POSIX.1 6

Portable Operating System Interface for Computer Environments (POSIX.1)

The IEEE Std 1003.1-1990 Portable Operating System Interface Part 1 (POSIX .1) is part of the POSIX series of standards for applications and user interfaces to open systems. POSIX.1 has also been adopted as international standard ISO/IEC 9945-1: 1990 by the International Organization for Standardization/International Electrotechnical Commission. It defines the applications interface to basic system services for input/output, file system access and process management, using the C programming language, which establishes standard semantics and syntax. Because this interface enables application writers to write portable applications, it has been named POSIX, an acronym for Portable Operating System Interface. POSIX.1 is based on the UNIX operating system and is derived from efforts of the /usr/group Standards Committee. Within this chapter the POSIX.1 standard is referred to in places as "the standard."

Amending POSIX.1: The IEEE Standard 1003.1b

IEEE Std 1003.1b–1993 is a standard that amends POSIX.1 to include extensions in support of realtime applications. (In earlier versions of this guide, IEEE Std 1003.1b was referred to as IEEE Std 1003.4 or POSIX.4. IEEE Std 1003.1b is now the preferred reference to this standard.) These extensions were ratified by the IEEE in September 1993. The functionality of SunOS 5.4 is intended to provide compliance and support of POSIX.1 as amended by IEEE Std 1003.1b–1993.

6

Scope

To comply with Section 1.3.1.2 (*Documentation*), this chapter describes the behavior of features in the SunOS 5.4 operating system which are described in the standard as *implementation-defined* or for which it is stated that implementations may vary. It does not describe any extensions or enhancements outside the scope of the standard.

Note – Section 2.2.1.2 of the standard defines the term *implementation-defined* as follows: "An indication that the implementation shall define and document the requirements for correct program constructs and correct data of a value or behavior."

The information contained within this chapter does not replace the standard; rather, it serves as an adjunct to the standard for supplying the technical information needed by application developers to write source code within the SunOS 5.4 operating system framework.

C Standard Compliance

The SunOS 5.4 operating system conforms to POSIX.1. The C language compiler and libraries conform to ANSI C.

Audience

This explication is for the experienced C programmer who, when writing an applications program designed to conform to the standard, needs to know the specific behavior of the *implementation-defined* features mentioned in the standard.

Each subsection is prefaced by the appropriate section taken directly from the standard and has the corresponding section number attached to the title.

Note – For maximum portability, applications should not depend upon any particular behavior that is *implementation-defined*.

Notation Used in the Remainder of this Chapter

The following format is used to identify which passage of text is quoted from the standard and which passage of text describes how the SunOS 5.4 operating system implements that area.

- **P.** stands for POSIX.
- **S.** stands for SunOS 5.4.

Section numbers cited in the remainder of this chapter correspond to those of the standard. When creating an application program, this format will help you to quickly locate additional information that you need from the standard.

Implementation-Defined Areas of POSIX.1

POSIX.1 Section 1, General

1.3.1 Implementation Conformance

1.3.1.1 Requirements
P. The conformance document shall define an environment in which an application can be run with the behavior specified by the standard.

S. To configure Solaris to run with the behavior specified by the standard, execute the following steps:

1. Edit the `/etc/saf/zsmon/_pmtab` file to turn off the `ttysoftcarrier` detect by changing the `ttya` and `ttyb` fields from `:y:` to `:n:`. (The colons (:) act as field separators).
2. In the last line of the `/kernel/drv/options.conf` file, revise the `:bd:` entry to `:cbd:` (The colons (:) act as field separators).
3. Disable `ypbind` to allow rebooting of the system:
   a. `cd /usr/lib/netsvc/yp`
   b. `mv ypbind ypbind-`
4. Set the `eeprom` variables that affect the `tty`:
   a. On the keyboard, hit STOP-A to display the `prom` prompt.

b. At the prompt, execute the following steps:

```
setenv ttya-ignore-cd false
setenv ttyb-ignore-cd false
setenv ttya-rts-dtr-off false
setenv ttyb-rts-dtr-off false
```

5. Reboot the system

1.3.1.2 Documentation
P. The conformance document shall contain a statement that indicates the full name, number, and date of the standard that applies.

S. The SunOS 5.4 operating system is a conforming implementation as defined in Section 1.3.1.2 (*Documentation)* of the IEEE Std 1003.1-1990 *Portable Operating System Interface (POSIX)-Part 1: System Application Program Interface [C Language].*

P. The conformance document may also list international software standards that are available for use by a Conforming POSIX.1 Application.

S. The ANSI X3.159-1989 C Language Standard.

1.3.3.2 C Standard Language Dependent System Support
P. Implementors shall meet the requirements of Section 8 using for reference the C Standard {2}. Implementors shall clearly document the version of the C Standard {2} referenced in fulfilling the requirements of Section 8.

S. The system provides an ANSI C Standard Language Binding as specified by X3.159-1989. This language binding is accessed by specifying either -Xa or -Xc on the cc command line.

POSIX.1 Section 2, Terminology and General Requirements

2.2.2 General Terms

2.2.2.4: appropriate privileges
P. An *implementation-defined* means of associating privileges with a process with regard to the function calls and function call options defined in the standard that need special privileges. There may be zero or more such means.

S. Appropriate privileges depend on file permissions and the user ID that executed the process; logging in as root or any user with a UID equal to zero; issuing the `su command` to change the UID to root or any user with UID equal to zero; successful execution of a file with the S_ISUID bit set and UID equal to zero.

2.2.2.9: character special file
P. One specific type of character special file is a terminal device file, whose access is defined in 7.1. Other character special files have no structure defined by this part of ISO/IEC 9945, and their use is unspecified by this part of ISO/IEC 9945.

S. In addition to terminal device files, the `/dev/ksyms` character special file is available. The `/dev/ksyms` structure is described in the `ksyms`(7) `man` page.

2.2.2.55: parent process ID
P. The parent process ID of a process is the process ID of its creator, for the lifetime of the creator. After the creator's lifetime has ended, the parent process ID is the process ID of an *implementation-defined* system process.

S. If a child process continues to exist after its creator process ceases to exist, the child process is inherited by `init`. The `init` process ID is 1.

2.2.2.57: pathname
P. A pathname that begins with two successive slashes may be interpreted in an *implementation-defined* manner.

S. Multiple successive slashes are considered the same as one slash.

2.2.2.68: process lifetime
P. When another process executes a `wait()` or `waitpid()` function for an inactive process, the remaining resources are returned to the system.

S. All resources except the session ID, the process ID and the process group ID are returned to the system.

2.2.2.69: read-only file system
P. A file system that has *implementation-defined* characteristics restricting modifications.

S. A read-only file system does not allow for modification of its files or directories.

2.2.2.83: supplementary group ID
P. Whether a process's effective group ID is included in or omitted from its list of supplementary group IDs is unspecified.

S. A process's effective group ID is included in its list of supplementary group IDs.

2.3 General Concepts

2.3.1: extended security controls
P. The access control and privilege mechanisms have been defined to allow *implementation-defined* extended security controls.

S. No extended security controls are supported.

2.3.2: file access permissions
P. Implementations may provide additional or alternate file access control mechanisms, or both.

S. There is no additional or optional file access control mechanism.

2.3.5: file times update
P. An implementation may update fields that are marked for update immediately, or may update such fields periodically.

S. The UFS file system updates periodically.

2.4 Error Numbers

P. Implementations may support additional errors not included in this clause, may generate errors included in this clause under circumstances other than those described in this clause or may contain extensions or limitations that prevent some errors from occurring.

S. In addition to the errors listed in this clause, Solaris supports the following errors:

Error	Error Code	Condition
EBADMSG	77	Message waiting to be read on a data stream is unreadable.
EMULTIHOP	74	Components of path require hopping to multiple remote machines and the file system does not allow it.
ENOLINK	67	Path points to remote machine; link to that machine has been severed or is no longer active.
ENOSR	63	Insufficient streams memory resources available in the system.
EOVERFLOW	79	Value too large to be stored in data type.

[EFAULT]
P. The reliable detection of this error is *implementation-defined*; however, implementations that do detect this condition shall use this value.

S. The functions listed below reliably detect a bad address and return EFAULT when the address is not in a page mapped into the process.

```
access          chdir           chmod
chown           clock_settime   clock_getres
creat           execl           execle
execlp          execv           execve
execvp          fcntl           fstat
getgroups       link            mkdir
nanosleep       open            read
rmdir           rename          sigpending
sigprocmask     sigsuspend      sigaction
sigwaitinfo     sigtimedwait    sigqueue
stat            tcgetattr       tcsetattr
timer_create    timer_gettime   timer_settime
times           uname           unlink
utime           write
```

[EFBIG]
P. The size of a file would exceed an *implementation-defined* maximum file size.

S. The maximum file size is defined by the `setrlimit()` function and can be retrieved by the `getrlimit()` function.

2.5 Primitive System Data Types

`<sys/types.h>`
P. Some data types used by the various system functions are not defined as part of this standard, but are *defined by the implementation*.

S. Additional fundamental data types are:

`uchar_t`	`ushort_t`	`o_mode_t`
`uint_t`	`ulong_t`	`o_dev_t`
`caddr_t`	`daddr_t`	`o_uid_t`
`major_t`	`minor_t`	`o_gid_t`
`key_t`	`o_nlink_t`	`hostid_t`
`addr_t`	`cnt_t`	`o_ino_t`
`label_t`	`paddr_t`	`use_t`
`sysid_t`	`index_t`	`lock_t`
`boolean_t`	`k_sigset_t`	`k_fltset_t`
`id_t`	`o_pid_t`	`clock_t`
`wchar_t`		

2.6 Environment Description

P. Environment variable names used or created by an application should consist solely of characters from the portable filename character set. Other characters may be permitted by an implementation; applications shall tolerate the presence of such names.

S. Any character except `NULL` and "=" is permitted; however applications should restrict characters to that of the portable filename character set to ensure portability.

2.7 C Language Definitions

2.7.2. POSIX.1 Symbols
P. Implementations, future versions of this part of ISO/IEC 9945, and other standards may define additional feature test macros.

S. Additional defined feature test macros: `_XOPEN_SOURCE`, `_POSIX_C_SOURCE`, `__REENTRANT` and `_KERNEL`. Use of these macros is described in the *X/Open Portability Guide Issue 3* and IEEE Std 1003.1b–1993.

2.8 Numerical Limits

P. The conformance document shall describe the limit values found in the <`limits.h`> header, stating values, the conditions under which those values may change, and the limits of those variations, if any.

S. <`limits.h`> contains the following magnitude limitations:

Name	Value	Comments
`AIO_LISTIO_MAX`	Undefined. Value may be configurable in future.	Maximum number of I/O operations in a single list I/O call supported by the implementation.
`AIO_MAX`	Undefined. Value may be configurable in future.	Maximum number of outstanding asynchronous I/O operations supported by the implementaton.
`AIO_PRIO_DELTA_MAX`	Undefined. Value may be configurable in future.	The maximum amount by which a process can decrease its asynchronous I/O priority level from its own scheduling priority.
`ARG_MAX`	1048320	Maximum length of arguments for the *exec* functions, in bytes, including environment data.
`CHILD_MAX`	Configurable with minimum value 25.	Maximum number of simultaneous processes per real user ID.
`DELAYTIMER_MAX`	Undefined. Value may be configurable in future.	Maximum number of timer expiration overruns.
`LINK_MAX`	32767	Maximum value of a files link count.
`MAX_CANON`	256	Maximum number of bytes in a terminal canonical input line.
`MAX_INPUT`	512	Minimum number of bytes for which space will be available in a terminal input queue; therefore, the maximum number of bytes a portable application may require to be typed.
`MQ_OPEN_MAX`	Undefined. Value may be configurable in future.	The maximum number of open message queue descriptors a process may hold.

MQ_PRIO_MAX	Undefined. Value may be configurable in future.	The maximum number of message priorities supported by the implementation.
NAME_MAX	Undefined. Depends on underlying file system type.	Maximum number of bytes in a file name (not a string length; count excludes a terminating null).
NGROUPS_MAX	16	Maximum number of simultaneous supplementary groups IDs per process.
OPEN_MAX	Default maximum 64 can be raised or lowered by calling setrlimit().	Maximum number of files that one process can have open at one time.
PAGESIZE	Undefined. Depends on system hardware.	Granularity in bytes of memory mapping and process memory locking.
PATH_MAX	1024	Maximum number of bytes in a pathname (not a string length; count excludes a terminating null).
PIPE_BUF	5120	Maximum number of bytes that can be written atomically when writing to a pipe.
RTSIG_MAX	Undefined. Value may be configurable in future.	Maximum number of realtime signals reserved for application use in this implementation.
SEM_NSEMS_MAX	Undefined. Value may be configurable in future.	Maximum number of semaphores that a process may have.
SEM_VALUE_MAX	Undefined. Value may be configurable in future.	Maximum value a semaphore may have.
SIGQUEUE_MAX	Undefined. Value may be configurable in future.	Maximum number of queued signals that a process may send and have pending at the receiver(s) at any time.
STREAM_MAX	Not defined. Depends on OPEN_MAX and number of files open not using fopen().	The number of streams that one process can have open at one time. If defined, it shall have the same value as {FOPEN_MAX} from the C Standard {2}.
SSIZE_MAX	2147483647	The maximum value that can be stored in an object of type *ssize_t*.
TIMER_MAX	Undefined. Value may be configurable in future.	Maximum number of timers per process supported by the implementation.

TZNAME_MAX	Undefined. Time zone name length is limited only by address space.	Maximum number of bytes supported for the name of a time zone. (Not of the TZ variable).

2.9 Symbolic Constants

P. The conformance document shall describe the limit values found in the <unistd.h> header, stating values, the conditions under which those values may change, and the limits of those variations, if any.

S. <unistd.h> contains the following values:

Name	Value	Comments
_POSIX_CHOWN_RESTRICTED*	Not defined. Depends on underlying file system type.	The use of the chown () function is restricted to a process with appropriate privileges, and to changing the group ID of a file only to the effective group ID of the process or to one of its supplementary group IDS.
_POSIX_FSYNC	1	fsync() is supported.
_POSIX_JOB_CONTROL	1	Job control is supported.
_POSIX_MAPPED_FILES	1	Mapped files are supported.
_POSIX_MEMLOCK	1	Memory locking is supported.
_POSIX_MEMLOCK_RANGE	1	Memory range locking is supported.
_POSIX_MEMORY_PROTECTION	1	Memory protection is supported.
_POSIX_NO_TRUNC*	Not defined. Depends on underlying file system type.	Pathname components longer than NAME_MAX generate an error.
_POSIX_REALTIME_SIGNALS	1	Realtime signal extension is supported.
_POSIX_SAVED_IDS	1	Saved IDs is supported.
_POSIX_SYNCHRONIZED_IO	1	Synchronized I/O is provided.
_POSIX_TIMERS	1	Clocks and timers are supported.
_POSIX_VERSION	199309L	Solaris conforms to IEEE Standard 1003.1-1990, as amended by the IEEE Standard for Realtime Extensions approved Sept. 1993.

_POSIX_VDISABLE*	0	Terminal special characters can be disabled using this character value.

* _POSIX_CHOWN_RESTRICTED and _POSIX_NO_TRUNC apply to all files on a native SunOS filesystem. _POSIX_VDISABLE applies to terminal files.

POSIX.1 Section 3, Process Primitives

3.1.1.2 Process Creation: Description

P. For the SCHED_FIFO and SCHED_RR scheduling policies, the child process shall inherit the policy and priority settings of the parent process during a fork() function. For other scheduling policies, the policy and priority settings on fork() are *implementation-defined*.

S. All new child processes inherit the parent's scheduling policy and parameters.

P. The child process has its own copy of the parent's open directory streams. Each open directory stream in the child process may share directory stream positioning with the corresponding directory stream of the parent.

S. Each open directory stream in the child process does not share directory stream positioning with the corresponding directory stream of the parent.

3.1.1.4 Errors

P. For each of the following conditions, if the condition is detected, the fork() function shall return -1 and set errno to the corresponding value:

[ENOMEM]	The process requires more space than the system is able to supply.

S. The fork() function detects the conditions and returns the corresponding errno value for [ENOMEM].

3.1.2.2 Execute a File: Description

P. The argument *file* is used to construct a pathname that identifies the new process image file. If the *file* argument contains a slash character, the *file* argument shall be used as the pathname for this file. Otherwise, the path prefix for this file is obtained by a search of the directories passed as the environment variable `PATH` (see 2.6). If this environment variable is not present, the results of the search are *implementation-defined*.

S. When `PATH` is not set, SunOS 5.4 supplies a default search path:

`/usr/sbin:/usr/bin` (if the real or effective UID is root.)

`/usr/bin:` (if neither the real nor the effective UID is that of root.)

P. The number of bytes available for the new process's combined argument and environment lists is `{ARG_MAX}`. The *implementation shall specify* in the system documentation whether any combination of null terminators, pointers, or alignment bytes are included in this total.

S. `{ARG_MAX}` is retrieved using the `sysconf()` function. This value includes the total number of bytes available for a new process' arguments, environment, and stack. {ARG_MAX} also includes initial pointers into the argument and environment vectors.

The space required for the arguments, environment and stack by an `execve()` call is determined by the following formula:

`space` = (((*na* + 4) * *bpw*) + *nc*) + *click*

space is then rounded up to the next click boundary. A click is the number of bytes that the system's memory-management facilities treats as a single unit. For all machines on which the SunOS 5.4 operating system is supported, one click equals 4096 bytes.

na is the count of arguments and environment variables. *bpw* is the number of bytes per word: 4 on all Sun systems. *nc* is the count of bytes in the argument and the environment vectors, including null terminators, and rounded up to the next word boundary.

P. For the `SCHED_FIFO` and `SCHED_RR` scheduling policies, the policy and priority settings shall not be changed by a call to an `exec` function. For other scheduling policies, the policy and priority settings on `exec` are *implementation-defined*.

S. For all scheduling policies, the policy and scheduling parameters are not changed by a call to an `exec` function.

P. Any outstanding asynchronous I/O operations may be canceled.Those asynchronous I/O operations which are not canceled shall complete as if the `exec` function had not yet occurred, but any associated signal notifications shall be suppressed. It is unspecified whether the `exec` function itself blocks awaiting such I/O completion. In no event, however, shall the new process image created by the `exec` function be affected by the presence of outstanding asynchronous I/O operations at the time the `exec` function is called. Whether any I/O is canceled, and which I/O may be canceled upon `exec` is *implementation-defined*.

S. Any cancelable asynchronous I/O operations are canceled.

3.1.2.4 Execute a File: Errors

P. If any of the following conditions occur, the `exec` functions shall return -1 and set `errno` to the corresponding value:

[EACCES]	Search permission is denied for a directory listed in the path prefix of the new process image file, or the new process image file denies execution permission, or the new process image file is not a regular file and the implementation does not support execution of files of its type.

S. Only regular files are supported by the `exec` function.

P. For each of the following conditions, if the condition is detected, the `exec` functions shall return -1 and return the corresponding value to `errno`:

[ENOMEM]	The new process image requires more memory than is allowed by the hardware or system-imposed memory management constraints.

S. The `exec` functions detect the conditions and return the corresponding `errno` value for [ENOMEM].

3.2.1.2 Wait for Process Termination: Description

`wait()`
P. An *implementation may define* additional circumstances under which `wait()` or `waitpid()` reports status. In these cases the interpretation of the reported status is *implementation-defined*.

S. A child that is being traced stops because it has reached a break point. `WIFSTOPPED(status)` will be true and `WSTOPSIG(status)` will yield the signal that caused the process to stop.

If a child that was formerly stopped by Job Control was continued, `WIFCONTINUED` (status) will be true.

The above will not be true unless the process is tracing a child. See the `proc(4) man` page for more information.

3.2.2.2 Terminate a Process: Description

`exit()`
P. Children of a terminated process shall be assigned a new parent process ID, corresponding to an *implementation-defined* system process.

S. The child's parent process ID becomes 1 which is the process ID of the `init` process.

P. Any outstanding cancelable asynchronous I/O operations may be canceled. Those asynchronous I/O operations which are not canceled shall complete as if the `_exit()` operation had not yet occurred, but any associated signal notifications shall be suppressed. The `_exit()` operation itself may or may not block awaiting such I/O completion. Whether any I/O is canceled, and which I/O may be canceled upon `_exit()`, is *implementation-defined*.

S. Any cancelable asynchronous I/O operations are canceled.

3.3.1.1 Signal Names

`<signal.h>`
P. An *implementation may define* additional signals that may occur in the system.

S. The additional signals generated are:

SIGILL	SIGTRAP
SIGEMT	SIGBUS
SIGSYS	SIGPWR
SIGWINCH	SIGURG
SIGPOLL	SIGVTALRM
SIGPROF	SIGXCPU
SIGXFSZ	SIGIOT
SIGLOST	SIGIO
SIGFREEZE	SIGTHAW

P. It is *implementation-defined* whether the realtime signal behavior specified in this section-specifically, the queueing of signals and the passing of application defined values-is supported for the signals defined in Table 3-1, Table 3-2 or Table 3-3 [of the standard].

S. The passing of application-defined values is supported for all signals. The queueing of signals is supported for all signals generated via `sigqueue()` or requested via a `sigevent.sigev_notify` value of SIGEV_SIGNAL.

3.3.1.2 Signal Generation and Delivery

Signals
P. If a subsequent occurrence of a pending signal is generated, it is *implementation-defined* as to whether the signal is delivered more than once.

S. Subsequent occurrences of signals are delivered more than once if the subsequent signal was generated via `sigqueue()` or requested via a `sigevent.sigev_notify` value of SIGEV_SIGNAL.

P. An *implementation shall document* any conditions not specified by this standard under which the implementation generates signals.

S. Conditions under which these additional signals are generated are:

Signal	Condition
SIGTRAP	Trace/breakpoint Trap
SIGWINCH	Window size change
SIGEMT	Emulation trap
SIGURG	Urgent socket condition
SIGPOLL	Pollable event
SIGBUS	Bus error
SIGVTALRM	Virtual timer expired

SIGILL	Illegal instruction
SIGSYS	Bad system call
SIGPROF	Profiling timer expired
SIGXCPU	`CPU` time limit exceeded
SIGPWR	Power fail/restart
SIGXFSZ	File size limit exceeded
SIGIO	On asynchronous I/O
SIGLOST	When a lock is broken. (See `lockd(8)`)
SIGFREEZE	Checkpoint freeze
SIGTHAW	Checkpoint thaw

3.3.1.3 Signal Actions

P. The following values are defined for `si_code`:

SI_USER	The signal was sent by the `kill()` function. The implementation may set `si_code` to `SI_USER` if the signal was sent by the `raise()` or `abort()` functions defined in the C Standard {2} or any similar functions provided as implementation extensions.
SI_QUEUE	The signal was sent by the `sigqueue()` function.
SI_TIMER	The signal was generated by the expiration of a timer set by `timer_settime()`.
SI_ASYNCIO	The signal was generated by the completion of an asynchronous I/O request.
SI_MESGQ	The signal was generated by the arrival of a message on an empty message queue.

If the signal was not generated by one of the functions or events listed above, the `si_code` shall be set to an *implementation-defined* value that is not equal to any of the values defined above.

S. SunOS defines other values of `si_code` for particular signals. Symbols for these values are described in the `siginfo (5)` manual pages. Due to compilation namespace requirements, these symbols are not defined in the POSIX compilation environment, and hence are not available to Strictly Conforming Applications. Aliases for the values of of `si_code` that begin with "SI" are not currently available.

3.3.2.2 Send a Signal to a Process: Description

`kill()`
P. An implementation that provides extended security controls may impose further *implementation-defined* restrictions on the sending of signals, including the null signal.

S. Extended security controls that impose further restrictions on the sending of signals are not provided.

3.3.3.4 Manipulate Signal Sets: Errors

P. For each of the following conditions, if the condition is detected, the `sigaddset()`, `sigdelset()`, and `sigismember()` functions shall return -1 and set `errno` to the corresponding value:

[EINVAL]	The value of the `signo` argument is an invalid or unsupported signal number.

S. The `sigaddset()`, `sigdelset()`, and `sigismember()` functions all detect [EINVAL].

3.3.4.2 Examine and Change Signal Action: Description

P. If SA_SIGINFO is not set in `sa_flags`, then the disposition of subsequent occurrences of `sig` when it is already pending is *implementation-defined*; and the signal-catching function shall be invoked with a single argument.

S. If SA_SIGINFO is not set in `sa_flags`, subsequent occurrences of `sig`, when it is already pending and queued, are quietly discarded.

3.3.6.4 Examine Pending Signals: Errors

P. This part of ISO/IEC 9945 does not specify any error conditions that are required to be detected for the `sigpending()` function. Some errors may be detected under conditions that are unspecified by this part of ISO/IEC 9945.

S. The `sigpending()` function detects [EFAULT].

POSIX.1 Section 4, Process Environment

4.2.4.4 Get User Name: Errors

P. This part of ISO/IEC 9945 does not specify any error conditions that are required to be detected for the `getlogin()` function. Some errors may be detected under conditions that are unspecified by this part of ISO/IEC 9945.

S. The system detects no errors for `getlogin()`.

4.4.1.2 Get System Name: Description

`<sys/utsname.h>`
P. The structure `utsname` is defined in the header `<sys/utsname.h>`, and contains at least the members shown in Table 4-1 of the standard. (Refer to the standard for the table members.)

Each of these data items is a null-terminated character array. The format of each member is *implementation-defined*.

S. The format of the members found in `<sys/utsname.h>` for `utsname` is type *char* [257].

4.5.1.4 Get System Time: Errors

P. This part of ISO/IEC 9945 does not specify any error conditions that are required to be detected for the `time()` function. Some errors may be detected under conditions that are unspecified by this part of ISO/IEC 9945.

S. No additional errors are detected for the `time()` function.

4.6.1.4 Environment Variables: Errors

P. This part of ISO/IEC 9945 does not specify any error conditions that are required to be detected for the `getenv()` function. Some errors may be detected under conditions that are unspecified by this part of ISO/IEC 9945.

S. No error conditions are detected for the `getenv()` function.

4.7.1.4 Generate Terminal Pathname: Errors

P. This part of ISO/IEC 9945 does not specify any error conditions that are required to be detected for the `ctermid()` function. Some errors may be detected under conditions that are unspecified by this part of ISO/IEC 9945.

S. No error conditions are detected for the `ctermid()` function.

4.7.2.4 Determine Terminal Device Name: Errors

P. This part of ISO/IEC 9945 does not specify any error conditions that are required to be detected for the `ttyname()` or `isatty()` functions. Some errors may be detected under conditions that are unspecified by this part of ISO/IEC 9945.

S. No error conditions are detected for the `ttyname()` or `isatty()` functions.

POSIX.1 Section 5, Files and Directories

5.1.1 Format of Directory Entries

P. The internal format of directories is unspecified.

S. For each directory a link count is maintained. This is the total number of directories that are listed in the directory, including "." and "..".

5.1.2.4 Directory Operations: Errors

P. For each of the following conditions, when the condition is detected, the `opendir()` function shall return a value of `NULL` and set `errno` to the corresponding value:

[EMFILE]	Too many file descriptors are currently open for the process.
[ENFILE]	Too many file descriptors are currently open in the system.

S. The `opendir()` function detects the conditions and returns the corresponding `errno` values for both [EMFILE] and [ENFILE].

P. For each of the following conditions, when the condition is detected, the `readdir()` function shall return a value of `NULL` and set `errno` to the corresponding value:

[EBADF]	The `dirp` argument does not refer to an open directory system.

S. If the `dirp` argument passed to `readdir()` does not point to an open directory stream, Solaris returns a `NULL` pointer and set `errno` to [EBADF].

P. For each of the following conditions, when the condition is detected, the `closedir()` function shall return a value of -1 and set `errno` to the corresponding value:

[EBADF]	The `dirp` argument does not refer to an open directory stream.

S. For the `closedir()` function, Solaris may detect the condition and set `errno` value to [EBADF].

5.2.2.4 Get Working Directory Pathname: Errors

P. For each of the following conditions, if the condition is detected, the `getcwd()` function shall return a value of `NULL` and set `errno` to the corresponding value:

[EACCES]	Read or search permission was denied for a component of the pathname.

S. For the `getcwd()` function, Solaris detects the conditions and returns the corresponding `errno` value for [EACCES].

5.3.1.2 Open a File: Description

`O_CREAT`
P. The file's group ID shall be set to the group ID of the directory in which the file is being created or to the effective group ID of the process.

S. When `O_CREAT` is set in `oflag` and bits in `mode` other than the file permission bits are set, the files group ID is set to the group ID of the parent directory if the S_ISGID bit is set in the directory in which the file is being created. If the S_ISGID bit is set in the parent directory, the group ID of the file is inherited from the parent directory; otherwise it is set to the group ID of the calling process.

`O_TRUNC`
P. If the file exists and is a regular file, and the file is successfully opened `O_RDWR` or `O_WRONLY`, it shall be truncated to zero length and the mode and owner shall be unchanged by this function call. `O_TRUNC` shall have no effect on `FIFO` special files or directories. Its effect on other file types is *implementation-defined*. The result of using `O_TRUNC` with `O_RDONLY` is undefined.

S. `O_TRUNC` has no effect on other file types.

5.3.3.2 Set File Creation Mask: Description

`umask()`
P. The `umask()` routine sets the process's file mode creation mask to `cmask` and returns the previous value of the mask. Only the file permission bits of `cmask` are used; the meaning of the other bits is *implementation-defined*.

S. The implementation ignores all but the file permission bits.

5.3.4.2 Link to a File: Description

P. The *existing* argument shall not name a directory unless the user has appropriate privileges and the implementation supports using `link()` on directories.

S. Linking of directories is supported if the user has appropriate privileges.

P. The implementation may require that the calling process has permission to access the existing file.

S. Solaris does not require that the calling process have permission to access the existing file when linking files.

5.4.1.2 Make a Directory: Description

`mkdir()`
P. When bits in `mode` other than the file permission bits are set, the meaning of these additional bits is *implementation-defined*.

S. The implementation ignores all but the file permission bits.

P. The directory's group ID shall be set to the group ID of the directory in which the directory is being created or to the effective group ID of the process.

S. A new directory's group ID is set to the group ID of the parent directory when the S_ISGID bit is set in the parent directory; otherwise it is set to the group ID of the calling process. The newly created directory inherits the S_ISGID bit.

5.4.2.2 Make a FIFO Special File: Description

`mkfifo()`
P. When bits in `mode` other than the file permission bits are set, the meaning of these additional bits is *implementation-defined.*

S. The implementation ignores all but the file permission bits.

P. The group ID of the `FIFO` shall be set to the group ID of the directory in which the `FIFO` is being created or to the effective group ID of the process.

S. If the S_ISGID bit is set in the parent directory, the group ID of the `FIFO` is inherited from the parent directory; otherwise it is set to the group ID of the calling process.

5.5.1.2 Remove Directory Entries: Description

P. The `path` argument shall not name a directory unless the process has appropriate privileges and the implementation supports using `unlink()` on directories.

S. `unlink()` is supported if the user has the appropriate privileges.

5.5.1.4 Remove Directory Entries: Errors

P. If any of the following conditions occur, the `unlink()` function shall return -1 and set `errno` to the corresponding value:

[EBUSY]	The directory named by the `path` argument cannot be unlinked because it is being used by the system or another process and the implementation considers this to be an error.

S. `unlink()` supports detection of [EBUSY].

5.5.2.2 Remove a Directory: Description

`rmdir()`
P. If the named directory is the root directory or the current working directory of any process, it is unspecified whether the function succeeds or whether it fails and sets `errno` to [EBUSY].

S. If the directory indicated in the call to `rmdir()` is a mount point for a mounted file system, `rmdir()` sets errno to [EBUSY] and returns -1.

5.5.2.4 Remove a Directory: Errors

P. If any of the following conditions occur, the `rmdir()` function shall return -1 and set `errno` to the corresponding value:

[EBUSY] The directory named by the `path` argument cannot be removed because it is being used by another process and the implementation considers this to be an error.

S. If the directory indicated in the call to `rmdir()` is a mount point for a mounted file system, `rmdir()` sets errno to [EBUSY] and returns -1.

5.5.3.2 Rename a File: Description

P. Write access permission is required for the directory containing *old* and the directory containing *new*. If the *old* argument points to the pathname of a directory, write access permission may be required for the directory named by *old*, and, if it exists, the directory named by *new*.

S. In a call to `rename()`, if the *old* argument points to the pathname of a directory, write access permission is not required for the directory named by *old* and if it exists, for the directory named by *new*.

5.5.3.4 Rename a File: Errors

P. If any of the following conditions occur, the `rename()` function shall return -1 and set `errno` to the corresponding value:

[EBUSY] The directory named by *old* or *new* cannot be renamed because it is being used by the system or another process and the implementation considers this to be an error.

S. [EBUSY] is returned only if the *new* directory is a mount point for a mounted file system.

5.6.1.2 File Characteristics: File Modes

`<sys/stat.h>`
P. Implementations may `OR` other *implementation-defined* bits into S_IRWXU, S_IRWXG, and S_IRWXO, but they shall not overlap any of the other bits defined in this standard.

S. The implementation also provides a bit identified by S_ISVTX. For a directory, this bit determines whether or not an unprivileged user may delete or rename another user's files from that directory (refer to `chmod(2)` for other files types).

5.6.2.2 Get File Status: Description

`stat()`, `fstat()`
P. An implementation that provides additional or alternate file access control mechanisms may, under *implementation-defined* conditions, cause the `stat()` and `fstat()` functions to fail.

S. No other conditions cause these functions to fail.

5.6.3.4 Check File Accessibility: Errors

P. For each of the following conditions, if the condition is detected, the `access()` function shall return -1 and set `errno` to the corresponding value:

[EINVAL] An invalid value was specified for `amode`.

S. For the `access()` function, Solaris detects the condition and returns the corresponding `errno` value for [EINVAL].

5.6.4.2 Change File Modes: Description

P. Additional *implementation-defined* restrictions may cause the S_ISUID and S_ISGID bits in `mode` to be ignored.

S. If the process has access permissions, there are no implementation-defined conditions under which this would be denied.

P. The effect on file descriptors for files open at the time of the `chmod()` or `fchmod()` function is *implementation-defined*.

S. Access permissions for open file descriptors that refer to files on local (UFS) mounted file systems are not affected by `chmod()` or `fchmod()`. Access permissions for descriptors referring to files on other file systems may change as a result of a successful `chmod()` or `fchmod()` call.

5.6.5.2 Change Owner and Group of a File: Description

`chown()`
P. If the *path* argument refers to a regular file, the set-user-ID (S_ISUID) and set-group-ID (S_ISGID) bits of the file mode shall be cleared upon successful return from `chown()`, unless the call is made by a process with appropriate privileges, in which case it is *implementation-defined* whether those bits are altered.

S. The S_ISUID and S_ISGID bits of the file mode remain unaltered when a call is made by a process with the appropriate privilege.

5.6.5.4 Change Owner and Group of a File: Errors

P. For each of the following conditions, if the condition is detected, the `chown()` function shall return -1 and set `errno` to the corresponding value:

[EINVAL]	The owner of group ID supplied is invalid and not supported by the implementation.

S. The `chown()` function does not detect [EINVAL].

5.7.1.4 Get Configurable Pathname Variables: Errors

P. If any of the following conditions occur, the `pathconf()` function shall return -1 and set `errno` to the corresponding value:

[EINVAL]	The value of `name` is invalid.

S. For the `pathconf()` function, Solaris does detect the condition and returns the corresponding `errno` value for [EINVAL].

P. For each of the following conditions, if the condition is detected, the `pathconf()` function shall return -1 and set `errno` to the corresponding value:

[EACCES]	Search permission is denied for a component of the path prefix.
[ENAMETOOLONG]	The length of the `path` argument exceeds {`PATH_MAX`}, or a pathname component is longer than {`NAME_MAX`} while {`_POSIX_NO_TRUNC`} is in effect.
[ENOENT]	The named file does not exist, or the `path` argument points to an empty string.
[ENOTDIR]	A component of the path prefix is not a directory.

S. For the `pathconf()` function, Solaris detects the conditions and returns the corresponding `errno` value for [EACCES], [ENAMETOOLONG], [ENOENT] and [ENOTDIR].

P. For each of the following conditions, if the condition is detected, the `fpathconf()` function shall return -1 and set `errno` to the corresponding value:

[EBADF]	The `fildes` argument is not a valid file descriptor.
[EINVAL]	The implementation does not support an association of the variable name with the specified file.

S. For the `fpathconf()` function, Solaris detects the conditions and returns the corresponding `errno` value for [EBADF] and [EINVAL].

POSIX.1 Section 6, Input and Output Primitives

6.3.1.2 Close a File: Description

P. When there is an outstanding cancelable asynchronous I/O operation against fildes when `close()` is called, that I/O operation may be canceled. An I/O operation which is not canceled completes as if the `close()` operation had not yet occurred. All operations which are not canceled shall complete as if the `close()` blocked until the operations completed. The `close()` operation itself

need not block awaiting such I/O completion. Whether any I/O operation is canceled, and which I/O operation may be canceled upon `close()`, is *implementation-defined*.

S.The Asynchronous Input and Output option is not supported in Solaris 2.4; hence, there is no implementation-specific behavior.

6.4.1.2 Read from a File: Description

`read()`
P. If a `read()` is interrupted by a signal after it has successfully read some data, either it shall return -1 with `errno` set to [EINTR], or it shall return the number of bytes read.

S. If a `read()` is interrupted by a signal after is has successfully read some data, it returns the number of bytes read.

P. If the file refers to a device special file, the result of subsequent `read()` requests after a `read()` has returned an EOF indication, is *implementation-defined*.

S. The result of this request is device dependent for standard `tty` devices**.** See Section 4 of the *SunOS 5.4 Reference Manual* for more information.

P. If the value of `nbyte` is greater than `{SSIZE_MAX}`, the result is *implementation-defined*.

S. Given a valid buffer, `nbyte` bytes will be transferred.

6.4.2.2 Write to a File: Description

`write()`
P. If `write()` is interrupted by a signal after it successfully writes some data, either it shall return -1 with `errno` set to [EINTR], or it shall return the number of bytes written.

S. If `write()` is interrupted by a signal after it successfully writes some data, it returns the number of bytes read.

P. If the value of `nbyte` is greater than `{SSIZE_MAX}`, the result is *implementation-defined*.

S. The write will not succeed; it returns –1 and sets `errno` to [EINVAL].

6.5.2.2 File Control: Description

P. If the system detects that sleeping until a locked region is unlocked would cause a deadlock, the `fcntl()` function shall fail with an [EDEADLK] error.

S. The `fcntl()` function detects [EDEADLK].

6.5.3.2 Reposition Read/Write File Offset: Description

`lseek()`
P. Some devices are incapable of seeking. The behavior of the `lseek()` function on such devices is *implementation-defined*.

S. On such devices, `lseek()` returns -1 with `errno` set to `[EINVAL]`.

6.6 File Synchronization

P. The hardware characteristics upon which the implementation relies to assure that data successfully transferred for synchronized I/O operations are *implementation-defined*.

S. The data is considered to be successfully transferred when the device driver operation completes successfully.

6.6.1.2 Synchronize a File's State: Description

P. The `fsync()` function can be used by the application to indicate that all data for the open file description named by *fildes* is to be transferred to the storage device associated with the file described by *fildes*, in an *implementation-defined* manner.

The *conformance document shall include* sufficient information for the user to determine whether it is possible to configure an application and installation to ensure that the data is stored with the degree of required stability for the intended use.

S. For files in a `ufs` filesystem, the physical transfer to the underlying device must successfully complete.

6.7.1.1 Data Definitions for Asynchronous Input and Output: Asynchronous I/O Control Block

P. Under *implementation-defined* circumstances, such as operation on a multiprocessor or when requests of differing priorities are submitted at the same time, the ordering restriction may be relaxed; *the implementation shall document* under what circumstances the ordering restriction may be relaxed.

S. The Asynchronous Input and Output option is not supplied in Solaris; hence, there is no implementation-specific behavior.

P. The relative priority of asynchronous I/O and synchronous I/O is *implementation-defined*. If `POSIX_PRIORITIZED` is defined, *the implementation shall define* for which files I/O prioritization is supported.

S. The Asynchronous Input and Output option is not supplied in Solaris; hence, there is no implementation-specific behavior.

6.7.7.2 Cancel Asynchronous I/O Request: Description

`aio_cancel()`
P. It is *implementation-defined* which operations are cancelable.

S. The Asynchronous Input and Output option is not supplied in Solaris; hence, there is no implementation-specific behavior.

POSIX.1 Section 7, Device- and Class-Specific Functions

7.1 General Terminal Interface

`terminal interface`
P. It is *implementation-defined* whether this interface supports network connections or synchronous ports or both. The conformance document shall describe which device types are supported by these interfaces.

S. SunSoft supports these interfaces for terminal devices, terminal multiplexers, and terminal pseudo-devices.

7.1.1.3 The Controlling Terminal

controlling terminal
P. The controlling terminal for a session is allocated by the session leader in an *implementation-defined* manner. If a session leader has no controlling terminal, and opens a terminal device file that is not already associated with a session without using the `O_NOCTTY` option, it is i*mplementation-defined* whether the terminal becomes the controlling terminal of the session leader.

S. If a session leader has no controlling terminal and opens a terminal device file that is not already associated with a session without using the `O_NOCTTY` option, the terminal then becomes the controlling terminal of the session leader.

7.1.1.5 Input Processing and Reading Data

input queue
P. The system may impose a limit, `{MAX_INPUT}`, on the number of bytes that may be stored in the input queue. The behavior of the system when this limit is exceeded is *implementation-defined*.

S. If the data in the driver's input queue exceeds `{MAX_INPUT}`, all the characters saved in the stream up to that point are discarded without notice. However, if `IMAXBEL` is set and the data in the driver input queue exceeds `{MAX_INPUT}`, the ASCII BEL character is echoed. Further input will not be stored, and any input already present in the input stream is not disturbed.

7.1.1.6 Canonical Mode Input Processing

`{MAX_CANON}`
P. If `{MAX_CANON}` is defined for this terminal device, it is a limit on the number of bytes in a line. The behavior of the system when this limit is exceeded is *implementation-defined*.

S. If the data in the line discipline buffer exceeds `{MAX_CANON}` in the canonical mode and `IMAXBEL` is not set, all the characters saved in the buffer up to that point are discarded without any notice. However, if `IMAXBEL` is set and the data in the line discipline buffer exceeds `{MAX_CANON}`, the ASCII BEL character is echoed. Further input will not be stored, and any input already present in the input stream is not disturbed.

7.1.1.7 Noncanonical Mode Input Processing

`MIN`
P. If `MIN` is greater than `{MAX_INPUT}`, the response to the request is *implementation-defined*.

S. The maximum value that can be stored for `MIN` in `c_cc [VMIN]` is 255, which is less than `{MAX_INPUT}` (512). The `MIN` value can never exceed `{MAX_INPUT}`.

7.1.1.8 Writing Data and Output Processing

P. The *implementation may provide* a buffering mechanism; as such, when a call to `write()` completes, all of the bytes written have been scheduled for transmission to the device, but the transmission will not necessarily have completed.

S. Solaris provides a buffering mechanism for `a write()` to a teminal device. The `write()` system call may complete and return a value to the user program, but the data sent downstream may flow control on one or more streams modules. To ensure that the data has been transmitted entirely, make a call to `tcdrain()`. On return from `tcdrain()`, the written data will be transmitted.

7.1.1.9 Special Characters

`START`, `STOP`
P. It is *implementation-defined* whether the `START` and `STOP` characters can be changed.

S. The `START` and `STOP` characters can be changed.

IEXTEN
P. A special character is recognized not only by its value, but also by its context; for example, an implementation may define multi-byte sequences that have a meaning different from the meaning of bytes when considered individually. Implementations may also define additional single-byte functions. These *implementation-defined* multibyte or single byte functions are recognized only if the IEXTEN flag is set; otherwise, data is received without interpretation, except as required to recognize the special characters defined in the subclass (7.1.1.9).

S. SunOS 5.4 does not recognize any multibyte input control sequences. The following single-byte special characters are recognized when IEXTEN is set:

WERASE	Erase last word typed
REPRINT	Reprint the current input
DISCARD	Discard output
LNEXT	Ignore any special meaning of the next character typed

7.1.2.2 Input Modes

c_iflag
P. In contexts other than asynchronous serial data transmission the definition of a break condition is *implementation-defined*.

S. The break condition is not defined for contexts other than asynchronous serial data.

P. The precise conditions under which STOP and START characters are transmitted are *implementation-defined*.

S. A STOP character is transmitted when the input data exceeds the high water mark of the queue. A START character is transmitted when the input data falls below the low water mark of the queue. If it is not a STREAMS device, the results are device-dependent.

P. The initial input control value after open() is *implementation-defined*.

S. The initial setting of the input mode flag is configurable. For more information, see the termio(7) man page.

7.1.2.3 Output Modes

P. If `OPOST` is set, output data is processed in an *implementation-defined* fashion so that lines of text are modified to appear appropriately on the terminal device, otherwise characters are transmitted without change.

S. The SunOS 5.4 operating system supports the following output control mode masks which are enabled by `OPOST`:

OLCUC	Map lower case to upper case on output.
ONLCR	Map NL to CR-NL on output.
OCRNL	Map CR to NL on output.
ONOCR	No CR output at column 0.
ONLRET	NL performs CR function.
OFILL	Use fill characters for delay.
OFDEL	Fill is DEL, else NUL.
NLDLY	Select new line delays:
NL0	No new line delay.
NL1	
CRDLY	Select carriage-return delays:
CR0	No carriage return delay.
CR1	
CR2	
CR3	
TABDLY	Select horizontal-tab delays or expansion:
TAB0	No horizontal tab delay.
TAB1	
TAB2	
TAB3	
XTABS	Expand tabs to spaces.
BSDLY	Select backspace delays:
BS0	No backspace delay.
BS1	
VTDLY	Select vertical-tab delays:
VT0	No vertical tab delay.
VT1	
FFDLY	Select form-feed delays:
FF0	No form feed delay.
FF1	

`open()`
P. The initial output control value after `open()` is *implementation-defined*.

S. The initial setting for the output control flag `oflag` is configurable. For more information, see the `termio(7) man` page.

7.1.2.4 Control Modes

`open()`
P. The initial hardware control value after `open()` is *implementation-defined*.

S. The initial hardware control value after `open()` is configurable. For more information, see the `termio(7) man` page.

7.1.2.5 Local Modes

`IEXTEN`
P. If `IEXTEN` is set, *implementation-defined* functions shall be recognized from the input data.

S. If `IEXTEN` and `ICANON` are set, the `WERASE`, `REPRINT`, `DISCARD`, and `LNEXT` functions are recognized from the input data.

P. It is *implementation-defined* how `IEXTEN` being set interacts with `ICANON`, `ISIG`, `IXON`, or `IXOFF`. If `IEXTEN` is not set, then *implementation-defined* functions shall not be recognized, and the corresponding input characters shall be processed as described for `ICANON`, `ISIG`, `IXON`, and `IXOFF`.

S. `IXON`, `ISIG`, and `IXOFF` flags are processed as they are defined in the standard, when `IEXTEN` is on or off.

In addition to the local mode masks listed in the standard, the SunOS 5.4 operating system supports the following functions:

`XCASE`	Canonical upper/lower presentation
`ECHOCTL`	Echo control characters as '`^C`', delete character as '`^?`'.
`ECHOPRT`	Echo erase character as character erased.
`ECHOKE`	BSSP_BS erase entire line on line kill.
`FLUSHO`	Output is being flushed.
`PENDIN`	Retype pending input at next read or input character.

P. The initial control value after `open()` is *implementation-defined*.

S. The initial setting for the local mode flag `lflag` is configurable. For more information, see the `termio(7) man` page.

7.1.2.6 Special Control Characters

P. The initial values of all control characters are *implementation-defined*.

S. The initial values of control characters are configurable and are set when the system boots. See the `termio(7) man` page for more information.

7.1.3.4 Baud Rate Functions: Errors

P. This part of ISO/IEC 9945 does not specify any error conditions that are required to be detected for the `cfgetispeed()`, `cfgetospeed()` `cfsetispeed()` or `cfsetospeed()` functions. Some errors may be detected under conditions that are unspecified by this part of ISO/IEC 9945.

S. The `cfgetispeed()`, `cfgetospeed()` `cfsetispeed()` and `cfsetospeed()` functions do not return any additional errors.

7.2.1.2 Get and Set State: Description

P. If the input and output baud rates differ and are a combination that is not supported, neither baud rate is changed.

S. Differing input and output baud rates are not supported.

7.2.2.2 Line Control Functions: Description

`tcsendbreak()`, `tcdrain()`, `tcflush()`, `tcflow()`
P. If the terminal is not using asynchronous serial data transmission, it is *implementation-defined* whether the `tcsendbreak()` function sends data to generate a break condition (as *defined by the implementation*) or returns without taking any action.

S. On non-asynchronous transmissions, `tcsendbreak()` does not send a break; it simply returns.

`tcsendbreak()`
P. If `duration` is not zero, it shall send zero-valued bits for an *implementation-defined* period of time.

S. For a delay of $n \neq 0$, `tcsendbreak()` is equivalent to `tcdrain()`.

POSIX.1 Section 8, Language-Specific Services for the C Programming Language

8.1.1 Referenced C Language Routines, Extensions to Time Functions

`TZ`
P. If `TZ` is of the first format (i.e., if the first character is a colon), the characters following the colon are handled in an *implementation-defined* manner.

S. The string following the colon refers to the file `/usr/share/lib/zoneinfo/`<*string*>. This file contains a timezone specification.

8.1.2.2 Extensions to `setlocale()`*: Description*

`setlocale()`
P. In addition to the value for "category" specified in the standard, the i*mplementation may define* additional categories.

S. In addition to the categories (environment variables) described in the standard, the SunOS 5.4 operating system supports the following:

`LC_MESSAGES`	Allows for display of alternate message texts

P. If no nonnull environment variable (`$LC_ALL`, `$LANG`, or the environment variable corresponding to the category being set) is present to supply a value for "locale" it is *implementation-defined* whether `setlocale()` sets the specified locale category to a systemwide default value or to "C or to POSIX".

S. The default locale is "C".

P. The possible actual values of the environment variables are *implementation-defined* and should appear in the system documentation.

S. The supported locales are “C”,“POSIX”, “de”,“fr”,“it”, “ja”, “japanese” and “sv”. The locale name “iso_8859_1” contains only the `LC_CTYPE` category, and is not an appropriate value for `LC_ALL` or the `LANG` environment variable. Other locale names may be available due to the addition of further Sun or third-party packages.

8.2.2.4 Open a Stream on a File Descriptor: Errors

P. This part of ISO/IEC does not specify any error conditions that are required to be detected for the `fdopen()` function. Some errors may be detected under conditions that are unspecified by this part of ISO/IEC 9945.

S. The `fdopen()` function detects no errors.

8.2.3 Interactions of Other File-Type C Functions

P. (5) Implementations shall assure that an application, even one consisting of several processes, shall yield correct results (no data is lost or duplicated when writing, all data is written in order, except as requested by seeks) when the rules above are followed, regardless of the sequence of handles used. When these rules are followed, it is *implementation-defined* whether, and under what conditions, all input is seen exactly once.

S. When applications follow the rules specified, all input is seen exactly once.

8.3.2.2 Set Time Zones: Description

`tzset()`
P. If `TZ` is absent from the environment, *implementation-defined* default time zone information shall be used.

S. If `TZ` is absent from the environment, then time zone information behaves as though `TZ` were set to `localtime`.

6

POSIX.1 Section 9, System Databases

9.1 System Databases

`/etc/passwd`
P. If the initial user program field is null, the system default is used.

S. If the user program field is null, the default program is `/usr/bin/sh`.

P. If the initial working directory field is null, the interpretation of that field is *implementation-defined*.

S. If the field is empty, the login fails.

9.2.1.4 Group Database Access: Errors

P. This part of ISO/IEC 9945 does not specify any error conditions that are required to be detected for the `getgrgid()` or `getgrnam()` functions. Some errors may be detected under conditions that are unspecified by this part of ISO/IEC 9945.

S. No error conditions are detected for the `getgrgid()` or `getgrnam()` functions.

9.2.2.4 User Database Access: Errors

P. This part of ISO/IEC 9945 does not specify any error conditions that are required to be detected for the `getpwuid()` or `getpwnam()` functions. Some errors may be detected under conditions that are unspecified by this part of ISO/IEC 9945.

S. No errors conditions are detected for the `getpwuid()` or `getpwnam()` functions.

POSIX.1 Section 10, Data Interchange Format

10.1 Archive/Interchange File Format

P. The *format-creating utility* is used to translate from the file system to the formats defined in this clause. The format-reading utility is used to translate from the formats defined in this clause to a file system. The interface to these utilities, including their name or names, is *implementation-defined*.

S. The `cpio` utility, when used with certain options, can be used to create and read these formats. For more information, see the `cpio man` page.

10.1.1 Extended `tar` Format

P. If an implementation supports the use of characters outside the portable filename character set in names for files, users, and groups, one or more *implementation-defined* encodings of these characters shall be provided for interchange purposes.

S. The use of all 8-bit characters is supported (except `NULL` and '`/`') in names for files, and all 8-bit characters (except `NULL` and colon) in names for users and/or groups. Characters are used in filenames exactly as they are read from the archive.

P. If a file name is found on the medium that would create an invalid file name, the implementation shall define if the data from the file is stored on the file hierarchy and under what name it is stored.

S. Any name that can be stored in a `tar` archive is interpreted as a valid file name, with the following exception: names with an embedded `NULL` will be truncated at the `NULL`.

10.1.2.1 Header

P. `c_rdev` shall contain *implementation-defined* information for character or block special files.

S. The `c_rdev` field contains `devmajor/devminor` device numbers. It is a 6-digit octal number and calculated as the major device number left-shifted by 8 `ORed` with the minor device number.

10.1.2.2 File Name

P. If a file name is found on the medium that would create an invalid pathname, the *implementation shall define* if the data from the file is stored on the file hierarchy and under what name it is stored.

S. All names will be legal in the file hierarchy.

P. If an implementation supports the use of characters outside the portable filename character set in names for files, users, and groups, one or more *implementation-defined* encodings of these characters shall be provided for interchange purposes.

S. The use of all 8-bit characters is supported (except `NULL` and '`/`') in names for files, and all 8-bit characters (except `NULL` and colon) in names for users and/or groups. Characters are used in file names exactly as they are read from the archive.

10.1.3 Multiple Volumes

archive/interchange file format
P. The format-reading utility shall, in an *implementation-defined* manner, determine what file to read as the next file.

S. The format reading utility opens `/dev/tty` and prompts the user for the next volume when an EOF is encountered.

POSIX.1 Section 11, Synchronization

11.2.3.2 Initialize/Open a Named Semaphore: Description

P. If *name* does not begin with the slash character, the effect is *implementation-defined*. The interpretation of slash characters other than the leading slash character in `name` is *implementation-defined*.

S. The Semaphore option is not supplied in Solaris 2.4; hence, there is no implementation-specific behavior.

POSIX.1 Section 12, Memory Management

P. Memory locking guarantees the residence of portions of the address space. It is *implementation-defined* whether locking memory guarantees fixed translation between virtual addresses (as seen by the process) and physical addresses.

S. Memory-locking does not lock in translation.

12.1.1.2 Lock/Unlock a Process's Address Space: Description

P. If `MCL_FUTURE` is specified, and the automatic locking of future mappings eventually causes the amount of locked memory to exceed the amount of available physical memory or any other *implementation-defined* limit, the behavior is *implementation-defined*. The manner in which the implementation informs the application of these situations is *implementation-defined*.

S. The `mmap()` function will return an error code of `[EAGAIN]` when resources do not permit the memory to be locked. If the mapping is an attempt to grow the stack, a SIGSEGV signal is sent to the process.

12.1.1.4 Lock/Unlock a Process's Address Space: Errors

P. For each of the following conditions, if the condition is detected, the `mlockall()` function shall return -1 and set `errno` to the corresponding value:

[ENOMEM]	Locking all of the pages currently mapped into the process's address space would exceed an *implementation-defined* limit on the amount of memory that the process may lock.

S. There is no per-process limit on how much memory may be locked.

12.1.2.4 Lock/Unlock a Range of Process Address Space: Errors

P. For each of the following conditions, if the condition is detected, the `mlock()` function shall return -1 and set `errno` to the corresponding value:

[ENOMEM]	Locking the pages currently mapped by the specified range would exceed an *implementation-defined* limit on the amount of memory that the process may lock.

S. There is no per-process limit on how much memory may be locked.

12.2.1.2 Map Process Addresses to a Memory Object: Description

P. `MAP_FIXED` informs the system that the value of `pa` shall be `addr` exactly. It is *implementation-defined* whether `MAP_FIXED` is supported.

S. Solaris 2.4 supports `MAP_FIXED`; however, its use is discouraged.

P. When `MAP_FIXED` is not set, the system uses `addr` in an *implementation-defined* manner to arrive at `pa`.

S. The value of `addr` is ignored. An unmapped part of the address is allocated.

12.3.1.2 Open a Shared Memory Object: Description

P. If `name` does not begin with the slash character, the effect is *implementation-defined*. The interpretation of slash characters other than the leading slash character in `name` is *implementation-defined*.

S. The Shared Memory Objects option is not supported in Solaris 2.4; hence, there is no implementation-specific behavior.

12.4.1.1.1 Process Memory Locking: Models

P. The page size is *implementation-defined* and is available to applications as a compile time symbolic constant or at run-time via `sysconf()`.

S. The page size is dependent on the underlying hardware.

POSIX.1 Section 13, Execution Scheduling

13.2 Scheduling Policies

P. Three scheduling policies are specifically required; others may be *implementation-defined*.

S. No other policies are defined by the implementation.

13.2.3 Scheduling Policies: SCHED_OTHER

P. Conforming implementations shall include one scheduling policy identified as `SCHED_OTHER` (which may execute identically with either the FIFO or round robin scheduling policy). *Conforming implementations shall document* the behavior of this policy as described in the definition of scheduling policy. The effect of scheduling processes with the `SCHED_OTHER` policy in a system in which other processes are executing under `SCHED_FIFO` or `SCHED_RR` shall thus be *implementation-defined*.

S. The Priority Scheduling Option is not supported in Solaris 2.4, hence, there is no implementation-specific behavior.

13.3.1.2 Set Scheduling Parameters: Description

P. The conditions under which one process has permission to change another process's scheduling parameters are *implementation-defined*.

S. If the target process has a `SCHED_FIFO` or `SCHED_RR` policy, the calling process must have a `SCHED_FIFO` or `SCHED_RR` policy or must have a UID of zero. If the target process has a `SCHED_OTHER` policy, the calling process must have either the same effective and real user ID as the target process's real or saved user ID or must have a UID of zero.

P. If the current scheduling policy for the process specified by `pid` is not `SCHED_FIFO` or `SCHED_RR`, including `SCHED_OTHER`, the result is *implementation-defined*.

S. If the target process has the `SCHED_OTHER` policy, the time-sharing scheduling parameters are set for the target process from the `sched_param` structure.

13.3.3.2 Set Scheduling Policy and Scheduling Parameters: Description

P. The conditions under which one process has the appropriate privilege to change another process's scheduling parameters are *implementation-defined*.

S. If the target process has a `SCHED_FIFO` or `SCHED_RR` policy, the calling process must have a `SCHED_FIFO` or `SCHED_RR` policy or must have a UID of zero. If the target process has a `SCHED_OTHER` policy, the calling process must have either the same effective and real user ID as the target process's real or saved user ID or must have a UID of zero.

P. Implementations may require that the requesting process have permission to set its own scheduling parameters or those of another process. Additionally, *implementation-defined* restrictions may apply as to the appropriate privileges required to set a process's own scheduling policy, or another process's scheduling policy, to a particular value.

S. If the target process has the `SCHED_OTHER` policy, the time-sharing scheduling parameters are set for the target process from the `sched_param` structure.

POSIX.1 Section 14, Clocks and Timers

14.2.1.2 Clock and Timer Functions: Description

P. The resolution of any clock can be obtained by calling `clock_getres()`. Clock resolutions are *implementation-defined* and are not settable by a process.

S. The clock resolution depends on the underlying hardware.

P. The effect of setting a clock via `clock_settime()` on armed pre-process timers associated with that clock is *implementation-defined*.

S. The timer expires at the same moment it would have expired had the clock not been changed.

P. The appropriate privilege to set a particular clock is *implementation-defined*.

S. Only a process with UID zero can set the clock `CLOCK_REALTIME`.

14.2.2.2 Create a Per-Process Timer: Description

P. The behavior for any other value of `sigev_notify` is *implementation-defined*.

S. No other value of `sigev_notify` is supported.

P. If `clock_id` specifies the `CLOCK_REALTIME` system clock, then the default signal, when `evp` is `NULL` shall be `SIGALRM`. For any other clock, the default signal number is *implementation-defined*.

S. No other clocks are supported.

14.2.4.2 Per-Process Timers: Description

P. The overrun count returned shall contain the number of extra timer expirations which occurred between the time the signal was generated (queued) and when it was delivered, up to but not including an *implementation-defined* maximum of {`DELAYTIMER_MAX`}.

S. The maximum overrun count is `INT_MAX`.

POSIX.1 Section 15, Message Passing

15.1.1. Data Definitions for Message Queues: Data Structures

P. The header <`mqueue.h`> shall define the following *implementation-defined* types:

`mqd_t` Used for message queue descriptors

S. The type `mqd_t` is declared:

```
typedef    void    *mqd_t;
```

P. The header <`mqueue.h`> defines the following *implementation-defined* structures:

`struct sigevent` As specified in 3.3.1

S.
```
struct sigevent {
        int             sigev_notify;   /*notification mode */
        int             sigev_signo;    /*signal number */
        union sigval    sigev_value;    /* signal value */
};
```

15.2.1.2 Open a Message Queue: Description

P. The interpretation of slash characters other than the leading slash character in `name` is *implementation-defined*.

S. The Message Passing Option is not supported in Solaris 2.4; hence, there is no implementation-specific behavior.

`O_CREAT`
P. The "file permission bits" shall be set to the value of mode. When bits in mode other than file permission bits are set, the effect is *implementation-defined*.

S. The Message Passing Option is not supported in Solaris 2.4; hence, there is no implementation-specific behavior.

P. If `attr` is `NULL`, the message queue is created with *implementation-defined* default message queue attributes.

S. The Message Passing Option is not supported in Solaris 2.4; hence, there is no implementation-specific behavior