidtool:net
online          9:17:58 svc:/system/sysidtool:system
```

Service States

A service can be in one of the following states (see Figure 5-4):

- Uninitialized – Uninitialized is the initial state for all instances. Services in this state are not yet running, and their configuration data is unread.
- Offline – Instances are in the offline state when their configuration has been read but they aren't running. Instances remaining in this state are usually the victim of unsatisfied dependencies or errors occurring during the start method.
- Online – The online state describes a running service with all dependencies met.
- Disabled – The disabled state is a result of the service instance being marked as disabled in the configuration data or explicitly disabled by the administrator. While the service may be startable, the administrator must interact with SMF in order to start the service.
- Degraded – The degraded state is when the service instance still meets most of its criteria for execution but has some limited set of failures which identify it as degraded.
- Maintenance – The maintenance state indicates the service is unavailable due to maintenance activities or requires administrator intervention. The maintenance state can be reached either by explicit administrative request or through an internal action by SMF in response to a non-transient error of the service or the state machine.

Services transition from one state to another either due to explicit administrative action or by SMF in response to dependency changes or error conditions. Figure 5-4 shows the SMF service states.

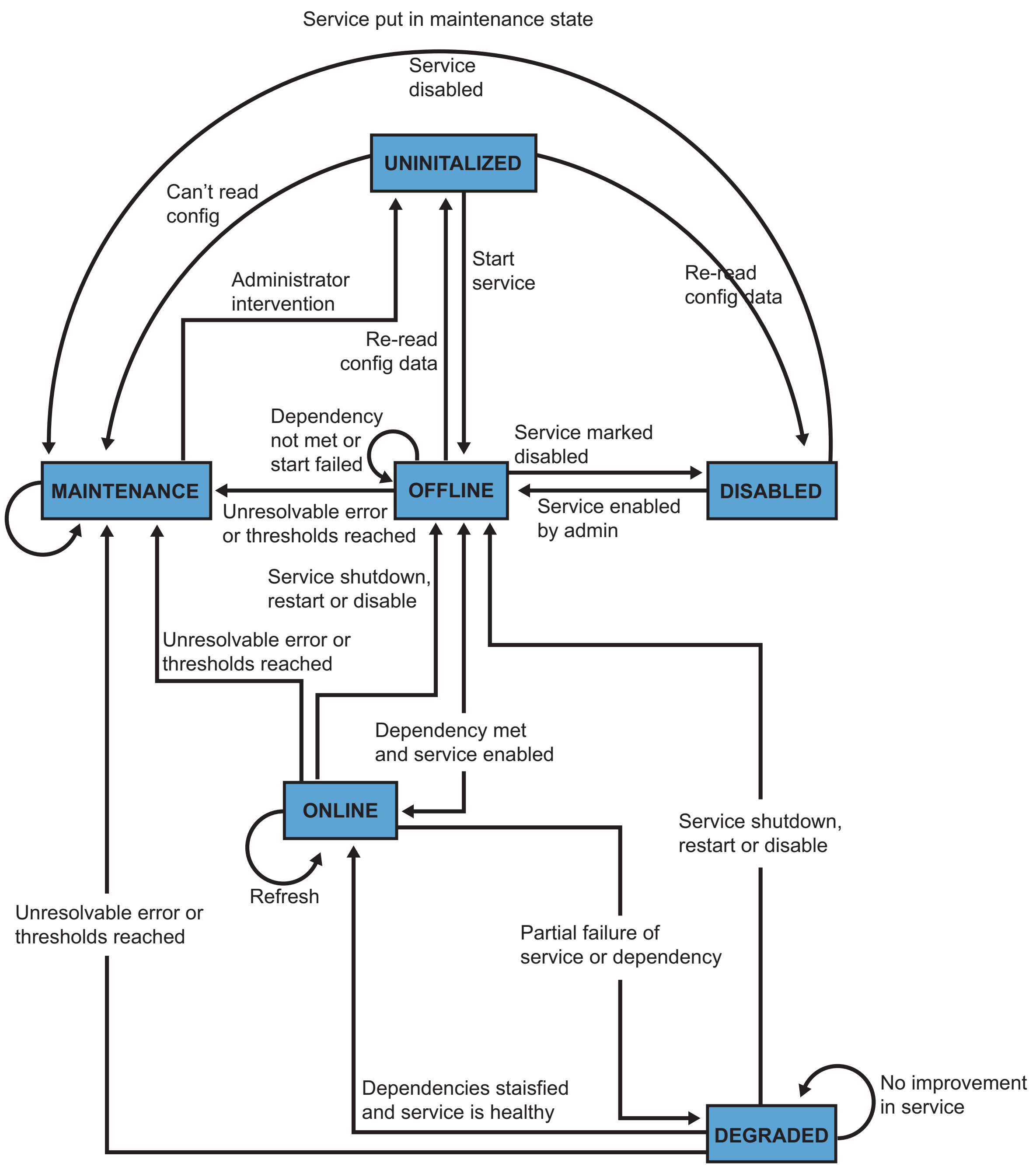

Service put in maintenance state
Service disabled
UNINITALIZED
Can't read config
Administrator intervention
Start service
Re-read config data
Re-read config data
Dependency not met or start failed
Service marked disabled
MAINTENANCE
OFFLINE
DISABLED
Unresolvable error or thresholds reached
Service enabled by admin
Service shutdown, restart or disable
Unresolvable error or thresholds reached
Dependency met and service enabled
ONLINE
Service shutdown, restart or disable
Refresh
Unresolvable error or thresholds reached
Partial failure of service or dependency
No improvement in service
Dependencies staisfied and service is healthy
DEGRADED

Figure 5-4 SMF Service States

Service Components

Services are composed of several components. A sample is:

- A mechanism to start and stop the service
- A mechanism to monitor and restart services
- A location for configuration data (properties)
- A location for error messages

SMF organizes services using profiles and manifests. A profile is used to set general settings for a system as to what services need to run. The profile files are usually found in the `/var/svc/profile` directory.

A manifest is used to describe a single service or set of related services. It is possible to specify configuration parameters for the service in the manifest as properties or property groups, or to have configuration parameters in a separate file. The manifest files are in the `/var/svc/manifest` directory tree. Both profiles and manifests are `xml` type files.

The following is an example of the `generic_open` profile:

```
sys-01# more generic_open.xml
<?xml version='1.0'?>
. . .
<service_bundle type='profile' name='generic_open'
         xmlns:xi='http://www.w3.org/2003/XInclude' >
  <!--
      Include name service profile, as set by system id tools.
  -->
  <xi:include href='file:/var/svc/profile/name_service.xml' />

<!--
      svc.startd(1M) services
  -->
  <service name='system/coreadm' version='1' type='service'>
    <instance name='default' enabled='true'/>
  </service>
  <service name='system/cron' version='1' type='service'>
    <instance name='default' enabled='true'/>
  </service>
  <service name='system/cryptosvc' version='1' type='service'>
    <instance name='default' enabled='true'/>
  </service>
. . .
<!--
        inetd(1M) services
  -->
  <service name='network/finger' version='1' type='service'>
    <instance name='default' enabled='true'/>
  </service>
  <service name='network/ftp' version='1' type='service'>
    <instance name='default' enabled='true'/>
  </service>
  <service name='network/login' version='1' type='service'>
    <instance name='rlogin' enabled='true'/>
  </service>
. . .
<!--
        inetd(1M) RPC services
  -->
  <service name='network/rpc/gss' version='1' type='service'>
    <instance name='ticotsord' enabled='true'/>
  </service>
  <service name='network/rpc/keyserv' version='1' type='service'>
    <instance name='default' enabled='true'/>
  </service>
  <service name='network/rpc/mdcomm' version='1' type='service'>
    <instance name='tcp' enabled='true'/>
  </service>
  <service name='network/rpc/meta' version='1' type='service'>
    <instance name='tcp' enabled='true'/>
  </service>
```

The `generic_open` profile contains several sections, as indicated by the comments. Each section lists the services that should be enabled and their instance name. This profile is always read when `svc.startd`(1M) starts.

A manifest is a list of things pertaining to each service. The list will contain the name of the service, the method to start and stop the service, and many other things. All manifests live in the `/var/svc/manifests` directory tree. This directory contains subdirectories that logically group services. For example, the system console will be found under `/var/svc/manifests/system/console-login.xml` and `telnet` is found under `/var/svc/manifests/network/telnet.xml`. The current directories found in the `/var/svc/manifests` directory are:

- `application`
- `device`
- `milestone`
- `network`
- `platform`
- `site`
- `system`

The following is a copy of the `system/coreadm.xml` manifest.

```
<service_bundle type='manifest' name='SUNWcsr:coreadm'>

<service
        name='system/coreadm'
        type='service'
        version='1'>

        <create_default_instance enabled='false' />

        <single_instance />

        <dependency
                name='usr'
                type='service'
                grouping='require_all'
                restart_on='none'>
                <service_fmri value='svc:/system/filesystem/minimal' />
        </dependency>

        <exec_method
                type='method'
                name='start'
                exec='/usr/bin/coreadm -u'
                timeout_seconds='3' />

        <exec_method
                type='method'
                name='stop'
                exec=':true'
                timeout_seconds='0' />

        <property_group name='startd' type='framework'>
                <propval name='duration' type='astring'
```

```
                        value='transient' />
        </property_group>

        <stability value='Unstable' />

        <template>
                <common_name>
                        <loctext xml:lang='C'>
                                System-wide core file configuration
service
                        </loctext>
                </common_name>

                <documentation>
                        <manpage
                                title='coreadm'
                                section='1M'
                                manpath='/usr/share/man' />
                </documentation>
        </template>
</service>

</service_bundle>
```

The use of the tags is as follows:

- `service_bundle` – Tag used to open and close the body of the manifest. The first portion of the name component specifies the package from which this service comes.
- `service` – Tag to specify services available in this manifest. This tag occurs only once in most manifests but may appear more than once (see the `/var/svc/manifests/system/device/devices-local.xml` file).
- `dependency` – Tag used to specify services on which this service is dependent. There may be multiple dependency tags.
- `exec_method` – Tag used to specify a method. A method defines what is used to execute the start and stop of a service. The recommended location of a method is `/lib/svc/method/`*`svc-name`* for integrated products and `/`*`basedir`*`/method/`*`svc-name`* for added applications.

- `property_group` – Tag used to specify values for property groups. Properties are grouped to make it easier to specify only the properties appropriate to the service being defined.

All manifests in the `/var/svc/manifest`s directory tree are read by `svc.startd` as it starts. If new services are found, they are imported into the repository.

Configuration information for services is maintained in the repository. This repository is accessed via the svc.configd daemon or through the use of the API interface. The disk copy of the configuration information is kept in the `/etc/svc/repository.db` file. SMF keeps snapshots of configuration changes so that a change can be backed out using the `svccfg`(1M) command if it does not work. As a last resort backup, the initial repository is kept in the `/lib/svc/seed/global.db` file. This file can be accessed by booting with the `boot -m seed` command (see `kernel`(1M)).

Error logs are found in the `/var/svc/log` directory. This directory contains a file for each service instance which has created log entries. Perhaps the easiest way to search for problems is to search for the words ERROR and WARNING in these log files.

Managing Services

One of the more significant benefits of SMF is your visibility into services and their dependencies. There are mechanisms to:

- Enable or disable service startup
- View and modify a service's dependencies
- View the current state of all services
- View and modify service startup configuration data

The tools responsible for running services and accessing the repository are:

- `svc.startd`(1M) – Responsible for starting and stopping services as requested
- `svc.configd`(1M) – Responsible for accessing the configuration repository
- `inetd`(1M) – Delegated restarter

The tools available for observing and managing services are:

- `svcs`(1) – Show services, their current state, and their dependencies
- `svcprop`(1) – Show property values for services
- `svcadm`(1M) – Manipulate service instances
- `svccfg`(1M) – Import, export and modify service configurations
- `inetadm`(1M) – Observe or configure inetd- controlled services

Changes to the `inetd` Daemon

The `inetd` daemon performs the same function as, but is implemented significantly different from, the daemon of the same name in Solaris 9 and prior Solaris operating system releases. In the current Solaris release, `inetd` is part of SMF and will run only within that facility.

The following is an example of trying to run `inetd` from the command line:

```
# inetd
inetd is now an smf(5) managed service and can no longer be run from the
command line. To enable or disable inetd refer to svcadm(1M) on
how to enable "svc:/network/inetd:default", the inetd instance.

The traditional inetd command line option mappings are:
        -d : there is no supported debug output
        -s : inetd is only runnable from within the SMF
        -t : See inetadm(1M) on how to enable TCP tracing
        -r : See inetadm(1M) on how to set a failure rate

To specify an alternative configuration file see svccfg(1M)
for how to modify the "start/exec" string type property of
the inetd instance, and modify it according to the syntax:
"/usr/lib/inet/inetd [alt_config_file] %m".

For further information on inetd see inetd(1M).
```

The `network/inetd:default` service instance is run by the SMF restarter (`svc.startd`). In turn, `inetd` is the restarter for the network facilities that it has managed in the past. The `inetd` daemon does not read the `inetd.conf` file for configuration information. If there is information in that file that needs to be converted for SMF, use the `inetconv`(1M) command.

The svcs Command

The svcs command displays the current state of system services. Using the svcs command with the -a option shows all services. Without the -a the svcs command shows only services which are running or available to run.

```
sys-01# svcs -a
STATE          STIME    FMRI
legacy_run     Aug_31   lrc:/etc/rcS_d/S10pfil
legacy_run     Aug_31   lrc:/etc/rcS_d/S29wrsmcfg
legacy_run     Aug_31   lrc:/etc/rcS_d/S35cacheos_sh
legacy_run     Aug_31   lrc:/etc/rcS_d/S41cachefs_root
legacy_run     Aug_31   lrc:/etc/rcS_d/S55fdevattach
legacy_run     Aug_31   lrc:/etc/rc2_d/S10lu
legacy_run     Aug_31   lrc:/etc/rc2_d/S20sysetup
. . .
disabled       Aug_31   svc:/platform/sun4u/mpxio-upgrade:default
disabled       Aug_31   svc:/network/dns/client:default
disabled       Aug_31   svc:/network/ldap/client:default
disabled       Aug_31   svc:/network/nis/client:default
disabled       Aug_31   svc:/network/nis/server:default
disabled       Aug_31   svc:/network/rpc/nisplus:default
disabled       Aug_31   svc:/network/dns/server:default
disabled       Aug_31   svc:/network/inetd-upgrade:default
disabled       Aug_31   svc:/platform/sun4u/sf880drd:default
disabled       Aug_31   svc:/system/consadm:default
disabled       Aug_31   svc:/application/print/cleanup:default
disabled       Aug_31   svc:/application/print/server:default
. . .
online         Aug_31   svc:/system/svc/restarter:default
online         Aug_31   svc:/milestone/name-services:default
online         Aug_31   svc:/network/loopback:default
online         Aug_31   svc:/network/initial:default
online         Aug_31   svc:/network/physical:default
online         Aug_31   svc:/network/service:default
online         Aug_31   svc:/network/ssh:default
online         Aug_31   svc:/milestone/devices:default
online         Aug_31   svc:/system/device/local:default
online         Aug_31   svc:/system/filesystem/minimal:default
online         Aug_31   svc:/network/rpc/bind:default
. . .
online         Aug_31   svc:/network/telnet:default
online         17:03:46 svc:/network/smtp:sendmail
offline        Aug_31   svc:/application/print/ipp-listener:default
```

To produce its output, the `svcs` command queries the configuration repository and retrieves the name and current state of each service, and the time it was started. Notice that only services started within the past 24 hours show the actual time stamp.

The `svcs` command has a `-p` option that allows you to see the processes that are associated with a service. The following example uses a pattern match to specify the services to display.

```
sys-01# svcs -p "*nfs*"
STATE          STIME    FMRI
disabled       Aug_31   svc:/network/nfs/rquota:ticlts
disabled       Aug_31   svc:/network/nfs/rquota:udp
online          8:52:05 svc:/network/nfs/server:default
                8:52:03     2389 statd
                8:52:03     2391 lockd
                8:52:03     2393 mountd
                8:52:03     2395 nfsd
                8:52:04     2397 nfsmapid
online          8:53:48 svc:/network/nfs/client:default
```

SMF also makes it easier to view the dependencies among various services. In earlier versions of Solaris, this was basically impossible without access to the service source code and a significant amount of time. SMF requires each service to describe its dependencies on other services, explicitly using service identifier strings. The `-d` option of the `svcs` command shows the services on which a service instance depends. The `-D` option shows the services which depend on a service instance.

The following example shows the services which depend on the `/system/filesystem/minimal:default` instance.

```
sys-01# svcs -d /system/filesystem/minimal:default
STATE          STIME    FMRI
online         Aug_31   svc:/system/device/local:default
online         Aug_31   svc:/system/filesystem/usr:default
```

The following example shows the services on which the `/system/filesystem/minimal:default` instance depends.

```
sys-01# svcs -D /system/filesystem/minimal:default
STATE          STIME    FMRI
online         Aug_31   svc:/system/cryptosvc:default
online         Aug_31   svc:/system/sysidtool:net
online         Aug_31   svc:/system/sysidtool:system
```

Being able to list dependencies of a service is very useful in troubleshooting service failures as well as helping to understand the consequences of taking a service down.

To see all configuration information about a service instance, use the `-l` option of the `svcs` command.

```
sys-01# svcs -l system/filesystem/minimal:default
fmri         svc:/system/filesystem/minimal:default
enabled      true
state        online
next_state   none
restarter    svc:/system/svc/restarter:default
dependency   require_all/none svc:/system/device/local (online)
dependency   require_all/none svc:/system/filesystem/usr (online)
```

The svcprop Command

The `svcprop` command allows you to see the properties associated with a service instance. The following example shows the properties for the syslog default instance.

```
sys-01# svcprop svc:/system/system-log:default
general/package astring SUNWcsr
general/enabled boolean true
restarter/contract count 41
restarter/start_pid count 593
restarter/auxiliary_state astring none
restarter/next_state astring none
restarter/state astring online
restarter/state_timestamp time 1093965480.562821000
restarter_actions/refresh integer
```

Specifying the service instead of the instance shows additional properties associated with the service.

```
sys-01# svcprop system/system-log
milestone/entities fmri svc:/milestone/single-user
milestone/grouping astring require_all
milestone/restart_on astring none
milestone/type astring service
dependents/system-log_single-user astring svc:/milestone/multi-user
general/entity_stability astring Unstable
general/single_instance boolean true
stop/exec astring :kill
stop/timeout_seconds count 3
```

```
stop/type astring method
start/exec astring /lib/svc/method/system-log
start/timeout_seconds count 3
start/type astring method
tm_man_syslogd/manpath astring /usr/share/man
tm_man_syslogd/section astring 1M
tm_man_syslogd/title astring syslogd
tm_common_name/C ustring system log
```

The `svcprop` command allows you to look at certain groups of properties by the use of the `-p` option. The following example shows the general properties for the spray service.

```
sys-01# svcprop -p general network/rpc/spray
general/entity_stability astring Unstable
general/restarter fmri svc:/network/inetd:default
```

The `svcadm` Command

The `svcadm` command is used to manipulate the state of services and to specify the milestone to which the machine should be brought. The subcommands of the `svcadm` command are:

- `enable` – Enable the specified service instance
- `disable` – Disable the specified service instance
- `restart` – Stop and then start the specified service instance
- `refresh` – Have the specified service instance re-read its configuration information
- `mark` – Assign the specified service instance to the specified state (`degraded` or `maintenance`)
- `restore` – Restore a service instance from its previous `degraded` or `maintenance` state
- `delegate` – Assign a new restarter for the specified service instance
- `milestone` – Restrict the set of services to those between the beginning of the graph and the specified milestone

When a service is disabled, all dependent services are also disabled. The `svcs -D` command can be used to see the impact of disabling a service.

```
# svcadm disable internet/http:apache
```

The disable setting not only persists across reboots, but also across software upgrades and patch installation. Use this command to disable any Solaris service.

A service is enabled using the `svcadm enable` command. Use the `-r` option in order to enable a service and all of its dependencies. To enable `sar` performance recording do the following:

```
# svcadm enable system/sar
```

To verify that the service is in fact running, look at the service with the `svcs` command.

```
# svcs -l system/sar:default
fmri         svc:/system/sar:default
enabled      true
state        online
next_state   none
restarter    svc:/system/svc/restarter:default
dependency   require_all/none svc:/system/filesystem/minimal (online)
```

The `milestone` subcommand is used to specify the milestone to which the system will change. This is basically the replacement for the `init` *n* command. This is a permanent change unless the `-t` (for temporary) option is used. The `milestone -t` option is frequently used to debug systems which are not booting properly.

```
# svcadm milestone -t all
```

Once the above command is running, the `svcs` command can be used to follow the progress of services being brought on-line.

The svccfg Command

The `svccfg`(1M) command can be used to either browse the SMF repository interactively or run a set of commands from a command file. An example of running the `svccfg` command interactively follows.

After starting the `svccfg` utility, the `list` subcommand prints a list of the service identifiers for all services installed on the system:

```
    example% svccfg
    svc:> list
    system/console-login
    milestone/devices
    system/device/local
    system/identity
    system/filesystem/local
    system/manifest-import
    system/filesystem/minimal
    milestone/multi-user-server
    milestone/multi-user
    milestone/name-services
    network/initial
    network/loopback
    network/physical
    system/svc/restarter
    system/filesystem/root
    milestone/single-user
    system/filesystem/usr
    network/rpc/bind
    network/inetd-upgrade
    system/utmp
    system/metainit
    system/mdmonitor
    smf/manifest
    ...
```

The `select` command identifies a service on which future `svccfg` commands should operate, similar to the concept of a shell's current working directory. SMF also supports multiple active instances of the same service on a single system, so you can use `svccfg` on service instance identifiers as well. The following examples use services that have only a single instance named `default`. Type the following commands to select the name service cache and list its instances.

```
    svc:> select system/name-service-cache
    svc:/system/name-service-cache> list
    :properties
    default
```

Notice the list contains not only the default instance but also the `:properties` value. The presence of this string in the `list` output identifies that there are properties related to the currently selected FMRI.

Type the `listprop` command to list the SMF properties associated with the name service cache:

```
svc:/system/name-service-cache> listprop
usr                          dependency
```

```
usr/entities               fmri     svc:/system/filesystem/usr
usr/grouping               astring  require_all
usr/restart_on             astring  none
usr/type                   astring  service
config_data                dependency
config_data/entities       fmri     file://localhost/etc/nscd.conf
file://localhost/etc/nsswitch.conf
config_data/grouping       astring  require_all
config_data/restart_on     astring  restart
config_data/type           astring  path
general                    framework
general/entity_stability   astring  Unstable
general/single_instance    boolean  true
stop                       method
stop/exec                  astring  :kill
stop/timeout_seconds       count    3
stop/type                  astring  method
start                      method
start/exec                 astring  /lib/svc/method/svc-nscd
start/timeout_seconds      count    30
start/type                 astring  method
tm_man_nscd                template
tm_man_nscd/manpath        astring  /usr/man
tm_man_nscd/section        astring  1M
tm_man_nscd/title          astring  nscd
tm_common_name             template
tm_common_name/C           ustring  "Name service cache daemon"
general                     framework
general/package             astring  SUNWcsr
general/enabled             boolean  true
restarter                   framework    NONPERSISTENT
restarter/contract          count    180
restarter/start_pid         count    2430
restarter/auxiliary_state   astring  none
restarter/next_state        astring  none
restarter/state             astring  online
restarter/state_timestamp   time     1094137041.968560000
restarter_actions           framework    NONPERSISTENT
restarter_actions/refresh   integer
```

You can modify a single property using the `setprop` command. For example, to set the start method timeout to 15 seconds, type:

```
svc:/system/name-service-cache> setprop start/timeout_seconds = 15
```

The property names, values, and meanings are explained in further detail in the SMF System Administration Guide documentation. You can also use the `editprop` command to edit groups of properties in your preferred text editor. SMF automatically stores a persistent snapshot of the changes made to the current configuration to serve as backup copy of your changes and to permit administrators to undo any configuration mistakes. The `listsnap` subcommand can be used to list configuration snapshots associated with the service instance:

```
svc:/system/name-service-cache> select default
svc:/system/name-service-cache:default> listsnap
initial
running
start
```

The snapshot of the current configuration used by the active service instance is shown in the list and is named `running`. The snapshot named `initial` is the initial system state immediately after install. To undo configuration changes, you can use the `revert` command to restore an earlier snapshot.

When you execute an undo operation with the `revert` command, SMF automatically restores your configuration settings and then starts, restarts, and stops services based on the new settings immediately and automatically.

The `inetadm` Command

The `inetadm`(1M) command allows observation and configuration of inetd-controlled services (services with `inetd` as the restarter). The capabilities of `inetadm` are a combination of the `svcs` command, the `svcadm` command, and the `svccfg` command.

The `inetadm` command with no arguments lists all the services under the control of the `inetd` daemon.

```
# inetadm
ENABLED   STATE          FMRI
disabled  disabled       svc:/network/rpc/ocfserv:default
disabled  disabled       svc:/network/lp:default
enabled   online         svc:/network/rpc/mdcomm:tcp
disabled  disabled       svc:/network/rpc/mdcomm:tcp6
enabled   online         svc:/network/rpc/meta:tcp
disabled  disabled       svc:/network/rpc/meta:tcp6
enabled   online         svc:/network/rpc/metamed:tcp
disabled  disabled       svc:/network/rpc/metamed:tcp6
```

```
enabled   online         svc:/network/rpc/metamh:tcp
disabled  disabled       svc:/network/rpc/metamh:tcp6
disabled  disabled       svc:/network/tname:default
enabled   online         svc:/network/security/ktkt_warn:ticotsord
enabled   online         svc:/network/telnet:default
enabled   online         svc:/network/rpc/smserver:default
enabled   online         svc:/network/rpc/gss:ticotsord
disabled  disabled       svc:/network/rpc/rex:tcp
disabled  disabled       svc:/network/uucp:default
disabled  disabled       svc:/network/chargen:dgram
disabled  disabled       svc:/network/chargen:stream
disabled  disabled       svc:/network/daytime:dgram
disabled  disabled       svc:/network/daytime:stream
. . .
```

The `-l` option of the `inetadm` command allows you to see all the properties for a particular service. Those values preceded by "default" are values inherited from the `inetd` service.

```
# inetadm -l network/telnet:default
SCOPE    NAME=VALUE
         name="telnet"
         endpoint_type="stream"
         proto="tcp6"
         isrpc=FALSE
         wait=FALSE
         exec="/usr/sbin/in.telnetd"
         user="root"
default  bind_addr=""
default  bind_fail_max=-1
default  bind_fail_interval=-1
default  max_con_rate=-1
default  max_copies=-1
default  con_rate_offline=-1
default  failrate_cnt=40
default  failrate_interval=60
default  inherit_env=TRUE
default  tcp_trace=FALSE
default  tcp_wrappers=FALSE
```

Services can be enabled and disabled with the `-e` and `-d` options of the inetadm command respectively. The following is an example of enabling the services to allow the `rdate` command to work.

```
# rdate localhost
rdate: connect: Connection refused
# inetadm -e network/time:dgram
# inetadm -e network/time:stream
# rdate localhost
Thu Sep  2 16:18:59 2004
```

The `-p` option of the `inetadm` command shows the service property values provided by the inetd service.

```
# inetadm -p
NAME=VALUE
bind_addr=""
bind_fail_max=-1
bind_fail_interval=-1
max_con_rate=-1
max_copies=-1
con_rate_offline=-1
failrate_cnt=40
failrate_interval=60
inherit_env=TRUE
tcp_trace=FALSE
tcp_wrappers=FALSE
```

It is also possible to modify the properties of the `inetd` service and any service that is inetd-controlled.

Troubleshooting

A common problem experienced by users new to SMF is the diagnosis of failure of a service to start either automatically at boot time or manually.

Debugging a Hang on Boot

To debug a system hang on boot, use the `-m` option of the `boot` command. For this type of problem specify `milestone=none` as the `-m` option (see `kernel`(1M)).

```
{1} ok boot -m milestone=none
Resetting ...

screen not found.
Can't open input device.
Keyboard not present.  Using ttya for input and output.

Sun Enterprise 420R (3 X UltraSPARC-II 450MHz), No Keyboard
OpenBoot 3.29, 1024 MB memory installed, Serial #16241000.
Ethernet address 8:0:20:f7:d1:68, Host ID: 80f7d168.
```

```
Rebooting with command: boot -m milestone=none
Boot device: /pci@1f,4000/scsi@3/disk@0,0:a  File and args: -m
milestone=none
SunOS Release 5.10 Version s10_64 64-bit
Copyright 1983-2004 Sun Microsystems, Inc.  All rights reserved.
Use is subject to license terms.
Requesting System Maintenance Mode

Type control-d to proceed with normal startup,
(or give root password for system maintenance):
```

Once you receive the sulogin prompt, log in with the root password. This brings the system to a console prompt with no services running.

```
single-user privilege assigned to /dev/console.
Entering System Maintenance Mode

Jul 28 11:53:07 su: ’su root’ succeeded for root on /dev/console
Sun Microsystems Inc.   SunOS 5.10      s10_64  May 2004
# svcs -a
STATE          STIME    FMRI
disabled       12:18:28 svc:/milestone/single-user:default
disabled       12:18:28 svc:/network/initial:default
disabled       12:18:28 svc:/network/loopback:default
disabled       12:18:28 svc:/network/physical:default
disabled       12:18:28 svc:/network/rpc/bind:default
disabled       12:18:28 svc:/system/device/local:default
disabled       12:18:28 svc:/system/filesystem/local:default
disabled       12:18:28 svc:/system/filesystem/minimal:default
disabled       12:18:28 svc:/system/filesystem/root:default
disabled       12:18:28 svc:/system/filesystem/usr:default
disabled       12:18:28 svc:/system/identity:domain
. . .
```

Next, you use the `svcadm` command with the `-t all` option to specify that all services should be started. The `all` milestone is a special one meaning all services possible. The `-t` option specifies to make the milestone change temporary.

```
# svcadm milestone -t all
# Configuring devices.
```

Progress of the service startup can be watched with the `svcs` command.

```
# svcs
STATE          STIME    FMRI
online         11:52:41 svc:/system/svc/restarter:default
online         11:54:05 svc:/network/loopback:default
online         11:54:05 svc:/system/filesystem/root:default
online         11:54:07 svc:/system/filesystem/usr:default
online         11:54:16 svc:/network/physical:default
online         11:54:17 svc:/system/identity:node
online         11:54:19 svc:/network/initial:default
online         11:54:19 svc:/network/service:default
online         11:54:23 svc:/milestone/devices:default
online         11:54:23 svc:/system/device/local:default
online         11:54:23 svc:/system/filesystem/minimal:default
online         11:54:23 svc:/system/sysevent:default
online         11:54:24 svc:/milestone/name-services:default
online         11:54:24 svc:/network/dns/client:default
online         11:54:24 svc:/network/ntp:default
online         11:54:24 svc:/system/manifest-import:default
online         11:54:24 svc:/system/rmtmpfiles:default
offline        11:54:04 svc:/milestone/multi-user:default
offline        11:54:04 svc:/milestone/single-user:default
offline        11:54:04 svc:/network/rpc/bind:default
. . .
```

Notice that the `milestone/multi-user` service is offline. To determine why, look at the dependencies for this service.

```
# svcs -l svc:/milestone/single-user:default
fmri         svc:/milestone/single-user:default
enabled      true
state        offline
next_state   none
restarter    svc:/system/svc/restarter:default
dependency   require_all/none svc:/system/sysidtool:net (offline)
svc:/system/sysidtool:system (offline)
dependency   optional_all/none svc:/network/physical (online)
dependency   require_any/none svc:/network/loopback (online)
dependency   require_all/none svc:/system/manifest-import (online)
dependency   require_all/none svc:/system/filesystem/minimal (online)
dependency   require_all/none svc:/system/identity:node (online)
dependency   require_all/none svc:/system/sysevent (online)
dependency   optional_all/none svc:/system/metainit (offline)
```

The above output shows that all dependencies are met. The next step is to look for errors in the error logs in the `/var/svc/log` directory.

Using Debug Mode

SMF can be put in a debug mode by using the `boot -m debug` command. This causes SMF to start all services serially and display messages on the console for all services.

```
Executing last command: boot -m debug
Boot device: /pci@1f,0/pci@1/scsi@8/disk@0,0:a  File and args: -m debug
SunOS Release 5.10 Version s10_66 64-bit
Copyright 1983-2004 Sun Microsystems, Inc.  All rights reserved.
Use is subject to license terms.
-
INIT: Executing svc.startd
Sep  3 08:04:00/1: Initialized restarter protocol
Sep  3 08:04:00/1: Initialized restarter
Sep  3 08:04:00/1: Initialized graph
Sep  3 08:04:00/6: Graph adding svc:/system/console-login:default.
Sep  3 08:04:00/6: Graph engine: Refreshing svc:/system/console-
login:default.
Sep  3 08:04:00/6: Graph adding svc:/system/sysidtool:net.
Sep  3 08:04:00/6: Graph engine: Refreshing svc:/system/sysidtool:net.
Sep  3 08:04:00/6: Graph adding svc:/system/identity:node.
Sep  3 08:04:00/6: Graph engine: Refreshing svc:/system/identity:node.
Sep  3 08:04:00/3: svc:/system/console-login:default is a wait-style
service
Sep  3 08:04:00/3: svc:/system/console-login:default: inserted instance
into restarter list
Sep  3 08:04:00/3: svc:/system/sysidtool:net is a transient-style service
Sep  3 08:04:00/3: svc:/system/sysidtool:net: inserted instance into
restarter list
Sep  3 08:04:00/3: svc:/system/identity:node is a transient-style service
Sep  3 08:04:00/3: svc:/system/identity:node: inserted instance into
restarter list
Sep  3 08:04:00/6: Graph adding svc:/network/physical:default.
Sep  3 08:04:00/6: Graph engine: Refreshing
svc:/network/physical:default.
Sep  3 08:04:00/6: Enabling svc:/network/physical:default.
Sep  3 08:04:00/6: Graph adding svc:/network/loopback:default.
Sep  3 08:04:00/6: Graph engine: Refreshing
svc:/network/loopback:default.
Sep  3 08:04:00/6: Enabling svc:/network/loopback:default.
Sep  3 08:04:00/6: Enabling svc:/system/identity:node.
Sep  3 08:04:00/6: Graph adding svc:/system/identity:domain.
Sep  3 08:04:00/6: Graph engine: Refreshing svc:/system/identity:domain.
Sep  3 08:04:00/6: Graph adding svc:/system/filesystem/minimal:default.
. . .
```

```
Sep  3 08:07:37/9: Propagating start of svc:/system/zones:default.
Sep  3 08:07:37/3: svc:/system/zones:default: trying to start instance
Sep  3 08:07:37/3: svc:/system/zones:default: start_instance -> is
already started
Sep  3 08:07:39/54: svc:/network/inetd:default: state updates for
svc:/network/rpc/smserver:default (5, 0)
Sep  3 08:07:39/9: Graph noting svc:/network/rpc/smserver:default online
-> online.
Sep  3 08:08:21/54: svc:/network/inetd:default: state updates for
svc:/network/telnet:default (5, 0)
Sep  3 08:08:21/9: Graph noting svc:/network/telnet:default online ->
online.
Sep  3 08:08:27 sys-01 login: ROOT LOGIN /dev/pts/1 FROM gateway
```

This approach is similar to putting `sh -x` in all of the `rc*.d` scripts. The console shows all the processing done by SMF. If this is done on a problem system, errors will show.

Debugging a Service

Here is an example of troubleshooting the `lpsched` service where it is failing to start with the command;

```
sys-02# svcadm enable /application/print/server
```

After running the previous command the service still shows disabled.

```
sys-02# svcs /application/print/server
STATE          STIME    FMRI
disabled       11:14:24 svc:/application/print/server:default
```

The first step would be to determine if all the dependencies are met. To do this, use the command;

```
sys-02# svcs -d /application/print/server:default
STATE          STIME    FMRI
sys-02
```

Since the command returned no dependencies, there is no need to check for services running that the print server service might require. This also means that the root of the problem lies with `svc.startd` not starting the service. If someone had made bad changes to `/application/print/server` we can revert to the last good known running state.

```
sys-02# svccfg
svc:> select /application/print/server:default
```

```
svc:/application/print/server:default> listsnap
initial
running
svc:/application/print/server:default>
```

This shows us that we could revert to the initial configuration for this service.

```
svc:/application/print/server:default> revert initial
svc:/application/print/server:default> listsnap
initial
running
previous
svc:/application/print/server:default>
```

Now try to start the service.

```
sys-02# svcadm -v enable /application/print/server
/application/print/server enabled.
sys-02# svcs /application/print/server
STATE          STIME    FMRI
online         11:43:50 svc:/application/print/server:default
sys-02#
```

The svcs command now shows that the service is running. The problem is fixed. If the print server still had not started, the error logs should be searched for problems.

```
sys-02# more /var/svc/log/application-print-server:default.log
Aug 25 11:43:50 Executing start method ("/lib/svc/method/print-server
start")
Print services started.
sys-02#
```

You can also use the following command to check for additional errors. The `-l` option to svcs will list the status of the FMRI. Any error or complaints from `svc.startd` will be reported here.

```
sys-02# svcs -l /application/print/server:default
fmri         svc:/application/print/server:default
enabled      true
state        online
next_state   none
restarter    svc:/system/svc/restarter:default
contract_id  122
sys-02#
```

Repository Problems

There are two types of problems that can occur with the repository. The repository can be corrupted or it can be inaccessible. The following is an example of an inaccessible repository:

```
# svccfg
svc:> select network/nfs/client
svccfg: Could not connect to repository server: repository server
unavailable.
```

The repository server is the `svc.configd` deamon. Either the `svc.configd` daemon is not running or the svc.startd daemon is not running. Look at the state of the `system/svc/restarter:default` service and the error logs for this service.

If the repository becomes unusable, you can restore the repository from backup data or copy in the initial seed repository and reboot. The seed is a bare bones `repository.db` file located in the `/lib/svc/seed/global.db` file. When the machine is booted it will normally come to single user mode if the repository is corrupted. At that point you should perform the following commands:

```
# mount -o remount /
# cp /lib/svc/seed/global.db /etc/repository.db
```

Section III: Dynamic Tracing With DTrace

Objectives

Upon completion of this section, you should be able to:

- Describe the fundamental concepts of DTrace
- Describe how to use DTrace

Module 6

DTrace Fundamentals

Objectives

Upon completion of this module, you should be able to:

- Describe the features of the Solaris Dynamic Tracing (DTrace) facility
- Describe the DTrace architecture
- List and enable probes, and create action statements and D scripts

Relevance

Discussion – The following questions are relevant to understanding DTrace:

- Would it be beneficial to be able to turn on trace points for any one of the majority of functions in the kernel?
- Would it be useful to know who is issuing `kill`(2) system calls?

Additional Resources

Additional resources – The following references provide additional information on the topics described in this module:

- *Solaris Dynamic Tracing Guide* (Beta). Sun Microsystems, Inc., part number 817-6223-05.
- Cantrill Bryan M., Michael W. Shapiro, and Adam H. Leventhal. "Dynamic Instrumentation of Production Systems." Draft paper accepted for presentation at 2004 USENIX Conference.
- The `dtrace(1M)` manual page.

DTrace Features

DTrace is a comprehensive dynamic tracing facility that is bundled into the Solaris™ 10 Operating System (Solaris 10 OS). It is intended for use by system administrators, service support personnel, kernel developers, application program developers, and users who are given explicit access permission to the DTrace facility

DTrace has the following features:

- Enables dynamic modification of the system to record *arbitrary* data
- Promotes tracing on *live* systems
- Is completely safe—its use cannot induce fatal failure
- Allows tracing of both the kernel program and user-level programs
- Functions with low overhead when tracing is enabled and zero overhead when tracing is not being performed.

Transient Failures

DTrace provides answers to the causes of transient failures. A transient failure is any unacceptable behavior that does not result in fatal failure of the system. The customer might report a clear, specific failure, such as:

- "Why is `read`(2) returning `EIO` `errno` values on a device that is not reporting any errors?"
- "Our application occasionally does not receive its timer signal."
- "Why is one of our threads missing a condition variable wakeup?"

Or the transient failure can be based on the customer's definition of "unacceptable" system operation:

- "We were expecting to accommodate 100 users per CPU, but we cannot support more than 60 users per CPU."
- "Why does system time go way up when I run application 'X'?"
- "Every morning between 9:30 a.m. and 10:00 a.m. the system is a real dog."

In these situations, you must understand the problem and either eliminate the performance inhibitors or reset the customer's expectations. Eliminating the performance inhibitors could involve:

- Adding more resources, such as memory or central processing units (CPUs)
- Re-configuring existing resources, for example, tuning parameters or re-writing software
- Lessening the load

How to Debug Transient Failures

DTrace was developed to provide a more efficient and cost-effective method of diagnosing transient failures. Historically users have debugged transient failures using process-centric tools such as `truss`(1), `pstack`(1), or `prstat`(1M). These tools were not designed to debug systemic problems. The tools that were intended for debugging systemic problems, such as `mdb`(1) and Solaris Crash Analysis Tool (Solaris CAT) are designed for postmortem analysis.

Debugging Using Postmortem Analysis

You can use postmortem analysis to debug transient problems by inducing fatal failure during the period of transient failure. This technique has the following disadvantages:

- Requires inducing fatal failure, which nearly always results in more downtime than the transient failure
- Requires solving a dynamic problem from a static snapshot of the system's state

Debugging Using Invasive Techniques

If existing tools cannot find the root cause of a transient failure, then you must use more invasive techniques. Typically this means developing custom instrumentation for the failing user program, the kernel, or both. This can involve using the Trace Normal Form (TNF) facility. The customer then reproduces the problem using the instrumented binaries. This technique requires:

- Running the instrumented binaries in production
- Reproducing a transient problem in a development environment

Such invasive techniques are undesirable because they are slow, error-prone, and often ineffective.

Relying on the existing static TNF trace points found in the kernel, which you can enable with the `prex`(1) command, is also unsatisfactory. The number of TNF trace points in the kernel is limited and the overhead is substantial.

DTrace Capabilities

The DTrace framework allows you to enable tens of thousands of tracing points called *probes*. When these instrumentation points are hit, you can display arbitrary data in the kernel (or user process).

An example of a probe provided by the DTrace framework is entry into any kernel function. Information that you can display when this probe *fires* includes:

- Any argument to the function
- Any global variable in the kernel
- A nanosecond timestamp of when the function was called
- A stack trace to indicate what code called this function
- The process that was running when the function was called
- The thread that made the call to this function

Using DTrace, you can explore all aspects of the Solaris 10 OS to:

- Understand how the software works
- Determine the root cause of performance problems
- Examine all layers of software sequentially from the user level to the kernel
- Track down the source of aberrant behavior

DTrace comes with powerful data management primitives to eliminate the need for postprocessing of gathered data. Unwanted data is pruned as close to the source as possible to avoid the overhead of generating and later filtering unwanted data.

DTrace also provides a mechanism to trace during boot and to retrieve all traced data from a kernel crash dump.

DTrace Architecture

DTrace helps you understand a software system by enabling you to dynamically modify the operating system kernel and user processes to record additional data that you specify at locations of interest called *probes*.

Probes and Probe Providers

A probe is a program location or activity—for example, every system clock tick—to which DTrace can bind a request to perform a set of *actions*, such as recording a stack trace, a timestamp, or the argument to a function.

How Probes Work

Probes are like programmable sensors inserted at strategic points of your Solaris 10 OS. You use DTrace to program the appropriate sensors to record the information that you want. As each probe fires, DTrace gathers the data from your probes and reports it back to you. If you do not specify any actions for a probe, DTrace simply records each time the probe fires and on what CPU.

DTrace provides tens of thousands of probes of various types. Probes are implemented by *probe providers*. A provider is a kernel module that enables a requested probe to fire when it is hit. An example of a provider is the “function boundary tracing” or `fbt` provider. It provides entry and return probes for almost every function in every kernel module.

How Probes Are Enabled

You define probes and actions using a programming language called D, which is based on the C programming language. Usually D programs are placed in script files ending in a `.d` suffix. The D programs are passed to a DTrace *consumer*. The primary, generic DTrace consumer is the `dtrace`(1M) command.

The user-specified D program is compiled by the DTrace consumer into a form referred to as D Intermediate Format (DIF), which is then sent to the DTrace framework within the kernel for execution. There, the probes that are named within the D program are enabled, and the corresponding provider performs the instrumentation required to activate them.

DTrace Components

DTrace has the following components: probes, providers, consumers, and the D programming language. The entire DTrace framework resides in the kernel. Consumer programs access the DTrace framework through a well-defined application programming interface (API).

Probes

A *probe* has the following attributes:

- It is made available by a *provider*.
- It identifies the *module* and *function* that it instruments.
- It has a *name*.

These four attributes define a *4-tuple* that uniquely identifies each probe:

```
provider:module:function:name
```

In addition, DTrace assigns a unique integer identifier to each probe.

Providers

A provider represents a methodology for instrumenting the system. Providers make probes available to the DTrace framework. A provider receives information from DTrace regarding when a probe is to be enabled and transfers control to DTrace when an enabled probe is hit

DTrace offers the following providers:

- The *function boundary tracing* (`fbt`) provider can dynamically trace the *entry* and *return* of every function in the kernel.
- The *syscall* provider can dynamically trace the entry and return of every Solaris system call.
- The *lockstat* provider can dynamically trace the kernel synchronization primitives.
- The *sched* provider can dynamically trace key scheduling events
- The *profile* provider enables you to add a configurable-rate timer interrupt to the system.
- The *dtrace* provider enables pre-processing and post-processing (as well as D program error-processing) capabilities.

- The *pid* provider enables function boundary tracing within a process as well as tracing of any instruction in the virtual address space of the process.
- The *vminfo* provider makes available probes that correspond to the kernel's virtual memory statistics.
- The *sysinfo* provider makes available probes that correspond to the kernel's "sys" statistics.
- The *proc* provider makes available probes that pertain to process and thread creation and termination as well as signals.
- The *mib* provider makes available probes that correspond to counters in the Solaris management information bases (MIBs), which are used by the simple network management protocol (SNMP).
- The *io* provider makes available probes giving details related to device input and output (I/O).
- The *fpuinfo* provider makes available probes that correspond to the simulation of floating point instructions on SPARC®-based microprocessors.

Note – You should check the *Solaris Dynamic Tracing Guide*, part number 817-6223, regularly for the addition of any new DTrace providers.

Consumers

A DTrace consumer is a process that interacts with DTrace. There is one main DTrace consumer called `dtrace`(1M). It acts as a generic front-end to the DTrace facility. Most other consumers are re-writes of previously existing utilities such as `lockstat`(1M).

There is no limit on the number of concurrent consumers. That is, many users can simultaneously run the `dtrace`(1M) command. DTrace handles the multiplexing.

D Programming Language

The D programming language enables you to specify probes of interest and bind actions to those probes. To do this, you construct scripts called D scripts. The nature of D scripts is similar to `awk`(1)'s "pattern action" pairs. The D programming language also borrows heavily from the C programming language. For safety, the D language does NOT support any of the C language control flow statements such as *while* loops or *if* statements.

Even if you have no experience with the C programming language or with `awk`(1), D programs are fairly easy to write and understand.

Features of the D language include the following:

- Enables complete access to kernel C types, such as `vnode_t`
- Provides complete access to kernel static and global variables
- Provides complete support for American National Standards Institute (ANSI)-C operators
- Supports *strings* as a built-in type (unlike C, which uses the ambiguous `char *` or `char[]` types).

Architecture Summary

To summarize, the DTrace facility consists of user-level consumer programs such as `dtrace`(1M), providers packaged as kernel modules, and a library interface for the consumer programs to access the DTrace facility through the `dtrace`(7D) kernel driver.

Figure 6-1 shows the overall DTrace architecture.

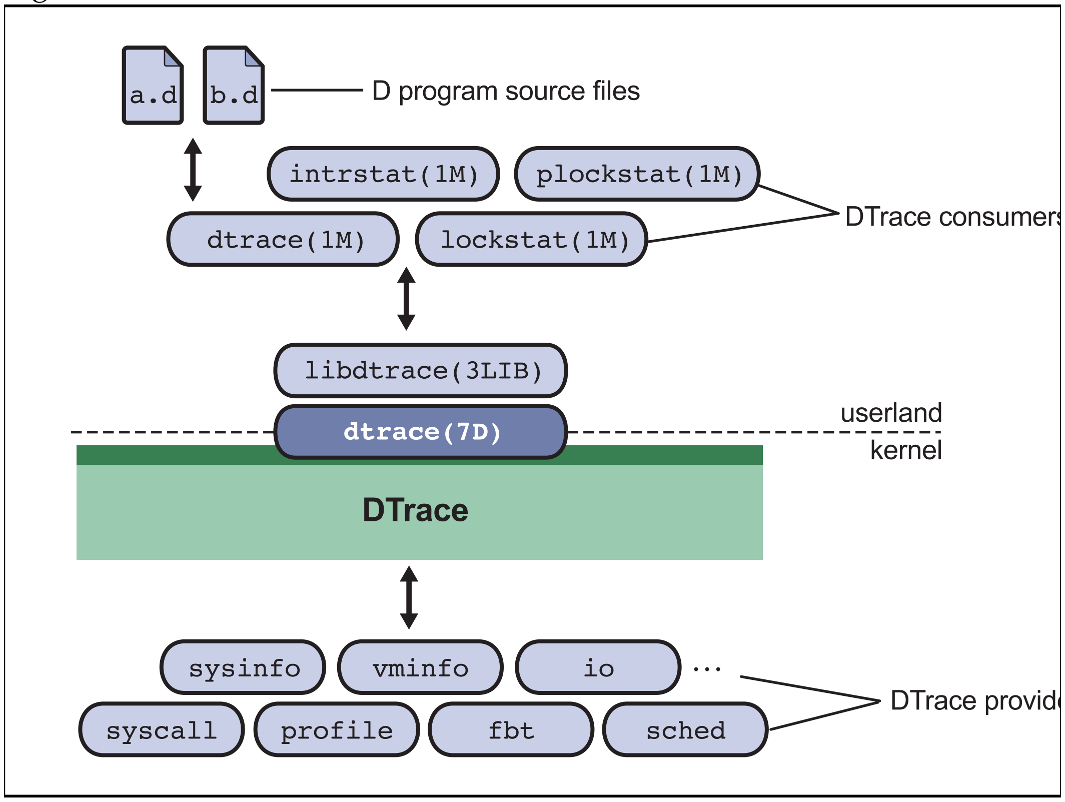


Figure 6-1 DTrace Architecture

DTrace Tour

In this section you tour the DTrace facility and learn to perform the following tasks:

- List the available probes using various criteria:
    - Probes associated with a particular function
    - Probes associated with a particular module
    - Probes with a specific name
    - All probes from a specific provider
- Explain how to enable probes
- Explain default probe output
- Describe action statements
- Create a simple D script

Listing Probes

You can list all DTrace probes with the `-l` option of the `dtrace`(1M) command:

```
# dtrace -l
  ID   PROVIDER            MODULE                          FUNCTION NAME
   1     dtrace                                                     BEGIN
   2     dtrace                                                     END
   3     dtrace                                                     ERROR
   4    syscall                                               nosys entry
   5    syscall                                               nosys return
   6    syscall                                               rexit entry
   7    syscall                                               rexit return
   8    syscall                                             forkall entry
   9    syscall                                             forkall return
  10    syscall                                                read entry
  11    syscall                                                read return
  12    syscall                                               write entry
  13    syscall                                               write return
  14    syscall                                                open entry
  15    syscall                                                open return
...
```

You can use an additional option to list specific probes, as follows:

- In a specific function: `-f` *`function`*

```
# dtrace -l -f cv_wait
   ID   PROVIDER            MODULE                          FUNCTION NAME
12126        fbt           genunix                           cv_wait entry
12127        fbt           genunix                           cv_wait return
...
```

- In a specific module: `-m` *`module`*

```
# dtrace -l -m sd
   ID   PROVIDER            MODULE                          FUNCTION NAME
18164        fbt                sd                            sdopen entry
18165        fbt                sd                            sdopen return
18166        fbt                sd                           sdclose entry
18167        fbt                sd                           sdclose return
18168        fbt                sd                        sdstrategy entry
18169        fbt                sd                        sdstrategy return
...
```

- With a specific name: `-n` *`name`*

```
# dtrace -l -n BEGIN
   ID   PROVIDER            MODULE                          FUNCTION NAME
    1     dtrace                                                     BEGIN
```

- From a specific provider: `-P` *`provider`*

```
# dtrace -l -P lockstat
   ID   PROVIDER            MODULE                          FUNCTION NAME
  467   lockstat           genunix                       mutex_enter adaptive-acquire
  468   lockstat           genunix                       mutex_enter adaptive-block
  469   lockstat           genunix                       mutex_enter adaptive-spin
  470   lockstat           genunix                        mutex_exit adaptive-release
  471   lockstat           genunix                     mutex_destroy adaptive-release
  472   lockstat           genunix                    mutex_tryenter adaptive-acquire
...
```

- Realize that a specific function or module can be supported by many providers:

```
# dtrace -l -f read
   ID   PROVIDER            MODULE                          FUNCTION NAME
   11    syscall                                                read entry
   12    syscall                                                read return
 3867    sysinfo           genunix                              read readch
 3871    sysinfo           genunix                              read sysread
 7454        fbt           genunix                              read entry
 7455        fbt           genunix                              read return
```

The previous output shows that for each probe, the following is displayed:

- The probe's uniquely assigned probe ID
- The provider name
- The module name (if applicable)
- The function name (if applicable)
- The probe name

Specifying Probes in DTrace

Probes are fully specified by separating each component of the 4-tuple with a colon:

provider:module:function:name

Empty components match anything. For example, `fbt::alloc:entry` specifies a probe with the following attributes:

- From the `fbt` provider
- In any module
- In the `alloc` function
- Named `entry`

Elements of the 4-tuple can be left off from the *left-hand* side. For example, `open:entry` matches probes from all providers and kernel modules that have a function name of `open` and a probe name of `entry`:

```
# dtrace -l -n open:entry
   ID   PROVIDER            MODULE                          FUNCTION NAME
   14    syscall                                                open entry
 7386        fbt           genunix                              open entry
```

Probe descriptions also support a pattern matching syntax similar to the shell *File Name Generation* syntax described in `sh`(1). The special characters `*`, `?`, and `[ ]` are all supported. For example, the `syscall::open*:entry` probe description matches both the `open` and `open64` system calls. The `?` character represents any single character in the name and `[ ]` characters lets you specify a choice of specific characters in the name.

Enabling Probes

Probes are enabled with the dtrace(1M) command by specifying them *without* the -l option. When enabled in this way, DTrace performs the *default* action when the probe fires. The default action indicates only that the probe fired. No other data is recorded. For example, the following code example enables every probe in the sd module:

```
# dtrace -m sd
dtrace: description 'sd' matched 530 probes
CPU     ID                    FUNCTION:NAME
  0  18168                 sdstrategy:entry
  0  18169                sdstrategy:return
  0  18680         ddi_xbuf_qstrategy:entry
  0  18681        ddi_xbuf_qstrategy:return
  0  18668              xbuf_iostart:entry
  0  18362          sd_xbuf_strategy:entry
  0  18406              sd_xbuf_init:entry
  0  18407             sd_xbuf_init:return
  0  18363         sd_xbuf_strategy:return
  0  18194     sd_mapblockaddr_iostart:entry
  0  18195    sd_mapblockaddr_iostart:return
  0  18196             sd_pm_iostart:entry
  0  18386               sd_pm_entry:entry
  0  18387              sd_pm_entry:return
  0  18197            sd_pm_iostart:return
  0  18198           sd_core_iostart:entry
  0  18428        sd_add_buf_to_waitq:entry
  0  18429       sd_add_buf_to_waitq:return
  0  18338            sd_start_cmds:entry
  0  18220         sd_initpkt_for_buf:entry
  0  18236          sd_fill_scsi1_lun:entry
  0  18237         sd_fill_scsi1_lun:return
  0  18221        sd_initpkt_for_buf:return
  0  18339            sd_start_cmds:return
  0  18199          sd_core_iostart:return
...
```

As you can see from the output, the default action displays the CPU where the probe fired, the DTrace assigned probe ID, the function where the probe fired, and the probe name.

To enable probes provided by the syscall provider:

```
# dtrace -P syscall
dtrace: description 'syscall' matched 452 probes
CPU     ID                    FUNCTION:NAME
  0     99                     ioctl:return
  0     98                      ioctl:entry
  0     99                     ioctl:return
  0     98                      ioctl:entry
  0     99                     ioctl:return
  0    234                  sysconfig:entry
  0    235                 sysconfig:return
  0    234                  sysconfig:entry
  0    235                 sysconfig:return
  0    168                  sigaction:entry
  0    169                 sigaction:return
  0    168                  sigaction:entry
  0    169                 sigaction:return
  0     98                      ioctl:entry
  0     99                     ioctl:return
  0    234                  sysconfig:entry
  0    235                 sysconfig:return
  0     38                        brk:entry
  0     39                       brk:return
...
```

To enable probes named zfod:

```
# dtrace -n zfod
dtrace: description 'zfod' matched 3 probes
CPU     ID                    FUNCTION:NAME
  0   3855                   anon_zero:zfod
  0   3855                   anon_zero:zfod
^C
```

To enable probes provided by the syscall provider in the open function, use the -n option with the fully specified 4-tuple syntax:

```
# dtrace -n syscall::open:
dtrace: description 'syscall::open:' matched 2 probes
CPU     ID                    FUNCTION:NAME
  0     14                       open:entry
  0     15                      open:return
  0     14                       open:entry
  0     15                      open:return
  0     14                       open:entry
  0     15                      open:return
```

```
^C
```

To enable the `entry` probe in the `clock` function (which should fire every 1/100th second):

```
# dtrace -n clock:entry
dtrace: description '::clock:entry' matched 1 probe
CPU     ID                    FUNCTION:NAME
  0   3967                      clock:entry
  0   3967                      clock:entry
  0   3967                      clock:entry
  0   3967                      clock:entry
  0   3967                      clock:entry
^C
```

DTrace Actions

Actions are user-programmable statements that are executed within the kernel by the DTrace virtual machine. The following are properties of actions:

- Actions are taken when a probe fires.
- Actions are completely programmable (in the D language).
- Most actions *record* some specified state in the system.
- Some actions can change the state of the system in a well-defined way.
  - These are called *destructive actions.*
  - Destructive actions are not allowed by default.
- Many actions use expressions in the D language.

For now, you will use D expressions that consist only of built-in D variables. The following are some of the most useful built-in D variables. See Appendix B for a complete list of the D built-in variables.

- `pid` – The current process ID
- `execname` – The current executable name
- `timestamp` – The time since boot in nanoseconds
- `curthread` – A pointer to the `kthread_t` structure that represents the current thread
- `probemod` – The current probe's module name
- `probefunc` – The current probe's function name

- probename – The current probe's name

There are also many built-in functions that perform actions. Start with the trace() function, which records the result of a D expression to the trace buffer. For example:

- trace(pid) traces the current process ID.
- trace(execname) traces the name of the current executable.
- trace(curthread->t_pri) traces the t_pri field of the current thread.
- trace(probefunc) traces the function name of the probe.

Actions are indicated by following a probe specification with "{ *action* }". For example:

```
# dtrace -n 'readch {trace(pid)}'
dtrace: description 'readch ' matched 4 probes
CPU     ID                    FUNCTION:NAME
  0   3829                      read:readch      4665
  0   3829                      read:readch      4665
  0   3829                      read:readch      4665
  0   3829                      read:readch      4665
  0   3829                      read:readch      4665
  0   3829                      read:readch      4665
  0   3829                      read:readch       279
  0   3829                      read:readch       279
  0   3829                      read:readch       279
...
```

In the last example the process identification number (PID) appears in the last column of output.

The following example traces the executable name:

```
# dtrace -m 'ufs {trace(execname)}'
dtrace: description 'ufs ' matched 889 probes
CPU     ID                    FUNCTION:NAME
  0  14977                ufs_lookup:entry   ls
  0  15748               ufs_iaccess:entry   ls
  0  15749              ufs_iaccess:return   ls
  0  14978               ufs_lookup:return   ls
  0  14977                ufs_lookup:entry   ls
  0  15748               ufs_iaccess:entry   ls
  0  15749              ufs_iaccess:return   ls
  0  14978               ufs_lookup:return   ls
  0  14977                ufs_lookup:entry   ls
  0  15748               ufs_iaccess:entry   ls
  0  15749              ufs_iaccess:return   ls
  0  14978               ufs_lookup:return   ls
  0  14977                ufs_lookup:entry   ls
...
  0  15005              ufs_rwunlock:entry   utmpd
  0  15006             ufs_rwunlock:return   utmpd
  0  14963                 ufs_close:entry   utmpd
  0  14964                ufs_close:return   utmpd
  0  15007                  ufs_seek:entry   utmpd
  0  15008                 ufs_seek:return   utmpd
  0  14963                 ufs_close:entry   utmpd
^C
```

The next action example traces the time of entry to each system call:

```
# dtrace -n 'syscall:::entry {trace(timestamp)}'
dtrace: description 'syscall:::entry ' matched 226 probes
CPU     ID                    FUNCTION:NAME
  0  18922                     ioctl:entry   243178113015475
  0  18922                     ioctl:entry   243178113050016
  0  19056                 sysconfig:entry   243178113102658
  0  19056                 sysconfig:entry   243178113109117
  0  18992                 sigaction:entry   243178113302561
  0  18992                 sigaction:entry   243178113318720
  0  18922                     ioctl:entry   243178113386576
  0  19056                 sysconfig:entry   243178113449599
  0  18864                       brk:entry   243178113467377
  0  18864                       brk:entry   243178113482713
  0  18922                     ioctl:entry   243178113741172
  0  18922                     ioctl:entry   243178113854123
  0  18922                     ioctl:entry   243178113876105
  0  18922                     ioctl:entry   243178113909138
  0  18886                     fstat:entry   243178114009371
  0  18838                     write:entry   243178114107163
  0  18838                     write:entry   243178114257629
  0  18838                     write:entry   243178114334694
...
```

Multiple actions can be specified; they must be separated by semicolons:

```
# dtrace -n 'zfod {trace(pid);trace(execname)}'
dtrace: description 'zfod ' matched 3 probes
CPU     ID                    FUNCTION:NAME
  0   3863              anon_zero:zfod    4972  dtrace
  0   3863              anon_zero:zfod    4813  sh
  0   3863              anon_zero:zfod    4973  vi
  0   3863              anon_zero:zfod    4973  vi
  0   3863              anon_zero:zfod    4973  vi
...
```

The following example traces the executable name in every entry to the `pagefault` function:

```
# dtrace -n 'fbt::pagefault:entry {trace(execname)}'
dtrace: description 'fbt::pagefault:entry ' matched 1 probe
CPU     ID                    FUNCTION:NAME
  0   2407              pagefault:entry   dtrace
  0   2407              pagefault:entry   dtrace
  0   2407              pagefault:entry   dtrace
  0   2407              pagefault:entry   sh
  0   2407              pagefault:entry   sh
  0   2407              pagefault:entry   sh
  0   2407              pagefault:entry   sh
  0   2407              pagefault:entry   sh
...
```

Writing D Scripts

Complicated DTrace enablings become difficult to manage on the command line. The `dtrace`(1M) command supports scripts, specified with the `-s` option. Alternatively, you can create executable DTrace interpreter files. Interpreter files always begin with:

```
#!/usr/sbin/dtrace -s
```

Executable D Scripts

For example, you can write a script to trace the executable name upon entry to each system call as follows:

```
# cat syscall.d
syscall:::entry
{
  trace(execname);
}
```

By convention, D scripts end with a `.d` suffix. You can run this D script as follows:

```
# dtrace -s syscall.d
dtrace: script 'syscall.d' matched 225 probes
CPU     ID                    FUNCTION:NAME
  0   312          pollsys:entry  java
  0    98            ioctl:entry  dtrace
  0    98            ioctl:entry  dtrace
  0   234        sysconfig:entry dtrace
  0   234        sysconfig:entry dtrace
  0   168        sigaction:entry dtrace
  0   168        sigaction:entry dtrace
  0    98            ioctl:entry  dtrace
  0   234        sysconfig:entry dtrace
  0    38              brk:entry  dtrace
```

If you give the `syscall.d` file execute permission and add a first line to invoke the interpreter, you can run the script by entering its name on the command line as follows:

```
# cat syscall.d
#!/usr/sbin/dtrace -s

syscall:::entry
{
  trace(execname);
}
# chmod +x syscall.d
# ls -l syscall.d
-rwxr-xr-x   1 root     other         62 May 12 11:30 syscall.d
# ./syscall.d
dtrace: script './syscall.d' matched 225 probes
CPU     ID                    FUNCTION:NAME
  0     98                  ioctl:entry   java
  0     98                  ioctl:entry   java
  0    312                pollsys:entry   java
  0    312                pollsys:entry   java
  0    312                pollsys:entry   java
  0     98                  ioctl:entry   dtrace
  0     98                  ioctl:entry   dtrace
  0    234              sysconfig:entry   dtrace
  0    234              sysconfig:entry   dtrace
```

D Literal Strings

The D language supports literal strings that you can use with the `trace` function as follows:

```
# cat string.d
fbt:sd:sdstrategy:entry
{
trace(execname);
trace(" is initiating a disk I/O\n");
}
```

The `\n` at the end of the literal string produces a new line. To run this script, enter the following:

```
# dtrace -s string.d
dtrace: script 'string.d' matched 1 probe
CPU     ID                    FUNCTION:NAME
  0  18222                 sdstrategy:entry   sendmail is initiating a disk I/O

  0  18222                 sdstrategy:entry   fsflush is initiating a disk I/O

  0  18222                 sdstrategy:entry   sched is initiating a disk I/O
```

The quiet mode option, `-q`, in `dtrace`(1M) tells DTrace to record only the actions explicitly stated. This option suppresses the default output normally produced by the `dtrace` command. The following example shows the use of the `-q` option on the `string.d` script:

```
# dtrace -q -s string.d
ls is initiating a disk I/O
cat is initiating a disk I/O
fsflush is initiating a disk I/O
vi is initiating a disk I/O
vi is initiating a disk I/O
```

Begin and End Probes

The simple `dtrace` provider has only three probes. They are `BEGIN`, `END`, and `ERROR`. The `BEGIN` probe fires before all others and performs pre-processing steps. For example, it enables you to initialize variables, as well as to display headings for output that is displayed by other actions that occur later. The `END` probe fires after all other probes have fired and enables you to perform post-processing. The `ERROR` probe fires when there are any runtime errors in your D programs. The following example shows a simple use of the `BEGIN` and `END` probes of the `dtrace` provider:

```
# cat beginEnd.d
#!/usr/sbin/dtrace -s
BEGIN
{
trace("This is a heading\n");
}

END
{
trace("This should appear at the END\n");
}
# ./beginEnd.d
dtrace: script './beginEnd.d' matched 2 probes
CPU     ID                    FUNCTION:NAME
  0      1                          :BEGIN   This is a heading

^C
  0      2                            :END   This should appear at the END

# dtrace -q -s beginEnd.d
This is a heading
^C
This should appear at the END
```

Note – The `END` probe does not fire until you interrupt (^C) the `dtrace` command.

Module 7

Using DTrace

Objectives

Upon completion of this module, you should be able to:

- Describe the DTrace performance monitoring capabilities
- Examine performance problems using the `vminfo` provider
- Examine performance problems using the `sysinfo` provider
- Examine performance problems using the `io` provider
- Use DTrace to obtain information about system calls
- Create D scripts that use arguments

Relevance

Discussion – The following questions are relevant to understanding how to use DTrace:

- What performance monitoring tools exist in the Solaris 10 OS?
- Would it be useful to know which process is making which system calls?
- What is the advantage of being able to pass arguments to a D script?

Additional Resources

Additional resources – The following references provide additional information on the topics described in this module:

- Cantrill, Bryan M., Michael W. Shapiro, and Adam H. Leventhal. "Dynamic Instrumentation of Production Systems." Draft paper accepted for presentation at 2004 USENIX Conference.
- *Solaris Dynamic Tracing Guide* (Beta). Sun Microsystems, Inc., part number 817-6223-05.
- `dtrace`(1M) manual page in the Solaris 10 OS manual pages, Solaris 10 Reference Manual Collection.

Understanding the DTrace Performance Monitoring Capabilities

A number of the new DTrace providers implement probes that correspond to existing Solaris OS performance monitoring tools:

- The `vminfo` provider – Implements probes that correspond to the `vmstat`(1M) tool
- The `sysinfo` provider – Implements probes that correspond to the `mpstat`(1M) tool
- The `io` provider – Implements probes that correspond to the `iostat`(1M) tool

In addition, the `syscall` provider implements probes that correspond to the `truss`(1) command.

Features of the DTrace Performance Monitoring Capabilities

Using the DTrace facility, you can extract the same information that the bundled tools provide, with significant added flexibility. DTrace enables you to gather only the specific information you need to diagnose the aberrant behavior. It also provides additional related information such as process and thread identification, stack traces, and other arbitrary kernel information available at the time the probes fire.

Aggregations

Aggregated data is more useful than individual data points in answering performance-related questions. For example, if you want to know the number of page faults by process, you do not necessarily care about each individual page fault. Rather, you want a table that lists the process names and the total number of page faults.

DTrace provides several built-in aggregating functions. An aggregating function has this property: If it is applied to subsets of a collection of gathered data and then applied again to the results, it returns the same result as it does when applied to the whole collection. Examples of aggregating functions are `count`(), `sum`(), `min`(), and `max`(); The *median* function is not an aggregating function because it lacks the above mentioned property.

DTrace is not required to store the entire set of data items for aggregations; it keeps a running count, needing only the current intermediate result and the new element. Intermediate results are kept per central processing unit (CPU), enabling a scalable implementation.

DTrace Aggregation Syntax

The general form of a DTrace aggregation is:

```
@name[ keys ] = aggfunc( args );
```

These variables are defined as follows:

- `name` – The name of the aggregation that is preceded by the `@` character
- `keys` – A comma-separated list of D expressions
- `aggfunc` – One of the DTrace aggregating functions
- `args` – A comma-separated list of arguments appropriate to the aggregating function

DTrace Aggregating Functions

Table 7-1 lists the DTrace aggregating functions.

Table 7-1 DTrace Aggregating Functions

Function Name	**Arguments**	**Result**
`count`	none	The number of times called.
`sum`	scalar expression	The total value of the specified expressions.
`avg`	scalar expression	The arithmetic average (mean) of the specified expressions.
`min`	scalar expression	The smallest value of the specified expressions.
`max`	scalar expression	The largest value of the specified expressions.

Table 7-1 DTrace Aggregating Functions

Function Name	Arguments	Result
`lquantize`	scalar expression, lower bound, upper bound, step value	A linear frequency distribution, sized by the specified range, of the values of the specified expression. Increments the value in the highest bucket that is less than the specified expression.
`quantize`	scalar expression	A power-of-two frequency distribution of the values of the specified expression. Increments the value in the highest power-of-two bucket that is less than the specified expression.

Example Use of Aggregating Function

In the following example, the `count` aggregating function is used to count the number of `write`(2) system calls per process:

```
# cat writes.d
#!/usr/sbin/dtrace -s
syscall::write:entry
{
@numWrites[execname] = count();
}
# dtrace -s writes.d
dtrace: script 'writes.d' matched 1 probe
^C
  dtrace                                                              1
  date                                                                1
  bash                                                                3
  grep                                                               20
  file                                                              197
  ls                                                                201
```

Arguments Supplied by Providers

The `syscall` provider gives you access to a system call's arguments, using the syntax `arg0`, `arg1`, `arg2`, for the function's first, second, third, and so on, arguments. The following example displays the average write size per process:

```
# cat writes2.d
#!/usr/sbin/dtrace -s
syscall::write:entry
{
@avgSize[execname] = avg(arg2);
}
# dtrace -s writes2.d
dtrace: script 'writes2.d' matched 1 probe
^C
  dtrace                                                         1
  bash                                                          27
  date                                                          29
  file                                                          37
  grep                                                          60
  ls                                                            68
```

Examining Performance Problems Using the vminfo Provider

The vminfo provider makes available probes from the virtual memory (vm) kernel statistics (kstat) kept by the kernel kstat facility. You can examine any unexplainable behavior observed from the vm specific output of the vmstat(1M) command using this DTrace provider. A probe provided by the vminfo provider fires immediately before the corresponding vm kstat value is incremented. To display both the names and the current values (counts) of the vm named kstat, one can use the kstat(1M) command as shown in the following command example.

```
# kstat -n vm
module: cpu                             instance: 0
name:   vm                          class:    misc
        anonfree                        0
        anonpgin                        4
        anonpgout                       0
        as_fault                        157771
        cow_fault                       34207
        crtime                          0.178610697
        dfree                           56
        execfree                        0
        execpgin                        3646
        execpgout                       0
        fsfree                          56
        fspgin                          16257
        fspgout                         57
        hat_fault                       0
        kernel_asflt                    0
        maj_fault                       6743
        pgfrec                          34215
        pgin                            9188
        pgout                           36
        pgpgin                          19907
        pgpgout                         57
        pgrec                           34216
        pgrrun                          4
        pgswapin                        0
        pgswapout                       0
        prot_fault                      39794
        rev                             0
```

```
scan                                28668
snaptime                            349429.087071013
softlock                            165
swapin                              0
swapout                             0
zfod                                12835
```

The vminfo Probes

Table 7-2 describes the `vminfo` probes.

Table 7-2 The `vminfo` Probes

Probe Name	Description
`anonfree`	Probe that fires when an unmodified anonymous page is freed as part of paging activity. Anonymous pages are those that are not associated with a file; memory containing such pages include heap memory, stack memory, or memory obtained by explicitly mapping `zero`(7D).
`anonpgin`	Probe that fires when an anonymous page is paged in from a swap device.
`anonpgout`	Probe that fires when a modified anonymous page is paged out to a swap device.
`as_fault`	Probe that fires when a fault is taken on a page and the fault is neither a protection fault nor a copy-on-write fault.
`cow_fault`	Probe that fires when a copy-on-write fault is taken on a page. The `arg0` argument contains the number of pages that are created as a result of the copy-on-write.
`dfree`	Probe that fires when a page is freed as a result of paging activity. When `dfree` fires, exactly one of the `anonfree`, `execfree,` or `fsfree` probes also subsequently fires.
`execfree`	Probe that fires when an unmodified executable page is freed as a result of paging activity.
`execpgin`	Probe that fires when an executable page is paged in from the backing store.
`execpgout`	Probe that fires when a modified executable page is paged out to the backing store. Most paging of executable pages occurs in terms of the `execfree` probe; the `execpgout` probe can only fire if an executable page is modified in memory, an uncommon occurrence in most systems.

Table 7-2 The `vminfo` Probes (Continued)

Probe Name	**Description**
`fsfree`	Probe that fires when an unmodified file system data page is freed as part of paging activity.
`fspgin`	Probe that fires when a file system page is paged in from the backing store.
`fspgout`	Probe that fires when a file system page is paged out to the backing store.
`kernel_asflt`	Probe that fires when a page fault is taken by the kernel on a page in its own address space. When the `kernel_asflt` probe fires, it is immediately preceded by a firing of the `as_fault` probe.
`maj_fault`	Probe that fires when a page fault is taken that results in input/output (I/O) from a backing store or swap device. Whenever `maj_fault` fires, it is immediately preceded by a firing of the `pgin` probe.
`pgfrec`	Probe that fires when a page is reclaimed from the free page list.
`pgin`	Probe that fires when a page is paged in from the backing store or from a swap device. This differs from the `maj_fault` probe in that the `maj_fault` probe only fires when a page is paged in as a result of a page fault; the `pgin` probe fires when a page is paged in, regardless of the reason.
`pgout`	Probe that fires when a page is paged out to the backing store or to a swap device.
`pgpgin`	Probe that fires when a page is paged in from the backing store or from a swap device. The only difference between the `pgpgin` probe and the `pgin` probe is that the `pgpgin` probe contains the number of pages paged in as the `arg0` argument. (The `pgin` probe always contains 1 in the `arg0` argument.)
`pgpgout`	Probe that fires when a page is paged out to the backing store or to a swap device. The only difference between the `pgpgout` probe and the `pgout` probe is that the `pgpgout` probe contains the number of pages paged out as the `arg0` argument. (The `pgout` probe always contains 1 in the `arg0` argument.)
`pgrec`	Probe that fires when a page is reclaimed.
`pgrrun`	Probe that fires when the pager is scheduled.
`pgswapin`	Probe that fires when a process is swapped in.
`pgswapout`	Probe that fires when a process is swapped out.
`prot_fault`	Probe that fires when a page fault is taken due to a protection violation.

Table 7-2 The vminfo Probes (Continued)

Probe Name	Description
rev	Probe that fires when the page daemon begins a new revolution through all pages.
scan	Probe that fires when the page daemon examines a page.
softlock	Probe that fires when a page is faulted as a part of placing a software lock on the page.
swapin	Probe that fires when a swapped-out process is swapped back in.
swapout	Probe that fires when a process is swapped out.
zfod	Probe that fires when a zero-filled page is created on demand.

How to Find the Source of Page Faults Using vminfo Probes

Consider the following example output, obtained by running the vmstat command.

```
# vmstat 5
 kthr      memory            page            disk          faults      cpu
 r b w   swap  free  re  mf pi po fr de sr s0 s2 s1 --   in   sy   cs us sy id
 0 0 0 648560 437016  3  11 13  0  0  0  8  0  1  0  0  406   42   50  0  0 100
 0 0 0 598912 396136  0  11 27  0  0  0  0  0  0  0  0  615  113   67  0  0 100
 0 0 0 598888 396112  0   1  0  0  0  0  0  0  0  0  0  604   69   47  0  0 100
 0 0 0 598864 396088  0   1  0  0  0  0  0  0  0  0  0  616   69   72  0  0 100
 0 0 0 598864 396088  0   0  0  0  0  0  0  0  0  0  0  619   73   89  0  0 100
 0 1 0 598104 393456  4  45 3588 0 0  0  0  0 474 0  0 2014 5138 1013  3 17 79
 0 0 0 595224 381544  0   2 5273 0 0  0  0  0 698 0  0 2593 7545 1448  3 31 66
 0 0 0 592024 368832  0   1 5509 0 0  0  0  0 725 0  0 2674 7840 1503  3 26 71
 0 0 0 588792 362640  1   3 3679 0 0  0  0  0 485 0  0 2009 5259 1027  3 20 77
 0 0 0 587984 361848  0   3  4  0  0  0  0  0  0  0  0  605   80   70  0  0 100
 0 0 0 587960 361800  0   4 20  0  0  0  0  0  2  0  0  624   74   91  0  0 100
 0 0 0 587944 361768  0   1  0  0  0  0  0  0  0  0  0  614   76   78  0  0 100
 0 0 0 587920 361744  0   1  0  0  0  0  0  0  0  0  0  616   69   80  0  0 100
 0 0 0 587848 361672  0   1  0  0  0  0  0  0 18  0  0  689   69   69  0  0 100
 0 0 0 587832 361656  0   1  0  0  0  0  0  0  0  0  0  611   74   67  0  0 100
 0 0 0 587808 361632  0   5  0  0  0  0  0  0  0  0  0  611   71   66  0  0 100
 0 0 0 587784 361608 40 193 844 0  0  0  0  0 107 0  0  953  905  260  3  5 92
 0 0 0 588184 362576  0   1  0  0  0  0  0  0  0  0  0  611   69   71  0  0 100
```

Here the pi column denotes the number of kilobytes paged in per second.

Executable Causing Page Faults

The `vminfo` provider makes it easy to discover more about the source of these page-ins. The following example uses an *unnamed* aggregation:

```
# dtrace -n 'pgin {@[execname] = count()}'
dtrace: description 'pgin ' matched 1 probe
^C
  utmpd                                                          2
  in.routed                                                      2
  init                                                           2
  snmpd                                                          5
  automountd                                                     5
  vi                                                             5
  vmstat                                                        17
  sh                                                            23
  grep                                                          33
  dtrace                                                        35
  bash                                                          62
  file                                                         198
  find                                                        4551
```

This output shows that the `find` command is responsible for most of the page-ins. For a more complete picture of the `find` command in terms of vm behavior, you can enable all `vminfo` probes. Before doing this, however, you must introduce a filtering capability of DTrace called a *predicate*.

Predicates

A D program consists of a set of probe clauses. A probe clause has the following general form:

```
probe descriptions
/ predicate /
{
action statements
}
```

Predicates are D expressions enclosed in slashes `/ /` that are evaluated at probe firing time to determine whether the associated actions should be executed. If the D expression evaluates to zero it is false; if it evaluates to non-zero it is true. Predicates are optional, but you must place them between the probe description and the action statements.

Details About the Executable Causing Page Faults

The following example examines the system's detailed vm behavior while the find command runs:

```
# cat find.d
#!/usr/sbin/dtrace -s
vminfo:::
/execname == "find"/
{ @[probename] = count(); }
```

Before running this D program, run a find command in the background while another utility uses up a substantial portion of the system's memory, as shown in the following example.

```
# (sleep 10 ; find / -name fubar & mkfile 300m /tmp/junk)&
[1] 840
# ps
   PID TTY         TIME CMD
   615 pts/2       0:00 sh
   841 pts/2       0:00 sleep
   625 pts/2       0:00 bash
   840 pts/2       0:00 bash
   842 pts/2       0:00 ps
# ps
   PID TTY         TIME CMD
   615 pts/2       0:00 sh
   843 pts/2       0:02 find
   625 pts/2       0:00 bash
   840 pts/2       0:00 bash
   845 pts/2       0:00 ps
   844 pts/2       0:02 mkfile
# ps
   PID TTY         TIME CMD
   615 pts/2       0:00 sh
   843 pts/2       0:08 find
   625 pts/2       0:00 bash
   846 pts/2       0:00 ps
[1]+  Done                    ( sleep 10 ; find / -name fubar & mkfile 300m /tmp/junk )
# ps
   PID TTY         TIME CMD
   615 pts/2       0:00 sh
   847 pts/2       0:00 ps
   625 pts/2       0:00 bash
```

The following dtrace command was started in another terminal window immediately after the above command group was started in the background.

```
# dtrace -s find.d
dtrace: script 'find.d' matched 44 probes
^C
  prot_fault                                                        2
  cow_fault                                                         8
  softlock                                                         11
  execpgin                                                         15
  kernel_asflt                                                     40
  zfod                                                             52
  as_fault                                                        170
  pgrec                                                          5417
  pgfrec                                                         5417
  maj_fault                                                     18068
  fspgin                                                        18103
  pgpgin                                                        18118
  pgin                                                          18118
```

You might wonder why, with such a large memory load, scans do not show up in the output of the dtrace command. This is because the pageout daemon is running during scans, not the find user process. The following example shows this behavior.

```
# cat mem.d
#!/usr/sbin/dtrace -s
vminfo:::
{ @vm[execname,probename] = count(); }
END
{
printa("%16s\t%16s\t%@d\n", @vm);
}

# dtrace -q -s mem.d
^C
           sleep              prot_fault        1
              rm              prot_fault        1
         pageout                     rev        1
          dtrace                  pgfrec        1
            bash            kernel_asflt        1
       in.routed                anonpgin        1
          mkfile              prot_fault        1
            find              prot_fault        1
          dtrace                   pgrec        1
          mkfile                execpgin        2
          mkfile            kernel_asflt       2
          vmstat              prot_fault       2
              rm                    zfod       3
            find                execpgin       3
           sleep                    zfod       3
          mkfile                    zfod       3
        sendmail                anonpgin       3
          mkfile               cow_fault       4
              rm               cow_fault       4
            bash                anonpgin       4
              rm               maj_fault       4
        sendmail                  pgfrec       4
           sleep               cow_fault       4
            find               cow_fault       4
        sendmail                   pgrec       4
...
            bash                   pgrec       205
         pageout                 fspgout       293
         pageout               anonpgout       293
         pageout                 pgpgout       293
         pageout                   pgout       293
         pageout               execpgout       293
         pageout                   pgrec       293
         pageout                anonfree       360
         pageout                execfree       510
            bash                as_fault       519
         pageout                  fsfree       519
           sched                   dfree       523
           sched                   pgrec       523
           sched                   pgout       523
           sched                 pgpgout       523
           sched               anonpgout       523
           sched                anonfree       523
           sched                 fspgout       523
           sched                  fsfree       523
           sched               execpgout       523
           sched                execfree       523
         pageout                   dfree       803
              rm                   pgrec       1388
              rm                  pgfrec       1388
            find               maj_fault       5067
            find                  fspgin       5085
            find                    pgin       5088
            find                  pgpgin       5088
         pageout                    scan       78852
```

The printa() built-in formatting function gives you increased control over the output of an aggregation. For example, consider the following code line:

```
{
printa("%16s\t%16s\t%@d\n", @vm);
}
```

It provides these formatting instructions:

- %16s\t%16s prints the first and second elements of the aggregation *keys* in a 16-character-wide column (right justified).
- \t outputs a <Tab>.
- %@d prints the aggregation value as a decimal number.

Note – Appendix A provides more details on the format letters available to the printa() function and the more general printf() function (which resembles the printf(3C) function from the Standard C Library).

Examining Performance Problems Using the sysinfo Provider

The sysinfo provider makes available probes that correspond to the "sys" kernel statistics. Because these statistics provide the input for system monitoring utilities such as mpstat(1M), the sysinfo provider enables quick exploration of observed aberrant behavior.

The sysinfo provider probes fire immediately before the sys named kstat is incremented. The following example displays the sys named kstat.

```
# kstat -n sys
module: cpu                             instance: 0
name:   sys                         class:    misc
        bawrite                         112
        bread                           6359
        bwrite                          1401
        canch                           374
        cpu_ticks_idle                  2782331
        cpu_ticks_kernel                46571
        cpu_ticks_user                  12187
        cpu_ticks_wait                  30197
        cpumigrate                      0
...
        syscall                         3991217
        sysexec                         1088
        sysfork                         1043
        sysread                         131334
        sysvfork                        47
        syswrite                        676775
        trap                            266286
        ufsdirblk                       1027383
        ufsiget                         1086164
        ufsinopage                      873613
        ufsipage                        2
        wait_ticks_io                   30197
        writech                         5144172931
        xcalls                          0
        xmtint                          0
```

The sysinfo Probes

Table 7-3 describes the sysinfo probes.

Table 7-3 The sysinfo Probes

Probe Name	Description
bawrite	Probe that fires when a buffer is about to be asynchronously written out to a device.
bread	Probe that fires when a buffer is physically read from a device. The bread probe fires after the buffer has been requested from the device, but before blocking pending its completion.
bwrite	Probe that fires when a buffer is about to be written out to a device synchronously or asynchronously.
cpu_ticks_idle	Probe that fires when the periodic system clock has determined that a CPU is idle. Note that this probe fires in the context of the system clock and therefore fires on the CPU running the system clock; one must examine the cpu_t argument (arg2) to determine the CPU that has been deemed idle.
cpu_ticks_kernel	Probe that fires when the periodic system clock has determined that a CPU is executing in the kernel. Note that this probe fires in the context of the system clock and therefore fires on the CPU running the system clock; one must examine the cpu_t argument (arg2) to determine the CPU that has been deemed to be executing in the kernel.
cpu_ticks_user	Probe that fires when the periodic system clock has determined that a CPU is executing in user mode. Note that this probe fires in the context of the system clock and therefore fires on the CPU running the system clock; one must examine the cpu_t argument (arg2) to determine the CPU that has been deemed to be running in user-mode.
cpu_ticks_wait	Probe that fires when the periodic system clock has determined that a CPU is otherwise idle, but on which some threads are waiting for I/O. Note that this probe fires in the context of the system clock and therefore fires on the CPU running the system clock; one must examine the cpu_t argument (arg2) to determine the CPU that has been deemed waiting on I/O.
idlethread	Probe that fires when a CPU enters the idle loop.
intrblk	Probe that fires when an interrupt thread blocks.

Table 7-3 The sysinfo Probes (Continued)

Probe Name	**Description**
inv_swtch	Probe that fires when a running thread is forced to involuntarily give up the CPU.
lread	Probe that fires when a buffer is logically read from a device.
lwrite	Probe that fires when a buffer is logically written to a device.
modload	Probe that fires when a kernel module is loaded.
modunload	Probe that fires when a kernel module is unloaded.
msg	Probe that fires when a msgsnd(2) or msgrcv(2) system call is made, but before the message queue operations have been performed.
mutex_adenters	Probe that fires when an attempt is made to acquire an owned adaptive lock. If this probe fires, one of the lockstat provider probes (adaptive-block or adaptive-spin) also fires.
namei	Probe that fires when a name lookup is attempted in the file system.
nthreads	Probe that fires when a thread is created.
phread	Probe that fires when a raw I/O read is about to be performed.
phwrite	Probe that fires when a raw I/O write is about to be performed.
procovf	Probe that fires when a new process cannot be created because the system is out of process table entries.
pswitch	Probe that fires when a CPU switches from executing one thread to executing another.
readch	Probe that fires after each successful read, but before control is returned to the thread performing the read. A read can occur through the read(2), readv(2), or pread(2) system calls. The arg0 argument contains the number of bytes that were successfully read.
rw_rdfails	Probe that fires when an attempt is made to read-lock a readers/writer lock when the lock is either held by a writer, or desired by a writer. If this probe fires, the lockstat provider's rw-block probe also fires.
rw_wrfails	Probe that fires when an attempt is made to write-lock a readers/writer lock when the lock is held either by some number of readers or by another writer. If this probe fires, the lockstat provider's rw-block probe also fires.
sema	Probe that fires when a semop(2) system call is made, but before any semaphore operations have been performed.

Table 7-3 The sysinfo Probes (Continued)

Probe Name	**Description**
sysexec	Probe that fires when an exec(2) system call is made.
sysfork	Probe that fires when a fork(2) system call is made.
sysread	Probe that fires when a read(2), readv(2) or pread(2) system call is made.
sysvfork	Probe that fires when a vfork(2) system call is made.
syswrite	Probe that fires when a write(2), writev(2), or pwrite(2) system call is made.
trap	Probe that fires when a processor trap occurs. Note that some processors (in particular, UltraSPARC® variants) handle some light-weight traps through a mechanism that does not cause this probe to fire.
ufsdirblk	Probe that fires when a directory block is read from the UFS file system. See the ufs(7FS) man page for details on UFS.
ufsiget	Probe that fires when an inode is retrieved. See the ufs(7FS) man page for details on UFS.
ufsinopage	Probe that fires after an in-core inode without any associated data pages has been made available for reuse. See the ufs(7FS) man page for details on UFS.
ufsipage	Probe that fires after an in-core inode with associated data pages has been made available for reuse and therefore after the associated data pages have been flushed to disk. See the ufs(7FS) man page for details on UFS.
wait_ticks_io	Probe that fires when the periodic system clock has determined that a CPU is otherwise idle, but on which some threads are waiting for I/O. Note that this probe fires in the context of the system clock and therefore fires on the CPU running the system clock; one must examine the cpu_t argument (arg2) to determine the CPU that has been deemed waiting on I/O. Note that there is no semantic difference between wait_ticks_io and cpu_ticks_io; wait_ticks_io exists purely for historical reasons.
writech	Probe that fires after each successful write, but before control is returned to the thread performing the write. A write can occur through the write(2), writev(2). or pwrite(2) system calls. The arg0 argument contains the number of bytes that were successfully written.

Table 7-3 The sysinfo Probes (Continued)

Probe Name	Description
xcalls	Probe that fires when a cross-call is about to be made. A cross-call is the operating system's mechanism for one CPU to request immediate work from another.

How to Use the quantize Aggregation Function With the sysinfo Probes

The quantize aggregation function displays a power-of-two frequency distribution bar graph of its argument. The following example shows how you can determine the size of reads being performed by all processes over a 10-second period. The arg0 argument for the sysinfo probes states the amount to increment the statistic; it is 1 for most sysinfo probes. Two exceptions are the readch and writech probes, for which the arg0 argument is set to the number of bytes read or written respectively.

```
# cat read.d
sysinfo:::readch
{
@[execname] = quantize(arg0);
}

tick-10sec
{
exit(0);
}
# dtrace -s read.d
dtrace: script 'read.d' matched 5 probes
CPU     ID                    FUNCTION:NAME
  0  36754                      :tick-10sec

  bash
           value  ------------- Distribution ------------- count
               0 |                                         0
               1 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ 13
               2 |                                         0

  file
           value  ------------- Distribution ------------- count
              -1 |                                         0
               0 |                                         2
               1 |                                         0
               2 |                                         0
               4 |                                         6
               8 |                                         0
              16 |                                         0
              32 |                                         6
              64 |                                         6
             128 |@@                                       16
             256 |@@@@                                     30
             512 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@           199
            1024 |                                         0
            2048 |                                         0
            4096 |                                         1
            8192 |                                         1
           16384 |                                         0

  grep
           value  ------------- Distribution ------------- count
              -1 |                                         0
               0 |@@@@@@@@@@@@@@@@@@@                      99
               1 |                                         0
               2 |                                         0
               4 |                                         0
               8 |                                         0
              16 |                                         0
              32 |                                         0
              64 |                                         0
             128 |                                         1
             256 |@@@@                                     25
             512 |@@@@                                     23
            1024 |@@@@                                     24
            2048 |@@@@                                     22
            4096 |                                         4
            8192 |                                         3
           16384 |                                         0
```

How to Find Cross-calls

Consider the following output from the `mpstat`(1M) command:

```
CPU minf mjf xcal  intr ithr  csw icsw migr smtx  srw syscl  usr sys  wt idl
  0 2189   0 1302    14    1  215   12   54   28    0 12995   13  14   0  73
  1 3385   0 1137   218  104  195   13   58   33    0 14486   19  15   0  66
  2 1918   0 1039    12    1  226   15   49   22    0 13251   13  12   0  75
  3 2430   0 1284   220  113  201   10   50   26    0 13926   10  15   0  75
```

The xcal and syscl columns display relatively high numbers, which might be affecting the system's performance. Yet the system is relatively idle, and is not spending time waiting on input/output (I/O). The xcal numbers are per-second and are read from the xcalls field of the sys kstat. To see which executables are responsible for the xcalls, enter the following dtrace(1M) command:

```
# dtrace -n 'xcalls {@[execname] = count()}'
dtrace: description 'xcalls ' matched 4 probes
^C
  find                                                             2
  cut                                                              2
  snmpd                                                            2
  mpstat                                                          22
  sendmail                                                       101
  grep                                                           123
  bash                                                           175
  dtrace                                                         435
  sched                                                          784
  xargs                                                        22308
  file                                                         89889
#
```

This output indicates the source of the cross-calls: some number of file(1) and xargs(1) processes are inducing the majority of them. You can find these processes using the pgrep(1) and ptree(1) commands:

```
# pgrep xargs
15973
# ptree 15973
204   /usr/sbin/inetd -s
  5650  in.telnetd
    5653  -sh
      5657  bash
        15970 /bin/sh ./findtxt configuration
          15971 cut -f1 -d:
            15973 xargs file
              16686 file /usr/bin/tbl /usr/bin/troff /usr/bin/ul
/usr/bin/vgrind /usr/bin/catman
```

The xargs and file commands appear to be part of a custom user shell script. You can locate this script as follows:

```
# find / -name findtxt
/users1/james/findtxt
# cat /users1/james/findtxt
#!/bin/sh
find / -type f | xargs file | grep text | cut -f1 -d: >/tmp/findtxt$$
cat /tmp/findtxt$$ | xargs grep $1
rm /tmp/findtxt$$
#
```

The script is running many processes concurrently with much inter-process communication occurring through *pipes*. This script appears to be quite resource intensive: it is trying to find every text file in the system and is then searching each one for some specific text. You expect these processes to run concurrently on this system's four processors while they send data to each other.

Stack Trace xcall Details

You can gather more details on which kernel code is involved in all of the cross-calls while the file and xargs commands are running. The following example uses the stack() built-in DTrace function as the aggregation key to show which kernel code is requesting the cross-call. The number of unique kernel stack traces is being counted.

```
# dtrace -n 'xcalls {@[stack()] = count()}'
dtrace: description 'xcalls ' matched 4 probes
^C
              SUNW,UltraSPARC-IIIi`send_mondo_set+0x9c
              unix`xt_some+0xc4
              unix`xt_sync+0x3c
              unix`hat_unload_callback+0x6ec
              unix`memscrub_scan+0x298
              unix`memscrubber+0x308
              unix`thread_start+0x4
                2

              SUNW,UltraSPARC-IIIi`send_mondo_set+0x9c
              unix`xt_some+0xc4
              unix`sfmmu_tlb_demap+0x118
              unix`sfmmu_hblk_unload+0x368
              unix`hat_unload_callback+0x534
              unix`memscrub_scan+0x298
              unix`memscrubber+0x308
              unix`thread_start+0x4
                2
...
              SUNW,UltraSPARC-IIIi`send_mondo_set+0x9c
              unix`xt_some+0xc4
              unix`xt_sync+0x3c
              unix`hat_unload_callback+0x6ec
              genunix`anon_private+0x204
              genunix`segvn_faultpage+0x778
              genunix`segvn_fault+0x920
              genunix`as_fault+0x4a0
              unix`pagefault+0xac
              unix`trap+0xc14
              unix`utl0+0x4c
             2303

              SUNW,UltraSPARC-IIIi`send_mondo_set+0x9c
              unix`xt_some+0xc4
              unix`sfmmu_tlb_range_demap+0x190
              unix`sfmmu_chgattr+0x2e8
              genunix`segvn_dup+0x3d0
              genunix`as_dup+0xd0
              genunix`cfork+0x120
              unix`syscall_trap32+0xa8
             7175

              SUNW,UltraSPARC-IIIi`send_mondo_set+0x9c
              unix`xt_some+0xc4
              unix`xt_sync+0x3c
              unix`sfmmu_chgattr+0x2f0
              genunix`segvn_dup+0x3d0
              genunix`as_dup+0xd0
              genunix`cfork+0x120
              unix`syscall_trap32+0xa8
            11492
```

As this output shows, the majority of the cross-calls are the result of a significant number of fork(2) system calls. (Shell scripts are notorious for abusing their fork(2) privileges.) Page faults of anonymous memory are also involved, which probably accounts for the large number of minor page faults seen in the mpstat output.

Examining Performance Problems Using the io Provider

The io provider makes available probes related to device input and output (I/O). The io provider is designed to enable quick exploration of behavior observed through I/O monitoring tools such as iostat(1M). The io provider describes the nature of the system's I/O by providing data such as the following:

- Device
- I/O type
- Process ID
- Application name
- File name
- File offset

The io Probes

Table 7-4 describes the io probes.

Table 7-4 The io Probes

Probe Name	Description
start	Probe that fires when an I/O request is about to be made to a peripheral device or to an NFS server. The buf(9S) structure corresponding to the I/O request is pointed to by the args[0] argument. The devinfo_t structure of the device to which the I/O is being issued is pointed to by the args[1] argument. The fileinfo_t structure of the file that corresponds to the I/O request is pointed to by the args[2] argument. Note that file information availability depends on the file system making the I/O request.
done	Probe that fires after an I/O request has been fulfilled. The buf(9S) structure corresponding to the I/O request is pointed to by the args[0] argument. The devinto_t structure of the device to which the I/O was issued is pointed to by the args[1] argument. The fileinfo_t structure of the file that corresponds to the I/O request is pointed to by the args[2] argument.

Table 7-4 The io Probes (Continued)

Probe Name	**Description**
wait-start	Probe that fires immediately before a thread begins to wait pending completion of a given I/O request. The buf(9S) structure corresponding to the I/O request for which the thread will wait is pointed to by the args[0] argument. The devinfo_t structure of the device to which the I/O was issued is pointed to by the args[1] argument. The fileinto_t structure of the file that corresponds to the I/O request is pointed to by the args[2] argument. Some time after the wait-start probe fires, the wait-done probe fires in the same thread.
wait-done	Probe that fires immediately after a thread wakes up from waiting for a pending completion of a given I/O request. The buf(9S) structure corresponding to the I/O request for which the thread was waiting is pointed to by the args[0] argument. The devinfo_t structure of the device to which the I/O was issued is pointed to by the args[1] argument. The fileinfo_t structure of the file that corresponds to the I/O request is pointed to by the args[2] argument. Some time after the wait-start probe fires, the wait-done probe fires in the same thread.

Information Available When io Probes Fire

The io probes fire for all I/O requests to peripheral devices, and for all file read and file write requests to an NFS server (except for metadata requests, such as readdir(3C)).

The io provider uses three I/O structures: the buf(9S) structure, the devinfo_t structure, and the fileinfo_t structure.

When the io probes fire, the following arguments are made available:

- args[0] – Set to point to the buf(9S) structure corresponding to the I/O request.
- args[1] – Set to point to the devinfo_t structure of the device to which the I/O was issued.
- args[2] – Set to point to the fileinfo_t structure containing file system related information regarding the issued I/O request.

The buf(9S) Structure

The buf(9S) structure is the abstraction that describes an I/O request. The address of this structure is made available to your D programs through the args[0] argument. Here is its definition:

```
struct buf {
    int b_flags;            /* flags */
    size t b_bcount;        /* number of bytes */
    caddr_t b_addr;         /* buffer address */
    uint64_t b_blkno;       /* expanded block # on device */
    uint64_t b_lblkno;      /* block # on device */
    size_t b_resid;         /* # of bytes not transferred */
    size t b_bufsize;       /* size of allocated buffer */
    caddr_t b_iodone;       /* I/O completion routine */
    int b_error;            /* expanded error field */
    dev_t b_edev;           /* extended device */
}
```

The b_flags member indicates the state of the I/O buffer and consists of a bitwise OR operator of different state values.

Table 7-5 shows the valid state values for the b_flags field.

Table 7-5 The b_flags Field Values

Flag Value	Description
B_DONE	Indicates the data transfer has completed.
B_ERROR	Indicates an I/O transfer error. It is set in conjunction with the b_error field.
B_PAGEIO	Indicates the buffer is being used in a paged I/O request. See the description of the b_addr field (Table 7-6) for more information.
B_PHYS	Indicates the buffer is being used for physical (direct) I/O to a user data area.
B_READ	Indicates that data is to be read from the peripheral device into main memory.
B_WRITE	Indicates that the data is to be transferred from main memory to the peripheral device.

Table 7-6 shows the field descriptions for the buf(9S) structure.

Table 7-6 The buf(9S) Structure Field Descriptions

Field	**Description**
b_bcount	Indicates the number of bytes to be transferred as part of the I/O request.
b_addr	Indicates the virtual address of the I/O request, unless B_PAGEIO is set. The address is a kernel virtual address unless B_PHYS is set, in which case it is a user virtual address. If B_PAGEIO is set, the b_addr field contains kernel private data. Note that either B_PHYS or B_PAGEIO or neither, can be set, but not both.
b_lblkno	Identifies which logical block on the device is to be accessed. The mapping from a logical block to a physical block (cylinder, track, and so on) is defined by the device.
b_resid	Indicates the number of bytes not transferred because of an error.
b_bufsize	Contains the size of the allocated buffer.
b_iodone	Identifies a specific routine in the kernel that is called when the I/O is complete.
b_error	Holds an error code returned from the driver in the event of an I/O error. b_error is set in conjunction with the B_ERROR bit set in the b_f1ags member.
b_edev	Contains the major and minor device numbers of the device accessed. Consumers can use the D built-in functions getmajor() and getminor() to extract the major and minor device numbers from the b_edev field.

The devinfo_t Structure

The devinfo_t structure provides information about a device. A pointer to this structure is available to D programs through the args[1] argument. Its members are as follows:

```
typedef struct devinfo {
    int dev_major;          /* major number */
    inc dev_minor;          /* minor number */
    inc dev_instance;       /* instance number */
    srring dev_name;        /* name of device */
    string dev_statname;    /* name of device + instance/minor */
    string dev_pathname;    /* pathname of device */
} devinfo_t;
```

Table 7-7 shows the field descriptions for the devinfo_t structure.

Table 7-7 The devinfo_t Structure Field Descriptions

Field	Description
dev_major	Indicates the major number of the device; see getmajor(9F).
dev_minor	Indicates the minor number of the device; see qetminor(9F).
dev_instance	Indicates the instance number of the device. The instance of a device is different from the minor number: where the minor number is an abstraction managed by the device driver, the instance number is a property of the device node. Device node instance numbers can be displayed with the prtconf(lM) command.
dev_name	Indicates the name of the device driver that manages the device. (Device driver names can be viewed with the -D option to prtconf(1M).)
dev_statname	Indicates the name of the device as reported by the iostat(1M) command. This name also corresponds to the name of the device as reported by the kstat(1M) command. This field is provided to enable aberrant iostat or kstat output to be correlated to actual I/O activity.
dev_pathname	Indicates the complete path of the device.

The fileinfo_t Structure

The fileinfo_t structure provides information about a file. The file to which an I/O corresponds is pointed to by the args[2] argument in the start, done, wait-start, and wait-done probes. Note that file information is contingent upon the file system providing this information when dispatching I/O requests; some file systems, especially third-party file systems, do not provide the information. Moreover, I/O requests for which there is no file information can emanate from the file system. For example, I/O to file system metadata is not associated with a specific file. Following is the definition of the fileinfo_t structure:

```
typedef struct fileinfo {
    strinq fi_name;         /* name (basename of fi_pathname) */
    string fi_dirname       /* directory (dirname of fi_pathname) */
    string fi_pathname;     /* full pathname */
    offset_t fi_offset;     /* offset within file */
    string fi_fs;           /* filesystem */
    string fi_mount         /* mount point of file system */
} fileinfo_t;
```

Table 7-8 shows the field descriptions for the fileinfo_t structure.

Table 7-8 The fileinfo_t Structure Field Descriptions

Field	**Description**
fi_name	Contains the name of the file without any directory components. If there is no file information associated with an I/O, the fi_name field is set to the string “<none>.” In rare cases, the pathname associated with a file is unknown; in this case, the fi_name field is set to the string “<unknown>.”
fi_dirname	Contains only the directory component of the file name. As with fi_name this can be set to “<none>” if there is no file information present, or to “<unknown>” if the pathname associated with the file is not known.
fi_pathname	Contains the complete pathname to the file. As with fi_name, this can be set to “<none>” if there is no file information present, or to “<unknown>” if the pathname associated with the file is not known.
fi_offset	Contains the offset within the file, or -1 if file information is not present or if the offset is otherwise unspecified by the file system.

How to Find I/O Problems

Consider the following output from the `iostat`(1M) command.

```
                  extended device statistics
device       r/s    w/s   kr/s   kw/s wait actv  svc_t  %w  %b
fd0          0.0    0.0    0.0    0.0  0.0  0.0    0.0   0   0
sd0          2.5  168.7   20.0 10937.7  0.0  3.7   21.7   0  75
sd2        106.6    0.0 4319.9    0.0  0.0  0.7    6.5   0  54
sd15         0.0    0.0    0.0    0.0  0.0  0.0    0.0   0   0
nfs1         0.0    0.0    0.0    0.0  0.0  0.0    0.0   0   0
                  extended device statistics
device       r/s    w/s   kr/s   kw/s wait actv  svc_t  %w  %b
fd0          0.0    0.0    0.0    0.0  0.0  0.0    0.0   0   0
sd0          0.5  168.7    4.0 16162.5  0.0  9.6   56.9   0  72
sd2         80.9    0.0 7570.5    0.0  0.0  1.1   13.2   0  68
sd15         0.0    0.0    0.0    0.0  0.0  0.0    0.0   0   0
nfs1         0.0    0.0    0.0    0.0  0.0  0.0    0.0   0   0
                  extended device statistics
device       r/s    w/s   kr/s   kw/s wait actv  svc_t  %w  %b
fd0          0.0    0.0    0.0    0.0  0.0  0.0    0.0   0   0
sd0          1.0  166.3    8.1 18973.0  0.0 24.5  146.5   1  88
sd2         43.8    0.0 10949.6    0.0  0.0  0.9   20.4   0  62
sd15         0.0    0.0    0.0    0.0  0.0  0.0    0.0   0   0
nfs1         0.0    0.0    0.0    0.0  0.0  0.0    0.0   0   0
                  extended device statistics
device       r/s    w/s   kr/s   kw/s wait actv  svc_t  %w  %b
fd0          0.0    0.0    0.0    0.0  0.0  0.0    0.0   0   0
sd0          1.0  189.5    8.0 11047.6  0.0  2.7   14.4   0  67
sd2        129.5    0.5 2836.3   14.5  0.0  0.7    5.6   0  59
sd15         0.0    0.0    0.0    0.0  0.0  0.0    0.0   0   0
nfs1         0.0    0.0    0.0    0.0  0.0  0.0    0.0   0   0
^C
```

This output indicates that a large amount of data is being read from disk drive `sd2` and written to disk drive `sd0`. Someone appears to be transferring many megabytes of data between these two drives. Both disks are sustained over 50% busy. Is someone running a file transfer command such as `tar`(1), `cpio`(1), `cp`(1), or `dd`(1M)? The `iosnoop.d` D script enables you to determine who is performing this I/O.

The iosnoop.d D Script

The following D script displays data that enables you to determine which commands are running, what type of I/O those commands are performing, and which peripheral devices are involved.

```
# cat iosnoop.d
#!/usr/sbin/dtrace -qs
BEGIN
{
printf("%16s %5s %40s %10s %2s %7s\n", "COMMAND", "PID", "FILE",
"DEVICE", "RW", "MS");
}

io:::start
{
start[args[0]->b_edev, args[0]->b_blkno] = timestamp;
command[args[0]->b_edev, args[0]->b_blkno] = execname;
mypid[args[0]->b_edev, args[0]->b_blkno] = pid;
}

io:::done
/start[args[0]->b_edev, args[0]->b_blkno]/
{
elapsed = timestamp - start[args[0]->b_edev, args[0]->b_blkno];
printf("%16s %5d %40s %10s %2s %3d.%03d\n", command[args[0]->b_edev,
args[0]->b_blkno], mypid[args[0]->b_edev, args[0]->b_blkno],
args[2]->fi_pathname, args[1]->dev_statname,
args[0]->b_flags&B_READ? "R": "W", elapsed/1000000, (elapsed/1000)%1000);
start[args[0]->b_edev, args[0]->b_blkno] = 0; /* free memory*/
command[args[0]->b_edev, args[0]->b_blkno] = 0; /* free memory */
mypid[args[0]->b_edev, args[0]->b_blkno] = 0; /* free memory */
}
```

You can decipher this D script as follows:

- You use the BEGIN probe to print out column headings.
- You use an associative array to store the nanosecond timestamp of when a particular I/O starts from a specific device. You must also store the executable name and PID of the command issuing the I/O request; this information is not available at I/O completion time because you are running in the context of an interrupt handler.
- When the I/O is done you determine the elapsed time and then print out the relevant information.

- You retrieve the file undergoing the I/O from the fileinfo_t structure; the args[2] argument is set up to point to the fileinfo_t structure when the done probe fires.
- You retrieve the iostat-compatible device name from the devinfo_t structure, which is pointed to by the args[1] argument.
- You use a D conditional expression to display "R" or "W" based on testing the B_READ bit in the b_flags field of the buf structure, which is pointed to by the args[0] argument.
- You use the D *modulo* operator (%) to determine the fractional portion of the time in milliseconds.
- Finally, you reset the start associative array element to prepare for the next I/O. Setting an associative array element to zero also deallocates the underlying dynamic memory that was being used.

The following output results from running the previous iosnoop.d script. It clearly shows who is performing the I/O operations. Someone is copying the shared object files from /usr/lib on drive sd2 to a backup directory on drive sd0.

```
# ./iosnoop.d
      COMMAND   PID                             FILE     DEVICE RW      MS
         bash   725                    /usr/bin/bash        sd2  R   9.471
         bash   725                         /usr/lib        sd2  R   7.128
         bash   725                         /usr/lib        sd2  R   3.193
         bash   725                         /usr/lib        sd2  R  11.283
         bash   725                   /lib/libc.so.1        sd2  R   7.696
         bash   725                 /lib/libnsl.so.1        sd2  R  10.293
         bash   768                 /lib/libnsl.so.1        sd2  R   0.582
           cp   768                   /lib/libc.so.1        sd2  R  10.154
           cp   768                   /lib/libc.so.1        sd2  R   7.262
           cp   768                   /lib/libc.so.1        sd2  R   9.914
           cp   768                /usr/lib/0@0.so.1        sd2  R   9.270
           cp   768                /usr/lib/0@0.so.1        sd2  R  13.654
           cp   768         /mnt/lib.backup/0@0.so.1        sd0  W   2.431
           cp   768                   /usr/lib/ld.so        sd2  R   6.890
           cp   768                   /usr/lib/ld.so        sd2  R   7.085
           cp   768                   /usr/lib/ld.so        sd2  R   0.376
           cp   768             /mnt/lib.backup/ld.so        sd0  W   6.698
           cp   768             /mnt/lib.backup/ld.so        sd0  W   6.437
           cp   768           /mnt/lib.backup/ld.so.1        sd0  W   4.394
           cp   768                        <unknown>        sd2  R   2.206
           cp   768           /mnt/lib.backup/ld.so.1        sd0  W   8.479
           cp   768           /mnt/lib.backup/ld.so.1        sd0  W   8.440
           cp   768              /usr/lib/lib300.so.1        sd2  R   5.771
           cp   768              /usr/lib/lib300.so.1        sd2  R   6.003
           cp   768              /usr/lib/lib300.so.1        sd2  R   0.530
```

```
              cp   768                     /usr/lib/lib300.so.1        sd2  R   7.912
              cp   768                                <unknown>        sd2  R   3.014
              cp   768                /mnt/lib.backup/lib300.so        sd0  W   7.861
              cp   768              /mnt/lib.backup/lib300.so.1        sd0  W   6.794
              cp   768                    /usr/lib/lib300s.so.1        sd2  R   3.326
              cp   768                    /usr/lib/lib300s.so.1        sd2  R   3.525
              cp   768                    /usr/lib/lib300s.so.1        sd2  R   0.553
              cp   768                    /usr/lib/lib300s.so.1        sd2  R   7.397
              cp   768               /mnt/lib.backup/lib300s.so        sd0  W   2.996
              cp   768             /mnt/lib.backup/lib300s.so.1        sd0  W   1.970
...
              cp   768                    /usr/dt/lib/libXm.so.3        sd2  R  32.020
              cp   768                    /usr/dt/lib/libXm.so.3        sd2  R   6.471
              cp   768                    /usr/dt/lib/libXm.so.3        sd2  R  14.494
              cp   768                                   <none>        sd0  R  10.184
              cp   768                    /usr/dt/lib/libXm.so.3        sd2  R  22.211
              cp   768            /mnt/lib.backup/libXm.so.1.2        sd0  W   9.777
              cp   768                    /usr/dt/lib/libXm.so.3        sd2  R  28.813
              cp   768            /mnt/lib.backup/libXm.so.1.2        sd0  W  26.279
              cp   768            /mnt/lib.backup/libXm.so.1.2        sd0  W  24.141
              cp   768            /mnt/lib.backup/libXm.so.1.2        sd0  W  22.075
              cp   768            /mnt/lib.backup/libXm.so.1.2        sd0  W  19.989
              cp   768            /mnt/lib.backup/libXm.so.1.2        sd0  W  21.710
              cp   768            /mnt/lib.backup/libXm.so.1.2        sd0  W  39.809
              cp   768            /mnt/lib.backup/libXm.so.1.2        sd0  W  37.459
              cp   768            /mnt/lib.backup/libXm.so.1.2        sd0  W  32.631
              cp   768            /mnt/lib.backup/libXm.so.1.2        sd0  W  30.378
              cp   768            /mnt/lib.backup/libXm.so.1.2        sd0  W  28.308
              cp   768            /mnt/lib.backup/libXm.so.1.2        sd0  W  29.701
              cp   768            /mnt/lib.backup/libXm.so.1.2        sd0  W  28.327
              cp   768                                <unknown>        sd2  R  24.986
              cp   768            /mnt/lib.backup/libXm.so.1.2        sd0  W  28.021
              cp   768            /mnt/lib.backup/libXm.so.1.2        sd0  W  26.601
              cp   768              /mnt/lib.backup/libXm.so.3        sd0  W   5.353
              cp   768              /mnt/lib.backup/libXm.so.3        sd0  W   4.603
              cp   768              /mnt/lib.backup/libXm.so.3        sd0  W  13.232
              cp   768              /mnt/lib.backup/libXm.so.3        sd0  W  11.242
              cp   768              /mnt/lib.backup/libXm.so.3        sd0  W  12.412
...
              cp   768       /usr/lib/libgtk-x11-2.0.so.0.100.0        sd2  R   2.374
              cp   768      /mnt/lib.backup/libgthread-2.0.so.0        sd0  W   7.732
              cp   768  /mnt/lib.backup/libgthread-2.0.so.0.7.0        sd0  W   7.605
              cp   768                                   <none>        sd2  R  10.678
              cp   768       /usr/lib/libgtk-x11-2.0.so.0.100.0        sd2  R   5.677
              cp   768       /usr/lib/libgtk-x11-2.0.so.0.100.0        sd2  R  39.864
              cp   768       /usr/lib/libgtk-x11-2.0.so.0.100.0        sd2  R  61.555
              cp   768       /usr/lib/libgtk-x11-2.0.so.0.100.0        sd2  R  17.175
              cp   768        /mnt/lib.backup/libgtk-x11-2.0.so        sd0  W  44.225
              cp   768        /mnt/lib.backup/libgtk-x11-2.0.so        sd0  W  42.075

^C
```

Obtaining System Call Information

System calls serve as the main interface between user-level applications and the kernel. You can learn much about the system by knowing the system calls that are being issued by the set of running applications.

Note – System calls are documented in Section 2 of the Solaris 10 OS manual pages.

Traditionally, system calls of an application were determined using the `truss`(1) command. The DTrace `syscall` provider, however, enables you to quickly gather more detailed data with which to analyze aberrant behavior related to system calls. For example, not only can DTrace show you the system calls being issued by a given application, but it can also indicate which applications are issuing a given system call. In addition, you can time (in nanoseconds) how long a particular system call takes, such as a `read`(2). These operations cannot be performed with the `truss`(1) command.

The `syscall` Provider

The `syscall` provider makes available a probe at the entry and return of every system call in the system. An example of a fully-specified probe description for the entry probe of the `read`(2) system call is:

```
syscall::read:entry
```

The probe for return from the `read`(2) system call is:

```
syscall::read:return
```

Note that the module name is undefined for the `syscall` provider probes.

System Call Names

The system call names are usually, but not always, the same as those documented in Section 2 of the Solaris 10 OS manual pages. The actual names are listed in the `/etc/name_to_sysnum` system file. Examples of system call names that do not match the manual pages are:

- `rexit` for `exit`(2)
- `gtime` for `time`(2)
- `semsy` for `semctl`(2), `semget`(2), `semids`(2), and `semtimedop`(2)
- `signotify`, which has no manual page, and is used for POSIX.4 message queues
- Large file system calls such as:
  - `creat64` for `creat`(2)
  - `lstat64` for `lstat`(2)
  - `open64` for `open`(2)
  - `mmap64` for `mmap`(2)

Arguments for `entry` and `return` Probes

For the `entry` probes, the arguments (`arg0`, `arg1`, ... `arg`*n*) are the arguments to the system call. For `return` probes, both `arg0` and `arg1` contain the same value: return value from the system call. You can check system call failure in the `return` probe by referencing the `errno` D variable. The following example shows which system calls are failing for which applications and with what `errno` value.

```
# cat errno.d
syscall:::return
/arg0 == -1 && execname != "dtrace"/
{
printf("%-20s %-10s %d\n",execname,probefunc,errno);
}
# dtrace -q -s errno.d
sac                  read       4
ttymon               pause      4
ttymon               read       11
nscd                 lwp_kill   3
in.routed            ioctl      12
in.routed            ioctl      12
tty                  open       2
tty                  stat       2
```

```
bash                    setpgrp    13
bash                    waitsys    10
bash                    stat64     2
snmpd                   ioctl      12
^C
```

The `errno.d` D program has a predicate that uses the "AND" operator: &&. The predicate states that the return from the system call must be `-1`, which is how all system calls indicate failure, and that the process executable name cannot be `dtrace`. The `printf` built-in D function uses the `%-20s` and `%-10s` format specifications to left-justify the strings in the given minimum column width.

D Script Example Using the `syscall` Provider

The following simple D script counts the number of system calls being issued system wide.

```
# cat syscall.d
#!/usr/sbin/dtrace -qs
syscall:::entry
{
@[probefunc] = count();
}
# ./syscall.d
^C
  mmap64                                                      1
  mkdir                                                       1
  umask                                                       1
  getloadavg                                                  1
  getdents64                                                  2
...
  stat                                                     1754
  ioctl                                                    1956
  close                                                    2708
  write                                                    2733
  mmap                                                     3006
  read                                                     3880
  sigaction                                                7886
  brk                                                     12695
```

The output indicates that the majority of the system calls are setting up signal handling (`sigaction`(2)) or growing the heap (`brk`(2)). The following D script enables you to discover who is making the `brk`(2) system calls.

```
# cat brk.d
#!/usr/sbin/dtrace -qs
syscall::brk:entry
{
@[execname] = count();
}
# ./brk.d
^C
  dtrace                                                         6
  prstat                                                        22
  nroff                                                         48
  cat                                                           48
  tbl                                                          142
  eqn                                                          144
  rm                                                           166
  ln                                                           166
  col                                                          222
  expr                                                         332
  head                                                         492
  fgrep                                                        492
  dirname                                                      581
  grep                                                         722
  instant                                                      738
  sh                                                           917
  nawk                                                         984
  sgml2roff                                                   1259
  nsgmls                                                     13296
# ps -ef | grep nsgmls
    root   591   590   2 07:56:32 pts/2       0:00 /usr/lib/sgml/nsgmls -
gl -m/usr/share/lib/sgml/locale/C/dtds/catalog -E0 /usr/s
# man nsgmls
No manual entry for nsgmls.
# man -k sgml
sgml           sgml (5)        - Standard Generalized Markup Language
solbook        sgml (5)        - Standard Generalized Markup Language
```

Apparently some process is working with the Standard Generalized Markup Language (SGML). Use the `ptree` command to see who is creating this process:

```
# ptree 591
#
```

The `ptree` command returns no results because the `nsgmls` process is too short-lived for the command to be run on it. You have learned, however, that the problem is not a long-lived process causing a memory leak. Now write a quick D script to print out the ancestry. You must keep trying the next previous parent iteratively, because many of the other processes involved are also short-lived.

Note – This particular D script fails if an ancestor does not exist. This is because the top ancestor, the `sched` process has no parent. You cannot harm the kernel even if a D script uses a bad pointer. The intent of this example is to show how you can quickly create custom D scripts to answer questions about system behavior. Many of your D scripts will be throw-away scripts that you will not re-use. You can fix the script by testing each parent pointer with a predicate before printing. You will see this fix later with the `ancestors3.d` D script.

```
# cat ancestors.d
#!/usr/sbin/dtrace -qs
syscall::brk:entry
/execname == "nsgmls"/
{
printf("process: %s\n", curthread->t_procp->p_user.u_psargs);
printf("parent: %s\n", curthread->t_procp->p_parent->p_user.u_psargs);
printf("grandparent: %s\n", curthread->t_procp->p_parent->p_parent->p_user.u_psargs);
printf("greatgrandparent: %s\n", curthread->t_procp->p_parent->p_parent->p_parent-
>p_user.u_psargs);
printf("greatgreatgrandparent: %s\n", curthread->t_procp->p_parent->p_parent->p_parent-
>p_parent->p_user.u_psargs);
printf("greatgreatgreatgrandparent: %s\n", curthread->t_procp->p_parent->p_parent-
>p_parent->p_parent->p_parent->p_user.u_psargs);
exit(0);
}
# ./ancestors.d
process: /usr/lib/sgml/nsgmls -gl -m/usr/share/lib/sgml/locale/C/dtds/catalog -E0 /usr/s
parent: /usr/lib/sgml/instant -d -c/usr/share/lib/sgml/locale/C/transpec/roff.cmap -s/u
grandparent: /bin/sh /usr/lib/sgml/sgml2roff /usr/share/man/sman4/rt_dptbl.4
greatgrandparent: sh -c cd /usr/share/man; /usr/lib/sgml/sgml2roff
/usr/share/man/sman4/rt_dptbl.
greatgreatgrandparent: catman
greatgreatgreatgrandparent: bash

# ps -ef | grep catman
    root  2333  2332   1 08:26:05 pts/1       0:03 catman
    root 16984  2880   0 08:41:10 pts/2       0:00 grep catman
# ptree 2333
299   /usr/sbin/inetd -s
  2324  in.rlogind
    2326  -sh
      2332  bash
        2333  catman
        17232 sh -c cd /usr/share/man; rm -f /usr/share/man/cat4/variables.4;
ln -s ../cat4/e
         17235 sh -c cd /usr/share/man; rm -f /usr/share/man/cat4/variables.4;
ln -s ../cat4/e
```

The previous output indicates that all of the brk(2) system calls resulted from the catman(1M) command, creating many short-lived children that issued this system call.

The curthread built-in D variable gives access to the address of the running kernel thread. Like the C language, the D language accesses members of a structure with the -> symbol when you have a pointer to that structure. Through this pointer to the kernel kthread_t structure, you can access the process name and arguments (kept in the proc_t structure's p_user structure) as well as any parent, grandparent, great-grandparent, and so on. To do this you follow the parent pointers back. Refer to the <sys/thread.h>, <sys/proc.h> and <sys/user.h> header files for details of these fields.

Figure 7-1 shows a diagram of the kernel data structures being accessed by this example.

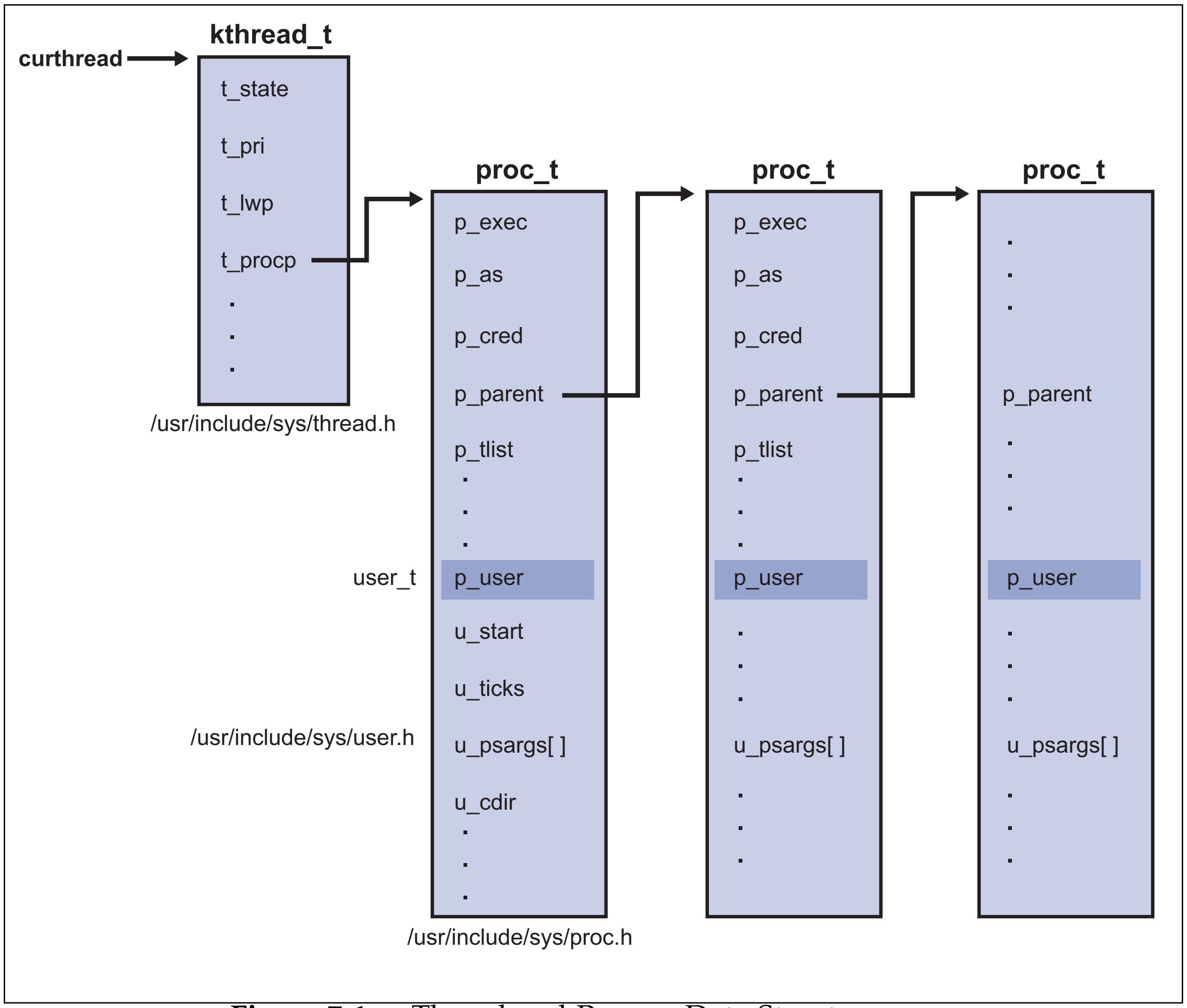


Figure 7-1 Thread and Process Data Structures

New Approach to Analyzing Transient Failures

As the previous example demonstrates, each result obtained from using the DTrace facility can lead to further questions, which are answered with available commands or with new D programs that you can write quickly. In this way, the DTrace facility significantly shortens the diagnostic loop:

hypothesis->instrumentation->data gathering->analysis->hypothesis

This tightened loop introduces a new paradigm for diagnosing transient failures. It enables the emphasis to shift from instrumentation to hypothesis, which is less labor intensive.

D Language Variables

The D language has five basic variable types:

- Scalar variables – Have fixed size values such as integers and pointers
- Associative arrays – Store values indexed by one or more keys, similar to aggregations
- Thread-local variables – Have one name, but storage is local to each separate kernel thread
- Clause-local variables – Appear when an action block is entered; storage is reclaimed after leaving probe clause
- Kernel external variables – Have access to all kernel global and static variables

The `command` and `mypid` variables in the `iosnoop.d` script are examples of D global associative array variables. Clause-local variables are similar to automatic or local variables in the C Language. They come into existence when an action block (tied to a specific probe) is entered and their storage is reclaimed when the action block is left. They help save storage and are quicker to access than associative arrays.

Note – For more information on D variables, refer to the *Solaris Dynamic Tracing Guide*, part number 817-6223-05.

You can access kernel global and static variables within your D programs. To access these external variables, you prefix the global kernel variable with the ` (back quote or grave accent) character. For example, to reference the `freemem` kernel global variable use: ``‘freemem``. If the variable is part of a kernel module that conflicts with other module variable names, use the ` character between the module name and the variable name. For example, `sd‘sd_state` references the `sd_state` variable within the `sd` kernel module.

Associative Arrays

Associative arrays enable the storing of scalar values in elements of an array (or table) that are identified by one or more sequences of comma-separated key fields (an *n-tuple*). The keys can be any combination of strings or integers. The following code example shows the use of an associative array to track how often any command issues more than a given number of any single system call:

```
# cat assoc2.d
#!/usr/sbin/dtrace -qs
syscall:::entry
{
++namesys[pid,probefunc];
x=namesys[pid,probefunc] > 5000 ? 1 : 0;
}
syscall:::entry
/x && execname != "dtrace"/
{
printf("Process: %d %s has just made more than 5000 %s calls\n", pid,
execname, probefunc);
namesys[pid,probefunc]=0;  /* lets reset the count and free memory */
}
# ./assoc2.d
Process: 14837 find has just made more than 5000 lstat64 calls
Process: 14837 find has just made more than 5000 lstat64 calls
Process: 14854 ls has just made more than 5000 lstat64 calls
Process: 14854 ls has just made more than 5000 acl calls
Process: 14854 ls has just made more than 5000 lstat64 calls

             ^C
```

The `assoc2.d D` program uses an associative array indexed by the unique combination of executable name and system call name. The `++` operator is incrementing the array element by one each time a process with that PID is making that system call. The array element, like all variables (except clause-local variables), is initialized to `0`. The second statement in the action block uses a *conditional expression* that has three parts:

```
expression?value1:value2
```

A conditional expression has the value of *`value1`* when the D *`expression`* is nonzero (true), and has the value of *`value2`* when the *`expression`* is zero (false). Therefore, in the `assoc2.d` D program, the global scalar variable `x` is `1` when that element of the associative array is greater than `5000,` and `0` when it is not greater than `5000`. The next action block is only executed if `x` is not `0` and the executable name is not `dtrace`. After printing the command that made more than `5000` of a given system call, you reset the array element to `0` to begin counting again. Note that a comment is used in this D program. Like comments in the C language, a comment in the D language is text that is enclosed between `/*` and `*/`.

Thread-Local Variables

Thread-local variables are useful when you wish to enable a probe and mark with a tag every thread that fires the probe. Thread-local variables share a common name but refer to separate data storage associated with each thread. Thread-local variables are referenced with the special identifier `self` followed by the two characters `->`, as shown in the following example:

```
syscall::read:entry
{
self->read = 1;
}
syscall::read:return
/self->read/
{
printf(“Same thread is returning from read”);
}
```

How to Time a System Call

Thread-local variables enable you to determine the amount of time a thread spends in any particular system call. The following example times how long the `grep`(1) command takes in each `read`(2) system call. It also displays the number of bytes read (`arg0` is the return value of `read`).

```
# cat timegrep.d
BEGIN
{
printf("size\ttime\n");
}
syscall::read:entry
/execname == "grep"/
{
self->start = timestamp;
}
syscall::read:return
/self->start/
{
printf("%d\t%d\n", arg0, timestamp-self->start);
self->start = 0; /* free memory */
}
# dtrace -q -s timegrep.d
size    time
8192    7108972
319     1526616
0       12112
3293    5663329
0       18816
^C
```

The first read took 7,108,972 nanoseconds or 7.1 milliseconds, which is reasonable for an 8-Kbyte disk read. As you might expect, the first read of `0` bytes took only 12 microseconds.

The next example uses an associative array to time every system call performed by the `grep` command.

```
# cat timesys.d
BEGIN
{
printf("System Call Times for grep:\n\n");
printf("%20s\t%10s\n", "Syscall", "Microseconds");
}
syscall:::entry
```

```
/execname == "grep"/
{
self->name[probefunc] = timestamp;
}
syscall:::return
/self->name[probefunc]/
{
printf("%20s\t%10d\n", probefunc,
(timestamp-self->name[probefunc])/1000);
self->name[probefunc] = 0; /* free memory */
}

# dtrace -q -s timesys.d
System Call Times for grep:

             Syscall    Microseconds
                mmap            50
         resolvepath            47
         resolvepath            67
                stat            37
                open            46
                stat            34
                open            32
...
                 brk            25
              open64            43
                read          8126
                 brk            20
                 brk            28
                read            24
               close            26
^C
```

Predictably, the system call that took the most time was `read`, because of the disk I/O wait time (the second `read` was of 0 bytes).

How to Follow a System Call

You can follow a system call from entry into the kernel through all subsequent internal kernel function calls and returns back to the original point of entry of the system call function. You do this by using the `syscall` and `fbt` providers together with a thread-local variable. The following example traces all of the functions involved in the `read`(2) system call as issued by the `grep`(1) command:

```
# cat follow.d
#!/usr/sbin/dtrace -s

syscall::read:entry
/execname == "grep"/
{
self->start = 1;
}

syscall::read:return
/self->start/
{
exit(0);
}

fbt:::
/self->start/
```

The `fbt` provider probe clause has an empty action. Action blocks are optional in a probe clause. The default action for DTrace tracks every time you enter and return from all kernel functions involved in a `read`(2) system call until you return. Option `-F` of the `dtrace`(1M) command indents the output of each nested function call and shows this with the `->` symbol; it un-indents the output when that function returns back up the call tree and shows this with the `<-` symbol.

```
# dtrace -F -s follow.d
dtrace: script 'follow.d' matched 35974 probes
CPU FUNCTION
  0  -> read32
  0  <- read32
  0  -> read
  0    -> getf
  0      -> set_active_fd
  0      <- set_active_fd
  0    <- getf
...
```

```
  0    <- ufs_rwlock
  0    -> fop_read
  0    <- fop_read
  0    -> ufs_read
  0      -> ufs_lockfs_begin
...
  0      -> rdip
  0        -> rw_write_held
  0        <- rw_write_held
  0        -> segmap_getmapflt
  0          -> get_free_smp
  0            -> grab_smp
  0              -> segmap_hashout
...
  0                  <- sfmmu_kpme_lookup
  0                  -> sfmmu_kpme_sub
...
  0              <- page_unlock
  0            <- grab_smp
  0            -> segmap_pagefree
  0              -> page_lookup_nowait
  0                -> page_trylock
...
  0          <- segmap_hashin
  0          -> segkpm_create_va
  0          <- segkpm_create_va
  0          -> fop_getpage
  0            -> ufs_getpage
  0              -> ufs_lockfs_begin_getpage
  0                -> tsd_get
...
  0              <- page_exists
  0              -> page_lookup
  0              <- page_lookup
  0              -> page_lookup_create
  0              <- page_lookup_create
  0              -> ufs_getpage_miss
  0                -> bmap_read
  0                  -> findextent
  0                  <- findextent
  0                <- bmap_read
  0                -> pvn_read_kluster
  0                  -> page_create_va
  0                    -> lgrp_mem_hand
...
  0                    <- page_add
  0                  <- page_create_va
  0                <- pvn_read_kluster
  0                -> pagezero
  0                  -> ppmapin
  0                    -> sfmmu_get_ppvcolor
  0                    <- sfmmu_get_ppvcolor
  0                    -> hat_memload
  0                      -> sfmmu_memtte
  0                      <- sfmmu_memtte
...
  0                        -> xt_some
  0                        <- xt_some
  0                      <- xt_sync
...
  0                  <- sema_init
  0                <- pageio_setup
  0                -> lufs_read_strategy
  0                  -> logmap_list_get
  0                  <- logmap_list_get
  0                  -> bdev_strategy
  0                    -> bdev_strategy_tnf_probe
  0                    <- bdev_strategy_tnf_probe
  0                  <- bdev_strategy
  0                  -> sdstrategy
  0                    -> getminor
...
  0                        <- drv_usectohz
  0                        -> timeout
  0                        <- timeout
  0                        -> timeout_common
...
  0                        <- getminor
  0                        -> scsi_transport
  0                        <- scsi_transport
  0                        -> glm_scsi_start
  0                          -> ddi_get_devstate
...
  0                                <- ddi_get_soft_state
  0                                -> pci_pbm_dma_sync
  0                                <- pci_pbm_dma_sync
  0                              <- pci_dma_sync
  0                            <- glm_start_cmd
  0                          <- glm_accept_pkt
  0                        <- glm_scsi_start
  0                      <- sd_start_cmds
  0                    <- sd_core_iostart
  0                  <- xbuf_iostart
  0                <- lufs_read_strategy
  0                -> biowait
  0                  -> sema_p
  0                    -> disp_lock_enter
  0                    <- disp_lock_enter
  0                    -> thread_lock_high
  0                    <- thread_lock_high
  0                    -> ts_sleep
  0                    <- ts_sleep
  0                    -> disp_lock_exit_high
  0                    <- disp_lock_exit_high
  0                    -> disp_lock_exit_nopreempt
  0                    <- disp_lock_exit_nopreempt
  0                    -> swtch
  0                      -> disp
  0                        -> disp_lock_enter
  0                        <- disp_lock_enter
  0                        -> disp_lock_exit
  0                        <- disp_lock_exit
  0                        -> disp_getwork
  0                        <- disp_getwork
  0                      <- disp
  0                    <- swtch
  0                    -> resume
  0                    <- resume
  0                    -> disp_lock_enter
,,,
  0          <- hat_page_getattr
  0        <- segmap_getmapflt
  0        -> uiomove
  0          -> xcopyout
  0          <- xcopyout
  0        <- uiomove
  0        -> segmap_release
  0          -> get_smap_kpm
...
  0          <- ufs_imark
  0        <- ufs_itimes_nolock
  0      <- rdip
...
  0      <- cv_broadcast
  0    <- releasef
  0  <- read
  0  <= read
```

Although more than half of the functions were removed from the previous output, the example shows that a great many functions are required to perform a disk file read. Some of these functions are as follows:

- `read` – `read`(2) system call entered
- `ufs_read` – UFS file being read
- `segmap_getmapflt` – Find `segmap` page for the I/O
- `segmap_pagefree` – Free underlying previous physical page tied to this `segmap` virtual page onto the *cachelist* (this policy replaced the old priority paging)
- `ufs_getpage` – Ask UFS to retrieve the page
- `page_lookup` – First check to see if the page is in memory (it is not)
- `page_create_va` – Get new physical page for the I/O
- `hat_memload` – Map the virtual page to the physical page
- `xt_some` – Issue cross-trap call to some CPUs
- `sdstrategy` – Issue Small Computer System Interface (SCSI) command to read page from disk into `segmap` page
- `timeout` – Prepare for SCSI timeout of disk read request
- `glm_scsi_start` – In `glm` host bus adapter driver
- `biowait` – Wait for block I/O
- `sema_p` – Use semaphore to wait for I/O
- `ts_sleep` – Put timesharing (TS) thread on sleep queue
- `swtch` – Do a *context switch* (have thread give up the CPU while it waits for the I/O)
- `disp_getwork` – Find another thread to run while this thread waits for its I/O
- `resume` – I/O has completed and CPU is returned to resume running
- `uimove` – Move data from kernel buffer (page) to user-land buffer
- `segmap_release` – Release `segmap` page for use by another I/O later
- `read` – Read operation ends

Creating D Scripts That Use Arguments

As with shell and other interpretive programming language commands such as the `perl`(1) command, you can use the `dtrace`(1M) command to create executable interpreter files. The file must start with the following line and must have execute permission:

```
#!/usr/sbin/dtrace -s
```

You can specify other options to the `dtrace`(1M) command on this line; be sure, however, to use only one dash (`-`) followed by the options, with `s` being last:

```
#!/usr/sbin/dtrace -qvs
```

You can also specify all options to the `dtrace`(1M) command by using `#pragma` lines inside the D script:

```
# cat mem2.d
#!/usr/sbin/dtrace -s

#pragma D option quiet
#pragma D option verbose

vminfo:::
{
@[execname,probename] = count();
}

END
{
printa("%-20s %-15s %@d\n", @);
}
```

Note – For the list of option names used in `#pragma` lines, see the *Solaris Dynamic Tracing Guide*, part number 817-6223-05.

Built-in Macro Variables

The D compiler defines a set of built-in macro variables that you can refer to inside a D script. These macro variables include:

- `$pid` – Process ID of `dtrace` interpreter running script
- `$ppid` – Parent process ID of `dtrace` interpreter running script
- `$uid` – Real user ID of user running script
- `$gid` – Real group ID of user running script
- `$0` – Name of script
- `$1`, `$2`, `$3`, and so on – First, second, third command-line arguments passed to script

The complete list of D macro variables is given in Appendix B. The following D script uses some of these D macro variables:

```
# cat params.d
#!/usr/sbin/dtrace -s
#pragma D option quiet

tick-2sec
/$1 == $11/
{
printf("name of script: %s\n", $0);
printf("pid of script: %d\n", $pid);
printf("9th arg passed to script: %d\n", $9);
exit(0);
}
# ./params.d 1 2 3 4 5 6 7 8 9 10 1
name of script: ./params.d
pid of script: 5363
9th arg passed to script: 9

# ./params.d 1 2 3 4 5 6 7 8 9 10 11
^C
```

The last invocation of the script did not output anything because the value of the first argument did not match the value of the eleventh argument. The following invocations show that the type and number of arguments must match those referenced inside the D script. This is an example of the error-checking capability of the DTrace facility:

```
# ./params.d 1 2 3 4 5 6 7 8 9
dtrace: failed to compile script ./params.d: line 6: macro argument $11
is not defined
# ./params.d 1 2 3 4 5 6 7 8 9 10 11 12 13
dtrace: failed to compile script ./params.d: line 12: extraneous argument
'13' ($13 is not referenced)
# ./params.d a b c d e f g h i j k
dtrace: failed to compile script ./params.d: line 6: failed to resolve a:
Unknown variable name
```

PID Argument Example

The following example passes the PID of a running `vi` process to the `syscalls2.d` D script. You use the `pgrep` command to determine the PID of the `vi` process. The D script terminates when the `vi` command exits.

```
# cat syscalls2.d
#!/usr/sbin/dtrace -qs

syscall:::entry
/pid == $1/
{
@[probefunc] = count();
}
syscall::rexit:entry
/pid == $1/
{
exit(0);
}
# pgrep vi
2208
# ./syscalls2.d 2208

  rexit                                                               1
  setpgrp                                                             1
  creat                                                               1
  getpid                                                              1
  open                                                                1
  lstat64                                                             1
  stat64                                                         1
  fdsync                                                         1
  unlink                                                         2
  close                                                          2
  alarm                                                          2
  lseek                                                          3
  sigaction                                                      5
  ioctl                                                         45
  read                                                         143
  write                                                        178
#
```

Executable Name Argument Example

In the following example the `ancestors.d` D script is modified to make it more general. Remember that this script was created because the processes involved were too short-lived for a `ptree` command to be executed on them.

The modified script can retrieve the ancestry back to the great-great-great-grandparent of any process you catch making any specified system call. You must pass the `execname` argument in quoted form: `'"`*`process_name`*`"'`. Therefore you must surround the double-quoted process name with single quotes so that the shell passes it to the script literally. Based on how the system call name is used, you do not need to quote the second argument. If you replace the `$1` reference with `$$1` then you can enter the process name on the command line without any quotes. The `$$` instead of `$` means to treat the argument as a string surrounded by double quotes.

```
# cat ancestors2.d
#!/usr/sbin/dtrace -qs

syscall::$2:entry
/execname == $1/
{
printf("process: %s\n", curthread->t_procp->p_user.u_psargs);
printf("parent: %s\n", curthread->t_procp->p_parent->p_user.u_psargs);
printf("grandparent: %s\n", curthread->t_procp->p_parent->p_parent-
>p_user.u_psargs);
printf("greatgrandparent: %s\n", curthread->t_procp->p_parent->p_parent-
>p_parent->p_user.u_psargs);
printf("greatgreatgrandparent: %s\n", curthread->t_procp->p_parent-
>p_parent->p_parent->p_parent->p_user.u_psargs);
printf("greatgreatgreatgrandparent: %s\n", curthread->t_procp->p_parent-
>p_parent->p_parent->p_parent->p_parent->p_user.u_psargs);
exit(0);
}

# ./ancestors2.d '"nsgmls"' brk
process: /usr/lib/sgml/nsgmls -gl -
m/usr/share/lib/sgml/locale/C/dtds/catalog -E0 /usr/s
parent: /bin/sh /usr/lib/sgml/sgml2roff /usr/share/man/sman2/fork.2
grandparent: /bin/sh /usr/lib/sgml/sgml2roff /usr/share/man/sman2/fork.2
greatgrandparent: sh -c cd /usr/share/man; /usr/lib/sgml/sgml2roff
/usr/share/man/sman2/fork.2
greatgreatgrandparent: catman
greatgreatgreatgrandparent: bash
```

You can run the same script with a different process name and system call, which shows the power of being able to pass in arguments to a D script:

```
# ./ancestors2.d '"vi"' sigaction
process: vi /etc/system
parent: bash
grandparent: -sh
greatgrandparent: in.rlogind
greatgreatgrandparent: /usr/sbin/inetd -s
greatgreatgreatgrandparent: /etc/init -
```

The `ancestors3.d` D script fixes the problem with trying to print non-existent ancestry and also uses the `$$1` syntax for referring to string arguments:

```
# cat ancestors3.d
#!/usr/sbin/dtrace -qs
syscall::$2:entry
/execname == $$1/
{
printf("process: %s\n", curthread->t_procp->p_user.u_psargs);
nextpaddr = curthread->t_procp->p_parent;
}
syscall::$2:entry
/(execname == $$1) && nextpaddr/
{
printf("parent: %s\n", nextpaddr->p_user.u_psargs);
nextpaddr = curthread->t_procp->p_parent->p_parent;
}
syscall::$2:entry
```

```
/(execname == $$1) && nextpaddr/
{
printf("grandparent: %s\n", nextpaddr->p_user.u_psargs);
nextpaddr = curthread->t_procp->p_parent->p_parent->p_parent;
}
syscall::$2:entry
/(execname == $$1) && nextpaddr/
{
printf("greatgrandparent: %s\n", nextpaddr->p_user.u_psargs);
nextpaddr = curthread->t_procp->p_parent->p_parent->p_parent->p_parent;
}
syscall::$2:entry
/(execname == $$1) && nextpaddr/
{
printf("greatgreatgrandparent: %s\n", nextpaddr->p_user.u_psargs);
nextpaddr = curthread->t_procp->p_parent->p_parent->p_parent->p_parent-
>p_parent;
}
syscall::$2:entry
/(execname == $$1) && nextpaddr/
{
printf("greatgreatgreatgrandparent: %s\n", nextpaddr->p_user.u_psargs);
exit(0);
}
# ./ancestors3.d cron read
process: /usr/sbin/cron
parent: /sbin/init
grandparent: sched
process: /usr/sbin/cron
parent: /sbin/init
grandparent: sched
process: /usr/sbin/cron
parent: /sbin/init
grandparent: sched
process: /usr/sbin/cron
parent: /sbin/init
grandparent: sched
^C
```

Custom Monitoring Tools

You can use the `vminfo`, `sysinfo`, and `io` providers to create your own custom tools resembling the `vmstat`(1M), `sar`(1), `mpstat`(1M), and `iostat`(1M) tools.

Example of a Custom Tool Resembling the `vmstat(1M)` Command

The following D script uses the `vminfo` provider to implement a tool similar to the `vmstat`(1M) command. It displays three fields from the `vmstat`(1M) command:

- `free` field – Displays the system's average value of `freemem` in kilobytes
- `re` field – Displays the average page reclaims per second
- `sr` field – Displays the average page scans per second performed by the `page` daemon

```
# cat vm.d
#!/usr/sbin/dtrace -qs

/*
 * Usage: vm.d interval count
 */

BEGIN
{
printf("%8s %8s %8s\n", "free", "re", "sr");
}

tick-1sec
{
++i;
@free["freemem"] = sum(8*`freemem);
}

vminfo:::pgrec
{
++re;
}

vminfo:::scan
{
++sr;
}

tick-1sec
/i == $1/
{
normalize(@free, $1);
printa("%8@d ", @free);
printf("%8d %8d\n", re/i, sr/i);
++n;
i = 0;
re = 0;
sr = 0;
clear(@free);
}

tick-1sec
/n == $2/
{
exit(0);
}
```

```
# ./vm.d 5 12
    free       re       sr
  385296        0        0
  385296        0        0
  385296        0        0
  385296        0        0
  316180        2        0
   22297        1    19040
    1976        2    31727
    1964        3    31727
    1971        2    31727
    1968        3    31727
    1964        3    31727
    1955        4    31728
```

Like the `vmstat`(1M) command, the `vm.d` script expects two arguments: the interval value and a count value. The `i`, `re`, `sr`, and `n` variables are D global scalar variables used for counting. Note the special reference to the kernel's `freemem` variable: `‘freemem`. The script multiplies `‘freemem` by 8 because it sums in units of kilobytes, not pages, and the assumption is that a page is 8 Kbytes in size. The script uses the `sum`() aggregation with the `normalize`() built-in function which divides the current sum by the interval value to get per second averages. The script also clears the running sum of `‘freemem` every interval with the `clear`() builtin function. The `printa`() built-in function which is covered in detail in Appendix A, prints the value of the `sum`() aggregation.

Because you are using integer-truncated arithmetic, you can lose some data. This is also true when using the `vmstat`(1M) command. For example, if there are only four page reclaims in the five-second interval, then the average per second shows as 0. This output shows that the system is experiencing sustained scanning of memory by the `page` daemon, as indicated by the consistently high number of scans per second. It also shows that someone has used most of the free memory within a short period of time, which explains the high scan rates.

Example of a Custom Tool Resembling the `sar -c` Command

The following D script uses the `sysinfo` provider to implement a tool similar to the `sar -c` command.

```
# cat sar-c.d
#!/usr/sbin/dtrace -qs
/*
 * Usage: ./sar-c.d interval count
 */
BEGIN
{
printf("%10s %10s %10s %10s %10s %10s %10s\n", "scall/s", "sread/s",
"swrit/s", "fork/s", "exec/s", "rchar/s", "wchar/s");
rchar = 0;
wchar = 0;
}

syscall:::entry
{
++scall;
}

sysinfo:::sysread
{
++sread;
}

sysinfo:::syswrite
{
++swrit;
}

sysinfo:::sysfork
{
++fork;
}

sysinfo:::sysexec
{
++exec;
}

sysinfo:::readch
{
rchar = rchar + arg0;
}

sysinfo:::writech
{
wchar = wchar + arg0;
}
```

```
tick-1sec
{
++i;
}

tick-1sec
/i == $1/
{
++n;
printf("%10d %10d %10d %10d %10d %10d %10d\n", scall/i,
sread/i, swrit/i, fork/i, exec/i, rchar/i, wchar/i);
i = 0;
scall = 0;
sread = 0;
swrit = 0;
fork = 0;
exec = 0;
rchar = 0;
wchar = 0;
}

tick-1sec
/n == $2/
{
exit(0);
}
# ./sar-c.d 5 6
   scall/s    sread/s    swrit/s     fork/s     exec/s    rchar/s    wchar/s
        43          0          0          0          0          0         15
        70          1          2          0          0          1         32
        42          2          2          0          0          2         17
        75          0          1          0          0        351         39
       436         26         34          3          3       3329        317
        38          0          0          0          0          0         15
```

Section IV: Solaris™ 10 Networking

Objectives

Upon completion of this section, you should be able to:

- Describe the changes to Internet Protocol (IP) in the OS
- Describe network filesystem changes (NFS) in the OS
- Describe the security feature changes in the OS
- Describe other networking featrue changes in the OS

Module 8

Changes to Internet Protocol Features

Objectives

This module is an overview of the new features included in the Solaris™ 10 Operating System (Solaris 10 OS) that are part of the Internet Protocol (IP) networking category.

Upon completion of this module, you should be able to:

- Understand the IP Quality of Service (IPQoS) feature
- Explain the FireEngine project's changes to the Solaris OS Transmission Control Protocol/Internet Protocol (TCP/IP) stack
- Explain the addition of TCP multi-data transmission
- Explain other miscellaneous changes to IP

IPQoS

IPQoS enables system administrators to provide different levels of network service to customers and to critical applications. By using IPQoS, the administrator can set up service-level agreements to provide clients with varying levels of service that are based on a price structure. A company could also use IPQoS to prioritize among applications so that critical applications get a higher quality of service than less critical applications.

The IPQoS architecture is an implementation of the Internet Engineering Task Force (IETF) Differentiated (`diffserv`) model as defined in Request For Comments (RFC) 2475. Scalable service differentiation in the Internet is achieved by means of IP-layer packet marking using the Differentiated Services (DS) field in the Internet Protocol version 4 (IPv4) and IPv6 headers. Packets are classified and marked to receive a particular per-hop forwarding behavior (PHB) on nodes along their path.

IPQoS Terminology

You should become familiar with the terms in Table 8-1 that are used in this module:

Table 8-1 IPQoS Terminology

Term	**Definitions**
IPQoS	IP Quality of Service
QoS	Quality of Service
CoS	Class of Service. An interface flag that indicates if an interface supports any form of Class of Service marking
DS	Differentiated Services
DS codepoint	A specific value of the DSCP portion of the DS field used to select a PHB
DSCP	Differentiated Services Code Point
PHB	Per-Hop-Behavior. The externally observable forwarding applied at a DS-compliant node to a DS behavior aggregate
AF PHB	Assured Forwarding PHB

Table 8-1 IPQoS Terminology (Continued)

Term	Definitions
EF PHB	Expedited Forwarding PHB
ToS	Type of Service field in the IPv4 header has been officially changed to DS field
RFC 2475	Request For Comments: 2475; *An Architecture for Differentiated Services*

The IPQoS is an implementation of the diffserv model that includes:

- Multifield classifier – Selects actions based on filters that configure the QoS policy of your organization
- Meter – Measures the network traffic in compliance with the Diffserv model
- Marker – Mark a packet's IP header with forwarding information for service differentiation
- Simple packet dropper – Drops packets based on service differentiation

Solaris™ OS IPQoS Implementation

The Solaris OS implementation has added:

- Flow accounting module that gathers statistics for traffic flows, through the `flowaact` command
- Statistics gathering for traffic classes, through the `kstat` command
- 802.1D user priority markings for virtual local area networks (VLANs)

Note – The Uniform Resource Locator (URL) and File Transfer Protocol (FTP) classifiers are in the IPQoS Functional Specification, v1.1 June 25, 2001, but have not been added.

The IPQoS feature allows for the prioritization, control, and gathering of accounting statistics on network traffic flows. Different levels of network service for users on a network, or priorities for selected applications, can be implemented based on the IPQoS configuration. The implementation supports IPv4 and IPv6 addressing, and allows for interoperability with IPsec (IPsec encrypted flows).

The QoS Policy

A QoS policy defines user or application priorities, and actions for handling different categories of traffic.

The QoS model allows for the classification of network traffic flows into CoSs. Each CoS has a defined mark and characteristic. Two defined CoS behaviors have been standardized:

- EF PHB [RFC 2598] is used to build a low loss, low latency, low jitter, assured bandwidth, end-to-end service through DS domains. This type of service simulates a virtual leased line across a set of routers.
- AF PHB [RFC 2597] is a means for a provider DS domain to offer different levels of forwarding assurances for IP packets received from a customer DS domain. The AF PHB provides delivery of IP packets in four independently forwarded AF classes. Within each AF class, an IP packet can be assigned one of three different levels of drop precedence.

The QoS policy for your network resides in a configuration file, `/etc/inet/ipqosinit.conf`. The file is created manually with a text editor, and the file name is provided as an argument to the `ipqosconf` configuration utility. The policy is written into the kernel IPQoS system when the policy is applied.

The configuration file consists of a tree of action statements. Each action statement contains a set of classes, filters, or parameters to be processed by the IPQoS module that is called in the action statement.

IPQoS Module

The IPQoS module definition indicates which module is to process the parameters in the QoS action statement. Table 8-2 is a list of the IPQoS modules:

Table 8-2 IPQoS Modules

Module Name	**Definition**
`ipgpc`	IP Generic Packet Classifier arranges traffic flows into classes that are based on characteristics from the IPQoS configuration file.
`dscpmk`	Differentiated Services Code Point Marker. Marker to be used to create DS codepoints in IP packet headers.
`dlcosmk`	Data Layer Class of Service Marker. Marker to be used with VLAN tag of an Ethernet frame header with a 3-bit user priority, indicating the CoS defining the PHB to be applied to the packet.
`tokenmt`	Meter that uses a single-rate or two-rate transmission metering scheme that can be made color aware.
`tswtclmt`	Time Sliding Window Three Conformance Level Meter. Meter that uses the time-sliding window transmission rate metering scheme.
`flowacct`	Flow-accounting module records information about traffic flows. To enable flow account, the Solaris OS `exacct` account facility and `acctadm` command also need to be used.

Note – The `dlcosmk` and `flowacct` modules are IPQoS additions and are not part of the `diffserv` model.

Figure 8-1 shows a common traffic flow sequence on an IPQoS-enabled machine:

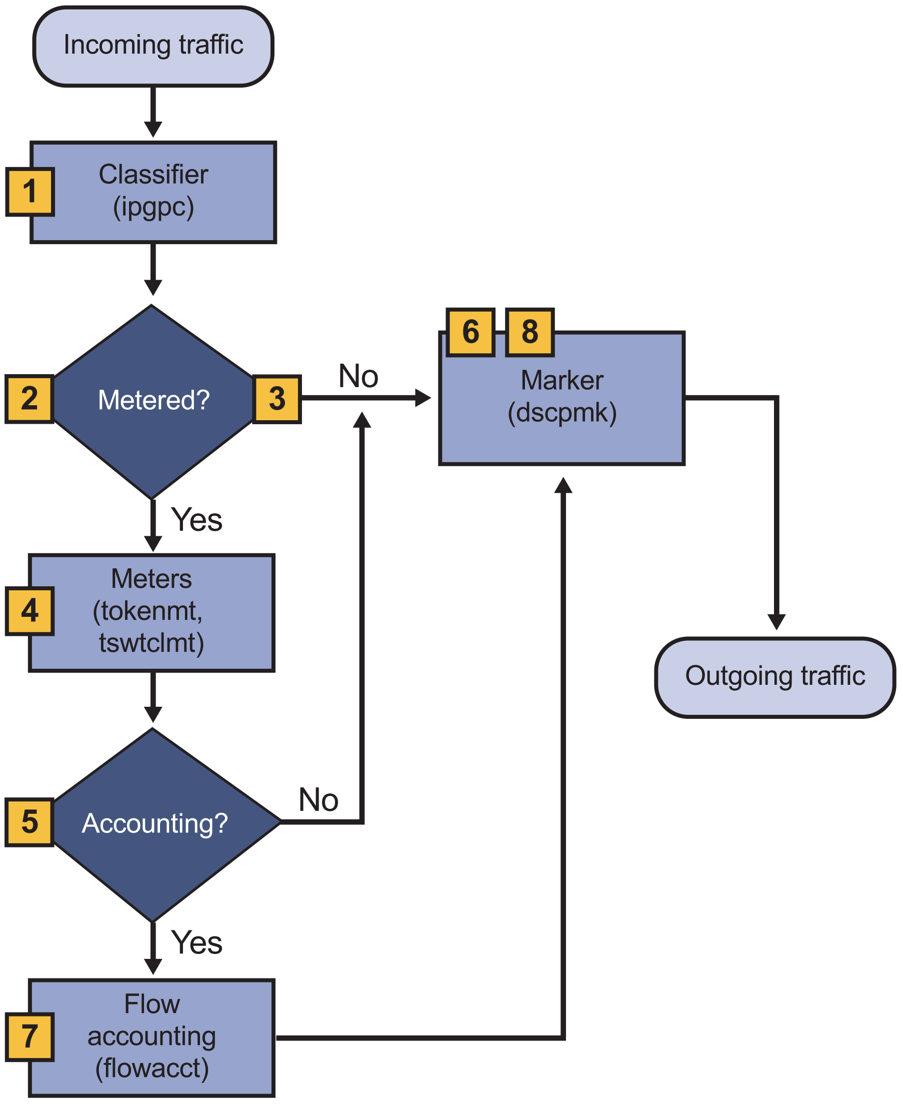


Figure 8-1 Traffic Flow Through the IPQoS Implementation of the `diffserv` Model

1. The classifier selects from the packet stream all packets that match the filtering criteria in the system's QoS policy.
2. The selected packets are then evaluated for the next action to be taken on them.
3. The classifier sends to the marker any traffic that does not require flow control. Traffic to be flow-controlled is sent to the meter.
4. The meter enforces the configured rate and assigns a traffic conformance value to the flow-controlled packets.

5. The flow-controlled packets are evaluated to determine if any packets require accounting.
6. The meter sends to the marker any traffic that does not require flow accounting.
7. The flow-accounting module gathers statistics on received packets, and sends the packets to the marker.
8. The marker assigns a DS codepoint to the packet header. This DSCP indicates the per-hop behavior that a `diffserv` model-aware system must apply to the packet.

Support for 802.1D User Priority Mapping

Support has been made available for Sun™ GigaSwift Ethernet adapters that includes:

- An accounting mechanism for flows
- Use of the CoS interface flag marking for user priority mapping. A VLAN device adds a 4-byte 802.1Q tag to outbound Ethernet frames. Part of this tag is a 3-bit User Priority field defined by 802.1D that is used to indicate the CoS at the Link Layer.

The CoS flag is automatically set during initialization if the interface supports CoS marking. The 802.1D tagging is available with the Cassini card and drivers, `ce`, on the SPARC® platform only. See the diagram in Figure 8-2.

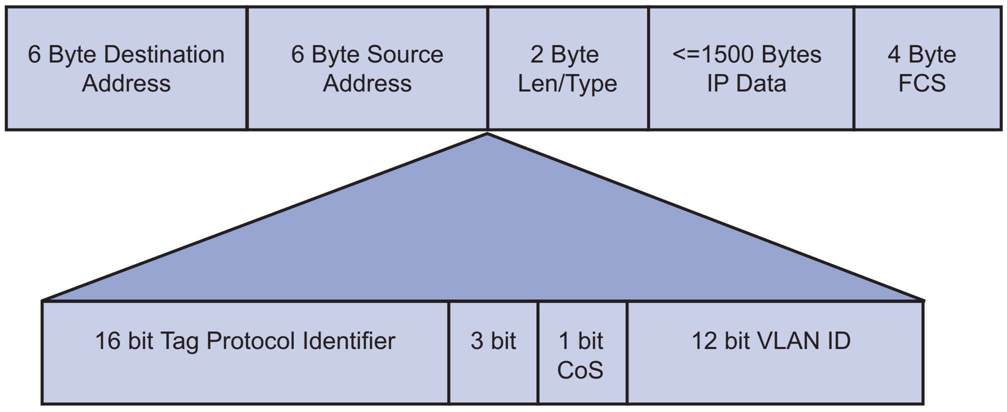


Figure 8-2 802.1Q frame with CoS

Configuring the IPQoS Facility

The `ipqosconf` utility is used to configure, commit, and flush the IPQoS facility.

The QoS policies are defined in an IPQoS configuration file, `/etc/inet/ipqosinit.conf`. Each policy is composed of a set of filters defining a group of network flows, and a sequence of actions to apply to each packet in that group.

Note – The configuration file is not required to be located in the `/etc/inet` directory, or to be named `ipqosinit.conf`, or end with `.qos`. But, if the configuration is committed to be valid across reboot, it must reside in `/etc/inet/ipqosinit.conf`.

By default the configuration is not preserved across reboot, so the `/etc/ipqosinit.conf` file can be committed using the `-c` option of the `ipqosconf` utility. This option makes the current applied configuration persistent across reboots.

The IPQoS feature is turned off by flushing the configuration using the `-f` option of the `ipqosconf` utility. If the feature was made persistent across reboot using the `-c` option, the `/etc/inet/ipqosinit.conf` file needs to be renamed or removed to prevent the feature from being re-enabled upon reboot.

Configuring the IPQoS Facility

As previously mentioned, the QoS policies are defined in the `/etc/inet/ipqosinit.conf` file. This file is not intuitive to configure and requires a general working knowledge of IPQoS. This example will walk you through a generic configuration to better aquaint you with an IPQoS configuration.

Let us assume that a publishing company wishes to reduce the priority of data on the companies network due to surfing traffic, and increase the priority of the data going to the print servers.

To accomplish this the following `/etc/inet/ipqosinit.conf` file could be used.

The first line of the file will always be a version line. This is mandatory.

```
fmt_version 1.0
```

In the Diffserv model, the classifier is responsible for organizing selected traffic flows into groups on which to apply different service levels. The QoS policy defines various criteria that must be present in packet headers. These criteria are called *selectors*. The `ipgpc` classifier compares these selectors against the headers of packets that are received by the IPQoS system. The `ipgpc` module then selects all matching packets.

The first action will be to call the `ipgpc` module, then create a class and filter for the data that will be prioritized. We will filter on port 515 (port used to talk to print servers) and port 80 (typical port used by web traffic). Any data the filter detects on these ports will be added to the appropriate class. The `next_action` directive of the class will then be acted on.

```
action {
        module ipgpc
        name ipgpc.classify

        class {
                name printing
                next_action markEF
        }
        class {
                name surfing
                next_action markAF13
        }

        filter {
                name printing_out
                dport 515
                direction LOCAL_OUT
                class printing
        }

        filter {
                name surfing_out
                dport 80
                direction LOCAL_OUT
                class surfing
        }
}
```

The next action in this example will use the dscpmk (Differentiated Services Code Point Marker) module. This module will mark the IP packet headers coming from the printing class above, with an AF (Assured forwarding) marker. This marker provides four different classes of forwarding behaviors that you can specify as well as a Drop Precedence (low, medium, high). A decimal 46 is known as an EF (Expedited Forwarding) codepoint that guarantees preferential treatment by all Diffserv routers en route to the packets' destination.

The dscp_map parameter is a 64-element array, which you populate with the (DSCP) value. dscp_map is used to map incoming DSCPs to outgoing DSCPs that are applied by the dscpmk marker.

The next_action continue indicates that we are done with the packet.

```
action {
        module dscpmk
        name markEF
        params {
                  dscp_map {0-63:46}
                next_action continue
        }
}
```

The last action listed is for all packets that belong to the class surfing. This action will mark the packet headers with an AF13 codepoint. An AF13 is a class 1 High-Drop Precedence mark. It will be listed in the action as a decimal 14 in the 64-element array. If there is any packet contention, this packet will lose and be dropped.

```
action {
        module dscpmk
        name markAF13
        params {
                dscp_map {0-63:14}
                next_action continue
        }
}
```

Now that the /etc/inet/ipqosinit.conf file is created, start the IPQoS service:

```
# ipqosconf -v -a /etc/inet/ipqosinit.conf
```

To get statistics on the IPQoS operation you can use use kstat -m command on the appropriate module.

```
# kstat -m ipgpc
# kstat -m tokenmt
# kstat -m dscpmk
```

To turn off IPQoS use the command;

```
# ipqosconf -f
```

The following variable can be set using the `ndd`(1m) command:

- `ip_policy_mask` – Enables or disables IPQoS processing in any of the following callout positions: forward outbound, forward inbound, local outbound, and local inbound. This parameter is a bitmask as follows:

Not Used	Not Used	Not Used	Not Used	Forward Outbound	Forward Inbound	Local Outbound	Local Inbound
X	X	X	X	0	0	0	0

A 1 in any of the position masks or disables IPQoS processing in that particular callout position. For example, a value of `0x01` disables IPQoS processing for all the local inbound packets. The default value is 0, meaning that IPQoS processing is enabled in all the callout positions.

0 (`0x00`) to 15 (`0x0F`) – A value of 15 indicates that IPQoS processing is disabled in all the callout positions.

When to Change – Change this parameter if you want to enable or disable IPQoS processing in any of the callout positions.

Table 8-3 shows the files relevant to this feature:

Table 8-3 IPQoS Files

Files and Manual Pages	**Description**
`ipqosconf`(1M)	Used to configure the IPQoS facility
`/etc/inet/ipqosinit.conf`	Configuration file
`dlcosmk`(7IPP)	Data layer CoS marker
`dscpmk`(7IPP)	DSCP marker
`flowacct`(7IPP)	Flow accounting module

Table 8-3 IPQoS Files (Continued)

Files and Manual Pages	**Description**
`ipgpc`(7IPP)	IP generic packet classifier
`ipqos`(7IPP)	IPQoS
`tokenmt`(7IPP)	Single and two rate three conformance level meter
`tswtclmt`(7IPP)	Time sliding window three conformance level meter
`/etc/init/ipqosconf.1.sample`	Sample configuration file for an application server
`/etc/init/ipqosconf.2.sample`	Sample configuration file that meters the traffic for a specified application
`/etc/init/ipqosconf.3.sample`	Sample configuration file that marks the Ethernet headers of web traffic with a given user priority

Installation, Upgrading, and Migration

The IPQoS feature is *off* by default after a standard installation. The feature is turned *on* by creating a QoS policy file and applying the configuration (causing the kernel to reflect the configuration) using the `-a` ***`file_name`*** option of the `ipqosconf` utility.

The FireEngine Project

The FireEngine project is a redesign of the Solaris OS networking stack to provide better performance and scalability while enabling a path for delivering advanced functionality. FireEngine uses a networking framework based on an IP classifier and vertical perimeters.

FireEngine Structure

The connection lookup for inbound packets is done using an IP classifier, based on the classification, the connection structure is identified. The classifier also becomes the database for storing a sequence of function calls necessary for all inbound and outbound packets. This allows the Solaris OS networking stack to change from the current message passing interface to a Berkeley Software Distribution (BSD) style function call interface.

FireEngine uses a per central processing unit (CPU) synchronization mechanism called vertical perimeter inside a merged TCP/IP module. The vertical perimeter is implemented using a serialization queue abstraction called squeue. This vertical perimeter allows only one thread to access a connection and guarantees much better data locality and only one thread actively processes a packet on a CPU, thus avoiding context switches and locks to protect connection states. A connection is tied to a vertical perimeter instance and packets for that connection are processed on that vertical perimeter.

Advantage of Merged Structures

The merged TCP/IP module is multithreaded (MT) module. This design allows shortened code paths and improves the interaction performance between TCP and IP and maintains the binding of a connection to a CPU.

The merge of TCP/IP into a single module disallows any STREAMs module to be inserted between the two. Previously, it was possible to insert a module between TCP and IP though it wasn't done as the interface and messages exchanged between TCP and IP were not public.

When the Network Interface Controller (NIC) receives a packet it interrupts the CPU as soon as possible. Then when IP receives a packet from the NIC, it classifies the packet and determines the connection structure and the instance of `squeue` the packet needs to be processed on. New incoming connections are assigned to the `squeue` instance attached to the interrupted CPU or the attachment to a CPU is based on a fan-out, across all CPUs, in a round robin balance scheme to avoid getting CPU limited. To optimize distribution between interrupt and non-interrupt processing, the NIC always send the packet to IP in a interrupt context and IP determines if fan-out or single CPU model should be used.

Vertical perimeters, or `squeues`, guarantee that only a single thread can process a given connection at any given time, thus serializing access to the TCP connection structure by multiple threads, both for read and write, in the merged TCP/IP modules.

Previously, TCP and IP were separate STREAMS modules. They communicated with each other using STREAMS message-passing mechanisms. This presented several problems:

- Extra overhead from `putnext()` call, moving packets between modules
- Potentially long call stacks
- Loss of context between modules
- Information gathered from incoming mblocks in one module must be gathered again in the next module
- Poor code and data locality

Because TCP and IP embed a lot of knowledge about each other, the networking stack is made more efficient and simpler by merging them so they can directly call each other. This eliminates any STREAMS related processing overhead and reduces the amount of CPU cycles spent on each incoming and outgoing network packet.

Most of the socket-related system calls show performance improvements using the merged TCP/IP module. The gains come from reducing the cost of plumbing a STREAM and the introduction of direct calls between TCP and IP.

Previously the IP module would go to exclusive mode when plumbing interfaces, during some multicast operations and during operations that required holding a global IP sequence queue lock. It could also take IP awhile to return to MT mode after returning from exclusive mode.

Executing any of the exclusive mode IP operations on a server with multiple CPUs and NICs when the network traffic was high and serving lots of connections would slow down the whole system, not just networking, and it would often take the system a long time to recover to normal MT behavior.

In the past, it was assumed that exclusive mode IP operations were highly infrequent but this has become invalid as severs now may have to plumb and unplumb thousands of tunnels or VLANs in a short amount of time.

Making IP and TCP fully multithreaded greatly reduces the amount of time required for plumbing interfaces, by using multiple CPUs simultaneously and removes the need of IP going to exclusive mode.

Due to the complexity of the changes required, the FireEngine project is divided into phases. This enables Solaris OS to incrementally deliver performance gains while minimizing the risk. Figure 8-3 shows the incremental enhancements made to the Solaris networking stack in the Solaris 10 release.

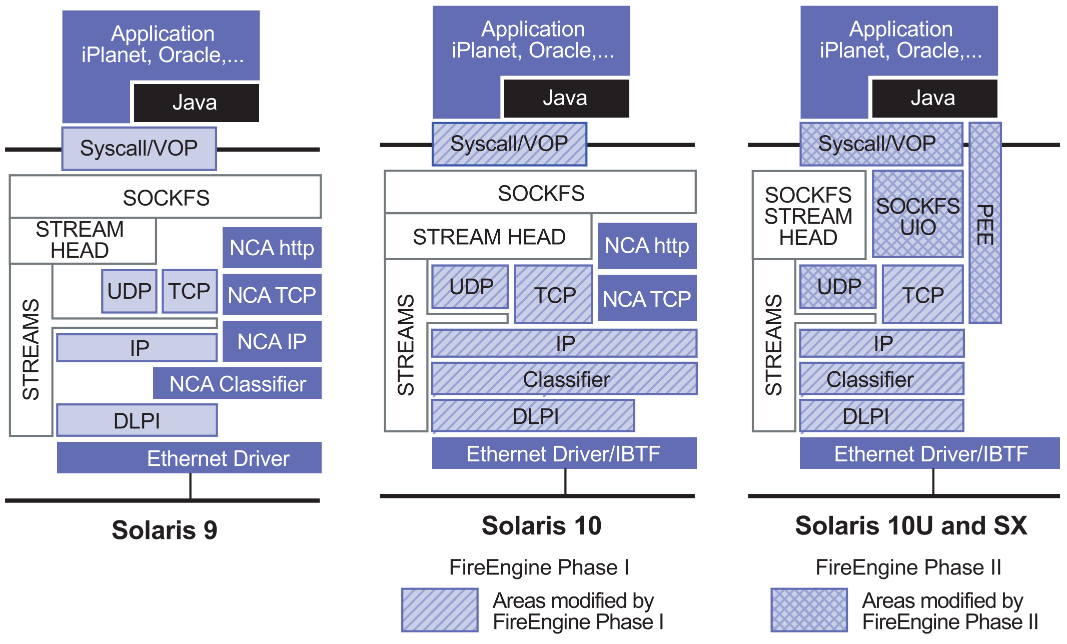


Figure 8-3   Comparision of the Solaris 9 and 10 Networking Structure

Tunable Parameters

No special configuration is required for FireEngine, however, the following tunable parameters have been added by the project.

- `ip_squeue_fan-out`

  Description – This parameter determines the mode of associating TCP/IP connections with squeues.

  A value of 0 associates a new TCP/IP connection with the CPU that creates the connection. A value of 1 associates the connection with a random CPU, effectively distributing the load across all CPUs and all squeues in the system. The default is 0.

  When to Change – You might consider setting this parameter to 1 to spread the load across all CPUs in certain situations. For example, when the number of CPUs exceed the number of NICs, and 1 CPU cannot handle the network load of a single NIC.

- `ip_squeue_bind`

  Description – Controls whether work threads are bound to a specific CPU. The default is 1.

- `ip_squeue_profile`

  Description – If set to 1, squeue profiling is enabled. The default is 1.

Using the the `kstat` Command for Statistics

Users have access to a set of Management Information Base (MIB) II statistics by using the `kstat`(1M) command.

```
# kstat tcp
module: tcp                             instance: 0
name:   tcp                          class:    mib2
        activeOpens                     17
        attemptFails                    8
...
# kstat ip
module: ip                              instance: 0
name:   ip                           class:    mib2
        addrEntrySize                   96
        crtime                          4.613633676
...
```

TCP Multi-Data Enhancement to the Solaris OS

Multi-Data Transmission (MDT) is a concept derived from a software driver's ability to simultaneously transmit multiple TCP segments or data packets. MDT provides a scheme for the software driver to transmit data using IP in a buffer of multiple packets, instead of a single packet at a time. A major benefit of MDT is that cumulative data transfer rates between the software driver and IP are significantly increased.

MDT enables the network stack to send more than one packet at a time to the network device driver during transmission. Enabling this feature reduces the per-packet processing costs by improving the host CPU utilization or network throughput.

Performance Benefit

MDT reduces the per-packet processing costs of network transmission, resulting in improved host CPU utilization and network throughput. Environments that use applications which perform bulk-data transfers should notice an increase in system performance.

MDT has been optimized and designed for bulk-data network traffic. This optimization applies to applications such as FTP and protocols such as network file server (NFS) bulk-data transfers. Post Office Protocol 3/Internet Access Message Protocol 4 (POP3/IMAP4) users can also benefit when downloading large messages or attachments.

In tests utilizing the `Netperf` utility, performance results are quite favorable, as the following shows:

```
E450 -to- Excalibur:
Without MDT (1 session) 450Mbps
With MDT (1 session) 650Mbps

E450 -to- E450:
Without MDT (1 session) 385Mbps
With MDT (1 session) 650Mbps
```

Configuring the MDT Feature

This section covers when to use MDT and how to enable the feature.

When to use MDT

You can use MDT when seeking an increase in system performance from applications that generate large volumes of TCP network traffic.

These applications include clusters requiring the following:

- High-speed backup capability
- Desktop users who need better throughput than 100 megabytes per second
- Workstations with computer-aided design/computer-aided manufacturing (CAD/CAM) applications
- Other high-performance computer simulation applications, financial services, and data warehouses

When MDT is activated on a system, the networking stack still makes the final decision as to whether MDT is used on a particular TCP connection.

This decision is based upon several factors:

- The size of available TCP data to transmit
- Availability of MDT support in the software driver (this is detected during negotiation)
- The presence of IPQoS policy (MDT cannot be used when IPQoS is enabled)
- The presence of IP Security Protocol (IPsec) global policy (MDT may still be used if per-connection policy is not present for the TCP connection)

How MDT Is Enabled

Before enabling the MDT feature, you must enable IP-driver new-style capability negotiation by placing the following line in the /etc/system file:

```
set ip:ip_use_dl_cap=1
```

Note – The DL_CAPABILITY structure negotiation is turned off by default in the Solaris 9 OS.To enable the MDT feature, use the ndd command, as follows:

```
# ndd -set /dev/ip ip_multidata_outbound 1
```

Note – MDT is turned off by default in Solaris 9 OS and enabled by default in Solaris 10 OS.

The ip_multidata_outbound parameter enables the network stack to send more than one packet at one time to the network device driver during transmission. Enabling this parameter reduces the per-packet processing costs by improving the host CPU utilization and/or network throughput.

The MDT feature is only effective for device drivers that support multi-data transmissions. By default, it is enabled: 0 = disabled, 1 = enabled.

If you do not want this parameter enabled for debugging purposes or for any other reasons, disable it.

How MDT Is Disabled

To disable the MDT feature, use the ndd command, as follows:

```
# ndd -set /dev/ip ip_multidata_outbound 0
```

Troubleshooting Tips

Troubleshooting tools such as `snoop` are not effective with MDT, because packets on the network still behave as regular IP packets. MDT only changes the messages passed within the stack, making it transparent outside the system.

Below are example of commands that show if you have MDT enabled and that also provide MDT status information.

The `ifconfig` command shows the status of the Cassini interface:

```
# ifconfig -a
lo0: flags=1000849<UP,LOOPBACK,RUNNING,MULTICAST,IPv4> mtu 8232 index 1
        inet 127.0.0.1 netmask ff000000
ce0: flags=1000843<UP,BROADCAST,RUNNING,MULTICAST,IPv4> mtu 1500 index 2
        inet 192.168.30.56 netmask ffffff00 broadcast 192.168.30.255
        ether 0:3:ba:23:f:27
#
```

Use the `ndd` command to determine if MDT is enabled:

```
# ndd -get /dev/ip ip_multidata_outbound
1
```

Use the `kstat` command to view statistics about the Cassini interface and MDT:

```
# kstat ce:0
module: ce                              instance: 0
name:   ce0                          class:    net
        alignment_err                   0
<output omitted>
        multircv                        0
        multixmt                        5
```

In Solaris 9 OS the `netstat -k` command provides a large amount of output. You can redirect the output of this command to a file, then edit the file to search for the Cassini interface and MDT:

```
# netstat -k > file
# vi file
search for ce0:

ce0:
ipackets 2015 ipackets64 2015 ierrors 0 opackets 9169 opackets64 9169
```

```
oerrors 0 collisions 0 rbytes 122030 rbytes64 122030 obytes 11279014
obytes64 11279014
multircv 0 multixmt 5 brdcstrcv 0 brdcstxmt 11 norcvbuf 0
noxmtbuf 0 first_collision 0 excessive_collisions 0 late_collisions 0
peak_attempts 0 length_err 0 alignment_err 0 crc_err 0 code_violations 0
ifspeed 1000000000 promisc off rev_id 17 xcvr_inits 1 xcvr_inuse 1
xcvr_addr 1 xcvr_id 2121811 cap_autoneg 1 cap_1000fdx 1 cap_1000hdx 1
<output omitted>

search for multidata:

multidata:
buf_size 120 align 64 chunk_size 128 slab_size 8192 alloc 0 alloc_fail 0
free 0 depot_alloc 0 depot_free 0 depot_contention 0 slab_alloc 0
slab_free 0 buf_constructed 0 buf_avail 0 buf_inuse 0 buf_total 0
buf_max 0 slab_create 0 slab_destroy 0 vmem_source 18 hash_size 0
hash_lookup_depth 0 hash_rescale 0 full_magazines 0 empty_magazines 0
magazine_size 7

multidata_pdslab:
buf_size 1472 align 64 chunk_size 1472 slab_size 16384 alloc 0 alloc_fail
0
free 0 depot_alloc 0 depot_free 0 depot_contention 0 slab_alloc 0
slab_free 0 buf_constructed 0 buf_avail 0 buf_inuse 0 buf_total 0
buf_max 0 slab_create 0 slab_destroy 0 vmem_source 18 hash_size 64
hash_lookup_depth 0 hash_rescale 0 full_magazines 0 empty_magazines 0
magazine_size 3

multidata_pattbl:
buf_size 32 align 64 chunk_size 64 slab_size 8192 alloc 0 alloc_fail 0
free 0 depot_alloc 0 depot_free 0 depot_contention 0 slab_alloc 0
slab_free 0 buf_constructed 0 buf_avail 0 buf_inuse 0 buf_total 0
buf_max 0 slab_create 0 slab_destroy 0 vmem_source 18 hash_size 0
hash_lookup_depth 0 hash_rescale 0 full_magazines 0 empty_magazines 0
magazine_size 15
```

In Solaris 10 OS the following `kstat`(1M) commands will return information similar to the previous commands listed for Solaris 9 OS. The following command is the equivalent of the `netstat -k` command which is no longer supported:

```
# kstat -p -s '*' > file
```

The following commands will report interface and IP Multidata transmission information from the kernel:

```
# kstat -p ce:0:ce0:*
# kstat -p unix:0:multidata:*
# kstat -p unix:0:multidata_pdslab:*
# kstat -p unix:0:multidata_pattbl:*
```

Other Changes to IP

The topics of this section include:

- The `routeadm`(1M) command
- Internet Protocol over InfiniBand (IPoIB)
- Disabling Non-decimal IP aliases on logical interfaces.
- IP Do not Fragment (DF) flag with application supplied header data.

The `routeadm`(1M) command is used to administer system-wide configuration for IP forwarding and routing. The command is used to enable or disable each function independently, overriding any system default setting for each function.

IP Routing Configuration

This section provides background information on IP forwarding and routing in the Solaris OS environment.

Configuring IPv4

A number of factors affect how IPv4 forwarding and routing is configured at boot time, and those have evolved since Solaris 2.5 OS. In Solaris 2.5 and 2.5.1 OS, having multiple IPv4 interfaces configured at boot time would result in IPv4 forwarding and routing being enabled. Solaris 2.6 OS introduced support for Dynamic Host Configuration Protocol (DHCP) and would disable IP forwarding if the `/etc/defaultrouter` file existed and DHCP was not enabled. This behavior has evolved since then, and IPv4 forwarding is now disabled if `/etc/defaultrouter` exists in all cases.

This is a problem for administrators who want to configure a default router on a system that is also designated as a router. Such administrators must be able to explicitly set the IPv4 forwarding and routing configuration of the system regardless of the presence of `/etc/defaultrouter`.

Configuring IPv6

The presence of the `/etc/inet/ndpd.conf` file currently determines the boot script behavior. If the file exists, IPv6 forwarding is enabled, and the `in.ripngd`(1M) routing daemon is started. Administrators that use the `ndpd.conf`(4) configuration file do not necessarily want IPv6 forwarding and routing automatically enabled. They need another mechanism to configure IPv6 forwarding and routing.

A new utility `routeadm`(1M) is introduced to allow the administrator to configure the system's IP forwarding and routing configuration. The documented usage of this command is as follows:

Using the `routeadm` Command

The first usage, `routeadm` with no arguments, reports the current configuration.

The second usage is:

```
routeadm [ -e {ipv4-forwarding | ipv4-routing | ipv6-forwarding | ipv6-
routing} ] \
        [ -d {ipv4-forwarding | ipv4-routing | ipv6-forwarding | ipv6-
routing} ] \
        [ -r {ipv4-forwarding | ipv4-routing | ipv6-forwarding | ipv6-
routing} ]
```

This usage allows for changes to the forwarding and routing configurations for IPv4 and IPv6. Changes made with this usage are persistent, but are not applied to the running system unless `routeadm`(1M) is called later with the `-u` option.

The `-e` option enables the specified configuration. The `-d` option disables the specified configuration. The `-r` option reverts the specified configuration to the system default, which means that the configuration decision for that setting is left to the system boot scripts' existing default logic. Multiple `-e`, `-d`, and `-r` options can be specified on the command line.

The third usage, `routeadm -u`, applies the current configuration to the running system. In other words, it enables or disables IP forwarding, and launches or kills routing daemons. It does not alter the state of the system for those settings that have been set to default. This option is meant for administrators to use if they do not wish to reboot to apply their changes.

The fact that the `routeadm`(1M) command does not apply the changes immediately to the running system is contrary to precedent set by utilities such as `coreadm`(1M). This is intentional since changes to the routing configuration of a system can be disruptive. Thus deliberate action is required from the administrator to apply the settings.

The `routeadm`(1M) command also implements a project-private `-b` option that, when used in conjunction with the -u option, allows the `inetinit` boot script to specify those configurations that have been set to default. This overrides the `-u` option's regular behavior of not altering system state for the settings that have been set to default. When specifying the `-b` option, `-e`, and `-d` options specify those default settings.

If the `routeadm` command has not been used to alter any settings, after a fresh install for example, all variables are set to "default", which implies that their semantics are left up to the default behavior of the boot scripts. The intent is that the initial introduction of this administrative interface won't change the default behavior of the system, which is described in the `routeadm`(1M) command.

The `routeadm`(1M) command uses the project private file `/etc/inet/routing.conf` to keep the state of these variables. Administrators should not directly edit this file, but instead use the `routeadm`(1M) command.

The `routeadm(1M)` Command Settings

The following is a description of each setting that the `routeadm`(1M) command administers.

- `ipv4-forwarding` – This controls the global forwarding policy for all IPv4 interfaces. If enabled, IP will forward IPv4 packets to and from interfaces when appropriate. If disabled, IP will not forward IPv4 packets to and from interfaces when appropriate. If not set or reverted, the system boot scripts' current default logic determines whether or not to configure IP to forward IPv4 packets.
- `ipv4-routing` – This setting determines whether or not an IPv4 routing daemon is run. The routing daemon for IPv4 is `/usr/sbin/in.routed`. If not set or reverted, the system boot scripts' current default logic determines whether or not to run `in.routed`.
- `ipv6-forwarding` – This controls the global forwarding policy for all IPv6 interfaces. If enabled, IP will forward IPv6 packets to and from interfaces when appropriate. If disabled, IP will not forward

IPv6 packets to and from interfaces when appropriate. If not set or reverted, the system boot scripts' current default logic determines whether or not to configure IP to forward IPv6 packets.

- `ipv6-routing` – This setting determines whether or not an IPv6 routing daemon is run. The routing daemon for IPv6 is `/usr/lib/inet/in.ripngd`. If not set or reverted, the system boot scripts' current default logic determines whether or not to run `in.ripngd`.

The forwarding settings are global settings. Each interface also has an `IFF_ROUTER` forwarding flag that determines whether packets can be forwarded to or from that particular interface, and that flag can be independently controlled by an `ifconfig`(1M) command router option. When the global forwarding setting is changed (`-u` is issued where the value is changed from enabled to disabled or vice-versa), all interface flags in the system are changed to reflect the new global policy simultaneously.

When a new interface is plumbed by the `ifconfig`(1M) command, the value of the interface-specific forwarding flag is set according to the current global forwarding value. Thus the forwarding value forms the default for all new interfaces.

Enabled settings are labeled "enabled", disabled settings are labeled as "disabled", and default settings are labeled as "default". Default settings are also followed by the system's current setting printed in parentheses. What is printed in parentheses is not the policy that the default logic in the boot script would choose, but the settings as examined on the running system.

The following is some sample output.

```
# routeadm
              Configuration   Current              Current
                     Option   Configuration        System State
        ---------------------------------------------------------
            IPv4 forwarding   default (disabled)   disabled
               IPv4 routing   default (disabled)   disabled
            IPv6 forwarding   default (disabled)   disabled
               IPv6 routing   default (disabled)   disabled
# routeadm -e ipv4-forwarding -e ipv4-routing
# routeadm
              Configuration   Current              Current
                     Option   Configuration        System State
        ---------------------------------------------------------
```

```
              IPv4 forwarding   enabled              disabled
                 IPv4 routing   enabled              disabled
              IPv6 forwarding   default (disabled)   disabled
                 IPv6 routing   default (disabled)   disabled
# routeadm -d ipv6-forwarding -d ipv6-routing
# routeadm
              Configuration   Current              Current
                       Option   Configuration        System State
        ---------------------------------------------------------
              IPv4 forwarding   enabled              disabled
                 IPv4 routing   enabled              disabled
              IPv6 forwarding   disabled             disabled
                 IPv6 routing   disabled             disabled
# routeadm -r ipv4-forwarding
# routeadm
                Configuration   Current              Current
                       Option   Configuration        System State
        ---------------------------------------------------------
              IPv4 forwarding   default (disabled)   disabled
                 IPv4 routing   enabled              disabled
              IPv6 forwarding   disabled             disabled
                 IPv6 routing   disabled             disabled
# routeadm -u
# routeadm
                Configuration   Current              Current
                       Option   Configuration        System State
        ---------------------------------------------------------
              IPv4 forwarding   default (disabled)   disabled
                 IPv4 routing   enabled              enabled
              IPv6 forwarding   disabled             disabled
                 IPv6 routing   disabled             disabled
```

IP Forwarding and Routing with DHCP

The DHCP protocol specification (RFC 2131) states that DHCP is not intended for use in configuring routers. For Solaris OS and before the addition of the `routeadm` command, IP forwarding and routing were disabled entirely on Solaris OS if any interface was configured by DHCP to comply with this RFC. This was a policy decision that was heavy handed, not considering that DHCP could configure some interfaces on a system, and others not.

To accommodate systems that may be forwarding IP packets between interfaces not configured through DHCP, the kernel does not enable the per-interface forwarding flag (`IFF_ROUTER`) on DHCP-configured interfaces when enabling the global IP forwarding setting. Other interfaces are subject to the policy settings described previously in this document.

IPoIB

IPoIB is a InfiniBand (IB) Data Link Provider Interface (DLPI) driver treating IB as a new data link (layer 2) type on which TCP/IP traffic can be transmitted. The wire protocol is specified by IETF standards for IP over IB.

IB is an emerging technology for interconnecting hosts and input/output (I/O) devices in a switched fabric. It is targeted as the interconnect technology for the future generation of computer room interconnect.

IB is a switched fabric interconnect designed for two functions:

1. Attach peripheral devices, such as network interface controllers (NICs) and storage controllers to a host.
2. Provide host to host communications.

Why IPoIB

The applications programs that directly use network communication almost exclusively use the sockets application programming interface (API) and TCP/IP protocols. A minority use UDP and raw IP. This will not change during the time period Sun adopts IB. Except for a very few applications for which the highest possible performance is critical, applications code will not change to take advantage of IB features.

IPoIB enables Solaris TCP/IP protocol suites to run on top of IB host channel adaptors (HCAs) with minimal changes The emphasis is on compatibility with the existing TCP, UDP, IPv4, and IPv6 implementation, at both the socket interface, the stack, and the wire protocol layer. IPoIB will also enable IB hosts to communicate with external IP networks through IB-to-IP gateways (layer 3).

IPoIB is implemented with the InfiniBand Driver (ibd(7D)), a driver that attaches to the bottom of the Solaris OS IP stack through the Generic Local Area Network (LAN) Driver (gld(7D)) that provides the DLPI network driver interface. This provides IPoIB encapsulation and obtains IB transport services through the IB Transport Framework (IBTF). This IPoIB DLIP driver will enable the same IP networking over IB that is currently delivered over Ethernet.

IPoIB MAC Address

IPoIB uses 20-byte media access control (MAC) addresses: 16 bytes are derived from the IB Global Identifier (GID) and four bytes are the Queue Pair Number (QPN) used in IB. The arp(1M) command has been changed to support variable length MAC addresses. Also, the ifconfig(1M), snoop(1M), and netstat(1M) commands have been changed to display the properly formatted 20-byte IB MAC address.

Other Utilities and IPoIB

ifconfig(1M) – The descriptive media name will be ipib (similar to ether for Ethernet) when the ifconfig command is used to display information about the media. The 20 bytes of link address appears in the format shown in this example:

```
# ifconfig ibd0
ibd0: flags=1000842<BROADCAST,RUNNING,MULTICAST,IPv4> mtu 2040 index 6
        inet  192.168.100.101 netmask ffffff00 broadcast 192.168.100.255
        ipib 0:1:80:17:0:0:0:0:0:0:0:4:0:2:c9:0:1:1c:95:51
```

IPoIB does not support the Reverse Address Resolution Protocol (RARP). The QPN can change from boot to boot and RARP requires a stable MAC address. Since RARP is not supported on IPoIB, the in.rarpd(7D) daemon will fail to be started on any IPoIB interface and the revarp operation of the ifconfig command will fail without transmitting any RARP packets or updating the interface's IP address.

```
# ifconfig ibd0 auto-revarp
# ifconfig ibd0
ibd0: flags=1000842<BROADCAST,RUNNING,MULTICAST,IPv4> mtu 2040 index 6
        inet 0.0.0.0 netmask ffffff00 broadcast 192.168.100.255
        ipib 0:1:80:17:0:0:0:0:0:0:0:4:0:2:c9:0:1:1c:95:51
```

`arp`(1M) and `netstat`(1M) – Other than the 20 bytes of link address display format, there is no change in how these commands work In the following example, note that the `arp -a` command invokes `netstat -p` for output:

```
# arp -a
Net to Media Table: IPv4
Device   IP Address               Mask      Flags   Phys Addr
------ -------------------- --------------- ----- ---------------
hme0   sun05-sinslab-gate   255.255.255.255       00:05:dd:c5:d7:00
ibd0   redder-ibd           255.255.255.255
00:01:80:17:00:00:00:00:00:00:04:00:02:c9:00:01:00:d0:51
ibd0   pap-ibd              255.255.255.255 SP
00:00:00:14:00:00:00:00:00:00:04:00:02:c9:00:01:1c:97:81
hme0   redder               255.255.255.255       08:00:20:cf:b5:05
hme0   pap                  255.255.255.255 SP    08:00:20:d9:04:97
ibd0   BASE-ADDRESS.MCAST.NET 240.0.0.0       SM
00:ff:ff:ff:ff:12:40:1b:00:00:00:00:00:00:00:00:00:00:00:00
hme0   BASE-ADDRESS.MCAST.NET 240.0.0.0       SM    01:00:5e:00:00:00
```

`snoop`(1M) – The `snoop` command supports packet filtering based on source and destination link addresses present on Ethernet, but unavailable in IPoIB. Thus all filtering expressions based on link level addresses and properties of these addresses (multicast, broadcast) will return the error message: `snoop: filtering option unavailable for IPIB`

The `snoop` command will display the 20-byte MAC address correctly:

```
# snoop -v -d ibd0
Using device /dev/ibd0 (promiscuous mode)
IPIB:  ----- IPIB Header -----
IPIB:
IPIB:  Packet 1 arrived at 16:30:40.44
IPIB:  Packet size = 60 bytes
IPIB:  Ethertype = 0806 (ARP)
IPIB:
ARP:  ----- ARP/RARP Frame -----
ARP:
ARP:  Hardware type = 32
ARP:  Protocol type = 0800 (IP)
ARP:  Length of hardware address = 20 bytes
ARP:  Length of protocol address = 4 bytes
ARP:  Opcode 1 (ARP Request)
ARP:  Sender's hardware address =
0:1:80:17:0:0:0:0:0:0:0:4:0:2:c9:0:1:0:d0:51
ARP:  Sender's protocol address = 192.168.100.101, redder-ibd.eng.sun.com
ARP:  Target hardware address = ?
```

```
ARP:  Target protocol address = 192.168.100.100, pap-ibd.eng.sun.com
ARP:
....
UDP:
DHCP: ----- Dynamic Host Configuration Protocol -----
DHCP:
DHCP: Hardware address type (htype) =  32 (IPIB)
DHCP: Hardware address length (hlen) = 0 octets
DHCP: Relay agent hops = 0
DHCP: Transaction ID = 0xe3981d8f
DHCP: Time since boot = 0 seconds
DHCP: Flags = 0x0000
DHCP: Client address (ciaddr) = 11.0.0.18
DHCP: Your client address (yiaddr) = 0.0.0.0
DHCP: Next server address (siaddr) = 0.0.0.0
DHCP: Relay agent address (giaddr) = 0.0.0.0
DHCP:
DHCP: ----- (Options) field options -----
DHCP:
DHCP: Message type = DHCPREQUEST
...
```

DHCP agent – When the Solaris DHCP agent runs on an IPoIB interface, it will create the file `/etc/dhcp/ibd#.dhc` and use information out of the `/etc/hostname.ibd#` file as mentioned in the `man` (`dhcpagent`(1M), `dhcp`(5)) pages.

The IETF IPoIB draft prescribes rules for DHCP packets that DHCP agents originate, and states that DHCP packets will use `htype` = 32, `hlen` = 0, `chaddr` = 0, and must use the client identifier option. DHCP agent honors these rules.

In the presence of user-specified DHCP client identifiers, no change of behavior is applied with respect to the client's use of the identifier. In the absence of a user-provided client identifier, the DHCP agent automatically determines the default client identifier. Also, the client sets the `BROADCAST` bit in `DISCOVER` and `REQUEST` messages signaling servers and relays to broadcast the response on the link.

DHCP server – The Solaris OS DHCP server expects the `chaddr` field in DHCP packets received from clients to have a valid link address of length `hlen`. When responding to client requests, the server uses these rules:

1. Check for `BROADCAST` bit in the DHCP request packet; if set, use IP-limited broadcast for the response.

2. Else, unicast to next hop if this is NOT a local (subnet) delivery.
3. (IPoIB add this) If the `htype` in the packet indicates IB, use unicast on the client's IP address;
4. Else, update the Address Resolution Protocol (ARP) cache with the client's IP and link address from the packet and, and if that works, use unicast on the client's IP address; if that fails, use an IP-limited broadcast.

Though this change was not essential, it prevents use of broadcast traffic when the server responds back to IPoIB clients in cases where the client does not set the `BROADCAST` bit. It also implies that the server itself will not populate ARP caches corresponding to IPoIB clients, relying on IPoIB ARP for that.

Jumpstart – Jumpstart™ software is a mechanism built on top of `inetboot`, dhcp, and `bootparams` that system administrators use to differentiate between client machines (using Interface link address or DHCP client identifiers) on a boot server to inform the server what software to install on particular machine or groups of machines.

Though there is no behavioral change to Jumpstart due to IPoIB, this is a note that RARP based Jumpstart will not be allowed over an IPoIB interface. Only DHCP-based Jumpstart will be allowed. Also note that OpenBoot™ programmable read-only memory (OBP) will enforce DHCP as the default mechanism while booting on IPoIB and flag error when instructed to do RARP-based installs.

Thus system administrators must use DHCP client identifiers to distinguish between IPoIB interfaces when using Jumpstart. The exact mechanism of how the client identifier for an interface can be set or obtained is platform specific.

Inetboot – In cases where an IPoIB interface is used for network bootup, for example using the `boot net` command, OBP guarantees that only DHCP will be used for IPoIB net booting and not RARP and the `bootparams`(4) database.

IPoIB in OBP will use user-provided or default assigned client-id for DHCP.

All changes to support a new media type are encapsulated in the Link Layer interface which is private to `inetboot`. Internally, this layer has functions to read and write network frames, do ARP operations, and return the link layer header length besides interface-specific parameters like `mtu`, `type`, and others.

The `inetboot` boot loader has intelligence about deciding whether the boot device is a network device (to prevent passing device arguments provided on the `boot net` command line to the kernel that Solaris OS will not understand). Given that the OBP `device_type` property on the IPoIB device node is not going to be `network`, this intelligence will now also match whether the `network-interface-type` property in the OBP is defined to identify a network device.

Configuring IPoIB

The `ibd` driver expects certain configuration of the InfiniBand Architecture (IBA) fabric before operation. Specifically, the IBA multicast group(s) representing the IPv4 broadcast and/or IPv6 all-node addresses must be created before initializing the device. All the IBA properties (`mtu`, `qkey`, `sl`, and others) of these groups must be the same.

To turn on the IPoIB service, add the following line to the `/kernel/drv/ib.conf` file using mechanisms described in the `ib`(7D) driver man page:

```
name="ib" class="root" port-svc-list="" vppa-svc-list="ipib"
```

Disabling Non-decimal Aliases on Logical Interfaces

Logical interfaces can no longer be plumbed or unplumbed using non-decimal aliases.

Solaris OS TCP/IP allows multiple logical interfaces to be associated with a physical network interface. This allows a single machine to be assigned multiple IP addresses, even though it may have only one network interface. Physical network interfaces have names of the form driver-name physical-unit-number, while logical interfaces have names of the form *`driver-name physical-unit-number`*:*`logical-unit-number`*. A physical interface is configured into the system using the `plumb` command. For example:

```
# ifconfig eri0 plumb
```

To delete a logical interface, use the `unplumb` option. For example:

```
# ifconfig eri0:1 down unplumb
```

Will delete the logical interface `eri0:1`.

In previous versions of Solaris OS, although undocumented, it was possible to use non-decimal numbers as the logical unit number (LUN).

Plumbing or unplumbing of invalid non-decimal representations (octal and hexadecimal [hex]) of logical interfaces was handled improperly. No error was returned for invalid octal and hex numbers and handling of non-decimal aliases was also not symmetric with respect to plumbing and unplumbing of logical interfaces.

For example:

```
# ifconfig hme0:099 plumb
```

This would result in the next available logical interface being assigned. The reason for that behavior was that the leading zero indicated an octal number, an octal 099 is a invalid number. Rather than return an error code indicating the invalid value, the result returned would be zero. Thus the `ifconfig` command would use the next available logical interface value.

Permitting non-decimal representations for interface identification was a bad architectural feature because it introduced ambiguity into the naming of interfaces. Two applications could configure the same interface through two different names.

Because the underlying functionality is unnecessary, undocumented, and hazardous to applications, it was removed rather than repaired. All attempts to use non-decimal aliases will cause IP to return an `EINVAL` (invalid) error message.

Use of non-decimal numbers for interface identification is not a documented feature, so no new documentation was needed.

IP DF Flag With Application-Supplied Header Data

This section describes a change in how the Solaris OS IP layer handles the fragment flag for application supplied headers.

When an application must supply the header to the IP layer, the application sends an `IP_HDRINCL` message to the IP. In prior Solaris OS releases, the IP layer determined if `ip_path_mtu_discovery` variable was set to `true`, which is the default, and then set the `DF` flag to `DF`, regardless of the setting that the application supplied.

The IP layer now will preserve the DF flag and use the value set in the application supplied header. Also, the man page, `IP`(7D), has been updated to list the fields that IP will not change under any condition when the `IP_HDRINCL` message is received with header data. These fields are as follows:

- Type of Service
- DF Flag
- Protocol
- Destination Address

Installation

Installation (IBoIP):

To turn on the IPoIB service, add the following line to the `/kernel/drv/ib.conf` file using mechanisms described in the `ib`(7D) driver man page:

```
name="ib" class="root" port-svc-list="" vppa-svc-list="ipib"
```

NFS Changes

Objectives

This module is an overview of the new features included in the Solaris™ 10 Operating System (Solaris 10 OS) that are in the Network File System (NFS) category. Upon completion of this module, you should be able to:

- Describe NFS enhancements for Solaris 9 OS First Customer Ship (FCS) through Solaris 10 OS FCS
- Describe the enhancements to Network File System version 4 (NFS version 4)
- Identify the common usage information relevant to product support
- Describe NFS hardware and software requirements

NFS Version 4

NFS version 4 is an Internet standard protocol used for distributed file sharing on a network.

NFS version 4 is a distributed file access protocol which owes its heritage to NFS protocol version 2, Request For Comment (RFC) 1094, and version 3, RFC 1813. Unlike earlier versions, the NFS version 4 protocol supports traditional file access while integrating support for file locking and the `mount` protocol. In addition, support for strong security (and its negotiation), compound operations, client caching, and internationalization have been added and NFS version 4 operates well in an Internet environment.

Features

The following are features of NFS version 4:

- Single protocol
- Strong security
- Compound procedures
- Extended attributes
- UTF-8 (UCS Transformation Format 8 (UTF-8), Universal Multiple-Octet Coded Character Set (UCS))
- Delegation

Single Protocol

NFS version 4 is stateful, and there are `OPEN` and `CLOSE` operations to obtain file data access and file-locking support integrated into the protocol. Functions previously handled by separate protocols (for example, `PORTMAP`, `MOUNT`, `NLM/NSM`, `ACL`) are incorporated into one protocol.

NFS version 4 handles file handle-to-path name mapping as well as byte range file locking. This removes the need for separate `lockd` and `mountd` daemons on the server and so reduces server-side support daemons and eases server-side implementation.

Pseudo-File System

Previous versions of NFS required use of the `mount` protocol, which does not use assigned ports. This made NFS hard to use through a firewall. Implementation of NFS version 4 must support Transmission Control Protocol/Internet Protocol (TCP/IP) to provide congestion control. NFS version 4 uses the well-defined port `2049`, thus improving firewall support.

NFS version 4 maps file handles to path names, which the `mountd` protocol did in previous versions of NFS. In NFS version 4, the server provides a root file handle that represents the top of the file system that the server exported. The server attaches multiple file systems with a pseudo-file system. The pseudo-file system provides paths that bridge non-exported portions of the real file system.

NFS version 4 servers create and maintain a pseudo-file system, which provides clients with seamless access to all exported objects on the server. Before NFS version 4, the pseudo-file system did not exist. Clients had to mount each shared server file system for access. Figure 9-1 shows an example:

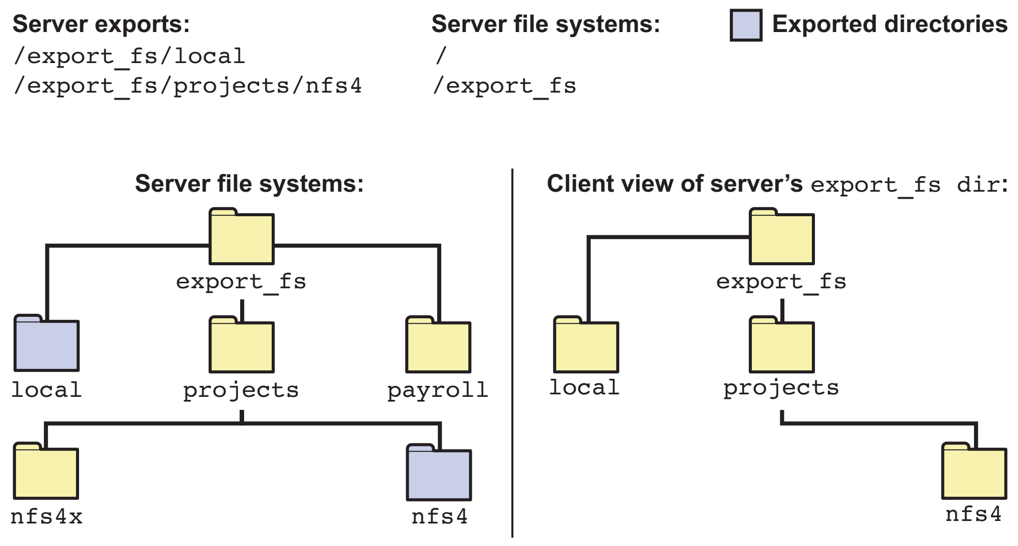


Figure 9-1 Views of the Server File System and Client File System

In Figure 9-1 the client cannot see the `payroll` directory and the `nfs4x` directory because these directories are not exported and do not lead to exported directories. However, the client can see the `local` directory because `local` is an exported directory. The `projects` directory is visible to the client because the `projects` directory leads to the `exported` directory, `nfs4`. Thus, portions of the server namespace that are not explicitly exported are bridged with a pseudo-file system that views only the exported directories and those directories that lead to server exports.

A pseudo-file system is a structure that contains only directories and is created by the server. The pseudo-file system permits a client to browse the hierarchy of exported file systems. Thus, the client's view of the pseudo-file system is limited to paths that lead to exported file systems.

Previous versions of NFS did not permit a client to traverse server file systems without mounting each file system. However, in NFS version 4, the server namespace does the following:

- Restricts the client's file-system view to directories that lead to server exports.
- Provides clients with seamless access to server exports without requiring that the client mount each underlying file system. See the previous example in Figure 9-1. However, different operating systems (OSs) might require the client to mount each server file system.

NFS version 3 is the default NFS version on Solaris 10 OS. The `nfs`(4) file in the `/etc/default` directory configures the client or server to use NFS version 4. In addition, the `mount` command (`mount_nfs` (1M)) can use the `vers=4` option to mount a file system using version 4 only.

Strong Security

NFS version 4 uses the remote procedure call (RPC) implementation of the General Security Service (GSS) framework to extend the basic security of RPC. This provides mechanisms for authentication, integrity, and privacy between the client and server.

Traditional RPC implementations included `AUTH_NONE`, `AUTH_SYS`, `AUTH_DH`, and `AUTH_KRB4` as security flavors. An additional security method of `RPCSEC_GSS` is introduced that uses the functionality of Generic Security Services Application Programming Interface (GSSAPI). This allows the RPC layer to use various security mechanisms without the additional implementation overhead of adding RPC security methods.

For NFS version 4, the `RPCSEC_GSS` security method must be used to enable the mandatory security mechanism. Other flavors, such as `AUTH_NONE`, `AUTH_SYS`, and `AUTH_DH` may be implemented as well.

The client negotiates with the server to determine the security mechanism that meets the requirements for the server and client. The `RPCSEC_GSS` framework delivers Sun Enterprise Authentication Mechanism™ (SEAM) software authentication.

Public Key Infrastructure (PKI) is supported by Low Infrastructure Public Key Mechanism (LIPKEY) and Simple Public Key Mechanism (SPKM) within the `RPCSEC_GSS` framework. These mechanisms allow authentication of a client without the requirement that the client have a key pair. The method is similar to what the Transport Layer Security (TLS) uses.

You can mix the security mechanisms on a single server, which allows security to be applied on a per-share basis.

To configure a Solaris 10 OS NFS version 4 server to use the `RPCSEC_GSS` security flavor with SEAM software, the administrator first edits the `/etc/nfssec.conf` file using the `nfssec` security modes described in the `nfssec`(5) man page to enable to enable the necessary security mode needed and then shares the file system with the `sec=mode` option.

The following is an example:

```
# share -F nfs -o sec=krb5 /export/home
```

Compound Procedures

To improve performance and Internet access, the NFS version 4 client combines multiple RPC request calls into a single compound procedure. By using compound procedures, clients can combine `LOOKUP`, `OPEN`, and `READ` operations in a single request. The server breaks the request into a list of separate requests. The server iterates through the list and performs each operation in the list until it reaches the end of the list or fails. The server then returns the results of the operations to the client.

The following is a simplified example of compound procedures. When reading the `/export/testdata` file, NFS versions 3 and 4 generate the following RPC calls:

NFS version 3	**NFS version 4**
`-> LOOKUP "export"`	`->OPEN "export/testdata"`
`<- OK`	`READ`
`->LOOKUP "testdata"`	`<- OPEN OK`
`<- OK`	`READ OK`
`-> ACCESS "testdata"`	`(sends data)`
`<- OK`	
`-> READ "testdata"`	

NFS version 3	NFS version 4
`<- OK`	
`(sends data)`	

Fewer RPC calls result in faster NFS response. This allows the client to tailor its request to appropriately match the operating environment of the client, thus enhancing cross-platform interoperability.

Extended Attributes

Earlier NFS versions used a fixed set of file and file system attributes that were modeled on the UNIX® type files and file systems. A non-UNIX-like server or client had to simulate those attributes, making implementation on a non-UNIX system difficult. NFS version 4 introduces three categories of attributes: mandatory, recommended, and named. All NFS version 4 clients and servers supported the mandatory attributes to ensure a minimum level of interoperability.

Not all clients or servers have to support the recommended attributes. This allows a server to support the attributes that apply to its operating environment. The client determines how to proceed if the server does not support a particular recommended attribute.

The named attribute is in the form of a byte stream that is associated with a file or file system and is referred to by a string name. This allows the client to associate data with a specific file or file system.

File handles are created on the server and contain information that uniquely identifies files and directories. In NFS versions 2 and 3, the server returned persistent file handles. This meant the client could guarantee that the server would generate a file handle that always referred to the same file. The following is an example:

- If a file was deleted and replaced with a file of the same name, the server would generate a new file handle for the new file. If the client used the old file handle, the server would return an error that the file handle was stale.
- If a file was renamed, the file handle would remain the same.
- If you had to reboot the server, the file handles would remain the same.

When the server received a request from a client that included a file handle, the resolution was straightforward, and the file handle always referred to the correct file.

This method of identifying files and directories for NFS operations was fine for most UNIX-based servers, but could not be implemented on servers that relied on other methods of identification such as a file's path name. To resolve this problem, the NFS version 4 protocol permits a server to declare that its file handles are volatile. Thus, a file handle could change. If the file handle does change, the client must find the new file handle.

Like NFS versions 2 and 3, the Solaris OS NFS version 4 server always provides persistent file handles. However, Solaris OS NFS version 4 clients that access non-Solaris OS NFS version 4 servers must support volatile file handles if the server uses them. Specifically, when the server tells the client that the file handle is volatile, the client must cache the mapping between path name and file handle. The client uses the volatile file handle until it expires. Upon expiration, the client does the following:

- Flushes the cached information that refers to that file handle
- Searches for that file's new file handle
- Retries the operation

UTF-8

File and directory names are UTF-8 encoded. This encoding includes 16 or 32 bit characters and allows one superset to handle all character sets. This allows the client and the server to be unaware of each other's localization and supports internationalization.

A UTF-8 string represents the `owner` and `owner_group` attributes (and also users and groups within the `ACL` attribute). This avoids presentation that is tied to a particular underlying implementation at the client or server. The client and server will have their own local representation of `owner` and `owner_group` that is used for local storage or presentation to the end user. When these attributes are transferred between the client and server, the local representation is translated to a syntax of the form *`user@dns_domain`*. For a client and server that do not use the same local representation, this allows translation to a common syntax that both can interpret.

The `nfsmapid` (1M) daemon maps to and from NFS version 4 `owner` and `owner_group` identification attributes and local user identification (UID) and group identification (GID) numbers that both the NFS version 4 client and server use. While `nfsmapid` has no external customer-accessible interfaces, the domain used can be configured by using the `NFSMAPID_DOMAIN` parameter in the `nfs` (4) configuration file.

Delegation

NFS version 4 provides both client support and server support for delegation. Delegation is a technique by which the server delegates the management of a file to a client. For example, the server could grant either a read delegation or a write delegation to a client. You can grant read delegations to multiple clients at the same time, because these read delegations do not conflict with each other. A write delegation can be to only one client, because a write delegation conflicts with any file accessed by any other client. While holding a write delegation, the client would not send various operations to the server because the client is guaranteed exclusive access to a file. Similarly, the client would not send various operations to the server while holding a read delegation because the server guarantees that no client can open the file in write mode.

The server alone decides whether to grant a delegation. A client does not request a delegation. The server decides based on the access patterns for the file. If several clients recently accessed a file in write mode, the server might not grant a delegation because this access pattern indicates the potential for future conflicts.

A conflict occurs when a client accesses a file in a manner that is inconsistent with the delegations that are currently granted for that file. For example, if a client holds a write delegation on a file and a second client opens that file for read or write access, the server recalls the first client's write delegation. Similarly, if a client holds a read delegation and another client opens the same file for writing, the server recalls the read delegation.

Note that in both situations, the second client is not granted a delegation because a conflict now exists. When a conflict occurs, the server uses a callback mechanism to contact the client that currently holds the delegation. Upon receiving this callback, the client sends the file's updated state to the server and returns the delegation. If the client fails to respond to the recall, the server revokes the delegation. In such instances, the server rejects all operations from the client for this file, and the client reports the requested operations as failures. Generally, these failures are reported to the application as input/output (I/O) errors.

To recover from these errors, the file must be closed and then reopened. Failures from revoked delegations can occur when a network partition exists between the client and the server while the client holds a delegation.

One server does not resolve access conflicts for a file that is stored on another server. Thus, an NFS server resolves only conflicts for files that it stores. Furthermore, in response to conflicts that are caused by clients that are running various versions of NFS, an NFS server can initiate only recalls to the client that is running NFS version 4. An NFS server cannot initiate recalls for clients that are running earlier versions of NFS.

The process for detecting conflicts varies. For example, unlike NFS version 4, because version 2 and version 3 do not have an open procedure, the conflict is detected only after the client attempts to read, write, or lock a file. The server's response to these conflicts varies also. The following are sample responses:

- For NFS version 3, the server returns the `JUKEBOX` error, which causes the client to halt the access request and retry later. The client prints the message: `File unavailable`.
- For NFS version 2, because an equivalent of the `JUKEBOX` error does not exist, the server makes no response, which causes the client to wait and then retry. The client prints the message `NFS server not responding`. Note that these conditions clear when the delegation conflict is resolved.

Solaris NFS version 4 servers do not manage conflict detection and resolution for locally accessed files. If a process running on the server accesses a local file that has been delegated, the NFS version 4 server is not notified and does not recall the delegation. Therefore, you should not enable server delegation if local access is permitted on the server. Your files can be corrupted if you enable delegation while permitting local access.

Note – By default, server delegation is disabled.

The NFS version 4 callback daemon, `nfs4cbd` (1M), provides the callback service on the client. This daemon is started automatically whenever a mount for NFS version 4 is enabled. By default, the client provides the necessary callback information to the server for all Internet transports that are listed in the `/etc/netconfig` system file. If the client is enabled for Internet Protocol version 6 (IPv6) and if the IPv6 address for the client's name can be determined, then the callback daemon accepts IPv6 connections.

The callback daemon uses a transient program number and a dynamically assigned port number. This information is provided to the server, and the server tests the callback path before granting any delegations. If the callback path fails, the server does not grant delegations, which is the only externally visible behavior.

Because callback information is embedded within an NFS version 4 request, the server cannot contact the client through a device that uses Network Address Translation (NAT). Also, the callback daemon uses a dynamic port number. Therefore, the server might not be able to traverse a firewall, even if that firewall enables normal NFS traffic on port 2049. In such situations, the server does not grant delegations.

Configuring NFS Version 4 Services

This section covers the procedures to configure an NFS version 4 server and client services.

Configuring an NFS Server

When configuring the NFS version 4 server, the first step is to add the appropriate entries in the `/etc/default/nfs` file. This file allows NFS to be configured without making changes to the `/etc/init.d/nfs.server` start scripts.

You must use the superuser account to edit the file.

1. Edit the `/etc/default/nfs` file.
2. Make the following entry:

   ```
   NFS_SERVER_VERSMAX=4
   ```

   While numerous parameters are supported, only those used to configure the NFS version 4 server are considered here.

   See the `nfs` (4) man page for a complete list of possible parameters.

```
NFS_SERVER_VERSMIN=num
NFS_SERVER_VERSMAX=num
```

The NFS server uses only NFS versions in the range these variables specify. Valid values or versions are: `2`, `3`, and `4`. As with the client, the default is to leave these variables commented out and the default minimum version is `2` while the default maximum version is `3`.

Note – If you want the server to provide only version 4, set the values for both `NFS_SERVER_VERSMAX` and `NFS_SERVER_VERSMIN` variables to `4`.

3. If required, make the following entry:

```
NFS_SERVER_DELEGATION=on
```

By default, this variable is commented out and the NFS server does not provide delegations to clients. The user can turn on delegations for all exported file systems by setting this variable to on (case-sensitive). This variable applies only to NFS version 4.

Caution – You should not enable server delegation if local access is permitted on the server. Your files can be corrupted if you enable delegation while permitting local access.

4. If required, make the following entry:

```
NFSMAPID_DOMAIN=my.comany.com
```

By default, the `nfsmapid` daemon uses the Domain Name Service (DNS) domain of the system. This setting overrides the default. This domain is used for identifying user and group attribute strings in the NFS version 4 protocol. Clients and servers must match with this domain for operation to proceed normally. This variable applies only to NFS version 4.

5. Determine if the NFS server is running:

```
# pgrep nfsd
```

If a Process ID (PID) is returned, you must use the following command to stop the service:

```
# /etc/init.d/nfs.server stop
```

6. Start the NFS service:

```
# /etc/init.d/nfs.server start
```

Configuring an NFS Client

When configuring the NFS version 4 client, the first step is to add the appropriate entries in the `/etc/default/nfs` file. This file allows NFS to be configured without making changes to the `/etc/init.d/nfs.client` start scripts.

You must use the superuser account to edit the file.

1. Edit the `/etc/default/nfs` file.
2. Insert the following line:

   ```
   NFS_CLIENT_VERSMAX=4
   ```

   While numerous parameters are supported, only those used to configure the NFS version 4 client are considered here.

   See the `nfs` (4) man page for a complete list of possible parameters.

   The NFS initiation process includes negotiating the protocol levels for servers and clients. If you do not specify the version level, then the best level is selected by default, based on the settings of the `NFS_CLIENT_VERSMIN` and `NFS_CLIENT_VERSMAX` keywords in the `/etc/default/nfs` file. If neither of these keywords is set, then the following rules are used:

   a. If both the client and the server can support version 3, then version 3 is used.

   b. If the client or the server can support only version 2, then version 2 is used.

   c. If either of these keywords is set in the `/etc/default/nfs` file, then the specified value replaces the default values. The default minimum value is `2` and the default maximum value is `3`. To find the version that the server supports, the NFS client begins with the setting for `NFS_CLIENT_VERSMAX` and continues to try each version until reaching the version setting for `NFS_CLIENT_VERSMIN`. As soon as the supported version is found, the process terminates. For example, if `NFS_CLIENT_VERSMAX=4` and `NFS_CLIENT_VERSMIN=2`, then the client attempts version 4 first, then version 3, and finally version 2.
3. Mount a file system.

   ```
   # mount server_name:share_point local_dir
   ```

   `server_name` – Provides the name of the server

share_point – Provides the path of the remote directory to be shared

local_dir – Provides the path of the local mount point

Using the Command Line to Enable NFS Version 4 on a Client

If you prefer to use the command line to enable NFS version 4 on a client, mount NFS version 4 on the client using the `mount` command:

```
# mount -o vers=4 server_name:/share_point /local_dir
```

See the `mount_nfs` (1M) man page for a complete list of mount options.

Tunable Kernel Parameters

The following list describes tunable kernel parameters related to NFS version 4. You can define these parameters in the `/etc/system` file, which is read during the boot process. You can identify each parameter by the name of the kernel module that it is in and a parameter name that identifies it. Each entry includes a description and suggestions when you might change the setting.

Solaris is a multi-threaded, scalable UNIX OS that runs on SPARC® and x86 processors. It is self-adjusting to system load and demands minimal tuning. A key consideration in system tuning is that setting various system variables is often the least effective task you can do to improve performance. Carefully consider the values in the file with respect to the environment in which they are applied. Make sure that you understand the behavior of a system before attempting to apply changes to the system variables that are described here.

Caution – The variables described here and their meanings can and do change from release to release. A release is either a Solaris OS update release or a new version such as Solaris 10 OS. Publication of these variables and their description does not preclude changes to the variables and descriptions without notice.

- `nfs:nfs4_cots_timeo`

  Description – Controls the default RPC timeout for NFS version 4 mounted file systems using connection oriented transports such as Transmission Control Protocol (TCP) for the transport protocol.

The NFS Version 4 protocol specification disallows retransmission over the same TCP connection. Thus, this parameter primarily controls how quickly the client responds to certain events, such as detecting a forced unmount operation or how quickly the server fails over to a new server.

When to Change – TCP does a good job ensuring requests and responses are delivered appropriately. However, if the round-trip times are very large in a particularly slow network, the NFS version 4 client might time out prematurely.

Increase this parameter to prevent the client from timing out incorrectly. The range of values is very large, so increasing this value to be too large might result in real situations where a retransmission was required to not be detected for long periods of time.

- `nfs:nfs4_pathconf_disable_cache`

  Description – Controls the caching of `pathconf` (2) information for NFS version 4 mounted file systems.

  When to Change – The `pathconf` information is cached on a per file basis. However, if the server can change the information for a specific file dynamically, use this parameter to disable caching because there is no mechanism for the client to validate its cache entry.

- `nfs:nfs4_lookup_neg_cache`

  Description – Controls whether a negative name cache is used for NFS version 4 mounted file systems. This negative name cache records filenames that were looked up, but were not found. The cache is used to avoid over-the-network lookup requests made for filenames that are already known to not exist.

  When to Change – For the cache to perform correctly, negative entries must be strictly verified before use. This consistency mechanism is relaxed slightly for read-only mounted file systems by assuming that the file system on the server is not changing or is changing very slowly and that it is okay for such changes to propagate slowly to the client. The consistency mechanism becomes the normal attribute cache mechanism in this case.

  If file systems are mounted read-only on the client, but are expected to change on the server and the client must see these changes immediately, use this parameter to disable the negative cache.

  If you disable the `nfs:nfs_disable_rddir_cache` parameter, you should probably also disable this parameter. For more information, see `nfs:nfs_disable_rddir_cache`.

- `nfs:nfs4_max_threads`

  Description – Controls the number of kernel threads that perform asynchronous I/O for the NFS version 4 client. Since NFS is based on RPC, and RPC is inherently synchronous, separate execution contexts are required to perform NFS operations that are asynchronous from the calling thread.

  The operations that can be executed asynchronously are read for read-ahead, write-behind, directory read-ahead, and cleanup operations that the client performs when it stops using a file.

  When to Change – Change this parameter to increase or reduce the number of simultaneous I/O operations that are outstanding at any given time. For example, for a very low bandwidth network, you might want to decrease this value so that the NFS client does not overload the network. Alternately, if the network is very high bandwidth and the client and server have sufficient resources, you might want to increase this value to more effectively use the available network bandwidth and the client and server resources.

- `nfs:nfs4_nra`

  Description – Controls the number of read-ahead operations that the NFS version 4 client queues when sequential access to a file is discovered. These read-ahead operations increase concurrency and read throughput. Each read-ahead request is generally for one logical block of file data.

  When to Change – Change this parameter to increase or reduce the number of read-ahead requests that are outstanding for a specific file at any given time. For example, for a very low bandwidth network or on a low memory client, you might want to decrease this value so that the NFS client does not overload the network or the system memory. Alternately, if the network is very high bandwidth and the client and server have sufficient resources, you might want to increase this value to more effectively use the available network bandwidth and the client and server resources.

- `nfs:nfs4_shrinkreaddir`

  Description – Some NFS servers might incorrectly handle NFS version 4 `READDIR` requests for more than 1024 bytes of directory information. This is due to a bug in the server implementation. However, this parameter contains a workaround in the NFS version 4 client.

  When this parameter is enabled, the client does not generate a `READDIR` request for larger than 1024 bytes of directory information. If this parameter is disabled, then the over-the-wire size is set to the minimum of either the size passed in by using the `getdents`(2) system call or by using `MAXBSIZE` variable setting, which is 8192 bytes.

When to Change – Examine the value of this parameter if an NFS version 4 only server is used and interoperability problems occur when trying to read directories. Enabling this parameter might degrade performance slightly for applications that read directories.

- `nfs:nfs4_bsize`

  Description – Controls the logical block size that the NFS version 4 client uses. This block size represents the amount of data that the client attempts to read from or write to the server when it must do an I/O.

  When to Change – Examine the value of this parameter when attempting to change the maximum data transfer size. Change this parameter in conjunction with the `nfs:nfs4_max_transfer_size` parameter. If larger transfers are desired, increase both parameters. If smaller transfers are desired, then just reducing this parameter should suffice.

- `nfs:nfs4_async_clusters`

  Description – Controls the mix of asynchronous requests that the NFS version 4 client generates. There are six types of asynchronous requests: read-ahead, putpage, pageio, readdir-ahead, commit, and inactive. The client attempts to round-robin between these different request types to attempt to be fair and not starve one operation type in favor of another.

  However, functionality in some NFS version 4 servers, such as write gathering, depends upon certain behaviors of existing NFS version 4 clients. In particular, this functionality depends upon the client sending multiple `WRITE` requests at approximately the same time. If one request at a time is removed from the queue, the client would defeat this server functionality, which is designed to enhance client performance.

  Thus, use this parameter to control the number of requests of each type that are sent before changing types.

  When to Change – Change this parameter to increase the number of each type of asynchronous operation that is generated before switching to the next type. This might help with server functionality that depends upon clusters of operations coming from the client.

- `nfs:nfs4_max_transfer_size`

  Description – Controls the maximum size of the data portion of an NFS version 4 `READ`, `WRITE`, `READDIR`, or `READDIRPLUS` request. This parameter controls both the maximum size of request that the server returns as well as the maximum size of a request that the client generates.

  When to Change – Change this parameter to tune the size of data transmitted over the network. In general, the `nfs:nfs4_bsize` parameter should also be updated to reflect changes in this parameter.

  For example, when attempting to increase the transfer size beyond 32 kilobytes (Kbytes), update `nfs:nfs4_bsize` to reflect the increased value. Otherwise, no change in the over-the-wire request size is seen. For more information, see `nfs:nfs4_bsize`.

  To use a smaller transfer size than the default, use the `mount` command's `-wsize` or `-rsize` options on a per-file system basis.

- `nfs:nfs4_dynamic`

  Description – Controls whether a feature known as dynamic retransmission is enabled for NFS version 4 mounted file systems using connectionless transports such as User Datagram Protocol (UDP). This feature attempts to reduce retransmissions by monitoring server response times and then adjusting RPC timeouts and read and write transfer sizes.

  When to Change – In a situation where server response or network load varies rapidly, the dynamic retransmission support might incorrectly increase RPC timeouts or reduce read and write transfer sizes unnecessarily. Disabling this functionality might result in increased throughput. However, disabling might also increase the visibility of the spikes due to server response or network load.

  This parameter is set per file system at mount time. To affect a particular file system, unmount and mount the file system after changing this parameter.

- `nfs:nfs4_do_symlink_cache`

  Description – Controls whether the contents of symbolic link files are cached for NFS version 4 mounted file systems.

  When to Change – If a server changes the contents of a symbolic link file without updating the modification time stamp on the file or if the granularity of the time stamp is too large, then changes to the contents of the symbolic link file might not be visible on the client for extended periods. In this case, use this parameter to disable the caching of symbolic link contents, thus making the changes visible to applications running on the client immediately.

Changed Packages and Files

- `SUNWnfssrc`
- `SUNWfscu`
- `SUNWnfssr`
- `SUNWnfssu`

There are two new daemons:

- `nfsmapid` (1M)
- `nfs4cbd` (1M)

There is one new configuration file:

`nfs` (4

Using NFS Over RDMA

Remote Direct Memory Access (RDMA) protocol is a technology for memory-to-memory transfer of data over high speed networks.

The InfiniBand Architecture (IBA) is an industry standard that defines a new high-speed switched fabric subsystem. Solaris 10 OS supports Internet Protocol (IP) over InfiniBand (IPoIB). IPoIB is compatible with existing TCP, UDP, IPv4, and IPv6. Support for IB has been added to the IP utilities `netstat`, `ifconfig`, and `snoop`.

RDMA provides remote data transfer directly to and from memory without central processing unit (CPU) intervention. RDMA over IB provides direct data placement, which eliminates data copies and, therefore, further eliminates CPU intervention. Thus, RDMA relieves not only the host CPU, but also reduces contention for the host memory and I/O buses. To provide this capability, RDMA combines the interconnect I/O technology of InfiniBand on SPARC platforms with the Solaris OS. Figure 9-2 shows the relationship of RDMA to other protocols, such as UDP and TCP.

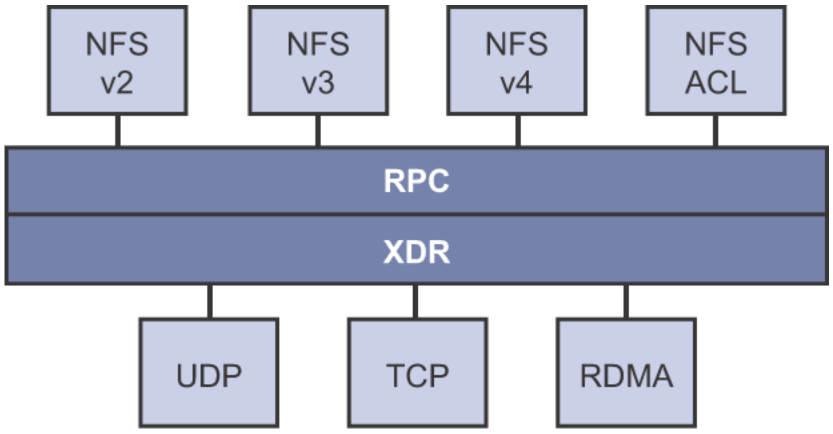


Figure 9-2 Relationship of RDMA to Other Protocols

NFS is a family of protocols layered over RPC. The External Data Representation (XDR) layer encodes RPC arguments and RPC results onto one of several RPC transports, such as UDP, TCP and RDMA.

Because RDMA is the default transport protocol for NFS, no special `share` or `mount` options are required to use RDMA on a client or server. RDMA is used if the Solaris OS detects IB host channel adaptor hardware on the system. The existing `automount` maps, `vfstab` file, and `dfstab` file work with the RDMA transport. NFS mounts over the RDMA transport occur transparently when IB connectivity exists on SPARC platforms between the client and the server. If the RDMA transport is not available on both the client and the server, the TCP transport is the initial fallback, followed by UDP if TCP is unavailable. Note, however, that if you use the `mount` option `proto=rdma`, NFS mounts are forced to use RDMA only.

Security Changes

Objectives

This module is an overview of the new features included in the Solaris™ Operating System 9 (Solaris 9 OS) through Solaris 10 Operating System (Solaris 10 OS) that are part of the Internet Protocol (IP) networking category.

Upon completion of this module, you should be able to:

- Explain the Solaris OS Cryptographic Framework (SCF)
- Understand the Hardware Accelerator for Internet Key Exchange (IKE)
- Explain and use the Solaris OS IP Filter firewall

SCF

The SCF provides cryptographic services to applications and kernel modules in a manner seamless to the end user. The SCF also brings direct cryptographic services, like encryption and decryption for files, to the end user through commands, a user-level programming interface, a kernel programming interface, and user-level and kernel-level frameworks.

The user-level framework is responsible for providing cryptographic services to consumer applications and the end user commands. The kernel-level framework provides cryptographic services to kernel modules and device drivers. Both frameworks give developers and users access to software optimized cryptographic algorithms.

The following terms are defined to aid the reader.

- AES – Advanced Encryption Standard, a United States Federal Government standard block cipher standard
- ARCFOUR – Also ARC4, the public implementation of RC4 (see RC4 in this list)
- Blowfish – A block cipher that Bruce Schneier designed in 1993
- CA – Certificate Authority
- DES – Data Encryption Standard, a cipher that the AES cipher has superseded as a United States Federal Government standard
- 3DES – A block cipher formed from the DES cipher
- DH – The Diffie-Hellman key agreement protocol, a protocol that allows two users to exchange a secret key over an insecure medium without any prior secrets
- DSA – Digital Signature Algorithm is a United States Federal Government standard for digital signatures
- ELF – Executable and Linkable Format
- HMAC – Keyed-hash message authentication code is a type of message authentication code (MAC) calculated using a cryptographic hash function in combination with a secret key
- MAC – Message Authentication Code is an algorithm for generating a short string of information used to authenticate a message
- MD5 – RSA Data Security, Inc.'s MD5 message-digest algorithm
- Mechanism – A process for implementing a cryptographic operation

- Provider – A cryptographic service: a user-level provider is a PKCS#11 library, a kernel software provider is a loadable kernel software module, and a kernel hardware provider is a cryptographic hardware device
- PBKDF2 – Key derivation function, KDF2 from PKCS#5 (see PKCS#5 in this list): Password-Based Cryptography (PBE)
- PKCS – Public-Key Cryptography Standards
- PKCS#5 – Password Based Cryptography Standard
- PKCS#11 – RSA Data Security, Inc's Cryptographic Token Interface Standard
- RSA – The RSA public-key cryptosystem
- RC4 – RSA Data Security's proprietary RC4 symmetric stream cipher
- Reader – The means by which information is exchanged with a device
- SHA-1 – The (revised) Secure Hash Algorithm
- Slot – A logical reader that potentially contains a token
- Token – The logical view of a cryptographic device that PKCS#11 defines

SCF Architecture Overview

Figure 10-1 is an overview of the SCF architecture.

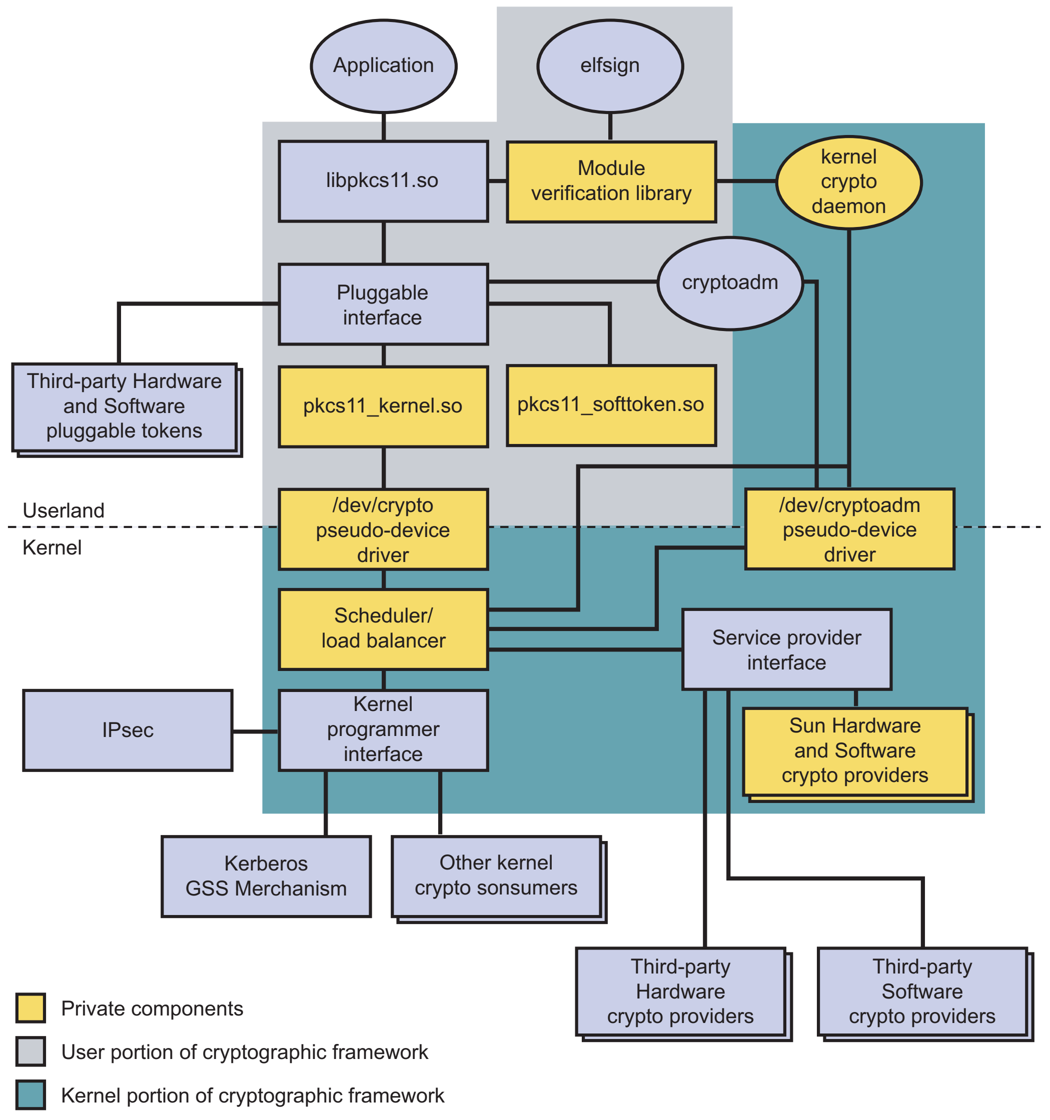

Application
elfsign
libpkcs11.so
Module verification library
kernel crypto daemon
Pluggable interface
cryptoadm
Third-party Hardware and Software pluggable tokens
pkcs11_kernel.so
pkcs11_softtoken.so
Userland
Kernel
/dev/crypto pseudo-device driver
/dev/cryptoadm pseudo-device driver
Scheduler/ load balancer
Service provider interface
IPsec
Kernel programmer interface
Sun Hardware and Software crypto providers
Kerberos GSS Merchanism
Other kernel crypto sonsumers
Third-party Hardware crypto providers
Third-party Software crypto providers
Private components
User portion of cryptographic framework
Kernel portion of cryptographic framework

Figure 10-1SCF Architecture

The SCF Architectural Components

Components found in the application layer generally are not intended to provide generalized features to other applications, but rather perform a single class of function. An example application would be a Sendmail mail server whose role is to route the transfer of an email message towards its recipient. The mail server may make use of cryptography in order to protect the message as it traverses a hop to the next mail server. However the mail server does not provide protection for any other type of communication between the nodes.

In Solaris's cryptographic architecture, applications will not directly contain cryptographic algorithms (for example, DES). Instead they will just make use of features of the SCF available from the Solaris Operating System. This reduces code redundancy and provides access to optimized algorithms for all applications.

PKCS#11 is used as the interface for applications. Applications cannot call individual cryptographic algorithms directly, but must provide key material, where appropriate, and specification of which algorithm(s) they wish to use. PKCS#11 contains more than just access to encryption primitives. However, currently SCF only provides implementations of the functions required for: encryption (DES, 3DES, AES, Blowfish, ARCFOUR) message digests (MD5 SHA), signing (RSA DSA DH) random number generation. Object management and storage functionality is not provided.

PKCS#11 is also used as the Service Provider Interface (SPI) for third parties to replace or augment provided algorithms. The framework does not limit the functionality of providers plugged in through the SPI; all PKCS#11 functionality is made available. In particular, if available from a provider, access to hardware based key-stores is available. Using PKCS#11 as the SPI allows easy access to third-party cryptographic tokens such as hardware accelerator cards and smartcard-like products used for secure key storage.

The PKCS#11 library uses a local policy file and internal metrics to decide which implementation of a given algorithm to use. For example, if 3DES is available in both hardware and software, the PKCS#11 library decides which to use based on the local policy file or internal metrics.

The `libpkcs11.so` Library

Normally applications that use PKCS#11 are either linked to a single vendors PKCS#11 implementation or they provide their own mechanism to manage the multiple libraries through dynamic loading. This is a significant burden on application vendors and does not lead to a consistent view of cryptography through the system.

SCF provides a pluggable framework library (`libpkcs11.so`) so that applications can link to and access multiple different PKCS#11 implementations, without the applications having to deal with dynamic loading and configuration themselves.

Each plug-in to the `libpkcs11.so` library appears as a PKCS#11 slot. Since a plug-in may provide more than one slot, `libpkcs11` maps all of the available slots for each plug-in into the list of slots that the library provides to the application.

Slot numbers need not be sequential, nor need they start at 0. The slot number is opaque to the application and it must not assume that the same functionality will be available at the same slot number in a future invocation. This usually requires an application to store a significant amount of state data to allow it to use the same slot on later invocations. To overcome this complexity, the `libpkcs11.so` library provides a meta slot.

The meta slot is intended for applications that just want to use PKCS#11 for cryptographic operations and do not need to store or retrieve objects on a specific token. The meta slot can provide a richer set of PKCS#11 mechanisms, the aggregate of those that each of the plug-ins provided, while reducing the complexity of the application code, since the meta slot does not need to deal with multiple PKCS#11 slots.

The meta slot allows session objects to be created but not persistent objects. This restriction is due to the complexities in the PKCS#11 specification with respect to object visibility and the potential need to login to a token before objects become visible. There is also no easy algorithm for choosing the token on which to store the objects and no easy way to retrieve the objects if the configuration changes.

The meta slot only exists if there is more than one slot available when all configured plug-ins are probed. The meta slot is the first slot returned from the `C_GetSlotList()` function. In the default installation of Solaris OS, with no additional third-party providers or cryptographic hardware, there is only the single slot that the `pkcs11_softtoken.so` library provides, so the meta slot does not exist.

The `pkcs11_softtoken.so` Library

The `pkcs11_softtoken.so` library provides a software-only implementation of the RSA Data Security's PKCS#11 version 2.11 specification.

The following cryptographic algorithms are implemented in the `pkcs11_softtoken.so` library:

- DES
- 3DES
- AES
- Blowfish
- ARCFOUR
- MD5
- SHA1
- RSA
- DSA
- DH

Due to import restrictions on cryptographic algorithms, in some countries it is necessary to supply two different versions of the library. The `pkcs11_softtoken.so` library contains implementations of the symmetric key algorithms with key lengths up to 128-bits; message digest and asymmetric key lengths are not limited. A second version of the `pkcs11_softtoken.so` library called `pkcs11_softtoken.extra.so` is available in the Solaris OS Encryption Kit, and may contain longer key lengths for some algorithms – AES, Blowfish and ArcFour – that support key lengths greater than 128 bits.

Since the `pkcs11_softtoken_extra.so` library is a true super set of the `pkcs11_softtoken.so` library, its installation disables the `pkc11_softtoken.so` library. This ensures that the only visible changes are implementations of the algorithms with longer key lengths.

Solaris OS Kernel Framework

SCF delivers a framework for the cryptographic services in the Solaris OS kernel. It includes a programming interface for kernel-level consumers and a system call interface for user libraries enabling both sides to access the cryptographic services in the Solaris OS kernel and a service provider interface for Sun and third party pluggable kernel cryptographic providers. An administrative interface is also provided as part of SCF.

The system call interface has to provide access to the full capabilities offered by kernel providers to a user-level library. Kernel providers plug-in into the kernel SPI using an adaptation of the PKCS#11 standard interface, which is the interface that all the hardware cryptographic product vendors currently support.

The kernel provides access to hardware accelerators. SCF imports the `/dev/crypto` and `sys/crypto.h` interfaces. The kernel presents an ordered list of slots through the interface. This list contains virtual slots, which are used to represent groups of hardware providers with the same mechanisms, and slots that correspond to specific hardware providers.

SCF supports Sun IPsec, Kerberos, and the Hardware Accelerator for IKE.

Cryptography Providers

The cryptographic providers are the actual cryptographic algorithms. This includes software implementations of 3DES, AES, RSA, and SHA-1 just to name a few in the Sun Software Crypto Plugin. These providers offer a PKCS #11 compliant user-level interface and PKCS #11 like kernel-level interface that is designed to plug into the cryptographic frameworks.

Since it is possible for 3rd parties to write new or replace existing algorithms used by the Solaris cryptographic framework, the framework validates that each plugin provider has an appropriate, embedded cryptographic signature. This signature limits who can write a provider that the framework will recognize and allow to operate and potentially limit the consumers who can make use of it.

The cryptoadm(1M) Utility

The cryptoadm utility displays cryptographic provider information for a system, configures the mechanism policy for each provider, and installs or uninstalls a cryptographic provider.

For kernel-level software providers, the cryptoadm utility provides two subcommands. The unload subcommand instructs the kernel to unload a kernel software providers module. The refresh subcommand resynchronizes the kernel-level cryptographic framework with the current kernel-level configuration.

For the kernel-level cryptographic framework daemon, kcfd(1M), the cryptoadm utility provides two subcommands. The start subcommand starts the daemon and the stop subcommand stops the daemon.

With the exception of the list and help subcommands and their options, the cryptoadm command must be run by a privileged user.

The following are examples of cryptoadm command usage. You will also use this command in the lab.

To list available providers, enter the following:

```
entropy%> cryptoadm list
user-level providers:
    /usr/lib/security/$ISA/pkcs11_kernel.so
    /usr/lib/security/$ISA/pkcs11_softtoken.so

kernel software providers:
    des
    aes
    blowfish
    arcfour
    sha1
    md5
    rsa

kernel hardware providers:
    dca/0
```

The preceding example lists the user-level providers as those libraries, and lists the kernel-level software providers by a generic name. The kernel-level hardware provider is listed as the device instance.

In this example the mechanisms are listed:

```
entropy%> cryptoadm list -m
user-level providers:
=====================
/usr/lib/security/$ISA/pkcs11_kernel.so: no slots presented.
/usr/lib/security/$ISA/pkcs11_softtoken.so:
CKM_DES_CBC,CKM_DES_CBC_PAD,CKM_DES_ECB,CKM_DES_KEY_GEN,
CKM_DES3_CBC,CKM_DES3_CBC_PAD,CKM_DES3_ECB,CKM_DES3_KEY_GEN,
CKM_AES_CBC,CKM_AES_CBC_PAD,CKM_AES_ECB,CKM_AES_KEY_GEN,
…

kernel software providers:
==========================
des: CKM_DES_ECB,CKM_DES_CBC,CKM_DES3_ECB,CKM_DES3_CBC
aes: CKM_AES_ECB,CKM_AES_CBC
arcfour: CKM_RC4
blowfish: CKM_BF_ECB,CKM_BF_CBC
sha1: CKM_SHA_1,CKM_SHA_1_HMAC,CKM_SHA_1_HMAC_GENERAL
md5: CKM_MD5,CKM_MD5_HMAC,CKM_MD5_HMAC_GENERAL
rsa: CKM_RSA_PKCS,CKM_RSA_X_509,CKM_MD5_RSA_PKCS,CKM_SHA1_RSA_PKCS

kernel hardware providers:
==========================
dca/0: CKM_DES_CBC,CKM_DES3_CBC,CKM_RSA_X_509
```

The output of the preceding command is shortened. Notice that a set of mechanism follows the generic provider name.

To list the mechanisms for a specific provider, in this case the hardware provider, enter the following:

```
entropy# cryptoadm list -m dca/0
dca/0: CKM_DES_CBC,CKM_DES3_CBC,CKM_RSA_X_509
```

List the available mechanisms, enter the following:

```
entropy# cryptoadm list -p dca/0
dca/0: all mechanisms are enabled.
```

To disable a specific mechanism, enter the following:

```
entropy# cryptoadm disable dca/0 CKM_DES_CBC
entropy# cryptoadm list -p dca/0
dca/0: all mechanisms are enabled except CKM_DES_CBC.
```

The digest(1M) Utility

The `digest` utility calculates the message digest of the given file or files using a specified algorithm. If more than one file is given, each line of the output is the digest of a single file.

One use of the `digest` command is with the Solaris OS Fingerprint Database to verify Solaris OS file integrity.

To list available digest algorithms, enter the following:

```
entropy%> digest -l
md5
sha1
```

Specify the digest algorithm to be used with the `-a` option.

```
entropy%> digest -a md5 /usr/bin/login
eed532cc83d97d726dbdb577ed85c415
```

Using the `-v` option displays verbose information similar to the Berkeley-style `md5sum` command:

```
entropy%> digest -v -a md5 /usr/bin/login
md5 (/usr/bin/login) = eed532cc83d97d726dbdb577ed85c415
```

The following example creates SHA1 digests of four files that are in the local `docs` directory and redirects the output to a file called `digest.docs.legal`.

```
entropy% digest -v -a sha1 ~/docs/* > ~/digest.docs.legal
entropy% more ~/digest.docs.legal
sha1 (docs/legal1) = 1df50e8ad219e34f0b911e097b7b588e31f9b435
sha1 (docs/legal2) = 68efa5a636291bde8f33e046eb33508c94842c38
sha1 (docs/legal3) = 085d991238d61bd0cfa2946c183be8e32cccf6c9
sha1 (docs/legal4) = f3085eae7e2c8d008816564fdf28027d10e1d983
```

The mac(1M) Utility

The `mac` utility calculates the message authentication code (MAC) of the given file, files, or standard in using the algorithm specified.

If more than one file is given, each line of output is the MAC of a single file.

The mac command is similar to the digest(1M) command but combines the input file with a key of some length. A digest is then returned of the combination.

For instance, when using the cram-md5 authentication method with Post Office Protocol 3 (POP3) or Internet Message Access Protocol (IMAP) mail services, a clear text password is stored on the server machine. When a user attempts to login to the mail service, the server sends a challenge in the form of a randomly generated key and other information in a message.

The user is prompted for a password. The password that the user enters is combined with the message, the random key is used to generate a MAC digest of the message, and the result is returned to the server. The server performs a similar operation using the stored password and random key. The digest is compared and, if the two digests match, the user is authenticated. So even though the password is used for authentication, the password was never sent in a message between the two machines.

The following example list the algorithms available for use and their required key lengths:

```
entropy%> mac -l
Algorithm       Keysize:  Min   Max
-----------------------------------
des_mac                    64    64
sha1_hmac                   8   512
md5_hmac                    8   512
```

This example uses the mac command to create an md5_hmac of a file with a manually entered key:

```
entropy%> mac -a md5_hmac /etc/hosts
Enter key:key_typed_in
2f3b84524c7fa61848c439af487006bc
```

The file is not changed. The manually entered key, called a passphrase, is manipulated into a more secure key using the PBKDF2 algorithm specified in PKCS#5.

The encrypt(1M) Utility

This utility encrypts or decrypts a given file or standard input using the algorithm specified. If no output file is specified, output is to standard out. If input and output are the same file, the encrypted output is written to a temporary work file in the same file system and then used to replace the original file.

On decryption, if the input and output are the same file, the clear text replaces the cipher text file.

The supported algorithms are displayed with their minimum and maximum key sizes in the `-l` option. The SCF provides these algorithms. Each supported algorithm is an alias of the PKCS #11 mechanism that is the most commonly used and least restricted version of a particular algorithm type. For example: `des` is an alias to CKM_DES_CBC_PAD mechanism and `arcfour` is an alias to CKM_RC4 mechanism. Algorithm variants with no padding or electronic codebook (ECB) modes are not supported.

In the following example, a file system is backed up to a tape device using the `ufsdump` command, then encrypted using the `encrypt` command and the arcfour algorithm. The key information has been previously stored in a file:

```
entropy%> ufsdump 0f - /var | encrypt -a arcfour -k \
/etc/mykeys/backup.k | dd of=/dev/rmt/0
```

This following is an example of restoring the preceding backup.

```
entropy%> decrypt -a arcfour -k /etc/mykeys/backup \
-i /dev/rmt/0 | ufsrestore vf -
```

The following example lists the available algorithms and key lengths:

```
entropy%> encrypt -l
Algorithm       Keysize:  Min   Max
-----------------------------------
aes                       128   128
arcfour                     8   128
des                        64    64
3des                      192   192
```

Hardware Acceleration for IKE

The IKE configuration file has been changed to add support for hardware acceleration.

To enable hardware acceleration, an optional keyword must be inserted into the IKE configuration file, `/etc/inet/ike/config`. Unlike other parameters in the file, it is read only once at IKE start time, and read again if a new configuration file is added or reloaded through the `ikeadm`(1m) utility, this parameter is still set to its run-time setting:

```
    pkcs11_path  path_to_PKCS-library
```

The string is an absolute pathname to a 32-bit PKCS#11 library, for example `/opt/SUNWconn/lib/libpkcs11.so`. If that library is present, IKE applies the `dlopen`() function to it and uses its PKCS#11 routines for IKE acceleration.

This feature works with the Sun™ Crypto Accelerator 4000 board or Sun Crypto Accelerator 1000 board along with other hardware accelerators that support the Cryptoki (PKCS#11) standard.

The feature also provides for the use of a hardware key store when running with a hardware cryptography engine. It it intended to work on any PKCS#11 shared library that provides support for hardware key stores. In particular the PKCS#11, which Solaris OS Encryption Framework provides, works and supports multiple hardware providers. When using a non-encryption framework PKCS#11 library, hardware providers are supported through the PKCS#11 implementation they provide.

The IP Filter Firewall

IP Filter provides packet filtering by IP address, port, protocol, network interface, and traffic direction. IP Filter can filter by individual source or destination IP address, or by ranges of addresses. It can also perform Network Address Translation (NAT), Port Address Translation (PAT), and be configured for stateful packet inspection. It is configured using a simple rules language. It provides simple command-line tools for administration, including tools to load rules into the kernel, do monitoring, logging, testing, and statistics. It is based on the open source IP Filter firewall version 4.0.33 that Darren Reed wrote and maintains.

Features of IP Filter

IP Filter provides a set of user-level utility programs and two kernel modules:

- `ipf`
- `pfil`

The `ipf` module is the core packet filtering and NAT logic, which uses the `pfil` module to hook into the system and gain control of each IP packet. The user-level utility programs push configuration data into the `ipf` kernel module through its `/dev/ipf` and `/dev/ipnat` interfaces and extract log and status information through its `/dev/ipl` and `/dev/ipstate` interfaces.

The `pfil` kernel module hooks into the Solaris OS IP stack through the use of a STREAMS module and the `autopush`(1M) and `sad`(7D) facilities. This means that, once the `autopush` facility is configured, the `pfil` STREAMS module is automatically pushed on any stream opened to a network device driver. IP Filter automatically configures the `autopush` facility in its startup script for any device driver that has an instance named in any `/etc/hostname.*` file. In a system that has already had all of its IP interfaces configured by creation of `/etc/hostname.*` files, IP Filter automatically configures the `autopush` facility at each reboot so that the `pfil` STREAMS module is pushed on all network interfaces and all packets are filtered. To minimize the effect on system operation, especially at boot time, when the IP Filter software is installed but not configured, no action is taken at boot when there is no IP Filter configuration file in the conventional location. Thus the `pfil` module is only has the `autopush` facility applied if IP Filter has been explicitly configured.

If a new kind of device is plumbed, for example, a `ce` interface is plumbed on a system that previously used only `hme` interfaces. A reboot or manual steps to configure autopush before plumbing any new interfaces are required to get IP Filter and the `pfil` module to recognize the new kind of Network Interface Card (NIC) device driver and reconfigure the `autopush` facility.

The user-level utilities facilitate logging, adding address and rule data, and report statistics:

- `ipf`(1M) – Used to add and remove rules in the `ipf` kernel module
- `ipfs`(1M) – The `ipfs` utility enables the saving of state information across reboots
- `ipfstat`(1M) – Reports on packet filter statistics and filter list
- `ipmon`(1M) – Used for logging
- `ipnat`(1M) – Used to add and remove NAT rules
- `ippool`(1M) – Used to manage information stored in the IP pools subsystem

Figure 10-2 shows the IP Filter packet processing sequence.

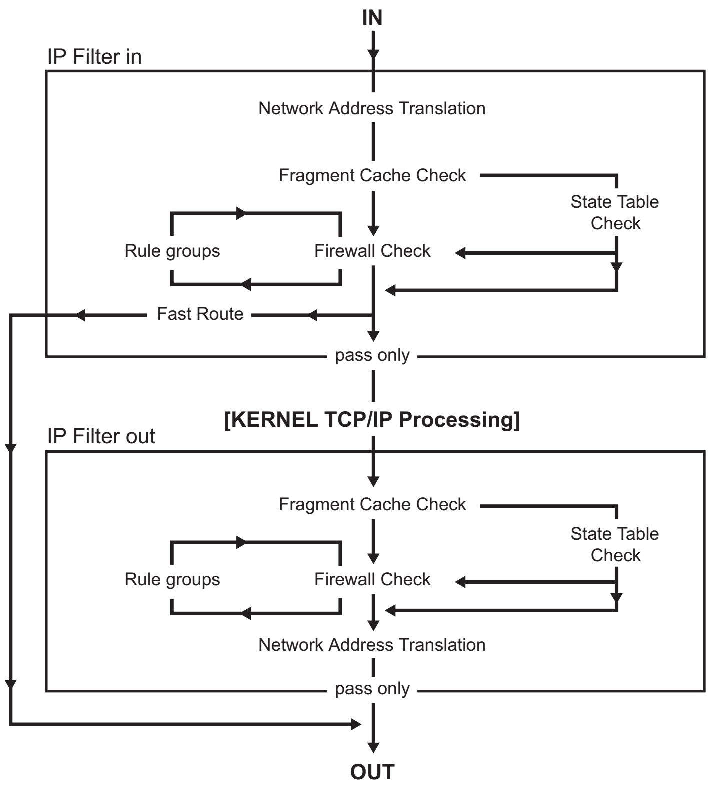


Figure 10-2IP Filter Packet Processing Sequence

- NAT work address translation is performed first on inbound packets, and last on outbound packets. Thus rules should be configured for real IP addresses, not for the NAT version of the IP address.
- The packet is checked against a matching rule, if an `auth` designations is in rules then a user-level application can be used to authenticate the packet, prompt for a password or use other authentications that the administrator might apply.
- Accounting for input and output rules, each time a rule match occurs, the byte count of the packet is added to the rule, allowing for cascading statistics to be collected.
- If the packet is designated as fast route, it bypasses the kernel entirely and is passed out through the indicated interface.
- The packet is checked against the state table and passes other operations, namely rule checks, if it is part of an existing stateful connection; otherwise, it is checked against the rule set.
- Outbound packets leave the kernel and progress through a similar sequence but in reverse, with accounting and NAT occurring after the rules check, assuming the packet is passed and the rule set does not block it.

Compatibility Issues

- IP Filter does not automatically disable or enable IP forwarding; the administrator must do this using the `ndd`(1M) or `routeadm`(1M) commands.
- IP Filter does not currently work with Internet Protocol version 6 (IPv6).
- Solaris 10 OS Zones – IP Filter works in a zone but it does not filter the loopback interfaces so it is not possible to filter between zones.
- Sun™ Cluster software – An `autopush`(1M) facility conflict results in the fact that IPFilter is not plumbed correctly and filtering is not enabled. No harm is done to the system; filtering is simply not enabled.
- IP Multipathing – IP Filter works with Internet Protocol Multipathing (IPMP) but not with the stateful packet filtering configuration.
- IP Tunnels – IP Filter does not work because you cannot apply the `autopush`(1M) facility to tunnels.

- PPP – IP Filter works if Point-to-Point Protocol (PPP) is plumbed, otherwise the `autopush`(1M) facility process must be done manually.

Configuring IP Filter

The IP Filter configuration files follow common UNIX® convention: the pound sign, #, is used for comments, a comment can be on the same line as a command or rule and extra white space is ignored but encouraged to aid in reading these files.

IP Pools

IP pools is a method to group IP addresses that IP Filter rules use. Configuration and use of IP pools is not required in most instances of a host-based firewall. The pools are configured in a manually edited configuration file, `/etc/ipf/ippool.conf`. This file is read at boot time.

Two types of pools are supported: a table type which can be a hash structure or a tree structure, and a group-map type. The `group-map` pool is used by using the `ipf` call function mechanism in the rule set, and the pools are applied to either the source or the destination address through the designated kernel function.

The following is an example of a pool entry:

```
table role = ipf type = tree number = 13
{ 10.1.1.1/32, 10.1.1.2/32, 192.168.1.0/24 };
```

This is a pool that includes the two IP addresses `10.1.1.1/32` and `10.1.1.2/32` and all of the addresses in the network `192.168.1.0/24`.

A rule that used the preceding pool entry might look like the following:

```
block in from pool/13 to any
```

Another example:

```
table role = ipf type = tree number = 100
{ 10.1.1.1/32, 245.2.0.0/16, !245.2.2.0/24 };
```

This pool includes the IP address `10.1.1.1/32` and network `254.2.0.0/16` but excludes the addresses in the `245.2.2.0/24` range.

A rule that used the preceding pool entry might look like the following:

```
pass in from pool/100 to any
IP pools can be configured as group-maps, for example:
group-map in role = ipf number = 1010
{ 192.168.1.1/32, group = 1020; 172.30.0.0/16, group = 1030; };
group-map out role = ipf number = 2010 group = 2020
{ 192.168.2.2/32; 172.16.0.0/16; 10.0.0.0/8, group = 2040; };
```

Rules that use a group-map pool use `fr_srcgrpmap` and `fr_dstgrpmap` functions, for example:

```
call now fr_srcgrpmap/1010 in all
call now fr_dstgrpmap/2010 out all
pass in all group 1020
block in all group 1030
pass out all group 2020
block out all group 2040
```

The IP pools must be loaded before the IP Filter rule sets is loaded. The `ippool`(1M) command is used to load the IP pools and the `-v` option adds verbose output:

```
# ippool -f /etc/ipf/ippool.conf -v
table role = ipf type = tree number = 100
             { 10.1.1.1/32, 245.2.0.0/16, ! 245.2.2.0/24 };
```

IP Filter rules

Rules are stored in a file, while the location and name are arbitrary rules that are typically found in the `/etc/ipf/ipf.conf` file. The administrator manually creates the rules file. If this file does not exist the firewall is not started at boot time.

Basic Rules

The basic rule syntax is as follows:

```
action direction packet
```

The simplest valid rules are as follows:

```
block in all
pass in all
log out all
count in all
```

The block in all rules is often entered as the last rule in the rule set to configure the security policy; all are denied unless specifically allowed.

Rules are read from the top of the list to the bottom. By default a packet is checked against all rules and the last rule that matched a packet is applied. This behavior is the opposite to that of most firewalls, and can cause mis-configured firewalls for those not familiar with IP Filter.

Source and Destination

Packets are matched based on IP addresses from a source to a destination.

The following is an example:

```
block in from 192.168.1.0/24 to any
```

This rule blocks inbound traffic from the network `192.168.1` to all destination address.

Two special keywords are use in packet matching: `all`, which indicates all source and destination addresses, and the `any` keyword that can be any source or any destination address.

Netmasks

As not all networks are formed with classical network boundaries, it is necessary to provide a mechanism to support Variable Length Subnet Masks (VLSM). This package provides several ways to do this. For example:

```
block in on eri0 from mynet/26 to any
block in on eri0 from mynet/255.255.255.192 to any
block in on eri0 from mynet mask 255.255.255.192 to any
block in on eri0 from mynet mask 0xffffffc0 to any
```

All these are valid and legal syntax.

The default netmask, when none is given, is `255.255.255.255` or `/32`.

To invert the match on a host name or network, include an ! before the name or number with no space between them.

Interfaces

To select the interface with which a packet is currently associated, either its destination as a result of route processing or where it has been received from, the `on` keyword is used. While not compulsory, it is recommended that each rule include the keyword for clarity. For example:

```
# drop all inbound packets from localhost coming from ethernet
block in on le0 from localhost to any
```

Protocols

A protocol can also be specified, for example:

```
block in proto icmp all
```

Since blocking all Internet Control Message Protocol (ICMP) traffic is not desirable, this rule can be fine-tuned:

```
block in proto icmp all icmp-type echo
```

Now this rule blocks only ICMP packets of type echo, thus preventing ping traffic.

The name of the protocol can be any valid name from the `/etc/protocols` file or a number. The one exception to this rule is Transmission Control Protocol (TCP)/User Datagram Format (UDP). If given in a rule set, the protocol name matches either of the two protocols. This is useful when setting up port restrictions. For example to prevent any packets destined for NFS, enter the following:

```
block in on eri0 proto tcp/udp from any to any port = 2049
```

ICMP

ICMP can be a source of a lot of trouble for Internet Connected networks. Blocking out all ICMP packets can be useful, but it disables some otherwise useful programs, such as ping. Filtering on ICMP type allows for pings (for example) to work. As an example:

```
# block all ICMP packets.
#
block in proto icmp all
#
# allow in ICMP echos and echo-replies.
#
```

```
pass in on le1 proto icmp from any to any icmp-type echo
pass in on le1 proto icmp from any to any icmp-type echorep
```

To specify an ICMP code, the numeric value must be used. So to block all port-unreachables, you enter the following:

```
#
# block all ICMP destination unreachable packets which are port-
unreachables
#
block in on le1 proto icmp from any to any icmp-type unreach code 3
```

The quick Keyword

The `quick` keyword changes the behavior of IP Filter and causes a matched rule to be applied with no other rules being checked. For instance:

```
block in quick on eri0 from 192.168.1.0/24 to any
```

If the packet matches this rule, then no other rule is checked and the packet is dropped.

By default every packet is checked against every rule until all rules have been checked, in which case the last match is applied, or until a packet matches a rule using the `quick` keyword.

Stateful Packet Filtering

Packet state filtering can be used for any TCP flow to short-cut later filtering. The *shortcuts* are kept in a table, with no alterations to the list of firewall rules. Subsequent packets, if a matching packet is found in the table, are not passed through the list making packet filtering much more efficient.

For TCP flows, the filter follows the ack/sequence numbers of packets and only allows packets through that fall inside the correct window.

```
# Keep state for all outgoing telnet connections
# and disallow all other TCP traffic.
#
pass out on le1 proto tcp from any to any port = telnet keep state
block out on le1 all
```

For UDP packets, packet exchanges are effectively stateless. However, if a packet is first sent out from a given port, a reply is usually expected in answer, in the reverse direction.

```
#
# allow UDP replies back from name servers
#
pass out on le1 proto udp from any to any port = domain keep state
```

The held UDP state is timed out in 60 seconds, as is the TCP state for entries added that do not have the `SYN` flag set. If an entry is created with the `SYN` flag set, any subsequent matching packet that does not have this flag set (that is, a `SYN-ACK`) causes it to be *timeless*. Actually the timeout defaults to five days, until either a `FIN` or `RST` flag is seen.

TCP Flags

Filtering on TCP flags is useful, but fraught with danger. Solaris OS IP Filter compares the flags present in each TCP packet, if asked, and matches if those present in the TCP packet are the same as in the IP Filter rule.

Flags are effective only for TCP filtering. Each of the letters possible represents one of the possible flags that can be set in the TCP header. The association is as follows:

- `F` – `FIN`
- `S` – `SYN`
- `R` – `RST`
- `P` – `PUSH`
- `A` – `ACK`
- `U` – `URG`

Some IP filtering and firewall packages allow you to filter out TCP packets that belong to an *established* connection. This is, simply put, filtering on packets which have the `ACK` bit set. The `ACK` bit is only set in packets transmitted during the life cycle of a TCP connection. This flag must be present from either end for data to be transferred. If you were using a rule worded something like the following,

```
allow proto tcp 10.1.0.0 255.255.0.0 port = 23 10.2.0.0 255.255.0.0
established
```

it could be rewritten as follows:

```
pass in proto tcp 10.1.0.0/16 port = 23 10.2.0.0/16 flags A/A
pass out proto tcp 10.1.0.0/16 port = 23 10.2.0.0/16 flags A/A
```

A more useful flag to filter on, for TCP connections, is the SYN flag. This is only set during the initial stages of connection negotiation, and for the very first packet of a new TCP connection, it is the only flag set. At all other times, an ACK or maybe even an URG/PUSH flag can be set. So, if you want to stop connections being made to the internal network (10.1.0.0) from the outside network, you might do something like the following:

```
# block incoming connection requests to my internal network
# from the big bad internet.
#
block in on eri0 proto tcp from any to 10.1.0.0/16 flags S/SA
```

If you wanted to block the replies to this (the SYN-ACK flags), then you might do:

```
block out on le0 proto tcp from 10.1.0.0 to any flags SA/SA
```

where SA represents the SYN-ACK flags both being set.

The flags after the / represent the TCP flag mask, indicating which bits of the TCP flags you are interested in checking. When using the SYN bit in a check, you should specify a mask to ensure that a packet with SYN and URG flags set cannot defeat your filter.

Flag S actually equates to `flags S/AUPRFS` and matches against only the SYN packet out of all six possible flags, while `flags S/SA` allow packets that may or may not have the URG, PSH, FIN, or RST flags set.

Fragments

IP fragments are bad news, in general. A recent study showed that IP fragments can pose a large threat to Internet firewalls, *if* rules are used that rely on data that might be distributed across fragments. To Solaris OS IP Filter, the threat is that the TCP flags field of the TCP packet might be in the second or third fragment, or possibly be believed to be in the first, when the field is actually in the second or third fragment.

It is possible to get rid of all IP fragments as follows:

```
    block in all with frag
```

The problem is that fragments can actually be a non-malicious. The really malicious ones can be grouped under the term *short fragments* and can be filtered, for instance, to get rid of all short IP fragments, too small for valid comparison.

```
    block in proto tcp all with short
```

IP Options

IP options have a bad name for being a general security threat. They can be of some use to programs such as traceroute, but many find this usefulness not worth the risk.

Filtering on IP options can be achieved two ways. The first is by naming them collectively and is done as follows:

```
#
# drop and log any IP packets with options set in them.
#
block in log all with ipopts
#
```

The second way is to actually list the names of the options you wish to filter.

```
#
# drop any loose (lsrr) or strict (ssrr) source routing options
#
block in quick all with opt lsrr
block in quick all with opt ssrr
```

Note – Options are matched explicitly, so if you listed the `lsrr` and `ssrr` options, the filter would only match packets with both options set.

It is also possible to select packets which *do not* have various options present in the packet header. For example, to allow telnet connections without any IP options present, the following would be done:

```
#
# Allow anyone to telnet in so long as they don't use IP options.
#
pass in proto tcp from any to any port = 23 with no ipopts
#
# Allow packets with strict source routing and no loose source routing
#
```

```
pass in from any to any with opt ssrr not opt lsrr
```

Filtering by Port

Filtering by port number only works with the TCP and UDP IP protocols. When specifying port numbers, either the number or the service name from the `/etc/services` file can be used. If the `proto` field is used in a filter rule, it is used in conjunction with the port name in determining the port number.

Table 10-1 shows the possible operands available for use with port numbers:

Table 10-1 Port Number Operands

Operand	Alias	Parameters	Result
<	lt	port#	True if port is less than given value
>	gt	port#	True if port is greater than given value
=	eq	port#	True if port is equal to the given value
!=	ne	port#	True if port is not equal to the given value
<=	le	port#	True if port is less than or equal to given value
=>	ge	port#	True if port is greater than or equal to given value

For example:

```
#
# allow any TCP packets from the same subnet
# as foo is on through to host
# 10.1.1.2 if they are destined for port 6667.
#
pass in proto tcp from fubar/24 to 10.1.1.2/32 port = 6667
#
# allow in UDP packets which are NOT from port 53
# and are destined for localhost
#
pass in proto udp from fubar port != 53 to localhost
```

Two range comparisons are also possible. The expression syntax is as follows:

- `port1# <> port2#` – True if port is less than port1 or greater than port2
- `port1# >< port2#` – True if port is greater than port1 and less than port2

In neither case, when the port number is equal to one of those given, does it match. For instance:

```
#
# block anything trying to get to X terminal ports, X:0 to X:9
#
block in proto tcp from any to any port 5999 >< 6010
#
# allow any connections to be made, except to BSD print/r-services
# this will also protect syslog.
#
block in proto tcp/udp all
pass in proto tcp/udp from any to any port 512 <> 515
```

The last one in the preceding could just as easily be done in the reverse fashion: allowing everything through and blocking only a small range. However, note that the port numbers differ due to the difference in the way they are compared.

```
#
# allow any connections to be made, except to BSD print/r-services
# this will also protect syslog.
#
pass in proto tcp/udp all
block in proto tcp/udp from any to any port 511 >< 516
```

Return Messages

To provide feedback to people trying to send packets through your filter which you wish to disallow, you can send back either an ICMP error (Destination Unreachable) or, if they are sending a TCP packet, a TCP RST (Reset).

What is the difference? Transmission Control Protocol/Internet Protocol (TCP/IP) stacks take longer to pass the ICMP errors back, through to the application, as they can often be due to temporary problems (network was unplugged for a second) and it is *incorrect* to shut down a connection for this reason. Others go to the other extreme and shut down all connections between the two hosts for which the ICMP error is received. The TCP RST, however, is for only *one* connection, cannot be used for more than one, and causes the connection to immediately shut down. So, for example, if you are blocking port `113`, and set up a rule to return a TCP RST rather than nothing or an ICMP packet, you do not experience any delay if the other end was attempting to make a connection to say, an `identd` daemon service.

Some examples are as follows:

```
#
# block all incoming TCP connections but send
#back a TCP-RST for ones to the ident port
#
block in proto tcp from any to any flags S/SA
block return-rst in quick proto tcp from any to any port = 113 flags S/SA
#
# block all inbound UDP packets and send back an ICMP error.
#
block return-icmp in proto udp from any to any
```

When returning ICMP packets, it is also possible to specify the type of ICMP error returned. This was requested so that traceroute traces could be forced to end elegantly. To do this, the requested ICMP unreachable code is placed in brackets following the *return-icmp* directive:

```
#
# block all inbound UDP packets and send back an ICMP error.
#
block return-icmp (3) in proto udp from any to any port > 30000
block return-icmp (port-unr) in proto udp from any to any port > 30000
```

Those two examples are equivalent, and return a ICMP port unreachable error packet in response to any UDP packet received destined for a port greater than 30,000.

Filtering IP Security Classes

For users who have packets which contain IP security bits, filtering on the defined classes and authority levels is supported. Currently, filtering on 16-bit authority flags is not supported.

As with `ipopts` and other IP options, it is possible to say that the packet only matches if a certain class is not present.

The following are examples of filtering on IP security options:

```
#
# drop all packets without IP security options
#
block in all with no opt sec
#
# only allow packets in and out on le0 which are top secret
#
block out on le1 all
pass out on le1 all with opt sec-class topsecret
block in on le1 all
pass in on le1 all with opt sec-class topsecret
```

Rule Groups

To aid in making rule processing more efficient, it is possible to setup rule *groups*. By default, all rules are in group 0 and all other groups have it as their ultimate parent. To start a new group, a rule includes a *head* statement. The rules in the group have a group statement.

```
Block in quick on hme0 all head 10
block in quick on hme0 from 192.0.2.0/24 to any group 10
block in quick on hme0 from 192.168.0.0/24 to any group 10
pass in on hme0 all group 10
block in quick on hme1 all head 11
block in quick on hme1 from 192.0.2.0/24 to any group 11
block in quick on hme1 from 192.168.0.0/24 to any group 11
pass in on hme1 all group 11
```

When a packet matched the rule with the `head` keyword, it then processes rules with the appropriate `group` keyword. This increases efficiency because of the fewer number of rules checked against the packet.

Depending on your situation, it may be prudent to group your rule by protocol, or various machines, or net blocks, or whatever makes it flow smoothly.

The ipf(1M) Command

The ipf command is used to load or remove a packet filter rule set to or from the kernel module. When adding a rule set, the command typically reads the rule set from a file.

Each rule that the ipf command processes is added to the kernel's internal lists if there are no parsing problems. Rules are added to the end of the internal lists, matching the order in which they appear when given to the ipf command.

The command's use is restricted through access to the /dev/ipf file. The default permissions of the /dev/ipf file require the ipf command to be run as root user for all operations.

The following are examples of using the ipf command.

To load a rule set from file, the -Fa options force the current rule set, if any, to be flushed before loading the rule set in the file /etc/ipf/ipf.conf:

```
entropy#> ipf -Fa -f /etc/ipf/ipf.conf
```

Using the -v option causes more verbose output and lists the rules as they are loaded:

```
entropy#> ipf -v -Fa -f /etc/ipf/ipf.conf
remove flags IO (49152)
removed 12 filter rules
block in quick on eri0(!) from 172.16.0.0/12 to any
block in quick on eri0(!) from 10.0.0.0/8 to any
block in quick on eri0(!) from 127.0.0.0/8 to any
block in quick on eri0(!) from 0.0.0.0/8 to any
block in quick on eri0(!) from 169.254.0.0/16 to any
block in quick on eri0(!) from 192.0.2.0/24 to any
block in quick on eri0(!) from 204.152.64.0/23 to any
block in quick on eri0(!) from 224.0.0.0/3 to any
block in log quick on eri0(!) from 192.168.100.0/24 to any
pass in quick on eri0(!) proto tcp/udp from 192.168.201.0/24 to
192.168.201.21/32 keep state
pass in quick on eri1(!) all
block in all
```

The ipfstat(1M) command can be used to display the rule set while it is loaded in the kernel module.

```
Entropy#> **ipfstat -ion**
empty list for ipfilter(out)
@1 block in quick on eri0 from 172.16.0.0/12 to any
@2 block in quick on eri0 from 10.0.0.0/8 to any
@3 block in quick on eri0 from 127.0.0.0/8 to any
@4 block in quick on eri0 from 0.0.0.0/8 to any
@5 block in quick on eri0 from 169.254.0.0/16 to any
@6 block in quick on eri0 from 192.0.2.0/24 to any
@7 block in quick on eri0 from 204.152.64.0/23 to any
@8 block in quick on eri0 from 224.0.0.0/3 to any
@9 block in log quick on eri0 from 192.168.100.0/24 to any
@10 pass in quick on eri0 proto tcp/udp from 192.168.201.0/24 to
192.168.201.21/32 keep state
@11 pass in quick on eri1 all
@12 block in all
```

In the previous example output, the `-ion` options display the following:

- Filter list used for the input side (`i`) of the kernel IP processing
- Filter list used for the output side (`o`) of the kernel IP processing
- The rule number (`n`) for each rule as it is printed.

Logging

Up to this point, all blocked and passed packets have been silently blocked and silently passed. Usually you want generate logging information to determine if your system is being attacked or to see what packet was blocked during troubleshooting.

To enable logging with a rule, use the `log` keyword. As an example:

```
block in log quick on eri0 from 192.168.100.0/24 to any
```

It is important to note that the presence of the `log` keyword only ensures that the packet is available to the ipfilter logging device; `/dev/ipl`. To actually see this log information, one must be running the `ipmon`(1M) utility, or some other utility that reads from the `/dev/ipl` device. The typical usage of the `log` keyword is coupled with `ipmon -s` command to log the information to the `syslogd`(1M) logging daemon. You can control the logging behavior of the `syslog`(3C) function by using log-level keywords in rules such as the following:

```
block in log level auth.info quick on eri0 from 207.92.207.0/24 to any
block in log level auth.alert quick on eri0 proto tcp from any to
207.92.207.0/24 port = 21
```

You can also tailor what information is being logged. For example, you might not be interested that someone attempted to probe your telnet port 500 times, but you might be interested that the person probed you once. You can use the `log` keyword with the `first` keyword to only log the first example of a packet. Of course, the notion of *first-ness* only applies to packets in a specific session, and for the typical blocked packet, you will be hard-pressed to encounter situations where this does what you expect. However, if used in conjunction with the pass and keep state, this can be a valuable keyword for keeping tabs on traffic.

Another useful thing you can do with the logs is to track interesting pieces of the packet besides the header information normally being logged. Solaris OS IP Filter gives you the first 128 bytes of the packet if you use the `log` keyword with the `body` keyword. You should limit the use of body logging, as it makes your logs very verbose, but for certain applications, it is often handy to be able to go back and take a look at the packet, or to send this data to another application that can examine it further.

Logging Packets to the Network

Logging packets to the network devices is supported for both packets being passed through the filter and those being blocked. For packets being passed on, the `dup-to` keyword must be used, but for packets being blocked, either the `to` keyword (which is more efficient) or the `dup-to` keyword can be used. The `to` keyword causes the packet to be redirected.

A good example of a use for this would be to implement an intrusion-detection network. For starters, you might want to hide the presence of your intrusion detection systems from your real network so that you can keep the systems from being detected.

For example, to send a copy of everything going out the `qfe0` interface off to your network on `qfe1`, you use the following rule in your filter list:

```
pass out on qfe0 dup-to qfe1 from any to any
```

If you do not care about passing the packet to its normal destination and the packet was going to block it anyway, you can just use the `to` keyword to push this packet past the normal routing table and force it to go out a different interface than it normally would.

You might also need to send the packet directly to a specific IP address on your network instead of just making a copy of the packet, sending it out the interface and hoping for the best. To do this, you modify your rule slightly:

```
# Log all short TCP packets to qfe1, with 192.168.100.50
# as the intended destination for the packet.
#
block in quick to qfe1:192.168.100.50/32 proto tcp all with short
#
# Log all connection attempts for TCP
#
pass in quick on qfe0 dup-to qfe1:192.168.100.50/32 proto tcp all flags
S/SA
```

Be warned that this method alters the copied packet's destination address, and might thus destroy the usefulness of the log. For this reason, it is recommended that you only use the known address method of logging when you can be certain that the address that you are logging to corresponds in some way to what you are logging for. For example, do not use `192.168.254.2` for logging for both your web server and your mail server, since you will have a hard time later trying to figure out which system was the target of a specific set of packets.

This technique can be used quite effectively if you treat an IP address on your drop-safe network in much the same way that you would treat a multicast group on the real Internet. For example `192.168.254.2` could be the channel for your Hypertext Transfer Protocol (HTTP) traffic analysis system, `23.23.23.23` could be your channel for telnet sessions, and so on. You do not even need to actually have this address set as an address or alias on any of your analysis systems. Normally, your Solaris OS IP Filter machine would need to apply the Address Resolution Protocol (ARP) for the new destination address, using `dup-to eri1:192.168.254.2` syntax style, of course. However, you can avoid that issue by creating a static ARP entry for this channel on your Solaris OS IP Filter system.

However, `dup-to eri1` syntax is generally all that is required to get a new copy of the packet over to your drop-safe network for logging and examination.

Additionally the `count` keyword can be used. The `count` keyword causes the packet to be included in the accounting statistics that the filter keeps, and has no effect on whether the packet is allowed through the filter. These statistics are viewable with the `ipfstat`(1M) command using the `-a` option, which displays the accounting filter list and shows bytes counted against each rule.

The `ipmon`(1M) Command

The `ipmon` command opens the `/dev/ipl` file for reading and awaits data to be saved from the packet filter. The binary data read from the device is reprinted in human readable form. However, IP addresses are not mapped back to host names, nor are ports mapped back to service names. The output goes to standard output, by default, or a filename, if specified on the command line. Should the `-s` option be used, output is sent instead to the `syslogd`(1M) logging deamon. Messages sent by means of `syslogd` logging deamon have the day, month, and year removed from the message, but the time, including microseconds, as recorded in the log, is still included.

```
entropy#> ipmon -s
```

Starts logging the `/dev/ipl` file to `syslog`. The default facility is `local0`.

To watch the state table in action, `ipmon -o S` would show this:

```
entropy#> ipmon -o S
01/08/2004 15:58:57.836053 STATE:NEW 100.100.100.1,53 -> 20.20.20.15,53
PR udp
01/08/2004 15:58:58.030815 STATE:NEW 20.20.20.15,123 -> 128.167.1.69,123
PR udp
01/08/2004 15:59:18.032174 STATE:NEW 20.20.20.15,123 -> 128.173.14.71,123
PR udp
01/08/2004 15:59:24.570107 STATE:EXPIRE 100.100.100.1,53 ->
20.20.20.15,53 PR udp \
 Pkts in 2 Bytes in 128 Pkts out 2 Bytes out 128
01/08/2004 16:03:51.754867 STATE:NEW 20.20.20.13,1019 ->
100.100.100.10,22 PR tcp
01/08/2004 16:04:03.070127 STATE:EXPIRE 20.20.20.13,1019 ->
100.100.100.10,22 PR tcp \
 Pkts in 31 Bytes in 2282 Pkts out 32 Bytes out 2322
```

NAT and PAT Rules

NAT rules are stored in their own file. The location and name is arbitrary, but the system reads NAT rules from the `/etc/ipf/ipnat.conf` file at boot time. The administrator manually edits this file.

The basic NAT keyword is `map`:

```
map hme0 10.0.0.0/8 -> 192.0.2.41/32
```

This rule tells Solaris OS IP Filter to change the source address of all packets passing out `hme0` with a source address within the `10.0.0.0/8` net block to a source address of `192.0.2.41`. When Solaris OS IP Filter does so, it automatically maintains a session state table of all packets for which it has rewritten addresses; using this table, it watches incoming packets that have a destination address `192.0.2.41` and rewrites as necessary, using the appropriate `10.0.0.0/8` address, any packets that are actually destined for an internal host.

The preceding rule does not remap ports, which can be a problem since multiple hosts in the `10.0.0.0/8` net block might simultaneously try to send outbound traffic from the same ephemeral port on the local host, causing a collision at the firewall. To prevent this, the rule can be modified using PAT:

```
map hme0 10.0.0.0/8 -> 192.0.2.41/32 portmap tcp/udp 20000:30000
```

The `portmap` keyword in the example causes all outbound packets to have the source ports remapped to a free port between `20000` and `30000`.

Besides mapping many IP addresses to one address, the `map` keyword can also be used to map many IP addresses to many IP addresses:

```
map hme0 10.0.0.0/8 -> 192.0.2.0/24 portmap tcp/udp 20000:30000
```

This might not work well because multiple sessions, such as HTTP, appear to have different source addresses within the `192.0.2.0/24` net block. To correct this, enter the following:

```
map-block hme0 10.0.0.0/8 -> 192.0.2.0/24
```

The `map-block` keyword causes session states to be tracked to ensure that all simultaneous sessions from the same internal address get rewritten with the same external address. The `map-block` keyword also automatically corrects port issues, so the `portmap` keyword is unnecessary.

A static one-to-one mapping can also be created. This is used when an internal host must have a connection that an external host initiates.

```
bimap hme0 10.0.0.4/32 -> 192.0.2.41/32
```

You can use the redirection facilities of NAT to instruct it to remap any connections destined for 192.0.2.41:80 to really point to 10.0.0.5:8000. This uses the `rdr` keyword:

```
rdr hme0 192.0.2.41/32 port 80 -> 10.0.0.5/32 port 8000
```

The `ipnat(1M)` Command

The `ipnat` command is used to add and remove NAT rules to and from the `ipf` kernel module. These rules are appended to the existing set of rules. Examples of the `ipnat` command:

```
entropy#> ipnat -C -f /etc/ipf/ipnat.conf
0 entries flushed from NAT list
```

The `-C` option causes any previous NAT rule set to be cleared and the new rule set is read in from the file `/etc/ipf/ipnat.conf`. The next command example uses the `-l` option to display the current rule set and active connections:

```
entropy#> ipnat -l
List of active MAP/Redirect filters:
map eri0 192.168.100.0/24 -> 192.168.201.12/32
bimap eri0 192.168.201.100/32 -> 192.168.100.50/32

List of active sessions:
```

Troubleshooting Solaris OS IP Filter

The `ipf`(1M) and `ipnat`(1M) commands both support a `-d` option for debugging. However, the output seems more suited to debugging a program than to troubleshooting. Both commands report syntax errors in the configuration file when you attempt to load the file. This can be useful to test the `ipf` command or NAT rule set before using it. Here is an example of using `ipnat` with the `-d` option while loading a rules set. The `ipf`(1M) command output is similar:

```
entropy#> ipnat -Cd -f /etc/ipf/ipnat.conf
2 entries flushed from NAT list
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 01 00 00 00 00
c0 a8 64 00 00 00 00 00 00 00 00 00 00 00 00 00
ff ff ff 00 00 00 00 00 00 00 00 00 00 00 00 00
c0 a8 c9 0c 00 00 00 00 00 00 00 00 00 00 00 00
ff ff ff ff 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 65 72 69 30
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 65 72 69 30
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 03 00 00 00 00
c0 a8 c9 64 00 00 00 00 00 00 00 00 00 00 00 00
ff ff ff ff 00 00 00 00 00 00 00 00 00 00 00 00
c0 a8 64 32 00 00 00 00 00 00 00 00 00 00 00 00
ff ff ff ff 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 65 72 69 30
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 65 72 69 30
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

This is an example of using the ipf(1M) command with a rule set that contains a rule with errors:

```
entropy#> ipf -Fa -f /etc/ipf/ipf.conf
syntax error error at "from", line 4
In this case the rule set file contained:
block in     quick on eri0 from 172.16.0.0/12 to any
block in     quick on eri0 from 10.0.0.0/8 to any
block in     quick on eri0 from 127.0.0.0/8 to any
block in     quick on from 0.0.0.0/8 to any
block in     quick on eri0 from 169.254.0.0/16 to any
...
```

The ipf(1M) command can use the -T option to list and change a number of Solaris OS IP Filter tunables. However, their use is currently undocumented:

```
entropy#> ipf -T list
fr_flags        min 0   max 0xffffffff  current 0
fr_active       min 0   max 0   current 0
fr_control_forwarding   min 0   max 0x1 current 0
fr_update_ipid  min 0   max 0x1 current 0
fr_chksrc       min 0   max 0x1 current 0
fr_pass min 0   max 0xffffffff  current 134217730
fr_unreach      min 0   max 0xff        current 13
fr_tcpidletimeout       min 0x1 max 0x7fffffff  current 864000
fr_tcpclosewait min 0x1 max 0x7fffffff  current 240
fr_tcplastack   min 0x1 max 0x7fffffff  current 240
fr_tcptimeout   min 0x1 max 0x7fffffff  current 240
fr_tcpclosed    min 0x1 max 0x7fffffff  current 120
fr_tcphalfclosed        min 0x1 max 0x7fffffff  current 14400
fr_udptimeout   min 0x1 max 0x7fffffff  current 240
fr_udpacktimeout        min 0x1 max 0x7fffffff  current 24
fr_icmptimeout  min 0x1 max 0x7fffffff  current 120
fr_icmpacktimeout       min 0x1 max 0x7fffffff  current 12
fr_statemax     min 0x1 max 0x7fffffff  current 4013
fr_statesize    min 0x1 max 0x7fffffff  current 5737
fr_state_lock   min 0   max 0x1 current 0
fr_state_maxbucket      min 0x1 max 0x7fffffff  current 26
fr_state_maxbucket_reset        min 0   max 0x1 current 1
ipstate_logging min 0   max 0x1 current 1
fr_nat_lock     min 0   max 0x1 current 0
ipf_nattable_sz min 0x1 max 0x7fffffff  current 16383
ipf_natrules_sz min 0x1 max 0x7fffffff  current 127
```

```
ipf_rdrrules_sz min 0x1 max 0x7fffffff  current 127
ipf_hostmap_sz  min 0x1 max 0x7fffffff  current 8191
fr_nat_maxbucket        min 0x1 max 0x7fffffff  current 28
fr_nat_maxbucket_reset  min 0   max 0x1 current 1
nat_logging     min 0   max 0x1 current 1
fr_defnatage    min 0x1 max 0x7fffffff  current 1200
fr_defnaticmpage        min 0x1 max 0x7fffffff  current 6
ipfr_size       min 0x1 max 0x7fffffff  current 257
fr_ipfrttl      min 0x1 max 0x7fffffff  current 120
ipl_suppress    min 0   max 0x1 current 1
ipl_buffer_sz   min 0   max 0   current 0
ipl_logmax      min 0   max 0x7fffffff  current 7
ipl_logall      min 0   max 0x1 current 0
```

The output is the parameter followed by its minimum setting, maximum setting, and current setting.

Some common problems that occur with NAT and filtering involve timeouts for idle or partially closed connections and running out of table space. For the former, the following entries in the `/etc/system` file might prove useful (note that the timers run *two ticks to a second*, so the 172800 ticks for the `ipf:fr_tcpidletimeout` variable listed in the following translates to 24 hours). A reboot is required to apply the changes.

```
* ipf: adjust the default tcp timeouts downward so that
* idle (dead) and half closed states get killed off
quicker.
set ipf:fr_tcpidletimeout = 172800
set ipf:fr_tcphalfclosed = 7200
```

On servers that experience a large amount of traffic where the state is kept as part of the rule sets, it is sometimes necessary to increase the state table size to accommodate the load. It is also wise to add `flags S` to each rule where state is kept, so that only the first packet of the session is recorded. To adjust the state table size so that there are enough buckets for large servers, add the following to the `/etc/system` file and reboot the machine:

```
* ipf: adjust the state table sizes so we have enough buckets.
* IPSTATE_MAX (=fr_statemax) should be ~70% of IPSTATE_SIZE
* IPSTATE_SIZE (=fr_statesize) has to be a prime number
set ipf:fr_statemax = 7000
set ipf:fr_statesize = 10009
```

To increase the table size for NAT, add the following to the `/etc/system` file and reboot the machine:

```
* ipf: adjust the NAT table sizes so we have enough buckets.
* generally you have fewer than 127 rules in ipnat.conf
* so no need to waste memory for more.
set ipf:ipf_nattable_sz = 10009
set ipf:ipf_natrules_sz = 127
set ipf:ipf_rdrrules_sz = 127
```

Troubleshooting With the `ipmon(1M)` Command

The `ipmon` command can be useful for debugging Solaris OS IP Filter. It can also show which packets have been logged. For example, when using state, you often encounter packets like this:

```
entropy# ipmon -o I
15:57:33.803147 ppp0 @0:2 b 100.100.100.103,443 -> 20.20.20.10,4923 PR
tcp len 20 1488 -A
```

The first field, `15:57:33.803147`, is a time stamp. The second field, `ppp0`, is the interface on which this event happened. The third field, `@0:2`, is the rule that caused the event to happen: rule `2` in rule group `0`. The fourth field, the little `b`, says that this packet was blocked; you generally ignore this unless you are logging passed packets as well, which would require a little `p` instead. The fifth and sixth fields indicate the origin and destination of this packet. The seventh and eighth fields, `PR` and `tcp` respectively, indicate the protocol. The ninth field, `len 20 1488`, indicates the size of the packet: `20` is the length of the IP header and `1488` is the length of the IP packet. The last part, `-A` in this case, tells you the flags that were on the packet. This one was an `ACK` packet.

Due to the often lagging nature of the Internet, packets might be regenerated. You might get two copies of the same packet. Your state rule, which tracks sequence numbers, will have already seen this packet, so the rule assumes that the packet is part of a different connection. Eventually this packet encounters a real rule and must be dealt with. The last packet of a session being closed might be logged, because the keep state code has already torn down the connection before the last packet has had a chance to make it to your firewall. This is normal. The following is another example packet that might be logged:

```
12:46:12.470951 iprb0 @0:1 S 20.20.20.254 -> 255.255.255.255 PR icmp len
20 9216 icmp 9/0
```

This is an ICMP router discovery broadcast. You can tell by the ICMP type 9/0.

Finally, the `ipmon` command also lets you look at the NAT table in action:

```
entropy# ipmon -o N
01/08/1999 05:30:02.466114 @2 NAT:RDR 20.20.20.253,113 <- -> \
 20.20.20.253,113 [100.100.100.13,45816]
01/08/1999 05:30:31.990037 @2 NAT:EXPIRE 20.20.20.253,113 <- -> \
 20.20.20.253,113 [100.100.100.13,45816] Pkt
```

This would be a redirection to an `identd` daemon that provides ident service for the hosts behind your NAT, since the hosts typically cannot provide this service for themselves with ordinary NAT.

Troubleshooting With the `ipfstat(1M)` Command

In its simplest form, the `ipfstat` command displays a table of interesting data about how your firewall is performing, such as how many packets have been passed or blocked, if the packets were logged or not, how many state entries have been made, and so on. The following is an example of something you might see from running the tool:

```
entropy# ipfstat
input packets:   blocked 99286 passed 1255609 nomatch 14686 counted 0
output packets:   blocked 4200 passed 1284345 nomatch 14687 counted 0
input packets logged: blocked 99286 passed 0
output packets logged: blocked 0 passed 0
packets logged:    input 0 output 0
log failures:     input 3898 output 0
fragment state(in):   kept 0 lost 0
fragment state(out):  kept 0 lost 0
packet state(in):    kept 169364   lost 0
packet state(out):   kept 431395   lost 0
ICMP replies:  0    TCP RSTs sent: 0
Result cache hits(in): 1215208 (out): 1098963
IN Pullups succeeded:  2    failed: 0
OUT Pullups succeeded: 0    failed: 0
Fastroute successes:  0    failures:    0
TCP cksum fails(in):  0    (out): 0
Packet log flags set: (0)
      none
```

The `ipfstat` command can also show you your current rule list. Using the `-i` or the `-o` flag shows the currently loaded rules for in or out, respectively. Adding a `-h` flag to this provides more useful information at the same time by showing you a *hit count* on each rule. For example:

```
entropy# ipfstat -ho
```

```
2451423 pass out on iprb0 from any to any
354727 block out on ppp0 from any to any
430918 pass out quick on ppp0 proto tcp/udp from 20.20.20.0/24 to any
keep state keep frags
```

From this, you can see that something abnormal may be going on, since you have a lot of blocked packets out-bound, even with a very permissive pass out rule. Something here may warrant further investigation, or it may be functioning perfectly by design. The `ipfstat` command cannot tell you if your rules are right or wrong, but it can tell you what is happening because of your rules.

To further debug your rules, you may want to use the `-n` flag, which shows the rule number next to each rule:

```
entropy# ipfstat -on
@1 pass out on iprb0 from any to any
@2 block out on ppp0 from any to any
@3 pass out quick on ppp0 proto tcp/udp from 20.20.20.0/24 to any keep
state keep frags
```

Another piece of interesting information that the `ipfstat` command can provide is information about the state table. This is done with the `-s` flag:

```
entropy# ipfstat -s
281458 TCP
319349 UDP
0 ICMP
19780145 hits
5723648 misses
0 maximum
0 no memory
1 active
319349 expired
281419 closed
100.100.100.1 -> 20.20.20.1 ttl 864000 pass 20490 pr 6 state 4/4
pkts 196 bytes 17394 987 -> 22 585538471:2213225493 16592:16500
pass in log quick keep state
pkt_flags & b = 2,       pkt_options & ffffffff = 0
pkt_security & ffff = 0, pkt_auth & ffff = 0
```

Here you see that you have one state entry for a TCP connection. The output varies slightly from version to version, but the basic information is the same. You can see in this connection that you have a fully established connection, which the 4/4 state represents. You can see that the state entry has a time to live (TTL) of 240 hours, which is an absurdly long time, but is the default for an established TCP connection. This TTL counter is decremented every second that the state entry is not used, and finally results in the connection being purged if it has been left idle. The TTL is also reset to 864000 whenever the state is used, ensuring that the entry does not time out while it is being actively used.

You can also see that 196 packets were passed consisting of about 17 kilobytes (Kbytes) worth of data over this connection. You can see the ports for both endpoints, in this case 987 and 22, which means that this state entry represents a connection from `100.100.100.1` port `987` to `20.20.20.1` port `22`.

The really big numbers in the second line are the TCP sequence numbers for this connection, which helps to ensure that someone cannot easily inject a forged packet into your session. The TCP window is also shown. The third line is a synopsis of the implicit rule that the keep state code generated, showing that this connection is an inbound connection.

Finally, the `ipfstat -t` command provides a constantly updated state summary that allows an administrator to monitor the setup and tear-down of state entries as they happen. Output is similar to the familiar `top` utility.

Using the `ipftest` Utility

The `ipftest` utility is in the `/usr/lib/ipf` directory.

The `ipftest` command is provided to test a set of filter rules without having to put them in place, in operation, and proceed to test their effectiveness. The hope is that this minimizes disruptions in providing a secure IP environment.

The `ipftest` command parses any standard rule set for use with the `ipf` command, applies input, and returns output as the result. However, the `ipftest` command returns one of three values for packets passed through the filter: `pass`, `block` or `nomatch`. This is to give operators a better idea of what is happening with packets passing through their filter rule set.

The `ipftest` command takes a number of options:

- `-v` – Verbose mode. This provide more information about which parts of rule-matching the input packet passes and fails.
- `-d` – Turn on filter-rule debugging. Currently, this only shows you what caused the rule to not match in the IP header checking (for example, addresses/netmasks).
- `-b` – Cause the output to be a brief summary (one-word) of the result of passing the packet through the filter; either `pass`, `block` or `nomatch`. This is used in the regression testing.
- `-I` *`interface`* – Set the interface name (used in rule matching) to be the name supplied. This is useful with the `-P`, `-S`, `-T` and `-E` options, where it is not otherwise possible to associate a packet with an interface. Normal *text packets* can override this setting.
- `-P` – The input file `-i` specifies is a binary file produced using the `libpcap` interface (`tcpdump` utility version 3). Packets are read from this file as being input (for rule purposes). An interface can be specified using `-I`.
- `-S` – The input file is to be in *snoop* format (see Request For Comments [RFC] 1761). Packets are read from this file and used as input from any interface. Currently, this might be the most useful input type.
- `-T` – The input file is to be text output from the `tcpdump` utility. The text formats which are currently supported are those that result from the following `tcpdump` utility option combinations:
  - `tcpdump -n`
  - `tcpdump -nq`
  - `tcpdump -nqt`
  - `tcpdump -nqtt`
  - `tcpdump -nqte`
- `-H` – The input file is to be hex digits, representing the binary makeup of the packet. No length correction is made, if an incorrect length is put in the IP header.
- `-X` – The input file is composed of text descriptions of IP packets.
- `-E` – The input file is to be text output from the `etherfind` utility. The text formats which are currently supported are those that result from the following `etherfind` utility option combinations:
  - `etherfind -n`
  - `etherfind -n -t`

- `-i` *`filename`* – Specify the filename from which to take input. The default is `STDIN`.
- `-r` *`filename`* – Specify the filename from which to read filter rules.

The `ipftest` command can take the output from a number of packet capture utilities, including `snoop`(1M), and use it as input to test the rule set.

The command also accepts formatted input from the command line to do some basic testing. For example:

```
entropy#> /usr/lib/ipf/ipftest -r /etc/ipf/ipf.conf
```

The preceding command tests the syntax of the rules file indicated and then waits for user input. The user enters the following:

```
in on eri0 tcp 169.254.1.34,12345 192.168.100.12,23
```

The utility generates the following output and waits for the next user entry:

```
input: in on eri0 tcp 169.254.1.34,12345 192.168.100.12,23
block ip 40(20) 6 169.254.1.34,12345 > 192.168.100.12,23
```

The output indicates that the packet was blocked, but not much else. You can get better output by restarting the command with other options: the `-v` verbose options:

```
entropy#> /usr/lib/ipf/ipftest -v -r /etc/ipf/ipf.conf
block in quick on eri0(!) from 172.16.0.0/12 to any
block in quick on eri0(!) from 10.0.0.0/8 to any
block in quick on eri0(!) from 127.0.0.0/8 to any
block in quick on eri0(!) from 0.0.0.0/8 to any
block in quick on eri0(!) from 169.254.0.0/16 to any
block in quick on eri0(!) from 192.0.2.0/24 to any
block in quick on eri0(!) from 204.152.64.0/23 to any
block in quick on eri0(!) from 224.0.0.0/3 to any
block in log quick on eri0(!) from 192.168.100.0/24 to any
pass in quick on eri0(!) proto tcp/udp from
192.168.201.0/24 to 192.168.201.21/32 keep state
pass in quick on eri1(!) all
block in all
```

The verbose option displays the rule set and the wait for input. This time rule set line 8 should block the input list. The input is as follows:

```
in on eri0 udp 192.168.100.45,12345 192.168.100.13,54
```

```
input: in on eri0 udp 192.168.100.45,12345 192.168.100.13,54

p:i
p:i
p:i
p:i
p:i
p:i
p:i
p:i
p:i=.8 *block ip 28(20) 17 192.168.100.45,12345 > 192.168.100.13,54
--------------
```

The utilities waits for the next input line. The input lines have the following syntax:

```
"in"|"out" "on" if ["tcp"|"udp"|"icmp"] srchost[,srcport]
dsthost[,destport] [FSRPAU]
```

In the preceding output, you can see that the interface (i) was matched on the first nine rules. The asterisk in the last line indicates the rule match. Where the rules contain no quick keywords, an asterisk indicates each matched rule.

The following example is an input line that represents a packet that the rule set will pass. This time, the interface is `eri1` and does not match any rule until the last rule that passes it.

```
in on eri1 tcp 192.168.100.45,12345 192.168.200.35,22
input: in on eri1 tcp 192.168.100.45,12345 192.168.200.35,22

p
p
p
p
p
p
p
p
p
p
p:i=.10 *pass ip 40(20) 6 192.168.100.45,12345 > 192.168.200.35,22
--------------
```

Installation, Upgrading, and Migration

Installation

IP Filter is installed as part of the core Solaris OS.

If Solaris OS IP Filter is installed but has not yet been activated, you can start packet filtering by un-plumbing and then plumbing each network interface with the `ifconfig`(1M) command after running the `ipf` command with the `-f` option. The first activation of Solaris OS IP Filter configures the `autopush`(1M) facility to push the STREAMS module onto all network interfaces. However, for this to take effect, the system must be rebooted or each network interface must be plumbed using the `ifconfig`(1M) command.

Upgrading

If an open source version of IP Filter is detected during a upgrade, the open source files are removed with the exception of configuration files, and the Solaris OS IP Filter is installed. This is deemed not to be a problem for customers as the open source version compiled for previous Solaris OS versions does not work correctly on Solaris 10 OS.

Migration

The IP Filter configuration files from previous open source version of IP Filter work with Solaris OS IP Filter. Solaris™ EFS (SunScreen Enterprise Firewall), if installed, must be removed or disabled.

For systems using Solaris OS IP Filter, when a new type of network interface is added to the system, the `autopush`(1M) mechanism must be configured to supply the `pfil` STREAMS module on the new device. This is only necessary if there were previously no other interfaces using the same device driver.

```
/etc/ipf/iu.ap
#      major   minor lastminor  modules
ce     -1      0       pfil
```

```
# autopush -f /etc/ipf/iu.ap
```

Module 11

Other Changes to Networking

Objectives

This module is an overview of the new features included in the Solaris™ 10 Operating System (Solaris 10 OS) that have not been addressed in the other networking categories.

Upon completion of this module, you should be able to:

- Describe and Configure the System Management Agent (SMA)
- Describe changes to Dynamic Host Configuration Protocol (DHCP) commands
- Identify changes to routing sockets

The SMA

The SMA is an implementation of an agent as described by Request For Comments (RFC) 3411 – *An Architecture for Describing Simple Network Management Protocol Management Frameworks*. The agent is used for management of systems using Simple Network Management Protocol (SNMP). It is lightweight, and portable to multiple operating systems (OSs) and platforms, including Solaris OS and Linux. It can be extended through the use of modules written to application programming interfaces (APIs) and Agent X, RFC 2741 – *Agent Extensibility Protocol*.

SMA is designed to be a standalone agent. This means multiple management applications can access it, provided they communicate with it using the SNMP protocols. The agent can also co-exist with existing SNMP agents and will replace some of the legacy SNMP agents offerings from Sun.

SMA is based on the Net-SNMP open source implementation version 5.0.9. This is described at `http://www.net-snmp.org/` and was formerly known as University of California Davis-Simple Network Management Protocol (UCD-SNMP). The SMA is designed to support the latest SNMP standards.

In the Solaris 10 release, the SMA can co-exist with the Solstice Enterprise Agents™ software. From an SNMP manager view, the SMA operates in the same way as the Solstice Enterprise Agents™ software. Unlike the Solstice Enterprise Agents software, the SMA supports SNMP version 3 (SNMPv3) and supports many more default Management Information Bases (MIB) than the Solstice Enterprise Agents software.

Table 11-1 show the MIBs that the SMA supports.

Table 11-1 SMA Supported MIBs

MIB	Defining RFC
SNMP-COMMUNITY MIB	RFC 2576
SNMPv2-TM (Transport Mappings)	RFC 3417
SNMP-MPD-MIB (Message Processing and Dispatching)	RFC 3412

Table 11-1 SMA Supported MIBs (Continued)

MIB	Defining RFC
SNMP-TARGET-MIB (Specification of targets for traps)	RFC 3413
SNMP-NOTIFICATION-MIB (Trap filtering)	RFC 3413
SNMP-PROXY-MIB (Trap forwarding)	RFC 3413
SNMP-USER-BASED-SM-MIB (USM for SNMPv3)	RFC 3414
SNMP-VIEW_BASED-ACM-MIB (VACM for SNMP)	RFC 3415
SNMPv2-MIB	RFC 3418
MIB II	RFC 1213
Host Resources MIB	RFC 2790

Note – The text files of MIB definitions are in the `/etc/sma/snmp/mibs/` directory. The Sun MIB is an additional MIB. It is the MIB II with Sun-specific object groups added. Sun MIB was originally provided in Solstice Enterprise Agents software beginning with the Solaris 2.6 OS.

SMA Architecture

The SMA implements the agent component of the SNMP management framework standards. There are several standards that form part of this framework. These include the following:

- AgentX Protocol: Industry standard mechanism defined in RFC 2741.
- USM: User-based Security Model for authentication and privacy. This is defined in RFC 3414.
- VACM: View-based Access Control Model for authorization. This is defined in RFC 3415.

Figure 11-1 shows the SMA Architecture:

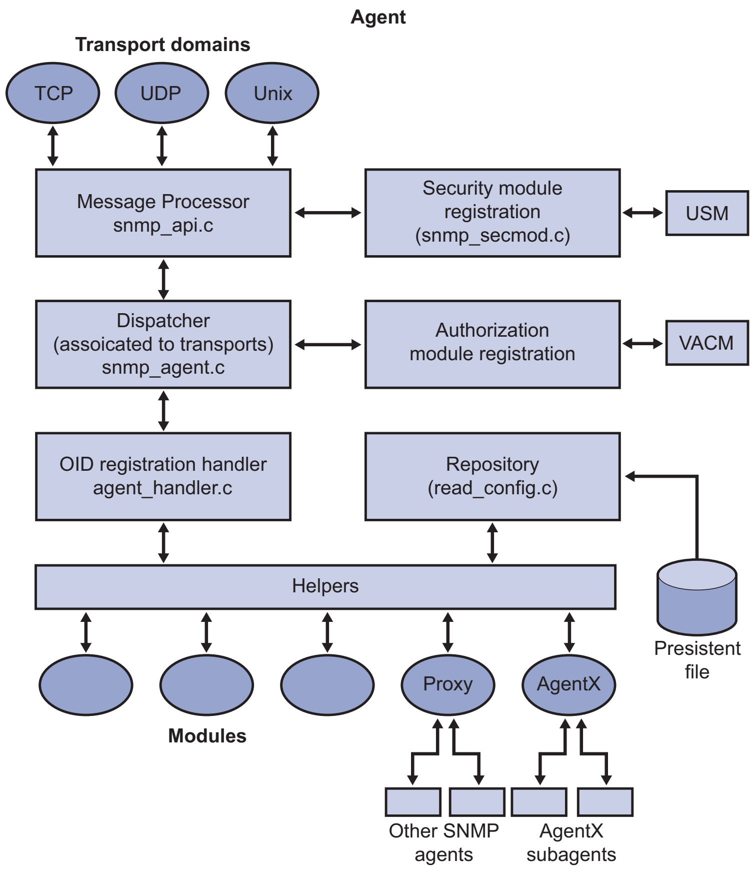


Figure 11-1SMA Architecture

The following list describes the components of the SMA agent.

- Transport domains – The SMA agent currently supports the transports Transmission Control Protocol (TCP), User Datagram Protocol (UDP), and UNIX® domain sockets. The agent can receive and transmit SNMP messages through these transports. The agent's implementation of each transport implements functions to send and receive raw SNMP data. The raw messages that the transport domains receive are passed to the message processor for further processing. The message processor also transfers raw SNMP messages to the transport domain for sending.
- Message processor – The message processor decodes raw SNMP messages into internal Protocol Data Unit (PDU) structures. The processor also encodes PDUs into raw SNMP messages. The SNMP messages are encoded by using Binary Encoding Rules, which are described in RFC 3416 and RFC 1157. The message processor also handles the security parameters in the SNMP messages. If the messages include USM security parameters, the message processor

passes the required parameters to the USM module. Additionally, trap messages from modules can be sent to the message processor for transmission.

- USM module – The USM module handles all processing that is required by the USM as defined in RFC 3414. The module also implements the SNMP-USER-BASED-SM-MIB as defined in the same RFC. The USM module, when initialized, registers with the agent infrastructure. The message processor invokes the USM module through this registration.

  The USM module decrypts incoming messages. The module then verifies authentication code and creates the PDU. For outgoing messages, the USM module encrypts the PDU and generates authentication codes. The module then passes the PDU to the message processor, which then invokes the dispatcher.

  The USM module's implementation of the SNMP-USER-BASED-SM-MIB enables the SNMP manager to issue commands to manage users and security keys. The MIB also enables the agent to ensure that a requesting user exists and has the proper authentication information. When authentication finishes, the agent carries out the request.

  The various keys that the USM module needs to perform encryption and authentication operations are stored persistently.

- Dispatcher – The dispatcher is responsible for routing messages to appropriate destinations. After the USM module processes an incoming message into PDUs, the dispatcher performs an authorization check. The registered access control module, which is the VACM module, does this authorization check. If the check succeeds, the dispatcher uses the agent registry to determine the module that has registered for the relevant object identifier. The dispatcher then invokes appropriate operations on the module. A particular request might cause the dispatcher to invoke several modules if the SNMP request contains multiple variables. The dispatcher tracks outstanding requests through session objects. Responses from the modules are then dispatched to the transports that are associated with the session objects. The message processor performs the appropriate message encoding.

- VACM Module – The VACM is described in RFC 3415. This RFC also defines the SNMP-VIEW-BASED-ACM-MIB. The MIB specifies objects that are needed to control access to all MIB data that is accessible through the SNMP agent. Upon initialization, the VACM module registers as the access control module with the agent infrastructure. The VACM module implements access control checks according to several parameters that are derived from the SNMP message. These parameters specify the following:

- The security model being used, which can be USM, v1 communities, or v2 communities
- The security name, which is the user name in USM, and the community string in v1 and v2
- The context
- The object being accessed
- The operation being performed

By implementing the SNMP-VIEW-BASED-ACM-MIB, the VACM module handles manipulation of various table entries that VACM mandates. These table entries are looked up in performing the VACM check and are maintained persistently in the agent configuration file.

- Repository – The agent configuration file, `snmpd.conf`, is the repository for the agent. Configuration tokens for various modules are stored in the repository. The modules have access to these configuration tokens when the modules are initialized. Modules can also register callback routines with the repository. The callbacks are invoked when the module state needs to be persisted, or written to disk to be retrieved later. Within the callbacks, each module is allowed to output its state. When the agent shuts down, the callbacks are used to save the module's state. An SNMP SET command can also be used to cause a module's state to be persisted.
- Proxy Module – The proxy module handles proxy forwarding of SNMP messages to and from other SNMP agents. The proxy module can also map between SNMPv1 and SNMPv2 protocols according to the rules specified in RFC 2576.

  The proxy module stores its configuration tokens in the agent configuration file. A particular configuration entry can associate the object identifiers (OIDs) within a context with another SNMP agent. The configuration file also specifies community strings and destination transport end points. By using these configuration tokens, the proxy module registers as the handler for the specific OIDs. When an incoming request for any of the proxy module's OIDs reaches the dispatcher, the dispatcher invokes the proxy module. The proxy module then issues appropriate SNMP requests to the target agents. Responses are returned to the dispatcher.

  The SMA uses the proxy module for interaction with the Solstice Enterprise Agents software.

- AgentX module – The AgentX module implements RFCs 2741 and 2742. The AgentX module registers as the handler for the AgentX-related registration tables defined in the AGENTX-MIB. The transports that are used for the AgentX protocol interactions are specified in the agent configuration file. In the SMA agent, the transports are typically UNIX domain sockets. When the AgentX module is initialized, the module creates sessions on these transports and registers as the handler for these sessions. In the SMA, the only allowable AgentX transport is UNIX domain sockets, so only sessions on UNIX domain sockets are created.

  When an AgentX subagent starts, the subagent sends its registration requests with messages that use the AgentX protocol to the master agent. The sessions that the AgentX module creates receive the requests. The message processor decodes the message, then invokes the AgentX module. The AgentX module, rather than the dispatcher module, is the handler for these sessions.

  The AgentX module then registers as the handler for OIDs that are specified in the subagent registration message. When the dispatcher receives requests for these OIDs, the requests are directed to the AgentX module, which in turn connects to the required subagent. Requests to unregistered OIDs are handled similarly.
- Extension Modules – Extension modules, which are depicted at the bottom of Figure 11-1 on page 11-4, are the means by which MIBs are implemented in the agent. An extension module registers with the agent all the OIDs that the module manages. The module also implements functionality to perform SNMP operations on the module's objects. As Figure 11-1 shows, helper routines or handlers in the API can be inserted between a module and the agent infrastructure. These handlers can have various functions, such as handling details of table iterations or providing debug output.

Security

The SMA supports SNMPv1, SNMPv2c and SNMPv3. The SNMPv1 and SNMPv2c authentication services are based on community strings defined on the management station. The SNMPv3 authentication service is based on users. Each request must contain either a community name or a user name depending upon the protocol being used.

The SNMPv3 authentication process implements the USM for getting a security name and security level from a user name. Similarly, both SNMPv1 and SNMPv2c determine the security level from the community string. The security name and a security level are then used together with a context string, a group name and a view name to perform access control. Access control is done through the VACM. This access control model is used after the authentication process. So while USM is for authentication, VACM is for authorization.

The agent provides the ability to manage user entries through the main `snmpd.conf` configuration file and by use of the `snmpusm` command, through the USM MIB. The USM MIB enables the SMA to find information about the user, including whether the user even exists. Every request from a user is checked against the USM MIB. If the user exists, the USM MIB checks the following:

- If the user is allowed authenticated requests.
- What type of authentication encoding is allowed.

The USM MIB uses the local store key to compute a new digest based upon the authentication protocol that a particular user in the MIB specifies. The computed digest is compared to the one saved from the incoming packet. If the digests are the same, it is authenticated.

When different levels of privileges are required, it is not enough to just assign users authentication and privacy passwords or keys. For example, it might be desirable to allow a user to write to a subagent but at the same time allow every user to read the status of the remote agent.

VACM uses a MIB to specify which user can access which part of the agent MIB. Access conditions include the following:

- A level of authentication: for example, read only or read and write
- A security model: for example, SNMPv1, SNMPv2c, or USM
- A user name
- A view type: for example, a part of the MIB (context) to which a user might have read access but not write access

VACM authorization is configured using the main `snmpd.conf` configuration file. A token is specified, followed by a set of attributes. There are also a number of wrapper tokens that are easier to use in a less complex SNMP environment.

"Configuring and Using SMA" on page 21 shows examples of USM and VACM configuration.

Extending the Agent

The Net-SNMP agent can be extended in the following ways:

- Modules can be loaded into the master agent's process image. A shared object is dynamically loaded into agent when the agent is running. The shared object registers the OIDs for the MIB that the shared object supports. The location of the shared object libraries for the module can be specified through SNMP requests or in the agent configuration file.
- Modules can be loaded into secondary SNMP subagents. Subagents are separate executable programs that can dynamically register themselves with the agent that is running on the designated SNMP port. The monitoring agent processes any SNMP request that comes to the SNMP port, and can send a request to a subagent if needed. In this scenario, the agent on the designated port is called the master agent. The AgentX RFCs 2741 and 2742 define the protocols between the subagent and master agent as well as the MIBs that contain details of the registrations.
- A module can be delivered as an SNMP agent. The master agent can interact with such agents through a proxy mechanism.

The SMA extends the Net-SNMP agent to support the following MIBs:

- MIB II – Described in RFC 1213, defines the structure of management information and the network management protocol for Transmission Control Protocol/Internet Protocol (TCP/IP)-based internets. The SMA implements all the object groups of MIB II except the Exterior Gateway Protocol (EGP) group.
- Host Resources MIB – Described in RFC 2790, defines the structure of management information for managing host systems. The SMA implements the same host resources MIB that is included in the base Net-SNMP agent.
- Sun MIB – The Sun MIB is the MIB II with Sun-specific object groups added. Sun MIB was originally provided in Solstice Enterprise Agents software beginning with the Solaris™ 2.6 OS. The SMA implements the following groups from the Sun MIB:
  - Sun System group
  - Sun Processes group

- Sun Host Performance group

Support for these MIBs is provided as static modules that run in the SMA agent. The agent was created by recompiling the Net-SNMP agent to include these modules.

The SMA provides an additional MIB for hardware, the Entity MIB, in an external dynamically loaded module. The Entity MIB is described in RFC 2737. The Entity MIB is implemented as a skeleton MIB, so that module developers can populate the various tables of the MIB. A single agent can use the Entity MIB for managing multiple logical entities and physical entities.

Content for Developers

The Solaris 10 OS distribution of SMA includes APIs and tools for developers that enable the agent to be extended through modules. Developers can use the tools and APIs to create a module to support the custom features of a managed device. A management program can then be used for monitoring and managing the device.

The developers content includes tools and Perl modules that the tools need, API libraries for Net-SNMP functions and an API library for using the Entity MIB functions. There is also a set of demo modules that demonstrate how to implement some types of data modeling.

"It migrates community strings from the Sun SNMP Management Agent for Sun Fire and Netra systems to the SMA. You are advised if an identical string is configured for both agents." on page 17 lists tools and libraries. Header files can be found in the `/usr/sfw/include` directory and some code examples for the Net-SMNP API function can be found in the `/usr/sfw/doc/sma_snmp/html/` directory.

Note – The documents in the `/usr/sfw/doc/sma_snmp/html/` directory are generated when the SMA agent is compiled. They are part of the Net-SNMP product, and Sun does not write or support them.

Table 11-2 lists and describes the demo modules found in the `/usr/demo/sma_snmp` directory.

Table 11-2 Demo Modules

Module Name	**Description**
`demo_module_1`	This code example shows you how to modify the code that the `mib2c -c mib2c.scalar.conf` command generates to perform a scalar data retrieval.
`demo_module_2`	This example performs data retrieval and data setting for a simple table that provides file monitoring.
`demo_module_3`	This code example shows you how to modify the code that the `mib2c -c mib2c.iterate.conf` command generates to perform data retrieval for a general table.
`demo_module_4`	This example module demonstrates the following features: • Automatic refresh of data at regular intervals • Checking for alarm condition at regular intervals and generate trap if needed • Reading threshold values • The use of the parameter `SNMP_CALLBACK_POST_READ_CONFIG`
`demo_module_5`	This code example demonstrates how to implement data persistence for a module across agent restarts.
`demo_module_6`	This code example demonstrates how to implement a module in such a way that multiple instances of the module can run simultaneously on a single host.
`demo_module_7`	This code example demonstrates how to dynamically update multi-instance modules.
`demo_module_8`	This code example creates an AgentX subagent that calls a module that returns load averages.

Table 11-2 Demo Modules (Continued)

Module Name	Description
`demo_module_9`	This code example demonstrates how to implement objects for long-running data collection. These objects are implemented in a way that allows the agent to respond to other requests while external data collection occurs. When data collection is complete, the agent can reply to the request.
`demo_module_10`	This example demonstrates a module design that handles long-running data collections so that an SNMP manager can poll their values. The example also shows how to implement objects that normally would block the agent as it waits for external events in such a way that the agent can continue responding to other requests while this implementation waits.
`demo_module_11`	This code example shows how the `libentity.so` module API functions might be used. The `libentity.so` module supports the Entity MIB.
`demo_module_12`	This code example shows how to generate code templates from a MIB for the SMA, and how to generate code templates from a MIB in Solstice Enterprise Agents software. This information should be helpful if you are migrating a Solstice Enterprise Agents software subagent to use as a module with the SMA.

Note – Technical support for developers of modules for the SMA is provided through the Net-SNMP open source community at `http://www.net-snmp.org`. You might find the developers discussion mailing list `net-snmp-coders@lists.sourceforge.net` to be helpful. An archive for the mailing list is at `http://sourceforge.net/mailarchive/forum.php?forum_id=7152`.

Using SMA With Solstice Enterprise Agents™ Software

Support for the Solstice Enterprise Agents software is to be discontinued in a future Solaris OS release. The Solstice Enterprise Agents software master agent is `snmpdx`. It is in the `/usr/lib/snmp/` directory. The SMA master agent, `snmpd`, replaces its functions. This is in the `/usr/sfw/sbin/` directory. For this reason, any Solstice Enterprise Agents software subagents that developers have created must at some point be migrated to use the SMA.

In this Solaris OS release, the Solstice Enterprise Agents software and associated subagents run concurrently with the SMA.

Figure 11-2 shows a Solstice Enterprise Agents software deployment before SMA. Solstice Enterprise Agents software includes a subagent, `mibiisa`, that implements MIB-II and the sun.mib. Another subagent acts as a mapper between SNMP and Distributed Management Interface (DMI) protocol and is also part of the Solstice Enterprise Agents software.

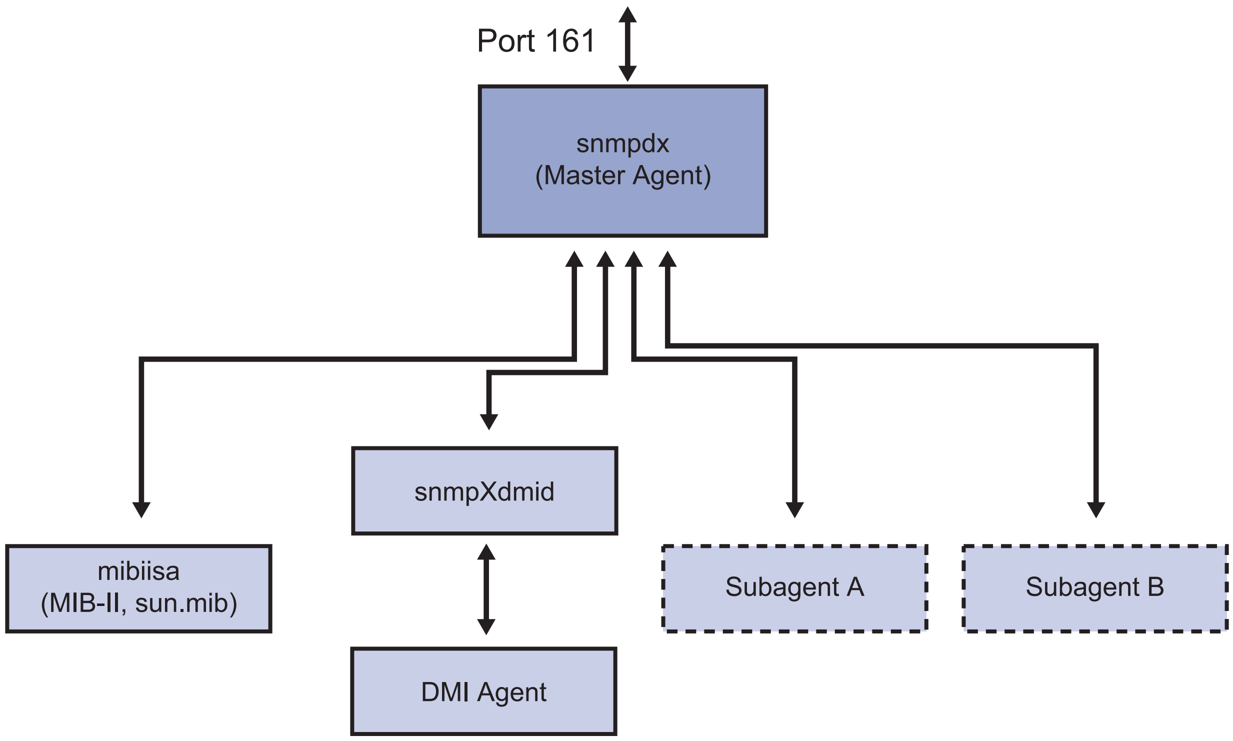


Figure 11-2 Solstice Enterprise Agents Software Before SMA

Once the SMA agent is installed on the system, the Solstice Enterprise Agents software master agent is reconfigured to run on an alternate port. The SMA master agent is configured in such a way that it communicates with the Solstice Enterprise Agents software master agent via the proxy mechanism. All the Solstice Enterprise Agents software subagents (except `mibiisa`) continue to work with the Solstice Enterprise Agents software master agent as their master agent. However, the developers of subagents are encouraged to use SMA master agents instead.

The functionality of the `snmpdx.mib` file implementation in Solstice Enterprise Agents software is handled by the existence of the SMA Solstice Enterprise Agents software proxy subagent. This subagent will intercept all Solstice Enterprise Agents software master agent management information requests bound for port `161`. The requests bound specifically for the subagent configuration table portion of the `snmpdx.mib` file implementation are processed to create dynamic proxies to represent dynamic Solstice Enterprise Agents software subagents that are coming on line and must be accessible by SMA. All other `snmpdx.mib` file traffic is forwarded to Solstice Enterprise Agents software.

Figure 11-3 shows a Solstice Enterprise Agents software deployment after SMA.

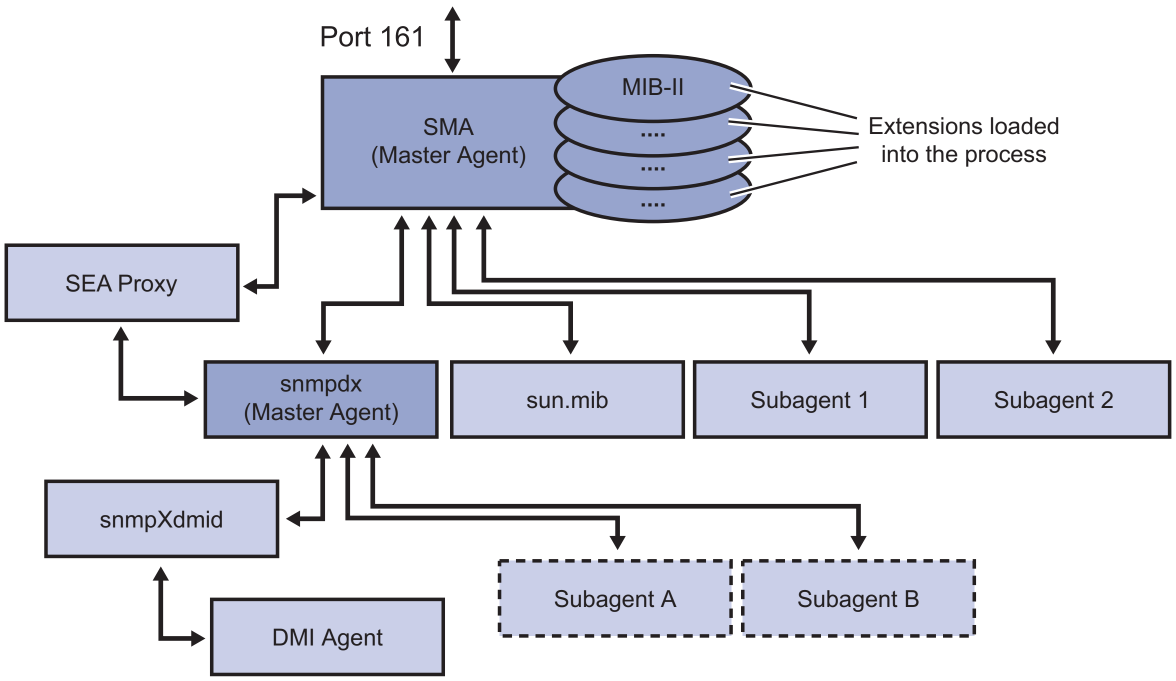


Figure 11-3 Solstice Enterprise Agents Software After SMA

Using SMA and JDMK™

Java™ Dynamic Management Kit (JDMK) enables SNMP-based instrumentation within the JDMK agent infrastructure. Like the SMA, JDMK also supports standards such as SNMP v3, SNMPv2c, SNMPv1, USM, and proxy. It does not support AgentX.

Though both JDMK and the SMA address SNMP instrumentation, JDMK is very suited to Java technology-based environments. The SMA is more suited to native C language based implementations.

In Sun systems where both JDMK and the SMA are present, the SMA by default resides on port `161`. JDMK agents can publish their SNMP MIBs by being proxied from the SMA. Proxy can be set up using the proxy forwarding mechanism within the SMA.

The master agent (SMA) and the proxied JDMK agent must handle security. Security parameters contained in the proxy definition are forwarded to the proxied JDMK agent. If the request passes the SMA authentication and authorization and is forwarded to its proxy handler then the dispatched request is proxied to the JDMK agent. The JDMK agent has its own local datastore that authorizes or rejects the message.

If several JDMK agents have the same MIB, SNMP contexts must be used in conjunction with proxying to differentiate between different instances of the same MIB. The context name can be based on either the process identification (PID) or on the port on which the JDMK agent is running.

Using SMA and Sun SNMP Management Agent for Sun Fire™ and Netra™ Systems

The `/usr/sfw/lib/sma_snmp/masfcnv` script is used to assist the system administrator in migrating an existing set of configuration files for the Sun SNMP Management Agent for Sun Fire™ and Netra™ systems to the SMA.

The `masfcnv`(1M) migration script performs the following functions:

- It migrates USM (SNMP) user names and passwords. The `masfcnv` migration script checks that USM user names migrated from the Sun SNMP Management Agent for Sun Fire and Netra systems to the

SMA do not already exist in the SMA. If there is a duplicate, you must determine if the identified user should be treated as the same user in the SMA.

You must also decide if you want to migrate the Sun SNMP Management Agent for Sun Fire and Netra systems key associated with that user or continue to use the existing SMA key for that user. The Sun SNMP Management Agent for Sun Fire and Netra systems only supports MD5 based keys. The SMA supports additional authentication schemes such as Secure Hash Algorithm (SHA) and encryption (Data Encryption Standard [DES]) for SNMP requests. A migrated user therefore cannot use these additional capabilities until the necessary keys are configured. However, access based on Message-Digest algorithm 5 (MD5) authentication is available to such users.

- It uses the template file at `/usr/sfw/lib/sma_snmp/snmpd.conf` to create a new `snmpd.conf` file. This new `snmpd.conf` agent configuration file is specifically for the Sun SNMP Management Agent for Sun Fire and Netra systems and is installed at the `/etc/opt/SUNWmasf/conf/snmpd.conf` file path. The Sun SNMP Management Agent for Sun Fire and Netra systems uses this to modify the SMA main configuration file at the `/etc/sma/snmp/snmpd.conf` file path and to modify the SMA persistent storage file at the `/var/sma_snmp/snmpd.conf` file path.
- It replaces Sun SNMP Management Agent for Sun Fire and Netra systems configuration files by a default configuration. This default configuration sets up the Sun SNMP Management Agent for Sun Fire and Netra systems as an AgentX subagent.
- It makes backups of the changed configuration files. Configuration file back-ups are made by appending the extension `.bak.`*`n`* to the filename where *`n`* is an optional number.
- It replaces the existing Sun SNMP Management Agent for Sun Fire and Netra systems startup script in the `/etc/init.d` file with a new script.
- It migrates the VACM configuration. The Sun SNMP Management Agent for Sun Fire and Netra systems configuration related to the OID space that the SUN MIB uses is migrated automatically. If there is VACM configuration related to other OIDs, such as VACM information related to the system branch in MIB-II, it is necessary to confirm if migration is required.

- It migrates trap destinations from the Sun SNMP Management Agent for Sun Fire and Netra systems to the SMA. Entries originally configured for both agents do not result in duplicate entries in the migrated configuration.
- It migrates community strings from the Sun SNMP Management Agent for Sun Fire and Netra systems to the SMA. You are advised if an identical string is configured for both agents.

Table 11-3 shows relevant files and man pages:

Table 11-3 Files and Man Pages

Files	**Description**
`sma_snmp`(5)	SMA overview man page
`/etc/init.d/init.sma`	Boot time start script, `init.sma`(1M)
`/usr/sfw/sbin/snmpd`	The daemon, `snmpd`(1M)
`/usr/sfw/sbin/snmptrapd`	Application that receives and logs SNMP TRAP and INFORM messages
`/etc/sma/snmp/mibs/`	MIB location: see Table 11-1
`snmp_config`(4)	Configuration overview man page
`snmp_variables`(4)	Describes the format for specifying variable names to SNMP tools
`snmpcmd`(1M)	Common options for SMA application
`/etc/sma/snmp/snmpd.conf`	Agent configuration file `snmpd.conf`(4)
`/etc/sma/snmp/snmptrapd.conf`	Configuration file for the Net-SNMP trap daemon
`/var/sma_snmp/snmpd.conf`	Persistent data file
`/usr/sfw/include/net-snmp/`	Header Files

Note – The main configuration file for the operation of the SMA is `snmpd.conf` and is in the `/etc/sma/snmp` directory. The Solstice Enterprise Agents software configuration file is also named `snmpd.conf`, and is in the `/etc/snmp/conf` directory. The SMA persistent storage file is also named `snmpd.conf`, and is in the `/var/sma_snmp/` directory. There is a separate `snmpd.conf` file in the `/usr/sfw/lib/sma_snmp` directory. This is the template file that the Sun Fire system migration script uses. The migration script uses this to modify the main SMA configuration file `snmpd.conf` described previously. The Sun SNMP Management Agent for Sun Fire and Netra systems uses an agent also named `snmpd`.

Table 11-4 shows the available tools:

Table 11-4 SMA Tools

Tools (located in `/usr/sfw/bin`)	**Description**
`encode_keychange`	Collects information to build a KeyChange encoding, per the textual convention given in RFC 2274, Section 5. Computes the value and prints it to standard output (stdout) as a hexadecimal string.
`fixproc`	Fixes a process named `proc` by performing the specified action. The actions can be check, kill, restart, exist, or fix. The action is specified on the command line or is read from a default database, which describes the default action to take for each process.
`mib2c`	A script that takes a MIB and converts it into C code. That C code can then be used as a template to implement your MIB.
`net-snmp-config`	Net-SNMP configuration options.
`snmpbulkget`	An SNMP application that uses the SNMP GETBULK request to efficiently query information on a network entity.
`snmpbulkwalk`	An SNMP application that uses the SNMP GETBULK request to efficiently query a network entity for a tree of information.
`snmpconf`	A configuration file setup command for agent.

Table 11-4 SMA Tools (Continued)

Tools (located in `/usr/sfw/bin`)	**Description**
`snmpdelta`	Monitors the specified integer valued OIDs, and reports changes over time.
`snmpdf`	A networked version of the `df`(1M) command.
`snmpget`	Invokes an SNMP GET request to query for information on a network entity.
`snmpgetnext`	An SNMP application that uses the SNMP GETNEXT request to query information on a network entity.
`snmpinform`	An SNMP application that uses the SNMP INFORM operation to send information to a network manager.
`snmpnetstat`	Symbolically displays the values of various network-related information retrieved from a remote system using the SNMP protocol.
`snmpset`	The SNMP SET request to set information on a network entity.
`snmpstatus`	An SNMP application that retrieves several important statistics from a network entity.
`snmptable`	An SNMP application that repeatedly uses the SNMP GETNEXT or GETBULK requests to query for information on a network entity.
`snmptest`	An SNMP application that can monitor and manage information on a network entity.
`snmptranslate`	An SNMP application that translates one or more SNMP OID values from their symbolic (textual) forms into their numerical forms (or vice-versa).
`snmptrap`	An SNMP application that uses the SNMP TRAP operation to send information to a network manager.
`snmpusm`	An SNMP application that can be used to do simple maintenance on an SNMP agent's USM table.

Table 11-4 SMA Tools (Continued)

Tools (located in `/usr/sfw/bin`)	Description
`snmpvacm`	An SNMP application that can be used to do simple maintenance on the VACM table.
`snmpwalk`	An SNMP application that uses SNMP GETNEXT requests to query a network entity for a tree of information.

Table 11-5 shows other relevant files.

Table 11-5 Other Files

Other Files	Description
`/usr/sfw/doc/sma_snmp/html/modules.html`	The module API documentation describes some of the Net-SNMP helper APIs as well as other APIs needed to develop SNMP modules.
`/usr/sfw/lib/libnetsnmp.so`	Core SNMP libraries. The 64-bit libraries are in the `/usr/sfw/lib/sparcv9` directory.
`/usr/sfw/lib/libnetsnmpagent.so.`	Agent and module libraries. The 64-bit libraries are in the `/usr/sfw/lib/sparcv9` directory.
`/usr/sfw/lib/libnetsnmphelpers.so`	Module helper libraries. The 64-bit libraries are in the `/usr/sfw/lib/sparcv9` directory.
`/usr/sfw/lib/libnetsnmpmibs.so`	MIB libraries. The 64-bit libraries are in the `/usr/sfw/lib/sparcv9` directory.
`/usr/sfw/lib/libentity.so`	Allows addition of content to Entity MIB. The 64-bit libraries are in the `/usr/sfw/lib/sparcv9` directory.

Configuring and Using SMA

This section covers starting, configuring, and using SMA.

Starting SMA

Starting and stopping the Solaris Management Agent is typically done using the `/etc/init.d/init.sma` script:

```
# /etc/init.d/init.sma start
```

Errors are logged to the `/var/log/snmpd.log` file. The agent does not start if another program is already using port `161`.

Once the agent is started using the `init.sma` script, it will start automatically at boot time thereafter. To disable the agent so that it no longer starts at boot time, you must change an entry in the `/etc/sma/snmp/snmpd.conf` file:

```
######################################################
# SECTION: Admins who want to disable the snmpd daemon from
# starting at boot time.
# Change DISABLE=NO to DISABLE=YES
# DO NOT DELETE
# DO NOT UNCOMMENT
#DISABLE=NO
#
# end ADMIN
```

Change this section to:

```
######################################################
# SECTION: Admins who want to disable the snmpd daemon from
# starting at boot time.
# Change DISABLE=NO to DISABLE=YES
# DO NOT DELETE
# DO NOT UNCOMMENT
#DISABLE=YES
#
# end ADMIN
```

Note – The `/etc/init.d/init.sma` script parses the `DISABLED=YES` token at boot time.

To restart the agent use the `init.sma` script, enter the following:

```
# /etc/init.d/init.sma restart
```

You should not send the agent a `SIGHUP` signal (see the `signal`(3HEAD) man page for more information), as data can be lost. For example, if you added users authentication, which is stored in the persistent data file `/var/sma_snmp/snmpd.conf`, changes are not saved if the agent was sent the `SIGHUP` signal but are saved if the `init.sma` script is used to restart the agent.

You can start the agent from the command line and there are a number of options that can be passed to the agent when using the command line. Options passed to the agent on the command line override options set in the `snmpd.conf`(4) file. Using the command line can be helpful when you try to debug problems.

The following example sets a debug token to proxy (`-Dproxy`) with output to the `STDERR` output stream (`-L`):

```
# /usr/sfw/sbin/snmpd -Dproxy -L
seaproxy_sendReq: SEA Master Agent not responding
proxy_config: entering
proxy_args: final args: 0 = snmpd-proxy
proxy_args: final args: 1 = -v
proxy_args: final args: 2 = 1
proxy_args: final args: 3 = localhost:16161
proxy_args: final args: 4 = .1.3.6.1.4.1.42.2.15
proxy_config: parsing args: 5
proxy_config: done parsing args
proxy_init: name = .1.3.6.1.4.1.42.2.15
proxy_init: registering at: SNMPv2-
SMI::enterprises.42.2.15
NET-SNMP version 5.0.9
```

The following example displays all compiled modules displayed to the `STDERR` output stream:

```
# /usr/sfw/sbin/snmpd -Dmib_init -L
mib_init: initializing: diskio
mib_init: initializing: system_mib
mib_init: initializing: sysORTable
```

```
mib_init: initializing: at
mib_init: initializing: interfaces
mib_init: initializing: snmp_mib
...
```

The following example turns on ALL debug tokens and displays to the STDERR output stream:

```
# /usr/sfw/sbin/snmpd -Dall -L
trace:  default_store.c, 191
netsnmp_ds_set_boolean: Setting APP:1 = 0/False
trace:  default_store.c, 191
netsnmp_ds_set_boolean: Setting LIB:11 = 1/True
trace:  default_store.c, 191
netsnmp_ds_set_boolean: Setting APP:1 = 0/False
trace:  agent_handler.c, 182
handler::register: Registering ::null at .0
trace:  agent_handler.c, 278
handler:inject: injecting bulk_to_next before null
trace:  agent_registry.c, 556
...
```

Start the agent on an alternate port as follows:

```
# /usr/sfw/sbin/snmpd TCP:localhost:16161
```

Configuring the snmpd.conf File

The primary configuration file for SMA is `snmpd.conf`. By default, the agent searches the following directories, in the order listed, and parses files with the extension `.conf` and `local.conf`:

- `/etc/sma/snmp`
- `/usr/sfw/lib`
- `$HOME/.snmp`

The variable SNMPCONFPATH can be used to change the search path for configuration files.

The `/usr/sfw/bin/snmpconf` utility is a simple script that walks you through setting up a configuration file, step-by-step. It works by asking you a series of questions. It creates the configuration file based on your responses.

In its default mode of operation, the snmpconf(1M) script prompts you with menus showing sections of the various configuration files it knows about. When you select a section, a submenu appears and lists the descriptions of the tokens that can be created in that section. After you select a description, you are prompted with questions that determine the specification of the selected token.

When you quit the snmpconf(1M) script, any configuration files that were edited are saved to the local directory. The snmpconf(1M) script supplies comments in the configuration files for each change.

The following are examples of using the snmpconf(1M) script:

The -G option alone displays a list of groups that are allowed with the -g option.

```
# /usr/sfw/bin/snmpconf -G

Known GROUPs of tokens:

  system_setup
  basic_setup
  monitoring_services
  access_control
  trapsinks
```

The -g basic_setup option specifies a select group of questions, in this case, a basic install. The -I option specifies a directory to which the configuration files are written when the script completes.

```
# /usr/sfw/bin/snmpconf -I /var/tmp -g basic_setup

The following installed configuration files were found:

   1:  /etc/sma/snmp/snmpd.conf
   2:  /var/tmp/snmpd.conf

Would you like me to read them in?  Their content will be merged with the
output files created by this session.

Valid answer examples: "all", "none","3","1,2,5"

Read in which (default = all): none
************************************************
*** Beginning basic system information setup ***
************************************************
```

```
Do you want to configure the information returned in the system MIB group
(contact info, etc)? (default = y): y

Configuring: syslocation
Description:
  The [typically physical] location of the system.
    Note that setting this value here means that when trying to
    perform an snmp SET operation to the sysLocation.0 variable will make
    the agent return the "notWritable" error code.  IE, including
    this token in the snmpd.conf file will disable write access to
    the variable.
    arguments:  location_string

The location of the system: Network Support Group

Configuring: syscontact
Description:
  The contact information for the administrator
    Note that setting this value here means that when trying to
    perform an snmp SET operation to the sysContact.0 variable will make
    the agent return the "notWritable" error code.  IE, including
    this token in the snmpd.conf file will disable write access to
    the variable.
    arguments:  contact_string

The contact information: Network Administrator

Finished Output: syscontact  "Network Administrator"
Do you want to properly set the value of the sysServices.0 OID (if you
don't know, just say no)? (default = y): y

Configuring: sysservices
Description:
  The proper value for the sysServices object.
    arguments:  sysservices_number

does this host offer physical services (eg, like a repeater) [answer 0 or
1]:0
does this host offer datalink/subnetwork services (eg, like a bridge): 0
does this host offer internet services (eg, supports IP): 1
does this host offer end-to-end services (eg, supports TCP): 1
does this host offer application services (eg, supports SMTP): 1

Finished Output: sysservices 76
**************************************
*** BEGINNING ACCESS CONTROL SETUP ***
```

```
**************************************
Do you want to configure the agent's access control? (default = y): y
Do you want to allow SNMPv3 read-write user based access (default = y): y

Configuring: rwuser
Description:
  a SNMPv3 read-write user
    arguments:  user [noauth|auth|priv] [restriction_oid]

The SNMPv3 user that should have read-write access: tim
The minimum security level required for that user [noauth|auth|priv,
default = auth]:
The OID that this community should be restricted to [if appropriate]:

Finished Output: rwuser  tim
Do another rwuser line? (default = y):n
Do you want to allow SNMPv3 read-only user based access (default = y): n
Do you want to allow SNMPv1/v2c read-write community access (default =
y): y

Configuring: rwcommunity
Description:
  a SNMPv1/SNMPv2c read-write access community name
    arguments:  community [default|hostname|network/bits] [oid]

Enter the community name to add read-write access for: secret
The hostname or network address to accept this community name from
[RETURN for all]:
The OID that this community should be restricted to [RETURN for no-
restriction]:

Finished Output: rwcommunity  secret
Do another rwcommunity line? (default = y): n
Do you want to allow SNMPv1/v2c read-only community access (default = y):
y

Configuring: rocommunity
Description:
  a SNMPv1/SNMPv2c read-only access community name
    arguments:  community [default|hostname|network/bits] [oid]

The community name to add read-only access for: public
The hostname or network address to accept this community name from
[RETURN for all]:
The OID that this community should be restricted to [RETURN for no-
restriction]:
```

```
Finished Output: rocommunity  public
Do another rocommunity line? (default = y): n
****************************************
*** Beginning trap destination setup ***
****************************************
Do you want to configure where and if the agent will send traps? (default
= y): y
Do you want the agent to send snmp traps on snmp authentication failures?
(default = y):

Configuring: authtrapenable
Description:
  Should we send traps when authentication failures occur
    arguments: 1 | 2   (1 = yes, 2 = no)

Should traps be sent when authentication failures occur? (1=yes, 2=no): 1

Finished Output: authtrapenable  1

Configuring: trapcommunity
Description:
  Default trap sink community to use
    arguments: community-string

The default community name to use when sending traps: public

Finished Output: trapcommunity  public
Do you want the agent to send snmpv2c informs to a trap receiver (default
= y): y

Configuring: informsink
Description:
  A SNMPv2c inform (acknowledged trap) receiver
    arguments: host [community] [portnum]

A host name that should receive the trap: snmp1
The community to be used in the trap sent [optional]: public
The port number the trap should be sent to [optional]: 161

Finished Output: informsink  snmp1 public 161
Do another informsink line? (default = y): n
Do you want the agent to send snmpv2c traps to a trap receiver (default =
y): n
****************************************
*** Beginning monitoring setup ***
```

```
****************************************
Do you want to configure the agent's ability to monitor various aspects
of your system? (default = y): y
Do you want to configure the agents ability to monitor processes?
(default = y): y

Configuring: proc
Description:
  Check for processes that should be running.
      proc NAME [MAX=0] [MIN=0]

      NAME:  the name of the process to check for.  It must match
             exactly (ie, http will not find httpd processes).
      MAX:   the maximum number allowed to be running.  Defaults to 0.
      MIN:   the minimum number to be running.  Defaults to 0.

    The results are reported in the prTable section of the UCD-SNMP-MIB
tree
    Special Case:  When the min and max numbers are both 0, it assumes
    you want a max of infinity and a min of 1.

Name of the process you want to check on: httpd
Maximum number of processes named 'httpd' that should be running [default
= 0]: 10
Minimum number of processes named 'httpd' that should be running [default
= 0]: 5

Finished Output: proc  httpd 10 5
Do another proc line? (default = y): n
Do you want to configure the agents ability to monitor disk space?
(default = y): y

Configuring: disk
Description:
  Check for disk space usage of a partition.
    The agent can check the amount of available disk space, and make
    sure it is above a set limit.

     disk PATH [MIN=100000]

     PATH:  mount path to the disk in question.
     MIN:   Disks with space below this value will have the Mib's
errorFlag set.
            Can be a raw byte value or a percentage followed by the %
            symbol.  Default value = 100000.
    The results are reported in the dskTable section of the UCD-SNMP-MIB
tree

Enter the mount point for the disk partion to be checked on: /export/home
Enter the minimum amount of space that should be available on
/export/home: 1500000

Finished Output: disk  /export/home 1500000
Do another disk line? (default = y): n
Do you want to configure the agents ability to monitor load average?
(default = y): y

Configuring: load
Description:
  Check for unreasonable load average values.
    Watch the load average levels on the machine.

     load [1MAX=12.0] [5MAX=12.0] [15MAX=12.0]

     1MAX:   If the 1 minute load average is above this limit at query
             time, the errorFlag will be set.
     5MAX:   Similar, but for 5 min average.
     15MAX:  Similar, but for 15 min average.

    The results are reported in the laTable section of the UCD-SNMP-MIB
tree

Enter the maximum allowable value for the 1 minute load average: 15
Enter the maximum allowable value for the 5 minute load average: 15
Enter the maximum allowable value for the 15 minute load average: 15

Finished Output: load  15 15 15
Do another load line? (default = y): n
Do you want to configure the agents ability to monitor file sizes?
(default = y): y

Configuring: file
Description:
  Check on the size of a file.
    Display a files size statistics.
    If it grows to be too large, report an error about it.

     file /path/to/file [maxsize_in_bytes]

       if maxsize is not specified, assume only size reporting is needed.
    The results are reported in the fileTable section of the UCD-SNMP-MIB
tree

Enter the path to the file you wish to monitor: /export/home
Enter the maximum size (in bytes) allowable for /export/home: 10000000

Finished Output: file  /export/home 10000000
Do another file line? (default = y): n


The following files were created:

  snmpd.conf installed in /var/tmp
```

Note – The `snmpd.conf` file is not required for the agent to start.

Adding USM Users

To create an SNMPv3 user, first stop the agent:

```
# /etc/init.d/init.sma stop
```

Then use the `/usr/sfw/bin/net-snmp-config` script to create a user. The script can be used to create read-write and read-only users.

```
# /usr/sfw/bin/net-snmp-config --create-snmpv3-user
Enter a SNMPv3 user name to create:
tim
Enter authentication pass-phrase:
12345678
Enter encryption pass-phrase:
  [press return to reuse the authentication pass-phrase]

adding the following line to /var/sma_snmp/snmpd.conf:
   createUser tim MD5 "12345678" DES
adding the following line to /etc/sma/snmp/snmpd.conf:
   rwuser tim
```

The following example passes the parameters on the command line:

```
# /usr/sfw/bin/net-snmp-config --create-snmpv3-user /
-ro -a 12345678 -X DES -A MD5 timw
```

```
adding the following line to /var/sma_snmp/snmpd.conf:
   createUser timw MD5 "12345678" DES
adding the following line to /etc/sma/snmp/snmpd.conf:
   rouser timw
```

Now the agent must be started:

```
# /etc/init.d/init.sma start
```

Configuring SMA Applications

The `snmpget`(1M) application can be used to test the user:

```
# /usr/sfw/bin/snmpget -v 3 -u timw -n "" -l authNoPriv /
-a MD5 -A 12345678 localhost sysUpTime.0
DISMAN-EVENT-MIB::sysUpTimeInstance = Timeticks: (227095) 0:37:50.95
```

There is a set of common options used with the agent application. For this command, the common options used are as follows:

- `-v` – Specifies the protocol version to use.
- `-u` – Security name used for authenticated SNMPv3 messages.
- `-n` – Destination context name used for SNMPv3 messages.
- `-l` – Security level used for SNMPv3 messages.
- `-a` – Authentication protocol used for authenticated SNMPv3 messages.
- `-A` – Authentication passphrase used for authenticated SNMPv3 messages.

Most of the agent application uses the common options, and they can be stored in the `snmp.conf`(4) file.

For example, the follow entries in the `/etc/sma/snmp/snmp.conf` file allow the user to execute the same `snmpget` command using the directives from the `snmp.conf`(4) file. Options passed on the command line override directives in the `snmp.conf` file.

```
defVersion 3
defSecurityName timw
defPassphrase 12345678
defContext ""
defAuthType MD5
defSecurityLevel authNoPriv
```

The command after the `snmp.conf` file is populated with the common option directives:

```
# /usr/sfw/bin/snmpget localhost sysUpTime.0
DISMAN-EVENT-MIB::sysUpTimeInstance = Timeticks:
(227095) 0:37:50.95
```

The `snmp.conf` file accepts other configuration directives for tools. The `doDebugging` directive turns on debug mode when set to `1`. When a tool is used, the debug information is sent to the `STDERR` output stream.

Configuring VACM

VACM is used to determine authorization of a user to read or write some part of the MIB tree. VACM uses four tokens in the `snmpd.conf` file:

- `view` – Defines portions of the MIB structure
- `com2sec` – Maps a community or source to a security name
- `group` – Defines the security model and security name or user
- `access` – Determines what group has access to which view

Figure 11-4 shows the VACM process:

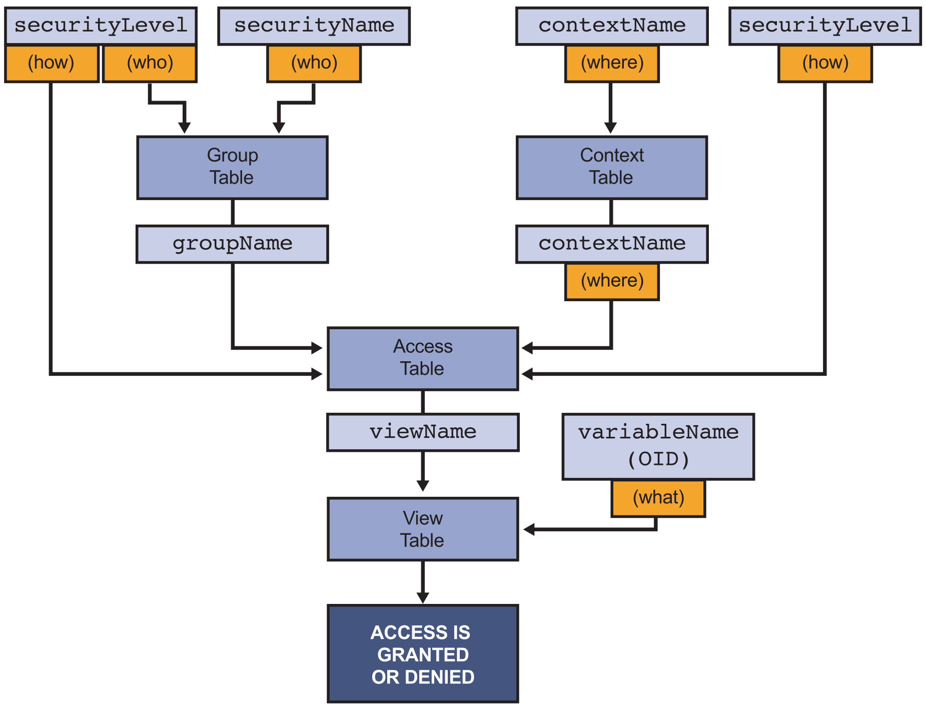


Figure 11-4 VACM Flowchart

VACM can be configured by using tokens in the `snmpd.conf` file.

In the `snmpd.conf` file, define the views to which demogroups are restricted (format: `view` *`NAME TYPE SUBTREE [MASK]`*):

```
view WriteView included .1.3.6.1.4.1.2021.14.1.1
view ReadView  included .1.3.6.1.4.1.2021.14.1.1
view ReadView  included .1.3.6.1.2.1.1
```

Map any SNMPv1 or SNMPv2 communities to a security name using the format `com2sec NAME SOURCE COMMUNITY`. For example:

```
com2sec v2cUser default demopublic
```

Define the group users and their access models using the format `group NAME MODEL SECURITY`. For example:

```
group demogroup usm timw
group demogroup v2c v2cUser
```

Define the access to a view using the format `access NAME CONTEXT MODEL LEVEL PREFX READ WRITE NOTIFY`. For example:

```
access demogroup "" any noauth prefix ReadView WriteView none
```

Test the view with the `snmpwalk` application:

```
# snmpwalk localhost /
SNMP-VIEW-BASED-ACM-MIB::vacmViewTreeFamilyTables
SNMP-VIEW-BASED-ACM-
MIB::vacmViewTreeFamilyMask."ReadView".7.1.3.6.1.2.1.1 = Hex-STRING: FF
SNMP-VIEW-BASED-ACM-
MIB::vacmViewTreeFamilyMask."ReadView".10.1.3.6.1.4.1.2021.14.1.1 = Hex-
STRING: FF FF
SNMP-VIEW-BASED-ACM-
MIB::vacmViewTreeFamilyMask."WriteView".10.1.3.6.1.4.1.2021.14.1.1 = Hex-
STRING: FF FF
SNMP-VIEW-BASED-ACM-MIB::vacmViewTreeFamilyMask."anonymousView000".1.1 =
Hex-STRING: FF
...
SNMP-VIEW-BASED-ACM-
MIB::vacmViewTreeFamilyStorageType."ReadView".7.1.3.6.1.2.1.1 = INTEGER:
permanent(4)
SNMP-VIEW-BASED-ACM-
MIB::vacmViewTreeFamilyStorageType."ReadView".10.1.3.6.1.4.1.2021.14.1.1
= INTEGER: permanent(4)
```

```
SNMP-VIEW-BASED-ACM-
MIB::vacmViewTreeFamilyStorageType."WriteView".10.1.3.6.1.4.1.2021.14.1.1
= INTEGER: permanent(4)
...
SNMP-VIEW-BASED-ACM-
MIB::vacmViewTreeFamilyStatus."ReadView".7.1.3.6.1.2.1.1 = INTEGER:
active(1)
SNMP-VIEW-BASED-ACM-
MIB::vacmViewTreeFamilyStatus."ReadView".10.1.3.6.1.4.1.2021.14.1.1 =
INTEGER: active(1)
SNMP-VIEW-BASED-ACM-
MIB::vacmViewTreeFamilyStatus."WriteView".10.1.3.6.1.4.1.2021.14.1.1 =
INTEGER: active(1)
```

Figure 11-5 shows VACM security relationships:

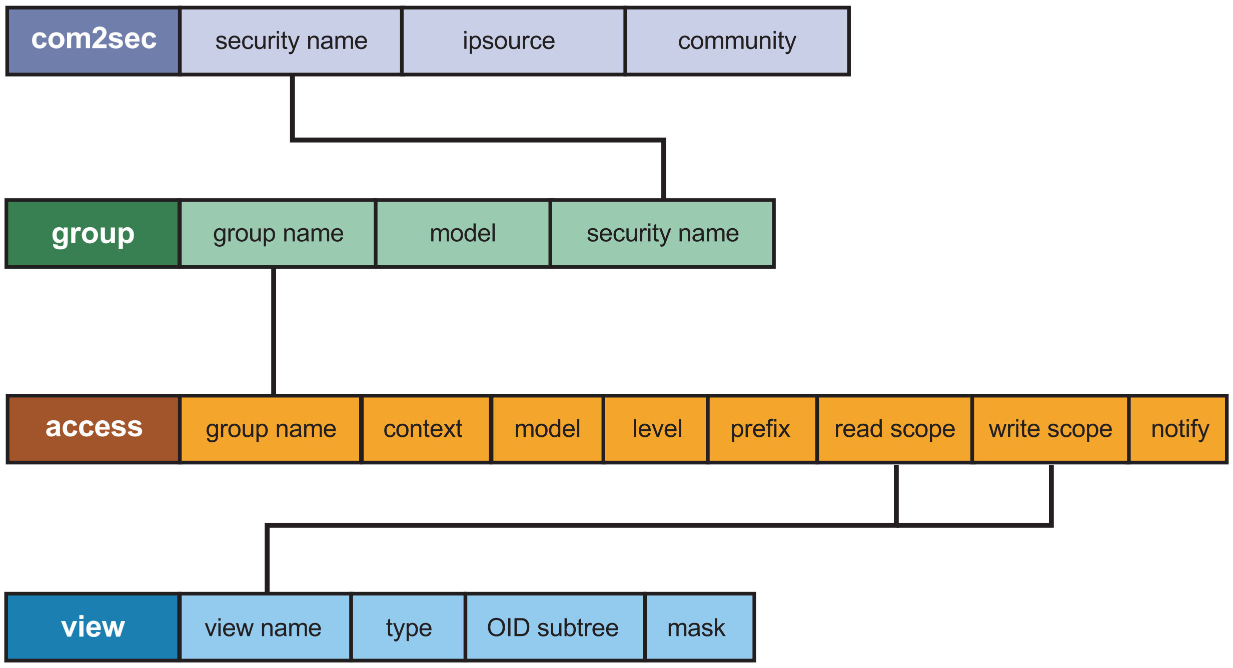


Figure 11-5VACM Security

Incorrectly configured access rights can lead to access being denied to key users. To remove a VACM configuration, the tokens should be removed from the `snmpd.conf` file and SMA restarted. The following `snmpvacm` commands can also be used to create, remove or change component of the VACM.

Use the `snmpvacm` command to add an access entry as follows:

```
# snmpvacm createAccess demogroup 3 2 2 ReadView WriteView
```

The first integer, `3`, represents the security model, in this case usm, while `2` would represent SNMPv2c and a `1` would be used for SNMPv1.

The second integer, `2`, is the minimum security level allowed, in this case authNoPriv, meaning that a user password is required. A `1` would indicate auth, which means only the user security name is checked. A `3` would represent authPriv, which requires a correct user name and password and requires the encryption key to encrypt the data stream.

The next integer, `2`, represents how the MIB context is matched. A `1` represents exact match while a `2` represents prefix, which allows a partially matched context string.

You can delete the access as follows:

```
# snmpvacm deleteAccess demogroup 3 2
```

The following example allows SMA to proxy incoming requests for JDMK to the proxied JDMK agent. In this example, a MIB is running on port `10161` and registers the MIB region `1.3.6.1.4.1.42.5000` under the context jdmkMib. The user, named `SecureUser`, must also exist in the JDMK USM. The `SecureUser` user must allow a security level of auth with Hashed Message Authentication Codes with Message-Digest algorithm 5 (HMACMD5) as the authentication algorithm. Add the following statement to the `snmpd.conf` file.

```
proxy --Cn jdmkMib -v3 -a MD5 -u SecureUser -l authNopriv -
A 12345678 localhost:10161 1.3.6.1.4.1.42.5000.2
```

DHCP Command Changes

The `dhcpconfig`(1M) command and the `dhtadm`(1M) command have had options added to them. The `dhcpagent`(1M) command has been enhanced to use the `client-ID` property in the `network-boot-arguments` property of the OpenBoot™ Programmable Read Only Memory (OBP).

Changes to the `dhcpconfig(1M)` Command

The `/etc/inet/dhcpsvc.conf` file is used to save service parameters for DHCP servers. The documentation for the file indicates that user should not edit the file manually. Rather, users should use the `dhcpconfig`(1M) command or the `dhcpmgr`(1M) graphical user interface (GUI).

The `dhcpconfig` command did not have facilities to change the service parameters and could not enable, disable, query, or re-enable the DHCP service.

Allow the setting of parameters in the `dhcpsvc`(4) file from the `dhcpconfig`(1M) command line:

A command line option has been added that takes a parameter keyword and a value. The keyword is checked against the internal list of valid keywords. Valid keywords are written back to the `dhcpsvc.conf`(4) file.

The following is from the `dhcpconfig`(1M) command man page:

`-P` Option – Configure the DHCP service parameters. The following pattern specifies each parameter and value:

`*parameter*[=*value*],...`

Where:

- *`parameter`* – One of the DHCP service parameters listed in the `dhcpsvc.conf`(4) file. If the corresponding value is not specified, the current parameter value appears. If parameter is not specified, all parameters and current values appear.
- *`value`* – Optional string to which to set the servers parameter if the value is acceptable. If the value is missing or is empty (""), the parameter and its current value are deleted.

After a parameter has changed, the DHCP server requires re-starting before you can use new parameter values.

Use the `-P` option for the `dhcpconfig`(1M) command to set, read, and clear parameter values as follows:

Note – For a list of possible parameters, see the `dhcpsvc.conf`(4) man page.

```
# dhcpconfig -P VERBOSE=TRUE
```

Invalid keywords return an appropriate error message.

```
# dhcpconfig -P BAD_PARAMETER=THIS_WILL_FAIL
dhcpconfig: Error - invalid DHCP server parameter BAD_PARAMETER.
```

Omitting the value for the parameter results in the current value of the parameter appearing.

```
# dhcpconfig -P VERBOSE
TRUE
```

Omitting the parameter results in the current value of all parameters appearing.

```
# dhcpconfig -P
DAEMON_ENABLED TRUE
RESOURCE SUNWfiles
RUN_MODE server
PATH /export/dhcp
CONVER 1
VERBOSE TRUE
```

Setting the parameter to an empty string (“”) results in the removal of the keyword from the `dhcpsvc.conf` file:

```
# dhcpconfig -P VERBOSE=""
# dhcpconfig -P VERBOSE
dhcpconfig: DHCP server parameter VERBOSE is not set.
```

Attempting to assign an invalid value to a parameter, or to write to a read-only parameter, results in an error message appearing.

```
# dhcpconfig -P LOGGING_FACILITY=8
dhcpconfig: Error - the value 8 is unacceptable for DHCP server parameter
LOGGING_FACILITY.
# dhcpconfig -P CONVER=2
dhcpconfig: Error - the DHCP server parameter CONVER cannot be changed.
```

Enabling and Disabling the DHCP Server Using the `dhcpconfig`(1M) Command

A command line option has been added that can be used to enable, disable, re-enable and query the DHCP server. Any change in state is written through to the `dhcpsvc.conf`(4) file parameter, `DAEMON_ENABLED`, and the DHCP server is appropriately started or stopped.

The following is from the `dhcpconfig`(1M) command man page:

`-S` Option – Control the DHCP service. The following sub-options are supported:

- `-d` – Disable and stop the DHCP service.
- `-e` – Enable and start the DHCP service.
- `-q` – Display the state of the DHCP service. The state is encoded into the exit status:

  `0` – DHCP service disabled and stopped

  `1` – DHCP service enabled and stopped

  `2` – DHCP service disabled and running

  `3` – DHCP service enabled and running
- `-r` – Enable and restart the DHCP service.

The `-S` command line option can be used to enable, disable, re-enable and query the DHCP service. Any change in state is written through to the `dhcpsvc.conf`(4) file parameter `DAEMON_ENABLED` and the DHCP service is appropriately started or stopped.

Enable the server as follows:

```
# dhcpconfig -S -e
DHCP server enabled.
DHCP server started.
```

Query the current state of the server as follows:

```
# dhcpconfig -S -q
DHCP server enabled.
DHCP server running.
```

Disable the server as follows:

```
# dhcpconfig -S -d
```

```
DHCP server disabled.
DHCP server shutdown.
```

Re-enable a disabled server as follows:

```
# dhcpconfig -S -r
DHCP server enabled.
DHCP server started.
```

Re-enable a running server as follows:

```
# dhcpconfig -S -r
DHCP server shutdown.
DHCP server started.
```

The re-enable (or restart) command line option is implemented as a shortcut for performing a disable and then enable. After a service parameter is changed, the new value becomes active only after the DHCP server is restarted.

The internal interfaces or methods for controlling the DHCP service are already present within the existing Java™ classes in use by the `dhcpconfig`(1M) and `dhcpmgr`(1M) commands. Therefore, the `libdhcpsvc.so` library required no changes.

Sending a SIGHUP `signal`(3HEAD) Using the `dhcpconfig`(1M) and `dhtadm`(1M) Commands

A new command line option is added to send a SIGHUP signal(3HEAD) to the `in.dhcpd`(1M) daemon after the specified operation completes. This is used when an operation updates the contents of the `dhcptab`(4) file and the `in.dhcpd`(1M) daemon must be updated.

For the `dhtadm`(1M) command, `-g` is a sub-option available with all of the main options (`-A`, `-B`, `-C`, `-D`, `-M`, `-P` or `-R`). For the `dhcpconfig`(1M) command, `-g` is a sub-option available only with the `-N`, `-X`, and `-I` options, which are the only options that affect the contents of the `dhcptab`(4) file and do not otherwise stop or restart the `in.dhcpd`(1M) daemon.

The next command does the following:

- Imports DHCP data from a file, (`/net/chaos/export/var/120301710_data`, containing data previously exported from a Solaris OS DHCP server

- Overwrites any conflicting data on the importing server
- Signals the daemon to reload the `dhcptab`(4) file once the import completes

```
# dhcpconfig -I /net/golduck/export/var/120301710_data -f -
g
```

The following command runs a series of `dhtadm`(1M) commands contained in a batch file and signals the daemon to reload the `dhcptab`(4) file once the commands are executed:

```
# dhtadm -B addmacros -g
```

The dhcpagent(1M) Command Uses the OBP `client-ID` Property

Version 4.1.4 and newer of the OBP let the user specify a desired `client-id` value either as an argument to the OBP `tftpboot` package or as a key and value attribute pair in the `network-boot-argument` property. The OBP publishes a `client-ID` property under the `/chosen` directory with the value that the user specifies.

When the `dhcpagent`(1M) command is used with the `-a` option, the `dhcpagent`(1M) command uses the `client-id` published as `/chosen:client-id` instead of any `/etc/default/dhcpagent` value or the default. If there is no `client-id` property under the `/chosen` directory, the `dhcpagent`(1M) command follows its legacy behavior.

Changes to Routing Sockets

Sockets are used as endpoints for communication. Process make connections to sockets using `connect`(3SOCKET) function. Changes were required to the stream socket when a `connect`(3SOCKET) function was used and to a special socket called the routing socket, described in the `route`(7P) man page. These changes were implemented to increase security and conform to current standards, Single UNIX Specification Version 3 (SUS3).

The Connect Problem

When a `connect(s, NULL, 0)` function is sent to a SOCK_DGRAM type socket, the connection is dissolved. However, when this null `connect`(3SOCKET) is used with a connection oriented socket of type SOCK_STREAM, it should fail with an error message indicating the connection was not terminated.

In the case of Solaris OS, the null `connect`(3SOCKET) function was allowed to succeed. The result is insecure in that it unbinds the Internet Protocol (IP) address while leaving the port still bound to the previous port number. This would allow a unprivileged process called by, say, the `rsh`(1M) command to unbind its standard input (STDIN) IP address and *listen* on the `rsh` port. Also, this behavior did not conform to SUS3.

The Routing Socket Problem

The kernel maintains a routing information database, which is used in selecting the appropriate network interface when transmitting packets. A user process maintains this database by sending messages over a routing socket.

Read-only operations of the `route`(1M) command, such as RTM_GET or monitor, should be allowed for all users. This was not possible because the `route`(1M) command checks for uid=`0`, the routing socket device minor permission entry is `rts:rts 0660 root sys`, and the `route`(1M) command interface requires root privilege for any access.

Due to the device permission, the routing socket should not allow unprivileged access, but that was not always the case as the routing socket code did not check for privileged access. Exploits of this would allow an unprivileged process to alter the contents of the kernel routing table.

The Connect Solution

A change was made to the `connect`(3SOCKET) function that prevents the `connect`(s, NULL, 0) function from succeeding. Instead, an error code was added, EISCONN (`The socket is already connected.`), and the binding of the socket is not changed. As a result of this change, the interface correctly conforms to the documented standard, SUS3.

The Routing Socket Solution

The device permissions for the routing socket were changed from `rts:rts 0660 root sys` to `rts:rts 0666 root sys`. Also, the code to implement a socket was fixed to ensure that only an RTM_GET request is allowed if an unprivileged process opens the stream. The `route`(1M) command was changed to no longer check for uid=`0`, and instead relies on the routing socket for the outcome of the request. Changes to the `route`(7P) command added an error code, EPREM, that is sent if the user does not have enough security credentials to alter the routing table.

The `route`(1M) command man page was altered to indicate that RTM_GET is allowed for non-privileged users and the `route`(7P) command man page was altered to document the new error code, EPERM, that is sent if the user does not have enough security credentials the alter the routing table.

These changes conform to the Least Privilege project (PSARC 2002/188) and to the standard, SUS3.