Silicon Graphics, Inc.

subject: **System Call Security Analysis**

date: **February 10, 1995**

from: **Casey Schaufler**

ABSTRACT

This document describes the security policies enforced by each of the Trusted IRIX/B system calls. It details how the calls are supposed to work. Any observed deviation from this description is an error in the system implementation which must be corrected.

Trusted IRIX/B - System Call Security Analysis

1. Task Scope

This analysis is intended to describe the security policy enforcement mechanisms which are used by each of the Trusted IRIX/B system calls. For each system call the differences in behavior between the Superuser and normal users, if any, are described. System audit requirements are not addressed here.

2. Trusted IRIX/B System Subjects

A subject in the Trusted IRIX/B System can be either a process, or a share group (shared process group). A process consists of a single thread of execution in its own isolated address space, while a share group consists of multiple threads of execution in a shared address space. However, these are essentially the same thing in that a single process can be viewed as merely the degenerate instance of a share group.

2.1 Process

2.1.1 Attributes

From a security standpoint a process as a Subject has the following attributes:

- Process ID
- Process Group ID
- Session ID
- UNIX DAC Subject Attributes:
  - Real User ID
  - Effective User ID
  - Saved User ID
  - Real Group ID
  - Effective Group ID
  - Saved Group ID
  - Group Membership List
  - File creation DAC bit mask (umask)
  - Close-on-exec flag [see *exec*(2)]
- MAC Label
- Audit Session ID, the User ID for audit records (set at login, cannot be changed)
- Audit capture event mask (controls which audit events will be captured)
- Socket Access Control List (used to set Socket ACL at socket creation)

Note that the list of Subject attributes is a proper subset of the list of attributes for the process as an Object. This list consists of the attributes which affect some aspect of the process acting as a Subject. For example, the signal masks are not used by the process as a Subject (sending signals), but are of importance to the process as an Object (receiving signals).

2.1.2 Data

As a Subject, the process is an active agent, and therefore does not have data like an Object. Processes, of course, are also Objects. The data for the Object aspects of a process is discussed in Section 3.1.2, "Process Object". In a sense, the Subject aspects of the process use the Object aspects by executing the text pages of the process and loading and storing data from the data and stack pages.

2.1.3 Names

The name of the Process Subject is the numeric process ID. To meet the requirements for association of auditable actions to be associated with an individual ADP system user, the system saves the User ID used to log in successfully and uses this saved ID for all audit records. This Audit Session ID is effectively a name of the Subject for the Audit subsystem.

2.1.4 Manipulation

A process is created with the *fork*(2) system call. (As discussed below, the *sproc*(2) call creates an additional member of a share group, which is implemented as a distinct process sharing the parent's address space. From a security standpoint, the entire share group is the subject.)

As a Subject, the process can be destroyed by issuing the *exit*(2) system call. A process as a Subject can also be destroyed by its own actions such as issuing an illegal memory reference, but the actual destruction mechanism occurs by treating the process as an Object for the receipt of an operating system generated signal. In this example, the process generates the illegal memory reference, the operating system generates a signal (SIGSEGV) as a result, and if this has not been arranged to be caught by the process as an Object, the operating system will terminate the process and generate a core image. These same signals may be sent from other processes. Signal handling is discussed in detail below, in Section 5.3.1.4.2.

fork(2) causes creation of a new "child" **process** which is an exact copy of the calling "parent" **process**. Among the information copied into the child are all the Subject Security Attributes discussed above, with the following exceptions:

- The child process has a unique process ID. As discussed below in the section on the Process Object, the process ID values are recycled, but the system guarantees that all active values are unique.
- The child process has a different parent process ID (i.e., the process ID of the parent process).

2.2 Share Group

Share groups are a collection of threads of execution which share a common address space. From a kernel implementation standpoint, each member of the share group is a full fledged process, having both a u vector and a process table instance (*struct proc*) (see Section 3.1.2 for a discussion of these structures). However, from a trusted systems standpoint, members of a share group are not isolated from each other, so must be treated as a single entity for protection purposes. Since the address space is common, as, for example, share group member A reads data into a buffer, it is immediately visible to share group member B, even though it is possible that share group member B never opened the file.

Member processes of a share group are created with the *sproc*(2) system call, a variant of the standard *fork*(2) system call. Like *fork*(2), *sproc*(2) creates a new **process** that is a clone of the process that called *sproc*(2). The difference is that after an *sproc*(2), the new "child" **process** potentially shares the virtual address space of the "parent" **process**, rather than simply being a copy of the "parent". The "parent" and the "child" each have their own program counter value and stack pointer, but all the text and data space is visible to both processes.

The origin of share groups resulted from the need to create algorithms which exploited multi-processor hardware by dividing up a calculation into pieces which could be computed in parallel on the individual processors of a multi-processor system. For a simple example, consider filling a matrix with 0's. This can be done in parallel because the results of one calculation do not depend on those of any other calculations. The result of suitably parallelizing the algorithm are an N-times speed up with N processors. (Real world examples are vastly more complex but share the same conceptual foundation.) This simple example illustrates why the share group must share the same address space; it would do no good to zero segments of a matrix in separate address spaces.

Share groups have been referred to as "lightweight processes", but that term has come to mean a slightly different concept and will not be used here.

2.2.1 Attributes

Share groups have the same attributes as processes as Subjects. Each share group member has a distinct process ID.

2.2.2 Data

Share groups have the same data as processes as Subjects.

2.2.3 Names

Each share group member has a distinct process ID. The Audit Session ID is set in the same way as it is for a process.

2.2.4 Manipulation

The semantics of share groups introduced above lead to some slight additional complexity in their manipulation. As mentioned, a single process is a degenerate instance of a share group. Additional process Objects within the share group are created by the *sproc*(2) system call. So, strictly speaking, the share group Subject comes into existence with the *fork* system call. This means that the *sproc* call does **NOT** create a new Subject; it creates a new process Object within the same share group Subject.

New share groups can also be created in a second way. Since a member of a share group is a full process, it is possible for one process in a share group to issue the *exec*(2) system call which breaks its association with the shared address space. This establishes a new share group, with that process as its only member.

To implement the share group semantics, it is necessary for each member process to have a private stack in the shared address space. (Here, "private" in the sense that though possibly visible to all members of the share group, manipulation of the stack by any process other than the owner could lead to incorrect operation.) Each member process also has a private Process Data Area (PRDA) which is used by Trusted IRIX/B processes to communicate with the system scheduler. In addition to virtual address space, members of a share group can share other attributes such as file tables, current working directories, effective user IDs and others described below.

The new "child" **process** resulting from *sproc*(2) differs from a normally forked process in the following ways:

- The child's stack is set to a virtual address that doesn't overlap the stack of the parent process. There is a maximum stack size different from the maximum allowable amount of virtual space per process. This value may be read and set using *prctl*(2) or *setrlimit*(2).
- If the PR_SADDR bit is set in *inh* then the new process will share ALL the virtual space of the parent, except the PRDA (see below). During a normal *fork*(2), the writable portions of the process's address space are marked copy-on-write. If either process writes into a given page, then a copy is made of the page and given to the process. Thus, writes by one process will not be visible to the other forks. With the PR_SADDR option of *sproc*(2), however, all the processes have read/write privileges to the entire virtual space.
- The new process can reference the parent's stack.
- The new process has its own PRDA which contains, among other things, the process ID. Part of the PRDA is used by the system, part by system libraries, and part is available to the application program [see <sys/**prctl.h**>]. The PRDA is at a fixed virtual address in each process which is given by the constant PRDA defined in **prctl.h**.
- The machine state (general/floating point registers) is not duplicated with the exception of the floating point control register. This means that if a process has enabled floating point traps, these will be enabled in the child process.
- The new process will be invoked as follows: **entry(arg)**

In addition to the attributes inherited during the *sproc* call itself, the *inh* flag to *sproc* can request that the new process have future changes in any member of the share group be applied to itself. A process can only request that a child process share attributes that it itself is sharing. The creator of a share group is effectively sharing everything. These persisting attributes are selectable via the *inh* flag:

PR_SADDR All virtual space attributes (shared memory, mapped files, data space) are shared. If one process in a share group attaches to a shared memory segment, all processes in the group can access that

segment.

PR_SFDS The open file table is kept synchronized. If one member of the share group opens a file, the open file descriptor will appear in the file tables of all members of the share group.

Note especially that the converse is also true: if one member closes a file, it is closed for all members of the share group. Note also that there is only one file pointer for each file descriptor shared within a shared process group.

PR_SDIR The current and root directories are kept synchronized. If one member of the group issues a *chdir*(2) or *chroot*(2) call, the current working directory or root directory will be changed for all members of the share group.

PR_SUMASK The file creation mask, *umask*, is kept synchronized.

PR_SULIMIT The limit on maximum file size is kept synchronized.

PR_SID The real and effective User and Group IDs are kept synchronized.

To take advantage of sharing all possible attributes, the constant PR_SALL may be used.

In addition to specifying shared attributes, the *inh* flag can be used to pass flags that govern certain operations within the *sproc*)(2) call itself. Currently one flag is supported, PR_BLOCK, which causes the calling process to be blocked [see *blockproc*(2)] before returning from a successful call. This can be used to allow the child process access to the parent's stack without the possibility of collision.

No scheduling synchronization is implied between shared processes: they are free to run on any processor in any sequence. Any required synchronization must be provided by the application using locks and semaphores [see *usinit*(3P)] or other mechanisms.

If one member of a share group exits or otherwise dies, its stack is removed from the virtual space of the share group. In addition, if the PR_SETEXITSIG option [see *prctl*(2)] has been enabled then all remaining members of the share group will be signaled.

Here are some respects in which the two types of processes are the same:

- The parent and child after an *sproc* each have a unique process ID, but are in the same process group.
- A signal sent to a specific process ID in a share group [see *kill*(2)] will be received by only the process to which it was sent. Other members of the share group will not be affected. A signal sent to an entire process group will be received by all the members of the process group, regardless of share group affiliations [see *killpg*(3B)].
- If the child process resulting from an *sproc* dies or calls *exit*(2), the parent process receives the SIGCLD signal [see *sigset*(2), *sigaction*(2), and *sigvec*(3B)].

3. Trusted IRIX/B System Kernel Objects

- Named Objects
  - File System Object
    - Block Device
    - Character Device
    - Directory
    - FIFO
    - File
    - Pipe
    - Symbolic Link
    - UNIX Domain Rendezvous Point
    - Process "file"
  - Process Object
    - Current Process
    - Another Process
  - System V IPC Object
    - Message Queue
    - Semaphore Set
    - Shared Memory Segment
  - BSD IPC Socket Object
- Public Objects
  - System Clock
  - Cycle Counter
  - Domain Name
  - Host Name
  - Host ID
  - Non-Volatile RAM
  - Scheduler
- Constant Objects
  - Page Size
  - Descriptor Table Size

At the highest level, Trusted IRIX/B System kernel objects can be grouped into three major categories:

- Named Objects
- Public Objects
- Constant Objects

Each of these categories is discussed in greater detail in the following sections.

3.1 Named Objects

Named objects are defined as objects which are directly manipulable at the TCB interface. In particular, named objects must have meaning to more than one subject. Named objects can be further divided into the following:

- File System Object
- Process Object
- System V IPC Object
- BSD IPC Socket Object

Each of the named objects in the Trusted IRIX/B system comprises two distinct parts, each of which is kept in one or more distinct storage objects. The first part consists of attributes associated with the object, which includes access control information. The second part consists of the actual data stored in the object. A generic named object would be as described in Figure 3.1-1.

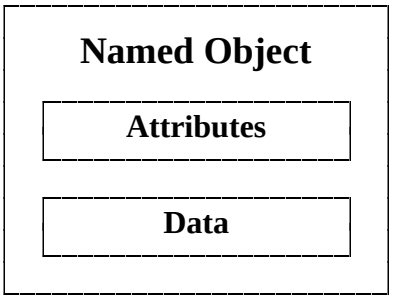

Figure 3.1-1. Named Object

3.1.1 File System Object

A *file system object* would be as described in Figure 3.1.1-1.

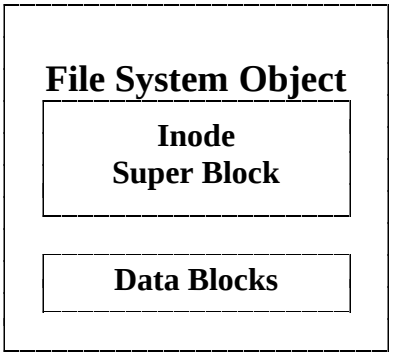

Figure 3.1.1-1. File System Object

3.1.1.1 Attributes

The attributes associated with a *file system object* are stored in its **inode** and the **super-block** of it's containing file system. These attributes consist of `stat`, `statfs`, and

`mac_label` structures.

The security relevent attributes are the MAC label (the `mac_label` returned by *getlabel*(2)), the user id of the file's owner (the `st_uid` field of the `stat` structure returned by *stat*(2)), the group id of the file's group (the `st_gid` field of the `stat` structure), and the file mode (the `st_mode` field of the `stat` structure).

3.1.1.2 Data

The data associated with a *file system object* is stored in zero or more **data blocks** that it contains. Data may be structured or arbitrary, depending on the type of *file system object*. If the data is structured, the structure is enforced by the TCB.

3.1.1.3 Names

A *file system object* can be named in one of the following ways:

- By a file descriptor (*fchmod,ftruncate)*
- By a path name (*stat,readlink)*

3.1.1.4 Manipulation

A *file system object* (with the exception of a directory, pipe, or process "file") can be deleted with the following system call:

- *unlink*(2)

Attributes associated with a *file system object* (with the exception of a pipe, symbolic link, or process "file") can be read with any of the following system calls:

- *access*(2)
- *fstat*(2)
- *fstatfs*(2)
- *lstat*(2)
- *stat*(2)
- *statfs*(2)
- *syssgi*(SGI_GETLABEL)(2)

Attributes associated with a *file system object* which is the root directory of a file system can be read with the following system call:

- *utssys*(ustat)(2)

Attributes associated with a *file system object* (with the exception of a pipe, symbolic link, or process "file") can be written with any of the following system calls:

- *chmod*(2)
- *chown*(2)

- *fchmod*(2)
- *fchown*(2)
- *syssgi*(SGI_SETLABEL)(2)
- *utime*(2)

The data associated with a *file system object* (with the exception of a symbolic link or UNIX Domain Rendezvous Point) can be read with the following system call:

- *read*(2)

The data associated with a *file system object* (with the exception of a symbolic link or UNIX Domain Rendezvous Point) can be written with the following system call:

- *write*(2)

3.1.1.4.1 Block Device

A *file system object* of type **block device** is created with the following system call:

- *mknod*(2)

Although allowed by the policy, some devices may not implement *read*(2) or *write*(2).

3.1.1.4.2 Character Device

A *file system object* of type **character device** is created with the following system call:

- *mknod*(2)

Although allowed by the policy, some devices may not implement *read*(2) or *write*(2).

3.1.1.4.3 Directory

A *file system object* of type **directory** is created with the following system call:

- *mkdir*(2)

A *file system object* of type **directory** is deleted with the following system call:

- *rmdir*(2)

The data associated with a *file system object* of type **directory** may be read with the following system call:

- *chdir*(2)
- *getdents*(2)

The data associated with a *file system object* of type **directory** may be written with any of the following system calls:

- *bind*(2) (for UNIX Domain sockets only)
- *creat*(2)

- *link*(2)
- *mkdir*(2)
- *mkfifo*(2)
- *mknod*(2)
- *open*(2)
- *rename*(2)
- *rmdir*(2)
- *symlink*(2)
- *sysacct*(2)
- *unlink*(2)

3.1.1.4.4 FIFO

A *file system object* of type **FIFO** is created with the following system call:

- *mknod*(2)

3.1.1.4.5 File

A *file system object* of type **file** is created with any of the following system calls:

- *creat*(2)
- *open*(2)
- *sysacct*(2)

The data associated with a *file system object* of type **file** may be written with any of the following system calls:

- *ftruncate*(2)
- *truncate*(2)

3.1.1.4.6 Pipe

A *file system object* of type **pipe** is created with the following system call:

- *pipe*(2)

A pipe has no path name associated with it. The *pipe* call returns two file descriptors which may be used in the *fstat*, *fstats*, *fchmod*, and *fchown* calls.

A *file system object* of type **pipe** is deleted when the file descriptors returned at its creation are all closed. The *pipe* becomes unusable when the last reader closes the read side file descriptor. Subsequent writes will generate SIGPIPE signals to the process doing the *write*.

Since pipes do not have names, their attributes can be read only with the following system calls:

- *fstat*(2)
- *fstats*(2)

These attributes are not used.

The attributes associated with a *file system object* of type **pipe** can be written only with the following system calls:

- *fchmod*(2)
- *fchown*(2)

Although the above system calls alter the attributes of the pipe, which can be observed with a subsequent *fstat* call, these attributes are not used by the system since the pipe is already open.

3.1.1.4.7 Symbolic Link

A *file system object* of type **symbolic link** may be created with any of the following system calls:

- *link*(2)
- *rename*(2)
- *symlink*(2)

The attributes associated with a *file system object* of type **symbolic link** can be read only with the following system call:

- *lstat*(2)

The attributes associated with a *file system object* of type **symbolic link** cannot be written.

The data associated with a *file system object* of type **symbolic link** can be read with the following system call:

- *readlink*(2)

The data associated with a *file system object* of type **symbolic link** cannot be written after it is created.

3.1.1.4.8 UNIX Domain Rendezvous Point

A *file system object* of type **UNIX Domain Rendezvous Point** is created with the following system call:

- *bind*(2)

No data is ever associated with a *file system object* of type **UNIX Domain Rendezvous Point**. Data can be neither read from nor written to this *file system object*.

A bit of discussion is required here. The *file system object* known as a **UNIX Domain Rendezvous Point** is used strictly as a container of extended attribute information for a

BSD IPC UNIX Domain socket. The *file system object* is in fact not actually a significant part of the data transfer mechanism. All it provides is a handle into the file system name space to use with the BSD IPC mechanism.

3.1.1.4.9 Process File

A *file system object* of type **process file** is the mechanism by which the data and attributes of a process can be manipulated through the file system. The name of a **process file** is its process ID represented in five characters of left zero filled ASCII. **Process file** *file system objects* exist in the */debug* file system, which is a mountable file system. Once mounted, each process currently active in the system appears as an entry in */debug* whose name is the ASCII representation of the process ID filled with leading zeros to five ASCII characters so they will sort numerically. Subject to the usual file system MAC and DAC checks, this file may be opened and the **process file** accessed via the file descriptor.

A *file system object* of type **process file** is created with any of the following system calls:

- *fork*(2)
- *sproc*(2)

A *file system object* of type **process file** is dependent upon existence of the associated process and therefore cannot be deleted via a system call.

The attributes associated with a *file system object* of type **process file** may be read and written with any of the following system calls:

- *fcntl*(2)
- *ioctl*(2)

The attributes associated with a **process file** are those of the underlying process. Because this view of the process is via the file system, the attributes are instantiated like those of any other file system object. **Process file** objects have an inode allocated, and are filled in from the attributes of the underlying process. The DAC access is set to owner read/write (0600). The owner and group are set to the saved (effective) User ID and Group ID of the file currently being executed. If the process uses the *exec*(2) system call to invoke a setuid/setgid program, the system invalidates any current file descriptors to prevent access to protected images. The attributes may be manipulated with *fcntl* and *ioctl* calls with command arguments that are specific to the **process file** object type. These operations require an open file descriptor.

The data associated with a *file system object* of type **process file** may be read with the following system call:

- *read*(2)

The data associated with a *file system object* of type **process file** may be written with the following system call:

- *write*(2)

The data of the **process file** object is the same as the data of the associated *process* object. Standard file system operations like read and write can be used to access the data of the process.

3.1.2 Process Object

A *process object* would be as described in Figure 3.1.2-1.

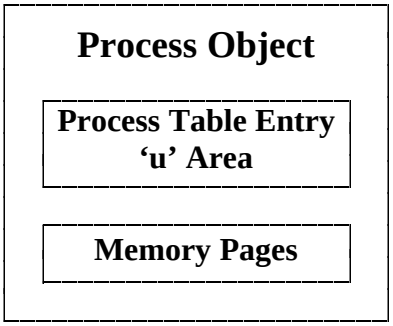

Figure 3.1.2-1. Process Object

3.1.2.1 Attributes

The attributes associated with a *process object* are stored in a pair of kernel data structures, the **Process Table Entry** and the **'u' area**. The relevant attributes of a *process object* are the audit ID, process ID, real, effective, and saved user IDs, session ID, process group ID, group ID, group access list, process socket ACL, and process label.

When quotas are enabled (which they are not in the Evaluated Configuration, this discussion is for completeness only), the quota structures for each file system, in effect, become process attributes, as they are checked when files are created or enlarged.

3.1.2.2 Data

The data associated with a *process object* is stored in the set of **memory pages** that it owns. The data consists of a set of linked **memory pages**.

3.1.2.3 Names

A *process object* is named by a numeric process ID. This name is unique on a machine at any given time. However, the names are recycled. A *process object* always knows it's own name and in many cases can manipulate itself implicitly. A process which invokes *brk*(2) enlarges the memory size of a process. Which process? The current process, of

course. Although the name was never explicitly given, is was implicitly known.

3.1.2.4 Manipulation

3.1.2.4.1 Current Process

A *process object* is deleted with the following system call:

- *exit*(2)

The attributes associated with a *process object* can be read by that process with any of the following system calls:

- *BSDgetpgrp*(2)
- *getgid*(2)
- *getpid*(2)
- *getuid*(2)
- *setpgrp*(2)
- *syssgi*(2) called with any of the following arguments:
  - SGI_GETGROUPS
  - SGI_GETPLABEL
  - SGI_GETPSOACL

The attributes associated with a *process object* can be written by that process with any of the following system calls:

- *BSDsetpgrp*(2)
- *setgid*(2)
- *setpgrp*(2)
- *setregid*(2)
- *setreuid*(2)
- *setuid*(2)
- *syssgi*(2) called with any of the following arguments:
  - SGI-SETGROUPS
  - SGI_SETPGID
  - SGI_SETPLABEL
  - SGI_SETPSOACL
  - SGI_SETSID

The data associated with a *process object* can be read and written directly by that process.

3.1.2.4.2 Another Process

A new *process object* is created by its parent with the following system call:

- *fork*(2)
- *sproc*(2)

The attributes associated with a *process object* can be read by another process with any of the following system calls:

- *BSDgetpgrp*(2)
- *syssgi*(2) called with any of the following arguments:
  - SGI_PROCSZ
  - SGI_RDNAME
  - SGI_RDUBLK

The attributes associated with a *process object* can be written by another process with any of the following system calls:

- *BSDsetpgrp*(2)
- *kill*(2)
- *procblk*(2)
- *ptrace*(2) called with any of the following arguments:
  - PTRC_CONTINUE
  - PTRC_STEP
  - PTRC_TERMINATE
- *syssgi*(SGI_SETPGID)(2)

The data associated with a *process object* can be read by another process with the following system call:

- *ptrace*(2) called with any of the following arguments:
  - PTRC_RD_I
  - PTRC_RD_D
  - PTRC_RD_REG
  - PTRC_RD_GPRS
  - PTRC_RD_FPRS

The data associated with a *process object* can be written by another process with the following system call:

- *ptrace*(2) called with any of the following arguments:
  - PTRC_WR_I
  - PTRC_WR_D
  - PTRC_WR_REG
  - PTRC_WR_GPRS
  - PTRC_WR_FPRS

3.1.3 System V IPC Object

A *System V IPC object* would be as described in Figure 3.1.3-1.

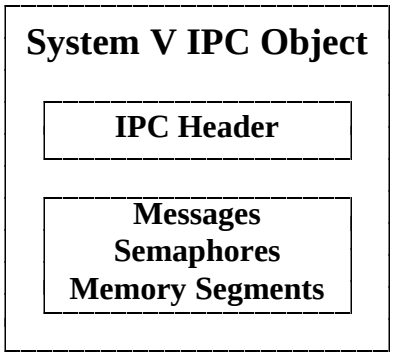

Figure 3.1.3-1. System V IPC Object

3.1.3.1 Attributes

The attributes associated with a *System V IPC object* are stored in its **IPC header**. The attributes associated with a *System V IPC object* are stored in system memory. The attributes associated with a *System V IPC object* are the owning and creating user IDs, the owning group ID, the access bits, and the object label.

3.1.3.2 Data

The data associated with a *System V IPC object* is stored in system memory. For *System V IPC objects* of type **message queue** or **semaphore set**, this memory is in the address space of the kernel. For *System V IPC objects* of type **shared memory segment**, this memory is in the address space of the process accessing the **shared memory segment**. Data may be structured or arbitrary, depending on the type of *System V IPC object*. If the data is structured, the structure is enforced by the TCB.

3.1.3.3 Names

The attributes associated with a *System V IPC object* can be named in one of the following ways:

- With a KEY
- With an ID

The data associated with a *System V IPC object* can be named in the following way:

- With an ID

3.1.3.4 Manipulation

3.1.3.4.1 Message Queue

A *System V IPC object* of type **message queue** is created with the following system call:

- *msgsys*(MSGGET)(2)

A *System V IPC object* of type **message queue** may be deleted with the following system call:

- *msgsys*(MSGCTL)(2) called with the following option:
  — IPC_RMID

The attributes associated with a *System V IPC object* of type **message queue** may be read with the following system call:

- *msgsys*(MSGCTL)(2) called with the following option:
  — IPC_STATE

The attributes associated with a *System V IPC object* of type **message queue** may be written with the following system call:

- *msgsys*(MSGCTL)(2) called with the following option:
  — IPC_SET

The data associated with a *System V IPC object* of type **message queue** may be read with the following system call:

- *msgsys*(MSVRCV)(2)

The data associated with a *System V IPC object* of type **message queue** may be written with the following system call:

- *msgsys*(MSGSND)(2)

3.1.3.4.2 Semaphore Set

A *System V IPC object* of type **semaphore set** is created with the following system call:

- *semsys*(SEMGET)(2)

A *System V IPC object* of type **semaphore set** may be deleted with the following system call:

- *semsys*(SEMCTL)(2) called with the following option:
  — IPC_RMID

The attributes associated with a *System V IPC object* of type **semaphore set** may be read with the following system call:

- *semsys*(SEMCTL)(2) called with the following option:
  — IPC_STATE

The attributes associated with a *System V IPC object* of type **semaphore set** may be written with the following system call:

- *semsys*(SEMCTL)(2) called with the following option:
  — IPC_SET

The data associated with a *System V IPC object* of type **semaphore set** may be read with any of the following system calls:

- *semsys*(SEMCTL)(2) called with any of the following options:
  — GETALL
  — GETCNT
  — GETPID
  — GETVAL
  — GETZCNT
- *semsys*(SEMOP)(2) called with the following option:
  — sem_op = 0

The data associated with a *System V IPC object* of type **semaphore set** may be written with any of the following system calls:

- *semsys*(SEMCTL)(2) called with any of the following options:
  — SETALL
  — SETVAL
- *semsys*(SEMOP)(2) called with the following option:
  — sem_op ≠ 0

3.1.3.4.3 Shared Memory Segment

A *System V IPC object* of type **shared memory segment** is created with the following system call:

- *shmsys*(SHMGET)(2)

A *System V IPC object* of type **shared memory segment** may be deleted with the following system call:

- *shmsys*(SHMCTL)(2) called with the following option:
  — IPC_RMID

The attributes associated with a *System V IPC object* of type **shared memory segment** may be read with the following system call:

- *shmsys*(SHMCTL)(2) called with the following option:
  — IPC_STATE

The attributes associated with a *System V IPC object* of type **shared memory segment** may be written with the following system call:

- *shmsys*(SHMCTL)(2) called with any of the following options:
  — IPC_SET
  — SHM_LOCK
  — SHM_UNLOCK

The data associated with a *System V IPC object* of type **shared memory segment** may be manipulated directly by the process or with the following system call:

- *shmsys*(SHMAT)(2)

3.1.4 BSD IPC Socket Object

A *BSD IPC socket object* would be as described in Figure 3.1.4-1.

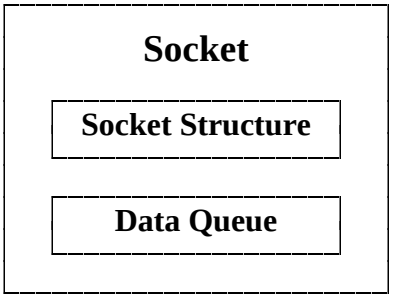

Figure 3.1.4-1. BSD IPC Socket Object

Sockets offer a wide range of options to the programmer. These include choice of the transport mechanism (domain) (in the evaluated configuration, this choice is between the Internet Domain and the UNIX Domain), and the characteristics of the socket, i.e., stream or datagram. Generally, these choices have little impact on the security decisions in the system, as the security checks are made at higher levels of abstraction, independent of these choices. Where options do have an impact, they are discussed. For example, the access control policies on write (sending data) are impacted by the choice of which transport mechanism is to be used, so this is discussed.

3.1.4.1 Attributes

The attributes associated with a *BSD IPC socket object* are stored in its **socket structure**. These attributes include information about the socket user ID, type, supporting protocol, socket state, and the transport type. (The socket user ID is initialized to be the Effective User ID of the socket creator. It can be altered only by the Superuser, as discussed below.)

For sockets defined in the UNIX domain, there is also an associated UNIX Domain Rendezvous Point, which is described above as a File System Object. As described below, the attributes of this file are used in making a Data Write Access Policy - Path Name decision on the file as part of the Data Write Access Policy - Sockets.

3.1.4.2 Data

The data associated with a *BSD IPC socket object* is stored in its **data queue** and is the data queued for receipt. The **data queue** contains only data which has passed the access checks dictated by the Data Write Access Policy - Sockets, described below.

3.1.4.3 Names

A *BSD IPC socket object* can be named in the following way:

- By a file descriptor

3.1.4.4 Manipulation

A *BSD IPC socket object* can be created with any of the following system calls:

- *accept*(2) (applicable only for stream protocols)
- *socket*(2)
- *socketpair*(2)

A *BSD IPC socket object* can be deleted with any of the following system calls:

- *close*(2)
- *exit*(2)

The attributes associated with a *BSD IPC socket object* can be read with any of the following system calls:

- *fcntl*(2)
- *getpeername*(2)
- *getsockname*(2)
- *getsockopt*(2)
- *ioctl*(2)
- *select*(2)

The attributes associated with a *BSD IPC socket object* can be written with any of the following system calls:

- *accept*(2) (applicable only for stream protocols)
- *bind*(2)
- *connect*(2) (applicable only for stream protocols)
- *fcntl*(2)
- *ioctl*(2)
- *listen*(2) (applicable only for stream protocols)
- *setsockopt*
- *shutdown*(2)
- *socket*(2)
- *socketpair*(2)

The data associated with a *BSD IPC socket object* can be read with any of the following system calls:

- *read*(2)
- *recv*(2)
- *recvfrom*(2)
- *recvmsg*(2)
- *syssgi*(SGI_RECVLMSG)(2)
- *syssgi*(SGI_RECVLUMSG)(2)

The data associated with a *BSD IPC socket object* can be written with any of the following system calls:

- *send*(2)
- *sendmsg*(2)
- *sendto*(2)
- *write*(2)

3.2 Public Objects

Public objects cannot be created with any system call. All *public objects* are created implicitly at system generation. *Public objects* cannot be deleted with any system call. All *public objects* are deleted implicitly at system termination. There are no attributes associated with a *public object*. *Public objects* are globally readable. Writing to a *public object* requires special (Superuser) privilege.

3.2.1 System Clock

The **system clock** can be read with any of the following system calls:

- *alarm*(2)
- *getitimer*(2)
- *bsdgettime*(2)
- *setitimer*(2)

The **system clock** can be modified with any of the following system calls:

- *adjtime*(2)
- *stime*(2)
- *sysmips*(STIME)(2)
- *sysmp*(MP_CLOCK)(2)
- *syssgi*(2) called with any of the following arguments:
  - SGI_DISABLE_FASTCLOCK
  - SGI_DISABLE_FTIMER
  - SGI_ENABLE_FASTCLOCK
  - SGI_ENABLE_FTIMER
  - SGI_GETTIMETRIM
  - SGI_QUERY_FTIMER

  - SGI_SETTIMEOFDAY
  - SGI_SETTIMETRIM
- *time*(2)
- *times*(2)

3.2.2 Cycle Counter

The **cycle counter** public object is a very fast, free running counter that may be mapped into the process address space. It cannot be set by software. The address to be used is returned with the following system call:

- *syssgi*(SGI_QUERY_CYCLECNTR)(2)

3.2.3 Domain Name

The **domain name** is not used in Trusted IRIX/B. The **domain name** can be read with the following system call:

- *getdomainname*(2)

The **domain name** can be written with the following system call:

- *setdomainname*(2)

3.2.4 Host Name

The **host name** can be read with the following system call:

- *gethostname*(2)

The **host name** can be written with the following system call:

- *sethostname*(2)

3.2.5 Host ID

The **host ID** can be read with the following system call:

- *gethostid*(2)

The **host ID** can be written with the following system call:

- *sethostid*(2)

3.2.6 Non-Volatile RAM

The system **non-volatile RAM** can be read with any of the following system calls:

- *sgikopt*(2)
- *syssgi*(SGI_GETNVRAM)(2)

The system **non-volatile RAM** can be written with the following system call:

- *syssgi*(SGI_SETNVRAM)(2)

3.2.7 Scheduler

A process may alter its own scheduling behavior with the *schedctl*(2) system call.

- *schedctl*(2)

3.3 Constant Objects

Constant objects are just like *public objects* with the exception that they cannot be written, only read. While they do not actually qualify as storage objects, they are included here for completeness.

3.3.1 Page Size

The system **page size** is returned by the system call:

- *getpagesize*(2)

3.3.2 Descriptor Table Size

The per process file **descriptor table size** limit is returned by the system call:

- *getdtablesize*(2)

4. Object Reuse

4.1 File System Objects

File system objects are all given fully specified attribute information upon creation. File system objects which contain fixed data (symbolic links and UNIX Domain Rendezvous Points) are initialized to contain only that data appropriate to their purpose. File system objects which contain data directly (files, directories, FIFOs) are created empty. File system objects which provide interfaces to devices (character and block devices) are dependent on the specific device involved. Those file system objects which grow (files and directories) are grown only to the extent necessary, with the data specified included. In the rare case of files with "holes", the data between the explicitly defined areas is zero filled.

4.2 Process Objects

Process objects are created only by the *fork*(2) system call, which creates a new process which is an identical copy (except for the call's return status) of the parent process, and the *sproc*(2) system call, which creates a new process Object sharing the same address space as the parent process. *sproc* creates a new process Object within the share group, but does **NOT** create a new Subject. (This subtle point is discussed above in Section 2.2 on Share Group Subjects.) Process growth (*brk*(2) and the process stack) is always done into zero-filled memory. Attribute information is always completely initialized for the new process.

4.3 System V IPC Objects

All System V IPC objects are volatile, and disappear on system restart. Semaphore sets have each individual semaphore set to zero. Shared memory segments are zeroed. Message queues are empty.

4.4 BSD IPC Socket Objects

BSD IPC Sockets are volatile, and disappear on system restart. They are fully initialized upon creation.

5. Policies

For this analysis to be valid it must be assumed that all of the system calls have in fact been implemented in such a way as to uniformly enforce the policies described. The primary mechanism for determining that the policy described is actually implemented is testing, combined with source code analysis.

5.1 File System Policies

5.1.1 Path Name Policies

5.1.1.1 Data Read Access Policy - Path Name

The **data read access policy** for system calls specifying a path name is to allow access if and only if the label of the process dominates the label of all of the directories in the path, the file permission bits allow a user with the process' user ID execute access to each of the directories in the path, the label of the process dominates the label of the file, and the file permission bits of the file allow a user with the process' user ID read access to the file.

5.1.1.2 Data Write Access Policy - Path Name

The **data write access policy** for system calls specifying a path name is to allow access if and only if the label of the process dominates the label of all of the directories in the path, the file permission bits allow a user with the process' user ID execute access to each of the directories in the path, the label of the process equals the label of the file, and the file permission bits of the file allow a user with the process' user ID write access to the file.

If the process is owned by the Superuser and running with the *System High Sensitivity Label*, and the Integrity Label of the process equals the Integrity Label of the file, the process may obtain data write access to the file even if the Sensitivity Label of the file is not the *System High Sensitivity Label*.

If the operation is to remove a directory entry and the sticky bit is set on the directory, and the object named by the entry is not writeable by the owner of the process and the process is owned by neither the directory's owner nor the owner of the object named by the entry nor the Superuser write access may be denied.

5.1.1.3 Attribute Read Access Policy - Path Name

The **attribute read access policy** for system calls specifying a path name is to allow access if and only if the label of the process dominates the label of all of the directories in the path, the file permission bits allow a user with the process' user ID execute access

to each of the directories in the path, and the label of the process dominates the label of the file.

5.1.1.4 Attribute Write Access Policy - Path Name

The **attribute write access policy** for system calls specifying a path name is to allow access if and only if the label of the process dominates the label of all of the directories in the path, the file permission bits allow a user with the process' user ID execute access to each of the directories in the path, the file is owned by the process' user ID, and the label of the process equals the label of the file.

If the process is owned by the Superuser and running with the *System High Sensitivity Label*, and the Integrity Label of the process equals the Integrity Label of the file, the process may obtain attribute write access to the file even if the Sensitivity Label of the file is not the *System High Sensitivity Label*.

In addition, if the request is to change the label attribute associated with a file, then there is the additional requirement that the newly requested label must be contained within the clearance range defined for the user on whose behalf the process is functioning.

5.1.1.5 Execute Access Policy - Path Name

The **execute access policy** for system calls specifying a path name is to allow access if and only if the label of the process dominates the label of all of the directories in the path, the file permission bits allow a user with the process' user ID execute access to each of the directories in the path, the label of the process dominates the label of the file, and the file permission bits of the file allow a user with the process' user ID execute access to the file.

5.1.2 File Descriptor Policies

5.1.2.1 Read Access Policy - File Descriptor

The **read access policy**, for either data or attributes, for system calls specifying a file descriptor is to allow reading.

5.1.2.2 Data Write Access Policy - File Descriptor

The **data write access policy** for system calls specifying a file descriptor is to allow writing if and only if write access was requested when the file descriptor was assigned.

5.1.2.3 Attribute Write Access Policy - File Descriptor

The **attribute write access policy** for system calls specifying a file descriptor is to allow writing if and only if the file is owned by the process' user ID. (The file need not have been opened for writing.)

5.2 Intraprocess Communication Policies

5.2.1 Data Read and Write Access Policy

A process can always read and write its own data in user address space. No system call is required to do this.

5.2.2 Attribute Read Access Policy

A process can always read its own attributes. No access control decisions are ever made.

5.2.3 Attribute Write Access Policy

Most process attributes either can never be written (i.e., audit ID, process ID), or can always be written (i.e., process group ID, process socket ACL), so that in those cases no access control decisions are ever made. However, five process attributes (i.e., user ID, group ID, session ID, group access list, and process label) are governed by specific attribute write access policies as follows:

5.2.3.1 User ID Write

If the process is owned by the Superuser, it can change its real and effective user ID. Otherwise, a process can change its real or effective user ID only if the requested value is equal to its real or saved user ID.

5.2.3.2 Group ID Write

If the process is owned by the Superuser, it can change its real and effective group ID. Otherwise, a process can change its real or effective group ID only if the requested value is equal to its real or saved group ID.

5.2.3.3 Session ID Write

A process can change its session ID, provided that it is not a process group leader and that no process other than the calling process has a process group ID that matches the calling process' process ID.

5.2.3.4 Group Access List Write

Only a process owned by the Superuser can change its group access list.

5.2.3.5 Label Write

If the process is owned by the Superuser, it can change its label. Otherwise, a process can change its label only if the requested label and the current label compare as equal.

5.3 Interprocess Communication Policies

5.3.1 Process Policies

5.3.1.1 Data Read Access Policy - Process

A process can read another process's data in user address space if and only if the label of the subject process dominates the label of the object process and the object process is a child of the subject process.

5.3.1.2 Data Write Access Policy - Process

A process can write another process's data in user address space if and only if the label of the subject process equals the label of the object process and the object process is a child of the subject process.

If the subject process is owned by the Superuser and running with the *System High Sensitivity Label*, and the Integrity Label of the subject process equals the Integrity Label of the object process, the subject process may obtain data write access to the object process even if the Sensitivity Label of the object process is not the *System High Sensitivity Label*.

5.3.1.3 Attribute Read Access Policy - Process

A process can read another process's attributes if and only if the label of the subject process dominates the label of the object process and if the subject process can provide the name (Process ID) of the object process when requesting the attributes.

5.3.1.4 Attribute Write Access Policy - Process

Most process attributes cannot be written by another process (i.e., audit ID, process ID, user ID, session ID, group ID, group access list, process socket ACL, and process label). Those that can (i.e., process group ID and sending a signal) are governed by specific

policies as follows:

5.3.1.4.1 Process Group ID Write

A process can change the process group ID of another process if and only if the label of the subject process equals the label of the object process and one of the following conditions is true:

- The subject process is owned by the Superuser.
- Both processes are members of the same session.
- The object process is a child of the subject process.

If the subject process is owned by the Superuser and running with the *System High Sensitivity Label*, and the Integrity Label of the subject process equals the Integrity Label of the object process, the subject process may obtain attribute write access to the object process even if the Sensitivity Label of the object process is not the *System High Sensitivity Label*.

5.3.1.4.2 Signal

A process can send a signal or a process block to another process if and only if one of the following conditions is true:

- The subject process is owned by the Superuser.
- The label of the subject process equals the label of the object process and the real or effective user ID of the subject process matches the real or saved user ID of the object process.

Additionally, a process can send a signal to another process if one of the following conditions is true:

- The label of the subject process equals the label of the object process and the object process is a child of the subject process.
- The label of the subject process equals the label of the object process and the signal is SIGCONT and the object process is a descendant of the subject process.
- The label of the subject process equals the label of the object process and the signal is SIGCONT and both the subject and object processes are members of the same session.

5.3.2 System V IPC Policies

5.3.2.1 KEYs

5.3.2.1.1 Attribute Read Access Policy - System V IPC

The **attribute read** access policy for System V IPC - **KEYs** is to allow access (that is, return an ID) if and only if the label of the process equals the label of the object.

If the process is owned by the Superuser and running with the *System High Sensitivity Label*, and the Integrity Label of the process equals the Integrity Label of the IPC object, the process may obtain read access to the IPC object's ID even if the label of the IPC object is not the *System High Sensitivity Label*.

5.3.2.1.2 Attribute Write Access Policy - System V IPC

The **attribute write** access policy for System V IPC - **KEYs** is a limited attribute write in that it only allows for creation of a new System V IPC object. The policy is that any one can create a new System V IPC object.

5.3.2.2 IDs

5.3.2.2.1 Read Access Policy - System V IPC

The **read** access policy, for either data or attributes, for System V IPC - **IDs** is to allow access if and only if the label of the process equals the label of the object and the mode bits of the object allow read access.

If the process is owned by the Superuser and running with the *System High Sensitivity Label*, and the Integrity Label of the process equals the Integrity Label of the IPC object, the process may obtain read access to the IPC object even if the label of the IPC object is not the *System High Sensitivity Label*.

5.3.2.2.2 Data Write Access Policy - System V IPC

The **data write** access policy for System V IPC - **IDs** is to allow access if and only if the label of the process equals the label of the object and the mode bits of the object allow write access.

If the process is owned by the Superuser and running with the *System High Sensitivity Label*, and the Integrity Label of the process equals the Integrity Label of the IPC object, the process may obtain data write access to the IPC object even if the label of the IPC object is not the *System High Sensitivity Label*.

5.3.2.2.3 Attribute Write Access Policy - System V IPC

The **attribute write** access policy for System V IPC - **IDs** is to allow access if and only if the label of the process equals the label of the object and the requesting process is the object owner or creator. The exception to this is the *msg_qbytes* attribute associated with

a message queue. Only the Superuser can change the *msg_qbytes* attribute of a System V IPC object of type message queue.

If the process is owned by the Superuser and running with the *System High Sensitivity Label*, and the Integrity Label of the process equals the Integrity Label of the IPC object, the process may obtain attribute write access to the IPC object even if the label of the IPC object is not the *System High Sensitivity Label*.

5.3.3 BSD IPC Policies

5.3.3.1 Read Access Policy - BSD IPC

The **read access policy**, for either data or attributes, with respect to the BSD IPC socket object type is to allow access if and only if the requesting process is the socket object creator or one of its descendants.

5.3.3.2 Data Write Access Policy - BSD IPC

The **data write access policy** with respect to the BSD IPC socket object type is to allow access (i.e., permit the write to complete) if and only if:

- (for UNIX Domain Sockets only) the Data Write Access Policy for Path Name allows access by the requesting process to the UNIX Domain Rendezvous Point associated with the receiving socket
- the label of the requesting process is equal to the label of the receiving socket
- the Socket Access Control List associated with the receiving socket permits access by the effective user ID (at time of creation) of the sending socket

The Socket Access Control List associated with the receiving socket will permit access if and only if one (or more) of the following conditions hold:

- the requesting process is owned by the Superuser
- the effective user ID (at time of creation) of the sending socket is contained in the Socket Access Control List
- the Socket Access Control List is set to the WILDCARD value

Only the Superuser can create a socket in the Internet address domain, which is set up for system management purposes (raw socket interface) and receives copies of all packets delivered on a specified network interface, and does not make any of the above checks.

5.3.3.3 Attribute Write Access Policy - BSD IPC

The **attribute write access policy** with respect to the BSD IPC socket object type is to allow access if and only if the requesting process is the socket object creator or one of its descendants. The exceptions to this are the socket user ID and the label attributes. Only the Superuser process can change the user ID or label of a socket it has created.

5.4 Public Object Policies

The **data read access policy** for *public objects* is to allow access.

The **data write access policy** for *public objects* is to deny access. However, write access to public objects is permitted if the requesting process is owned by the Superuser.

5.5 Superuser Policies

The *Superuser* is the privileged user. The *Superuser* is permitted to modify *public objects*. The *Superuser* is permitted to modify the system configuration in a well defined set of ways.

6. Modules

For purposes of discussion, each system call will be considered a module. The source modules involved actually include everything under `usr/src/uts/mips`.

Although most system calls correspond directly to the library interfaces described in Section 2 manual pages, several system calls are common, more primitive implementations of basic functionality, and the implementation of the interfaces described in Section 2 manual pages is done in the library routines. In the following, this situation is indicated when the names for the "Interfaces" item are different from the section title, e.g. "procblock" has the interfaces "blockproc", "unblockproc", "blockprocall", and "unblockprocall".

There are a few system calls which are private interfaces intended to be used by a single external service, and which do not have corresponding Section 2 manual pages. These are documented here.

6.1 BSDgetpgrp

Interfaces: BSDgetpgrp(2)

Access Control Decisions: The Attribute Read Access Policy for Intraprocess Communication and the Process Attribute Read Access Policy for Interprocess Communication are enforced.

6.2 BSDsetpgrp

Interfaces: BSDsetpgrp(2)

Access Control Decisions: The Attribute Write Access Policy, Session ID Write, for Intraprocess Communication, and the Process Attribute Write Access Policy, Process Group ID Write, for Interprocess Communication are enforced.

6.3 accept

Interfaces: accept(2)

Access Control Decisions: The Attribute Write Access Policy for BSD IPC Socket object types is enforced.

Object Reuse Implications: The socket data structure is complete initialized.

6.4 access

Interfaces: access(2)

Access Control Decisions: The Attribute Read Access Policy for Path Name is enforced.

6.5 adjtime

Interfaces: adjtime(2)

Access Control Decisions: Only the Superuser is allowed to use this call.

6.6 alarm

Interfaces: alarm(2)

Access Control Decisions: No access control decisions are made.

6.7 async_daemon

Interfaces: async_daemon()
This is a private interface used by NFS service processes who donate their user context for NFS file system actions.

Access Control Decisions: Only the Superuser is allowed to use this call.

6.8 bind

Interfaces: bind(2)

Access Control Decisions: The Data Write Access Policy for Path Name is enforced on the directory in which the file (UNIX Domain Rendezvous Point) is to be placed, and the Attribute Write Access Policy for BSD IPC Socket object types is enforced.

Object Reuse Implications: The new object is created with zero length. All attribute information is initialized.

6.9 brk

Interfaces: brk(2) sbrk(2)

Access Control Decisions: No access control decisions are made.

Object Reuse Implications: The process address space is increased by this call. All new space is zero filled.

6.10 bsdgettime

Interfaces: gettimeofday(3b)

Access Control Decisions: No access control decisions are made.

6.11 cachectl

Interfaces: cachectl(2)

Access Control Decisions: No access control decisions are made.

6.12 cacheflush

Interfaces: cacheflush(2)

Access Control Decisions: No access control decisions are made.

6.13 chdir

Interfaces: chdir(2)

Access Control Decisions: The Data Read Access Policy for Path Name is enforced on the specified directory.

6.14 chmod

Interfaces: chmod(2)

Access Control Decisions: The Attribute Write Access Policy for Path Name is enforced.

6.15 chown

Interfaces: chown(2) BSDchown(2)

Access Control Decisions: The Attribute Write Access Policy for Path Name is enforced.

6.16 chroot

Interfaces: chroot(2)

Access Control Decisions: Only the Superuser is allowed to use this call. The Data Read Access Policy for Path Name is enforced on the specified directory.

6.17 close

Interfaces: close(2)

Access Control Decisions: No access control decisions are made.

6.18 connect

Interfaces: connect(2)

Access Control Decisions: The Attribute Write Access Policy for BSD IPC Socket object types is enforced.

6.19 creat

Interfaces: creat(2)

Access Control Decisions: The Data Write Access Policy for Path Name is enforced on the directory in which the file is to be placed. If the file already exists, the Data Write Access Policy for Path Name is enforced on the file.

Object Reuse Implications: The new object is created with zero length. All attribute information is initialized.

6.20 dup

Interfaces: dup(2)

Access Control Decisions: No access control decisions are made.

6.21 exec

Interfaces: execv(2) execve(2) execvp(2)
The *exec* system call is: exec(char * file, char *argv[])
The other interfaces mentioned in the manual pages are mapped onto this system call by the library.

Access Control Decisions: The Execute Access Policy for Path Name is enforced on the file.

6.22 exece

Interfaces: execl(2) execle(2) execlp(2)
The *exece* system call is: exece(char * file, char *argv[], char *arge[])
The other interfaces mentioned in the manual pages are mapped onto this call by the library.

Access Control Decisions: The Execute Access Policy for Path Name is enforced on the file.

6.23 exit

Interfaces: exit(2) _exit(2)

Access Control Decisions: There are no policy decisions unique to *exit*. Other processes may be *kill*ed as a side effect of this call. The security semantics of the *kill* call apply.

6.24 exportfs

Interface:
This is a private interface used by the *exportfs*(1m) utility.
The *exportfs* system call is: exportfs(char *dir, struct export *ex)
In essence, the contents of a line in the */etc/exports* file are placed in dir and ex. This call informs the NFS support in the kernel to accept requests for client access to the named file system.

Access Control Decisions: Only the Superuser is allowed to use this call.

6.25 fchdir

Interfaces: fchdir(2)

Access Control Decisions: The Data/Attribute Read Access Policy for File Descriptor is enforced.

6.26 fchmod

Interfaces: fchmod(2)

Access Control Decisions: The Attribute Write Access Policy for File Descriptor is enforced.

6.27 fchown

Interfaces: fchown(2) BSDfchown(2)

Access Control Decisions: The Attribute Write Access Policy for File Descriptor is enforced.

6.28 fcntl

Interfaces: fcntl(2)

Access Control Decisions: The Data/Attribute Read Access Policy for File Descriptor, and the Attribute Read and Attribute Write Access Policies for BSD IPC Socket object types are enforced for read and write operations, respectively.

6.29 fork

Interfaces: fork(2)

Access Control Decisions: No access control decisions are made.

Object Reuse Implications: A new process object is created which is a copy of the existing object. No part of the new object is uninitialized.

6.30 fstat

Interfaces: fstat(2)

Access Control Decisions: The Data/Attribute Read Access Policy for File Descriptor is enforced.

6.31 fstatfs

Interfaces: fstatfs(2)

Access Control Decisions: The Data/Attribute Read Access Policy for File Descriptor is enforced.

6.32 fsync

Interfaces: fsync(2)

Access Control Decisions: The Data/Attribute Read Access Policy for File Descriptor is enforced.

6.33 ftruncate

Interfaces: ftruncate(2)

Access Control Decisions: The Data Write Access Policy for File Descriptor is enforced.

6.34 getdents

Interfaces: getdents(2)

Access Control Decisions: The Data/Attribute Read Access Policy for File Descriptor is enforced.

6.35 getdomainname

Interfaces: getdomainname(2)

Access Control Decisions: No access control decisions are made.

6.36 getgid

Interfaces: getegid(2) getgid(2)

Access Control Decisions: The Attribute Read Access Policy for Intraprocess Communication is enforced.

6.37 gethostid

Interfaces: gethostid(2)

Access Control Decisions: No access control decisions are made.

6.38 gethostname

Interfaces: gethostname(2)

Access Control Decisions: No access control decisions are made.

6.39 getitimer

Interfaces: getitimer(2)

Access Control Decisions: No access control decisions are made.

6.40 getmsg

Interfaces: getmsg(2)

Access Control Decisions: The Data/Attribute Read Access Policy for File Descriptor is enforced.

6.41 getpagesize

Access Control Decisions: No access control decisions are made.

6.42 getpeername

Interfaces: getpeername(2)

Access Control Decisions: The Attribute Read Access Policy for BSD IPC Socket object types is enforced.

6.43 getpid

Interfaces: getpid(2) getppid(2)

Access Control Decisions: The Attribute Read Access Policy for Intraprocess Communication is enforced.

6.44 getrlimit

Interfaces: getrlimit(2)

Access Control Decisions: No access control decisions are made.

6.45 getsockname

Interfaces: getsockname(2)

Access Control Decisions: The Attribute Read Access Policy for BSD IPC Socket object types is enforced.

6.46 getsockopt

Interfaces: getsockopt(2)

Access Control Decisions: The Attribute Read Access Policy for BSD IPC Socket object types is enforced.

6.47 getuid

Interfaces: geteuid(2) getuid(2)

Access Control Decisions: The Attribute Read Access Policy for Intraprocess Communication is enforced.

6.48 ioctl

Interfaces: ioctl(2)

Only ioctls with security relevant behavior are addressed here. Please refer to Appendix C "Supported *ioctl* Summary" for a complete list of *ioctl*s within the evaluated configuration.

6.48.1 I_PEEK

Access Control Decisions: Only the Superuser is allowed to use this call.

6.48.2 I_STR

Access Control Decisions: Only the Superuser is allowed to use this call.

6.48.3 SIOCADDMULTI

Access Control Decisions: Only the Superuser is allowed to use this call.

6.48.4 SIOCADDRT

Access Control Decisions: Only the Superuser is allowed to use this call.

6.48.5 SIOCDARP

Access Control Decisions: Only the Superuser is allowed to use this call.

6.48.6 SIOCDELMULTI

Access Control Decisions: Only the Superuser is allowed to use this call.

6.48.7 SIOCDELRT

Access Control Decisions: Only the Superuser is allowed to use this call.

6.48.8 SIOCSARP

Access Control Decisions: Only the Superuser is allowed to use this call.

6.48.9 SIOCSETLABEL

Access Control Decisions: Only the Superuser is allowed to use this call.

6.48.10 SIOCSETRTINFO

Access Control Decisions: Only the Superuser is allowed to use this call.

6.48.11 SIOCSETUID

Access Control Decisions: Only the Superuser is allowed to use this call.

6.48.12 SIOCSIFADDR

Access Control Decisions: Only the Superuser is allowed to use this call.

6.48.13 SIOCSIFBRDADDR

Access Control Decisions: Only the Superuser is allowed to use this call.

6.48.14 SIOCSIFDSTADDR

Access Control Decisions: Only the Superuser is allowed to use this call.

6.48.15 SIOCSIFFLAGS

Access Control Decisions: Only the Superuser is allowed to use this call.

6.48.16 SIOCSIFHEAD

Access Control Decisions: Only the Superuser is allowed to use this call.

6.48.17 SIOCSIFLABEL

Access Control Decisions: Only the Superuser is allowed to use this call.

6.48.18 SIOCSIFMETRIC

Access Control Decisions: Only the Superuser is allowed to use this call.

6.48.19 SIOCSIFNETMASK

Access Control Decisions: Only the Superuser is allowed to use this call.

6.48.20 SIOCSIFSTATS

Access Control Decisions: Only the Superuser is allowed to use this call.

6.48.21 SIOCSIFUID

Access Control Decisions: Only the Superuser is allowed to use this call.

6.48.22 TCSETLABEL

Access Control Decisions: Only the Superuser is allowed to use this call.

6.48.23 TIOCSPGRP

Access Control Decisions: Only the Superuser is allowed to use this call.

6.48.24 TIOCSTI

Access Control Decisions: Only the Superuser is allowed to use this call.

6.49 kill

Interfaces: kill(2) raise (3c)

Access Control Decisions: The Process Attribute Write Access Policy, Signal, for Interprocess Communication is enforced.

6.50 link

Interfaces: link(2)

Access Control Decisions: The Data Write Access Policy for Path Name is enforced on the directory in which the link is to be placed.

Object Reuse Implications: The new object is created with zero length. All attribute information is initialized.

6.51 listen

Interfaces: listen(2)

Access Control Decisions: The Attribute Write Access Policy for BSD IPC Socket object types is enforced.

6.52 lstat

Interfaces: lstat(2)

Access Control Decisions: The Attribute Read Access Policy for Path Name is enforced.

6.53 madvise

Interfaces: madvise(2)

Access Control Decisions: No access control decisions are made.

6.54 mkdir

Interfaces: mkdir(2)

Access Control Decisions: The Data Write Access Policy for Path Name is enforced on the directory to which the directory is to be added.

Object Reuse Implications: The new object is created with zero length. All attribute information is initialized.

6.55 mknod

Interfaces: mkfifo(2) mknode(2)

Access Control Decisions: The Data Write Access Policy for Path Name is enforced on the directory in which the special file is to be placed. Only the Superuser is allowed to create anything other than FIFOs with this call.

Object Reuse Implications: The new object is created with zero length. All attribute information is initialized.

6.56 mmap

Interfaces: mmap(2)

Access Control Decisions: The Data/Attribute Read Access Policy for File Descriptor is enforced.

6.57 mount

Interfaces: mount(2)

Access Control Decisions: Only the Superuser is allowed to use this call. The Data Write Access Policy for Path Name is enforced on the directory on which the file system is to be mounted.

6.58 mprotect

This call is currently not implemented.

Interfaces: mprotect (no man page)

Access Control Decisions: No access control decisions are made.

6.59 msgsys

6.59.1 MSGCTL

Interfaces: msgctl(2)

6.59.1.1 IPC_RMID

Access Control Decisions: The System V IPC, IDs, Attribute Write Access Policy for Interprocess Communication is enforced.

6.59.1.2 IPC_SET

Access Control Decisions: The System V IPC, IDs, Attribute Write Access Policy for Interprocess Communication is enforced.

6.59.1.3 IPC_STATE

Access Control Decisions: The System V IPC, IDs, Read Access Policy for Interprocess Communication is enforced.

6.59.2 MSGGET

Interfaces: msgget(2)

Access Control Decisions: The System V IPC, KEYs, Attribute Read and Attribute Write Access Policies for Interprocess Communication are enforced.

Object Reuse Implications: The new object is created with zero length. All attribute information is initialized.

6.59.3 MSGRCV

Interfaces: msgrcv(2)

Access Control Decisions: The System V IPC, IDs, Read Access Policy for Interprocess Communication is enforced.

6.59.4 MSGSND

Interfaces: msgsnd(2)

Access Control Decisions: The System V IPC, IDs, Data Write Access Policy for Interprocess Communication is enforced.

6.60 msync

Interfaces: msync(2)

Access Control Decisions: No access control decisions are made.

6.61 munmap

Interfaces: munmap(2)

Access Control Decisions: No access control decisions are made.

6.62 nfs_getfh

Interface:
This call is a private interface for rpc.mountd (the NFS mount daemon). It returns the NFS file handle (an internal identifier) associated with the file name supplied as an argument. This is used to generate the reply message to the client requesting the NFS mount.
The *nfs_getfh* system call is: nfs_getfh(char * path, fhandle_t *fh)

Access Control Decisions: Only the Superuser is allowed to use this call.

6.63 nfssvc

Interface:
This is a private interface used by the NFS daemons to inform the kernel that they are available to be used for NFS purposes.
The *nfssvc* system call does not return and is: nfssvc()
NFS daemons "donate" their process context to the NFS service in the kernel to allow for NFS events to be waited for in a user process context. The kernel can only wait for something in a user process context.

Access Control Decisions: Only the Superuser is allowed to use this call.

6.64 nice

Interfaces: nice(2)

Access Control Decisions: No access control decisions are made.

6.65 open

Interfaces: open(2)

Access Control Decisions: The Data Read Access Policy for Path Name is enforced. If write access is requested the Data Write Access Policy for Path Name is enforced. If the file does not exist and the `O_CREAT` mode is specified, the Data Write Access Policy for Path Name is enforced on the directory in which the file is to be placed.

Object Reuse Implications: The new object is created with zero length. All attribute information is initialized.

6.66 pagelock

Interfaces: mpin(2) munpin(2)

Access Control Decisions: No access control decisions are made.

6.67 pause

Interfaces: pause(2)

Access Control Decisions: No access control decisions are made.

6.68 pipe

Interfaces: pipe(2)

Access Control Decisions: No access control decisions are made.

Object Reuse Implications: The new object is created with zero length. All attribute information is initialized.

6.69 plock

Interfaces: plock(2)

Access Control Decisions: Only the Superuser is allowed to use this call.

6.70 poll

Interfaces: poll(2)

Access Control Decisions: The Data/Attribute Read Access Policy for File Descriptor is enforced.

6.71 prctl

Interfaces: prctl(2)

Access Control Decisions: No access control decisions are made.

6.72 procblk

6.72.1 0

Interfaces: blockproc(2)

Access Control Decisions: The Process Attribute Write Access Policy, Signal to block, for Interprocess Communication is enforced.

6.72.2 1

Interfaces: unblockproc(2)

Access Control Decisions: The Process Attribute Write Access Policy, Signal to block, for Interprocess Communication is enforced.

6.72.3 2

Interfaces: setblockproccnt(2)

Access Control Decisions: The Process Attribute Write Access Policy, Signal to block, for Interprocess Communication is enforced.

6.72.4 3

Interfaces: blockprocall(2)

Access Control Decisions: The Process Attribute Write Access Policy, Signal to block, for Interprocess Communication is enforced.

6.72.5 4

Interfaces: unblockprocall(2)

Access Control Decisions: The Process Attribute Write Access Policy, Signal to block, for Interprocess Communication is enforced.

6.72.6 5

Interfaces: setblockproccntall(2)

Access Control Decisions: The Process Attribute Write Access Policy, Signal to block, for Interprocess Communication is enforced.

6.73 profil

Interfaces: profil(2)

Access Control Decisions: No access control decisions are made.

6.74 ptrace

Interfaces: ptrace(2)

Access Control Decisions: The Process Data Read and Process Data Write Access Policies for Interprocess Communication are enforced for read and write operations, respectively. The Process Attribute Write Access Policy, Signal, for Interprocess Communication is enforced.

6.75 putmsg

Interfaces: putmsg(2)

Access Control Decisions: The Data Write Access Policy for File Descriptor is enforced.

6.76 quotactl

Interfaces: quotactl(2)

Quota support is available on the system, but in the Evaluated Configuration it is disabled. The following information is presented here for completeness.

6.76.1 Q_ACTIVATE

Access Control Decisions: Only the Superuser is allowed to use this call.

6.76.2 Q_GETQUOTA

Access Control Decisions: No access control decisions are made.

6.76.3 Q_QUOTAOFF

Access Control Decisions: Only the Superuser is allowed to use this call.

6.76.4 Q_QUOTAON

Access Control Decisions: Only the Superuser is allowed to use this call.

6.76.5 Q_SETQLIM

Access Control Decisions: Only the Superuser is allowed to use this call.

6.76.6 Q_SETQUOTA

Access Control Decisions: Only the Superuser is allowed to use this call.

6.76.7 Q_SYNC

Access Control Decisions: No access control decisions are made.

6.77 read

Interfaces: read(2)

Access Control Decisions: If the descriptor refers to a file system object the Data/Attribute Read Access Policy for File Descriptor is enforced. This may be a destructive operation on some devices, hence it may be necessary to prevent processes at higher labels from *read*ing such devices to block this covert channel. If the descriptor refers to a BSD IPC object the Data Read Access Policy for BSD IPC Socket object types is enforced.

6.78 readlink

Interfaces: readlink(2)

Access Control Decisions: The Data Read Access Policy for Path Name is enforced.

6.79 recv

Interfaces: recv(2)

Access Control Decisions: The Data Read Access Policy for BSD IPC Socket object types is enforced.

6.80 recvfrom

Interfaces: recvfrom(2)

Access Control Decisions: The Data Read Access Policy for BSD IPC Socket object types is enforced.

6.81 recvmsg

Interfaces: recvmsg(2)

Access Control Decisions: The Data Read Access Policy for BSD IPC Socket object types is enforced.

6.82 rename

Interfaces: rename(2)

Access Control Decisions: The Data Write Access Policy for Path Name is enforced on the directory to which the links is to be added, and to the directory from which it is being removed. The Attribute Read Access Policy for Path Name is enforced on the object.

Object Reuse Implications: The old name is removed from the containing directory.

6.83 rmdir

Interfaces: rmdir(2)

Access Control Decisions: The Data Write Access Policy for Path Name is enforced on the directory from which the directory is to be removed. The Attribute Write Access Policy for Path Name is enforced on the object.

Object Reuse Implications: The old name is removed from the containing directory.

6.84 seek

Interfaces: lseek(2)

Access Control Decisions: The Data/Attribute Read Access Policy for File Descriptor is enforced.

6.85 select

Interfaces: select(2)

Access Control Decisions: The Data/Attribute Read Access Policy for File Descriptor and the Attribute Read Access Policy for BSD IPC Socket object types are enforced.

6.86 semsys

6.86.1 SEMCTL

Interfaces: semctl(2)

6.86.1.1 GETALL

Access Control Decisions: The System V IPC, IDs, Read Access Policy for Interprocess Communication is enforced.

6.86.1.2 GETCNT

Access Control Decisions: The System V IPC, IDs, Read Access Policy for Interprocess Communication is enforced.

6.86.1.3 GETPID

Access Control Decisions: The System V IPC, IDs, Read Access Policy for Interprocess Communication is enforced.

6.86.1.4 GETVAL

Access Control Decisions: The System V IPC, IDs, Read Access Policy for Interprocess Communication is enforced.

6.86.1.5 GETZCNT

Access Control Decisions: The System V IPC, IDs, Read Access Policy for Interprocess Communication is enforced.

6.86.1.6 IPC_RMID

Access Control Decisions: The System V IPC, IDs, Attribute Write Access Policy for Interprocess Communication is enforced.

6.86.1.7 IPC_SET

Access Control Decisions: The System V IPC, IDs, Attribute Write Access Policy for Interprocess Communication is enforced.

6.86.1.8 IPC_STATE

Access Control Decisions: The System V IPC, IDs, Read Access Policy for Interprocess Communication is enforced.

6.86.1.9 SETALL

Access Control Decisions: The System V IPC, IDs, Data Write Access Policy for Interprocess Communication is enforced.

6.86.1.10 SETVAL

Access Control Decisions: The System V IPC, IDs, Data Write Access Policy for Interprocess Communication is enforced.

6.86.2 SEMGET

Interfaces: semget(2)

Access Control Decisions: The System V IPC, KEYs, Attribute Read and Attribute Write Access Policies for Interprocess Communication are enforced.

Object Reuse Implications: The new object is created with zero length. All attribute information is initialized.

6.86.3 SEMOP

Interfaces: semop(2)

6.86.3.1 sem_op=0

Access Control Decisions: The System V IPC, IDs, Read Access Policy for Interprocess Communication is enforced.

6.86.3.2 sem_op≠0

Access Control Decisions: The System V IPC, IDs, Data Write Access Policy for Interprocess Communication is enforced.

6.87 send

Interfaces: send(2)

Access Control Decisions: The Data Write Access Policy for BSD IPC Socket object types is enforced.

6.88 sendmsg

Interfaces: sendmsg(2)

Access Control Decisions: The Data Write Access Policy for BSD IPC Socket object types is enforced.

6.89 sendto

Interfaces: sendto(2)

Access Control Decisions: The Data Write Access Policy for BSD IPC Socket object types is enforced.

6.90 setdomainname

Interfaces: setdomainname(2)

Access Control Decisions: Only the Superuser is allowed to use this call.

6.91 setgid

Interfaces: setgid(2)

Access Control Decisions: The Attribute Write Access Policy, Group ID Write, for Intraprocess Communication is enforced.

6.92 sethostid

Interfaces: sethostid(2)

Access Control Decisions: Only the Superuser is allowed to use this call.

6.93 sethostname

Interfaces: sethostname(2)

Access Control Decisions: Only the Superuser is allowed to use this call.

6.94 setitimer

Interfaces: setitimer(2)

Access Control Decisions: No access control decisions are made.

6.95 setpgrp

Interfaces: getpgrp(2) setpgrp(2)

Access Control Decisions: The Attribute Read Access Policy and Attribute Write Access Policy, Session ID Write, for Intraprocess Communication are enforced.

6.96 setregid

Interfaces: setregid(2)

Access Control Decisions: The Attribute Write Access Policy, Group ID Write, for Intraprocess Communication is enforced.

6.97 setreuid

Interfaces: setreuid(2)

Access Control Decisions: The Attribute Write Access Policy, User ID Write, for Intraprocess Communication is enforced.

6.98 setrlimit

Interfaces: setrlimit(2)

Access Control Decisions: No access control decisions are made.

6.99 setsockopt

Interfaces: setsockopt(2)

Access Control Decisions: The Attribute Write Access Policy for BSD IPC Socket object types is enforced.

6.100 setuid

Interfaces: setuid(2)

Access Control Decisions: The Attribute Write Access Policy, User ID Write, for Intraprocess Communication is enforced.

6.101 sgikopt

Interfaces: sgikopt(2)

Access Control Decisions: No access control decisions are made.

6.102 sginap

Interfaces: sginap(2)

Access Control Decisions: No access control decisions are made.

6.103 shmsys

6.103.1 SHMAT

Interfaces: shmat(2)

Access Control Decisions: The System V IPC, IDs, Read and Data Write Access Policies for Interprocess Communication are enforced for read and write, respectively.

6.103.2 SHMCTL

Interfaces: shmctl(2)

6.103.2.1 IPC_RMID

Access Control Decisions: The System V IPC, IDs, Attribute Write Access Policy for Interprocess Communication is enforced.

6.103.2.2 IPC_SET

Access Control Decisions: The System V IPC, IDs, Attribute Write Access Policy for Interprocess Communication is enforced.

6.103.2.3 IPC_STATE

Access Control Decisions: The System V IPC, IDs, Read Access Policy for Interprocess Communication is enforced.

6.103.2.4 SHM_LOCK

Access Control Decisions: Only the Superuser is allowed to use this call.

6.103.2.5 SHM_UNLOCK

Access Control Decisions: Only the Superuser is allowed to use this call.

6.103.3 SHMDT

Interfaces: shmdt(2)

Access Control Decisions: No access control decisions are made.

6.103.4 SHMGET

Interfaces: shmget(2)

Access Control Decisions: The System V IPC, KEYs, Attribute Read and Attribute Write Access Policies for Interprocess Communication are enforced.

Object Reuse Implications: The new object is initialized to zero.

6.104 shutdown

Interfaces: shutdown(2)

Access Control Decisions: The Attribute Write Access Policy for BSD IPC Socket object types is enforced.

6.105 signal

Interfaces: sighold(2) sigignore(2) signal(2) sigpause(2) sigrelse(2) sigset(2)

Access Control Decisions: No access control decisions are made.

6.106 sigreturn

Interfaces:
This is a private interface used to return from the delivery of a signal to the user

process.
The *sigreturn* system call is: sigreturn(struct sigcontext *scp)

Access Control Decisions: No access control decisions are made.

6.107 socket

Interfaces: socket(2)

Access Control Decisions: The Attribute Write Access Policy for BSD IPC Socket object types is enforced.

Object Reuse Implications: The socket data structure is completely initialized.

6.108 socketpair

Interfaces: socketpair(2)

Access Control Decisions: The Attribute Write Access Policy for BSD IPC Socket object types is enforced.

Object Reuse Implications: The socket data structure is completely initialized.

6.109 sproc

Interfaces: sproc(2)

Access Control Decisions: No access control decisions are made.

Object Reuse Implications: A new process object is created which is a copy of the existing object, except for the parts which are shared. No part of the new object is uninitialized.

6.110 stat

Interfaces: stat(2)

Access Control Decisions: The Attribute Read Access Policy for Path Name is enforced.

6.111 statfs

Interfaces: statfs(2)

Access Control Decisions: The Attribute Read Access Policy for Path Name is enforced.

6.112 stime

Interfaces: stime(2)

Access Control Decisions: Only the Superuser is allowed to use this call.

6.113 symlink

Interfaces: symlink(2)

Access Control Decisions: The Data Write Access Policy for Path Name is enforced on the directory to which the symbolic link is to be added.

Object Reuse Implications: The new object is created with zero length. All attribute information is initialized.

6.114 sync

Interfaces: sync(2)

Access Control Decisions: No access control decisions are made.

6.115 sysacct

Interfaces: acct(2)

Access Control Decisions: Only the Superuser is allowed to use this call. The Data Write Access Policy for Path Name is enforced.

Object Reuse Implications: The new object is created with zero length. All attribute information is initialized.

6.116 sysfs

Interfaces: sysfs(2)

Access Control Decisions: No access control decisions are made.

6.117 sysmips

Interfaces: sysmips(2)

6.117.1 FLUSH_CACHE

Access Control Decisions: No access control decisions are made.

6.117.2 MIPS_FIXADE

Access Control Decisions: No access control decisions are made.

6.117.3 MIPS_FPSIGINTR

Access Control Decisions: No access control decisions are made.

6.117.4 SETNAME

Access Control Decisions: Only the Superuser is allowed to use this call.

6.117.5 SMIPSSWPI

6.117.5.1 add

Access Control Decisions: Only the Superuser is allowed to use this call.

6.117.5.2 delete

Access Control Decisions: Only the Superuser is allowed to use this call.

6.117.5.3 list

Access Control Decisions: No access control decisions are made.

6.117.6 STIME

Access Control Decisions: Only the Superuser is allowed to use this call.

6.118 sysmp

Interfaces: sysmp(2)

6.118.1 MP_CLOCK

Access Control Decisions: Only the Superuser is allowed to use this call.

6.118.2 MP_EMPOWER

Access Control Decisions: Only the Superuser is allowed to use this call.

6.118.3 MP_ISOLATE

Access Control Decisions: Only the Superuser is allowed to use this call.

6.118.4 MP_KERNADDR

Access Control Decisions: No access control decisions are made.

6.118.5 MP_MUSTRUN

Access Control Decisions: No access control decisions are made.

6.118.6 MP_NAPROCS

Access Control Decisions: No access control decisions are made.

6.118.7 MP_NONPREEMPTIVE

Access Control Decisions: Only the Superuser is allowed to use this call.

6.118.8 MP_NPROCS

Access Control Decisions: No access control decisions are made.

6.118.9 MP_PGSIZE

Interfaces: getpagesize(2)

Access Control Decisions: No access control decisions are made.

6.118.10 MP_PREEMPTIVE

Access Control Decisions: Only the Superuser is allowed to use this call.

6.118.11 MP_RESTRICT

Access Control Decisions: Only the Superuser is allowed to use this call.

6.118.12 MP_RUNANYWHERE

Access Control Decisions: No access control decisions are made.

6.118.13 MP_SAGET

Access Control Decisions: No access control decisions are made.

6.118.14 MP_SAGET1

Access Control Decisions: No access control decisions are made.

6.118.15 MP_SASZ

Access Control Decisions: No access control decisions are made.

6.118.16 MP_SCHED

Interfaces: getpriority(2) setpriority(2) schedctl(2)

Access Control Decisions: The priority of a process can be set to non-degrading only by the Superuser. The priority of another process can be set only by the Superuser. The "nice" value of a process can be set only by the Superuser.

6.118.17 MP_STAT

Access Control Decisions: No access control decisions are made.

6.118.18 MP_UNISOLATE

Access Control Decisions: Only the Superuser is allowed to use this call.

6.119 syssgi

Interfaces: syssgi(2)

6.119.1 SGI_BDFLUSHCNT

Access Control Decisions: Only the Superuser is allowed to use this call.

6.119.2 SGI_DISABLE_FASTCLOCK

Access Control Decisions: Only the Superuser is allowed to use this call.

6.119.3 SGI_DISABLE_FTIMER

Access Control Decisions: No access control decisions are made.

6.119.4 SGI_ENABLE_FASTCLOCK

Access Control Decisions: Only the Superuser is allowed to use this call.

6.119.5 SGI_ENABLE_FTIMER

Access Control Decisions: No access control decisions are made.

6.119.6 SGI_GETGROUPS

Interfaces: getgroups(2)

Access Control Decisions: The Attribute Read Access Policy for Intraprocess Communication is enforced.

6.119.7 SGI_GETLABEL

Interfaces: getlabel(2)

Access Control Decisions: The Attribute Read Access Policy for Path Name is enforced.

6.119.8 SGI_GETNVRAM

Access Control Decisions: No access control decisions are made.

6.119.9 SGI_GETPLABEL

Interfaces: getplabel(2)

Access Control Decisions: The Attribute Read Access Policy for Intraprocess Communication is enforced.

6.119.10 SGI_GETPSOACL

Interfaces: getpsoacl(2)

Access Control Decisions: The Attribute Read Access Policy for Intraprocess Communication is enforced.

6.119.11 SGI_GETTIMETRIM

Access Control Decisions: No access control decisions are made.

6.119.12 SGI_IDBG

Access Control Decisions: Only the Superuser is allowed to use this call.

6.119.13 SGI_INVENT

Access Control Decisions: No access control decisions are made.

6.119.14 SGI_LOADATTR

Interfaces:
This is a private interface used by the *macplang*(1m) program to initialize the MAC database in a file system, typically as part of system installation.
The *syssgi*(SGI_LOADATTR) system call is: syssgi(SGI_LOADATTR, char * attribute_file)

Access Control Decisions: Only the Superuser is allowed to use this call.

6.119.15 SGI_NETPROC

Interfaces:
This is an internal interface used to start the kernel internal TCP/IP network processing in the pre-emptable context of a user mode task instead of the interrupt task normally used for this purpose. This improves performance on multiprocessor systems. This call is made only from the program /usr/etc/rnetd, and never returns. rtnetd is started by the /etc/init.d/network script on multi-processor systems. Internally, the system allows only one instance of this network process, a second invocation of the call will fail. Only the context of the process is used, similarly to that of the biod and nfsd processes.

Access Control Decisions: No access control decisions are made.

6.119.16 SGI_NFSCNVT

Interfaces:
This is an internal interface used by rpc.lockd, which is not supported in the Evaluated Configuration.
The *syssgi*(SGI_NFSCNVT) system call is: syssgi(SGI_NFSCNVT, int nfs_handle, int openmode)
where nfs_handle is an internal NFS file handle. NFS file handles are treated as bit patterns with no semantics by the NFS client. The call transforms the handle into an internal vnode pointer and then calls the same routine as open() does after it processes the user supplied name into a vnode pointer. Thus, this call will be subject to the same access control and auditing as a regular open call.

Access Control Decisions: Only the Superuser is allowed to use this call. The Data Read and Data Write Access Policies for Path Name are enforced for read and write operations, respectively.

6.119.17 SGI_PATHCONF

Interfaces: fpathconf(2) pathconf(2)

Access Control Decisions: No access control decisions are made.

6.119.18 SGI_PLANGMOUNT

Interfaces: mac_mount in mount(2).

Access Control Decisions: Only the Superuser is allowed to use this call. The Data Write Access Policy for Path Name is enforced on the mounted-on file name.

6.119.19 SGI_PROCSZ

Access Control Decisions: The Process Attribute Read Access Policy for Interprocess Communication is enforced.

6.119.20 SGI_QUERY_CYCLECNTR

Access Control Decisions: No access control decisions are made.

6.119.21 SGI_QUERY_FTIMER

Access Control Decisions: No access control decisions are made.

6.119.22 SGI_R4K_CERRS

Interfaces:
This is a diagnostic interface which returns a structure containing a set of ECC statistics for the cache memory. It is not used in normal operations.

Access Control Decisions: No access control decisions are made.

6.119.23 SGI_RDNAME

Access Control Decisions: The Process Attribute Read Access Policy for Interprocess Communication is enforced.

6.119.24 SGI_RDUBLK

Access Control Decisions: The Process Attribute Read Access Policy for Interprocess Communication is enforced.

6.119.25 SGI_READB

Interfaces:
This is a private interface used by the file system maintenance routines to access the disk devices at the block (vs. byte) level.
The *syssgi*(SGI_READB) system call is: syssgi(SGI_READB, int fd, char * buf, int blk, int nblks)
where fd is the file descriptor for a previously opened character device file, buf is the address of the buffer into which the data will be read, blk is the block (512 bytes) on which to start the read, and nblks is the number of 512 byte blocks to read.

Access Control Decisions: The Data/Attribute Read Access Policy for File Descriptor is enforced.

6.119.26 SGI_RECVLMSG

Interfaces: recvlmsg(2)

Access Control Decisions: The Data Read Access Policy for BSD IPC Socket object types is enforced.

6.119.27 SGI_RECVLUMSG

Interfaces: recvlumsg(2)

Access Control Decisions: The Data Read Access Policy for BSD IPC Socket object types is enforced.

6.119.28 SGI_REVOKE

Accesss Control Decisions: Only the Superuser is allowed to use this call.

6.119.29 SGI_RMI_FIXECC

Interfaces:
This is a private interface to enable/disable ECC in the memory subsystem of CHALLENGE/Onyx family of Symmetric Multiprocessor (SMP) systems. For the evaluated configuration, this call is a noop. It is present for hardware diagnostic and debugging purposes and is not used in normal operations.

Access Control Decisions: Only the Superuser is allowed to use this call.

6.119.30 SGI_RUSAGE

Interfaces: syssgi(2), getrusage(3)

Access Control Decisions: No access control decisions are made.

6.119.31 SGI_SATCTL

All SGI_SATCTL calls are allowed to be used only by the Superuser. The function is determined by the second argument to the *syssgi*(SGI_SATCTL, XXX,...) call.

6.119.31.1 SATCTL_AUDIT_OFF

Interfaces: satoff(2)

Access Control Decisions: Only the Superuser is allowed to use this call.

6.119.31.2 SATCTL_AUDIT_ON

Interfaces: saton(2)

Access Control Decisions: Only the Superuser is allowed to use this call.

6.119.31.3 SATCTL_AUDIT_QUERY

Interfaces: satstate(2)

Access Control Decisions: Only the Superuser is allowed to use this call.

6.119.31.4 SATCTL_GET_SAT_ID

Interfaces: satgetid(2)

Access Control Decisions: Only the Superuser is allowed to use this call.

6.119.31.5 SATCTL_LOCALAUDIT_OFF

Access Control Decisions: Only the Superuser is allowed to use this call.

6.119.31.6 SATCTL_LOCALAUDIT_ON

Access Control Decisions: Only the Superuser is allowed to use this call.

6.119.31.7 SATCTL_LOCALAUDIT_QUERY

Access Control Decisions: Only the Superuser is allowed to use this call.

6.119.31.8 SATCTL_SET_SAT_ID

Interfaces: satsetid(2)

Access Control Decisions: Only the Superuser is allowed to use this call.

6.119.31.9 SGI_REGISTER_SATD

Interfaces:
This is a private interface used by the external system audit trail daemon, *satd* to inform the kernel that it is the audit capture daemon. Only one such daemon may so register, the second attempt will return an EBUSY error.

Access Control Decisions: Only the Superuser is allowed to use this call.

6.119.32 SGI_SATREAD

Interfaces: satread(2)

Access Control Decisions: Only the Superuser is allowed to use this call.

6.119.33 SGI_SATWRITE

Interfaces: satwrite(2)

Access Control Decisions: Only the Superuser is allowed to use this call.

6.119.34 SGI_SET_AUTOPWRON

Interfaces:
This is a private interface used by the *wakeupat*(1m) command. Its purpose is to arrange for the system to be powered up in a specified number of seconds, specified as the system is being shutdown. (*wakeupat* takes an argument like *at*(1), this is translated into the number of seconds to wait until automatically powering up the system.)

Access Control Decisions: Only the Superuser is allowed to use this call.

6.119.35 SGI_SETGROUPS

Interfaces: setgroups(2)

Access Control Decisions: Only the Superuser is allowed to use this call. The Attribute Write Access Policy, Group Access List Write, for Intraprocess Communication is enforced.

6.119.36 SGI_SETLABEL

Interfaces: setlabel(2)

Access Control Decisions: Only the Superuser is allowed to use this call. The Attribute Write Access Policy for Path Name is enforced.

6.119.37 SGI_SETLED

Interfaces:
The *syssgi*(SGI_SETLED) system call is: syssgi(SGI_SETLED, int pattern)
where pattern is a 0 indicating the CPU LED is to be set to green, or non-zero indicating it is to be set to amber.

Access Control Decisions: No access control decisions are made.

6.119.38 SGI_SETNVRAM

Access Control Decisions: Only the Superuser is allowed to use this call.

6.119.39 SGI_SETPGID

Interfaces: setpgid(2)

Access Control Decisions: The Attribute Write Access Policy, Group ID Write, for Intraprocess Communication and the Process Attribute Write Access Policy, Process Group ID Write, for Interprocess Communication are enforced.

6.119.40 SGI_SETPLABEL

Interfaces: setplabel(2)

Access Control Decisions: The Attribute Write Access Policy, Label Write, for Intraprocess Communication is enforced.

6.119.41 SGI_SETPSOACL

Interfaces: setpsoacl(2)

Access Control Decisions: No access control decisions are made.

6.119.42 SGI_SETSID

Interfaces: setsid(2)

Access Control Decisions: The Attribute Write Access Policy, Session ID Write, for Intraprocess Communication is enforced.

6.119.43 SGI_SETTIMEOFDAY

Interfaces: settimeofday (3b)

Access Control Decisions: Only the Superuser is allowed to use this call.

6.119.44 SGI_SETTIMETRIM

Access Control Decisions: Only the Superuser is allowed to use this call.

6.119.45 SGI_SIGACTION

Interfaces: sigaction(2)

Access Control Decisions: No access control decisions are made.

6.119.46 SGI_SIGALTSTACK

Interfaces: sigaltstack(2)

Access Control Decisions: No access control decisions are made.

6.119.47 SGI_SIGSTACK

Interfaces:

This is the SVR4 version of sigstack. The call interface is identical to the

sigstack(2), but the entry point is *sigaltstack*. BSD 4.3 applications would use *sigstack* while SVR4 applications would use *sigaltstack*. Both have the same effect, that of specifying an alternate address to be used as the stack when processing a signal received by the process.

Access Control Decisions: No access control decisions are made.

6.119.48 SGI_SIGSTATUS

Interfaces:
This is an internal interface used by the BSD 4.3 and POSIX signal handling code, and by the BSD 4.3 setjmp/longjmp code to query and set the signal status.
The *syssgi*(SGI_SIGSTATUS) system call is: syssgi(SGI_SIGSTATUS, sigset_t *newmask, sigset_t *newonstack, sigset_t *oldmask, sigset_t *oldonstack)
If newmask is non-NULL, this value will become the new signal mask. If newonstack is non-NULL, it will be used to set the alternate signal stack flag. If this newonstack is 0, the flag will be cleared; if non-zero it will be set. This flag controls whether the alternate signal stack will be used. If oldmask is non-NULL, the current signal mask will be returned. If oldonstack is non-NULL, the current value of the alternate signal stack flag will be returned.

Access Control Decisions: No access control decisions are made.

6.119.49 SGI_SIGPENDING

Interfaces: sigpending(2)

Access Control Decisions: No access control decisions are made.

6.119.50 SGI_SIGPROCMASK

Interfaces: sigprocmask(2)

Access Control Decisions: No access control decisions are made.

6.119.51 SGI_SIGSUSPEND

Interfaces: sigsuspend(2)

Access Control Decisions: No access control decisions are made.

6.119.52 SGI_SPROFIL

Interfaces: sprofil(2)

Access Control Decisions: No access control decisions are made.

6.119.53 SGI_SSYNC

Access Control Decisions: No access control decisions are made.

6.119.54 SGI_SYSCONF

Interfaces: sysconf(2)

Access Control Decisions: No access control decisions are made.

6.119.55 SGI_SYSID

Access Control Decisions: No access control decisions are made.

6.119.56 SGI_TUNE

Access Control Decisions: Only the Superuser is allowed to use this call.

6.119.57 SGI_UNLOADATTR

Interfaces:
This is a private interface to disassociate the attribute database from the file system, effectively turning off MAC information for the file system. This is not normally used, but is present for debugging and initialization problems.

Access Control Decisions: Only the Superuser is allowed to use this call.

6.119.58 SGI_WAIT4

Interfaces: wait4(2) waitpid(2)

Access Control Decisions: No access control decisions are made.

6.119.59 SGI_WRITEB

Interfaces:
This is a private interface used by the file system maintenance routines to access the disk devices at the block (vs. byte) level.
The *syssgi*(SGI_WRITEB) system call is: syssgi(SGI_WRITEB, int fd, char * buf, int blk, int nblks)
where fd is the file descriptor for a previously opened character device file, buf is the address of the buffer into which the data will be written, blk is the block (512

bytes) on which to start the write, and nblks is the number of 512 byte blocks to be written.

Access Control Decisions: The Data Write Access Policy for File Descriptor is enforced.

6.120 texturebind

Interfaces: texturebind(2)

Access Control Decisions: No access control decisions are made. This is an obsolete interface which does nothing under Trusted IRIX/B.

6.121 time

Interfaces: time(2)

Access Control Decisions: No access control decisions are made.

6.122 times

Interfaces: times(2)

Access Control Decisions: No access control decisions are made.

6.123 truncate

Interfaces: truncate(2)

Access Control Decisions: The Data Write Access Policy for Path Name is enforced.

6.124 uadmin

Interfaces: uadmin(2)

Access Control Decisions: Only the Superuser is allowed to use this call.

6.125 ulimit

Interfaces: ulimit(2)

6.125.1 1

Interfaces:

Access Control Decisions: No access control decisions are made.

6.125.2 2

Interfaces:

Access Control Decisions: No access control decisions are made.

6.125.3 3

Interfaces:

Access Control Decisions: No access control decisions are made.

6.125.4 4

Interfaces: getdtablesize(2)

Access Control Decisions: No access control decisions are made.

6.126 umask

Interfaces: umask(2)

Access Control Decisions: No access control decisions are made.

6.127 umount

Interfaces: umount(2)

Access Control Decisions: Only the Superuser is allowed to use this call. The Data Write Access Policy for Path Name is enforced on the directory on which the file system is to be mounted.

6.128 unlink

Interfaces: remove (3c) unlink(2)

Access Control Decisions: The Data Write Access Policy for Path Name is enforced on the directory from which the link is to be removed. The Attribute Write Access Policy for Path Name is enforced on the object.

6.129 utime

Interfaces: utime(2)

Access Control Decisions: The Attribute Write Access Policy for Path Name is enforced.

6.130 utssys

6.130.1 uname

Interfaces: uname(2)

Access Control Decisions: No access control decisions are made.

6.130.2 ustat

Interfaces: ustat(2)

Access Control Decisions: Only the Superuser is allowed to use this call.

6.131 vhangup

Interfaces: vhangup(2)

Access Control Decisions: Only the Superuser is allowed to use this call.

6.132 wait

Interfaces: wait(2)

Access Control Decisions: No access control decisions are made.

6.133 wait3

Interfaces: wait3(2)

Access Control Decisions: No access control decisions are made.

6.134 write

Interfaces: write(2)

Access Control Decisions: If the descriptor refers to a file system object the Data Write Access Policy for File Descriptor is enforced. If the descriptor refers to a BSD IPC Socket object the Data Write Access Policy for BSD IPC Socket

object types is enforced.

A. Appendix A: Objects and Their Manipulation

Tables A-1 through A-9 summarize Trusted IRIX/B System Object manipulation. Each table identifies the system object type(s) which it is addressing. Within each table, the system calls which may manipulate the identified object type(s) are associated with the specific form of manipulation; i.e., create, delete, attribute read, attribute write, data read, or data write.

TABLE A-1. File System Object Manipulation

Object Type	Create	Delete	Attribute		Data	
			Read	Write	Read	Write
Block Device	mknod	unlink	access fstat fstatfs lstat stat statfs syssgi __SGI_GETLABEL	chmod chown fchmod fchown syssgi __SGI_SETLABEL utime	read	write
Character Device	mknod	unlink	access fstat fstatfs lstat stat statfs syssgi __SGI_GETLABEL	chmod chown fchmod fchown syssgi __SGI_SETLABEL utime	read	write
Directory	mkdir	rmdir	access fstat fstatfs lstat stat statfs syssgi __SGI_GETLABEL utssys __ustat*	chmod chown fchmod fchown syssgi __SGI_SETLABEL utime	chdir getdents read	bind** creat link mkdir mkfifo mknod open rename rmdir symlink sysacct unlink write
FIFO	mknod	unlink	access fstat fstatfs lstat stat statfs syssgi __SGI_GETLABEL	chmod chown fchmod fchown syssgi __SGI_SETLABEL utime	read	write
File	creat open	unlink	access fstat	chmod chown	read	ftruncate truncate

Object Type	Create	Delete	Attribute Read	Attribute Write	Data Read	Data Write
	sysacct		fstatfs lstat stat statfs syssgi __SGI_GETLABEL	fchmod fchown syssgi __SGI_SETLABEL utime		write
Pipe	pipe	close	fstat fstatfs	fchmod fchown	read	write
Symbolic Link	link rename symlink	unlink	lstat	√	readlink	√
UNIX Domain Rendezvous Point	bind	unlink	access fstat fstatfs lstat stat statfs syssgi __SGI_GETLABEL	chmod chown fchmod fchown syssgi __SGI_SETLABEL utime	√	√
Process "File"	fork sproc	√	fcntl ioctl	fcntl ioctl	read	write

* For attributes associated with a *file system object* which is the root directory of a file system.

** For UNIX domain sockets only.

√ Action cannot be performed via a system call.

TABLE A-2. Current Process Object Manipulation

		Attribute		Data	
Create	**Delete**	**Read**	**Write**	**Read**	**Write**
√	exit	BSDgetpgrp getgid getpid getuid setpgrp syssgi __SGI_GETGROUPS __SGI_GETPLABEL __SGI_GETPSOACL	BSDsetpgrp setgid setpgrp setregid setreuid setuid syssgi __SGI_SETGROUPS __SGI_SETPGID __SGI_SETPLABEL __SGI_SETPSOACL __SGI_SETSID	D/A	D/A

√ Action cannot be performed via a system call.

D/A Data associated with a *process object* can be red and written directly by that process.

TABLE A-3. Another Process Object Manipulation

		Attribute		Data	
Create	**Delete**	**Read**	**Write**	**Read**	**Write**
fork sproc	√	BSDgetpgrp syssgi __SGI_PROCSZ __SGI_RDNAME __SGI_RDUBLK	BSDsetpgrp kill procblk ptrace __PTRC_CONTINUE __PTRC_STEP __PTRC_TERMINATE syssgi __SGI_SETPGID	ptrace __PTRC_RD_I __PTRC_RD_D __PTRC_RD_REG __PTRC_RD_GPRS __PTRC_RD_FPRS	ptrace __PTRC_WR_I __PTRC_WR_D __PTRC_WR_REG __PTRC_WR_GPRS __PTRC_WR_FPRS

√ Action cannot be performed via a system call.

TABLE A-4. System V IPC Message Queue Object Manipulation

		Attribute		Data	
Create	**Delete**	**Read**	**Write**	**Read**	**Write**
msgsys __MSGGET	msgsys __MSGCTL ____IPC_RMID	msgsys __MSGCTL ____IPC_STATE	msgsys __MSGCTL ____IPC_SET	msgsys __MSGRCV	msgsys __MSGSND

TABLE A-5. System V IPC Semaphore Set Object Manipulation

		Attribute		Data	
Create	**Delete**	**Read**	**Write**	**Read**	**Write**
semsys __SEMGET	semsys __SEMCTL ____IPC_RMID	semsys __SEMCTL ____IPC_STATE	semsys __SEMCTL ____IPC_SET	semsys __SEMCTL ____GETALL ____GETCNT ____GETPID ____GETVAL ____GETZCNT semsys __SEMOP ____sem_op=0	semsys __SEMCTL ____SETALL ____SETVAL semsys __SEMOP ____sem_op≠0

TABLE A-6. System V IPC Shared Memory Segment Object Manipulation

		Attribute		Data	
Create	**Delete**	**Read**	**Write**	**Read**	**Write**
shmsys __SHMGET	shmsys __SHMCTL ____IPC_RMID	shmsys __SHMCTL ____IPC_STATE	shmsys __SHMCTL ____IPC_SET ____SHM_LOCK ____SHM_UNLOCK	D/A shmsys __SHMAT	D/A shmsys __SHMAT

D/A Data associated with a *System V IPC object* of type **shared memory segment** may be manipulated directly by the process.

TABLE A-7. BSD IPC Socket Object Manipulation

		Attribute		Data	
Create	**Delete**	**Read**	**Write**	**Read**	**Write**
accept	close	fcntl	accept	read	send
socket	exit	getpeername	bind	recv	sendmsg
socketpair		getsockname	connect	recvfrom	sendto
		getsockopt	fcntl	recvmsg	write
		ioctl	ioctl	syssgi	
		select	listen	__SGI_RECVLMSG	
			setsockopt	syssgi	
			shutdown	__SGI_RECVLUMSG	
			socket		
			socketpair		

TABLE A-8. Public Object Manipulation

Object Type	Attribute				Data	
	Create	Delete	Read	Write	Read	Write
System Clock	√	√	√	√	alarm bsdgettime getitimer setitimer	adjtime stime sysmips __STIME sysmp __MP_CLOCK syssgi __SGI_DISABLE_FASTCLOCK __SGI_DISABLE_FTIMER __SGI_ENABLE_FASTCLOCK __SGI_ENABLE_FTIMER __SGI_GETTIMETRIM __SGI_QUERY_FTIMER __SGI_SETTIMEOFDAY __SGI_SETTIMETRIM time times
Cycle Counter	√	√	√	√	syssgi __SGI_QUERY_ CYCLECNTR	√
Domain Name	√	√	√	√	getdomainname	setdomainname
Host Name	√	√	√	√	gethostname	sethostname
Host ID	√	√	√	√	gethostid	sethostid
NVRAM	√	√	√	√	sgikopt syssgi __SGI_GETNVRAM	syssgi __SGI_SETNVRAM
Scheduler	√	√	√	√	schedctl	schedctl

√ Action cannot be performed via a system call.

TABLE A-9. Constant Object Manipulation

Object Type	Attribute				Data	
	Create	Delete	Read	Write	Read	Write
Page Size	√	√	√	√	getpagesize	√
Descriptor Table Size	√	√	√	√	getdtablesize	√

√ Action cannot be performed via a system call.

B. Appendix B: Use By Policy

Tables B-1 through B-8 summarize Trusted IRIX/B System Security Policy enforcement at the system call level. Tables B-1 through B-6 correspond to the Path Name, File Descriptor, Intraprocess Communication, Interprocess Communication - Process IPC, Interprocess Communication - System V IPC, and Interprocess Communication - BSD (Socket) IPC policies, respectively. Tables B-7 and B-8 address Superuser policy and normal user restrictions. Each table identifies the Trusted IRIX/B System Security Policy which it is addressing. Within each table, the system calls which enforce the identified security policy are enumerated.

TABLE B-1. Path Name Policy Enforcement

System Call	Policy				
	Data Read	**Data Write**	**Attribute Read**	**Attribute Write**	**Execute**
access			•		
bind**		•			
chdir	•				
chmod				•	
chown				•	
chroot*	•				
creat		•			
exec					•
exece					•
link		•			
lstat			•		
mkdir		•			
mknod**		•			
mount*		•			
open	•	•			
readlink	•				
rename		•	•		
rmdir		•		•	
stat			•		
statfs			•		
symlink		•			
sysacct*		•			
syssgi, SGI_GETLABEL			•		
syssgi, SGI_NFSCNVT*	•	•			
syssgi, SGI_PLANGMOUNT*		•			
syssgi, SGI_SETLABEL*				•	
truncate		•			
umount*		•			
unlink		•		•	
utime				•	

* - Only the Superuser is allowed to use these calls. Please refer to Table B-7 "Superuser Only Policy Enforcement".

** - These system calls involve normal user restricted behavior. Please refer to Table B-8 "Superuser Impact".

TABLE B-2. File Descriptor Policy Enforcement

System Call	Policy		
	Read	Data Write	Attribute Write
fchdir	•		
fchmod			•
fchown			•
fcntl	•		
fstat	•		
fstatfs	•		
fsync	•		
ftruncate		•	
getdents	•		
getmsg	•		
mmap	•		
poll	•		
putmsg		•	
read	•		
seek	•		
select	•		
syssgi, SGI_READB	•		
syssgi, SGI_WRITEB		•	
write		•	

TABLE B-3. Intraprocess Communication Policy Enforcement

System Call	Policy Attribute	
	Read	**Write**
BSDgetpgrp	•	
BSDsetpgrp		Session ID
getgid	•	
getpid	•	
getuid	•	
setgid		Group ID
setpgrp	•	Session ID
setregid		Group ID
setreuid		User ID
setuid		User ID
syssgi, SGI_GETGROUPS	•	
syssgi, SGI_GETPLABEL	•	
syssgi, SGI_GETPSOACL	•	
syssgi, SGI_SETGROUPS*		Group Access List
syssgi, SGI_SETPGID		Group ID
syssgi, SGI_SETPLABEL		Label
syssgi, SGI_SETSID		Session ID

* - Only the Superuser is allowed to use this call.
Please refer to Table B-7 "Superuser Only Policy Enforcement".

TABLE B-4. Process IPC Policy Enforcement

System Call	Policy			
	Data		Attribute	
	Read	Write	Read	Write
BSDgetpgrp			•	
BSDsetpgrp				Process Group ID
kill				Signal to Block
procblk				Signal to Block
ptrace	•	•		Signal
syssgi, SGI_PROCSZ			•	
syssgi, SGI_RDNAME			•	
syssgi, SGI_RDUBLK			•	
syssgi, SGI_SETPGID				Process Group ID

TABLE B-5. System V IPC Policy Enforcement

System Call	Policy				
	KEYs		IDs		
	Attribute		Data/Attribute	Data	Attribute
	Read	Write	Read	Write	Write
msgsys, MSGCTL, IPC_RMID					•
msgsys, MSGCTL, IPC_SET**					•
msgsys, MSGCTL, IPC_STATE			•		
msgsys, MSGGET	•	•			
msgsys, MSGRCV			•		
msgsys, MSGSND				•	
semsys, SEMCTL, GETALL			•		
semsys, SEMCTL, GETCNT			•		
semsys, SEMCTL, GETPID			•		
semsys, SEMCTL, GETVAL			•		
semsys, SEMCTL, GETZCNT			•		
semsys, SEMCTL, IPC_RMID					•
semsys, SEMCTL, IPC_SET					•
semsys, SEMCTL, IPC_STATE			•		
semsys, SEMCTL, SETALL				•	
semsys, SEMCTL, SETVAL				•	
semsys, SEMGET	•	•			
semsys, SEMOP, sem_op=0			•		
semsys, SEMOP, sem_op≠0				•	
shmsys, SHMAT			•	•	
shmsys, SHMCTL, IPC_RMID					•
shmsys, SHMCTL, IPC_SET					•
shmsys, SHMCTL, IPC_STATE			•		
shmsys, SHMDT					
shmsys, SHMGET	•	•			

** - This system call involves normal user restricted behavior. Please refer to Table B-8 "Superuser Impact".

TABLE B-6. BSD IPC Policy Enforcement

System Call	Policy		
	Data/Attribute Read	**Data Write**	**Attribute Write**
accept			•
bind**			•
connect			•
fcntl	•		•
getpeername	•		
getsockname	•		
getsockopt	•		
listen			•
read	•		
recv	•		
recvfrom	•		
recvmsg	•		
select	•		
send		•	
sendmsg		•	
sendto		•	
setsockopt**			•
shutdown			•
socket**			•
socketpair			•
syssgi, SGI_RECVLMSG	•		
syssgi, SGI_RECVLUMSG	•		
write		•	

** - These system calls involve normal user restricted behavior. Please refer to Table B-8 "Superuser Impact".

TABLE B-7. Superuser Only Policy Enforcement

System Call	Other Policies
adjtime	
async_daemon	
chroot	Data Read Access Policy for Path Name
exportfs	
ioctl, I_PEEK	
ioctl, I_STR	
ioctl, SIOCADDMULTI	
ioctl, SIOCADDRT	
ioctl, SIOCDARP	
ioctl, SIOCDELMULTI	
ioctl, SIOCDELRT	
ioctl, SIOCSARP	
ioctl, SIOCSETLABEL	
ioctl, SIOCSETRTINFO	
ioctl, SIOCSETUID	
ioctl, SIOCSIFADDR	
ioctl, SIOCSIFBRDADDR	
ioctl, SIOCSIFDSTADDR	
ioctl, SIOCSIFFLAGS	
ioctl, SIOCSIFHEAD	
ioctl, SIOCSIFLABEL	
ioctl, SIOCSIFMETRIC	
ioctl, SIOCSIFNETMASK	
ioctl, SIOCSIFSTATS	
ioctl, SIOCSIFUID	
ioctl, TCSETLABEL	
ioctl, TIOCSPGRP	
ioctl, TIOCSTI	
mount	Data Write Access Policy for Path Name
nfs_getfh	
nfssvc	
plock	
quotactl, Q_ACTIVATE	
quotactl, Q_QUOTAOFF	
quotactl, Q_QUOTAON	
quotactl, Q_SETQLIM	
quotactl, Q_SETQUOTA	
setdomainname	
sethostid	
sethostname	
shmsys, SHMCTL, SHM_LOCK	

System Call	Other Policies
shmsys, SHMCTL, SHM_UNLOCK	
stime	
sysacct	Data Write Access Policy for Path Name
sysmips, SETNAME	
sysmips, SMIPSSWPI, add	
sysmips, SMIPSSWPI, delete	
sysmips, STIME	
sysmp, MP_CLOCK	
sysmp, MP_EMPOWER	
sysmp, MP_ISOLATE	
sysmp, MP_NONPREEMPTIVE	
sysmp, MP_PREEMPTIVE	
sysmp, MP_RESTRICT	
sysmp, MP_UNISOLATE	
syssgi, SGI_BDFLUSHCNT	
syssgi, SGI_DISABLE_FASTCLOCK	
syssgi, SGI_ENABLE_FASTCLOCK	
syssgi, SGI_IDBG	
syssgi, SGI_LOADATTR	
syssgi, SGI_NFSCNVT	Data Read Access Policy for Path Name Data Write Access Policy for Path Name
syssgi, SGI_PLANGMOUNT	Data Write Access Policy for Path Name
syssgi, SGI_REVOKE	
syssgi, SGI_RMI_FIXECC	
syssgi, SGI_SATCTL, SATCTL_AUDIT_OFF	
syssgi, SGI_SATCTL, SATCTL_AUDIT_ON	
syssgi, SGI_SATCTL, SATCTL_AUDIT_QUERY	
syssgi, SGI_SATCTL, SATCTL_GET_SAT_ID	
syssgi, SGI_SATCTL, SATCTL_LOCALAUDIT_OFF	
syssgi, SGI_SATCTL, SATCTL_LOCALAUDIT_ON	
syssgi, SGI_SATCTL, SATCTL_LOCALAUDIT_QUERY	
syssgi, SGI_SATCTL, SATCTL_SET_SAT_ID	
syssgi, SGI_SATCTL, SGI_REGISTER_SATD	
syssgi, SGI_SATREAD	
syssgi, SGI_SATWRITE	
syssgi, SGI_SET_AUTOPWRON	
syssgi, SGI_SETGROUPS	Attribute Write Access Policy, Group Access List Write, for Intraprocess Communication
syssgi, SGI_SETLABEL	Attribute Write Access Policy for Path Name
syssgi, SGI_SETNVRAM	
syssgi, SGI_SETTIMEOFDAY	
syssgi, SGI_SETTIMETRIM	

System Call	Other Policies
syssgi, SGI_TUNE syssgi, SGI_UNLOADATTR uadmin umount utssys, ustat vhangup	Data Write Access Policy for Path Name

TABLE B-8. Superuser Impact

System Call	Normal User Restrictions
bind	May only bind to port# ≥ 1026
mknod	May only create FIFO objects
msgsys, MSGCTL, IPC_SET	May only set other than *msg_qbytes*
setsockopt	May only set *optname* to other than SO_CHAMELEON
socket	May only set *type* to other than SOCK_RAW
sysmp, MP_SCHED	May only downgrade priority

C. Appendix C: Supported *ioctl* Summary

The *ioctl()* system call operates on an open file descriptor returned from a successful *open()* of a device special file, or on a socket descriptor returned from a successful *socket()* call.

Based on the information stored in the inode referenced by the file or socket descriptor, the ioctl command is dispatched through the file system type table to a file system type specific ioctl routine. Note: a file system type does not imply a file system on disk; sometimes it is just a convenient classification as in the case of sockets.

TABLE C-1. File System Type to Routine Correspondence

File System Type	Routine
Common	com_ioctl()
Debug	dbg_ioctl()
EFS	efs_ioctl()
NFS	nfs_ioctl()
BSD Networking	soc_ioctl()

Each of these file system specific routines either handle the ioctl command, or dispatch it to more specialized routines based on the "device" characteristics specified in the inode.

C.1 Common File System Ioctl Routines

The routine com_ioctl() handles the common file system ioctl commands. Based on the type of the device, the ioctl command is either dispatched to the streams ioctl routine, strioctl(), or to a driver specific ioctl routine found by indexing into the cdevsw table with the major number of the device.

C.1.1 Streams Ioctl Routines

The routine strioctl() handles streams ioctl commands which include most TTY related ioctl commands. Depending on the ioctl command, the call is either handled in this routine or passed on to a streams driver module.

FIONBIO	No Access Checks
FIONREAD	No Access Checks
I_FDINSERT	No Access Checks
I_FIND	No Access Checks
I_FLUSH	No Access Checks
I_GETSIG	No Access Checks

I_GRDOPT	No Access Checks
I_LINK	No Access Checks
I_LOOK	No Access Checks
I_NREAD	No Access Checks
I_PEEK	**Superuser Only**
I_POP	No Access Checks
I_PUSH	No Access Checks
I_RECVFD	No Access Checks
I_SENDFD	No Access Checks
I_SETSIG	No Access Checks
I_SRDOPT	No Access Checks
I_STR	**Superuser Only**
I_UNLINK	No Access Checks
TCBLKMD	No Access Checks
TCFLSH	No Access Checks
TCGETA	No Access Checks
TCSBRK	No Access Checks
TCSETA	No Access Checks
TCSETAF	No Access Checks
TCSETAW	No Access Checks
TCSETLABEL	**Superuser Only**
TCXONC	No Access Checks
TIOCCONS	No Access Checks
TIOCFLUSH	No Access Checks
TIOCGPGRP	No Access Checks
TIOCGWINSZ	No Access Checks
TIOCMGET	No Access Checks
TIOCNOTTY	No Access Checks
TIOCPKT	No Access Checks
TIOCSPGRP	**Superuser Only**
TIOCSTI	**Superuser Only**
TIOCSWINSZ	No Access Checks
oFIONREAD	No Access Checks
oTCGETA	No Access Checks
oTCSETA	No Access Checks
oTCSETAF	No Access Checks
oTCSETAW	No Access Checks

C.1.1.1 Calcomp Streams Driver

The calcomp graphics device is not supported in the evaluated configuration. However, since the streams driver is included in the kernel, the ioctl commands are listed here for completeness.

IDEVCHANGEVALUATORS	No Access Checks
IDEVENABLEBUTTONS	No Access Checks
IDEVENABLEVALUATORS	No Access Checks
IDEVGETBUTTONS	No Access Checks
IDEVGETDEVICEDESC	No Access Checks
IDEVGETTRANSFORM	No Access Checks
IDEVGETVALUATORDESC	No Access Checks
IDEVGETVALUATORS	No Access Checks
IDEVINITDEVICE	No Access Checks
IDEVOTHERCONTROL	No Access Checks
IDEVPTRCONTROL	No Access Checks
IDEVSETPTR	No Access Checks
IDEVSETPTRBOUNDS	No Access Checks
IDEVSETPTRMODE	No Access Checks
IDEVSETTRANSFORM	No Access Checks
IDEVSETVALUATORDESC	No Access Checks
IDEVSETVALUATORS	No Access Checks

C.1.1.2 Wacom Streams Driver

The wacom graphics tablet is not supported in the evaluated configuration. However, since the streams driver is included in the kernel, the ioctl commands are listed here for completeness.

IDEVCHANGEVALUATORS	No Access Checks
IDEVENABLEBUTTONS	No Access Checks
IDEVENABLEVALUATORS	No Access Checks
IDEVGETBUTTONS	No Access Checks
IDEVGETDEVICEDESC	No Access Checks
IDEVGETTRANSFORM	No Access Checks
IDEVGETVALUATORDESC	No Access Checks
IDEVGETVALUATORS	No Access Checks
IDEVINITDEVICE	No Access Checks
IDEVOTHERCONTROL	No Access Checks
IDEVPTRCONTROL	No Access Checks
IDEVSETPTR	No Access Checks
IDEVSETPTRBOUNDS	No Access Checks
IDEVSETPTRMODE	No Access Checks
IDEVSETTRANSFORM	No Access Checks
IDEVSETVALUATORDESC	No Access Checks
IDEVSETVALUATORS	No Access Checks

C.1.1.3 Tablet Streams Driver

The generic graphics tablet is not supported in the evaluated configuration. However, since the streams driver is included in the kernel, the ioctl commands are listed here for completeness.

IDEVCHANGEVALUATORS	No Access Checks
IDEVENABLEBUTTONS	No Access Checks
IDEVENABLEVALUATORS	No Access Checks
IDEVGETBUTTONS	No Access Checks
IDEVGETDEVICEDESC	No Access Checks
IDEVGETTRANSFORM	No Access Checks
IDEVGETVALUATORDESC	No Access Checks
IDEVGETVALUATORS	No Access Checks
IDEVINITDEVICE	No Access Checks
IDEVOTHERCONTROL	No Access Checks
IDEVPTRCONTROL	No Access Checks
IDEVSETPTR	No Access Checks
IDEVSETPTRBOUNDS	No Access Checks
IDEVSETPTRMODE	No Access Checks
IDEVSETTRANSFORM	No Access Checks
IDEVSETVALUATORDESC	No Access Checks
IDEVSETVALUATORS	No Access Checks

C.1.1.4 Sball Streams Driver

Space ball is not supported in the evaluated configuration. However, since the streams driver is included in the kernel, the ioctl commands are listed here for completeness.

IDEVCHANGEVALUATORS	No Access Checks
IDEVENABLEBUTTONS	No Access Checks
IDEVENABLEVALUATORS	No Access Checks
IDEVGETBUTTONS	No Access Checks
IDEVGETDEVICEDESC	No Access Checks
IDEVGETTRANSFORM	No Access Checks
IDEVGETVALUATORDESC	No Access Checks
IDEVGETVALUATORS	No Access Checks
IDEVINITDEVICE	No Access Checks
IDEVOTHERCONTROL	No Access Checks
IDEVOTHERQUERY	No Access Checks
IDEVRINGBELL	No Access Checks
IDEVSETPTR	No Access Checks
IDEVSETPTRBOUNDS	No Access Checks

IDEVSETPTRMODE	No Access Checks
IDEVSETTRANSFORM	No Access Checks
IDEVSETVALUATORDESC	No Access Checks
IDEVSETVALUATORS	No Access Checks

C.1.1.5 Dial Streams Driver

Dial and button box are not supported in the evaluated configuration. However, since the streams driver is included in the kernel, the ioctl commands are listed here for completeness.

IDEVCHANGEVALUATORS	No Access Checks
IDEVENABLEBUTTONS	No Access Checks
IDEVENABLEVALUATORS	No Access Checks
IDEVGETBUTTONS	No Access Checks
IDEVGETDEVICEDESC	No Access Checks
IDEVGETLEDS	No Access Checks
IDEVGETSTRDPY	No Access Checks
IDEVGETSTRDPYDESC	No Access Checks
IDEVGETTRANSFORM	No Access Checks
IDEVGETVALUATORDESC	No Access Checks
IDEVGETVALUATORS	No Access Checks
IDEVINITDEVICE	No Access Checks
IDEVPTRCONTROL	No Access Checks
IDEVSETLEDS	No Access Checks
IDEVSETPTR	No Access Checks
IDEVSETPTRBOUNDS	No Access Checks
IDEVSETPTRMODE	No Access Checks
IDEVSETSTRDPY	No Access Checks
IDEVSETTRANSFORM	No Access Checks
IDEVSETVALUATORDESC	No Access Checks
IDEVSETVALUATORS	No Access Checks

C.1.1.6 Keyboard Streams Driver

IDEVENABLEBUTTONS	No Access Checks
IDEVGETBUTTONS	No Access Checks
IDEVGETDEVICEDESC	No Access Checks
IDEVGETKEYMAPDESC	No Access Checks
IDEVGETLEDS	No Access Checks
IDEVINITDEVICE	No Access Checks

IDEVKEYBDCONTROL	No Access Checks
IDEVOTHERCONTROL	No Access Checks
IDEVOTHERQUERY	No Access Checks
IDEVRINGBELL	No Access Checks
IDEVSETLEDS	No Access Checks

C.1.1.7 Mouse Streams Driver

IDEVCHANGEVALUATORS	No Access Checks
IDEVENABLEBUTTONS	No Access Checks
IDEVENABLEVALUATORS	No Access Checks
IDEVGETBUTTONS	No Access Checks
IDEVGETDEVICEDESC	No Access Checks
IDEVGETTRANSFORM	No Access Checks
IDEVGETVALUATORDESC	No Access Checks
IDEVGETVALUATORS	No Access Checks
IDEVINITDEVICE	No Access Checks
IDEVPTRCONTROL	No Access Checks
IDEVSETPTR	No Access Checks
IDEVSETPTRBOUNDS	No Access Checks
IDEVSETPTRMODE	No Access Checks
IDEVSETTRANSFORM	No Access Checks
IDEVSETVALUATORDESC	No Access Checks
IDEVSETVALUATORS	No Access Checks

C.1.2 Driver Ioctl Routines

The following driver ioctl routines are accessed through the cdevsw table:

C.1.2.1 smfdioctl()

Floppy disk drives are not supported in the evaluated configuration. However, since the driver is included in the kernel, the ioctl commands are listed here for completeness.

DIOCDRIVETYPE	No Access Checks
DIOCFMTTRK	No Access Checks
DIOCFORMAT	No Access Checks
DIOCGETVH	No Access Checks
DIOCODRIVETYPE	No Access Checks
DIOCREADCAPACITY	No Access Checks
DIOCSELECT	No Access Checks

DIOCSELFLAGS	No Access Checks
DIOCSENSE	No Access Checks
DIOCTEST	No Access Checks
SMFDEJECT	No Access Checks
SMFDMEDIA	No Access Checks
SMFDNORETRY	No Access Checks
SMFDRECAL	No Access Checks
SMFDREQSENSE	No Access Checks
SMFDSETMODE	No Access Checks

C.1.2.2 dksioctl()

The disk devices are protected from access by untrusted users. Only TCB processes can open a disk device in a properly configured system.

0xdae	No Access Checks
DIOCADDBB	No Access Checks
DIOCDRIVETYPE	No Access Checks
DIOCFORMAT	No Access Checks
DIOCGETVH	No Access Checks
DIOCODRIVETYPE	No Access Checks
DIOCRDEFECTS	No Access Checks
DIOCREADCAPACITY	No Access Checks
DIOCSE	No Access Checks
DIOCSENSE	No Access Checks
DIOCSETVH	No Access Checks
DIOCTEST	No Access Checks

C.1.2.3 lvioctl()

The logical volume devices are protected from access by untrusted users. Only TCB processes can open a logical volume device in a properly configured system.

DIOCGETLV	No Access Checks
DIOCSETLV	No Access Checks

C.1.2.4 qcntlioctl()

QIOCATTACH	No Access Checks
QIOCDETACH	No Access Checks

QIOCSERVICED	No Access Checks

C.1.2.5 tpscioctl()

No ioctl commands currently implemented.

C.1.2.6 klogioctl()

FIONREAD	No Access Checks
FIONBIO	No Access Checks
FIOASYNC	No Access Checks
TIOCSPGRP	No Access Checks
TIOCGPGRP	No Access Checks

C.1.2.7 cnioctl()

TCSETLABEL	**Superuser Only**

Any other ioctl commands are passed to strioctl().

C.1.2.8 syioctl()

All ioctl commands are passed to strioctl().

C.1.2.9 usioctl()

UIOCUNBLOCK	No Access Checks
UIOCAUNBLOCK	No Access Checks
UIOCNOIBLOCK	No Access Checks
UIOCBLOCK	No Access Checks
UIOCABLOCK	No Access Checks
UIOCINIT	No Access Checks
UIOCATTACHSEMA	No Access Checks
UIOCBLOCK	No Access Checks
UIOCNOIBLOCK	No Access Checks
UIOCUNBLOCK	No Access Checks
UIOCABLOCK	No Access Checks
UIOCAUNBLOCK	No Access Checks
UIOCATTACHSEMA	No Access Checks

UIOCINIT	No Access Checks

C.1.2.10 zeroioctl()

No ioctl commands are currently implemented.

C.1.2.11 mmioctl()

No ioctl commands are currently implemented.

C.1.2.12 imonioctl()

The imon device is protected from access by untrusted users. It is not used in the evaluated configuration.

IMONIOC_EXPRESS	No Access Checks
IMONIOC_QTEST	No Access Checks
IMONIOC_REVOKE	No Access Checks

C.1.2.13 fsioctl()

BALLOC	No Access Checks
BFREE	No Access Checks
ICOMMIT	No Access Checks
ILOCK	No Access Checks
IUNLOCK	No Access Checks
TSTALLOC	No Access Checks
TSTFREE	No Access Checks

C.1.2.14 dsioctl()

DS_CLEAR	No Access Checks
DS_CONF	No Access Checks
DS_ENTER	No Access Checks
DS_GET	No Access Checks
DS_RESET	No Access Checks
DS_SET	No Access Checks

C.1.2.15 gfxioctl()

GFX_ATTACH_BOARD	No Access Checks
GFX_BLANKSCREEN	No Access Checks
GFX_DETACH_BOARD	No Access Checks
GFX_DOWNLOAD	No Access Checks
GFX_GETBOARDINFO	No Access Checks
GFX_GETNUM_BOARDS	No Access Checks
GFX_INITIALIZE	No Access Checks
GFX_IS_MANAGED	No Access Checks
GFX_LABEL	No Access Checks
GFX_MAPALL	No Access Checks
GFX_REATTACH_BOARD	No Access Checks
GFX_SET_DIAG_FLAG	No Access Checks
GFX_START	No Access Checks
GFX_SYNC	No Access Checks

C.1.2.16 plpioctl()

FIONBIO	No Access Checks
PLPIOCIGNACK	No Access Checks
PLPIOCREAD	No Access Checks
PLPIOCRESET	No Access Checks
PLPIOCRTO	No Access Checks
PLPIOCSETMASK	No Access Checks
PLPIOCSTATUS	No Access Checks
PLPIOCSTROBE	No Access Checks
PLPIOCTYPE	No Access Checks
PLPIOCWTO	No Access Checks
PLPIOMODE	No Access Checks

C.1.2.17 hdspioctl()

HDSP_RESET	No Access Checks
HDSP_ACTIVATE	No Access Checks
HDSP_FIND_MINOR	No Access Checks
HDSP_ACQUIRE_AUDIO_RB	No Access Checks
HDSP_GET_HRBSTATUS	No Access Checks
HDSP_SET_HRBSTATUS	No Access Checks
HDSP_SET_AUDIO_PARMS	No Access Checks
HDSP_GET_AUDIO_PARMS	No Access Checks

C.2 Debug File System Ioctl Routines

The dbg_ioctl() routine handles ioctl commands for the debug file system. The ioctl commands are:

PIOCRUN	No Access Checks
PIOCGTRACE	No Access Checks
PIOCGHOLD	No Access Checks
PIOCSHOLD	No Access Checks
PIOCSTRACE	No Access Checks
PIOCSSIG	No Access Checks
PIOCGETPR	No Access Checks
PIOCGETU	No Access Checks
PIOCKILL	No Access Checks
PIOCUNKILL	No Access Checks
PIOCMAXSIG	No Access Checks
PIOCGFAULT	No Access Checks
PIOCSFAULT	No Access Checks
PIOCCFAULT	No Access Checks
PIOCACTION	No Access Checks
PIOCCRED	No Access Checks
PIOCGROUPS	No Access Checks
PIOCNMAP	No Access Checks
PIOCMAP	No Access Checks
PIOCOPENM	No Access Checks
PIOCSENTRY	No Access Checks
PIOCSEXIT	No Access Checks
PIOCGENTRY	No Access Checks
PIOCGEXIT	No Access Checks
PIOCSFORK	No Access Checks
PIOCSRLC	No Access Checks
PIOCSET	No Access Checks
PIOCRFORK	No Access Checks
PIOCRRLC	No Access Checks
PIOCRESET	No Access Checks
PIOCGREG	No Access Checks
PIOCSREG	No Access Checks
PIOCGFPREG	No Access Checks
PIOCSFPREG	No Access Checks
PIOCNICE	No Access Checks
PIOCSTOP	No Access Checks
PIOCWSTOP	No Access Checks
PIOCSTATUS	No Access Checks
PIOCNWATCH	No Access Checks

PIOCSWATCH	No Access Checks
PIOCGWATCH	No Access Checks
PIOCUSAGE	No Access Checks

C.3 EFS File System Ioctl Routines

Ioctl commands passed to efs_ioctl() are dispatched to com_ioctl().

C.4 NFS File System Ioctl Routines

Ioctl commands passed to nfs_ioctl() are dispatched to com_ioctl().

C.5 BSD Networking (Socket File System) Ioctl Routines

The soc_ioctl() routine handles ioctl commands for socket objects and network interfaces. The ioctl command is either handled in this routine, dispatched to the network interface ioctl routine, or to a protocol dependent routine.

C.5.1 soc_ioctl()

FIONBIO	No Access Checks
FIOASYNC	No Access Checks
FIONREAD	No Access Checks
SIOCNREAD	No Access Checks
FIOSETOWN	No Access Checks
SIOCSPGRP	No Access Checks
FIOGETOWN	No Access Checks
SIOCGPGRP	No Access Checks
SIOCATMARK	No Access Checks
SIOCGETLABEL	No Access Checks
SIOCSETLABEL	**Superuser Only**
SIOCGETACL	No Access Checks
SIOCSETACL	No Access Checks
SIOCGETUID	No Access Checks
SIOCSETUID	**Superuser Only**

Ioctl commands that are not listed above are passed either to the network interface ioctl routine, ifioctl(), or to the routing ioctl routine, rtioclt(), or to a protocol specific ioctl routine.

C.5.2 ifioctl()

The ifioctl() routine handles those ioctl commands which operate on a network interface:

SIOCADDMULTI	**Superuser Only**
SIOCDARP	**Superuser Only**
SIOCDELMULTI	**Superuser Only**
SIOCGARP	No Access Checks
SIOCGIFCONF	No Access Checks
SIOCGIFFLAGS	No Access Checks
SIOCGIFLABEL	No Access Checks
SIOCGIFMETRIC	No Access Checks
SIOCGIFSTATS	No Access Checks
SIOCGIFUID	No Access Checks
SIOCSARP	**Superuser Only**
SIOCSIFFLAGS	**Superuser Only**
SIOCSIFHEAD	**Superuser Only**
SIOCSIFLABEL	**Superuser Only**
SIOCSIFMETRIC	**Superuser Only**
SIOCSIFSTATS	**Superuser Only**
SIOCSIFUID	**Superuser Only**

Ioctl commands not listed above, or which require interface specific processing are passed to the interface ioctl routines described in Section 1.5.8 "Physical Interface" below.

C.5.3 rtioctl()

The rtioctl() routine handles those ioctl commands that involve routing:

SIOCADDRT	**Superuser Only**
SIOCDELRT	**Superuser Only**
SIOCSETRTINFO	**Superuser Only**
SIOCGETRTINFO	No Access Checks

C.5.4 raw_usrreq()

The raw_usrreq() routine handles ioctl commands that are specific to sockets using raw socket protocol. These requests are passed to rawioctl():

SIOCGIFADDR	No Access Checks
SIOCSIFADDR	No Access Checks
SIOCRAWSTATS	No Access Checks

SIOCADDSNOOP	No Access Checks
SIOCDELSNOOP	No Access Checks
SIOCSNOOPLEN	No Access Checks
SIOCSNOOPING	No Access Checks
SIOCERRSNOOP	No Access Checks
SIOCDRAINMAP	No Access Checks

Ioctl commands not listed above, or which require interface specific processing are passed to the interface ioctl routines described in Section 1.5.8 "Physical Interface" below.

C.5.5 tcp_usrreq(PRU_CONTROL,)

This routine handles ioctl commands that are specific to sockets using tcp protocol:

SIOCSIFADDR	**Superuser Only**
SIOCSIFNETMASK	**Superuser Only**
SIOCSIFDSTADDR	**Superuser Only**
SIOCSIFBRDADDR	**Superuser Only**
SIOCGIFADDR	No Access Checks
SIOCGIFBRDADDR	No Access Checks
SIOCGIFDSTADDR	No Access Checks
SIOCGIFNETMASK	No Access Checks

Ioctl commands not listed above, or which require interface specific processing are passed to the interface ioctl routines described in Section 1.5.8 "Physical Interface" below.

C.5.6 udp_usrreq(PRU_CONTROL,)

This routine handles ioctl commands that are specific to sockets using udp protocol. The upd protocol shares a common code path with tcp for ioctl commands. Interface specifics are passed to the interface ioctl described in Section 1.5.8 "Physical Interface" below.

C.5.7 uipc_usrreq()

This routine handles ioctl commands that are specific to UNIX Domain sockets. No UNIX Domain specific ioctl commands are currently supported.

C.5.8 Physical Interface

When a higher level network interface or protocol routine does not handle an ioctl command, or requires interface specific processing, the command is passed to the ioctl routine for the physical interface. Any access checks are implemented at a higher layer in

the network code.

C.5.8.1 ether_ioctl()

SIOCSIFADDR	No Access Checks
SIOCSIFFLAGS	No Access Checks
SIOCADDMULTI	No Access Checks
SIOCDELMULTI	No Access Checks
SIOCADDSNOOP	No Access Checks
SIOCDELSNOOP	No Access Checks

The ether_ioctl() routine may pass the command to a hardware specific routine, in the evaluated configuration seeq_ioctl().

C.5.8.2 lpioctl()

SIOCSIFADDR	No Access Checks
SIOCADDMULTI	No Access Checks
SIOCDELMULTI	No Access Checks

C.5.8.3 seeq_ioctl()

SIOCADDMULTI	No Access Checks
SIOCDELMULTI	No Access Checks

CONTENTS

1. Task Scope 1

2. Trusted IRIX/B System Subjects 2
2.1 Process 2
2.1.1 Attributes 2
2.1.2 Data 2
2.1.3 Names 3
2.1.4 Manipulation 3
2.2 Share Group 3
2.2.1 Attributes 4
2.2.2 Data 4
2.2.3 Names 4
2.2.4 Manipulation 4

3. Trusted IRIX/B System Kernel Objects 7
3.1 Named Objects 7
3.1.1 File System Object 8
3.1.1.1 Attributes 8
3.1.1.2 Data 9
3.1.1.3 Names 9
3.1.1.4 Manipulation 9
3.1.1.4.1 Block Device 10
3.1.1.4.2 Character Device 10
3.1.1.4.3 Directory 10
3.1.1.4.4 FIFO 11
3.1.1.4.5 File 11
3.1.1.4.6 Pipe 11
3.1.1.4.7 Symbolic Link 12
3.1.1.4.8 UNIX Domain Rendezvous Point 12
3.1.1.4.9 Process File 13
3.1.2 Process Object 14
3.1.2.1 Attributes 14
3.1.2.2 Data 14
3.1.2.3 Names 14
3.1.2.4 Manipulation 15
3.1.2.4.1 Current Process 15
3.1.2.4.2 Another Process 15
3.1.3 System V IPC Object 16
3.1.3.1 Attributes 17
3.1.3.2 Data 17
3.1.3.3 Names 17
3.1.3.4 Manipulation 17
3.1.3.4.1 Message Queue 18
3.1.3.4.2 Semaphore Set 18

3.1.3.4.3 Shared Memory Segment 19
3.1.4 BSD IPC Socket Object 20
3.1.4.1 Attributes 20
3.1.4.2 Data 21
3.1.4.3 Names 21
3.1.4.4 Manipulation 21
3.2 Public Objects 22
3.2.1 System Clock 22
3.2.2 Cycle Counter 23
3.2.3 Domain Name 23
3.2.4 Host Name 23
3.2.5 Host ID 23
3.2.6 Non-Volatile RAM 23
3.2.7 Scheduler 24
3.3 Constant Objects 24
3.3.1 Page Size 24
3.3.2 Descriptor Table Size 24

4. Object Reuse 25
4.1 File System Objects 25
4.2 Process Objects 25
4.3 System V IPC Objects 25
4.4 BSD IPC Socket Objects 25

5. Policies 26
5.1 File System Policies 26
5.1.1 Path Name Policies 26
5.1.1.1 Data Read Access Policy - Path Name 26
5.1.1.2 Data Write Access Policy - Path Name 26
5.1.1.3 Attribute Read Access Policy - Path Name 26
5.1.1.4 Attribute Write Access Policy - Path Name 27
5.1.1.5 Execute Access Policy - Path Name 27
5.1.2 File Descriptor Policies 27
5.1.2.1 Read Access Policy - File Descriptor 27
5.1.2.2 Data Write Access Policy - File Descriptor 27
5.1.2.3 Attribute Write Access Policy - File Descriptor 28
5.2 Intraprocess Communication Policies 28
5.2.1 Data Read and Write Access Policy 28
5.2.2 Attribute Read Access Policy 28
5.2.3 Attribute Write Access Policy 28
5.2.3.1 User ID Write 28
5.2.3.2 Group ID Write 28
5.2.3.3 Session ID Write 28
5.2.3.4 Group Access List Write 29
5.2.3.5 Label Write 29
5.3 Interprocess Communication Policies 29

5.3.1 Process Policies 29
5.3.1.1 Data Read Access Policy - Process 29
5.3.1.2 Data Write Access Policy - Process 29
5.3.1.3 Attribute Read Access Policy - Process 29
5.3.1.4 Attribute Write Access Policy - Process 29
5.3.1.4.1 Process Group ID Write 30
5.3.1.4.2 Signal 30
5.3.2 System V IPC Policies 30
5.3.2.1 KEYs 30
5.3.2.1.1 Attribute Read Access Policy - System V IPC 31
5.3.2.1.2 Attribute Write Access Policy - System V IPC 31
5.3.2.2 IDs 31
5.3.2.2.1 Read Access Policy - System V IPC 31
5.3.2.2.2 Data Write Access Policy - System V IPC 31
5.3.2.2.3 Attribute Write Access Policy - System V IPC 31
5.3.3 BSD IPC Policies 32
5.3.3.1 Read Access Policy - BSD IPC 32
5.3.3.2 Data Write Access Policy - BSD IPC 32
5.3.3.3 Attribute Write Access Policy - BSD IPC 32
5.4 Public Object Policies 33
5.5 Superuser Policies 33

6. Modules 34
6.1 BSDgetpgrp 34
6.2 BSDsetpgrp 34
6.3 accept 34
6.4 access 34
6.5 adjtime 35
6.6 alarm 35
6.7 async_daemon 35
6.8 bind 35
6.9 brk 35
6.10 bsdgettime 35
6.11 cachectl 36
6.12 cacheflush 36
6.13 chdir 36
6.14 chmod 36
6.15 chown 36
6.16 chroot 36
6.17 close 36
6.18 connect 37
6.19 creat 37

6.20 dup 37
6.21 exec 37
6.22 exece 37
6.23 exit 38
6.24 exportfs 38
6.25 fchdir 38
6.26 fchmod 38
6.27 fchown 38
6.28 fcntl 38
6.29 fork 39
6.30 fstat 39
6.31 fstatfs 39
6.32 fsync 39
6.33 ftruncate 39
6.34 getdents 39
6.35 getdomainname 40
6.36 getgid 40
6.37 gethostid 40
6.38 gethostname 40
6.39 getitimer 40
6.40 getmsg 40
6.41 getpagesize 40
6.42 getpeername 40
6.43 getpid 41
6.44 getrlimit 41
6.45 getsockname 41
6.46 getsockopt 41
6.47 getuid 41
6.48 ioctl 41
6.48.1 I_PEEK 42
6.48.2 I_STR 42
6.48.3 SIOCADDMULTI 42
6.48.4 SIOCADDRT 42
6.48.5 SIOCDARP 42
6.48.6 SIOCDELMULTI 42
6.48.7 SIOCDELRT 42
6.48.8 SIOCSARP 42
6.48.9 SIOCSETLABEL 42
6.48.10 SIOCSETRTINFO 42
6.48.11 SIOCSETUID 43
6.48.12 SIOCSIFADDR 43
6.48.13 SIOCSIFBRDADDR 43
6.48.14 SIOCSIFDSTADDR 43
6.48.15 SIOCSIFFLAGS 43
6.48.16 SIOCSIFHEAD 43
6.48.17 SIOCSIFLABEL 43

6.48.18 SIOCSIFMETRIC 43
6.48.19 SIOCSIFNETMASK 43
6.48.20 SIOCSIFSTATS 43
6.48.21 SIOCSIFUID 44
6.48.22 TCSETLABEL 44
6.48.23 TIOCSPGRP 44
6.48.24 TIOCSTI 44
6.49 kill 44
6.50 link 44
6.51 listen 44
6.52 lstat 44
6.53 madvise 45
6.54 mkdir 45
6.55 mknod 45
6.56 mmap 45
6.57 mount 45
6.58 mprotect 46
6.59 msgsys 46
6.59.1 MSGCTL 46
6.59.1.1 IPC_RMID 46
6.59.1.2 IPC_SET 46
6.59.1.3 IPC_STATE 46
6.59.2 MSGGET 46
6.59.3 MSGRCV 46
6.59.4 MSGSND 47
6.60 msync 47
6.61 munmap 47
6.62 nfs_getfh 47
6.63 nfssvc 47
6.64 nice 47
6.65 open 48
6.66 pagelock 48
6.67 pause 48
6.68 pipe 48
6.69 plock 48
6.70 poll 48
6.71 prctl 49
6.72 procblk 49
6.72.1 0 49
6.72.2 1 49
6.72.3 2 49
6.72.4 3 49
6.72.5 4 49
6.72.6 5 50
6.73 profil 50
6.74 ptrace 50

6.75 putmsg 50
6.76 quotactl 50
6.76.1 Q_ACTIVATE 50
6.76.2 Q_GETQUOTA 50
6.76.3 Q_QUOTAOFF 51
6.76.4 Q_QUOTAON 51
6.76.5 Q_SETQLIM 51
6.76.6 Q_SETQUOTA 51
6.76.7 Q_SYNC 51
6.77 read 51
6.78 readlink 51
6.79 recv 51
6.80 recvfrom 52
6.81 recvmsg 52
6.82 rename 52
6.83 rmdir 52
6.84 seek 52
6.85 select 53
6.86 semsys 53
6.86.1 SEMCTL 53
6.86.1.1 GETALL 53
6.86.1.2 GETCNT 53
6.86.1.3 GETPID 53
6.86.1.4 GETVAL 53
6.86.1.5 GETZCNT 53
6.86.1.6 IPC_RMID 54
6.86.1.7 IPC_SET 54
6.86.1.8 IPC_STATE 54
6.86.1.9 SETALL 54
6.86.1.10 SETVAL 54
6.86.2 SEMGET 54
6.86.3 SEMOP 54
6.86.3.1 sem_op=0 54
6.86.3.2 sem_op≠0 55
6.87 send 55
6.88 sendmsg 55
6.89 sendto 55
6.90 setdomainname 55
6.91 setgid 55
6.92 sethostid 55
6.93 sethostname 56
6.94 setitimer 56
6.95 setpgrp 56
6.96 setregid 56
6.97 setreuid 56
6.98 setrlimit 56

6.99 setsockopt 56
6.100 setuid 57
6.101 sgikopt 57
6.102 sginap 57
6.103 shmsys 57
6.103.1 SHMAT 57
6.103.2 SHMCTL 57
6.103.2.1 IPC_RMID 57
6.103.2.2 IPC_SET 57
6.103.2.3 IPC_STATE 58
6.103.2.4 SHM_LOCK 58
6.103.2.5 SHM_UNLOCK 58
6.103.3 SHMDT 58
6.103.4 SHMGET 58
6.104 shutdown 58
6.105 signal 58
6.106 sigreturn 58
6.107 socket 59
6.108 socketpair 59
6.109 sproc 59
6.110 stat 59
6.111 statfs 59
6.112 stime 60
6.113 symlink 60
6.114 sync 60
6.115 sysacct 60
6.116 sysfs 60
6.117 sysmips 60
6.117.1 FLUSH_CACHE 60
6.117.2 MIPS_FIXADE 61
6.117.3 MIPS_FPSIGINTR 61
6.117.4 SETNAME 61
6.117.5 SMIPSSWPI 61
6.117.5.1 add 61
6.117.5.2 delete 61
6.117.5.3 list 61
6.117.6 STIME 61
6.118 sysmp 61
6.118.1 MP_CLOCK 61
6.118.2 MP_EMPOWER 62
6.118.3 MP_ISOLATE 62
6.118.4 MP_KERNADDR 62
6.118.5 MP_MUSTRUN 62
6.118.6 MP_NAPROCS 62
6.118.7 MP_NONPREEMPTIVE 62
6.118.8 MP_NPROCS 62

6.118.9 MP_PGSIZE 62
6.118.10 MP_PREEMPTIVE 62
6.118.11 MP_RESTRICT 62
6.118.12 MP_RUNANYWHERE 63
6.118.13 MP_SAGET 63
6.118.14 MP_SAGET1 63
6.118.15 MP_SASZ 63
6.118.16 MP_SCHED 63
6.118.17 MP_STAT 63
6.118.18 MP_UNISOLATE 63
6.119 syssgi 63
6.119.1 SGI_BDFLUSHCNT 63
6.119.2 SGI_DISABLE_FASTCLOCK 64
6.119.3 SGI_DISABLE_FTIMER 64
6.119.4 SGI_ENABLE_FASTCLOCK 64
6.119.5 SGI_ENABLE_FTIMER 64
6.119.6 SGI_GETGROUPS 64
6.119.7 SGI_GETLABEL 64
6.119.8 SGI_GETNVRAM 64
6.119.9 SGI_GETPLABEL 64
6.119.10 SGI_GETPSOACL 65
6.119.11 SGI_GETTIMETRIM 65
6.119.12 SGI_IDBG 65
6.119.13 SGI_INVENT 65
6.119.14 SGI_LOADATTR 65
6.119.15 SGI_NETPROC 65
6.119.16 SGI_NFSCNVT 66
6.119.17 SGI_PATHCONF 66
6.119.18 SGI_PLANGMOUNT 66
6.119.19 SGI_PROCSZ 66
6.119.20 SGI_QUERY_CYCLECNTR 66
6.119.21 SGI_QUERY_FTIMER 66
6.119.22 SGI_R4K_CERRS 67
6.119.23 SGI_RDNAME 67
6.119.24 SGI_RDUBLK 67
6.119.25 SGI_READB 67
6.119.26 SGI_RECVLMSG 67
6.119.27 SGI_RECVLUMSG 67
6.119.28 SGI_REVOKE 68
6.119.29 SGI_RMI_FIXECC 68
6.119.30 SGI_RUSAGE 68
6.119.31 SGI_SATCTL 68
6.119.31.1 SATCTL_AUDIT_OFF 68
6.119.31.2 SATCTL_AUDIT_ON 68
6.119.31.3 SATCTL_AUDIT_QUERY 68
6.119.31.4 SATCTL_GET_SAT_ID 69

6.119.31.5 SATCTL_LOCALAUDIT_OFF 69
6.119.31.6 SATCTL_LOCALAUDIT_ON 69
6.119.31.7 SATCTL_LOCALAUDIT_QUERY 69
6.119.31.8 SATCTL_SET_SAT_ID 69
6.119.31.9 SGI_REGISTER_SATD 69
6.119.32 SGI_SATREAD 69
6.119.33 SGI_SATWRITE 69
6.119.34 SGI_SET_AUTOPWRON 70
6.119.35 SGI_SETGROUPS 70
6.119.36 SGI_SETLABEL 70
6.119.37 SGI_SETLED 70
6.119.38 SGI_SETNVRAM 70
6.119.39 SGI_SETPGID 70
6.119.40 SGI_SETPLABEL 71
6.119.41 SGI_SETPSOACL 71
6.119.42 SGI_SETSID 71
6.119.43 SGI_SETTIMEOFDAY 71
6.119.44 SGI_SETTIMETRIM 71
6.119.45 SGI_SIGACTION 71
6.119.46 SGI_SIGALTSTACK 71
6.119.47 SGI_SIGSTACK 71
6.119.48 SGI_SIGSTATUS 72
6.119.49 SGI_SIGPENDING 72
6.119.50 SGI_SIGPROCMASK 72
6.119.51 SGI_SIGSUSPEND 72
6.119.52 SGI_SPROFIL 72
6.119.53 SGI_SSYNC 73
6.119.54 SGI_SYSCONF 73
6.119.55 SGI_SYSID 73
6.119.56 SGI_TUNE 73
6.119.57 SGI_UNLOADATTR 73
6.119.58 SGI_WAIT4 73
6.119.59 SGI_WRITEB 73
6.120 texturebind 74
6.121 time 74
6.122 times 74
6.123 truncate 74
6.124 uadmin 74
6.125 ulimit 74
6.125.1 1 75
6.125.2 2 75
6.125.3 3 75
6.125.4 4 75
6.126 umask 75
6.127 umount 75
6.128 unlink 75

6.129 utime 76
6.130 utssys 76
6.130.1 uname 76
6.130.2 ustat 76
6.131 vhangup 76
6.132 wait 76
6.133 wait3 76
6.134 write 76

A. Appendix A: Objects and Their Manipulation 78

B. Appendix B: Use By Policy 85

C. Appendix C: Supported *ioctl* Summary 96
C.1 Common File System Ioctl Routines 96
C.1.1 Streams Ioctl Routines 96
C.1.1.1 Calcomp Streams Driver 97
C.1.1.2 Wacom Streams Driver 98
C.1.1.3 Tablet Streams Driver 99
C.1.1.4 Sball Streams Driver 99
C.1.1.5 Dial Streams Driver 100
C.1.1.6 Keyboard Streams Driver 100
C.1.1.7 Mouse Streams Driver 101
C.1.2 Driver Ioctl Routines 101
C.1.2.1 smfdioctl() 101
C.1.2.2 dksioctl() 102
C.1.2.3 lvioctl() 102
C.1.2.4 qcntlioctl() 102
C.1.2.5 tpscioctl() 103
C.1.2.6 klogioctl() 103
C.1.2.7 cnioctl() 103
C.1.2.8 syioctl() 103
C.1.2.9 usioctl() 103
C.1.2.10 zeroioctl() 104
C.1.2.11 mmioctl() 104
C.1.2.12 imonioctl() 104
C.1.2.13 fsioctl() 104
C.1.2.14 dsioctl() 104
C.1.2.15 gfxioctl() 105
C.1.2.16 plpioctl() 105
C.1.2.17 hdspioctl() 105
C.2 Debug File System Ioctl Routines 106
C.3 EFS File System Ioctl Routines 107
C.4 NFS File System Ioctl Routines 107
C.5 BSD Networking (Socket File System) Ioctl Routines 107
C.5.1 soc_ioctl() 107
C.5.2 ifioctl() 108

C.5.3 rtioctl() 108
C.5.4 raw_usrreq() 108
C.5.5 tcp_usrreq(PRU_CONTROL,) 109
C.5.6 udp_usrreq(PRU_CONTROL,) 109
C.5.7 uipc_usrreq() 109
C.5.8 Physical Interface 109
C.5.8.1 ether_ioctl() 110
C.5.8.2 lpioctl() 110
C.5.8.3 seeq_ioctl() 110

LIST OF FIGURES

Figure 3.1-1. Named Object 8
Figure 3.1.1-1. File System Object 8
Figure 3.1.2-1. Process Object 14
Figure 3.1.3-1. System V IPC Object 17
Figure 3.1.4-1. BSD IPC Socket Object 20

LIST OF TABLES

TABLE A-1. File System Object Manipulation 79
TABLE A-2. Current Process Object Manipulation 81
TABLE A-3. Another Process Object Manipulation 81
TABLE A-4. System V IPC Message Queue Object Manipulation 82
TABLE A-5. System V IPC Semaphore Set Object Manipulation 82
TABLE A-6. System V IPC Shared Memory Segment Object Manipulation 82
TABLE A-7. BSD IPC Socket Object Manipulation 83
TABLE A-8. Public Object Manipulation 84
TABLE A-9. Constant Object Manipulation 84
TABLE B-1. Path Name Policy Enforcement 86
TABLE B-2. File Descriptor Policy Enforcement 87
TABLE B-3. Intraprocess Communication Policy Enforcement 88
TABLE B-4. Process IPC Policy Enforcement 89
TABLE B-5. System V IPC Policy Enforcement 90
TABLE B-6. BSD IPC Policy Enforcement 91
TABLE B-7. Superuser Only Policy Enforcement 92
TABLE B-8. Superuser Impact 95
TABLE C-1. File System Type to Routine Correspondence 96