

RadiSys.

Ultra C Library Reference

Version 2.4.1

www.radisys.com

World Headquarters
5445 NE Dawson Creek Drive • Hillsboro, OR
97124 USA

Phone: 503-615-1100 • Fax: 503-615-1121
Toll-Free: 800-950-0044

International Headquarters
Gebouw Flevopoort • Televisieweg 1A
NL-1322 AC Almere, The Netherlands
Phone: 31 36 5365595 • Fax: 31 36 5365620

RadiSys Microware Communications Software Division, Inc.
1500 N.W. 118th Street
Des Moines, Iowa 50325
515-223-8000
Revision H
November 2001

Copyright and publication information

This manual reflects version 2.4.1 of Ultra C. Reproduction of this document, in part or whole, by any means, electrical, mechanical, magnetic, optical, chemical, manual, or otherwise is prohibited, without written permission from RadiSys Microware Communications Software Division, Inc.

Disclaimer

The information contained herein is believed to be accurate as of the date of publication. However, RadiSys Corporation will not be liable for any damages including indirect or consequential, from use of the OS-9 operating system, Microware-provided software, or reliance on the accuracy of this documentation. The information contained herein is subject to change without notice.

Reproduction notice

The software described in this document is intended to be used on a single computer system. RadiSys Corporation expressly prohibits any reproduction of the software on tape, disk, or any other medium except for backup purposes. Distribution of this software, in part or whole, to any other party or on any other system may constitute copyright infringements and misappropriation of trade secrets and confidential processes which are the property of RadiSys Corporation and/or other parties. Unauthorized distribution of software may cause damages far in excess of the value of the copies involved.

November 2001

Copyright ©2001 by RadiSys Corporation.
All rights reserved.

EPC, INtime, iRMX, MultiPro, RadiSys, The Inside Advantage, and ValuPro are registered trademarks of RadiSys Corporation. ASM, Brahma, DAI, DAQ, MultiPro, SAIB, Spirit, and ValuePro are trademarks of RadiSys Corporation.

DAVID, MAUI, OS-9, and OS-9000, are registered trademarks of RadiSys Microware Communications Software Division, Inc. FasTrak, Hawk, SoftStax, and UpLink are trademarks of RadiSys Microware Communications Software Division, Inc.

† All other trademarks, registered trademarks, service marks, and trade names are the property of their respective owners.

Table of Contents

Chapter 1: Overview

7

8	Using Functions and Macros
10	Library Files
12	Program Globals
12	Class 1 Globals
13	Class 2 Globals
15	Header Files
39	The C Library
39	Communicating with Environment Functions
40	Character Classification Functions
42	Character Conversion Functions
43	File Positioning Functions
43	GetStat/SetStat Functions
49	Input/Output Device Functions
50	Input/Output Functions
57	Interprocess Communication Functions
64	Interrupt Manipulation Functions
65	Mathematical Functions
67	Memory Management Functions
70	Shared Data Access Functions
71	Miscellaneous Functions
74	Module Manipulation Functions
78	Multibyte Character and String Functions
79	OS System Functions
80	POSIX Messaging Functions
82	Process Manipulation Functions
85	Resource Locks
86	Searching and Sorting Functions

87	String Handling Functions
89	Terminal Manipulation Functions (Termcap)
90	Time Functions
95	Processor-Specific Functions
96	Implementation-Defined Behavior
98	ANSI “Core” Signals
98	OS-9 “Core” Signals
98	OS-9 68K Family Signals
98	OS-9 80x86 Family Signals
99	OS-9 PowerPC™ Family Signals
101	Processor-Specific Family Signals
101	68K Family Processors
102	80x86 Family Processors
102	PowerPC Family Processors
103	ARM Family Processors
103	SH Family Processors
104	SPARC Family Processors
108	Equivalents for the sys_clib.l Functions
116	The Function Description
116	Syntax
117	OS-9 Attributes
117	Library
117	Errors
117	References to Other Calls
118	C Shared Library

Chapter 2: Functions

129

1190	Default Handling
1190	Ignore Handling
1190	Available Signals
1191	ANSI “Core” Signals
1192	OS-9 for 68K/OS-9 “Core” Signals
1193	680x0: OS-9 for 68K, OS-9

1193	680x0: OS-9
1194	80x86: OS-9
1195	PowerPC: OS-9
1195	Function Handling
1195	Asynchronous Signals

Appendix A: Prototyping sys_clib.l Functions

1343

1344	Function Declarations and K&R Compiler vs. Prototypes and ANSI C
1346	"ANSI"fyng Code and Function Prototypes

Index

1347

Product Discrepancy Report

1449

Chapter 1: Overview

Ultra C library calls perform a variety of functions, from classifying characters to managing memory. This section contains information helpful for using the Ultra C library. This information includes:

- [Using Functions and Macros](#)
- [Library Files](#)
- [Program Globals](#)
- [Header Files](#)
- [The C Library](#)
- [C Shared Library](#)

Using Functions and Macros

Ultra C supports both functions and macros:

- A **function** consists of compiled C statements.
- A **macro** is an identifier defined to represent a value or expression.

In most cases, the differences between the two are transparent.



WARNING

Textual replacement in macro expansion can cause arguments of macros to be evaluated repeatedly. Macro implementations of ANSI Standard Library functions are required to evaluate arguments exactly once and to have sufficient parentheses to avoid operator precedence trouble.

You can bypass the macro replacement when you compile. This may be desirable when a large macro is used many times and the space saved by calling the function is outweighed by the extra time needed to call the function. For example, to override the replacement/instantiation of the `putc` macro use:

```
(putc)(c, fp);
```

or

```
#undef putc
```

This results in a call to the `putc` function in the library rather than the replacement of the `putc` macro.

All American National Standards Institute (ANSI) macros have a library counterpart of the same name, with the exception of the following:

- `assert()`
- `setjmp()`
- `va_arg()`
- `va_end()`
- `va_start()`

This is not true for non-ANSI macros.

Library Files

The following are important files used during the compilation process. These files are located in the MWOS structure under LIB for the hardware platform.

Libraries provided with Ultra C are listed in **Table 1-1**.

Table 1-1 Libraries Provided with this Release

Library	Contains
clib.l	The ANSI standard library
cstart.r	Start-up code for compiled programs (with the -bc option K & R compatible source mode)
acstart.r	By default (same as cstart.r but compiled with ANSI defined)
acstarta.r	icode linked version of the above
ansi_cstart.r	Same as acstart.r, mainly for the 8.3 filenames on DOS
ansi_cstarta.r	Same as acstarta.r
cstarta.r	With -bc -j option (icode linked version, backward compatibility - K&R mode)
sacstart.r	ANSI version of sysstart.r
scstart.r	Used to make system state processes
scstarta.r	Version of scstart.r used when -j option used
sysansi_cstart.r	Same as sacstart.r
sysansi_cstarta.r	Same as sacstrta.r
syscstart.r	Same as scstart.r (8.3 format)

Table 1-1 Libraries Provided with this Release

Library	Contains
syscstarta.r	same as scstarta.r (8.3 format)
cpu.l	CPU dependent functions
mq.l	POSIX messaging functions
os_lib.l	The system call functions
sys_clib.l	The non-ANSI C library functions
termlib.l	The termcap database screen manipulation functions
sclib.l	To minimize code size, smaller versions of some library functions are available in the libraries sclib.l and sclib.il.
unix.l	The UNIX-like library functions



For More Information

The primary purpose of the `sys_clib.l` library is to provide a library for compatibility with the Microware K&R C compiler. For additional information, refer to [Appendix A: Prototyping sys_clib.l Functions](#).



For More Information

For information on `sclib.l`, small library functions, see Chapter 2 of [Using Ultra C/C++](#).

Program Globals

Ultra C defines a number of globals for use by library functions and/or the application. They are divided into two classes:

1. Globals that are unlikely to change from release to release, and
2. Globals that are used by library functions that may change from release to release (see header file `cglob.h` for current information).

Class 1 Globals

<code>u_int32 _totmem</code>	Total number of bytes that are in the data area for the program (includes stack, global data, and parameters)
<code>u_int16 _pathcnt</code>	Number of open paths the process inherited from the parent process (usually 3)
<code>void *_sysglob</code>	Pointer to the kernel's system globals. This variable is only initialized when the program is a system state process (otherwise it is NULL). OS-9 for 68K only.
<code>mh_exec *_modhead</code>	Pointer to the primary module for the current process
<code>char *_modname</code>	Pointer to the module name for the current process (unmodifiable).
<code>u_int32 _procid</code>	Process ID of the current process
<code>u_int32 _maxstack</code>	Maximum number of bytes consumed from the stack so far. This variable does not contain a useful value unless stack checking is enabled. Print this variable prior to exiting to examine how much stack space your application required.

```
void *_glob_data
char **_environ
char **environ
```

Pointer to the current processes global static storage. `_glob_data` can be used as an argument to `_os_intercept()`.

Pointer to the environment variable strings for the current process. This variable is used when forking other processes to pass the current environment along.

Same as `_environ`, except that it is available only in K & R source mode for OS-9 for 68K. `environ` is provided for compatibility with Microware K & R C-compiler.

Class 2 Globals

```
void *_sttop
void *_mtop

void *_stbot
void *_csl_a6

void *_csl_glob

u_int32 _sbsize

FILE (*_fcbs)[ ]
void *_ofcbs
int32 _stklimit
```

Address of the top of the stack

Address of data memory. `_mtop` is used to check for stack overflow.

Deepest stack pointer so far

Global static storage pointer for attached csl. **OS-9 for 68K only.**

Global static storage pointer for attached csl. **OS-9 only.**

Size of process memory area. `_sbsize` is used by `sbrk()`.

Pointer to `_niob` array

Pointer to compatibility `_iob` array

Amount of verified area left on stack. This value contains the number of bytes that can be consumed from the stack before further checking needs to be done.

`_mainid`

The process ID of the original process.

For non-threaded programs,
`_mainid==procid`. For threaded
programs, one thread is
`_mainid==_procid` and all others
contain different IDs.

Header Files

Most of the Ultra C library calls are declared in header files. A header file declares a set of related functions and includes any necessary types and additional macros needed to use the functions. The contents of these header files are available through the `#include` preprocessing directive. Header files may be included in any order.

Ultra C provides the following header files.

Table 1-2 Ultra C Header Files

File	Description
alarm.h	Used by functions that use the OS-9 alarm functions. <code>alarm.h</code> contains prototypes for the following functions: <code>alm_atdate()</code> <code>alm_atjul()</code> <code>alm_cycle()</code> <code>alm_delete()</code> <code>alm_set()</code> <code>_os9_alarm_atdate()</code> <code>_os_alarm_atime()</code> <code>_os9_alarm_atjul()</code> <code>_os_alarm_cycle()</code> <code>_os_alarm_delete()</code> <code>_os_alarm_reset()</code> <code>_os_alarm_set()</code> <code>_os9_salarm_atdate()</code> <code>_os_salarm_atime()</code> <code>_os9_salarm_atjul()</code> <code>_os_salarm_cycle()</code> <code>_os9_salarm_delete()</code> <code>_os_salarm_reset()</code> <code>_os_salarm_set()</code> <code>_os9_salarm_set()</code>
assert.h †	Defines the <code>assert</code> macro. <code>assert.h</code> contains prototypes for the following function: <code>assert()</code>

† Indicates a standard ANSI header.

Table 1-2 Ultra C Header Files (continued)

File	Description
aton.h	Used for converting alphabetic characters to numeric values. aton.h contains prototypes for the following function:
	<code>_aton()</code>
cache.h	Defines the CPU internal cache control information. cache.h contains prototypes for the following functions:
	<code>_os_cache()</code> <code>_os_scache()</code>
cdi.h (OS-9 for 68K only)	Contains CD-I specific functions. cdi.h contains prototypes for the following function:
	<code>_os9_gs_cdfd()</code>
cglob.h	Declares Ultra C global variables.
ctype.h †	Declares functions that are useful for testing and mapping characters. ctype.h contains prototypes for the following functions:
	<code>isalnum()</code> <code>isalpha()</code> <code>_isascii()</code> , <code>isascii()</code> <code>iscntrl()</code> <code>isdigit()</code> <code>_iseuc_kana()</code> <code>_iseuc1()</code> , <code>_iseuc2()</code> <code>isgraph()</code> <code>_jisjis_kana()</code> <code>islower()</code> <code>isprint()</code> <code>ispunct()</code> <code>_issjis1()</code> <code>_issjis2()</code> <code>isspace()</code> <code>isupper()</code> <code>isxdigit()</code> <code>toascii()</code> , <code>_toascii()</code> <code>tolower()</code> <code>_tolower()</code> <code>toupper()</code> <code>_toupper()</code>

† Indicates a standard ANSI header.

Table 1-2 Ultra C Header Files (continued)

File	Description
dexec.h	Defines information related to debugger controlled processes. dexec.h contains prototypes for the following functions: <code>_os_dexec()_os_dexit() _os_dfork()_os_dforkm() _os9_dfork()</code>
dir.h	Defines structures and functions for BSD 4.2 directory calls. dir.h contains prototypes for the following functions: <code>closedir()opendir() readdir()rewinddir() seekdir()telldir()</code>
direct.h (OS-9 for 68K only)	Defines the RBF data structures. This is obsolete; it has been replaced with rbf.h.
errno.h †	Defines macros relating to error reporting. errno.h contains a prototype for the following function: <code>_os_perr()</code>

† Indicates a standard ANSI header.

Table 1-2 Ultra C Header Files (continued)

File	Description
events.h	Used for functions using the OS-9 event functions. events.h contains prototypes for the following functions:
	<code>_ev_creat()_ev_delete()</code> <code>_ev_info()_ev_link()</code> <code>_ev_pulse()_ev_read()</code> <code>_ev_set()_ev_setr()</code> <code>_ev_signal()_ev_unlink()</code> <code>_ev_wait()_ev_waitr()</code> <code>_os_ev_allclr()_os_ev_allset()</code> <code>_os_ev_anyclr()_os_ev_anyset()</code> <code>_os_ev_change()_os_ev_creat()</code> <code>_os_ev_delete()_os_ev_info()</code> <code>_os_ev_link()_os_ev_pulse()</code> <code>_os_ev_read()_os_ev_set()</code> <code>_os_ev_setand()_os_ev_setor()</code> <code>_os_ev_setr()_os_ev_setxor()</code> <code>_os_ev_signal()_os_ev_tstset()</code> <code>_os_ev_unlink()_os_ev_wait()</code> <code>_os9_ev_wait()_os_ev_waitr()</code> <code>_os9_ev_waitr()_os9_ev_info()</code>
float.h †	Defines the limits for floating point numbers.
funcs.h	Symbolic names for the system calls. funcs.h contains a prototype for the following function:
	<code>_os9_setsvc()</code>

† Indicates a standard ANSI header.

Table 1-2 Ultra C Header Files (continued)

File	Description
getset.h	Used to read and set values of special purpose registers, device control registers, processor registers, and time-based registers. getset.h contains prototypes for the following functions: <code>_get_<name>() _set_<name>()</code>
io.h	Used for functions performing input/output. io.h contains prototypes for the following functions: <code>_os_alias() _os_attach() _os_detach() _os_get_ioproc() _os_getdl() _os_getpd() _os_rdalst() _os_tranpn()</code>
ioctl.h	Contains definitions for ioctl() library function.
ioedt.h	Defines the current I/O interface editions. ioedt.h contains a prototype for the following function: <code>_os_gs_edt()</code>
limits.h †	Defines macros that expand to various limits and parameters.
locale.h †	Used for localization. locale.h contains prototypes for the following functions: <code>localeconv() setlocale()</code>

† Indicates a standard ANSI header.

Table 1-2 Ultra C Header Files (continued)

File	Description
lock.h	<p>Used for the OS-9 resource lock functions. <code>lock.h</code> contains prototypes for the following functions:</p>
	<code>_os_acqlk() _os_caqlk()</code> <code>_os_crlk() _os_dellk()</code> <code>_os_rellk() _os_waitlk()</code>
math.h †	<p>Defines the mathematical functions and macros available with Ultra C. <code>math.h</code> contains prototypes for the following functions:</p>
	<code>acos() asin()</code> <code>atan() atan2()</code> <code>ceil() cos()</code> <code>cosh() exp()</code> <code>fabs() floor()</code> <code>fmod() frexp()</code> <code>hypot() ldexp()</code> <code>log() log10()</code> <code>modf() pow()</code> <code>sin() sinh()</code> <code>sqrt() tan()</code> <code>tanh()</code>

† Indicates a standard ANSI header.

Table 1-2 Ultra C Header Files (continued)

File	Description
memory.h	Functions related to acquiring, examining, and releasing memory. <code>memory.h</code> contains prototypes for the following functions: <code>_os9_allbit()</code> <code>_os9_delbit()</code> <code>_os_get_blkmap()</code> <code>_os_mem()</code> <code>_os_move()</code> <code>_os9_schbit()</code> <code>_os_sqmem()</code> <code>_os9_sqmem()</code> <code>_os_srtmem()</code> <code>_os9_translate()</code> <code>_sqmem()</code> <code>_srtmem()</code> <code>sqcmem()</code>
moddir.h	Module directory information. <code>moddir.h</code> contains prototypes for the following functions: <code>_os_altdmdir()</code> <code>_os_chmdir()</code> <code>_os_cmdperm()</code> <code>_os_delmdir()</code> <code>_os_fmod()</code> <code>_os_get_mdp()</code> <code>_os_makmdir()</code>

† Indicates a standard ANSI header.

Table 1-2 Ultra C Header Files (continued)

File	Description
modes.h	<p>Defines the access modes and permissions for files and directories. modes.h contains prototypes for the following functions:</p> <pre data-bbox="498 407 901 1102"> access() attach() chdir() chown() chmod() chxdir() close() creat() create() detach() dup() lseek() mkdir() mknod() open() _os_chdir() _os_close() _os_create() _os9_create() _os_delete() _os_dup() _os_ioconfig() _os_mkdir() _os_open() _os_read() _os_readln() _os_seek() _os9_ss_close() _os9_ss_open() _os_write() _os_writeln() read() readln() unlink() unlinkx() write() writeln()</pre>

† Indicates a standard ANSI header.

Table 1-2 Ultra C Header Files (continued)

File	Description
module.h	<p>Module-related data structures and functions. module.h contains prototypes for the following functions:</p> <pre data-bbox="508 406 982 1033"> crc()_get_module_dir() make_module()modcload() _mkdata_module()modlink() modload()modloadp() munlink()munload() _os_crc()_os_datmod() _os_get_moddir()_os_initdata() _os_iodel()_os_link() _os_linkm()_os_load() _os_loadp()_os_mkmodule() _os_modaddr()_os_setcrc() _os_slink()_os_slinkm() _os_tlinkm()_os_tlink() _os_unlink()_os_unload() _os_vmodul()_os9_vmodul() _prgname()_setcrc() _sliblink()_subcall()</pre>
mqueue.h	<p>POSIX messaging data structures and functions. mqueue.h contains prototypes for the following functions:</p> <pre data-bbox="508 1213 1210 1308"> mq_close; mq_getattr; mq_notify; _mq_notify_write; mq_open; mq_receive; mq_send; mq_setattr; mq_unlink</pre>

† Indicates a standard ANSI header.

Table 1-2 Ultra C Header Files (continued)

File	Description
os9def.h	A series of macros mapping UNIX macros and function calls to their OS-9 versions.
	<code>alarm()</code> <code>alloca()</code> <code>bcmp()</code> <code>bcopy()</code> <code>buildpath()</code> <code>bzero()</code> <code>cbrt()</code> <code>crtolf()</code> <code>dup2()</code> <code>execl()</code> <code>execle()</code> <code>execv()</code> <code>execve()</code> <code>execvp()</code> <code>ffs()</code> <code>getgid()</code> <code> getopt()</code> <code>getppid()</code> <code>getwd()</code> <code>ioctl()</code> <code>isatty()</code> <code>lftocr()</code> <code>pclose()</code> <code>pipe()</code> <code>popen()</code> <code>putenv()</code> <code>readv()</code> <code>setgid()</code> <code>tempnam()</code> <code>writev()</code>
os9time.h	Defines interval timer support for OS-9.
	<code>gettimeofday()</code>
os9types.h	Synchronous I/O multiplexing. <code>os9types.h</code> contains a prototype for the following function: <code>select()</code>
pmodes.h	POSIX file access and permission modes for OS9000.

† Indicates a standard ANSI header.

Table 1-2 Ultra C Header Files (continued)

File	Description
process.h	<p>Process descriptor and functions that manipulate processes. process.h contains prototypes for the following functions:</p> <pre data-bbox="508 407 1005 1396"> chainc(), chain() _cpymem()_getpid() _get_process_desc() _get_process_table() getuid()_os9_allpd() _os_alltsk()_os_aallocproc() _os_aproc()_os9_aproc() _os_chain()_os_chainm() _os_chkmem()_os_cpy_ioproc() _os_cpymem()_os_ddlk() _os_deltsk()_os_exec() os9exec()_os_exit() _os_findpd()_os9_findpd() _os_fork()_os9fork(), os9forkc() _os_forkm()_os_get_prtbl() _os_gprdsc()_os9_gspump() _os_id()_os9_id() _os_ioexit()_os_iofork() _os9_ioqueue()_os_nproc() _os9_panic()_os_permit() _os_protect()_os9_retpd() _os_rtnprc()_os_setpr() _os_setuid()_os_suspend() _os_sysdbg()_os_uacct() _os_wait()_setpr() setuid()_sysdbg() wait()</pre>

† Indicates a standard ANSI header.

Table 1-2 Ultra C Header Files (continued)

File	Description
procid.h	Process descriptor data structure. procid.h is obsolete; it has been replaced with process.h.

† Indicates a standard ANSI header.

Table 1-2 Ultra C Header Files (continued)

File	Description
pthread.h	<p>POSIX thread support routines.</p> <p>The following functions are included in pthread.h:</p> <pre data-bbox="508 416 1210 1456"> pthread_attr_destroy() pthread_attr_getdetachstate() __pthread_attr_getinitfunction() __pthread_attr_getpriority() pthread_attr_getstackaddr() pthread_attr_getstacksize() pthread_attr_init () pthread_attr_setdetachstate() __pthread_attr_setinitfunction() __pthread_attr_setpriority() pthread_attr_setstackaddr() pthread_attr_setstacksize(), pthread_cancel() pthread_cleanup_pop(), pthread_cleanup_push() pthread_cond_broadcast() pthread_cond_destroy(), pthread_cond_init() pthread_cond_signal() pthread_cond_timedwait(), pthread_cond_wait() pthread_condattr_destroy() pthread_condattr_getpshared() pthread_condattr_init() pthread_condattr_setpshared() pthread_create(), pthread_detach() pthread_equal(), pthread_exit() pthread_getspecific(), __pthread_getstatus() __pthread_interrupt(), __pthread_interrupt_clear(), pthread_join() pthread_key_create() pthread_key_delete() pthread_mutex_destroy() pthread_mutex_getprioceiling()</pre>

† Indicates a standard ANSI header.

Table 1-2 Ultra C Header Files (continued)

File	Description
pthread.h (cont.)	<p><code>pthread_mutex_init()</code>, <code>pthread_mutex_lock()</code> <code>pthread_mutex_setprioceiling()</code> <code>pthread_mutex_trylock()</code> <code>pthread_mutex_unlock()</code> <code>pthread_mutexattr_destroy()</code> <code>pthread_mutexattr_getprioceiling()</code> <code>pthread_mutexattr_getprotocol()</code> <code>pthread_mutexattr_getpshared()</code> <code>pthread_mutexattr_init()</code> <code>pthread_mutexattr_setprioceiling()</code> <code>pthread_mutexattr_setprotocol()</code> <code>pthread_mutexattr_setpshared()</code> <code>pthread_once()</code>, <code>_pthread_resume()</code> <code>pthread_self()</code>, <code>pthread_setcancelstate()</code> <code>pthread_setcanceltype()</code>, <code>_pthread_setpr()</code> <code>_pthread_setsignalrange()</code> <code>pthread_setspecific()</code> <code>_pthread_setsuspendable()</code> <code>_pthread_setunsuspendable()</code> <code>_pthread_suspend()</code>, <code>pthread_testcancel()</code></p>
pwd.h	Contains the password file definitions.
	<code>endpwent()</code> <code>fgetpwent()</code> <code>getnextpwnam()</code> <code>getpw()</code> <code>getpwent()</code> <code>getpwnam()</code> <code>getpwuid()</code> <code>setpwent()</code>

† Indicates a standard ANSI header.

Table 1-2 Ultra C Header Files (continued)

File	Description
rbf.h	Used for functions using the random block file manager (RBF). rbf.h contains prototypes for the following functions: <code>_gs_gfd()_os_gs_cstats() _os_gs_dsize()_os_gs_fd() _os_gs_fdaddr()_os_gs_fdinf() _os_gs_parity()_os_ss_cache() _os_ss_fd()_os_ss_flushmap() _os_ss_hdlink()_os_ss_lock() _os_ss_ticks()_os_ss_wtrack() _ss_lock()_ss_pfd() _ss_tiks()_ss_wtrk()</code>
regexp.h	Contains definitions for regular expression support. <code>regcomp()regerror() regex(), regexec()</code>

† Indicates a standard ANSI header.

Table 1-2 Ultra C Header Files (continued)

File	Description
regs.h	<p>Used to include the proper register header file for the target processor. <code>regs.h</code> contains prototypes for the following functions:</p> <pre data-bbox="508 407 964 877"> change_const()change_static() get_const()get_static() inc() inl() inw()irq_change() irq_disable()irq_enable() irq_maskget()irq_restore() irq_save()make_gdesc() _os_cache()(OS-9 only) _os_firq()_os_irq() _os9_irq()(OS-9 for 68K only) _os_scache()_os_sysid() outc() outl() outw()</pre>
sbf.h	<p>Used for functions using the sequential block file manager (SBF). <code>sbf.h</code> contains prototypes for the following functions:</p> <pre data-bbox="508 1060 978 1157"> _os_ss_erase()_os_ss_rfm() _os_ss_reten()_os_ss_skipend() _os_ss_skip()_os_ss_wfm()</pre>

† Indicates a standard ANSI header.

Table 1-2 Ultra C Header Files (continued)

File	Description
scf.h	<p>Used for functions using the sequential character file manager (SCF). scf.h contains prototypes for the following functions:</p> <pre>_os_ss_dcoff()_os_ss_dcon() _os_ss_dopt()_os_ss_dsrt() _os_ss_enrt()_os_ss_fillbuff() _ss_dcoff()_ss_dcon() _ss_dsrt()_ss_enrt()</pre>
sched.h	<p>Used for the POSIX schedule family of functions. sched.h contains prototypes for the following function:</p> <pre>sched_yield()</pre>
semaphore.h	<p>Used for functions using binary semaphores. semaphore.h contains prototypes for the following functions:</p> <pre>_os_sema_init()_os_sema_p()_os_sema_term() _os_sema_v()</pre>
setjmp.h †	<p>Used for bypassing the normal function call and return mechanism. setjmp.h contains prototypes for the following functions:</p> <pre>longjmp()setjmp()</pre>
setsys.h	<p>System global symbolic names. This is obsolete. It has been replaced with sysglob.h. setsys.h contains prototypes for the following functions:</p> <pre>_getsys()_setsys()</pre>

† Indicates a standard ANSI header.

Table 1-2 Ultra C Header Files (continued)

File	Description
<code>settrap.h</code>	Data structures and functions used to trap user-state exceptions. <code>settrap.h</code> contains prototypes for the following functions:
	<code>_os_strap()</code> <code>_os9_strap()</code>
<code>sg_codes.h</code>	Symbolic names for SetStats and GetStats. <code>sg_codes.h</code> contains prototypes for the following functions:
	<code>getstat()</code> <code>_gs_devn()</code> <code>_gs_eof()</code> <code>_gs_opt()</code> <code>_gs_pos()</code> <code>_gs_rdy()</code> <code>_gs_size()</code> <code>_os_getstat()</code> <code>_os_gs_cpypd()</code> <code>_os_gs_devnm()</code> <code>_os_gs_devtyp()</code> <code>_os_gs_dopt()</code> <code>_os_gs_eof()</code> <code>_os9_gs_free()</code> <code>_os_gs_luopt()</code> <code>_os_gs_popt()</code> <code>_os_gs_pos()</code> <code>_os_gs_ready()</code> <code>_os_gs_size()</code> <code>_os_setstat()</code> <code>_os_sgetstat()</code> <code>_os_sgs_devnm()</code> <code>_os_ss_attr()</code> <code>_os_ss_break()</code> <code>_os_ss_dopt()</code> <code>_os_ss_link_adj()</code> <code>_os_ss_luopt()</code> <code>_os_ss_popt()</code> <code>_os_ss_relea()</code> <code>_os_ss_rename()</code> <code>_os_ss_reset()</code> <code>_os_ss_sendsig()</code> <code>_os_ss_size()</code> <code>setstat()</code> <code>_ss_attr()</code> <code>_ss_opt()</code> <code>_ss_rel()</code> <code>_ss_rest()</code> <code>_ss_size()</code> <code>_ss_ssig()</code>
<code>sgstat.h</code>	Path descriptor option section definitions.

† Indicates a standard ANSI header.

Table 1-2 Ultra C Header Files (continued)

File	Description
signal.h †	Used for handling various signals. signal.h contains prototypes for the following functions:
	<code>intercept()</code> <code>kill()</code> <code>_os_clrsigs()</code> <code>_os_intercept()</code> <code>_os_rte()</code> <code>_os_send()</code> <code>_os_siglngth()</code> <code>_os_sigmask()</code> <code>_os_sigreset()</code> <code>_os_sigrs()</code> <code>_os_sleep()</code> <code>_os9_sleep()</code> <code>pause()</code> <code>raise()</code> <code>sigmask()</code> <code>signal()</code> <code>sleep()</code> <code>tsleep()</code>
stat.h	Contains definitions for <code>stat()</code> and <code>fstat()</code> support.
stdarg.h †	Used for advancing through a list of arguments of variable length and type. stdarg.h contains the macros for:
	<code>va_arg()</code> <code>va_end()</code> <code>va_start()</code>
stddef.h †	Used for common definitions (<code>ptrdiff_t</code> , <code>size_t</code> , and <code>wchar_t</code>) and the following macro:
	<code>offsetof()</code>

† Indicates a standard ANSI header.

Table 1-2 Ultra C Header Files (continued)

File	Description
stdio.h †	Used for performing input and output. stdio.h contains prototypes for the following functions: <code>_cleareof(), cleareof() clearerr()_errmsg() fclose()fdopen() feof() ferror() fflush()fgetc() fgetpos()fgets() _fileno(), fileno() fopen() fprintf()fputc() fputs()fread() freopen()fscanf() fseek()fsetpos() ftell()fwrite() getc(), getchar() gets() getw() mktemp() perror() prerr()printf() putc(), putchar() puts() putw() remove()rename() rewind()scanf() setbuf()setvbuf() sprintf()sscanf() tmpfile()tmpnam() ungetc()vfprintf() vprintf()vsprintf()</code>

† Indicates a standard ANSI header.

Table 1-2 Ultra C Header Files (continued)

File	Description
stdlib.h †	<p>Used for the standard library functions. stdlib.h contains prototypes for the following functions:</p> <pre>abort() abs() atexit() atof() atoi() atol() _atou()bsearch() calloc() div() dtoa() ebrk() exit() _exit() free() _freemem(), freemem() _freemin()ibrk() getenv() labs() _lcalloc()ldiv() _lfree()_lmalloc() _lrealloc()malloc() _mallocmin()mblen() mbstowcs()mbtowc() qsort(), _prealloc_rand(), rand() realloc()sbrk() srand() stacksiz(), _stacksiz() strtod()strtol() strtoul()system() wcstombs()wctomb()</pre>

† Indicates a standard ANSI header.

Table 1-2 Ultra C Header Files (continued)

File	Description
string.h †	<p>Used for manipulating arrays of character type. string.h contains prototypes for the following functions:</p> <pre data-bbox="498 406 806 803"> memchr() memcmp() memcpy() memmove() memset() strcat() strchr() strcmp() strcoll() strcpy() strcspn() strerror() strlen() strncat() strncmp() strncpy() strupr() strrchr() strspn() strstr() strtok() strxfrm()</pre>
strings.h	<p>Used for non-ANSI string functions. strings.h contains prototypes for the following functions:</p> <pre data-bbox="498 950 887 1157"> _cmpnam() findnstr() findstr() index() _os_cmpnam() _os_prsnam() pause() _prsnam() rindex() _strass() strhcpy()</pre>
svctbl.h	<p>Contains information for the service table. svctbl.h contains a prototypes for the following function:</p> <pre data-bbox="498 1344 692 1378"> _os_setsvc()</pre>

† Indicates a standard ANSI header.

Table 1-2 Ultra C Header Files (continued)

File	Description
sysglob.h	Defines the system global variables. sysglob.h contains prototypes for the following functions: <code>_getsys()_os_config()</code> <code>_os_getsys()_os_setsys()</code> <code>_setsys()</code>
sys/wait.h	Contains prototype for the following function: <code>waitpid()</code>
termcap.h	Used for the terminal capability functions. termcap.h contains prototypes for the following functions: <code>tgetent()tgetflag()</code> <code>tgetnum()tgetstr()</code> <code>tgoto() tputs()</code>
termio.h	Contains definition to support ioctl.
threads.h	Process thread support routines. threads.h contains prototypes for the following functions: <code>_os_thexit()</code> <code>_os_thfork()</code> <code>_os_thread()</code>

† Indicates a standard ANSI header.

Table 1-2 Ultra C Header Files (continued)

File	Description
time.h †	Used for manipulating time. time.h contains prototypes for the following functions:
	<code>asctime()</code> <code>clock()</code> <code>ctime()</code> <code>difftime()</code> <code>getime()</code> <code>gmtime()</code> <code>gregorian()</code> <code>_julian()</code> <code>localtime()</code> <code>mktime()</code> <code>_os_getime()</code> <code>_os9_getime()</code> <code>_os_gregorian()</code> <code>_os_julian()</code> <code>_os_setime()</code> <code>_os9_setime()</code> <code>setime()</code> <code>strftime()</code> <code>_sysdate()</code> <code>sys_mktime()</code> <code>time()</code>
times.h	Contains definitions to support times. <code>times()</code>
types.h	Defines <code>typedefs</code> for consistently sized and signed types.
un.h	Contains the <code>sockaddr_un</code> structure definition.
virtual.h	Contains prototype for the following functions: <code>_os_transadd()</code>

† Indicates a standard ANSI header.

The C Library

This section includes a listing of the available C functions, grouped according to functionality. Each listing includes the function's name, description, and the header file in which it is defined.



Note

Documented features and/or functionality classified as “undefined” by the relevant ANSI/ISO C specification are subject to change in future version of the Ultra C libraries. Be certain that the code you are using does not depend on the behavior of undefined situations, as documented by the ANSI/ISO specification.

Communicating with Environment Functions

The Ultra C libraries contain the functions listed in **Table 1-3** for communicating with the environment.

Table 1-3 Environment Communications Functions

Function	ANSI Standard	Description	Header File
<code>abort()</code>	•	Abnormal program termination	<code>stdlib.h</code>
<code>atexit()</code>	•	Register function to call at normal program termination	<code>stdlib.h</code>
<code>_exit()</code>		Task termination	<code>stdlib.h</code>
<code>exit()</code>	•	Task termination	<code>stdlib.h</code>

Table 1-3 Environment Communications Functions

Function	ANSI Standard	Description	Header File
<code>getenv()</code>	•	Get value for environment name	<code>stdlib.h</code>
<code>_os_exit()</code>		Terminate calling process	<code>process.h</code>
<code>system()</code>	•	Shell command execution	<code>stdlib.h</code>

Character Classification Functions

The character classification functions are really macros defined in `<ctype.h>`. Be careful when using these macros to avoid macro expansion side-effects. The macros provide a platform independent method of character classification. **Table 1-4** identifies character classification functions.

Table 1-4 Character Classification Functions

Function	ANSI Standard	Description	Header File
<code>isalnum()</code>	•	Test if parameter is alphanumeric	<code>ctype.h</code>
<code>isalpha()</code>	•	Test if parameter is alphabetic	<code>ctype.h</code>
<code>_isascii(), isascii()</code>		Test if parameter is ASCII	<code>ctype.h</code>
<code>iscntrl()</code>	•	Test if parameter is control code	<code>ctype.h</code>
<code>isdigit()</code>	•	Test if parameter is digit	<code>ctype.h</code>

Table 1-4 Character Classification Functions (continued)

Function	ANSI Standard	Description	Header File
<code>isgraph()</code>	•	Test if parameter is printing character	<code>ctype.h</code>
<code>_isjis_kana()</code>		Test if parameter is Hankaku-Kana	<code>ctype.h</code>
<code>islower()</code>	•	Test if parameter is lowercase	<code>ctype.h</code>
<code>isprint()</code>	•	Test if parameter is printable character	<code>ctype.h</code>
<code>ispunct()</code>	•	Test if parameter is punctuation character	<code>ctype.h</code>
<code>_issjis1()</code>		Test if parameter is the first byte of Kanji	<code>ctype.h</code>
<code>_issjis2()</code>		Test if parameter is the second byte of Kanji	<code>ctype.h</code>
<code>isspace()</code>	•	Test if parameter is white space	<code>ctype.h</code>
<code>isupper()</code>	•	Test if parameter is uppercase	<code>ctype.h</code>
<code>isxdigit()</code>	•	Test if parameter is hexadecimal	<code>ctype.h</code>

Character Conversion Functions

The character conversion functions allow you to convert a string of characters to their numeric representation and to change the case of characters. **Table 1-5** identifies character conversion functions.

Table 1-5 Character Conversion Functions

Function	ANSI Standard	Description	Header File
<code>atof()</code>	•	Alpha to floating conversion	<code>stdlib.h</code>
<code>atoi()</code>	•	Alpha to integer conversion	<code>stdlib.h</code>
<code>atol()</code>	•	Alpha to long conversion	<code>stdlib.h</code>
<code>_aton()</code>		Alpha to numeric translation	<code>aton.h</code>
<code>_atou()</code>		Alpha to unsigned conversion	<code>stdlib.h</code>
<code>_dtoa()</code>		Double to ASCII conversion	<code>stdlib.h</code>
<code>toascii()</code> , <code>_toascii()</code>		Integer to ASCII translation	<code>ctype.h</code>
<code>_tolower()</code>		Convert character to lowercase	<code>ctype.h</code>
<code>tolower()</code>	•	Convert character to lowercase	<code>ctype.h</code>
<code>_toupper()</code>		Convert character to uppercase	<code>ctype.h</code>
<code>toupper()</code>	•	Convert character to uppercase	<code>ctype.h</code>

File Positioning Functions

The functions in **Table 1-6** are provided for file positioning.

Table 1-6 File Positioning Functions

Function	ANSI Standard	Description	Header File
<code>fgetpos()</code>	•	Get current position in file	stdio.h
<code>fseek()</code>	•	Reposition file pointer	stdio.h
<code>fsetpos()</code>	•	Reposition file pointer	stdio.h
<code>ftell()</code>	•	Get current position in file	stdio.h
<code>_os_seek()</code>		Reposition file pointer	modes.h
<code>rewind()</code>	•	Return file pointer to zero	stdio.h

GetStat/SetStat Functions

The standard library provides the `GetStat` and `SetStat` functions for determining and setting individual device parameters that are not uniform on all devices or that are highly hardware dependent.

Table 1-7 GetStat/SetStat Functions

Function	ANSI Standard	Description	Header File
<code>getstat()</code>		Get file status	sg_codes.h
<code>_gs_devn()</code>		Get device name	sg_codes.h

Table 1-7 GetStat/SetStat Functions (continued)

Function	ANSI Standard	Description	Header File
<code>_gs_eof()</code>		Check for end of file	<code>sg_codes.h</code>
<code>_gs_gfd()</code>		Get file descriptor sector	<code>none</code>
<code>_gs_opt()</code>		Get path options	<code>sg_codes.h</code>
<code>_gs_pos()</code>		Get current file position	<code>sg_codes.h</code>
<code>_gs_rdy()</code>		Test for data available	<code>sg_codes.h</code>
<code>_gs_size()</code>		Get current file size	<code>sg_codes.h</code>
<code>_os_getstat()</code>		Get file/device status	<code>sg_codes.h</code>
<code>_os_getsys()</code>		Examine system global variable	<code>sysglob.h</code>
<code>_os9_gs_cdfd()</code>		Return file descriptor	<code>cdi.h</code>
<code>_os_gs_cpypd()</code>		Copy contents of path descriptor	<code>sg_codes.h</code>
<code>_os_gs_cstats()</code>		Get cache status information	<code>rbf.h</code>
<code>_os_gs_devnm()</code>		Return device name	<code>sg_codes.h</code>
<code>_os_gs_devtyp()</code>		Return device type	<code>sg_codes.h</code>
<code>_os_gs_dopt()</code>		Read device path options	<code>sg_codes.h</code>
<code>_os_gs_dsize()</code>		Get size of SCSI devices	<code>rbf.h</code>

Table 1-7 GetStat/SetStat Functions (continued)

Function	ANSI Standard	Description	Header File
<code>_os_gs_edt()</code>		Get I/O interface edition number	<code>ioedt.h</code>
<code>_os_gs_eof()</code>		Test for end of file	<code>sg_codes.h</code>
<code>_os_gs_fd()</code>		Read file descriptor sector	<code>rbf.h</code>
<code>_os_gs_fdaddr()</code>		Get file descriptor block address for open file	<code>rbf.h</code>
<code>_os_gs_fdinf()</code>		Get specified file descriptor sector	<code>rbf.h</code>
<code>_os9_gs_free()</code>		Return amount of free space on device	<code>sg_codes.h</code>
<code>_os_gs_luopt()</code>		Read logical unit options	<code>sg_codes.h</code>
<code>_os_gs_parity()</code>		Calculate parity of file descriptor	<code>rbf.h</code>
<code>_os_gs_popt()</code>		Read path descriptor option section	<code>sg_codes.h</code>
<code>_os_gs_pos()</code>		Get current file position	<code>sg_codes.h</code>
<code>_os_gs_ready()</code>		Test for data ready	<code>sg_codes.h</code>
<code>_os_gs_size()</code>		Return current file size	<code>sg_codes.h</code>
<code>_os_setstat()</code>		Set file/device status	<code>sg_codes.h</code>
<code>_os_setsvc()</code>		Adds or replaces a system service	<code>funcs.h</code>

Table 1-7 GetStat/SetStat Functions (continued)

Function	ANSI Standard	Description	Header File
<code>_os_setsys()</code>		Set system global variables	<code>sysglob.h</code>
<code>_os_sgetstat()</code>		GetStat call using system path number	<code>sg_codes.h</code>
<code>_os_sgs_devnm()</code>		Return device name	<code>sg_codes.h</code>
<code>_os_ss_attr()</code>		Set file attributes	<code>sg_codes.h</code>
<code>_os_ss_break()</code>		Break serial connection	<code>sg_codes.h</code>
<code>_os_ss_cache()</code>		Enable/disable RBF caching	<code>rbf.h</code>
<code>_os9_ss_close()</code>		Notify driver that path has been closed	<code>modes.h</code>
<code>_os_ss_dcoff()</code>		Send signal when data carrier detect line goes false	<code>scf.h</code>
<code>_os_ss_dcon()</code>		Send signal when data carrier detect line goes true	<code>scf.h</code>
<code>_os_ss_dopt()</code>		Set device path options	<code>sg_codes.h</code>
<code>_os_ss_dsrts()</code>		Disable RTS line	<code>scf.h</code>
<code>_os_ss_enrts()</code>		Enable RTS line	<code>scf.h</code>
<code>_os_ss_erase()</code>		Erase tape	<code>sbf.h</code>

Table 1-7 GetStat/SetStat Functions (continued)

Function	ANSI Standard	Description	Header File
<code>_os_ss_fd()</code>		Write file descriptor sector	<code>rbf.h</code>
<code>_os_ss_fillbuff()</code>		Fill path buffer with data	<code>scf.h</code>
<code>_os_ss_flushmap()</code>		Flush cached bit map information for RBF device	<code>rbf.h</code>
<code>_os_ss_hdlink()</code>		Make hard link to existing file	<code>rbf.h</code>
<code>_os_ss_lock()</code>		Lock out record	<code>rbf.h</code>
<code>_os_ss_luopt()</code>		Write logical unit options	<code>sg_codes.h</code>
<code>_os9_ss_open()</code>		Notify driver that path has been opened	<code>modes.h</code>
<code>_os_ss_popt()</code>		Write option section of path descriptor	<code>sg_codes.h</code>
<code>_os_ss_relea()</code>		Release device	<code>sg_codes.h</code>
<code>_os_ss_rename()</code>		Rename file	<code>sg_codes.h</code>
<code>_os_ss_reset()</code>		Restore head to track zero	<code>sg_codes.h</code>
<code>_os_ss_reten()</code>		Retention pass on tape drive	<code>sbf.h</code>
<code>_os_ss_rfm()</code>		Skip tape marks	<code>sbf.h</code>

Table 1-7 GetStat/SetStat Functions (continued)

Function	ANSI Standard	Description	Header File
<code>_os_ss_sendsig()</code>		Send signal on data ready	<code>sg_codes.h</code>
<code>_os_ss_size()</code>		Set file size	<code>sg_codes.h</code>
<code>_os_ss_skip()</code>		Skip blocks	<code>sbf.h</code>
<code>_os_ss_skipend()</code>		Skip to end of tape	<code>sbf.h</code>
<code>_os_ss_ticks()</code>		Wait specified number of ticks for record release	<code>rbf.h</code>
<code>_os_ss_wfm()</code>		Write tape marks	<code>sbf.h</code>
<code>_os_ss_wtrack()</code>		Write (format) track	<code>rbf.h</code>
<code>setstat()</code>		Set file status	<code>sg_codes.h</code>
<code>_ss_attr()</code>		Set file attributes	<code>sg_codes.h</code>
<code>_ss_dcoff()</code>		Send data carrier lost signal to process	<code>scf.h</code>
<code>_ss_dcon()</code>		Send data carrier present signal to process	<code>scf.h</code>
<code>_ss_dsrts()</code>		Disable RTS line	<code>scf.h</code>
<code>_ss_enrts()</code>		Enable RTS line	<code>scf.h</code>
<code>_ss_lock()</code>		Lock out record	<code>rbf.h</code>
<code>_ss_opt()</code>		Set path options	<code>sg_codes.h</code>

Table 1-7 GetStat/SetStat Functions (continued)

Function	ANSI Standard	Description	Header File
<code>_ss_pfd()</code>		Put file descriptor sector	<code>rbf.h</code>
<code>_ss_rel()</code>		Release device	<code>sg_codes.h</code>
<code>_ss_rest()</code>		Restore device	<code>sg_codes.h</code>
<code>_ss_size()</code>		Set current file size	<code>sg_codes.h</code>
<code>_ss_ssig()</code>		Send signal on data ready	<code>sg_codes.h</code>
<code>_ss_tiks()</code>		Wait for record release	<code>rbf.h</code>
<code>_ss_wtrk()</code>		Write track	<code>rbf.h</code>

Input/Output Device Functions

The following functions are available for performing input/output on devices.

Table 1-8 Input/Output Functions

Function	ANSI Standard	Description	Header File
<code>inc()</code>		Get character from I/O address	<code>regs.h</code>
<code>inl()</code>		Get integer from I/O address	<code>regs.h</code>
<code>inw()</code>		Get word from I/O address	<code>regs.h</code>

Table 1-8 Input/Output Functions (continued)

Function	ANSI Standard	Description	Header File
<code>outc()</code>		Write character to I/O address	<code>regs.h</code>
<code>outl()</code>		Write integer to I/O address	<code>regs.h</code>
<code>outw()</code>		Write word to I/O address	<code>regs.h</code>

Input/Output Functions

The following functions make up the I/O functions of the Ultra C libraries. ANSI prototypes for non-ANSI functions are achieved by defining the macro name `_OPT_PROTOS` during the compile.

Table 1-9 Ultra C I/O Functions

Function	ANSI Standard	Description	Header File
<code>access()</code>		Determine file accessibility	<code>modes.h</code>
<code>attach()</code>		Attach to device	<code>modes.h</code>
<code>chdir()</code>		Change current data directory	<code>modes.h</code>
<code>chmod()</code>		Change file access permissions	<code>modes.h</code>
<code>chown()</code>		Change owner of file	<code>modes.h</code>
<code>chxdir()</code>		Change current execution directory	<code>modes.h</code>

Table 1-9 Ultra C I/O Functions (continued)

Function	ANSI Standard	Description	Header File
<code>_cleareof()</code> , <code>cleareof()</code>		Clear end of file condition	stdio.h
<code>clearerr()</code>	•	Clear error condition	stdio.h
<code>close()</code>		Close path	modes.h
<code>closedir()</code>		Close named directory stream	dir.h
<code>creat()</code>		Create file	modes.h
<code>create()</code>		Create file	modes.h
<code>detach()</code>		Detach device	modes.h
<code>dup()</code>		Duplicate path	modes.h
<code>fclose()</code>	•	Close file	stdio.h
<code>fdopen()</code>		Attach path to file pointer	stdio.h
<code>feof()</code>	•	Check buffered file for end of file	stdio.h
<code>ferror()</code>	•	Check buffered file for error condition	stdio.h
<code>fflush()</code>	•	Flush file's buffer	stdio.h
<code>fgetc()</code>	•	Get character from file	stdio.h
<code>fgets()</code>	•	Get string from file	stdio.h

Table 1-9 Ultra C I/O Functions (continued)

Function	ANSI Standard	Description	Header File
<code>_fileno()</code> , <code>fileno()</code>		Determine path number from file	stdio.h
<code>fopen()</code>	•	Open file	stdio.h
<code>fprintf()</code>	•	Formatted output	stdio.h
<code>fputc()</code>	•	Output character to file	stdio.h
<code>fputs()</code>	•	Output string to file	stdio.h
<code>fread()</code>	•	Read data from file	stdio.h
<code>freopen()</code>	•	Re-open file	stdio.h
<code>fscanf()</code>	•	Input string conversion	stdio.h
<code>fwrite()</code>	•	Write data to file	stdio.h
<code>getc()</code> , <code>getchar()</code>	•	Get next character from file/standard input	stdio.h
<code>gets()</code>	•	Get string from file	stdio.h
<code>getw()</code>		Read word from file	stdio.h
<code>lseek()</code>		Position file pointer	modes.h
<code>mkdir()</code>		Create directory	modes.h
<code>mknod()</code>		Create directory	modes.h
<code>mktemp()</code>		Create unique file name	stdio.h

Table 1-9 Ultra C I/O Functions (continued)

Function	ANSI Standard	Description	Header File
<code>open()</code>		Open file	<code>modes.h</code>
<code>opendir()</code>		Open directory	<code>dir.h</code>
<code>_os_alias()</code>		Create device alias	<code>io.h</code>
<code>_os_attach()</code>		Attach new device to system	<code>io.h</code>
<code>_os_chdir()</code>		Change working directory	<code>modes.h</code>
<code>_os_close()</code>		Close path to file/device.	<code>modes.h</code>
<code>_os_cpy_ioproc()</code>		Get pointer to I/O process descriptor.	<code>process.h</code>
<code>_os_create()</code>		Create path to new file	<code>modes.h</code>
<code>_os9_create()</code>		Create path to new file	<code>modes.h</code>
<code>_os_delete()</code>		Delete file	<code>modes.h</code>
<code>_os_detach()</code>		Remove device from system	<code>io.h</code>
<code>_os_dup()</code>		Duplicate path	<code>modes.h</code>
<code>_os_get_ioproc()</code>		Get pointer to I/O process descriptor	<code>io.h</code>
<code>_os_getd1()</code>		Get system I/O device list head pointer	<code>io.h</code>
<code>_os_iocconfig()</code>		Configure an element of process/system I/O	<code>modes.h</code>

Table 1-9 Ultra C I/O Functions (continued)

Function	ANSI Standard	Description	Header File
<code>_os_iodel()</code>		Check for use of I/O module	<code>module.h</code>
<code>_os_ioexit()</code>		Terminate I/O for exiting process	<code>process.h</code>
<code>_os_iofork()</code>		Set up I/O for new process	<code>process.h</code>
<code>_os9_ioqueue()</code>		Enter I/O queue	<code>process.h</code>
<code>_os_mkdir()</code>		Make new directory	<code>modes.h</code>
<code>_os_open()</code>		Open path to file or device	<code>modes.h</code>
<code>_os_rdalst()</code>		Copy system alias list	<code>io.h</code>
<code>_os_read()</code>		Read data from file or device	<code>modes.h</code>
<code>_os_readln()</code>		Read text line with editing	<code>modes.h</code>
<code>_os_tranpn()</code>		Translate user path to system path	<code>io.h</code>
<code>_os_write()</code>		Write data to file or device	<code>modes.h</code>
<code>_os_writeln()</code>		Write line of text with editing	<code>modes.h</code>
<code>pffinit()</code>		Initialize for float output (obsolete)	none
<code>pflinit()</code>		Initialize for longs output (obsolete)	none
<code>printf()</code>	•	Formatted output	<code>stdio.h</code>

Table 1-9 Ultra C I/O Functions (continued)

Function	ANSI Standard	Description	Header File
<code>putc()</code> , <code>putchar()</code>	•	Put next character to file/standard out	<code>stdio.h</code>
<code>puts()</code>	•	Output string to file	<code>stdio.h</code>
<code>putw()</code>		Output word to file	<code>stdio.h</code>
<code>read()</code>		Read bytes from path	<code>modes.h</code>
<code>readdir()</code>		Return pointer	<code>dir.h</code>
<code>readln()</code>		Read bytes from path	<code>modes.h</code>
<code>remove()</code>	•	Remove file	<code>stdio.h</code>
<code>rename()</code>	•	Rename file	<code>stdio.h</code>
<code>seekdir()</code>		Set position of next readdir	<code>dir.h</code>
<code>scanf()</code>	•	Input strings conversion	<code>stdio.h</code>
<code>setbuf()</code>	•	Fix file buffer	<code>stdio.h</code>
<code>setvbuf()</code>	•	Set up buffer for I/O stream	<code>stdio.h</code>
<code>sprintf()</code>	•	Formatted strings output	<code>stdio.h</code>
<code>sscanf()</code>	•	Input strings conversion	<code>stdio.h</code>
<code>telldir()</code>		Return current location	<code>dir.h</code>
<code>tmpfile()</code>	•	Create temporary binary file	<code>stdio.h</code>

Table 1-9 Ultra C I/O Functions (continued)

Function	ANSI Standard	Description	Header File
<code>ungetc()</code>	•	Unget character	<code>stdio.h</code>
<code>unlink()</code>		Unlink (delete) file	<code>modes.h</code>
<code>unlinkx()</code>		Unlink (delete) file	<code>modes.h</code>
<code>vfprintf()</code>	•	Print to file	<code>stdio.h</code>
<code>vprintf()</code>	•	Print to standard output	<code>stdio.h</code>
<code>vsprintf()</code>	•	Print to string	<code>stdio.h</code>
<code>write()</code>		Write bytes to path	<code>modes.h</code>
<code>writeln()</code>		Write bytes to path	<code>modes.h</code>

Interprocess Communication Functions

Functions available for interprocess communication are shown in **Table 1-10**. ANSI prototypes for non-ANSI functions are achieved by defining the macro name _OPT_PROTOS during the compile.

Table 1-10 Interprocess Communication Functions

Function	ANSI Standard	Description	Header File
alm_atdate()		Send signal at Gregorian date/time	alarm.h
alm_atjul()		Send signal at Julian date/time	alarm.h
alm_cycle()		Send signal at specified time intervals	alarm.h
alm_delete()		Remove pending alarm request	alarm.h
alm_set()		Send signal after specified time interval	alarm.h
_ev_create()		Create event	events.h
_ev_delete()		Delete event	events.h
_os9_ev_info()		Returns event information in the caller's buffer	events.h
_ev_info()		Get event information	events.h

Table 1-10 Interprocess Communication Functions (continued)

Function	ANSI Standard	Description	Header File
<code>_ev_link()</code>		Link to existing event	<code>events.h</code>
<code>_ev_pulse()</code>		Signal event occurrence	<code>events.h</code>
<code>_ev_read()</code>		Read event without waiting	<code>events.h</code>
<code>_ev_set()</code>		Set event variable and signal event occurrence	<code>events.h</code>
<code>_ev_setr()</code>		Set relative event variable and signal event occurrence	<code>events.h</code>
<code>_ev_signal()</code>		Signal event occurrence	<code>events.h</code>
<code>_ev_unlink()</code>		Unlink event	<code>events.h</code>
<code>_ev_wait()</code>		Wait for event	<code>events.h</code>
<code>_ev_waitr()</code>		Wait for relative event	<code>events.h</code>
<code>intercept()</code>		Set up process signal handler	<code>signal.h</code>
<code>_os9_alarm_atdate()</code>		Send signal at Gregorian date/time	<code>alarm.h</code>
<code>_os9_alarm_atjul()</code>		Send signal at Julian date/time	<code>alarm.h</code>

Table 1-10 Interprocess Communication Functions (continued)

Function	ANSI Standard	Description	Header File
<code>_os_alarm_atime()</code>		Send signal at specified time	alarm.h
<code>_os_alarm_cycle()</code>		Send signal at specified time intervals	alarm.h
<code>_os_alarm_delete()</code>		Remove pending alarm request	alarm.h
<code>_os_alarm_reset()</code>		Reset existing alarm request	alarm.h
<code>_os_alarm_set()</code>		Send signal after specified time interval	alarm.h
<code>_os_clrsigs()</code>		Clear process signal queue	signal.h
<code>_os_ev_allclr()</code>		Wait for all bits defined by mask to become clear	events.h
<code>_os_ev_allset()</code>		Wait for event to occur	events.h
<code>_os_ev_anycclr()</code>		Wait for event to occur	events.h
<code>_os_ev_anyset()</code>		Wait for event to occur	events.h
<code>_os_ev_change()</code>		Wait for event to occur	events.h
<code>_os_ev_creat()</code>		Create new event	events.h

Table 1-10 Interprocess Communication Functions (continued)

Function	ANSI Standard	Description	Header File
<code>_os_ev_delete()</code>		Delete existing event	events.h
<code>_os_ev_info()</code>		Return event information	events.h
<code>_os_ev_link()</code>		Link to existing event by name	events.h
<code>_os_ev_pulse()</code>		Signal event occurrence	events.h
<code>_os_ev_read()</code>		Read event value without waiting	events.h
<code>_os_ev_set()</code>		Set event variable and signal event occurrence	events.h
<code>_os_ev_setand()</code>		Set event variable and signal event occurrence	events.h
<code>_os_ev_setor()</code>		Set event variable and signal event occurrence	events.h
<code>_os_ev_setr()</code>		Set relative event variable and signal event occurrence	events.h
<code>_os_ev_setxor()</code>		Set event variable and signal event occurrence	events.h
<code>_os_ev_signal()</code>		Signal event occurrence	events.h

Table 1-10 Interprocess Communication Functions (continued)

Function	ANSI Standard	Description	Header File
<code>_os_ev_tstset()</code>		Wait for event to occur	<code>events.h</code>
<code>_os_ev_unlink()</code>		Unlink event	<code>events.h</code>
<code>_os_ev_wait()</code>		Wait for event to occur	<code>events.h</code>
<code>_os9_ev_wait()</code>		Wait for event to occur	<code>events.h</code>
<code>_os_ev_waitr()</code>		Wait for relative event to occur	<code>events.h</code>
<code>_os9_ev_waitr()</code>		Wait for relative event to occur	<code>events.h</code>
<code>_os_intercept()</code>		Set up signal intercept trap	<code>signal.h</code>
<code>_os9_salarm_atdate()</code>		Execute system-state subroutine at Gregorian date/time	<code>alarm.h</code>
<code>_os_salarm_atime()</code>		Execute system-state subroutine at specified time	<code>alarm.h</code>
<code>_os9_salarm_atjul()</code>		Execute system-state subroutine at Julian date/time	<code>alarm.h</code>
<code>_os_salarm_cycle()</code>		Set cyclic alarm clock	<code>alarm.h</code>

Table 1-10 Interprocess Communication Functions (continued)

Function	ANSI Standard	Description	Header File
<code>_os9_salarm_cycle()</code>		Execute system-state subroutine every n ticks/seconds	alarm.h
<code>_os_salarm_delete()</code>		Remove pending alarm request	alarm.h
<code>_os_salarm_reset()</code>		Reset alarm clock	alarm.h
<code>_os_salarm_set()</code>		Set alarm clock	alarm.h
<code>_os9_salarm_set()</code>		Execute system-state subroutine after specified time interval	alarm.h
<code>_os_sema_init()</code>		Initialize semaphore data structure for use	semaphore.h
<code>_os_sema_p()</code>		Reserve semaphore (acquire exclusive access)	semaphore.h
<code>_os_sema_term()</code>		Terminate use of semaphore data structure	semaphore.h
<code>_os_sema_v()</code>		Release semaphore (release exclusive access)	semaphore.h
<code>_os_send()</code>		Send signal to another process	signal.h

Table 1-10 Interprocess Communication Functions (continued)

Function	ANSI Standard	Description	Header File
<code>_os_siglngj()</code>		Set signal mask value and return on specified stack image	<code>signal.h</code> <code>types.h</code>
<code>_os_sigmask()</code>		Mask/unmask signals during critical code	<code>signal.h</code>
<code>_os_sigreset()</code>		Reset signal intercept context stack	<code>signal.h</code>
<code>_os_sigtrs()</code>		Resize current process' signal queue block	<code>signal.h</code>
<code>pause()</code>		Wait for signal	<code>signal.h</code>
<code>raise()</code>	•	Send signal to program	<code>signal.h</code>
<code>sigmask()</code>		Control process' signal handling	<code>signal.h</code>
<code>signal()</code>	•	Specify signal handling	<code>signal.h</code>

Interrupt Manipulation Functions

The following functions are provided for interrupt manipulation. ANSI prototypes for non-ANSI functions are achieved by defining the macro name `_OPT_PROTOS` during the compile.

Table 1-11 Interrupt Manipulation Functions

Function	ANSI Standard	Description	Header File
<code>irq_change()</code>		Set IRQ level	<code>regs.h</code>
<code>irq_disable()</code>		Mask interrupts	<code>regs.h</code>
<code>irq_enable()</code>		Unmask interrupts	<code>regs.h</code>
<code>irq_maskget()</code>		Mask interrupts; return previous level	<code>regs.h</code>
<code>irq_restore()</code>		Restore interrupt level	<code>regs.h</code>
<code>irq_save()</code>		Return current interrupt level	<code>regs.h</code>
<code>_os_firq()</code>		Add or remove device from fast IRQ table	<code>regs.h</code>
<code>_os_irq()</code>		Add or remove device from IRQ table	<code>regs.h</code>
<code>_os9_irq()</code>		Add or remove device from IRQ table	<code>regs.h</code>
<code>_os_rte()</code>		Return from interrupt exception	<code>signal.h</code>

Mathematical Functions

The following are the available mathematical functions.

Table 1-12 Mathematical Functions

Function	ANSI Standard	Description	Header File
<code>abs()</code>	•	Integer absolute value	<code>stdlib.h</code>
<code>acos()</code>	•	Arc cosine function	<code>math.h</code>
<code>asin()</code>	•	Arc sine function	<code>math.h</code>
<code>atan()</code>	•	Arc tangent function	<code>math.h</code>
<code>atan2()</code>	•	Arc tangent function	<code>math.h</code>
<code>ceil()</code>	•	Ceiling function	<code>math.h</code>
<code>cos()</code>	•	Cosine function	<code>math.h</code>
<code>cosh()</code>	•	Hyperbolic cosine function	<code>math.h</code>
<code>div()</code>	•	Compute quotient and remainder	<code>stdlib.h</code>
<code>exp()</code>	•	Exponential function	<code>math.h</code>
<code>fabs()</code>	•	Floating absolute value	<code>math.h</code>
<code>floor()</code>	•	Floor function	<code>math.h</code>
<code>fmod()</code>	•	Compute floating point remainder	<code>math.h</code>

Table 1-12 Mathematical Functions (continued)

Function	ANSI Standard	Description	Header File
<code>frexp()</code>	•	Return portions of floating point number	<code>math.h</code>
<code>hypot()</code>		Euclidean distance function	<code>math.h</code>
<code>labs()</code>	•	Compute absolute value	<code>stdlib.h</code>
<code>ldexp()</code>	•	Multiply float by exponent of 2	<code>math.h</code>
<code>ldiv()</code>	•	Compute quotient and remainder	<code>stdlib.h</code>
<code>log()</code>	•	Natural logarithm	<code>math.h</code>
<code>log10()</code>	•	Base-ten logarithm	<code>math.h</code>
<code>modf()</code>	•	Return parts of real number	<code>math.h</code>
<code>pow()</code>	•	Power function	<code>math.h</code>
<code>rand()</code>	•	Return random integer	<code>stdlib.h</code>
<code>sin()</code>	•	Sine function	<code>math.h</code>
<code>sinh()</code>	•	Hyperbolic sine function	<code>math.h</code>
<code>sqrt()</code>	•	Square root function	<code>math.h</code>
<code>srand()</code>	•	Set seed for random number generator	<code>stdlib.h</code>
<code>tan()</code>	•	Tangent function	<code>math.h</code>
<code>tanh()</code>	•	Hyperbolic tangent function	<code>math.h</code>

Memory Management Functions

The memory management functions are provided for requesting and freeing memory in a machine and operating system independent manner. ANSI prototypes for non-ANSI functions are achieved by defining the macro name `_OPT_PROTOS` during the compile.

Table 1-13 Memory Management Functions

Function	ANSI Standard	Description	Header File
<code>calloc()</code>	•	Allocate storage for array	<code>stdlib.h</code>
<code>_cpymem()</code>		Copy external memory	<code>process.h</code>
<code>ebrk()</code>		Get external memory	<code>stdlib.h</code>
<code>free()</code>	•	Return memory	<code>stdlib.h</code>
<code>_freemem()</code> , <code>freemem()</code>		Determine size of unused stack area	<code>stdlib.h</code>
<code>_freemin()</code>		Set memory reclamation bound	<code>stdlib.h</code>
<code>ibrk()</code>		Request internal memory	<code>stdlib.h</code>
<code>_lalloc()</code>		Allocate storage for array (low-overhead)	<code>stdlib.h</code>
<code>_lfree()</code>		Return memory (low-overhead)	<code>stdlib.h</code>
<code>_lmalloc()</code>		Allocate memory from arena (low-overhead)	<code>stdlib.h</code>
<code>_lrealloc()</code>		Resize block of memory (low-overhead)	<code>stdlib.h</code>

Table 1-13 Memory Management Functions (continued)

Function	ANSI Standard	Description	Header File
<code>malloc()</code>	•	Allocate memory from arena	<code>stdlib.h</code>
<code>_mallocmin()</code>		Set minimum allocation size	<code>stdlib.h</code>
<code>memchr()</code>	•	Memory search	<code>string.h</code>
<code>memcmp()</code>	•	Compare memory	<code>string.h</code>
<code>memcpy()</code>	•	Copy memory	<code>string.h</code>
<code>memmove()</code>	•	Move memory	<code>string.h</code>
<code>memset()</code>	•	Fill memory	<code>string.h</code>
<code>_os_alltsk()</code>		Allocate task	<code>process.h</code>
<code>_os_chkmem()</code>		Check memory block's accessibility	<code>process.h</code>
<code>_os_cpymem()</code>		Copy external memory	<code>process.h</code>
<code>_os_deltsk()</code>		Deallocate process descriptor	<code>process.h</code>
<code>_os_get_blkmap()</code>		Get free memory block map	<code>memory.h</code>
<code>_os_mem()</code>		Resize data memory area	<code>memory.h</code>
<code>_os_move()</code>		Move data (low bound first)	<code>memory.h</code>

Table 1-13 Memory Management Functions (continued)

Function	ANSI Standard	Description	Header File
<code>_os_permit()</code>		Allow access to memory block	<code>process.h</code>
<code>_os_protect()</code>		Prevent access to memory block	<code>process.h</code>
<code>_os_srqmem()</code>		System memory request	<code>memory.h</code>
<code>_os9_srqmem()</code>		System memory request	<code>memory.h</code>
<code>_os_srtmem()</code>		Return system memory	<code>memory.h</code>
<code>_os_transadd()</code>		Translate memory address (OS-9)	<code>virtual.h</code>
<code>_os9_translate()</code>		Translate memory address (OS-9 for 68K)	<code>memory.h</code>
<code>realloc()</code>	•	Resize block of memory	<code>stdlib.h</code>
<code>sbrk()</code>		Extend data memory segment	<code>stdlib.h</code>
<code>srqcmem()</code>		Allocate colored memory	<code>memory.h</code>
<code>_srqmem()</code>		System memory request	<code>memory.h</code>
<code>_srtmem()</code>		System memory return	<code>memory.h</code>

Shared Data Access Functions

The following two C library functions can be helpful for porting an existing subroutine module. They access two different kinds of data: shared global data and thread-specific data.

The shared global data is automatically shared among all modules that have the same value of `_pthread` (i.e. the application and the subroutine module). The thread-specific data is unique to each thread and is visible to all modules that have the same value of `_pthread`.

The functions described below must be used to access this data.

Table 1-14 Shared Data Access Functions

Function	Description
<code>_pthread_local_slot()</code>	Used when reading or writing thread-specific versions of “core” C run-time variables.
<code>_pthread_global_slot()</code>	Used when reading or writing global versions of “core” C run-time variables.

Miscellaneous Functions

Miscellaneous functions in **Table 1-15** appear in the Ultra C libraries to relieve you from routine tasks. ANSI prototypes for non-ANSI functions are achieved by defining the macro name `_OPT_PROTOS` during the compile.

Table 1-15 Miscellaneous Functions

Function	ANSI Standard	Description	Header File
<code>assert()</code>	•	Place diagnostics in programs	<code>assert.h</code>
<code>change_static()</code>		Change current static storage pointer	<code>regs.h</code>
<code>_cmpnam()</code>		Compare two strings	<code>strings.h</code>
<code>_errmsg()</code>		Print error message	<code>stdio.h</code>
<code>get_static()</code>		Return current static storage pointer	<code>regs.h</code>
<code>getuid()</code>		Determine user ID number	<code>process.h</code>
<code>localeconv()</code>	•	Numeric formatting convention inquiry	<code>locale.h</code>
<code>longjmp()</code>	•	Non-local goto	<code>setjmp.h</code>
<code>_os9_allbit()</code>		Set bits in bit map	<code>memory.h</code>
<code>_os_cache()</code>		Cache control	<code>cache.h</code>
<code>_os_config()</code>		Configure an element	<code>sysglob.h</code>
<code>_os_cmpnam()</code>		Compare two names	<code>strings.h</code>

Table 1-15 Miscellaneous Functions (continued)

Function	ANSI Standard	Description	Header File
<code>_os9_delbit()</code>		Deallocate in bit map	<code>memory.h</code>
<code>_os_dexec()</code>		Execute debugged program	<code>dexec.h</code>
<code>_os_exit()</code>		Exit debugged program	<code>dexec.h</code>
<code>_os9_findpd()</code>		Find fixed block of memory	<code>process.h</code>
<code>_os_getpd()</code>		Find path descriptor	<code>io.h</code>
<code>_os9_panic()</code>		System catastrophic occurrence	<code>process.h</code>
<code>_os_perr()</code>		Print error message	<code>errno.h</code>
<code>_os_prsnam()</code>		Parse path name	<code>strings.h</code>
<code>_os9_retpd()</code>		Return fixed block of memory	<code>process.h</code>
<code>_os_scache()</code>		Cache control	<code>regs.h</code>
<code>_os9_schbit()</code>		Search bit map for free area	<code>memory.h</code>
<code>_os_setsvc()</code>		Service request table initialization	<code>svctbl.h</code>
<code>_os9_setsvc()</code>		Service request table initialization	<code>funcs.h</code>
<code>_os_setuid()</code>		Set user ID number	<code>process.h</code>
<code>_os_strap()</code>		Set error trap handler	<code>settrap.h</code>

Table 1-15 Miscellaneous Functions (continued)

Function	ANSI Standard	Description	Header File
<code>_os9_strap()</code>		Set error trap handler	<code>settrap.h</code>
<code>_os_sysid()</code>		Return system identification	<code>regs.h</code>
<code>_os_uacct()</code>		User accounting	<code>process.h</code>
<code>pause()</code>		Parse disk file (RBF) pathlist	<code>strings.h</code>
<code>perror()</code>	•	Map error number	<code>stdio.h</code>
<code>prerr()</code>		Print error message	<code>stdio.h</code>
<code>_prgname()</code>		Get module name	<code>module.h</code>
<code>_prsnam()</code>		Parse path name segment	<code>strings.h</code>
<code>setjmp()</code>	•	Non-local goto	<code>setjmp.h</code>
<code>setlocale()</code>	•	Locale control	<code>locale.h</code>
<code>setuid()</code>		Set user ID	<code>process.h</code>
<code>_stacksiz(), _stacksiz()</code>		Get size of stack used	<code>stdlib.h</code>
<code>_strass()</code>		Structure assignment	<code>strings.h</code>
<code>tmpnam()</code>	•	Generate unique valid filename	<code>stdio.h</code>
<code>va_arg()</code>	•	Get parameter in variable parameter list	<code>stdarg.h</code>

Table 1-15 Miscellaneous Functions (continued)

Function	ANSI Standard	Description	Header File
<code>va_end()</code>	•	End references to current variable parameter list	<code>stdarg.h</code>
<code>va_start()</code>	•	Initialize variable parameter list	<code>stdarg.h</code>

Module Manipulation Functions

Table 1-16 shows the module manipulation functions. ANSI prototypes for non-ANSI functions are achieved by defining the macro name `_OPT_PROTOS` during the compile.

Table 1-16 Module Manipulation Functions

Function	ANSI Standard	Description	Header File
<code>crc()</code>		Calculate module CRC	<code>module.h</code>
<code>_get_module_dir()</code>		Get module directory entry	<code>module.h</code>
<code>make_module()</code>		Create module	<code>module.h</code>
<code>_mkdata_module()</code>		Create data memory module	<code>module.h</code>
<code>modcload()</code>		Load module into colored memory	<code>module.h</code>
<code>modlink()</code>		Link to memory module	<code>module.h</code>

Table 1-16 Module Manipulation Functions (continued)

Function	ANSI Standard	Description	Header File
<code>modload()</code>		Load and link to memory module	<code>module.h</code>
<code>modloadp()</code>		Load and link to memory module using PATH	<code>module.h</code>
<code>munlink()</code>		Unlink from module	<code>module.h</code>
<code>munload()</code>		Unload module	<code>module.h</code>
<code>_os_altdir()</code>		Set alternate working module directory	<code>moddir.h</code>
<code>_os_chainm()</code>		Execute new primary module given pointer to module	<code>process.h</code>
<code>_os_chmdir()</code>		Change process' current module directory	<code>moddir.h</code>
<code>_os_cmdperm()</code>		Change permissions of module directory	<code>moddir.h</code>
<code>_os_crc()</code>		Generate CRC	<code>module.h</code>
<code>_os_datmod()</code>		Create data module	<code>module.h</code>
<code>_os_delmdir()</code>		Delete existing module directory	<code>moddir.h</code>
<code>_os_fmod()</code>		Find module directory entry	<code>module.h</code>

Table 1-16 Module Manipulation Functions (continued)

Function	ANSI Standard	Description	Header File
<code>_os_get_mdp()</code>		Get current and alternate module directory pathlists	<code>moddir.h</code>
<code>_os_get_moddir()</code>		Get copy of module directory	<code>module.h</code>
<code>_os_initdata()</code>		Initialize static storage from module	<code>module.h</code>
<code>_os_link()</code>		Link to memory module	<code>module.h</code>
<code>_os_linkm()</code>		Link to memory module by module pointer	<code>module.h</code>
<code>_os_load()</code>		Load module(s) from file	<code>module.h</code>
<code>_os_loadp()</code>		Load and link to a memory module using PATH	<code>module.h</code>
<code>_os_mkdir()</code>		Create new module directory	<code>moddir.h</code>
<code>_os_mkmodule()</code>		Create module of specified color type	<code>module.h</code>
<code>_os_modaddr()</code>		Find module given pointer	<code>module.h</code>
<code>_os_setcrc()</code>		Generate valid CRC in module	<code>module.h</code>
<code>_os_slink()</code>		Install subroutine module	<code>module.h</code>

Table 1-16 Module Manipulation Functions (continued)

Function	ANSI Standard	Description	Header File
<code>_os_slink()</code>		Install user subroutine library module by module pointer	module.h
<code>_os_tlink()</code>		Install system state trap handling module	module.h
<code>_os_tlinkm()</code>		Install user trap handling module by module pointer	module.h
<code>_os_unlink()</code>		Unlink module by address	module.h
<code>_os_unload()</code>		Unlink module by name	module.h
<code>_os_vmodul()</code>		Verify module	module.h
<code>_os9_vmodul()</code>		Validate module	module.h
<code>_setcrc()</code>		Re-validate module CRC	module.h
<code>_subcall()</code>		Call subroutine module	module.h

Multibyte Character and String Functions

Five functions, identified in **Table 1-17**, are available for handling multibyte characters and strings.

Table 1-17 Multi-byte Character/String Functions

Function	ANSI Standard	Description	Header File
<code>mblen()</code>	•	Determine number of bytes in multibyte character	stdlib.h
<code>mbstowcs()</code>	•	Convert sequence of multibyte characters	stdlib.h
<code>mbtowc()</code>	•	Determine number of bytes in multibyte characters	stdlib.h
<code>wcstombs()</code>	•	Convert sequence of wide characters to multibyte characters	stdlib.h
<code>wctomb()</code>	•	Convert wide character to multibyte character	stdlib.h

OS System Functions

The standard library provides a large number of functions directly accessing system calls. Functions available to provide access to selected system calls are identified in **Table 1-18**. ANSI prototypes for non-ANSI functions are achieved by defining the macro name _OPT_PROTO during the compile.

Table 1-18 OS System Functions

Function	ANSI Standard	Description	Header File
_getsys()		Get system global variables	setsys.h
_os_sysdbg()		Call system debugger	process.h
_setsys()		Set/examine system global variables	setsys.h
_sysdbg()		Call system debugger	process.h

POSIX Messaging Functions

The OS-9 implementation of the POSIX specification supports a message queue that is either visible between process or between threads. Currently, only the threaded message queue is supported. You distinguish which type you wish to create by the message queue name passed to `mq_open`. If the name starts with a '/', then a process wide message queue is assumed. Otherwise, a threaded message queue is created.

There is a thread and non-thread version of the POSIX message library. At this time, OS-9 does not support process-wide message queues. Therefore, the non-threaded version of the library is merely a placeholder until the process-wide message queues are implemented. The functions all return `ENOSYS`—the POSIX error code that means not implemented.

The supported POSIX messaging functions are listed in **Table 1-19**.

Table 1-19 POSIX Messaging Functions

Function	POSIX Standard	Description	Header File
<code>mq_close</code>	•	Close a Message Queue	<code>mqueue.h</code>
<code>mq_getattr</code>	•	Get Message Queue Attributes	<code>mqueue.h</code>
<code>mq_notify</code>	•	Notify Process That a Message is Available on a Queue	<code>mqueue.h</code>
<code>_mq_notify_write</code>		Notify Process That a Space in the Message Queue is Available	<code>mqueue.h</code>
<code>mq_open</code>	•	Open a Message Queue	<code>mqueue.h</code>

Table 1-19 POSIX Messaging Functions (continued)

Function	POSIX Standard	Description	Header File
<code>mq_receive</code>	•	Receive a Message From a Message Queue	<code>mqueue.h</code>
<code>mq_send</code>	•	Send a Message to a Message Queue	<code>mqueue.h</code>
<code>mq_setattr</code>	•	Set Message Queue Attributes	<code>mqueue.h</code>
<code>mq_unlink</code>	•	Remove a Message Queue	<code>mqueue.h</code>

Process Manipulation Functions

The standard library provides functions for manipulating processes as identified in **Table 1-20**. ANSI prototypes for non-ANSI functions are achieved by defining the macro name _OPT_PROTOS during the compile.

Table 1-20 Process Manipulation Functions

Function	ANSI Standard	Description	Header File
<code>chainc()</code> , <code>chain()</code>		Load and execute new module	<code>process.h</code>
<code>getpid()</code>		Determine process ID number	<code>process.h</code>
<code>_get_process_desc()</code>		Get process descriptor copy	<code>process.h</code>
<code>_get_process_table()</code>		Get process table entry	<code>process.h</code>
<code>kill()</code>		Send signal to process	<code>signal.h</code>
<code>_os9_allpd()</code>		Allocate fixed-length block of memory	<code>process.h</code>
<code>_os_allocproc()</code>		Allocate process descriptor	<code>process.h</code>
<code>_os_aproc()</code>		Insert process in active process queue	<code>process.h</code>
<code>_os9_aproc()</code>		Enter process in active process queue	<code>process.h</code>

Table 1-20 Process Manipulation Functions (continued)

Function	ANSI Standard	Description	Header File
<code>_os_chain()</code>		Execute new primary module	<code>process.h</code>
<code>_os_dfork()</code>		Fork process under control of debugger	<code>dexec.h</code>
<code>_os9_dfork()</code>		Fork process under control of debugger	<code>dexec.h</code>
<code>_os_dforkm()</code>		Fork process under control of debugger	<code>dexec.h</code>
<code>_os_exec()</code>		Start child process	<code>process.h</code>
<code>_os_findpd()</code>		Find process descriptor	<code>process.h</code>
<code>_os_fork()</code>		Create new process	<code>process.h</code>
<code>_os_forkm()</code>		Create new process by module pointer	<code>process.h</code>
<code>_os_get_prtbl()</code>		Get copy of process descriptor block table	<code>process.h</code>
<code>_os_getstat()</code>		Get file/device status	<code>sg_codes.h</code>
<code>_os_getsys()</code>		Examine system global variable	<code>sysglob.h</code>
<code>_os_gprdsc()</code>		Get process descriptor copy	<code>process.h</code>
<code>_os9_gspump()</code>		Return data for specified process' memory map	<code>process.h</code>

Table 1-20 Process Manipulation Functions (continued)

Function	ANSI Standard	Description	Header File
<code>_os_id()</code>		Get process ID/user ID	process.h
<code>_os9_id()</code>		Get process ID/user ID	process.h
<code>_os_nproc()</code>		Start next process	process.h
<code>_os_rtnprc()</code>		Deallocate process descriptor	process.h
<code>_os_setpr()</code>		Set process priority	process.h
<code>_os_sleep()</code>		Put calling process to sleep	signal.h
<code>_os9_sleep()</code>		Put calling process to sleep	signal.h
<code>_os_suspend()</code>		Suspend process	process.h
<code>_os_wait()</code>		Wait for child process to terminate	process.h
<code>os9exec()</code>		OS-9 for 68K system call processing	process.h
<code>os9fork()</code> , <code>os9forkc()</code>		Create process	process.h
<code>os9kexec()</code>		OS-9 create process	types.h
<code>setpr()</code>		Set process priority	process.h

Table 1-20 Process Manipulation Functions (continued)

Function	ANSI Standard	Description	Header File
<code>sleep()</code>		Suspend execution for specified time	<code>signal.h</code>
<code>tsleep()</code>		Sleep for specified interval	<code>signal.h</code>
<code>wait()</code>		Wait for process termination	<code>process.h</code>

Resource Locks

On OS-9 for 68K systems, the functions identified in **Table 1-21** are available for use with resource locks. For more information about resource locks, refer to the **OS-9 for 68K Technical Manual**.

Table 1-21 Resource Lock Functions

Function	ANSI Standard	Description	Header File
<code>_os_acqlk()</code>		Acquire ownership of resource lock	<code>lock.h</code>
<code>_os_caqlk()</code>		Conditionally acquire ownership of resource lock	<code>lock.h</code>
<code>_os_crlk()</code>		Create new resource lock descriptor	<code>lock.h</code>
<code>_os_ddlk()</code>		Check for deadlock situation	<code>process.h</code>
<code>_os_dellk()</code>		Delete existing lock descriptor	<code>lock.h</code>

Table 1-21 Resource Lock Functions (continued)

Function	ANSI Standard	Description	Header File
<code>_os_rellk()</code>		Release ownership of resource lock	<code>lock.h</code>
<code>_os_waitlk()</code>		Activate next process waiting to acquire lock	<code>lock.h</code>

Searching and Sorting Functions

Functions included for searching and sorting are identified in **Table 1-22**.

Table 1-22 Search and Sort Functions

Function	ANSI Standard	Description	Header File
<code>bsearch()</code>	•	Search array	<code>stdlib.h</code>
<code>qsort()</code>	•	Quick sort	<code>stdlib.h</code>

String Handling Functions

The C language does not have a character string data type. Instead, it stores strings as character arrays and the standard library provides functions to manipulate them. The functions are identified in [Table 1-23](#). ANSI prototypes for non-ANSI functions are achieved by defining the macro name _OPT_PROTOS during the compile.

Table 1-23 String Handling Functions

Function	ANSI Standard	Description	Header File
<code>findnstr()</code>		Search string for pattern	<code>strings.h</code>
<code>findstr()</code>		Search string for pattern	<code>strings.h</code>
<code>index()</code>		Search for character in string	<code>strings.h</code>
<code>rindex()</code>		Search for character in string	<code>strings.h</code>
<code>strcat()</code>	•	String catenation	<code>string.h</code>
<code>strchr()</code>	•	Locate string	<code>string.h</code>
<code>strcmp()</code>	•	String comparison	<code>string.h</code>
<code>strcoll()</code>	•	String comparison	<code>string.h</code>
<code>strcpy()</code>	•	String copy	<code>string.h</code>
<code>strcspn()</code>	•	Get string length	<code>string.h</code>
<code>strerror()</code>	•	Map error message string	<code>string.h</code>
<code>strftime()</code>	•	Place formatted time in buffer	<code>time.h</code>
<code>strhcpy()</code>		Copy old OS-9 for 68K strings	<code>strings.h</code>

Table 1-23 String Handling Functions (continued)

Function	ANSI Standard	Description	Header File
<code>strlen()</code>	•	Determine string length	<code>string.h</code>
<code>strncat()</code>	•	String catenation	<code>string.h</code>
<code>strcmp()</code>	•	String comparison	<code>string.h</code>
<code>strcpy()</code>	•	String copy	<code>string.h</code>
<code>strpbrk()</code>	•	Locate first occurrence of string	<code>string.h</code>
<code>strrchr()</code>	•	Locate last occurrence of string	<code>string.h</code>
<code>strspn()</code>	•	Compute string length	<code>string.h</code>
<code>strstr()</code>	•	Locate first occurrence of string	<code>string.h</code>
<code>strtod()</code>	•	String to double conversion	<code>stdlib.h</code>
<code>strtok()</code>	•	Break string into tokens	<code>string.h</code>
<code> strtol()</code>	•	String to long conversion	<code>stdlib.h</code>
<code> strtoul()</code>	•	String to unsigned long conversion	<code>stdlib.h</code>
<code>strxfrm()</code>	•	Transform string	<code>string.h</code>

Terminal Manipulation Functions (Termcap)

Table 1-24 Termcap Functions

Function	ANSI Standard	Description	Header File
<code>tgetent()</code>		Get termcap entries	<code>termcap.h</code>
<code>tgetflag()</code>		Check terminal capability presence	<code>termcap.h</code>
<code>tgetnum()</code>		Get terminal capability ID	<code>termcap.h</code>
<code>tgetstr()</code>		Get terminal capability	<code>termcap.h</code>
<code>tgoto()</code>		Get cursor movement capability	<code>termcap.h</code>
<code>tputs()</code>		Output capability string	<code>termcap.h</code>

Time Functions

The TZ environment variable is used to specify the time zone for the C functions to use. TZ should have the following format:

`zzz[+/-n][:ddd]`

`zzz` is one of the supported time zone names listed in [Table 1-25](#).

Table 1-25 Time Zones and Descriptions

Zone	Description
GMT/UTC	Greenwich Mean Time (or Coordinated Universal Time)
PST/PDT	USA Pacific Standard Time/Daylight Savings Time
MST/MDT	USA Mountain Standard Time/Daylight Savings Time
CST/CDT	USA Central Standard Time/Daylight Savings Time
EST/EDT	USA Eastern Standard Time/Daylight Savings Time
YST	Yukon Standard Time (Most of Alaska)
AST	Aleutian/Hawaiian Standard Time
EET	Eastern European Time
CET	Central European Time
WET	Western European Time
JST	Japan Standard Time
MIT	Midway Islands Time
HST	Hawaii Standard Time

Table 1-25 Time Zones and Descriptions (continued)

Zone	Description
PNT	Phoenix Standard Time
IET	Indiana Eastern Standard Time
PRT	Puerto Rico Standard Time
CNT	Canada Newfoundland Time
AGT	Argentina Standard Time
BEZ	Brazilian Standard Time
CAT	Central Africa
ECT	European Central Time
ART	Arabic Standard Time
EAT	Eastern African Time
MET	Middle Eastern Time
NET	Near East Time
PLT	Pakistan Lahore Time
IST	India Standard Time
BST	Bangladesh Standard Time
VST	Vietnam Standard Time
CTT	China Taiwan Standard Time

Table 1-25 Time Zones and Descriptions (continued)

Zone	Description
ACT	Australia Central Time
AET	Australian Eastern Time
SST	Solomon Standard Time
NST	New Zealand Standard Time

n is the optional number of minutes east (+) or west (-) of the time zone.

ddd is the option controlling Daylight Savings Time.

Recognized codes for ddd are shown in **Table 1-26**.

Table 1-26 Valid ddd Codes

Value	Description
no	Do not use Daylight Savings Time
usa	Conforms to the US Uniform Time Act of 1967 and its various amendments through 1987
eur	Observe European Daylight Savings Time

For example, the following command specifies that the compiler should use the USA Eastern Daylight Savings Time and should conform to the US Uniform Time Act of 1967:

```
setenv TZ EDT:usa
```

The American time zones default to usa, with the exception of AST which defaults to no. The European time zones default to eur and GMT/UTC defaults to no.

If the `TZ` environment variable is not set and, if on OS-9, `m_tmzone` is not set, `time()` uses USA Central Standard Time (CST).

In addition, the time functions, with the exception of `clock()`, can be used in both system and user state. However, to use the functions in system state, you will need to define `_environ` must be defined as follows:

```
void * xenviron;
void **_environ = &xenviron
```

Ultra C provides time functions identified in **Table 1-27**. ANSI prototypes for non-ANSI functions are achieved by defining the macro name `_OPT_PROTO` during the compile.

Table 1-27 Time Functions

Function	ANSI Standard	Description	Header File
<code>asctime()</code>	•	Convert broken-down time to string format	<code>time.h</code>
<code>clock()</code>	•	Get processor time	<code>time.h</code>
<code>ctime()</code>	•	Convert calendar time to string format	<code>time.h</code>
<code>difftime()</code>	•	Find temporal difference	<code>time.h</code>
<code>getime()</code>	•	Get system time	<code>time.h</code>
<code>gmtime()</code>	•	Convert calendar time to Greenwich Mean Time	<code>time.h</code>
<code>_gregorian()</code>		Convert date/time to Gregorian value	<code>time.h</code>
<code>_julian()</code>		Convert date/time to Julian value	<code>time.h</code>

Table 1-27 Time Functions (continued)

Function	ANSI Standard	Description	Header File
<code>localtime()</code>	•	Convert calendar time to local time	<code>time.h</code>
<code>mktime()</code>	•	Convert broken-down time to calendar time	<code>time.h</code>
<code>_os_gettime()</code>		Get system date and time	<code>time.h</code>
<code>_os9_gettime()</code>		Get system date/time	<code>time.h</code>
<code>_os_gregorian()</code>		Get Gregorian date	<code>time.h</code>
<code>_os_julian()</code>		Get Julian date	<code>time.h</code>
<code>_os_setime()</code>		Set system date/time	<code>time.h</code>
<code>_os9_setime()</code>		Set system date/time	<code>time.h</code>
<code>setime()</code>		Set system time	<code>time.h</code>
<code>strftime()</code>	•	Place formatted time in buffer	<code>time.h</code>
<code>sys_mktime()</code>		Convert local time to Greenwich Mean Time	<code>time.h</code>
<code>_sysdate()</code>		Get current system date/time	<code>time.h</code>
<code>time()</code>	•	Get calendar time	<code>time.h</code>

Processor-Specific Functions

Functions available only on the 80x86 family of systems are identified in **Table 1-28**. ANSI prototypes for non-ANSI functions are achieved by defining the macro name _OPT_PROTOS during the compile.

Table 1-28 80x86-Only Functions

Function	ANSI Standard	Description	Header File
<code>get_current_tss()</code>		Get current task segment	<code>regs.h</code>
<code>get_excpt_base()</code>		Return exception table base address	<code>regs.h</code>
<code>get_gdtr()</code>		Get global descriptor pointer	<code>regs.h</code>
<code>make_gdesc()</code>		Make global descriptor table entry	<code>regs.h</code>
<code>make_idesc()</code>		Make interrupt descriptor table entry	<code>regs.h</code>
<code>set_excpt_base()</code>		Set exception table base address	<code>regs.h</code>
<code>set_gdtr()</code>		Set global descriptor pointer	<code>regs.h</code>

These functions are available only on the PowerPC processors as identified in **Table 1-29**. ANSI prototypes for non-ANSI functions are achieved by defining the macro name _OPT_PROTOS during the compile.

Table 1-29 Power PC-Only Functions

Function	ANSI Standard	Description	Header File
<code>_get_<name>()</code>		Read the value of SPRs, DCRs, PREGs, and TBs	getset.h
<code>_set_<name>()</code>		Set the value of SPRs, DCRs, PREGs, and TBs	getset.h

Implementation-Defined Behavior

The following are the implementation-defined issues pertaining to the Ultra C library. Each bulleted item contains one implementation-defined issue. The number in parentheses included with each bulleted item indicates the location in the ANSI specification where you can find more information.

- The null pointer constant to which the macro NULL expands (4.1.5).
NULL expands to `(void *) 0`.

- The diagnostic printed by and the termination behavior of the `assert()` function (4.2).

`assert()` prints a line with the following syntax (if the `NDEBUG` macro is not defined) if the assertion fails:

Assertion failed:<expression>, file <file>, line <line>\n

<expression> is the expression being tested.

<file> is the file name of the file containing the `assert` macro.

<line> is the line number of the file containing the `assert` macro.

The `abort()` function is called after the assertion failure line is printed. This raises the `SIGABRT` signal that may be handled by the application.



For More Information

Refer to the description of `signal()` for more information about `SIGABRT`.

- The sets of characters tested for by the `isalnum()`, `isalpha()`, `iscntrl()`, `islower()`, `isprint()`, and `isupper()` functions (4.3.1).

`isalnum()` returns non-zero for any character which `isdigit()` or `isalpha()` returns non-zero.

`isalpha()` returns non-zero for any character which `islower()` or `isupper()` returns non-zero.

`iscntrl()` returns non-zero for ASCII characters with values from 0 to 0x1f and 0x7f.

`islower()` returns non-zero for ASCII characters with values from 0x61 to 0x7a.

`isprint()` returns non-zero for ASCII characters with values from 0x20 to 0x7e.

`isupper()` returns non-zero for ASCII characters with values from 0x41 to 0x5a.

- The values returned by the mathematics functions on domain errors (4.5.1).

The mathematics functions return `HUGE_VAL` on domain errors.

- Whether the mathematics functions set the integer expression `errno` to the value of the macro `ERANGE` on underflow range errors (4.5.1).
`errno` is not set to `ERANGE` on underflow errors.
- Whether a domain error occurs or zero is returned when the `fmod()` function has a second argument of zero (4.5.6.4).
`fmod()` returns zero if it has a second argument of zero.
- The set of signals for the `signal()` function (4.7.1.1).
The set of signals for the signal function varies with the operating system and the processor. A set of signals is also available regardless of operating system.

ANSI “Core” Signals

SIGABRT	SIGFPE	SIGILL	SIGINT
SIGSEGV	SIGTERM		

OS-9 “Core” Signals

SIGKILL	SIGWAKE	SIGQUIT	SIGHUP
SIGALRM	SIGPIPE	SIGUSR1	SIGUSR2

OS-9 68K Family Signals

SIGADDR	SIGCHK	SIGTRAPV	SIGPRIV
SIGTRACE	SIG1010	SIG1111	

OS-9 80x86 Family Signals

SIGGPROT	SIGSTACK	SIGSEGNP	SIGINVTSS
SIGDBLFLT	SIGBNDCHK	SIGBRKPT	SIGNMI
SIGDBG			

OS-9 PowerPC™ Family Signals

SIGCHK

SIGINST

SIGPRIV

SIGALIGN

- The semantics for each signal recognized by the [signal\(\)](#) function (4.7.1.1).



For More Information

For information related to the exceptions, see the reference manual for your specific processor supplied by the manufacturer.

Table 1-30 Signals

Signal	Condition
SIGABRT	Abnormal termination such as initiated by the abort() function.
SIGFPE	An erroneous arithmetic operation such as zero divide or an operation resulting in overflow. This signal is generated on target-specific conditions.
SIGILL	Detection of an invalid function image such as illegal instruction. This signal is generated on target-specific conditions.
SIGINT	Receipt of an interactive attention signal generated by the keyboard interrupt character on OS-9 for 68K and OS-9.
SIGSEGV	An invalid access to storage. This signal is generated on target-specific conditions.
SIGTERM	A termination request sent to the program.

Table 1-31 More Signals

Signal	Condition
SIGKILL	A death request sent to the program. This signal cannot be ignored or handled.
SIGWAKE	A wakeup request sent to the program. If a process is active when this signal arrives, it is ignored. Otherwise, the process is activated, but the signal handling function is not called.
SIGQUIT	Receipt of an interactive abort signal generated by the keyboard quit character on OS-9.
SIGHUP	Modem hangup signal. This signal is sent to a process when a modem connection is lost.
SIGALRM	Not automatically sent. Programs can use this signal for an alarm call.
SIGPIPE	Not automatically sent.
SIGUSR1	Condition is user defined.
SIGUSR2	Condition is user defined.

Processor-Specific Family Signals



For More Information

For information related to the exceptions, see the reference manual for your specific processor.

68K Family Processors

The OS-9 68K family signals sent when their associated exception occurs are identified in **Table 1-32**.

Table 1-32 OS-9 68K Family Exception Signals

Signal	Exception
SIGADDR	Address Error
SIGCHK	CHK Instruction
SIGTRAPV	TRAPV Instruction
SIGPRIV	Privilege Violation
SIGTRACE	Trace
SIG1010	Line 1010 Emulation
SIG1111	Line 1111 Emulation

80x86 Family Processors

The OS-9 80x86 family signals shown in **Table 1-33** are sent when their associated exception occurs.

Table 1-33 OS-9 80x86 Family Exception Signals

Signal	Exception
SIGGPROT	General Protection
SIGSTACK	Stack Exception
SIGSEGNP	Segment Not Present
SIGINVVTSS	Invalid TSS
SIGDBLFLT	Double Fault
SIGBNDCHK	Bounds Check
SIGBRKPT	Breakpoint
SIGNMI	Non-Maskable Interrupt
SIGDBG	Debug Exceptions

PowerPC Family Processors

The OS-9 PowerPC family signals shown in **Table 1-34** are sent when their associated exception occurs.

Table 1-34 OS-9 PowerPC Family Exception Signals

Signal	Exception
SIGCHECK	Machine Check
SIGINST	Instruction Access

Table 1-34 OS-9 PowerPC Family Exception Signals

Signal	Exception
SIGPRIV	Privilege Violation
SIGALIGN	Alignment

ARM Family Processors

The OS-9 ARM family signal shown in **Table 1-35** is sent when its associated exception occurs.

Table 1-35 OS-9 ARM Family Exception Signals

Signal	Exception
SIGALIGN	Alignment

SH Family Processors

The OS-9 SH3 family signal shown in **Table 1-36** is sent when the associated exception occurs.

Table 1-36 OS-9 SH3 Family Exception Signals

Signal	Exception
SIGALIGN	Alignment

SPARC Family Processors

The OS-9 SPARC family signals shown in [Table 1-37](#) are sent when their associated exception occurs.

Table 1-37 OS-9 SPARC Family Exception Signals

Signal	Exception
SIGALIGN	Alignment
SIGWINDOWOV	Window Overflow
SIGWINDOWUV	Window Underflow
SIGATAGOV	Tag Overflow
SIGCPE	Coprocessor Exception

- The default handling and the handling at program startup for each signal recognized by the `signal()` function (4.7.1.1).
For each signal recognized by `signal()`, the default handling is program termination and the program start-up condition is `SIG_DFL`. The program terminates with an exit status related to the exception or the signal number for non-exception related signals.
- If the equivalent of `signal(sig, SIG_DFL)`; is not executed prior to the call of the signal handler, the blocking of the signal that is performed (4.7.1.1).
`signal (sig, SIG_DFL)` is executed before the call to the signal handler. Signals are blocked for the duration of the signal handler, unless the program takes some action that unblocks `signal()`. Refer to your operating system technical manual for more information.
- Whether the default handling is reset if the `SIGILL` signal is received by a handler specified to the `signal` function (4.7.1.1).
The default handling is reset on all signals.

- Whether the last line of a text stream requires a terminating newline character (4.9.2).

A terminating newline is not required as the last character of a text stream.
- Whether space characters that are written out to a text stream immediately before a newline character appear when read in (4.9.2).

Space characters written to a text stream immediately before a newline character appears when read.
- The number of null characters that may be appended to data written to a binary stream (4.9.2).

Null characters are not appended to data written to a binary stream.
- Whether the file position indicator of an append mode stream is initially positioned at the beginning or end of the file (4.9.3).

The file position indicator of an append mode stream is initially positioned at the beginning of the file. Regardless, all writes to a stream in append mode are started at the current end-of-file.
- Whether a write on a text stream causes the associated file to be truncated beyond that point (4.9.3).

No truncation of this sort occurs.
- The characteristics of file buffering (4.9.3).

The file buffering decision is made when the first request is made to the stream. If the buffering type and size have not been set explicitly by `setvbuf()`, the determination is made based on the device with which the stream is associated. If the device is interactive, line buffering is used. Otherwise, full buffering is used. If the device is RBF (Random Block File Manager), the buffer size is $2 * \text{current_sector_size}$.
- Whether a zero-length file actually exists (4.9.3).

A zero length file actually does exist.
- The rules for composing valid file names (4.9.3).

The rules vary depending on the operating system:

OS-9 for 68K	File names can contain 1 to 28 upper or lower case letters, digits, underscores (_), periods (.), or dollar signs (\$).
OS-9	File names can contain 1 to 43 upper or lower case letters, digits, underscores (_), periods (.), or dollar signs (\$).

- Whether the same file can be open multiple times (4.9.3).

A program can open a file more than once, but fully buffered reading and writing to the same file has undefined results.
- The effect of the `remove()` function on an open file (4.9.4.1).

`remove()` returns -1 with `errno` set to `EOS_SHARE` if called with the name of an open file.
- The effect if a file with the new name exists prior to a call to the `rename()` function (4.9.4.2).

Assuming the calling program has write permission, the file with the new name is deleted and recreated with the new information.
- The output for `%p` conversion in the `fprintf()` function (4.9.6.1).

The pointer to `void` is printed as an unsigned hexadecimal integer.
- The input for `%p` conversion in the `fscanf()` function (4.9.6.2).

The input for `%p` conversion is an unsigned hexadecimal integer.
- The interpretation of a ‘-‘ character that is neither the first nor the last character in the scanlist for `%[` conversion in the `fscanf()` function (4.9.6.2).

A hyphen (-) character is interpreted simply as the addition of – to the scanlist.
- The value to which the macro `errno` is set by the `fgetpos()` or `ftell()` function on failure (4.9.9.1, 4.9.9.4).

`errno` is set to the operating system specific error that is returned by either `_os_gs_size()` (in append mode) or `_os_gs_pos()`.
- The messages generated by the `perror()` function (4.9.10.4).

The messages generated by the `perror()` function are located in the system `errmsg` file (`/dd/SYS/errmsg`, `/h0/SYS/errmsg`, or `/d0/SYS/errmsg`). Refer to your operating system technical manual for detailed information. Error messages are generally in the form:

```
<message>: #<major>:<minor> <detail>
```

`<message>` is the user-passed message string.
`<major>` is the family of error.
`<minor>` is the error number within the family.
`<detail>` is the information from the system `errmsg` file (if available).

- The behavior of the `calloc()`, `malloc()`, or `realloc()` function if the size requested is zero (4.10.3).

The `calloc()`, `malloc()`, and `realloc()` functions return `NULL` if the requested size is zero.

- The behavior of the `abort()` function with regard to open and temporary files (4.10.4.1).

If the `SIGABRT` signal handler returns, the program is terminated abnormally flushing open streams, closing open streams, and removing temporary files.

- The status returned by the `exit()` function if the value of the argument is other than zero, `EXIT_SUCCESS`, or `EXIT_FAILURE` (4.10.4.3).

The integer value passed to `exit()` is the exit status of the program.

- The set of environment names and the method for altering the environment list used by the `getenv()` function (4.10.4.3).

The common set of environment names depends on the operating system. Refer to your operating system's user manual for more information.

The environment list is implemented as a `NULL` terminated array of pointers to characters. Each string is in the form:

`<name>=<value>`

`<name>` is the name of the environment variable.

`<value>` is the current value of the environment variable.

To modify the list, you must either find the existing entry or enlarge the array and add an entry. The base of the array is stored in the external variable `_environ`.

- The contents and mode of execution of the string by the `system()` function (4.10.4.5).

The contents of an execution string given to `system()` can be any valid shell command line. The shell used to execute the string is either the current value of the `SHELL` environment variable or `shell` if `SHELL` is unavailable.

- The contents of the error message strings returned by the `strerror()` function (4.11.6.2).

The strings generated by the `strerror()` function are located in the system `errmsg` file (`/dd/SYS/errmsg`, `/h0/SYS/errmsg`, or `/d0/SYS/errmsg`). Refer to your operating system technical manual for detailed information. Error messages are generally in the form:

```
#<major>:<minor> <detail>
```

<major> is the family of error.

<minor> is the error number within the family.

<detail> is the information from the system errmsg file (if available).

- The local time zone and Daylight Savings Time (4.12.1).

The local time zone and Daylight Savings Time are determined by the TZ environment variable on OS-9 for 68K and the TZ variable or the m_tmzone field in the init module on OS-9. Failing to find user defined values, the defaults are Central Standard Time and Central Daylight Time.

- The era for the `clock()` function (4.12.2.1).

The era for `clock()` is the elapsed ticks since the program was forked.

Equivalents for the sys_clib.l Functions

The functions in the `sys_clib.l` library exist mainly for compatibility with the Microware K & R C compiler, although the calls are accepted in all modes. Whenever possible, you should use the functions in the `os_lib.l` and `clib.l` libraries that have the same functionality. The following is a list of the `sys_clib.l` functions and the corresponding calls in either `os_lib.l` or `clib.l`.

Table 1-38 sys_clib.l Function Equivalents

sys_clib.l Function	Corresponding Function
<code>access()</code>	<code>_os_open()</code>
<code>create()</code>	<code>_os_create()</code>
<code>alm_atdate()</code>	<code>_os_alarm_atime() /</code> <code>_os9_alarm_atdate()</code>
<code>detach()</code>	<code>_os_detach()</code>

Table 1-38 sys_clib.l Function Equivalents (continued)

sys_clib.l Function	Corresponding Function
alm_atjul()	_os_alarm_atime() / _os9_alarm_atjul()
dup()	_os_dup()
alm_cycle()	_os_alarm_cycle()
ebrk()	malloc()
alm_delete()	_os_alarm_delete()
_errmsg()	strerror()
alm_set()	_os_alarm_set()
_ev_create()	_os_ev_create()
attach()	_os_attach()
_ev_delete()	_os_ev_delete()
_ev_info()	_os_ev_info()
chainc(), chain()	_os_chain()
_ev_link()	_os_ev_link()
chdir()	_os_chdir()
_ev_pulse()	_os_ev_pulse()
chmod()	_os_ss_attr()
_ev_read()	_os_ev_read()
chown()	_os_ss_fd()

Table 1-38 sys_clib.l Function Equivalents (continued)

sys_clib.l Function	Corresponding Function
<code>_ev_set()</code>	<code>_os_ev_set()</code>
<code>chkdir()</code>	<code>_os_chdir()</code>
<code>_ev_setr()</code>	<code>_os_ev_setr()</code>
<code>_cleareof(), cleareof()</code>	<code>clearerr()</code>
<code>_ev_signal()</code>	<code>_os_ev_signal()</code>
<code>_ev_unlink()</code>	<code>_os_ev_unlink()</code>
<code>close()</code>	<code>_os_close()</code>
<code>_ev_wait()</code>	<code>_os_ev_wait() / _os9_ev_wait()</code>
<code>closedir()</code>	<code>_os_close()</code>
<code>_ev_waitr()</code>	<code>_os_ev_waitr() / _os9_ev_waitr()</code>
<code>_cmpnam()</code>	<code>_os_cmpnam()</code>
<code>_exit()</code>	<code>_os_exit()</code>
<code>_cpymem()</code>	<code>_os_cpymem()</code>
<code>fdopen()</code>	none
<code>crc()</code>	<code>_os_crc()</code>
<code>findnstr()</code>	<code>strstr()</code>
<code>creat()</code>	<code>_os_create()</code>
<code>findstr()</code>	<code>strstr()</code>

Table 1-38 sys_clib.l Function Equivalents (continued)

sys_clib.l Function	Corresponding Function
<code>_get_module_dir()</code>	<code>_os_get_moddir()</code>
<code>makdir()</code>	<code>_os_mkdir()</code>
<code>_get_process_desc()</code>	<code>_os_gprdesc()</code>
<code>make_module()</code>	<code>_os_mkmodule()</code>
<code>_get_process_table()</code>	<code>_os_get_prtbl()</code>
<code>_mkdata_module()</code>	<code>_os_datmod()</code>
<code>mkttime()</code>	<code>_os_gettime() / _os9_gettime()</code>
<code>mknod()</code>	<code>_os_mkdir()</code>
<code>getpid()</code>	<code>_os_id()</code>
<code>mktemp()</code>	<code>tmpnam()</code>
<code>getstat()</code>	<code>_os_getstat()</code>
<code>modcload()</code>	<code>_os_loadap()</code>
<code>_getsys()</code>	<code>_os_getsys()</code>
<code>modlink()</code>	<code>_os_link()</code>
<code>getuid()</code>	<code>_os_id()</code>
<code>modload()</code>	<code>_os_load()</code>
<code>getw()</code>	<code>getc() getw()</code> is replaced by performing two calls to <code>getc()</code>
<code>modloadap()</code>	<code>_os_loadap()</code>

Table 1-38 sys_clib.l Function Equivalents (continued)

sys_clib.l Function	Corresponding Function
<code>_gregorian()</code>	<code>_os_gregorian()</code>
<code>munlink()</code>	<code>_os_unlink()</code>
<code>_gs_devn()</code>	<code>_os_gs_devn()</code>
<code>munload()</code>	<code>_os_unload()</code>
<code>_gs_eof()</code>	<code>_os_gs_eof()</code>
<code>open()</code>	<code>_os_open()</code>
<code>_gs_gfd()</code>	<code>_os_gs_fd()</code>
<code>opendir()</code>	<code>_os_open()</code>
<code>_gs_opt()</code>	<code>_os_gs_popt()</code>
<code>os9exec()</code>	<code>_os_exec()</code>
<code>_gs_pos()</code>	<code>_os_gs_pos()</code>
<code>os9fork(), os9forkc()</code>	<code>_os_fork()</code>
<code>_gs_rdy()</code>	<code>_os_gs_ready()</code>
<code>_gs_size()</code>	<code>_os_gs_size()</code>
<code>os9kexec()</code>	<code>_os_exec()</code>
<code>hypot()</code>	none
<code>pause()</code>	<code>_os_prsnam()</code>
<code>ibrk()</code>	<code>malloc()</code>
<code>pause()</code>	<code>sleep()</code>

Table 1-38 sys_clib.l Function Equivalents (continued)

sys_clib.l Function	Corresponding Function
index()	strchr()
pffinit()	none
intercept()	_os_intercept()
pflinit()	none
_julian()	_os_julian()
prerr()	strerror()
kill()	_os_send()
_prgname()	argv[0]
lseek()	_os_seek()
_prsnam()	_os_prsnam()
putw()	putc() putc(), putw() is replaced by performing two calls to putc()
_ss_enrts()	_os_ss_enrts()
read()	_os_read()
_ss_lock()	_os_ss_lock()
readdir()	_os_read()
_ss_opt()	_os_ss_popt()
readln()	_os_readln()
_ss_pfd()	_os_ss_fd()

Table 1-38 sys_clib.l Function Equivalents (continued)

sys_clib.l Function	Corresponding Function
<code>rewinddir()</code>	<code>_os_seek()</code>
<code>_ss_rel()</code>	<code>_os_ss_relea()</code>
<code>rindex()</code>	<code>strrchr()</code>
<code>_ss_rest()</code>	<code>_os_ss_reset()</code>
<code>sbrk()</code>	<code>malloc()</code>
<code>_ss_size()</code>	<code>_os_ss_size()</code>
<code>seekdir()</code>	<code>_os_seek()</code>
<code>_ss_ssig()</code>	<code>_os_ss_sendsig()</code>
<code>_setcrc()</code>	<code>_os_setcrc()</code>
<code>_ss_tiks()</code>	<code>_os_ss_ticks()</code>
<code>setime()</code>	<code>_os_setime() / _os9_setime()</code>
<code>_ss_wtrk()</code>	<code>_os_ss_wtrack()</code>
<code>setpr()</code>	<code>_os_setpr()</code>
<code>_strass()</code>	<code>memcpy()</code> on structure assignment
<code>setstat()</code>	<code>_os_setstat()</code>
<code>strhcpy()</code>	none
<code>_setsys()</code>	<code>_os_setsys()</code>
<code>sys_mktime()</code>	<code>mktime()</code> for user programs

Table 1-38 sys_clib.l Function Equivalents (continued)

sys_clib.l Function	Corresponding Function
<code>setuid()</code>	<code>_os_setuid()</code>
<code>_sysdate()</code>	<code>_os_setime() / _os9_setime()</code>
<code>sigmask()</code>	<code>_os_sigmask()</code>
<code>_sysdbg()</code>	<code>_os_sysdbg()</code>
<code>sleep()</code>	<code>_os_sleep() / _os9_sleep()</code>
<code>telldir()</code>	<code>_os_gs_pos()</code>
<code>srqcmem()</code>	<code>_os_srqmem()</code>
<code>tsleep()</code>	<code>_os_sleep()</code>
<code>_srqmem()</code>	<code>_os_srqmem() / _os9_srqmem()</code>
<code>unlink()</code>	<code>_os_delete()</code>
<code>_srtmem()</code>	<code>_os_srtmem()</code>
<code>unlinkx()</code>	<code>_os_delete()</code>
<code>_ss_attr()</code>	<code>_os_ss_attr()</code>
<code>wait()</code>	<code>_os_wait()</code>
<code>_ss_dcoff()</code>	<code>_os_ss_dcoff()</code>
<code>write()</code>	<code>_os_write()</code>
<code>_ss_dcon()</code>	<code>_os_ss_dcon()</code>
<code>writeln()</code>	<code>_os writeln()</code>

Table 1-38 sys_clib.I Function Equivalents (continued)

sys_clib.I Function	Corresponding Function
<code>_ss_dsrts()</code>	<code>_os_ss_dsrts()</code>
-	-

The Function Description

Each function description includes a minimum of the following sections:

- Syntax
- OS-9 Attributes
- Description
- Library

In addition, descriptions may also contain sections for the following:

- Examples
- Errors
- References to other calls (See Also)

Syntax

The syntax shows how the function and parameters look if written as a C function definition, even if the actual function is a macro or is written in assembly language.

For example, the syntax for `fopen()` appears as follows:

```
#include <stdio.h>
FILE *fopen(const char *name, const char *action);
```

This indicates that `fopen()` requires the `<stdio.h>` header file, returns a pointer to a structure of type `FILE`, and requires two parameters, both pointers to constant character strings. The parameter names are suggestions only; you can use any name.

OS-9 Attributes

The OS-9 Attributes section lists various attributes of each function in relation to OS-9—including whether the function is compatible with OS-9 and/or OS-9 for 68K; whether the function is in user state and/or system state; whether the function is safe for use in a threaded application; and whether the function is re-entrant. Note that functions which are not re-entrant should not be used in signal handlers unless signals are masked before the functions are used and unmasked after the functions return.

Library

The library field indicates the libraries that contain the function.

Errors

When an error occurs, C functions typically return an error code in the global variable `errno`. You must include the file `<errno.h>` in C programs to declare `errno`.

References to Other Calls

Many functions also have a “See Also” section. Functions listed in the “See Also” section are related functions or are called by the function being described.

C Shared Library

The following two CSL (C Shared Library) files are available when configuring OS-9 systems:

- `csl`
- `mt_csl`

Both files contain a module named `csl`. Use the file `csl` for systems that execute no threaded `csl`-using applications. Use the file `mt_csl` for systems that execute both threaded and non-threaded `csl`-using applications.

Table 1-39 lists all functions available in `csl.1`. **Table 1-40** lists all functions available in `mt_csl.1`.

Table 1-39 CSL Function List

Function	Description
<code>access()</code>	Determine file accessibility.
<code>asctime()</code>	Convert broken-down time to string format.
<code>atexit()</code>	Specify function to call at normal program termination.
<code>atof()</code>	Alpha to floating conversion.
<code>atoi()</code>	Alpha to interger conversion.
<code>atol()</code>	Alpha to long converstion.
<code>_aton()</code>	Alpha to numeric translation.
<code>_atou()</code>	Alpha to unsigned converstion.

Table 1-39 CSL Function List

Function	Description
<code>calloc()</code>	Allocate storage for array.
<code>chmod()</code>	Change file access permissions.
<code>chown()</code>	Change owner of file.
<code>clearerr()</code>	Clear error condition.
<code>close()</code>	Close path.
<code>create()</code>	Create file.
<code>_dtoa()</code>	Double to ASCII conversion.
<code>_errmsg()</code>	Print error message.
<code>_exit()</code>	Task termination.
<code>fclose()</code>	Close file.
<code>fdopen()</code>	Attach path to file pointer.
<code>feof()</code>	Check buffered file for end of file.
<code>ferror()</code>	Check buffered file for error condition.
<code>fflush()</code>	Flush file's buffer.
<code>fgetc()</code>	Get character from file.
<code>fgetpos()</code>	Get current position in file.
<code>fgets()</code>	Get string from file.

Table 1-39 CSL Function List

Function	Description
<code>fopen()</code>	Open file.
<code>fprintf()</code>	Formatted output.
<code>fputc()</code>	Output character to file.
<code>fputs()</code>	Output string to file.
<code>fread()</code>	Read data from file.
<code>free()</code>	Return memory.
<code>_freemin()</code>	Set memory reclamation bound.
<code>freopen()</code>	Re-open file.
<code>fscanf()</code>	Input string conversion.
<code>fseek()</code>	Reposition file pointer.
<code>fsetpos()</code>	Set current file position.
<code>ftell()</code>	Report file pointer position.
<code>fwrite()</code>	Write data to file.
<code>getc()</code> , <code>getchar()</code>	Get next character from file, stdin.
<code>getenv()</code>	Get value for environment name.
<code>getpid()</code>	Return group ID.
<code>gets()</code>	Get string from file.

Table 1-39 CSL Function List

Function	Description
<code>_getsys()</code>	Get system global variables.
<code>getuid()</code>	Determine user ID number.
<code>gmtime()</code>	Convert calendar time to Greenwich Mean Time.
<code>kill()</code>	Send Signal to Process
<code>_lalloc()</code>	Allocate storage for array (low-overhead).
<code>_lfree()</code>	Return memory (low overhead).
<code>_lmalloc()</code>	Allocate memory from arena (low-overhead).
<code>localeconv()</code>	Numeric formatting convention inquiry.
<code>localtime()</code>	Convert Calendar Time to Local Time.
<code>_realloc()</code>	Resize block of memory (low-overhead).
<code>lseek()</code>	Position file pointer.
<code>malloc()</code>	Allocate memory from arena.
<code>_mallocmin()</code>	Set minimum allocation size.
<code>mbstowcs()</code>	Convert sequence of multibyte characters.
<code>mbtowc()</code>	Determine number of bytes in multibyte characters.
<code>memcpy()</code>	Copy memory.
<code>memmove()</code>	Move memory.

Table 1-39 CSL Function List

Function	Description
<code>memset()</code>	File memory.
<code>mktime()</code>	Convert Broken-Down Time to Calendar Time.
<code>open()</code>	Open file.
<code>opendir()</code>	Open directory.
<code>_os_loadp()</code>	Load and link to memory module using PATH.
<code>_parsepath()</code>	Parse disk file (RBF) pathlist.
<code>perror()</code>	Map error number.
<code>pow()</code>	Power function.
<code>printf()</code>	Formatted output.
<code>putc(), putchar()</code>	Put next character to file, standard out.
<code>puts()</code>	Output string to file.
<code>read()</code>	Read bytes from path.
<code>readdir()</code>	Return pointer.
<code>readln()</code>	Read bytes from path.
<code>realloc()</code>	Resize block of memory.
<code>remove()</code>	Remove file.
<code>rename()</code>	Rename file.

Table 1-39 CSL Function List

Function	Description
<code>rewind()</code>	Return file pointer to zero.
<code>rewinddir()</code>	Reset position of directory stream.
<code>scanf()</code>	Input strings conversion.
<code>seekdir()</code>	Set position of next readdir.
<code>setlocale()</code>	Locale control.
<code>_setsys()</code>	Set/Examine system global variables.
<code>setvbuf()</code>	Set up buffer for I/O stream
<code>sprintf()</code>	Formatted output.
<code>sscanf()</code>	Input strings conversion.
<code>strcat()</code>	String catenation.
<code>strchr()</code>	Locale string.
<code>strcmp()</code>	String comparison.
<code>strcpy()</code>	String copy.
<code>strerror()</code>	Map error message string.
<code>strftime()</code>	Place formatted time in buffer.
<code>strlen()</code>	Determine string length.
<code>strncat()</code>	String catenation.

Table 1-39 CSL Function List

Function	Description
<code>strncmp()</code>	String comparison.
<code>strncpy()</code>	String copy.
<code>strrchr()</code>	Locate last occurrence of string.
<code>strtod()</code>	String to double conversion.
<code>strtok()</code>	Break string into tokens.
<code>strtol()</code>	String to long conversion.
<code>strtoul()</code>	String to long conversion.
<code>system()</code>	Shell command execution.
<code>telldir()</code>	Return current location.
<code>time()</code>	Get Calendar Time.
<code>tmpfile()</code>	Create temporary binary file.
<code>tmpnam()</code>	Generate unique valid filename.
<code>ungetc()</code>	Unget character.
<code>vfprintf()</code>	Print to file.
<code>vprintf()</code>	Print to standard output.
<code>vsprintf()</code>	Print to string.
<code>wcsrtombs()</code>	Convert sequence of wide chars to multibyte chars

Table 1-39 CSL Function List

Function	Description
wctomb()	Convert wide character to multibyte character.
write()	Write bytes to path.
writeln()	Write bytes to path.

Table 1-40 MT_CSL Function List

Function	Description
pthread_attr_destroy()	Free thread attribute object.
pthread_attr_getdetachstate()	Get detach state attribute.
_pthread_attr_getinitfunction()	Get initialization function attribute.
_pthread_attr_getpriority()	Get priority attribute.
pthread_attr_getstackaddr()	Get stack address attribute.
pthread_attr_getstacksize()	Get stack size attribute.
pthread_attr_init()	Allocate thread creation attribute object.
pthread_attr_setdetachstate()	Set detached state attribute.
_pthread_attr_setinitfunction()	Set initialization function attribute.

Table 1-40 MT_CS Function List

Function	Description
<code>_pthread_attr_setpriority()</code>	Set priority attribute.
<code>pthread_attr_setstackaddr()</code>	Set stack address attribute.
<code>pthread_attr_setstacksize()</code>	Set stack size attribute.
<code>pthread_cancel()</code>	Cancel target thread.
<code>pthread_cleanup_pop()</code>	Pop cleanup routine.
<code>pthread_cleanup_push()</code>	Push cleanup routine.
<code>pthread_cond_broadcast()</code>	Release threads waiting for condition variable.
<code>pthread_cond_init()</code>	Allocate condition variable object.
<code>pthread_cond_signal()</code>	Release thread waiting for condition variable.
<code>pthread_cond_timedwait()</code>	Wait on condition variable for specified interval.
<code>pthread_cond_wait()</code>	Wait on condition variable.
<code>pthread_create()</code>	Create new thread.
<code>pthread_detach()</code>	Orphan target thread.
<code>pthread_exit()</code>	Terminate thread.
<code>pthread_getspecific()</code>	Get thread-specific data pointer.

Table 1-40 MT_CSL Function List

Function	Description
<code>_pthread_getstatus()</code>	Get thread status information.
<code>_pthread_interrupt()</code>	Interrupt target thread.
<code>_pthread_interrupt_clear()</code>	Clear interrupt request for target thread.
<code>pthread_join()</code>	Wait for target thread to terminate.
<code>pthread_key_create()</code>	Create thread-specific data key.
<code>pthread_key_delete()</code>	Delete thread-specific data key.
<code>pthread_mutex_destroy()</code>	Free mutex object.
<code>pthread_mutex_getprioceiling()</code>	Get mutex priority ceiling.
<code>pthread_mutex_init()</code>	Allocate mutex object.
<code>pthread_mutex_lock()</code>	Lock mutex object.
<code>pthread_mutex_setprioceiling()</code>	Set mutex priority ceiling.
<code>pthread_mutex_trylock()</code>	Lock mutex object (non-blocking)
<code>pthread_mutex_unlock()</code>	Unlock mutex object.
<code>pthread_once()</code>	Execute routine once per process.

Table 1-40 MT_CSL Function List

Function	Description
<code>_pthread_resume()</code>	Decrement suspension counter.
<code>pthread_setcancelstate()</code>	Set cancel state.
<code>pthread_setcanceltype()</code>	Set cancel type.
<code>_pthread_setpr()</code>	Set priority for target thread.
<code>_pthread_setsignalrange()</code>	Set range of signal values.
<code>pthread_setspecific()</code>	Set thread-specific data pointer.
<code>_pthread_setsuspendable()</code>	Decrement suspendability counter.
<code>_pthread_setunsuspendable()</code>	Increment suspendability counter.
<code>_pthread_suspend()</code>	Increment suspension counter.

Chapter 2: Functions

This chapter includes library function definitions in alphabetical order according to some special rules.

- Special characters (not letters, numbers, or underscores) are listed first.
- Function calls are listed in alphabetic order next without regard for numbers and underscores.
- If two function calls are identical using these rules, then they are alphabetized according to the following order:
 - 1.Symbols
 - 2.Underscores
 - 3.Alphabetic characters
 - 4.Numbers

abort()

Abnormal Program Termination

Syntax

```
#include <stdlib.h>
void abort(void);
```

Description

abort() abnormally terminates a program by raising the signal SIGABRT. However, if the signal SIGABRT is being caught and the signal handler does not return, the program is not terminated. If SIGABRT returns, the program is terminated abnormally. This flushes and closes the open streams and removes temporary files. The process returns the status EXIT_FAILURE.



Note

abort() does not return to the caller.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

clib.l

abs()

Integer Absolute Value

Syntax

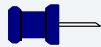
```
#include <stdlib.h>
int abs(int value);
```

Description

`abs()` returns the absolute value of its integer parameter.

`value` is the integer parameter. (Input)

`abs(0x80000000)` returns
0x80000000 as the result.



Note

Applying `abs()` to the most negative integer yields a result that is the most negative integer.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

access()

Determine File Accessibility

Syntax

```
#include <modes.h>
int access(
            const char    *name,
            int           perm);
```

Description

`access()` returns zero if the mode(s) specified in the file permissions are correct for the user to access the specified file.

name	is a pointer to the name of the file. (Input)
perm	may be any legal mode as defined in the <code>modes.h</code> header file. (Input)
	Use a mode value of zero to verify that a file exists. Only the lower 16 bits of <code>perm</code> are used. If the file cannot be accessed, -1 is returned and the appropriate error code is placed in <code>errno</code> .
	This value may not be compatible with other systems.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`sys_clib.l`

acos()

Arc Cosine Function

Syntax

```
#include <math.h>
double acos(double x);
```

Description

acos() returns the arc cosine of x (input), in the range of $[0, \pi]$ radians. A domain error, EDOM stored in errno, occurs for parameters not in the range $[-1, +1]$. On error, acos() returns HUGE_VAL.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

clib.l

Possible Error

EDOM

alarm()

Sends signal SIGALRM

Syntax

```
#include <UNIX/os9def.h>
unsigned int alarm(unsigned int seconds);
```

Description

alarm() requests that a signal be sent to the requesting process after a specified number of seconds. Unless caught or ignored, the signal terminates the process. alarm() sends signal SIGALRM when the alarm expires.

alarm() requests are not stacked; successive calls reset the alarm clock. If the argument is 0, any alarm() request is canceled.

alarm() returns -1 and sets the global value errno if an error occurs.

seconds	is a specified amount of time after which a signal is sent. (Input)
---------	---

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

unix.l

See Also

[_os9_alarm_atdate\(\)](#)
[_os_alarm_set\(\)](#)

alloca()

Allocates Bytes of Space in Stack Frame

Syntax

```
#include <UNIX/os9def.h>
void *alloca(unsigned int size);
```

Description

alloca() allocates `size` number of zero bytes from the `malloc()` pool. This space is not automatically freed.

alloca() returns a pointer to the allocated bytes. If an error occurs, `NULL` is returned and the global variable `errno` is set.

<code>size</code>	is the number of zero bytes from the <code>malloc()</code> pool. (Input)
-------------------	---

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

`unix.l`

alm_atdate()

Set Alarm at Gregorian Date/Time

Syntax

```
#include <alarm.h>
```

```
int alm_atdate(  
    int      sigcode,  
    int      time,  
    int      date);
```

Description

`alarm_atdate()` requests that a signal be sent to the requesting process at a specific Gregorian time and date. The time and date must be in the following format:

Figure 2-1 `alm_atdate()` Time and Date Format

	Byte 0	Byte 1	Byte 2	Byte 3
Time:	0	Hour (0-23)	Minute	Second
Date:	Year (two bytes)		Month	Day

sigcode specifies the signal to be sent to the caller. (Input)

time specifies the time to send the signal.
(Input)

date specifies the Gregorian date for the signal to be sent. (Input)

Because the system time and date may be changed, the alarm signal is sent when the system time and date become greater than or equal to the alarm time.

If an error occurs, -1 is returned and the appropriate error code is placed in the global variable `errno`.

If errors do not occur, `alm_atdate()` returns the alarm ID.



For More Information

Refer to your operating system technical manual for information about alarms.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe
Re-entrant:	No

Library

`sys_clib.l`

See Also

[`_os9_alarm_atdate\(\)`](#)
[`_os_alarm_set\(\)`](#)

alm_atjul()

Set Alarm at Julian Date/Time

Syntax

```
#include <alarm.h>

int alm_atjul(
    int      sigcode,
    int      time,
    int      date);
```

Description

alm_atjul() requests that a signal be sent to the requesting process at a specific Julian date and time.

sigcode	specifies the signal to be sent to the caller. (Input)
time	should contain the time at which to send the signal, expressed as the number of seconds after midnight. (Input)
date	must contain the Julian date on which to send the signal. (Input)
	The Julian date is the number of days since 1 January 4713 B.C., the beginning of the Julian period.



Note

A Julian day begins at midnight on OS-9 systems. Standard Julian days begin twelve hours earlier, at noon. For example, 1:00 a.m. January 2, 4713 B.C. is one hour after the beginning of Julian Day 1 for OS-9, but thirteen hours after the beginning in standard Julian time.

Because the system time and date may be changed, the alarm signal is sent when the system time and date become greater than or equal to the alarm time.

If an error occurs, -1 is returned and the appropriate error code is placed in the global variable `errno`.

If errors do not occur, `alm_atjul()` returns the alarm ID.



For More Information

Refer to your operating system technical manual for information about alarms.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`sys_clib.l`

See Also

[`_os9_alarm_atjul\(\)`](#)
[`_os_alarm_set\(\)`](#)

alm_cycle()

Set Alarm at Specified Time Intervals

Syntax

```
#include <alarm.h>

int alm_cycle(
    int      sigcode,
    int      time_interval);
```

Description

alm_cycle() sends a signal to the requesting process after the specified time has elapsed and then resets the alarm to provide a recurring periodic signal.

sigcode	specifies the signal to be sent to the caller. (Input)
time_interval	specifies the periodic interval at which the signal is to be sent. (Input)
	For example, if the request is made at time x and time_interval is t , the signal is sent to the requesting process at times $(x + t)$, $(x + 2t)$, $(x + 3t)$, and so forth until the alarm is deleted.
	If the most significant bit of time_interval is set, time_interval is assumed to be in 256ths of a second. Otherwise, time_interval is assumed to be in units of system clock ticks (refer to CLOCKS_PER_SEC in the time.h header file). The minimum time_interval allowed is two system clock ticks.

If an error occurs, -1 is returned and the appropriate error code is placed in the global variable `errno`.

If errors do not occur, `alm_cycle()` returns the alarm ID.



For More Information

Refer to your operating system technical manual for information about alarms.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

[alm_delete\(\)](#)
[alm_set\(\)](#)
[alm_cycle\(\)](#)
[alm_delete\(\)](#)
[_os_alarm_set\(\)](#)

alm_delete()

Remove Pending Alarm Request

Syntax

```
#include <alarm.h>
```

```
int alm_delete(int alarmid);
```

Description

`alm_delete()` cancels the specified alarm request.

alarmid is the alarm ID of alarm to cancel. (Input)

If `alarmid` is zero, all pending alarm requests are cancelled.

If an error occurs, -1 is returned and the appropriate error code is placed in the global variable `errno`.

If errors do not occur, `alm_delete()` returns zero.



For More Information

Refer to the ***OS-9 Technical Manual*** for information about alarms.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe
Re-entrant:	Yes

Library

sys lib.l

See Also

_os_alarm_delete()

alm_set()

Set Alarm after Specified Time Interval

Syntax

```
#include <alarm.h>
```

```
int alm_set(          int    sigcode,  
                  int    time);
```

Description

`alm_set()` requests that a signal be sent to the requesting process after the specified time has elapsed. For example, if the request is made at time x and `time` is t , the signal is sent at time $(x + t)$, unless the alarm request is cancelled.

sigcode is the signal code to be sent to the requesting process. (Input)

time specifies the time interval. (Input)

If the most significant bit of `time` is set, `time` is assumed to be in 256ths of a second. Otherwise, `time` is assumed to be in units of system clock ticks (refer to `CLOCKS_PER_SEC` in the `time.h` header file). The minimum `time` allowed is two system clock ticks.

If an error occurs, -1 is returned and the appropriate error code is placed in the global variable `errno`.

If errors do not occur, `alm_set()` returns the alarm ID.



For More Information

Refer to your operating system technical manual for information about alarms.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe
Re-entrant:	Yes

Library

sys_clib.l

See Also

[_os_alarm_set\(\)](#)

asctime()

Convert Broken-Down Time to String Format

Syntax

```
#include <time.h>
char *asctime(struct tm *tp);
```

Description

`asctime()` converts the broken-down time structure into the following 26-byte string format, including the terminating `\0`:

`xxx mmm dd hh:mm:ss yyyy\n\0`

`xxx` is one of the following days of the week:

Table 2-1 Days of Week

Sun	Mon	Tue	Wed	Thu	Fri	Sat

`mmm` is one of the following months of the year:

Table 2-2 Months of Year

Jan	Feb	Mar	Apr	May	Jun
Jul	Aug	Sep	Oct	Nov	Dec

`dd` specifies the day of the month.

`hh:mm:ss` specifies the time in hours, minutes, and seconds.

`YYYY` specifies the year.

`tp` is a pointer to the broken-down time structure. (Input)



Note

`asctime()` returns a pointer to a static area which may be overwritten. To insure data integrity, use the string or save it immediately.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`clib.l`

See Also

`ctime()`
`localtime()`
`gmtime()`
`sys_mktime()`
`strftime()`

asin()

Arc Sine Function

Syntax

```
#include <math.h>
double asin(double x);
```

Description

asin() returns the arc sine of x (input), in the range $[-\pi/2, +\pi/2]$ radians. A domain error, EDOM stored in errno, occurs for parameters not in the range $[-1, +1]$. On error, asin() returns HUGE_VAL.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

clib.l

Possible Error

EDOM

assert()

Place Diagnostics in Programs

Syntax

```
#include <assert.h>
void assert(int expression);
```

Description

`assert()` is a macro that places diagnostics in programs. If expression is false, `assert()` writes a message about the failed call to standard error. The following message is printed:

Assertion failed: expression, file filename,
line line#\n

expression is the text of the parameter. (Input)

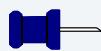
`filename` is the name of the file containing the assert macro.

`line#` is the line number of the file containing the `assert` macro.



Note

`assert()` never returns a value.



Note

If `NDEBUG` is defined as a macro name when the `assert.h` header file is included, `assert()` is defined as follows:

```
#define assert(ignore) ((void)0)
```

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`assert()` is a macro defined in `assert.h`.

See Also

[abort\(\)](#)
[raise\(\)](#)
[signal\(\)](#)

Syntax

```
#include <math.h>
double atan(double x);
```

Description

atan() returns the arc tangent of x (input), in the range $[-\pi/2, +\pi/2]$ radians.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

clib.lib

atan2()

Arc Tangent Function

Syntax

```
#include <math.h>
double atan2(
    double      y,
    double      x);
```

Description

atan2() returns the arc tangent of y (input) divided by x (input)--that is, $\text{atan}(y/x)$, in the range $[-\pi, \pi]$ radians. atan2() uses the signs of both y and x to determine the quadrant of the returned value. A domain error, EDOM placed in errno, occurs if both y and x are zero. On error, atan2() returns HUGE_VAL.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

clib.l

Possible Error

EDOM

atexit()

Specify Function to Call at Normal Program Termination

Syntax

```
#include <stdlib.h>
int atexit(void (*func)(void));
```

Description

atexit() specifies a function to be called without parameters at normal program termination. Thirty-two functions may be specified.

func is a pointer to the function. (Input)

atexit() returns zero if it succeeds. Otherwise, it returns a non-zero value.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

clib.lib

See Also

[exit\(\)](#)

atof()

Alpha to Floating Conversion

Syntax

```
#include <stdlib.h>
double atof(const char *string);
```

Description

atof() converts a string into its equivalent representation in type double.

string is a pointer to the string. (Input)

Except that no error indication is given, atof() is equivalent to the following:

```
strtod(nptr, (char **)NULL)
```

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

clib.1

See Also

[strtod\(\)](#)

atoi()

Alpha to Integer Conversion

Syntax

```
#include <stdlib.h>
int atoi(const char *string);
```

Description

atoi() converts a string into its equivalent representation in type int.

string is a pointer to the string. (Input)

Except that no error indication is given, atoi() is equivalent to the following:

```
(int) strtol(nptra, (char**)NULL, 10)
```

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

clib.l

See Also

[strtol\(\)](#)

atol()

Alpha to Long Conversion

Syntax

```
#include <stdlib.h>
long atol(const char *string);
```

Description

atol() converts a string into its equivalent representation in type long.

string is a pointer to the string. (Input)

Except that no error indication is given, atol() is equivalent to the following:

```
strtol(nptr, (char**)NULL, 10)
```

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

clib.l

See Also

[strtol\(\)](#)

_aton()

Alpha to Numeric Translation

Syntax

```
#include <aton.h>
_number_type _aton(
    char      **buf,
    _number   *value);
```

Description

`_aton()` converts a string into its associated numeric value, if possible.

`buf` is a pointer to a pointer to a string containing a printable representation of a number expressed in base ten.
(Input/Output)

The number may be an integer or a real number. You may also express the number in the following exponential format:

<number> (E | e) (- | +) <exponent>

`value` is a pointer to one of the three possible values that can be returned. (Output)

Table 2-3 `_aton()` Return Values

Value	Description
<code>_ERROR</code>	No recognizable number is found, or the value cannot be represented (it would cause overflow or underflow).
<code>_INTEGER</code>	The recognized number is an integer.
<code>_REAL</code>	The recognized number is not an integer.

`_aton()` updates `buf` to point to the character just after the recognized string. It updates `value` to contain the recognized number, unless an error occurs.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`clib.lib`

_atou()

Alpha to Unsigned Conversion

Syntax

```
#include <stdlib.h>
unsigned _atou(const char *string);
```

Description

`_atou()` converts a string into its appropriate unsigned numeric value, if possible.

`string`

is a pointer to a string containing a printable representation of a number expressed in the format below. (Input)

[+ / -] <digits>

`_atou()` treats `long` and `int` values identically.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`clib.l`

attach()

Attach to Device

Syntax

```
#include <modes.h>
#include <io.h>
dev_list *attach(
    const char    *name,
    int           mode);
```

Description

attach() causes a new device to become known to the system or verifies that the device is already attached.

If the device descriptor is found and the device is not already attached, attach() links to its file manager and device driver, and places their addresses in a new device table entry.

name	is a pointer to the name of the device descriptor. (Input)
mode	is the access mode. (Input) Possible access modes include FAM_READ, FAM_WRITE, S_IREAD, and S_IWRITE. mode may be used to verify that subsequent read and/or write operations are permitted. Only the lower 16 bits of mode are used.

If the attach fails, -1 is returned and the appropriate error code is placed in the global variable errno.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

sys_clib.l

See Also

[_os_attach\(\)](#)

[I\\$Attach](#)

[I_ATTACH](#)

OS-9 for 68K Technical Manual

OS-9 Technical Manual

bcmpl()

Compares Bytes from Two Memory Areas

Syntax

```
#include <UNIX/os9def.h>
int bcmp(
    void          *addr1,
    void          *addr2,
    unsigned int   len);
```

Description

`bcmp()` compares the memory at the first address (`addr1`) against the memory at the second address (`addr2`). The function returns zero if the memory areas are identical, non-zero otherwise. Both memory areas are assumed to be the given length (`len`). If `len` is zero, the function returns zero.

`addr1` is the first memory address. (Input)

`addr2` is the second memory address. (Input)

`len` is the length of both addresses. (Input)

Attributes

Operating System: OS-9 and OS-9 for 68K

State: User and System

Threads: Safe

Re-entrant: Yes

Library

`unix.l`

See Also

[bcopy\(\)](#)

[bzero\(\)](#)

[ffs\(\)](#)

bcopy()

Copies Bytes from One Memory Area to Another Memory Area

Syntax

```
#include <UNIX/os9def.h>
void bcopy(
            void          *addr1,
            void          *addr2,
            unsigned int   len);
```

Description

`bcopy()` copies bytes, of length `len`, from one memory area (`addr1`) to another (`addr2`). Overlapping memory areas are handled correctly.

<code>addr1</code>	is the first memory area. (Input)
<code>addr2</code>	is the second memory area. (Output)
<code>len</code>	is the length of <code>addr1</code> and <code>addr2</code> . (Input)

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`unix.l`

See Also

[bcmp\(\)](#)
[bzero\(\)](#)
[ffs\(\)](#)

bsearch()

Search Array

Syntax

```
#include <stdlib.h>
void *bsearch(
    const void      *key,
    const void      *base,
    size_t          nmemb,
    size_t          size,
    int (*compar)  (const void *,
                     const void *));
```

Description

`bsearch()` searches an array of `nmemb` objects for an element that matches `key`.

key	is a pointer to the key object. (Input)
base	is a pointer to the initial element of the array. (Input)
nmemb	specifies the number of array elements. (Input)
size	specifies the size of each array element. (Input)

compar

is a pointer to a comparison function.
(Input/Output)

compar is called with two parameters that point to the key object and to an array element, in that order. compar returns an integer that is:

- Negative if key is considered less than the array element.
- Zero if key is equal to the array element.
- Positive if key is considered greater than the array element.

The array consists of all elements that compare less than, equal to, and greater than the key object, in that order.

bsearch() returns a pointer to a matching array element. If no match is found, it returns a null pointer. If two elements compare as equal, which element is matched is unspecified.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

clib.lib

buildpath()

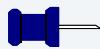
Adds a Filename to the Root Directory Variable

Syntax

```
#include <UNIX/os9def.h>
char *buildpath(const char *filename);
```

Description

`buildpath()` adds `filename` (input) to the value of the environment variable `XOS9ROOTDIR`. If `XOS9ROOTDIR` is not set, a default value of `/h0` is used.



Note

`buildpath()` returns the final string in a static buffer that should be saved if additional calls are made to `buildpath()`. For example, with `XOS9ROOTDIR` set to `/dd/ETC`:

```
buildpath(xfile);
```

returns the string: `/dd/ETC/xfile`

without `XOS9ROOTDIR` set:

```
buildpath(xfile);
```

returns the string: `/h0/xfile`

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`unix.l`

bzero()

Clear Bytes In Memory

Syntax

```
#include <UNIX/os9def.h>
void bzero(
            void             *addr,
            unsigned int     len);
```

Description

`bzero()` clears bytes in memory starting from an address (`addr`) for a number of bytes (`len`).

`addr` is a memory address. (Input/Output)

`len` is the number of bytes. (Input)

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`unix.l`

See Also

[bcopy\(\)](#)
[bcmpl\(\)](#)
[ffs\(\)](#)

calloc()

Allocate Storage for Array

Syntax

```
#include <stdlib.h>
void *calloc(
    size_t      nmemb,
    size_t      size);
```

Description

`calloc()` allocates space for an array. The allocated memory is cleared to zeroes.

`nmemb` is the number of elements in the array.
(Input)

`size` is the size of each array element. (Input)

If the allocation is successful, `calloc()` returns a pointer to the area. If the allocation fails or if `size` is zero, `NULL` is returned. The space designated by the returned pointer is suitably aligned to hold an object of any type.

Attributes

Operating System: OS-9 and OS-9 for 68K

State: User and System

Threads: Safe

Re-entrant: No

Standards: ANSI

Library

`clib.l`

See Also

[ebrk\(\)](#)
[free\(\)](#)
[malloc\(\)](#)
[_os_srqmem\(\)](#)
[_os9_srqmem\(\)](#)
[realloc\(\)](#)

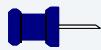
cbrt()Cube Root

Syntax

```
#include <UNIX/os9def.h>
double cbrt(double x);
```

Description

`cbrt()` returns the cube root of `x` (input).

**Note**

The declaration `double cbrt()` must be included in any code that uses `cbrt()`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`unix.l`

ceil()

Ceiling Function

Syntax

```
#include <math.h>
double ceil(double x);
```

Description

ceil() returns the smallest integer, as a double, that is not less than x (input).

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

clib.l

See Also

[floor\(\)](#)

chainc(), chain()

Load and Execute New Module

Syntax

```
#include <process.h>
int chainc(
    const char *modname,
    int parmsize,
    const char *parmPTR,
    int type,
    int lang,
    int datasize,
    int prior,
    int pathent);
int chain(
    const char *modname,
    int parmsize,
    const char *parmPTR,
    int type,
    int lang,
    int datasize,
    int prior);
```

Description

`chainc()` executes a new program without the overhead of creating a new process. It is functionally similar to `os9forkc()` followed by `exit()`, but with less system overhead.

`chainc()` effectively resets the calling process' program and data areas and begins executing a new primary module. Open paths are not closed or otherwise affected.

modname	is a pointer to a null-terminated module name. (Input)
parmsize	is generally <code>strlen(parmptr)</code> . (Input)
parmPTR	is a pointer to a null-terminated string to be passed to the new module. (Input)

type and lang	specify the desired type and language of the module; values of zero indicate any type or language. Only the lower 16 bits of type and lang are used. (Input)
datasize	allocates extra memory to the new program. If no extra memory is required, datasize can be zero. (Input)
prior	is the new priority at which to run the program. Specify zero if no priority change is desired. Only the lower 16 bits of prior are used. (Input)
pathent	is the number of open paths for the new process to inherit. Only the lower 16 bits of pathent are used. (Input)

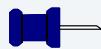
Regardless of whether the chain is successful, `chainc()` never returns to the caller; you must verify that the program to chain to exists and is executable before chaining. Use `modlink()` to check the module directory for the program or `modload()` to check the execution directory.

`chain()` is the same as `chainc()` without the pathent parameter. The number of open paths for the new process to inherit is three.



For More Information

[_os_exec\(\)](#) is the preferred method by which to chain to C programs.



Note

Beware of chaining to a system state program. `chain()` is an historical function and is likely to be removed in a future release. `_os_exec()` is the recommended function.

Threaded programs may only chain from the `main()` thread and may not chain if multiple threads are currently active within the process.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`modlink()`
`modload()`
`_os_chain()`
`os9fork()`, `os9forkc()`
`_os_fork()`
`_os_exec()`
`F$Chain`, `F$Fork`
`F_CHAIN`, `F_FORK`

OS-9 for 68K Technical Manual
OS-9 Technical Manual

change_const()

Change Current Static Storage Pointer

Syntax

```
#include <regs.h>
void *change_const(void *constptr);
void *_change_const(void *constptr);
```

Description

`change_const()` and `_change_const()` change the current constant data pointer and return the previous constant data pointer.

`constptr` is the value of the new constant data pointer. (Input)

These functions are not present in the libraries for a given processor if a constant data pointer is not applicable.

Attributes

Operating System: OS-9

State: User and System

Threads: Safe

Re-entrant: Yes

Library

`change_const()` - `cpu.l`, `_change_const()` - `os.lib.l`

change_static()

Change Current Static Storage Pointer

Syntax

```
#include <regs.h>
void *change_static(void *dest_stat);
void *_change_static(void *dest_stat);
```

Description

change_static() and _change_static() change the current static storage pointer and return the previous static storage pointer.

dest_stat is the new static storage pointer. (Input)

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

change_static() - cpu.l, _change_static() - os_lib.l

chdir()

Change Current Data Directory

Syntax

```
#include <modes.h>
int chdir(const char *dirname);
```

Description

`chdir()` changes the current data directory for the calling process.

`dirname` is a pointer to a string containing a path name for the directory. (Input)

`chdir()` returns zero after a successful call. If `dirname` is not a directory path name or some other error occurs, -1 is returned; and the appropriate error code is placed in the global variable `errno`.



Note

`chdir()` changes the data directory only for the program containing the function call, not the shell that executes the program. Use the built-in shell command `chd` to change the shell's data directory.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also[_os_chdir\(\)](#)[I\\$ChgDir](#)[I_CHDIR](#)[chd](#)[***OS-9 for 68K Technical Manual***](#)[***OS-9 Technical Manual***](#)[***Using OS-9 for 68K*** and \[***Using OS-9***\]\(#\)](#)

chmod()

Change File Access Permissions

Syntax

```
#include <modes.h>
int chmod(
            const char      *name,
            int              perm);
```

Description

chmod() changes the access permission bits associated with a file.

name must be a pointer to a string containing a file name. (Input)

perm should contain the desired bit pattern for the file permissions. Only the lower 16 bits of perm are used. (Input)

chmod() returns zero after a successful call. If the caller is not entitled to change the access permissions or if the file cannot be found, -1 is returned and the appropriate error code is placed in the global variable errno.



Note

Only the super user or the owner of the file may change the access permissions.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

sys_clib.l

See Also

[_os_ss_attr\(\)](#)

I\$SetStt

I_SETSTAT, SS_ATTR

attr

OS-9 for 68K Technical Manual

OS-9 Technical Manual

Using OS-9 for 68K and *Using OS-9*

chown()

Change Owner of File

Syntax

```
#include <modes.h>
int chown(
            const char      *name,
            int              newowner);
```

Description

chown() changes the owner number of a file.

name	is a pointer to the name of the file. (Input)
newowner	is the new owner ID to assign to the file. The owner ID consists of a group ID and a user ID. (Input)



For More Information

Refer to your operating system technical manual for information about how the owner ID is stored.

chown() returns zero after a successful call. If the caller does not have permission to change the owner ID or the file cannot be found, -1 is returned and the appropriate error code is placed in the global variable errno.



Note

Only the super user may change the owner ID.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

sys_clib.l

See Also

[_os_ss_fd\(\)](#)
[I\\$SetStt](#)
[I_SETSTAT, SS_FD](#)

OS-9 for 68K Technical Manual
OS-9 Technical Manual

chxdir()

Change Current Execution Directory

Syntax

```
#include <modes.h>
int chxdir(const char *dirname);
```

Description

`chxdir()` changes the current execution directory for the calling process.

`dirname` is a pointer to a string containing the directory name. (Input)

`chxdir()` returns 0 after a successful call. If `dirname` is not a directory path name or some other error occurs, -1 is returned and the appropriate error code is placed in the global variable `errno`.



Note

`chxdir()` changes the execution directory only for the program containing the function call, not the shell that executes the program. Use the built-in shell command `chx` to change the shell's execution directory.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also[_os_chdir\(\)](#)[I\\$ChgDir](#)[I_CHDIR](#)[chx](#)*[OS-9 Technical Manual](#)**[OS-9000 Technical Manual](#)**[Using OS-9 for 68K](#)* and *[Using OS-9](#)*

_cleareof(), cleareof()

Clear End of File Condition

Syntax

```
#include <stdio.h>
void _cleareof(FILE *stream);
void cleareof(FILE *stream);
```

Description

`_cleareof()` is a macro that resets the end-of-file condition that causes `feof()` to return a non-zero value.

`stream` is a pointer to the stream. (Input)

You can use this to retry an input operation on a terminal after end-of-file is encountered. `getc()` does not read characters until the end-of-file condition is cleared.



Note

If you are executing the compiler in strictly conforming ANSI mode, only `_cleareof()` is available. Otherwise, both calls are available.



Note

These calls are implemented as macros in the `stdio.h` header file.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

sys_clib.l

See Also

[feof\(\)](#)

[getc\(\)](#), [getchar\(\)](#)

clearerr()

Clear Error Condition

Syntax

```
#include <stdio.h>
void clearerr(FILE *stream);
```

Description

`clearerr()` clears the end-of-file and error indicators for a stream if compiled in strictly conforming ANSI mode. Otherwise, only the error indicator is cleared.

`stream` is a pointer to the stream to clear. (Input)

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`clib.l`

See Also

[_cleareof\(\)](#), [cleareof\(\)](#)
[feof\(\)](#)
[ferror\(\)](#)
[fopen\(\)](#)
[getc\(\)](#), [getchar\(\)](#)

clock()

Get Processor Time

Syntax

```
#include <time.h>
clock_t clock(void);
```

Description

`clock()` returns a value (in tick units) approximating the processor time used by the current process. The returned value may be divided by `CLOCKS_PER_SEC` to determine the processor time in seconds. `CLOCKS_PER_SEC` is defined in the `time.h` header file to be the number of ticks per second.

The era for `clock()` is the elapsed ticks for this process since the program was forked.

If `clock()` cannot determine the processor time, it returns (`clock_t`) -1 to indicate an error.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.lib`

See Also

`F$SetSys` and `F$Time`
`F_SETSYS` and `F_TIME`

OS-9 for 68K Technical Manual
OS-9 Technical Manual

close()Close Path

Syntax

```
#include <modes.h>
int close(int path);
```

Description

`close()` closes an open path. The path number is usually obtained by a previous call to `open()`, `creat()`, `create()`, or `dup()`. The standard paths 0, 1, and 2 (standard input, standard output, and standard error, respectively) are not normally closed by user programs.

`path` specifies the open path to close. (Input)

If an error occurs during the close, -1 is returned and the appropriate error code is placed in the global variable `errno`.

**Note**

Be sure to use only a path number, not a file pointer assigned by `fopen()`.

**Note**

The thread enabled version of this function contains a cancel point. For more information see ***Using OS-9 Threads***.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`open()`
`creat()`
`create()`
`dup()`
`_os_close()`

closedir()

Close Named Directory Stream

Syntax

```
#include <dir.h>
void closedir(DIR *dirp);
```

Description

`closedir()` closes the named directory stream and frees the structure associated with `dirp`.

`dirp` is a pointer to the directory stream.
(Input)

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

`sys_clib.l`

See Also

[opendir\(\)](#)
[readdir\(\)](#)
[rewinddir\(\)](#)
[seekdir\(\)](#)
[telldir\(\)](#)

_cmpnam()

Compare Two Strings

Syntax

```
#include <strings.h>
error code _cmpnam(
    const uchar      *pattern,
    const uchar      *string,
    uint32           length);
```

Description

`_cmpnam()` compares the target string to the pattern string to determine if they match.

target	is a pointer to the target string. (Input) The target name must be terminated by a null byte. Upper and lower case characters are considered to match.
pattern	is a pointer to the pattern string. (Input) Two metacharacters are recognized in the pattern string: <ul style="list-style-type: none">• A question mark (?) matches any single character.• An asterisk (*) matches any string of characters.
patlen	specifies the length of the pattern string. (Input)

`_cmpnam()` returns zero if the strings match. -1 is returned if no match occurs.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

sys_clib.l

See Also

[_os_cmpnam\(\)](#)

F\$CmpNam

F_CMPNAM

OS-9 for 68K Technical Manual
OS-9 Technical Manual

cos()

Cosine Function

Syntax

```
#include <math.h>
double cos(double x);
```

Description

`cos()` returns the cosine of `x` (input) as a double. The value of `x` is in radians. A domain error, `EDOM` placed in `errno`, occurs if `x` is ∞ or $-\infty$. `cos()` returns `HUGE_VAL` on error.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

Possible Error

`EDOM`

Syntax

```
#include <math.h>
double cosh(double x);
```

Description

`cosh()` returns the hyperbolic cosine of `x` (input). The value of `x` is in radians. If the magnitude of `x` is too large, a range error, `ERANGE` placed in `errno`, occurs and the properly signed `HUGE_VAL` is returned.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

Possible Error

`ERANGE`

_cpymem()

Copy External Memory

Syntax

```
#include <process.h>
int _cpymem(
    int          pid,
    int          count,
    void        *from,
    void        *into);
```

Description

`_cpymem()` copies memory owned by another process or the system into a buffer.

`pid` is the process ID number of the external process. (Input)

If `pid` is zero, the system's address space is assumed. Only the lower 16 bits of `pid` are used.

`count` is the number of bytes to copy. (Input)

`from` is a pointer to the address in the process' address space from which to copy. (Input)

`into` is a pointer to the buffer into which to copy the memory. (Output)

If an error occurs, -1 is returned and the appropriate error code is placed in the global variable `errno`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

sys_clib.l

See Also

[_os_cpymem\(\)](#)

[F\\$CpyMem](#)

[F_CPYMEM](#)

OS-9 for 68K Technical Manual

OS-9 Technical Manual

crc()Calculate Module CRC

Syntax

```
#include <module.h>
int crc(
    void             *ptr,
    unsigned int     count,
    int              *accum);
```

Description

`crc()` generates or checks the CRC (cyclic redundancy check) values of sections of memory. Compilers, assemblers, and other module generators use `crc()` to generate a valid module CRC.

ptr	is a pointer to the data. (Input)
count	specifies the byte count for the data. (Input)
accum	points to location at which the CRC accumulator is passed. (Input/Output)
	<code>crc()</code> also stores the updated CRC accumulator at the location pointed to by accum.

To generate the CRC for a module:

1. Initialize the accumulator to -1 (0xffffffff).
2. Perform the CRC over the module.
3. Call `crc()` with a NULL value for `ptr`.
4. Complement the CRC accumulator.
5. Write the contents of the accumulator to the module.



For More Information

Refer to your operating system technical manual for more information on how your operating system uses CRCs for modules.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

sys_clib.l

See Also

[_os_crc\(\)](#)

F\$CRC

F_CRC

OS-9 for 68K Technical Manual

OS-9 Technical Manual

Example

The following code fragment illustrates a technique for generating a CRC for a module:

```
#include <stdio.h>
#include <module.h>

main()
{
    char *ptr;
    int count,accum = -1;
    /* one or more calls to crc */
    crc(ptr,count,&accum);

    /* The final call to crc(). At */
    /* crc(ptr,count,&accum);

    /* this point the entire module less the four crc
    /* bytes has been processed */
    crc(NULL,0,&accum);
    /* this is a special case to */
    /* run a null byte through the */
    /* crc calculation */

    accum = ~accum; /* complement the accumulator.*/
    /* Previous calls have left the */
    /* most significant byte as 0xff */
    /* Complementing changes this byte*/
    /* to zero */

    fwrite(&accum,1,sizeof accum,fp);
    /* finally write out the four crc bytes */
}
```

creat()

Create File

Syntax

```
#include <modes.h>
int creat(
    const char *name,
    int mode);
```

Description

`creat()` returns a path number to a newly created file. The file is available for writing.

`name` is a pointer to the newly created file.
(Input)

If `name` is the name of an existing file, the file is truncated to zero length and the ownership and permissions remain unchanged.

mode specifies the access modes for the file.
(Input)

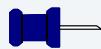
The modes.h header file contains the valid mode values. Only the lower 16 bits of mode are used.

If an error occurs, -1 is returned and the appropriate error code is placed in the global variable `errno`.



Note

You do not need to specify `write` access in `mode` to write to the file. You cannot create directories with this call; instead, use `os.makedirs()`.



Note

The thread enabled version of this function contains a cancel point. For more information see *Using OS-9 Threads*.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

sys_clib.l

See Also

open()	<i>OS-9 for 68K Technical Manual</i>
_os_create()	<i>OS-9 Technical Manual</i>
_os_mkdir()	<i>OS-9 Technical Manual</i>
_os_ss_size()	<i>OS-9 Technical Manual</i>
I\$Create and I\$SetStt	<i>OS-9 for 68K Technical Manual</i>
I_CREATE and I_SETSTAT	<i>OS-9 Technical Manual</i>
SS_SIZE	<i>OS-9 Technical Manual</i>

create()Create File

Syntax

```
#include <modes.h>
int create(
            const char          *name,
            int                  mode,
            int                  perm,
            {int initial_size});
```

Description

`create()` returns a path number to the newly created file.

`name` is a pointer to the newly created file.
(Input)

`mode` specifies the access modes for the file.
(Input)

The `modes.h` header file contains the valid `mode` values. Only the lower 16 bits of `mode` are used.

`perm` specifies the file permission attributes.
(Input)

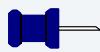
The `modes.h` header file contains the valid `perm` values. Only the lower 16 bits of `perm` are used.

`initial_size`

may be used to indicate the file's initial allocation size. (Input)

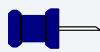
For disk files, the allocation is made even though the file size does not change. For pipe files, the pipe buffer is set to this value. `initial_size` may be used only if the `FAM_SIZE` (or `S_ISIZE`) bit is set in `mode`.

If the file already exists or any other error occurs, `-1` is returned and the appropriate error code is placed in the global variable `errno`.



Note

`create()` is similar to the `creat()` call. However, `create()` allows the caller to specify the exact file attributes desired and does not truncate the file if it is already present. You cannot create directories with this call; instead, use `_os_mkdir()`.



Note

The thread enabled version of this function contains a cancel point. For more information see ***Using OS-9 Threads***.

Attributes

Operating System:

OS-9 and OS-9 for 68K

State:

User and System

Threads:

Safe

Re-entrant:

Yes

Library

`sys_clib.l`

See Also

[_os_create\(\)](#)

[_os_mkdir\(\)](#)

[_os_ss_size\(\)](#)

I\$Create and I\$SetStat

I_CREATE and I_SETSTAT

SS_SIZE

OS-9 for 68K Technical Manual

OS-9 Technical Manual

OS-9Technical Manual

crtolf()

Convert Carriage Return Characters to Linefeed Characters

Syntax

```
#include <UNIX/os9def.h>
void crtolf(
    char          *buf ,
    unsigned int   n );
```

Description

`crtolf()` converts all carriage return characters to linefeed characters in `buf` for a span of `n` characters.

<code>buf</code>	is a pointer to a buffer. (Input/Output).
<code>n</code>	is the number of characters spanned. (Input)

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`unix.l`

ctime()

Convert Calendar Time to String Format

Syntax

```
#include <time.h>
char *ctime(time_t *t);
```

Description

ctime() converts the calendar time *t* into the following 26-byte string format, including the terminating \0:

xxx mmm dd hh:mm:ss yyyy\n\0

xxx is one of the following days of the week:

Sun Mon Tue Wed Thu Fri Sat

mmm is one of the following months of the year:

Jan	Feb	Mar	Apr	May	Jun
Jul	Aug	Sep	Oct	Nov	Dec

dd specifies the day of the month.

hh:mm:ss specifies the time in hours, minutes, and seconds.

yyyy specifies the year.

ctime(t) is the equivalent of
asctime(localtime(t)). (Input)



Note

ctime() returns a pointer to a static area which may be overwritten. To ensure data integrity, use the string or save it immediately.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

clib.l

See Also

[asctime\(\)](#)
[gmtime\(\)](#)
[localtime\(\)](#)

detach()

Detach Device

Syntax

```
#include <io.h>
#include <modes.h>
int detach(dev_list *ptr);
```

Description

`detach()` removes a device from the system device table if no other process is using it.

`ptr` is a pointer returned by either `attach()` or `_os_attach()`. (Input)

If an error occurs, -1 is returned and the appropriate error code is placed in the global variable `errno`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

sys_clib.l

See Also

`_os_attach()`
`_os_detach()`

I\$Detach
I_DETACH

OS-9 for 68K Technical Manual

OS-9 Technical Manual

difftime()

Find Temporal Difference

Syntax

```
#include <time.h>
double difftime(
    time_t      time1,
    time_t      time0);
```

Description

`difftime()` returns the difference in seconds between `time1` and `time0`. This value is returned as a `double`. If you could subtract the `time_t` values, the return value would be: `time1 - time0`.

`time0` and `time1` are the compared values for `difftime()`. (Input)

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

See Also

[time\(\)](#)

div()

Compute Quotient and Remainder

Syntax

```
#include <stdlib.h>
div_t div(
    int      numer,
    int      denom);
```

Description

`div()` returns the quotient and remainder of the division of the numerator by the denominator.

numer is the numerator. (Input)

denom is the denominator. (Input)

If the division is inexact, the resulting quotient is the integer of lesser magnitude that is the nearest to the algebraic quotient. If the result cannot be represented, the behavior is undefined. Otherwise, `quot * denom + rem` equals `numer`.

`div()` returns a structure of type `div_t`, comprising both the quotient and the remainder. The `div_t` structure contains the `quot` and `rem` fields.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

c1ib.l

_dto(a)

Double to ASCII Conversion

Syntax

```
#include <stdlib.h>
int _dto(a(
    double   fnum,
    char     *buff,
    int      tot_digs,
    int      frac_digs,
    int      *sign);
```

Description

`_dto(a()` converts a floating point number of type `double` to the ASCII string equivalent. The conversion terminates when:

- The requested number of digits are converted.
- The specified number of digits after the decimal point is reached.

`fnum` is the floating point number to convert.
(Input)

`buff` is a pointer to a buffer for the converted ASCII string. (Output)

`_dto(a()` returns the signed decimal exponent. The returned string only contains mantissa digits. A null byte is appended to the end of `buff`. `buff` should be one byte larger than the requested number of digits.

`tot_digs` is the number of digits requested. (Input)

`frac_digs` is the specified number of digits following the decimal point. (Input)

sign is a pointer to the sign. (Input)

Set the sign in `sign` as follows:

- 1 for a negative sign.
 - 0 for a positive sign.

If `fnum` represents $\pm\infty$ or an IEEE 754 Not a Number and `tot_digs` is greater than or equal to three, `buff` contains, respectively, `Inf` or `NaN`. If `tot_digs` is less than three, `buff` contains, respectively, `I` or `N`. In these cases, `_ dtoa()` returns `DBL_MAX_10_EXP`.

`_ dtoa()` returns no errors.

Attributes

Operating System: OS-9 and OS-9 for 68K

State: User and System

Threads: Safe

Re-entrant: Yes

Library

clib.l

dup()Duplicate Path

Syntax

```
#include <modes.h>  
  
int dup(int path);
```

Description

`dup()` returns a synonymous path number for an existing file or device. The lowest available path number is used.

`path` specifies the path number. (Input)

`dup()` increments the link count of a path descriptor and returns a different synonymous path number. The path descriptor is not cloned.

**Note**

It is usually not a good idea for more than one process to perform I/O on the same path concurrently.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`_os_dup()`

`I$Dup`

`I_DUP`

OS-9 for 68K Technical Manual

OS-9 Technical Manual

dup2()

Specifies the Value of Duplicated Path

Syntax

```
#include <UNIX/os9def.h>
int dup2(int fd1, int fd2);
```

Description

`dup2()` returns a synonymous path number for an existing file or device and specifies the value of the new duplicated path.

fd1 specifies the original path number.
(Input)

fd2 specifies the value of the duplicated path. (Input/Output)

If path `fd2` is already in use, it is first closed.

Attributes

Operating System: OS-9 and OS-9 for 68K

State: User and System

Threads: Safe

Re-entrant: No

Library

unix.1

Syntax

```
#include <stdlib.h>
extern int _memmins;
void *ebrk(unsigned int size);
```

Description

ebrk() returns the amount of memory specified by `size`. The memory is obtained from the system using the `F$SRqMem` (for OS-9 for 68K) or `_os_srqmem()` (for OS-9) system request. It is intended for general purpose memory allocation.

`size` specifies the amount of memory. (Input)

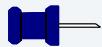
To reduce the overhead involved in requesting small quantities of memory, ebrk() requests memory from the system in a minimum size determined by the global variable `_memmins` and satisfies the user requests from this memory space. `_memmins` is initially set to 8192. ebrk() grants memory requests from this memory space provided the requests are no larger than the amount of space.

If the request is larger than the available space, ebrk() wastes the rest of the space and tries to get enough memory from the system to satisfy the request.

This method works well for programs that need to get large amounts of not necessarily contiguous memory in little bits and cannot afford the overhead of `malloc()`.

Changing the `_memmins` variable causes ebrk() to use that value as the `F$SRqMem` (for OS-9 for 68K) or `F_SRQMEM` (for OS-9) memory request size.

If the memory request is granted, a pointer (even-byte aligned) to the block is returned. If the request is not granted, -1 is returned and the appropriate error code is placed in the global variable `errno`.



Note

The memory obtained from `ebrk()` is not returned until the process terminates.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

`sys_clib.l`

See Also

[sbrk\(\)](#)
[ibrk\(\)](#)
[malloc\(\)](#)
[_os_srqmem\(\)](#)
`F$SRqMem`
`F_SRQMEM`

OS-9 for 68K Technical Manual
OS-9 Technical Manual

endpwent()

Closes Password File

Syntax

```
#include <UNIX/pwd.h>
void endpwent();
```

Description

`endpwent()` can be used to close a password file after processing is complete.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

unix.l

See Also

[fgetpwent\(\)](#)
[getpw\(\)](#)
[fgetpwent\(\)](#)
[getpwnam\(\)](#)
[getpwuid\(\)](#)
[setpwent\(\)](#)

Syntax

```
#include <stdio.h>
int _errmsg(
            int             nerr,
            const char     *msg,
            [u_int32]        arg1,
            u_int32         arg2,
            u_int32         arg3);
```

Description

`_errmsg()` displays an error message and the program name on the standard error path.

<code>nerr</code>	specifies the error number returned by <code>_errmsg().</code> (Input)
<code>msg</code>	is a pointer to the error message. (Input) The message string is displayed in the following format: <code>prog: <message text></code>
<code>prog</code>	is the module name of the program, and <code><message text></code> is the string pointed to by <code>msg</code> .
<code>[arg1-arg3]</code>	Input

For added flexibility in message printing, `msg` can be a conversion string suitable for `fprintf()` with up to three additional parameters of any integral type. `nerr` is returned as the value of the function so `_errmsg()` can be used as a parameter to a function such as `exit()` or `prerr().`

Example

Assume the program calling the function is named `foobar`:

```
Call:    errmsg(1,"programmed message\n");
Prints:  toobar: programmed message
Call:    exit(_errmsg(errno,
                     "unknown option '%c'\n",'q'));
Prints:  foobar: unknown option 'q'
```

Then exits with `errno` as its termination status.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

`sys_clib.l`

See Also

[fprintf\(\)](#)
[_prgname\(\)](#)

Syntax

```
#include <events.h>
int _ev_creat(
    int          ev_value,
    int          wait_inc,
    int          signal_inc,
    const char  *ev_name);
```

Description

`_ev_creat()` creates an event.

ev_value	is the initial value for the event. (Input)
wait_inc	is the wait increment. (Input)
signal_inc	It is added to the value of the event when a successful <code>_ev_wait()</code> , <code>_ev_waitr()</code> , <code>_os_ev_wait()</code> , or <code>_os_ev_waitr()</code> is performed.
ev_name	is the signal increment. (Input)

An event ID number is returned if the event is successfully created.

If an error occurs, -1 is returned and the appropriate error code is placed in the global variable `errno`.



For More Information

Refer to the ***OS-9 Technical Manual*** for more information about events.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

sys_clib.l

See Also

_ev_signal()	<i>OS-9 for 68K Technical Manual</i>
_ev_wait()	<i>OS-9 Technical Manual</i>
_ev_waitr()	<i>OS-9 Technical Manual</i>
_os_ev_creat()	<i>OS-9 Technical Manual</i>
_os_ev_signal()	<i>OS-9 Technical Manual</i>
_os9_ev_wait()	<i>OS-9 Technical Manual</i>
_os_ev_waitr()	<i>OS-9 Technical Manual</i>
F\$Event	
F_EVENT	
EV_CREAT	
EV_SIGNAL	
EV_WAIT	
EV_WAITR	

Syntax

```
#include <events.h>
int _ev_delete(const char *ev_name);
```

Description

`_ev_delete()` deletes an event. The use count for the event must be zero before the event can be deleted.

`ev_name` is a pointer to the name of the event.
(Input)

If an error occurs, -1 is returned and the appropriate error code is placed in the global variable `errno`.



For More Information

Refer to your operating system technical manual for more information about events.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

[_os_ev_delete\(\)](#)

`F$Event`

`F_EVENT, EV_DELETE`

OS-9 for 68K Technical Manual

OS-9 Technical Manual

Syntax

```
#include <events.h>
int _ev_info(
    int      ev_index,
    int      ev_infostr,
    int      *ev_buffer);
```

Description

`_ev_info()` returns information about an event. The event table is indexed from zero to one less than the maximum number of events allowed on the system.

ev_index	corresponds to the starting point in the event table to search for an event. (Input)
ev_infostr	Input
ev_buffer	is a pointer to the event structure buffer used to hold the event information, if found. (Output)

If `ev_index` is greater than all active events in the table, -1 is returned and the appropriate error code is placed in the global variable `errno`.



For More Information

Refer to your operating system technical manual for more information about events.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`_os_ev_info()`

`F$Event`

`F_EVENT, EV_INFO`

OS-9 for 68K Technical Manual

OS-9 Technical Manual

Syntax

```
#include <events.h>
int _ev_link(const char *ev_name);
```

Description

`_ev_link()` links to an existing event. An event ID number is returned if the event is successfully linked.

`ev_name` is a pointer to the name of the event.
(Input)

If an error occurs, -1 is returned and the appropriate error code is placed in the global variable `errno`.



For More Information

Refer to your operating system technical manual for more information about events.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

[_os_ev_link\(\)](#)

F\$Event

F_EVENT, EV_LINK

OS-9 for 68K Technical Manual

OS-9 Technical Manual

Syntax

```
#include <events.h>
int _ev_pulse(
    int      ev_id,
    int      ev_value,
    int      allflag);
```

Description

`_ev_pulse()` indicates that an event has occurred. The current event value is saved. The event variable is set to the value given by `ev_value`, and the normal signal increment is not applied. The saved event value is restored after activating processes, if any.

<code>ev_id</code>	is the event ID returned from a call to <code>_ev_creat()</code> , <code>_ev_link()</code> , <code>_os_ev_creat()</code> , or <code>_os_ev_link()</code> . (Input)
<code>ev_value</code>	sets the value of the event variable. (Input)
<code>allflag</code>	indicates which process(es) to activate. (Input) Only the lower 16 bits of <code>allflag</code> are used. <ul style="list-style-type: none">• If <code>allflag</code> is zero, the first process waiting for the event is activated.• If <code>allflag</code> is <code>0x8000</code>, all processes waiting for the event that have a value in range are activated.

If an error occurs, `-1` is returned and the appropriate error code is placed in the global variable `errno`. Refer to your operating system technical manual for more information about events.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

sys_clib.l

See Also

[_ev_create\(\)](#)
[_ev_link\(\)](#)
[_os_ev_create\(\)](#)
[_os_ev_link\(\)](#)
[_os_ev_pulse\(\)](#)

F\$Event
F_EVENT
V_CREAT
EV_LINK
EV_PULSE

OS-9 for 68K Technical Manual
OS-9 Technical Manual
OS-9 Technical Manual
OS-9 Technical Manual
OS-9 Technical Manual

_ev_read()

Read Event Without Waiting

Syntax

```
#include <events.h>
int _ev_read(int ev_id);
```

Description

`_ev_read()` reads the value of an event, without waiting for or affecting the event variable, and returns the current value of the event.

`ev_id` identifies the event. (Input)

If an error occurs, -1 is returned and the appropriate error code is placed in the global variable `errno`. Clear the global variable `errno` prior to calling `_ev_read()` to distinguish between return of an event value of -1 and an error.



For More Information

Refer to your operating system technical manual for more information about events.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

Example

```
errno = 0;
if ((ev_val = _ev_read(ev_id)) == -1 && errno)
    exit(_errmsg(errno,
        "can't read event value - "));
```

See Also

[_os_ev_read\(\)](#)

F\$Event

F_EVENT

EV_READ

OS-9 for 68K Technical Manual

OS-9 Technical Manual

OS-9 Technical Manual

Syntax

```
#include <events.h>
int _ev_set(
    int      ev_id,
    int      ev_value,
    int      allflag);
```

Description

`_ev_set()` indicates that an event has occurred. The event variable is set to the value given by `ev_value`, and the normal signal increment is not applied. Processes waiting for the event are then activated.

<code>ev_id</code>	is the event ID returned from a call to <code>_ev_creat()</code> , <code>_ev_link()</code> , <code>_os_ev_creat()</code> , or <code>_os_ev_link()</code> . (Input)
<code>ev_value</code>	sets the value of the event variable. (Input)
<code>allflag</code>	specifies the process(es) to activate. (Input) Only the lower 16 bits of <code>allflag</code> are used. <ul style="list-style-type: none">• If <code>allflag</code> is zero, the first process waiting for the event is activated.• If <code>allflag</code> is 0x8000, all processes waiting for the event that have a value in range are activated.

OS-9 for 68K: `_ev_set()` returns the value of the event before changing to `ev_value`. If an error occurs, -1 is returned and the appropriate error code is placed in the global variable `errno`. Clear the global variable `errno` prior to calling `_ev_set()` to distinguish between return of a prior event value of -1 and an error.

OS-9: `_ev_set()` returns zero. If an error occurs, -1 is returned and the appropriate error code is placed in the global variable `errno`.



For More Information

Refer to your operating system technical manual for more information about events.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

[`_ev_creat\(\)`](#)
[`_ev_link\(\)`](#)
[`_os_ev_creat\(\)`](#)
[`_os_ev_link\(\)`](#)
[`_os_ev_set\(\)`](#)

`F$Event`

`F_EVENT`

`EV_CREAT`

`EV_LINK`

`EV_SET`

OS-9 for 68K Technical Manual

OS-9 Technical Manual

OS-9 Technical Manual

OS-9 Technical Manual

OS-9 Technical Manual

Syntax

```
#include <events.h>
int _ev_setr(
    int      ev_id,
    int      ev_value,
    int      allflag);
```

Description

`_ev_setr()` indicates that an event has occurred. The event variable is incremented by the value given by `ev_value`. Processes waiting for the event are then activated.

<code>ev_id</code>	is the event ID returned from a call to <code>_ev_creat()</code> , <code>_ev_link()</code> , <code>_os_ev_creat()</code> , or <code>_os_ev_link()</code> . (Input)
<code>ev_value</code>	specifies the amount to increment the event variable. (Input)
<code>allflag</code>	specifies the process(es) to activate. (Input) Only the lower 16 bits of <code>allflag</code> are used. <ul style="list-style-type: none">• If <code>allflag</code> is zero, the first process waiting for the event is activated.• If <code>allflag</code> is 0x8000, all processes waiting for the event that have a value in range are activated.

OS-9 for 68K: `_ev_setr()` returns the value of the event before changing by `ev_value`. If an error occurs, -1 is returned and the appropriate error code is placed in the global variable `errno`. Clear the global variable `errno` prior to calling `_ev_setr()` to distinguish between return of a prior event value of -1 and an error.

OS-9: `_ev_setr()` returns 0. If an error occurs, -1 is returned and the appropriate error code is placed in the global variable `errno`.



For More Information

Refer to your operating system technical manual for more information about events.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

[`_ev_creat\(\)`](#)
[`_ev_link\(\)`](#)
[`_os_ev_creat\(\)`](#)
[`_os_ev_link\(\)`](#)
[`_os_ev_setr\(\)`](#)

`F$Event`

`F_EVENT`

`EV_CREAT`

`EV_LINK`

`EV_SETR`

OS-9 for 68K Technical Manual

OS-9 Technical Manual

OS-9 Technical Manual

OS-9 Technical Manual

OS-9 Technical Manual

Syntax

```
#include <events.h>
int _ev_signal(
    int      ev_id,
    int      allflag);
```

Description

`_ev_signal()` indicates that an event has occurred. The current event variable is updated by the signal increment which was given when the event was created.

ev_id	is the event ID returned from a call to <code>_ev_creat()</code> , <code>_ev_link()</code> , <code>_os_ev_creat()</code> , or <code>_os_ev_link()</code> . (Input)
allflag	specifies the process(es) to activate. (Input) Only the lower 16 bits of allflag are used. <ul style="list-style-type: none">• If allflag is zero, the first process waiting for the event is activated.• If allflag is 0x8000, all processes waiting for the event that have a value in range are activated.

OS-9 for 68K: `_ev_signal()` returns the value of the event before incremented by the signal increment. If an error occurs, -1 is returned and the appropriate error code is placed in the global variable `errno`. Clear the global variable `errno` prior to calling `_ev_signal()` to distinguish between return of a prior event value of -1 and an error.

OS-9: `_ev_signal()` returns zero. If an error occurs, -1 is returned and the appropriate error code is placed in the global variable `errno`.



For More Information

Refer to your operating system technical manual for more information about events.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

sys_clib.l

See Also

[_ev_create\(\)](#)
[_ev_link\(\)](#)
[_os_ev_create\(\)](#)
[_os_ev_link\(\)](#)
[_os_ev_signal\(\)](#)

F\$Event

F_EVENT

EV_CREAT

EV_LINK

EV_SIGNAL

OS-9 for 68K Technical Manual

OS-9 Technical Manual

_ev_unlink()

Unlink Event

Syntax

```
#include <events.h>
int _ev_unlink(int ev_id);
```

Description

`_ev_unlink()` decrements the link count of an event. When the link count becomes zero, you can delete the event using `_ev_delete()` or `_os_ev_delete()`.

`ev_id` specifies the event ID.

If an error occurs, -1 is returned and the appropriate error code is placed in the global variable `errno`.



For More Information

Refer to your operating system technical manual for more information about events.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

[_ev_delete\(\)](#)
[_os_delete\(\)](#)
[_os_ev_unlink\(\)](#)

F\$Event
F_EVENT
EV_DELETE
EV_UNLINK

OS-9 for 68K Technical Manual
OS-9 Technical Manual
OS-9 Technical Manual
OS-9 Technical Manual

Syntax

```
#include <events.h>
int _ev_wait(
    int          ev_id,
    int          ev_min,
    int          ev_max);
```

Description

`_ev_wait()` waits for an event to occur. The event value is compared to the range values specified by `ev_min` and `ev_max`. If the event value is not in the specified range, the process waits until some other process places the value within the range. Once in range, the wait increment is applied to the event value. The actual event value is returned as the value of the function.

<code>ev_id</code>	specifies the event ID. (Input)
<code>ev_min</code>	specifies the minimum range value for the event. (Input)
<code>ev_max</code>	specifies the maximum range value for the event. (Input)

`_ev_wait()` returns the value of the event after completion of the wait. If an error occurs, -1 is returned and the appropriate error code is placed in the global variable `errno`. Clear the global variable `errno` prior to calling `_ev_wait()` to distinguish between return of a current event value of -1 and an error. Further, if a signal was received by the process, the event value returned is outside of the range `ev_min` to `ev_max`.

```
errno = 0;
if ((ev_val = _ev_wait(ev_id, -5, 0)) == -1 && errno)
    exit(_errormsg(errno,
                    "error waiting on event - "));
if (ev_val < -5 || ev_val > 0)
    _errormsg(1,
                "received signal while waiting for event\n");
```



For More Information

Refer to your operating system technical manual for more information about events.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

sys_clib.l

See Also

[_os_ev_wait\(\)](#)
[_os9_ev_wait\(\)](#)
F\$Event
F_EVENT
EV_WAIT

OS-9 for 68K Technical Manual
OS-9 Technical Manual
OS-9 Technical Manual

Syntax

```
#include <events.h>
int _ev_waitr(
    int          ev_id,
    int          ev_min,
    int          ev_max);
```

Description

`_ev_waitr()` waits for the specified event to occur. The event value is compared to the relative values specified by `ev_min` and `ev_max`. The current event value is added to the range values before the comparison. If the event value is not in the specified range, the process waits until some other process places the value within the range. Once in range, the wait increment is applied to the event value. The actual event value is returned as the value of the function.

`ev_id` specifies the event ID for the event.
(Input)

`ev_min` specifies the minimum range value for
the event. (Input)

`ev_max` specifies the maximum range value for
the event. (Input)

`_ev_waitr()` returns the value of the event after completion of the wait. If an error occurs, -1 is returned and the appropriate error code is placed in the global variable `errno`. Clear the global variable `errno` prior to calling `_ev_waitr()` to distinguish between return of a current event value of -1 and an error. Further, if a signal was received by the process, the event value returned is outside of the relative range `ev_min` to `ev_max`.



For More Information

Refer to your operating system technical manual for more information about events.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

sys_clib.l

See Also

[_os_ev_waitr\(\)](#)

F\$Event

F_EVENT, EV_WAITR

OS-9 for 68K Technical Manual

OS-9 Technical Manual

Syntax

```
#include <UNIX/os9def.h>
int execl(
    const char      *path,
    const char      *arg0,
    ...);
```

Description

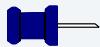
`execl()` is the UNIX-like exec interface to the `_os_chain()` system call.

`path` is the pathlist to the program on which to chain. (Input)

`arg0` (input), and any additional variable arguments, are the arguments to pass to the chained process. `arg0` is conventionally the name of the chained program. The list of variable arguments is terminated by a NULL entry.

The global variable `_environ` is used for the environment for the call to `_os_chain()`.

`execl()` returns -1 and sets the global variable `errno` to indicate an error. `execl()` only returns an error if setting up for the `_os_chain()` call fails. Once an attempt to chain occurs, the process exits if any error occurs.



Note

`_os_chain()` does not do the normal PATH searching that the shell does.

A threaded process may only chain from the `main()` thread and may not chain if multiple threads are active in the process.

To use `execl()` link with `unix.l`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

`unix.l`

See Also

`execle()`
`execv()`
`execvp()`

Syntax

```
#include <UNIX/os9def.h>
int execle(
            const char    *path,
            const char    *arg0,
            ...,
            const char    *envp[ ] );
```

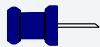
Description

`execle()` is the UNIX-like exec interface to the `_os_chain()` system call.

path	is the pathlist to the program on which to chain. (Input)
arg0	is conventionally the name of the chained program. (Input)
arg0, and any additional variable arguments, are the arguments to pass to the chained process.	

The list of variable arguments is terminated by a NULL entry. The NULL entry is followed by an environment pointer to use for the call to `_os_chain()`. `envp` is a pointer to an array of strings that constitute the environment of the process.

`execle()` returns -1 and sets the global variable `errno` to indicate an error. `execle()` only returns an error if setting up for the `_os_chain()` call fails. Once an attempt to chain occurs, the process exits if any error occurs.



Note

`_os_chain()` does not do the normal PATH searching that the shell does.

A threaded process may only chain from the `main()` thread and may not chain if multiple threads are active in the process.

To use `execl()` link with `unix.l`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

`unix.l`

See Also

[execl\(\)](#)
[execv\(\)](#)
[execvp\(\)](#)

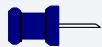
Syntax

```
#include <UNIX/os9def.h>
int execv(
            const char      *path,
            char           *const argv[]);
```

Description

`execv()` is a UNIX-like interface to the `_os_chain()` system call. The argument to `execv()` is an argument vector to pass to the chained process and the path to the program on which to chain. The last argument should be a `NULL` entry.

`argv[0]` is the name of the program being chained to. (Input)



Note

`_os_chain()` does not do the normal PATH searching that the shell does.

A threaded process may only chain from the `main()` thread and may not chain if multiple threads are active in the process.

To use `execv()` link with `unix.l`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

unix.l

See Also

[execl\(\)](#)

[execle\(\)](#)

[execvp\(\)](#)

Syntax

```
#include <UNIX/os9def.h>
int execv(
    const char *path,
    char      *const argv[ ],
    char      *const envp[ ]);
```

Description

`execve()` is a UNIX-like interface to the `_os_chain()` system call. The argument to `execve()` is an argument vector to pass to the chained process and the path to the program on which to chain. The last argument should be a `NULL` entry.

path	is the pathlist to the program on which to chain. (Input)
argv[0]	is the name of the program being chained to. (Input)
envp	is a pointer to a null-terminated array of character pointers to null-terminated strings. (Input)



Note

`_os_chain()` does not do the normal PATH searching that the shell does.

A threaded process may only chain from the `main()` thread and may not chain if multiple threads are active in the process.

To use `execv()` link with `unix.l`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

unix.l

See Also

[execl\(\)](#)
[execle\(\)](#)
[execvp\(\)](#)

Syntax

```
#include <UNIX/os9def.h>
int execvp(
            const char    *file,
            char          *const argv[ ]);
```

Description

`execvp()` is a UNIX-like interface to the `_os_chain()` system call. The argument to `execvp()` is an argument vector to pass to the chained process and the path to the program to chain to. The last argument should be a NULL entry.

file	Input
argv[0]	is the name of the program being chained to. (Input)



Note

`_os_chain()` does not do the normal PATH searching that the shell does.

A threaded process may only chain from the `main()` thread and may not chain if multiple threads are active in the process.

To use `execvp()` link with `unix.l`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

unix.l

See Also

[execl\(\)](#)

[execle\(\)](#)

[execv\(\)](#)

Syntax

```
#include <stdlib.h>
void _exit(int status);
```

Description

`_exit()` immediately terminates a program.

`status` is the exit status. (Input)

The parent process can use `status` to determine the program's success or failure.

An exit status of zero is considered normal termination. Most programs (especially the shell) interpret a non-zero value as an error code.

`_exit()` cannot return to its caller, and it does not flush and close standard I/O buffers.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

[_os_exit\(\)](#)

`F$Exit`

`F_EXIT`

OS-9 for 68K Technical Manual

OS-9 Technical Manual

Syntax

```
#include <stdlib.h>
void exit(int status);
```

Description

`exit()` is the normal means of terminating a task. All functions specified by `atexit()` are called in the reverse order of the order in which they were specified.

All open streams with unwritten buffered data are flushed, all open streams are closed, and all files created by `tmpfile()` are removed. Control is returned to the host environment.

`status` is the exit status. (Input)

- If `status` is zero or `EXIT_SUCCESS`, zero is communicated to the parent process executing the `_os_wait()`.
- If `status` is `EXIT_FAILURE`, one is communicated.

`exit()` cannot return to its caller.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`clib.lib`

See Also

[atexit\(\)](#)
[_os_exit\(\)](#)
[_os_wait\(\)](#)
[tmpfile\(\)](#)
[F\\$Exit](#)
[F_EXIT](#)

OS-9 for 68K Technical Manual

OS-9 Technical Manual

Syntax

```
#include <math.h>
double exp(double x);
```

Description

`exp()` returns the exponential function of `x` (input), e ($2.71828..$) raised to the `x` power. A range error, `ERANGE` stored in `errno`, occurs if the magnitude of `x` is too large. In this case, `exp()` returns `HUGE_VAL`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

Possible Error

`ERANGE`

Syntax

```
#include <math.h>
double fabs(double x);
```

Description

`fabs()` returns the absolute value of `x` (input).

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

Syntax

```
#include <stdio.h>
int fclose(FILE *stream);
```

Description

`fclose()` flushes the file pointed to by `stream` and closes the associated file. Any unwritten buffered data for the stream are written to the file; any unread buffered data are discarded. The stream is disassociated from the file. If the associated buffer was automatically allocated, it is deallocated. If `stream` was returned from `tmpfile()`, it is deleted.

`stream` is a pointer to the C I/O FILE structure.
(Input)

`fclose()` returns zero if the stream was successfully closed. Otherwise, EOF is returned.



For More Information

The `sclib.l` and `sclib.il` libraries contain a smaller version of the `fclose()` function. Refer to the ***Using Ultra C/C++*** manual for more information about the small library functions.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

clib.l

See Also

[fflush\(\)](#)
[fopen\(\)](#)
[getc\(\), getchar\(\)](#)
[putc\(\), putchar\(\)](#)
[tmpfile\(\)](#)

Syntax

```
#include <stdio.h>
FILE *fdopen(
            int          path,
            const char  *action);
```

Description

`fdopen()` returns a file pointer to the file specified by a currently open path.

<code>path</code>	is the path to attach. (Input)
<code>action</code>	is a pointer to the file action string. (Input)
	The <code>action</code> must be compatible with the access mode of the given path. The valid actions are listed in Table 2-4 .

Table 2-4 Valid Actions

Action	Description
r	Open for reading
w	Open for writing
a	Append (write) at the end of the file or create file for writing if <code>path</code> does not exist
r+	Open for update
w+	Create for update

Table 2-4 Valid Actions (continued)

Action	Description
a+	Create or open for update at end of file
d	Directory read

Use `fdopen()` when you require some special processing not provided by `fopen()` for opening files.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

`sys_clib.l`

See Also

[fopen\(\)](#)

[freopen\(\)](#)

feof()

Check Buffered File for End of File

Syntax

```
#include <stdio.h>
int feof(FILE *stream);
```

Description

`feof()` tests the end-of-file indicator for the specified stream. `feof()` returns a non-zero value only if the end-of-file indicator is set for stream.

stream is a pointer to the C I/O FILE structure.
(Input)

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

clib.l

See Also

```
clearerr()
fopen()
```

ferror()

Check Buffered File for Error Condition

Syntax

```
#include <stdio.h>
int ferror(FILE *stream);
```

Description

`ferror()` tests the error indicator for the specified stream. `ferror()` returns a non-zero value only if the error indicator is set for `stream`.

<code>stream</code>	is a pointer to the C I/O FILE structure. (Input)
---------------------	--

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

See Also

[clearerr\(\)](#)

[fopen\(\)](#)

Syntax

```
#include <stdio.h>
int fflush(FILE *stream);
```

Description

`fflush()` writes any unwritten data for the specified stream to the file. Unwritten data is written if the stream was opened in either `write` or `update` mode.

`stream` is a pointer to the C I/O `FILE` structure. (Input)

If `stream` is a null pointer, `fflush()` performs this flushing action on all streams opened in either `write` or `update` mode and containing unwritten data.

`fflush()` returns `EOF` if a write error occurs or if the stream is read-only. Otherwise, it returns zero.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`clib.l`

See Also

[fopen\(\)](#)
[getc\(\)](#), [getchar\(\)](#)
[putc\(\)](#), [putchar\(\)](#)

Syntax

```
#include <UNIX/os9def.h>
int ffs(unsigned int i);
```

Description

`ffs()` finds the first bit set in the argument passed to it and returns the index of that bit. Bits are numbered starting at one (1) from the least significant. A return value of zero indicates that the value passed to the function was zero.

i	Input
---	-------

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

unix.l

Syntax

```
#include <stdio.h>
int fgetc(FILE *stream);
```

Description

`fgetc()` returns a character, as an unsigned converted to an `int`, from the file and advances the associated file position indicator for the stream, if defined.

<code>stream</code>	is a pointer to the C I/O <code>FILE</code> structure. (Input)
	If the stream is at end-of-file, the end-of-file indicator for the stream is set and <code>fgetc()</code> returns <code>EOF</code> . If a read error occurs, the error indicator for the stream is set and <code>fgetc()</code> returns <code>EOF</code> .

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`clib.l`

See Also

`getc()`, `getchar()`

fgetpos()

Get Current Position in File

Syntax

```
#include <stdio.h>
int fgetpos(
    FILE      *stream,
    fpos_t    *pos);
```

Description

`fgetpos()` stores the current value of the file position indicator. The stored value contains information usable by `fsetpos()` for repositioning the stream to its position at the time of the `fgetpos()` call.

<code>stream</code>	is a pointer to the C I/O <code>FILE</code> structure. (Input)
<code>pos</code>	is a pointer to the file position indicator. (Output)

If successful, `fgetpos()` returns zero. Otherwise, it returns a non-zero value and the appropriate error code is placed in the global variable `errno`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`clib.l`

See Also

[fsetpos\(\)](#)

fgetpwent()

Get Password File Entry

Syntax

```
#include <UNIX/pwd.h>
struct passwd *fgetpwent(FILE *f);
```

Description

fgetpwent() returns a pointer to the next password structure in the stream (f), which matches the format of the password file /dd/SYS/password.

f Input

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

unix.l

See Also

[endpwent\(\)](#)
[getpw\(\)](#)
[getpwent\(\)](#)
[getpwnam\(\)](#)
[getpwuid\(\)](#)
[setpwent\(\)](#)

Syntax

```
#include <stdio.h>
char *fgets(
            char      *ptr,
            int       cnt,
            FILE     *stream);
```

Description

`fgets()` reads at most one less than the specified number of characters from the stream into an array. No additional characters are read after a newline character (which is returned) or after end-of-file. A null character is added to the end of the string.

<code>ptr</code>	is a pointer to the array into which to read the characters. (Output)
<code>cnt</code>	is the number of characters to read. (Input)
<code>stream</code>	is a pointer to the C I/O <code>FILE</code> structure. (Input)

`fgets()` returns `ptr` if successful. If the end-of-file is encountered and no characters have been read into the array, the contents of the array remain unchanged and a null pointer is returned. If a read error occurs during the operation, the array contents are indeterminate and a null pointer is returned.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

clib.l

See Also

[fgetc\(\)](#)

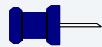
Syntax

```
#include <stdio.h>
int _fileno(FILE *stream);
int fileno(FILE *stream);
```

Description

`_fileno()` returns the path number associated with the specified file pointer. The path number is not valid unless (`stream->_flag & _INIT`) is non-zero. This call is only available as a macro.

`stream` is a pointer to the C I/O FILE structure.
(Input)



Note

If you are executing the compiler in strictly conforming ANSI mode, only `_fileno()` is available. Otherwise, both calls are available.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

These macros are defined in the `stdio.h` header file.

See Also

`fopen()`
`freopen()`

Syntax

```
#include <strings.h>
int findnstr(
    int          pos,
    const char  *string,
    const char  *pattern,
    int          len);
```

Description

`findnstr()` searches the string for the first occurrence of the pattern. It starts at position `pos` (where the first position is one, not zero). The returned value is the position of the first matched character of the pattern in the string, or zero if a match is not found. `findnstr()` stops searching at `len`; it may continue past null bytes.

<code>pos</code>	specifies the starting position. (Input)
<code>string</code>	is a pointer to the string to search. (Input)
<code>pattern</code>	is a pointer to the pattern to use for the search. (Input)
<code>len</code>	specifies the final position to be searched. (Input)



Note

The current implementation does not use the most efficient pattern matching algorithm. `strstr()` is preferred for very long strings.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`findstr()`
`strstr()`

Syntax

```
#include <strings.h>
int findstr(
    int          pos,
    const char *string,
    const char *pattern);
```

Description

`findstr()` searches the string for the first instance of the pattern. It starts at position `pos` (where the first position is one, not zero). The returned value is the position of the first matched character of the pattern in the string, or zero if a match is not found. `findstr()` stops searching when a null byte is found in `string`.

<code>pos</code>	specifies the starting position. (Input)
<code>string</code>	is a pointer to the string to search. (Input)
<code>pattern</code>	is a pointer to the pattern to use for the search. (Input)



Note

The current implementation does not use the most efficient pattern matching algorithm. `strstr()` is preferred for very long strings.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`findnstr()`
`strstr()`

Syntax

```
#include <math.h>
double floor(double x);
```

Description

`floor()` returns the largest integer, as a double, that is not greater than `x` (input).

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.lib`

See Also

[ceil\(\)](#)

Syntax

```
#include <math.h>
double fmod(
    double      x,
    double      y);
```

Description

`fmod()` computes the floating-point remainder of `x` (input) divided by `y` (input). If `y` is non-zero, `fmod()` returns the value with the same sign as `x` and a magnitude less than the magnitude of `y`. If `y` is zero, zero is returned.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

Possible Error

`EDOM`

Syntax

```
#include <stdio.h>
FILE *fopen(
            const char      *name,
            const char      *mode);
```

Description

`fopen()` opens a file and associates a stream with it. If an error occurs, `fopen()` returns NULL.

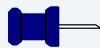
name	is a pointer to the name of the file. (Input)
mode	is a pointer to the access mode. (Input)
	mode may be one of the following:

Table 2-5 fopen() Mode types

Mode	Description
r	Open text file for reading
w	Truncate to zero length or create text file for writing
a	Append; open or create text file for writing at end-of-file
rb	Open binary file for reading
wb	Truncate to zero length or create binary file for writing
ab	Append; open or create binary file for writing at end-of-file
r+	Open text file for update (reading and writing)

Table 2-5 fopen() Mode types (continued)

Mode	Description
w+	Truncate to zero length or create text file for update
a+	Append; open or create text file for update, writing at end-of-file
r+b, rb+	Open binary file for update (reading and writing)
w+b, wb+	Truncate to zero length or create binary file for update (reading and writing).
a+b, ab+	Append; open or create binary file for update, writing at end-of-file
d	Directory read (OS-9 extension); this is only available in K & R source mode.

**Note**

Appending an `x` to any action(s) indicates that names are relative to the current execution directory. The `x` also implies that the file should have the execute permission bit set. This option is only available for OS-9 in non-ANSI mode.

**Note**

In modes other than compatibility mode, appending a `d` to any action(s) indicates that the file is a directory.

mode must point to a string and not a character; `fopen("fun" , "r")` is correct, `fopen("fun" , 'r')` is not correct.

If the file does not exist or cannot be read, opening a file with read mode fails.

When you open a file with append mode, the file position indicator is initially positioned at the beginning of the file. However, all subsequent writes to the file are forced to the then current end-of-file, regardless of intervening calls to `fseek()`. If an error occurs when opening a file with append mode, refer to **fopen Append Bit** subsection in the appropriate processor-specific chapter of the *Ultra C/C++ Processor Guide*.

When a file is opened with update mode, both input and output may be performed on the associated stream. However, output may not be directly followed by input without an intervening call to `fflush()` or to a file positioning function (`fseek()`, `fsetpos()`, or `rewind()`). Likewise, input may not be directly followed by output without an intervening call to a file positioning function, unless the input operation encounters end-of-file.

When opened, a stream is fully buffered only if it does not refer to an interactive device. The error and end-of-file indicators for the stream are cleared.

Three file pointers are available and are considered open the moment the program runs:

<code>stdin</code>	standard input	I/O is to path 0
<code>stdout</code>	standard output	I/O is to path 1
<code>stderr</code>	standard error	I/O is to path 2

These macros are defined in the `stdio.h` header file.



Note

You cannot open a directory for writing with `fopen()`.

Example

Examples using `fopen()` are found in **Table 2-6**.

Table 2-6 `fopen()` Examples

<code>fp = fopen(fred,wx);</code>	Creates a file for writing in the execution directory
<code>fp = fopen(moe,r);</code>	Opens an existing file for reading
<code>fp = fopen(stooge/curly,w+);</code>	Creates a file for reading and writing

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`clib.l`

See Also

`creat()`
`fflush()`
`fseek()`
`getc()`, `getchar()`
`open()`
`_os_create()`
`_os9_create()`
`_os_open()`
`putc()`, `putchar()`

Syntax

```
#include <stdio.h>
int fprintf(
            FILE           *stream,
            const char     *format, ...);
```

Description

`printf()`, `fprintf()`, and `sprintf()` are standard C library functions that perform formatted output. Each of these functions converts, formats, and prints any parameters as indicated by the control string.

`fprintf()` writes output to a stream.

stream	is a pointer to the C I/O <code>FILE</code> structure. (Output)
format	is a pointer to a control string. (Input) <code>format</code> determines the format, type, and number of parameters that the function expects. If the control string does not correctly match the parameters, the results are not predictable.

`fprintf()` returns when the end of the format string is encountered. A format specification consists of a percent (%) character followed by (in this order):

1. Zero or more flags that modify the meaning of the conversion specification. You may use the following flags in any order:

Table 2-7 `fprintf()` Flags

Flag	Description
-	The result of the conversion is left-justified within the field. If this flag is not specified, the result is right-justified.
+	The result of a signed conversion always begins with a plus or minus sign. If this flag is not specified, the result begins with a sign only when a negative value is converted.
space	If the first character of a signed conversion is not a sign or if a signed conversion results in no characters, a space is prefixed to the result. If the space and + flags both appear, the space flag is ignored.
#	<p>The result is to be converted to an alternate form. For o conversion, it increases the precision to force the first digit of the result to zero.</p> <p>For x (or X) conversion, a non-zero result has a 0x (or 0X) prefix.</p> <p>For e, E, f, g, and G conversions, the result always contains a decimal point character, even if no digits follow. Normally, a decimal point character appears in the result of these conversions only if a digit follows it. For g and G conversions, trailing zeros are not removed from the result.</p> <p>For other conversions, the behavior is undefined.</p>
0	For d, i, o, u, x, X, e, E, f, g, and G conversions, leading zeros, following any indication of sign or base, pad the field width; no space padding is performed. If the 0 and - flags both appear, the 0 flag is ignored. For d, i, o, u, x, and X conversions, the 0 flag is ignored if a precision is specified. For other conversions, the behavior is undefined.

2. An optional minimum field width. If the converted value has fewer characters than the field width, it is padded with spaces (by default) on the left (or right, if the left adjustment flag has been given) to the field width. The field width takes the form of an asterisk (*) or a decimal integer.
3. An optional precision that gives:
 - The minimum number of digits to appear for the d, i, o, u, x, and X conversions.
 - The number of digits to appear after the decimal point character for e, E, and f conversions.
 - The maximum number of significant digits for the g and G conversions.
 - The maximum number of characters to be written from a string in s conversion.

The precision takes the form of a period (.) followed either by an asterisk (*) or by an optional decimal integer. If only the period is specified, the precision is taken as zero. If a precision appears with any other conversion specifier, the behavior is undefined.

4. An optional h, l, or L that specifies:

Table 2-8 fprintf() Optional flag descriptions

Flag	Description
h	The following d, i, o, u, x, or X conversion specifier applies to a short int or an unsigned short int parameter, or the following n conversion specifier applies to a pointer to a short int parameter.
l	The following d, i, o, u, x, or X conversion specifier applies to a long int or an unsigned long int parameter, or the following n conversion specifier applies to a pointer to a long int parameter.
L	The following e, E, f, g, or G conversion specifier applies to a long double parameter. The following d, i, o, u, x or X conversion specifier applies to a long long int parameter.

If an `h`, `l`, or `L` appears with any other conversion specifier, the behavior is undefined.

5. A character that specifies the type of conversion to apply.

An asterisk (`*`) may indicate the field width, precision, or both. In this case, an `int` parameter supplies the field width or precision. The parameters specifying field width, precision, or both appear in that order before the parameter, if any, to convert.

A negative field width parameter is taken as a `-` flag followed by a positive field width.

A negative precision parameter is taken as if the precision was omitted.

The conversion specifiers and their meanings are:

Table 2-9 `fprintf()` Conversion Specifiers

Flag	Description
<code>c</code>	The <code>int</code> parameter is converted to an <code>unsigned char</code> , and the resulting character is written.
<code>d</code> , <code>i</code>	The <code>int</code> parameter is converted to a signed decimal in the style <code>[-]dddd</code> . The precision specifies the minimum number of digits to appear. If the value being converted can be represented in fewer digits, it is expanded with leading zeros. The default precision is 1. The result of converting a zero value with a precision of zero is no characters.
<code>e</code> , <code>E</code>	The <code>double</code> parameter is converted in the style <code>[-]d.ddde±dd</code> , where there is one digit before the decimal point character (which is non-zero if the parameter is non-zero) and the number of digits after the decimal point character is equal to the precision. The default precision is 6. If the precision is zero and the <code>#</code> flag is not specified, no decimal point character appears. The value is rounded to the appropriate number of digits. The <code>E</code> conversion specifier produces a number with <code>E</code> instead of <code>e</code> introducing the exponent. The exponent always contains at least two digits. If the value is zero, the exponent is zero.

Table 2-9 `fprintf()` Conversion Specifiers (continued)

Flag	Description
f	The double parameter is converted to decimal notation in the style [-]ddd.ddd. The number of digits after the decimal point character is equal to the precision specification. The default precision is 6. If the precision is zero and the # flag is not specified, no decimal point character appears. If a decimal point character appears, at least one digit appears before it. The value is rounded to the appropriate number of digits.
g, G	The double parameter is converted in the style f or e (or in style E in the case of a G conversion specifier), with the precision specifying the number of significant digits. If the precision is zero, it is taken as one. The style used depends on the value converted; style e (or E) is used only if the exponent resulting from the conversion is less than -4 or greater than or equal to the precision. Trailing zeros are removed from the fractional portion of the result. A decimal point character appears only if it is followed by a digit.
n	The parameter is a pointer to an integer into which is written the number of characters written to the output stream so far by this call to <code>fprintf()</code> . No parameter is converted.
o, u, x, X	The unsigned int parameter is converted to unsigned octal (o), unsigned decimal (u), or unsigned hexadecimal notation (x or X) in the style dddd. The letters abcdef are used for x conversion and letters ABCDEF for X conversion. The precision specifies the minimum number of digits to appear. If the value being converted can be represented in fewer digits, it is expanded with leading zeros. The default precision is one. The result of converting a zero value with a precision of zero is no characters.
p	The parameter is a pointer to a void. The pointer value is converted to a sequence of printable characters as an unsigned hexadecimal integer.

Table 2-9 `fprintf()` Conversion Specifiers (continued)

Flag	Description
s	The parameter is a pointer to an array of character type. Characters from the array are written up to, but not including, a terminating null character. If the precision is specified, no more than that many characters are written. If the precision is not specified or is greater than the size of the array, the array contains a null character.
%	A % is written. No parameter is converted. The complete conversion specification is %%.

If a conversion specification is invalid, the behavior is undefined. If any parameter is, or points to, a union or an aggregate (except for an array of character type using %s conversion or a pointer using %p conversion), the behavior is undefined. A non-existent or small field width never truncates a field. If the result of a conversion is wider than the field width, the field expands to contain the conversion result. `fprintf()` returns the number of characters transmitted, or a negative value if an output error occurred.



For More Information

The `sclib.l` and `sclib.il` libraries contain a smaller version of the `fprintf()` function. Refer to the ***Using Ultra C/C++*** manual for more information about the small library functions.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

clib.l

See Also

[printf\(\)](#)

[sprintf\(\)](#)

Syntax

```
#include <stdio.h>
int fputc(
    int      chr,
    FILE    *stream);
```

Description

`fputc()` converts a character to an `unsigned char`. It writes the character to the output stream at the position indicated by the associated file position indicator for the stream, if defined, and advances the indicator appropriately. If the file cannot support positioning requests or if the stream was opened with append mode, the character is appended to the output stream.

`chr` is the character to convert. (Input)

`stream` is a pointer to the C I/O `FILE` structure. (Output)

`fputc()` returns the character written. If a write error occurs, the error indicator for the stream is set and `fputc()` returns `EOF`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`clib.l`

See Also

[fgetc\(\)](#)

fputs()

Output String to File

Syntax

```
#include <stdio.h>
int fputs(
    const char      *str,
    FILE            *stream);
```

Description

`fputs()` writes a string to a stream. The terminating null character is not written.

<code>str</code>	is a pointer to the string. (Input)
<code>stream</code>	is a pointer to the stream to write. (Output)

`fputs()` returns `EOF` if a write error occurs. Otherwise, it returns a non-negative value.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`clib.lib`

See Also

[fgets\(\)](#)

Syntax

```
#include <stdio.h>
int fread(
    void           *ptr,
    size_t          size,
    size_t          nmemb,
    FILE           *stream);
```

Description

`fread()` reads up to `nmemb` elements into an array. The file position indicator for the stream, if defined, is advanced by the number of characters successfully read. If an error occurs, the resulting value of the file position indicator for the stream is indeterminate. If a partial element is read, its value is indeterminate.

<code>ptr</code>	is a pointer into the array. (Output)
<code>size</code>	specifies the size of each array element. (Input)
<code>nmemb</code>	specifies the number of array elements to read. (Input)
<code>stream</code>	is a pointer to the C I/O <code>FILE</code> structure. (Input)

`fread()` returns the number of elements successfully read, which may be less than `nmemb` if a read error or end-of-file is encountered. If `size` or `nmemb` is zero, `fread()` returns zero and the array contents and stream state remain unchanged.



For More Information

The `sclib.l` and `sclib.il` libraries contain a smaller version of the `fread()` function. Refer to the ***Using Ultra C/C++*** manual for more information about the small library functions.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`clib.l`

See Also

`fwrite()`

Syntax

```
#include <stdlib.h>
void free(void *ptr);
```

Description

`free()` deallocates the space pointed to by `ptr`. If `ptr` (input) is a null pointer, no action occurs. Otherwise, if the parameter does not match a pointer previously returned by `calloc()`, `malloc()`, or `realloc()`, or if the space has been deallocated by a call to `free()` or `realloc()`, the behavior is undefined.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	no
Standards:	ANSI

Library

`clib.l`

See Also

`calloc()`
`malloc()`
`realloc()`

_freemem(), freemem()

Determine Size of Unused Stack Area

Syntax

```
#include <stdlib.h>
int freemem(void);
int _freemem(void);
```

Description

`freemem()` returns the number of bytes allocated for the stack that have not been used.

If compiler stack checking is enabled, the stack is checked for possible overflow before entering a function. The lowest address the stack pointer has reached is retained so `freemem()` can report the number of bytes between the stack limit and the lowest stack value as the unused stack memory.



Note

The program must be compiled with stack checking code in effect for `freemem()` to return a correct result. `freemem()` is historical; avoid using it in new code as it is likely to be removed in a future release.

These calls are not available when compiled in strictly conforming ANSI mode.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

cstart.r

See Also

[stacksiz\(\)](#), [_stacksiz\(\)](#)

_freemin()

Set Memory Reclamation Bound

Syntax

```
#include <stdlib.h>
void _freemin(int size);
```

Description

`_freemin()` allows you to set the minimum size for a free block of memory or concatenation of free blocks for `free()` to return to the operating system. If a program is known to use a lot of memory, you can instruct `free()` never to return memory to the operating system.

- | | |
|------|---|
| size | is the minimum size for a free block of memory or the concatenation of free blocks. (Input) |
|------|---|
- If `size` is positive, it is used as the minimum number of bytes for a free block of memory to be a candidate for return to the operating system.
 - If `size` is negative, `free()` never tries to return memory allocated by the ANSI memory allocation functions (`calloc()`, `malloc()`, or `realloc()`) to the system.

You do not need to use `_freemin()` to return memory to the system. The default minimum size to return is 4K. `_freemin()` ignores any specified non-negative `size` smaller than 4K.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	no
Standards:	ANSI

Library

clib.l

See Also

[calloc\(\)](#)

[free\(\)](#)

[malloc\(\)](#)

[realloc\(\)](#)

Syntax

```
#include <stdio.h>
FILE *freopen(
    const char      *name,
    const char      *action,
    FILE            *stream);
```

Description

`freopen()` substitutes a file name for the file currently open, if any. The original file is closed, even if the opening of the new file does not succeed.

`freopen()` is usually used to associate the `stdin`, `stdout`, or `stderr` file pointers with a file instead of a terminal device.

name	is a pointer to the file name to use for a substitute. (Input)
action	specifies the action to take. (Input) Refer to <code>fopen()</code> for details on the action values.
stream	is a pointer to the C I/O <code>FILE</code> structure. (Input)

`freopen()` returns the `stream` passed or a null pointer if the open fails.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

clib.l

See Also

[fopen\(\)](#)

Return Portions of Floating Point Number

Syntax

```
#include <math.h>

double frexp(
    double      value,
    int        *exp);
```

Description

`frexp()` breaks a floating point value into a normalized fraction and an integral exponent of two. `frexp()` returns the fraction x , such that $1/2 \leq x < 1$ and $\text{value} = x * 2^{\text{exp}}$.

`value` is the floating point value. (Input)

`exp` is a pointer to the integral exponent of two. (Input)

Attributes

Operating System: OS-9 and OS-9 for 68K

State: User and System

Threads: Safe

Re-entrant: Yes

Standards: ANSI

Library

clib.lib

Syntax

```
#include <stdio.h>
int fscanf(
    FILE *stream,
    const char *format, ...);
```

Description

`fscanf()` reads input from a stream.

stream	is a pointer to the C I/O <code>FILE</code> structure. (Input)
format	is a pointer to a character sequence. (Input)
	<code>format</code> specifies the allowable input sequences and specifies how to convert these sequences for assignments. If the format is exhausted while parameters remain, the excess parameters are evaluated, but are otherwise ignored.

The format is a multibyte character sequence, beginning and ending in its initial shift state. The format is composed of zero or more directives:

- One or more white space characters.
- An ordinary multibyte character (neither % nor a white space character).
- A conversion specification.

A percent (%) character introduces each conversion specification. After the %, the following appear in sequence:

- An optional assignment-suppressing asterisk (*) character.
- An optional non-zero decimal integer that specifies the maximum field width.
- An optional h, l, or L indicating the size of the receiving object.

The conversion specifiers d, i, and n are preceded by h if the corresponding parameter is a pointer to a short int rather than pointing to int, by l if it is a pointer to a long int or by L if it is a pointer to long long int.

The conversion specifiers o, u, and x are preceded by h if the corresponding parameter is a pointer to an unsigned short int rather than pointing to an unsigned int, by l if it is a pointer to an unsigned long int or by L if it is a pointer to an unsigned long long int.

The conversion specifiers e, f, and g are preceded by l if the corresponding parameter is a pointer to a double rather than pointing to a float, or by L if it is a pointer to a long double.

If an h, l, or L appears with any other conversion specifier, the behavior is undefined.

- A character that specifies the type of conversion to apply. The following conversion specifiers are valid:

Table 2-10 fscanf() Conversion Specifiers

Flag	Description
c	Matches a character sequence of the number specified by the field width. By default, the field width is one. The corresponding parameter is a pointer to the initial character of an array large enough to accept the sequence. No null character is added.
d	Matches an optionally signed decimal integer, whose format is the same as expected for the subject sequence of strtol(). The base parameter is 10. The corresponding parameter is a pointer to an integer.

Table 2-10 fscanf() Conversion Specifiers (continued)

Flag	Description
e, f, g	Matches an optionally signed floating point number, whose format is the same as expected for the subject string of <code>strtod()</code> . The corresponding parameter is a pointer to a float.
i	Matches an optionally signed integer, whose format is the same as expected for the subject sequence of <code>strtol()</code> . The base parameter is 0. The corresponding parameter is a pointer to an integer.
n	No input is consumed. The corresponding parameter is a pointer to an integer into which to write the number of characters read from the input stream so far by this call to <code>fscanf()</code> . Executing a %n directive does not increment the assignment count returned when <code>fscanf()</code> finishes execution.
o	Matches an optionally signed octal integer, whose format is the same as expected for the subject sequence of <code>strtoul()</code> . The base parameter is 8. The corresponding parameter is a pointer to an unsigned integer.
p	Matches the set of sequences that may be produced by the %p conversion of <code>fprintf()</code> . The corresponding parameter is a pointer to a void. The input item is interpreted the same as %x. If the input item is a value converted earlier during the same program execution, the resulting pointer compares equal to that value. Otherwise, the behavior of the %p conversion is undefined.
s	Matches a sequence of non-white space characters. The corresponding parameter is a pointer to the initial character of an array large enough to accept the sequence and a terminating null character. The terminating null character is added automatically.
u	Matches an optionally signed decimal integer, whose format is the same as expected for the subject sequence of <code>strtoul()</code> . The base parameter is 10. The corresponding parameter is a pointer to an unsigned integer.
x	Matches an optionally signed hexadecimal integer, whose format is the same as expected for the subject sequence of <code>strtoul()</code> . The base parameter is 16. The corresponding parameter is a pointer to an unsigned integer.

Table 2-10 fscanf() Conversion Specifiers (continued)

Flag	Description
[Matches a non-empty character sequence from a set of expected characters called the <code>scanfset</code> . The corresponding parameter is a pointer to the initial character of an array large enough to accept the sequence and a terminating null character. The terminating null character is added automatically. The conversion specifier includes all subsequent characters in the format string, up to and including the matching right bracket (<code>]</code>). The characters between the brackets are the <code>scanlist</code> . The <code>scanlist</code> is the <code>scanfset</code> , unless the character after the left bracket is a caret (<code>^</code>). In this case, the <code>scanfset</code> contains all characters not appearing in the <code>scanlist</code> between the <code>^</code> and the <code>]</code> . If the conversion specifier begins with <code>[]</code> or <code>[^]</code> , the <code>]</code> is in the <code>scanlist</code> and the next right bracket character ends the specification. Otherwise, the first right bracket character ends the specification. A hyphen (<code>-</code>) has no special meaning; it is matched like any other character.
%	Matches a single <code>%</code> . No conversion or assignment occurs. The complete conversion specification is <code>%%</code> .

The conversion specifiers `E`, `G`, and `x` are also valid and behave the same as, respectively, `e`, `g`, and `x`.

If a conversion specification is invalid, the behavior is undefined.

`fscanf()` executes each directive of the format in turn. If a directive fails, `fscanf()` returns. Failures are described as **input failures** or **matching failures**.

- An input failure is caused by the unavailability of input characters.
- A matching failure is caused by inappropriate input.

A directive composed of white space character(s) is executed by reading input up to the first non-white space character or until no more characters can be read. The first non-white space character remains unread.

A directive containing an ordinary multibyte character is executed by reading the next characters of the stream. If one character differs from one in the directive, the directive fails, and the differing and subsequent characters remain unread.

A directive containing a conversion specification defines a set of matching input sequences for each specifier. A conversion specification is executed in the following steps:

- Input white space characters (as specified by `isspace()`) are skipped, unless the specification includes a `[`, `c`, or `n` specifier.
- An input item is read from the stream, unless the specification includes an `n` specifier. An input item is defined as the longest matching sequence of input characters, unless it exceeds a specified field width. If it exceeds the field width, it is the initial subsequence of that length in the sequence. The first character, if any, after the input item remains unread. If the length of the input item is zero, the execution of the directive fails. This condition is a matching failure, unless an error prevented input from the stream. If an error prevented input from the stream, it is an input failure.
- Except in the case of a `%` specifier, the input item (or, in the case of a `%n` directive, the count of input characters) is converted to a type appropriate to the conversion specifier. If the input item is not a matching sequence, the execution of the directive fails. This condition is a matching failure. Unless assignment suppression was indicated by an asterisk (`*`), the conversion result is placed in the object pointed to by the first parameter following the format parameter that has not already received a conversion result. If this object does not have an appropriate type or if the conversion result cannot be represented in the space provided, the behavior is undefined.

If the end-of-file is reached during input, conversion terminates. If end-of-file occurs before any characters matching the current directive have been read, other than leading white space where permitted, execution of the current directive terminates with an input failure. Otherwise, unless execution of the current directive is terminated with a matching failure, execution of the following directive, if any, is terminated with an input failure.

If conversion terminates on a conflicting input character, the offending input character is left unread in the input stream. Trailing white space, including newline characters, is left unread unless matched by a directive. You must use the `%n` directive to determine the success of literal matches and suppressed assignments.

`fscanf()` returns `EOF` if an input failure occurs before any conversion. Otherwise, `fscanf()` returns the number of input items assigned which can be fewer than provided for, or zero in the event of an early matching failure.



For More Information

The `sclib.l` and `sclib.il` libraries contain a smaller version of the `fscanf()` function. Refer to the ***Using Ultra C/C++*** manual for more information about the small library functions.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`clib.l`

See Also

[scanf\(\)](#)
[sscanf\(\)](#)
[strtod\(\)](#)
[strtol\(\)](#)
[strtoul\(\)](#)

Syntax

```
#include <stdio.h>
int fseek(
    FILE      *stream,
    long int   offset,
    int        place);
```

Description

`fseek()` sets the file position indicator for a stream.

stream	is a pointer to the C I/O FILE structure. (Input)
offset	specifies an amount to offset the file position indicator. (Input)
place	specifies where to place the file position indicator. (Input) place has three possible values:

Table 2-11 fseek() place Values

Value	Description
SEEK_SET	The new file position is offset.
SEEK_CUR	The new file position is the current file position + offset. offset can be negative.
SEEK_END	The new file position is the current file size + offset (usually offset is negative or zero in this case). A successful call to <code>fseek()</code> clears the end-of-file indicator for the stream and undoes the effects of the <code>ungetc()</code> function on the same stream. After an <code>fseek()</code> call, the next operation may be either input or output.

`fseek()` returns non-zero only for a request that cannot be satisfied.



For More Information

The `sclib.l` and `sclib.il` libraries contain a smaller version of the `fseek()` function. Refer to the ***Using Ultra C/C++*** manual for more information about the small library functions.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`clib.l`

See Also

[ftell\(\)](#)
[getc\(\), getchar\(\)](#)
[putc\(\), putchar\(\)](#)
[ungetc\(\)](#)

Syntax

```
#include <stdio.h>
int fsetpos(
    FILE             *stream,
    const fpos_t     *pos);
```

Description

`fsetpos()` sets the file position indicator for a stream.

`stream` is a pointer to the C I/O `FILE` structure.
(Input)

`pos` is a pointer to the value of the file
position indicator. (Input)
`pos` is returned from `fgetpos()`.

A successful call to `fsetpos()` clears the end-of-file indicator for the stream and undoes any effects of `ungetc()` on the same stream. After an `fsetpos()` call, the next operation on an update stream may be either input or output.

If successful, `fsetpos()` returns zero. Otherwise, it returns a non-zero value and places the appropriate error code in the global variable `errno`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

clib.l

See Also

[fgetpos\(\)](#)

Syntax

```
#include <UNIX/stat.h>
int fstat(
    unsigned int fd,
    struct stat *buf);
```

Description

`fstat()` obtains the same information about an open file referenced by the argument descriptor, as would be obtained by a `stat` call.

<code>fd</code>	Input
<code>buf</code>	is a pointer to a <code>stat</code> structure into which information about the file is placed. (Output)

`fstat()` returns zero on success. It returns -1 on failure and sets `errno` to indicate the error.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

`unix.l`

See Also

[stat\(\)](#)

Syntax

```
#include <stdio.h>
long ftell(FILE *stream);
```

Description

`ftell()` returns the current value of the file position indicator as the number of bytes from the beginning of the file for a stream. This is not necessarily a meaningful measure of the number of characters written or read.

`stream` is a pointer to the C I/O `FILE` structure.
(Input)

If successful, `ftell()` returns the current value of the file position indicator for the stream. On failure, `ftell()` returns `-1L` and the appropriate error code is placed in the global variable `errno`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`clib.lib`

See Also

`fseek()`
`getc()`, `getchar()`
`putc()`, `putchar()`

Syntax

```
#include <stdio.h>
size_t fwrite(
    const void *ptr,
    size_t      size,
    size_t      nmemb,
    FILE        *stream);
```

Description

`fwrite()` writes up to `nmemb` elements from an array to the stream. The file position indicator for the stream, if defined, is advanced by the number of characters successfully written. If an error occurs, the resulting value of the file position indicator for the stream is indeterminate.

<code>ptr</code>	is a pointer to the array. (Input)
<code>size</code>	specifies the size of each array element. (Input)
<code>nmemb</code>	specifies the number of array elements to write. (Input)
<code>stream</code>	is a pointer to the C I/O <code>FILE</code> structure. (Output)

`fwrite()` returns the number of elements successfully written. This is less than `nmemb` only if a write error is encountered.

A write to a text stream does not cause the associated file to truncate beyond that point.



For More Information

The `sclib.l` and `sclib.il` libraries contain a smaller version of the `fwrite()` function. Refer to the ***Using Ultra C/C++*** manual for more information about the small library functions.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`clib.l`

See Also

[`fread\(\)`](#)

get<name>()

Read Value of SPR, DCR, PREG, and TB

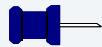
Syntax

```
#include <getset.h>
#include <types.h>
u_int32 _get_<name>(void);
```

Description

The `_get_<name>()` class of functions can be used to read the value of Special Purpose Registers (SPRs), Device Control Registers (DCRs), Processor Registers (PREGs), and Time-Based Registers (TBs).

`name` is any of the names listed in the following table. There is one exception to the following table, the `dec SPR` on the 601 can be read from user-state with the function `_get_user_dec()`.



Note

All functions are provided. It is the programmer's responsibility to avoid using system-state only registers from user-state and accessing registers that do not exist on the target processor. All registers are accessible from system-state. Only those noted are available from user-state. Attempting to use system state registers from user state causes unpredictable results.

Table 2-12 get<name> Valid Names and Associated Processors

<name>	Register Type	Processor(s)					Writable
		403	505	601	603	User	
bar	SPR			•			•
bear	DCR		•				
besr	DCR		•				•
br0	DCR		•				•
br1	DCR		•				•
br2	DCR		•				•
br3	DCR		•				•
br4	DCR		•				•
br5	DCR		•				•
br6	DCR		•				•
br7	DCR		•				•
cmpa	SPR			•			•
cmpb	SPR			•			•
cmpc	SPR			•			•
cmpd	SPR			•			•

**Table 2-12 get<name> Valid Names and Associated Processors
(continued)**

<name>	Register Type	Processor(s)					Writable
		403	505	601	603	User	
cmpe	SPR			•			•
cmpf	SPR		•				•
cmpg	SPR			•			•
cmph	SPR			•			•
counta	SPR			•			•
countb	SPR			•			•
cr	PREG	•	•	•	•	•	•
ctr	SPR	•	•	•	•	•	•
dabr	SPR			•			•
dac1	SPR	•					•
dac2	SPR	•					•
dar	SPR		•	•	•		•
dbat01	SPR				•		•
dbat0u	SPR				•		•
dbat11	SPR				•		•
dbat1u	SPR				•		•

**Table 2-12 get<name> Valid Names and Associated Processors
(continued)**

<name>	Register Type	Processor(s)				User	Writable
		403	505	601	603		
dbat21	SPR				•		•
dbat2u	SPR				•		•
dbat31	SPR				•		•
dbat3u	SPR				•		•
dbcrr	SPR	•					•
dbsr	SPR	•					•
dccr	SPR	•					•
dcmp	SPR				•		•
dear	SPR	•					
dec	SPR		•	•	•		•
der	SPR		•				•
dmacc0	DCR	•					•
dmacr0	DCR	•					•
dmacr1	DCR	•					•
dmacr2	DCR	•					•
dmacr3	DCR	•					•

**Table 2-12 get<name> Valid Names and Associated Processors
(continued)**

<name>	Register Type	Processor(s)					User	Writable
		403	505	601	603			
dmact0	DCR	•						•
dmact1	DCR	•						•
dmact2	DCR	•						•
dmact3	DCR	•						•
dmada0	DCR	•						•
dmada1	DCR	•						•
dmada2	DCR	•						•
dmada3	DCR	•						•
dmasa0	DCR	•						•
dmasa1	DCR	•						•
dmasa2	DCR	•						•
dmasa3	DCR	•						•
dmasr	DCR	•						•
dmiss	SPR				•			•
dpdr	SPR		•					•
dsisr	SPR		•	•	•			•

**Table 2-12 get<name> Valid Names and Associated Processors
(continued)**

<name>	Register Type	Processor(s)				User	Writable
		403	505	601	603		
ear	SPR			•	•		•
ecr	SPR		•				•
eid	SPR		•				•
eie	SPR		•				•
esr	SPR	•					•
evpr	SPR	•					•
exier	DCR	•					•
exisr	DCR	•					•
fpecr	SPR		•				•
fpscr	PREG	•	•	•	•	•	•
hash1	SPR				•		•
hash2	SPR				•		•
hid0	SPR			•	•		•
hid1	SPR			•			•
hid15	SPR			•			•
hid2	SPR			•			•

**Table 2-12 get<name> Valid Names and Associated Processors
(continued)**

<name>	Register Type	Processor(s)					User	Writable
		403	505	601	603			
hid5	SPR			•				•
iabr	SPR			•	•			•
iac1	SPR	•						•
iac2	SPR	•						•
ibat01	SPR			•	•			•
ibat0u	SPR			•	•			•
ibat11	SPR			•	•			•
ibat1u	SPR			•	•			•
ibat21	SPR			•	•			•
ibat2u	SPR			•	•			•
ibat31	SPR			•	•			•
ibat3u	SPR			•	•			•
icadr	SPR		•					•
iccr	SPR	•						•
iccst	SPR			•				•
icdat	SPR			•				•

**Table 2-12 get<name> Valid Names and Associated Processors
(continued)**

<name>	Register Type	Processor(s)					User	Writable
		403	505	601	603			
icmp	SPR				•		•	
ictrl	SPR		•				•	
imiss	SPR				•		•	
iocr	DCR	•					•	
lctrl11	SPR		•				•	
lctrl12	SPR		•				•	
lr	SPR	•	•	•	•	•	•	
mq	SPR			•		•	•	
nri	SPR		•				•	
msr	PREG	•	•	•	•		•	
pbl1	SPR	•					•	
pbl2	SPR	•					•	
pbu1	SPR	•					•	
pbu2	SPR	•					•	
pir	SPR			•			•	
pit	SPR	•					•	

**Table 2-12 get<name> Valid Names and Associated Processors
(continued)**

<name>	Register Type	Processor(s)					User	Writable
		403	505	601	603			
pvr	SPR	•		•	•			
rpa	SPR				•		•	
rtcl	SPR			•			•	
rtcu	SPR			•			•	
sdr1	SPR			•	•		•	
sprg0	SPR	•		•	•		•	
sprg1	SPR	•		•	•		•	
sprg2	SPR	•		•	•		•	
sprg3	SPR	•		•	•		•	
sr0	PREG			•	•		•	
sr1	PREG			•	•		•	
sr2	PREG			•	•		•	
sr3	PREG			•	•		•	
sr4	PREG			•	•		•	
sr5	PREG			•	•		•	
sr6	PREG			•	•		•	

**Table 2-12 get<name> Valid Names and Associated Processors
(continued)**

<name>	Register Type	Processor(s)					User	Writable
		403	505	601	603			
sr7	PREG			•	•			•
sr8	PREG			•	•			•
sr9	PREG			•	•			•
sr10	PREG			•	•			•
sr11	PREG			•	•			•
sr12	PREG			•	•			•
sr13	PREG			•	•			•
sr14	PREG			•	•			•
sr15	PREG			•	•			•
srr0	SPR	•	•	•	•			•
srr1	SPR	•	•	•	•			•
srr2	SPR	•						•
srr3	SPR	•						•
tbhi	SPR	•						•
tbl	SPR		•		•			•
tbl	TB		•		•	•		*

**Table 2-12 get<name> Valid Names and Associated Processors
(continued)**

<name>	Register Type	Processor(s)					User	Writable
		403	505	601	603			
tblo	SPR	•					•	
tbu	SPR		•		•		•	
tbu	TB		•		•	•	•	*
tcr	SPR	•					•	
tsr	SPR	•					•	
xer	SPR	•	•	•	•	•	•	

Legend:

- # User-state read version is _get_user_<name>
- * Write version is system state only

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

cpu.l

See Also

set<name>()

getc(), getchar()

Get Next Character from File, Stdin

Syntax

```
#include <stdio.h>
int getc(FILE *stream);
int getchar(void);
```

Description

`getc()` returns a character, as an unsigned converted to an `int`, from the file and advances the associated file position indicator for the stream, if defined. If `getc()` is implemented as a macro, it may evaluate `stream` more than once. Therefore, the parameter should never be an expression with side effects. `getchar()` is a macro that is equivalent to `getc(stdin)`.

stream is a pointer to the C I/O FILE structure.
(Input)

`getc()` returns the next character from the input stream pointed to by `stream`. If the stream is at end-of-file, the end-of-file indicator for the stream is set and `getc()` returns `EOF`. If a read error occurs, the error indicator for the stream is set and `getc()` returns `EOF`.

`getchar()` returns the next character from the input stream pointed to by `stdin`. If the stream is at end-of-file, the end-of-file indicator for the stream is set and `getchar()` returns `EOF`. If a read error occurs, the error indicator for the stream is set and `getchar()` returns `EOF`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

c lib.l

See Also

[fgetc\(\)](#)
[fopen\(\)](#)
[putc\(\), putchar\(\)](#)

get_const()

Get Current Constant Data

Syntax

```
#include <regs.h>
void *get_const(void);
void *_get_const(void);
```

Description

`get_const()` and `_get_const()` return the current constant data pointer. These functions are not present in the libraries for a given processor if a constant data pointer is not applicable.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`get_const()` - `cpu.lib`, `_get_const()` - `os.lib`

get_current_tss()

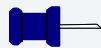
Get Current Task Segment

Syntax

```
#include <types.h>
#include <regs.h>
u_int32 *get_current_tss(void);
```

Description

`get_current_tss()` returns the current task segment.



Note

`get_current_tss()` is only supported on the 80x86 family of processors version of the compiler.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`cpu.l`

getenv()

Get Value for Environment Name

Syntax

```
#include <stdlib.h>
char *getenv(const char *name);
```

Description

`getenv()` searches the environment list for a string that matches the specified name.

name is a pointer to the name. (Input)

`getenv()` returns a pointer to a string associated with the matched list member. The string pointed to cannot be modified by the program. If the specified name cannot be found, a null pointer is returned.



For More Information

The common set of environment names depends on the operating system. Refer to your operating system's user manual for more information.

The environment list is implemented as a NULL terminated array of pointers to characters. Each string is in the form:

<name>=<value>

`<name>` is the name of the environment variable.

`<value>` is the current value of the environment variable.

To modify the list, you must either find the existing entry or enlarge the array and add an entry. The base of the array is stored in the external variable `environ`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

clib.l

See Also

[_os_chain\(\)](#)
[_os_chainm\(\)](#)
[_os_dfork\(\)](#)
[_os9_dfork\(\)](#)
[_os_dforkm\(\)](#)
[_os_exec\(\)](#)
[_os_fork\(\)](#)
[os9fork\(\), os9forkc\(\)](#)
[_os_forkm\(\)](#)

getgid()

Return Group ID

Syntax

```
#include <UNIX/os9def.h>
unsigned int getgid();
```

Description

getgid() returns the group identification of the current process.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

unix.l

get_excpt_base()

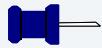
Return Exception Table Base Address

Syntax

```
#include <types.h>
#include <regs.h>
u_int32 *get_excpt_base(void);
```

Description

`get_excpt_base()` returns the exception table base address.



Note

`get_excpt_base()` is only supported on the 80x86 family of processors version of the compiler.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

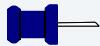
cpu.l

Syntax

```
#include <types.h>
#include <regs.h>
void *get_gdtr(void);
```

Description

`get_gdtr()` returns a pointer to the global descriptor.



Note

`get_gdtr()` is only supported on the 80x86 family of processors version of the compiler.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

cpu.l

Syntax

```
#include <time.h>
int getime(struct sgtbuf *timebuf);
```

Description

`getime()` returns the system time in a time buffer. The time units are defined in the `time.h` header file.

`timebuf` is a pointer to the time buffer. (Output)

If successful, `getime()` returns the buffer pointer.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

[_os_getime\(\)](#)

`F$Time`

`F_TIME`

OS-9 for 68K Technical Manual

OS-9 Technical Manual

_get_module_dir()

Get Module Directory Entry

Syntax

```
#include <module.h>
int _get_module_dir(
    void *buffer,
    int count);
```

Description

`_get_module_dir()` copies the system's module directory into the buffer for inspection. A maximum of `count` bytes are copied.

`_get_module_dir()` returns the number of bytes actually copied.

`buffer` is a pointer to the buffer. (Output)

`count` specifies the maximum number of bytes to copy. (Input)

If an error occurs, `-1` is returned and the appropriate error code is placed in the global variable `errno`.

Attributes

Operating System: OS-9 for 68K

State: User and System

Threads: Safe

Re-entrant: Yes

Library

`sys_clib.l`

See Also

[_os_get_moddir\(\)](#)

`F$GModDr`

OS-9 for 68K Technical Manual

getnextpwnam()

Get Next Password File Entry for Name

Syntax

```
#include <UNIX/pwd.h>
struct passwd *getnextpwnam(const char *name);
```

Description

name	getnextpwnam() returns the next password file entry associated with name. (Input)
	getnextpwnam() returns NULL if no more password file entries are found for the name.

This function is useful for OS-9 systems where users have more than one password file entry.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

unix.l

See Also

[endpwent\(\)](#)
[fgetpwent\(\)](#)
[getpwent\(\)](#)
[getpwuid\(\)](#)
[getpwnam\(\)](#)
[setpwent\(\)](#)

getopt()

Return Next Option Letter in String

Syntax

```
#include <UNIX/os9def.h>
extern char *optarg;
extern int optind, opterr;
int getopt(
            int             argc,
            char           *const *argv,
            const char     *optstring);
```

Description

`getopt()` returns the next option letter in `argv` that matches a letter in `optstring`. `optstring` must contain the option letters the command using `getopt()` recognizes. If a letter is followed by a colon, the option is expected to have an argument, or a group of arguments, which must be separated from it by white space.

`optarg` is set to point to the start of the option argument on return from `getopt()`.

`getopt()` places in `optind` the `argv` index of the next argument to be processed. `optind` is external and is initialized to 1 before the first call to `getopt()`.

When all options have been processed (up to the first non-option argument), `getopt()` returns -1. The special option -- may be used to delimit the end of the options. When it is encountered, -1 is returned and -- is skipped.

`getopt()` prints an error message on the standard error and returns a question mark (?) when it encounters an option letter not included in `optstring` or no option-argument after an option that expects one. This error message may be disabled by setting `opterr` to 0.

<code>argc</code>	Input
<code>argv</code>	Input
<code>optstring</code>	Input

For example, this code fragment shows how one might process the arguments for a command that can take the mutually exclusive options **a** and **b**, and the option **o**, which requires an option argument:

```
main(argc, argv);
int argc;
char **argv;
{
int c;
extern char *optarg;
extern int optind;

.

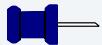
.

while ((c = getopt(argc, argv, "abo:")) != -1)
    switch (c) {
        case 'a':
            if (bflg)
                errflg++;
            else
                aflag++;
            break;
        case 'b':
            if (aflg)
                errflg++;
            else
                bproc ();
            break;
        case 'o':
            ofile = optarg;
            break;
        case '?':
            errflg++;
    }
if (errflg) {
    (void)fprintf(stderr, "usage: . . .");
    exit (2);
}
for (; optind < argc; optind++) {
    if (access(argv[optind], 4)) {

.

.

    }
}
```



Note

Changing the value of the variable `optind`, or calling `getopt()` with different values of `argv`, may lead to unexpected results.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Unsafe
Re-entrant:	No

Library

`unix.l`

getpid()

Return Process ID Number

Syntax

```
#include <process.h>
int getpid(void);
```

Description

getpid() returns the system process ID number for the calling process. You can use this number for tasks such as creating unique file names.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

sys_clib.l

See Also

[_os_id\(\)](#)
[_os9_id\(\)](#)
F\$ID
F_ID

OS-9 for 68K Technical Manual
OS-9 Technical Manual

getppid()

Return Parent Process ID Number

Syntax

```
#include <UNIX/os9def.h>
unsigned int getppid(void);
```

Description

getppid() returns the system process ID number for the parent of the calling process. You can use this number for tasks such as creating unique file names.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

unix.l

See Also

[getpid\(\)](#)

[_get_process_desc\(\)](#)

Get Process Descriptor Copy

Syntax

```
#include <process.h>
int _get_process_desc(
    int          pid,
    int          count,
    pr_desc     *buffer);
```

Description

`_get_process_desc()` copies a process descriptor into the buffer for inspection.

`pid` is the process ID number of the desired process. (Input)

Only the lower 16 bits of `pid` are used.

`count` specifies the maximum number of bytes to copy. (Input)

`buffer` is a pointer to the buffer. (Output)

If an error occurs, `-1` is returned and the appropriate error code is placed in the global variable `errno`.

The following is an example call:

```
procid pbuf;
_get_process_desc(pid,sizeof(pbuf),&pbuf);
```

Attributes

Operating System: OS-9 for 68K

State: User and System

Threads: Safe

Re-entrant: Yes

Library

`sys_clib.l`

See Also

[_os_gprdsc\(\)](#)
[F\\$GPrDsc](#)

OS-9 for 68K Technical Manual

[_get_process_table\(\)](#)

Get Process Table Entry

Syntax

```
#include <process.h>
int _get_process_table(
    void      *buffer,
    int       count);
```

Description

`_get_process_table()` copies the system's process descriptor table into the buffer for inspection. A maximum of `count` bytes are copied. The number of bytes actually copied is returned.

`buffer` is a pointer to the buffer. (Output)

`count` specifies the maximum number of bytes to copy. (Input)

If an error occurs, -1 is returned and the appropriate error code is placed in the global variable `errno`.

Attributes

Operating System: OS-9 and OS-9 for 68K

State: User and System

Threads: Safe

Re-entrant: Yes

Library

`sys_clib.l`

See Also

[`_os_get_prtbl\(\)`](#)

F\$GPrDBT

OS-9 for 68K Technical Manual

getpw()

Get Name from Password File

Syntax

```
#include <UNIX/pwd.h>
int getpw(
    int      uid,
    char    *buf);
```

Description

`getpw()` searches the password file for the (numerical) user identification number and fills in the buffer with the corresponding name. It returns non-zero if the uid could not be found. The name is null-terminated.

`uid` is the user identification number. (Input)
`buf` is the buffer. (Output)

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

unix.l

See Also

```
endpwent()
fgetpwent()
getpwent()
getpwuid()
getpwnam()
setpwent()
```

Syntax

```
#include <UNIX/pwd.h>
struct passwd *getpwent(void);
```

Description

`getpwent()`, `getpwuid()` and `getpwnam()` each return a pointer to an object with the following structure containing the fields of a line in the password file. Each line in the file contains a `passwd` structure, declared in the `<UNIX/pwd.h>` header file:

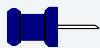
```
struct passwd {
    char *pw_name;      * login name
    char *pw_passwd;    * password
    int pw_uid;         * user ID
    int pw_gid;         * group ID
    int pw_prio;        * priority
    char *pw_age;        * unsupported -zero length string
    char *pw_comment;   * unsupported -zero length string
    char *pw_gecos;     * unsupported -zero length string
    char *pw_dir;        * initial data directory
    char *pw_xdir;       * initial execution directory
    char *pw_shell;      * initial command executed
    char *pw_junk;        * undefined
};
```

The fields `pw_age`, `pw_comment`, `pw_gecos`, and `pw_junk` are unused. When first called, `getpwent()` returns a pointer to the first `passwd` structure in the file; thereafter, it returns a pointer to the next `passwd` structure in the file; so successive calls can be used to search the entire file. `getpwuid()` searches from the beginning of the file until a numerical user ID matching `uid` is found and returns a pointer to the particular structure in which it was found. `getpwnam()` searches from the beginning of the file until a login name matching `name` is found, and returns a pointer to the particular structure in which it was found. If an end-of-file or an error is encountered on reading, these functions return a NULL pointer.

A call to `setpwent()` has the effect of rewinding the password file to allow repeated searches. `endpwent()` may be called to close the password file when processing is complete.

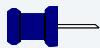
`fgetpwent()` returns a pointer to the next `passwd` structure in the stream `f`, which matches the format of the password file `/dd/SYS/password`.

`getpwent()`, `getpwuid()`, and `getpwnam()` return a pointer to `struct passwd` on success. On EOF or error, or if the requested entry is not found, they return `NULL`.



Note

The above routines use the standard I/O library, which increases the size of programs not otherwise using standard I/O more than might be expected.



Note

All information is contained in a static area which is overwritten by subsequent calls to these functions, so it must be copied if it is to be saved.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

unix.1

See Also

[endpwent\(\)](#)
[getpwent\(\)](#)
[getpw\(\)](#)
[getpwuid\(\)](#)
[getpwnam\(\)](#)
[setpwent\(\)](#)

getpwnam()

Get Password File Name

Syntax

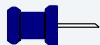
```
#include <UNIX/pwd.h>
struct passwd *getpwnam(const char *nam);
```

Description

`getpwnam()` returns a pointer to an object with the structure shown in `getpwent()` containing the fields of a line in the password file that matches the name `nam`. Each line in the file contains a `passwd` structure, declared in the `<UNIX/pwd.h>` header file.

`nam`

is the name matching the fields in the password file. (Input)



Note

The above routines use the standard I/O library, which increases the size of programs not otherwise using standard I/O more than might be expected.



Note

All information is contained in a static area which is overwritten by subsequent calls to these functions, so it must be copied if it is to be saved.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

unix.l

See Also

[endpwent\(\)](#)
[fgetpwent\(\)](#)
[getpw\(\)](#)
[getpwent\(\)](#)
[getpwuid\(\)](#)
[setpwent\(\)](#)

getpwuid()

Get Password File User ID

Syntax

```
#include <UNIX/pwd.h>
struct passwd *getpwuid(unsigned int uid);
```

Description

`getpwuid()` returns a pointer to an object with the structure shown in `getpwent()` containing the fields of a line in the password file that matches the specified user identification. Each line in the file contains a `passwd` structure, declared in the `<UNIX/pwd.h>` header file.

uid is the specified user identification. (Input)



Note

The above routines use the standard I/O library, which increases the size of programs not otherwise using standard I/O more than might be expected.



Note

All information is contained in a static area which is overwritten by subsequent calls to these functions, so it must be copied if it is to be saved.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

unix.1

See Also

[endpwent\(\)](#)
[fgetpwent\(\)](#)
[getpw\(\)](#)
[getpwent\(\)](#)
[getpwnam\(\)](#)
[setpwent\(\)](#)

gets()

Get String from File

Syntax

```
#include <stdio.h>
char *gets(char *str);
```

Description

`gets()` reads characters from the standard input file into an array until end-of-file is encountered or a newline character is read. Any newline character is discarded, and a null character is written immediately after the last character read into the array.

`str` is a pointer to the array. (Output)

`gets()` returns `str` if successful. If end-of-file is encountered and no characters have been read into the array, the contents of the array remain unchanged and a null pointer is returned. If a read error occurs during the operation, the array contents are indeterminate and a null pointer is returned.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

c lib. l

See Also

fgets()

Syntax

```
#include <sgstat.h>
#include <sg_codes.h>

int getstat(
    int          code,
    int          path,
    struct sgbuf *buffer);      * code = 0

int          getstat(
int          code,
int          path);           * code = 1 or 6

int          getstat(
int          code,
int          path,
int          *size);          * code = 2

int          getstat(
int          code,
int          path,
int          *pos);           * code = 5
```

Description

`getstat()` is historical from the 6809 C Compiler. It accesses a few I\$GetStt system functions.

`code` is defined as shown in **Table 2-13**.
 (Input)

Table 2-13 getstat() Code Values

Code	Description
0	<code>buffer</code> is the address of the 128 byte buffer into which the path option bytes are copied. The <code>sgstat.h</code> header file contains a <code>struct</code> defined for use by the program.
1	This applies only to SCF devices and is used to test for data ready. The return value is the number of bytes available or <code>-1</code> if an error occurred.
2	<code>size</code> is the address of the long integer into which the current file size is placed. The function returns <code>-1</code> on error and zero on success.
5	<code>pos</code> is the address of the long integer into which the current file position is placed. The function returns <code>-1</code> on error and zero on success.
6	The function returns <code>-1</code> on either EOF or error, and 0 if not at EOF.

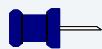


Note

Only the lower 16 bits of `code` are used.

path	must be the path number of an open file. (Input)
buffer	is the address of the 128 byte buffer into which the path option bytes are copied. (Output)
size	is the address of the long integer into which the current file size is placed. (Output)
pos	is the address of the long integer into which the current file position is placed. (Output)

When `getstat()` returns with the value -1, the appropriate error code is placed in the global variable `errno`.



Note

`getstat()` is supported for the convenience of programs ported from the 6809.

Attributes

Operating System:	OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

[_os_gs_devnm\(\)](#)
[_os_gs_eof\(\)](#)
[_os_gs_fd\(\)](#)
[_os_gs_fdinf\(\)](#)
[_os_gs_popt\(\)](#)
[_os_gs_pos\(\)](#)
[_os_gs_ready\(\)](#)
[_os_gs_size\(\)](#)
[I\\$GetStt](#)

OS-9 for 68K Technical Manual

get_static()

Return Current Static Storage Pointer

Syntax

```
#include <regs.h>
void *get_static(void);
void *_get_static(void);
```

Description

`get_static()` and `_get_static()` return a pointer to the current static storage.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`get_static()` - `cpu.l`, `_get_static` - `os_lib.l`

See Also

[change_static\(\)](#)

_getsys()

Get System Global Variables

Syntax

```
#include <sysglob.h>
#include <setsys.h>
int _getsys(
    int    glob,
    int    size);
```

Description

`_getsys()` examines a system global variable. These variables are defined in `setsys.h` for C programs. Each of these variables begin with a `D_` prefix.

`glob` is the system global variable to examine. (Input)

Only the lower 16 bits of `glob` are used.

`size` is the size of the variable. (Input)

`_getsys()` returns the value of the variable if the examine request succeeds. If the request fails, `-1` is returned and the appropriate error code is placed in the global variable `errno`.

Attributes

Operating System:	OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

[_os_getsys\(\)](#)

[_os_setsys\(\)](#)

[F\\$SetSys](#)

OS-9 for 68K Technical Manual

gettimeofday()

Get Current Time

Syntax

```
#include <UNIX/os9time.h>
int gettimeofday(
    struct timeval *tp,
    struct timezone *tzp);
```

Description

The system's notion of the current time is obtained with the `gettimeofday()` call. The current time is expressed in elapsed seconds and microseconds since January 1, 1970 (zero hour). The resolution of the system clock is hardware dependent. The time may be updated continuously, or in "ticks."

`gettimeofday()` returns zero (0) on success and -1 on failure and sets `errno` to indicate the error.

`tp` points to a `timeval` structure, which includes the following members:

long tv_sec;	* seconds since Jan. 1, 1970
long tv_usec;	* and microseconds

If `tp` is a NULL pointer, the current time information is not returned.

`tzp` is not supported by `unix.l`. (Input)

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

unix.1

getuid()

Determine User ID Number

Syntax

```
#include <process.h>
int getuid(void);
```

Description

`getuid()` returns the group/user ID of the current process. The upper word (two bytes) of the value is the group number; the lower word is the user number.

If unsuccessful, `-1` is returned and the appropriate error code is placed in the global variable `errno`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`_os_id()`
`_os9_id()`
`F$ID`
`F_ID`

OS-9 for 68K Technical Manual
OS-9 Technical Manual

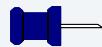
Syntax

```
#include <stdio.h>
int getw(FILE *stream);
```

Description

`getw()` returns a word from a file converted to an integer. It returns -1 on error. Therefore, use the macros `feof()` and `ferror()` to check the success of `getw()`.

<code>stream</code>	is a pointer to the C I/O <code>FILE</code> structure. (Input)
---------------------	---



Note

`getw()` is a machine-dependent function because the size of a word varies from machine to machine. `getw()` assumes no particular alignment in the file. `getw()` reads two bytes from the file and sign-extends them to four bytes.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

`sys_clib.l`

See Also

`feof()`
`ferror()`

getwd()

Get Working Directory Path Name

Syntax

```
#include <UNIX/os9def.h>
char *getwd(char pathname[MAXPATHLEN]);
```

Description

getwd() copies the absolute pathname of the current working directory to pathname (output) and returns a pointer to the result.

getwd() returns NULL and global variable errno if an error occurs.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

unix.l

Syntax

```
#include <time.h>
struct tm *gmtime(time_t *tp);
```

Description

gmtime() converts the calendar time contained in a time structure to Coordinated Universal Time (UTC). gmtime() returns a pointer to that object or a null pointer if UTC is not available.

tp is a pointer to the time structure. (Input)



Note

gmtime() returns a pointer to a static area which may be overwritten. To ensure data integrity, use the structure or save it immediately.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

clib.lib

See Also

[mktime\(\)](#) (for more information on the time structure tp)
[time\(\)](#)

_gregorian()

Convert Date/Time to Gregorian Value

Syntax

```
#include <time.h>
int _gregorian(int *time, int *date);
```

Description

`_gregorian()` converts the time and date from Julian format to the Gregorian equivalents.

time and date	are pointers to the Julian time and date values. (Input/Output)
	The values must be in the following format:

Figure 2-2 `_gregorian()` Input Date and Time Format

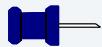
Time:	Seconds Since Midnight (0-86399)
Date:	Julian Day Number

`_gregorian()` modifies the values to the Gregorian format:

Figure 2-3 `_gregorian()` Output Date and Time Format

Time:	0	Hour (0-23)	Minute	Second
Date:	Year (2-Bytes)		Month	Day
	Byte 0	Byte 1	Byte 2	Byte 3

If successful, `_gregorian()` returns zero. Otherwise, -1 is returned.



Note

Be careful to pass pointers for the date and time values.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

sys_clib.l

See Also

[_os_julian\(\)](#) (OS-9 for 68K)
[_os_gregorian\(\)](#) (OS-9 for 68K)

_gs_devn()

Get Device Name

Syntax

```
#include <sg_codes.h>
int _gs_devn(int path, char *buffer);
```

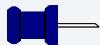
Description

`_gs_devn()` allows you to determine the name of the device open on a path.

`path` is the path number of an open path.
(Input)

`buffer` is a pointer to a character array into which the null-terminated device name is placed. (Output)

If the path number is invalid, `-1` is returned and the appropriate error code is placed in the global variable `errno`.



Note

You must reserve a buffer of at least 32 bytes to receive the device name. Some networked devices using NFM should have a buffer size of 128 or 256 bytes for the `_gs_devn()`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

[_os_gs_devnm\(\)](#)

[I\\$GetStt](#)

[I_GETSTAT, SS_DEVNAME](#)

OS-9 for 68K Technical Manual

OS-9 Technical Manual

_gs_eof()

Check for End of File

Syntax

```
#include <sg_codes.h>
int _gs_eof(int path);
```

Description

`_gs_eof()` determines if the file open on `path` is at the end of file. If it is at the end of file, 1 is returned. Otherwise, 0 is returned.

<code>path</code>	specifies the path number of the open file. (Input)
	If <code>path</code> is invalid, -1 is returned and the appropriate error code is placed in the global variable <code>errno</code> .



Note

`_gs_eof()` cannot determine end-of-file on SCF devices. SCF devices return an `EOS_EOF` error when the EOF character is read. Do not use this call if you are using the buffered I/O facility on the path. Instead, use `feof()`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

[feof\(\)](#)
[_os_gs_eof\(\)](#)
[I\\$GetStt](#)
[I_GETSTAT](#)
[SS_EOF](#)

OS-9 for 68K Technical Manual
OS-9 Technical Manual
OS-9 Technical Manual

_gs_gfd()

Get File Descriptor Sector

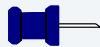
Syntax

```
#include <rbf.h>
int _gs_gfd(
    int          path,
    struct fd   *buffer,
    int          count);
```

Description

`_gs_gfd()` places a copy of the RBF file descriptor sector of the file open on `path` into a buffer. The structure `fd`, declared in the `rbf.h` header file, provides a convenient means to access the file descriptor information.

path	specifies the path number of the open file. (Input)
buffer	is a pointer to the buffer in which to place the file descriptor information. (Output)
count	specifies the maximum number of bytes to copy. (Input)
	If an error occurs, -1 is returned and the appropriate error value is placed in the global variable <code>errno</code> .



Note

Be sure the buffer is large enough to hold `count` bytes. A file descriptor is one sector in length and a sector may be between 256 bytes and 32K in size. If an entire descriptor is desired, you must first find the sector size on the device and allocate a buffer. The sector size can be found in the path descriptor's option section. See `_os_gs_popt()` and your operating system technical manual for more information. Generally, the `struct fd` in `rbf.h` is only large enough to hold a 256-byte file descriptor.



Note

This call is effective only on RBF devices.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`_os_gs_fd()`
`_os_gs_popt()`
`_os_ss_fd()`
`I$GetStt`
`I_GETSTAT, SS_FD`

OS-9 for 68K Technical Manual
OS-9000 Technical Manual

Syntax

```
#include <sg_codes.h>
int _gs_opt(
    int             path,
    struct sgbuf   *buffer);
```

Description

`_gs_opt()` copies the options section of the path descriptor open on path into a buffer. `sgbuf` provides a convenient means to access the individual option values. `sgbuf` is declared in the `sgstat.h` header file.

`path` is the path number of the open path descriptor. (Input)

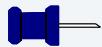
`buffer` is a pointer to the buffer. (Output)

If the path is invalid, -1 is returned and the appropriate error code is placed in the global variable `errno`.



Note

Be sure the buffer is large enough to hold the options. Declaring the buffer as type `struct sgbuf` is safe because this structure is predefined to be large enough for the options.



Note

If you use `_gs_opt()` on OS-9, you must use the OS-9 for 68K header file. `_gs_opt()` is provided on OS-9 solely for compatibility with OS-9 for 68K. If the OS-9 for 68K header file is unavailable, change the source code to use `_os_gs_popt()` or `_os_gs_luopt()`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

[`_os_gs_popt\(\)`](#)
[`_os_ss_popt\(\)`](#)

`I$GetStt`
`I_GETSTAT`
`SS_PATHOPT`
`tmode`

OS-9 for 68K Technical Manual
OS-9 Technical Manual
OS-9 Technical Manual
Using OS-9 for 68K or *Using OS-9*

_gs_pos()

Get Current File Position

Syntax

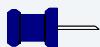
```
#include <sg_codes.h>
int _gs_pos(int path);
```

Description

`_gs_pos()` returns the value of the file pointer for the file open on path.

path is the path number of the open file.
(Input)

If the path is invalid or the device is not an RBF device, -1 is returned and the appropriate error code is placed in the global variable `errno`.



Note

This call is effective only on RBF devices. Do not use this call if buffered I/O is being performed on the path; instead use `fseek()`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

[ftell\(\)](#)

[_os_gs_pos\(\)](#)

[I\\$GetStt](#)

[I_GETSTAT, SS_POS](#)

OS-9 for 68K Technical Manual

OS-9 Technical Manual

_gs_rdy()

Test for Data Available

Syntax

```
#include <sg_codes.h>
int _gs_rdy(int path);
```

Description

`_gs_rdy()` checks the device open on `path` to see if data is waiting to be read. Read requests to the operating system wait until enough bytes have been read to satisfy the specified byte count, thereby suspending the program until the read is satisfied.

`_gs_rdy()` can determine the number of bytes, if any, waiting to be read, and then issue a read request for only the number of bytes actually available.

`path` is the path number of the open file.
(Input)

If the path is invalid or no data is available to be read, `-1` is returned, and the appropriate error code is placed in the global variable `errno`. Otherwise, the number of bytes available to be read is returned.



Note

This call is effective only on RBF, SCF, or pipe devices. `_gs_rdy()` is not intended for use with the buffered I/O calls such as `getc()`; unpredictable results may occur. Use low-level functions when using `_gs_rdy()`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

sys_clib.l

See Also

[_os_gs_ready\(\)](#)

[_os_read\(\)](#)

[_os_readln\(\)](#)

[I\\$GetStt](#)

[I_GETSTAT](#)

[SS_READY](#)

OS-9 for 68K Technical Manual

OS-9 Technical Manual

OS-9 Technical Manual

Syntax

```
#include <sg_codes.h>
int _gs_size(int path);
```

Description

`_gs_size()` determines the current size of the file open on `path`.

`path` is the path number of the open file.
(Input)

If the path is invalid or the device is not an RBF device, -1 is returned and the appropriate error code is placed in the global variable `errno`. Otherwise, the file size is returned.



Note

This call is effective only on RBF and PIPEMAN devices.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

[_os_gs_size\(\)](#)

[I\\$GetStt](#)

[I_GETSTAT](#)

[SS_SIZE](#)

OS-9 for 68K Technical Manual

OS-9 Technical Manual

OS-9 Technical Manual

Syntax

```
#include <math.h>
double hypot(double x, double y);
```

Description

hypot() returns the Euclidean distance function:

$$\sqrt{x * x + y * y}$$

hypot() ignores overflows.

x and y are inputs.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

sys_clib.l

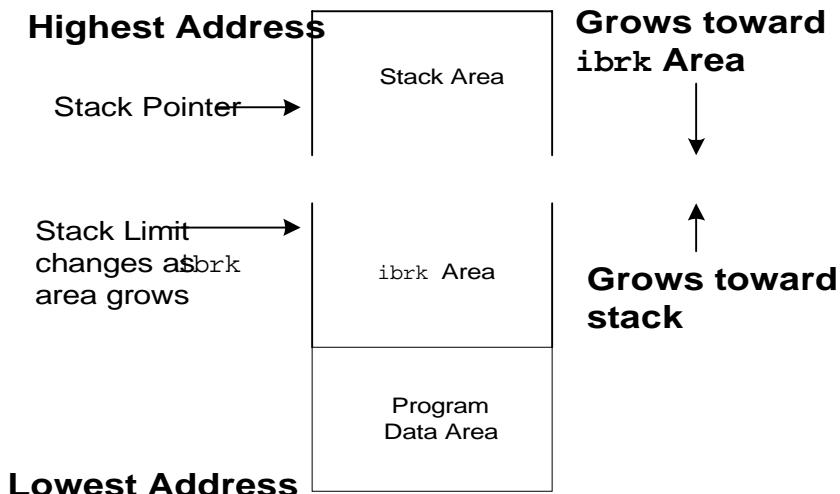
Syntax

```
#include <stdlib.h>
char *ibrk(unsigned size);
```

Description

ibrk() returns a pointer to a memory block. The returned pointer is aligned to a word boundary. The memory from which ibrk() grants requests is the area between the end of the data allocation and the stack:

Figure 2-4 ibrk() Memory Stack Diagram



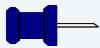
`size`

specifies the size, in bytes, of the memory block. (Input)

If the requested size would cause the `ibrk` area to cross the stack pointer, the request fails. You can use `freemem()` to determine the amount of stack remaining which is also the remaining `ibrk` area.

`ibrk()` is useful to request memory from a fixed-size area of memory, unlike `ebrk()` whose available memory is that of the entire system. The C I/O library functions request the first 2K of I/O buffers from this area, the remainder from `ebrk()`.

If successful, `ibrk()` returns zero. Otherwise, -1 is returned and the appropriate error code is placed in the global variable `errno`.



Note

Be very careful not to crowd out the stack with `ibrk()` calls. When stack checking is in effect, the program aborts with a ***Stack Overflow*** message if insufficient stack area exists to call a function.

Attributes

Operating System:

OS-9 and OS-9 for 68K

State:

User and System

Threads:

Safe

Re-entrant:

Yes

Library

`sys_clib.l`

See Also

`ebrk()`

`_freemem()`, `freemem()`

`sbrk()`

`stacksiz()`, `_stacksiz()`

inc()

Get Character From I/O Address

Syntax

```
#include <regs.h>
#include <types.h>
u_char inc(void *address);
```

Description

`inc()` returns the character indicated by the specified I/O address on the 80x86 processors or the memory address on all other processors.

address is the specified I/O address. (Input)

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

cpu.l

See Also

```
inl( )  
inw( )  
outc( )  
outl( )  
outw( )
```

Syntax

```
#include <strings.h>
char *index(
            const char    *ptr,
            int          ch);
```

Description

`index()` returns a pointer to the first occurrence of a character in a string. If the character is not found, `index()` returns `NULL`.

`ptr` is a pointer to the string. (Input)

ch is the character for which to search.
(Input)

Only the lower 8 bits of ch are used.

Attributes

Operating System: OS-9 and OS-9 for 68K

State: User and System

Threads: Safe

Re-entrant: Yes

Library

sys_clib.l

Example

This example looks for a period (.) in string and sets the pointer `ptr`. Note that `ch` is a character, not a pointer to a character:

```
func()
{
    char *ptr,*string;
    if((ptr = index(string,'.')))
        process(ptr);
    else printf("No '.' found!\n");
}
```

See Also

[rindex\(\)](#)
[strchr\(\)](#)
[strrchr\(\)](#)

inI()

Get Integer from I/O Address

Syntax

```
#include <regs.h>
#include <types.h>
u_int32 inl(void *address);
```

Description

`inl()` returns the integer indicated by the specified I/O address on the 80x86 processors or the memory address on all other processors.

address is the specified I/O address. (Input)

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

cpu.l

See Also

```
inc( )  
inw( )  
outc( )  
outl( )  
outw( )
```

intercept()

Set Up Process Signal Handler

Syntax

```
#include <signal.h>
int intercept(void (*icpthand)(int));
```

Description

`intercept()` instructs the operating system to pass control to a signal handler function when the process receives a signal.

`icpthand` is a pointer to the function.

If the signal handler function declares an int parameter, the function has access to the value of the received signal. On return from the signal handler function, the process resumes at the point in the program where it was interrupted by the signal.

If `icptphand` is zero, the signal handler is removed.

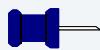


Note

A program does not receive an abort or quit signal from the keyboard (usually `^C` and `^E`) unless the program performs output to the terminal. This is because the operating system sends the abort/quit signals to the last process to perform I/O to the terminal. If you run the program from the shell and type `^E` before the program performs I/O to the terminal, the shell receives the signal and kills the running program. If a program requires immediate control of the terminal, perform some I/O to one of the standard paths such as printing a program banner or getting the terminal options with `_gs_opt()`.

Signal handlers dealing with asynchronous signals should be careful about the following:

- Any I/O using the C library (for example, `printf()`) cannot be performed inside both the intercept handler function and the main program. Consequently, avoid I/O within intercept functions.
- Calling any function that may be working with static storage structures.
- Calling any function that is not re-entrant.
- Modifying any static storage structure other than a variable of type `volatile sig_atomic_t`.



Note

Generally, `intercept()` should not be used with `signal()`. Only signals that are under `SIG_DFL` handling reach a handler installed with this function.

You cannot use floating point math within the intercept handler function.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`_os_intercept()`
`_os_send()`
`_os_sigmask()`
`signal()`

F\$ICpt	<i>OS-9 for 68K Technical Manual</i>
F\$Send	<i>OS-9 for 68K Technical Manual</i>
F\$SigMask	<i>OS-9 for 68K Technical Manual</i>
F_ICPT	<i>OS-9 Technical Manual</i>
F_SEND	<i>OS-9 Technical Manual</i>
F_SIGMASK	<i>OS-9 Technical Manual</i>

Example

As an example, this program is designed to clean up work files and exit when it receives a keyboard quit signal (signal 2 which, normally, is caused by typing ^E on the terminal):

```
#include <stdio.h>

char *tempname = "tempfile";
FILE *tempfp;
int quittime = 0;

/* the signal handler */
gotsignl(signum)
int signum;
{
    switch(signum) {
        case 2:                      * the quit signal
            quittime = 1;
            break;
        default:                     * ignore all others
            break;
    }
}
main()
{
    if((tempfp = fopen(tempname,"w")) == NULL)
        exit(_errmsg(errno),
              "can't open file - %s\n",tempname));
    intercept(gotsignl);
    do {
        do_work();
    } while(!quittime);
    fclose(tempfp);
    unlink(tempname);
    exit(_errmsg(1,"quittin' time!!!\n"));
}
```

Syntax

```
#include <ioctl.h>
#include <UNIX/os9def.h>
int ioctl(
    unsigned      int fd,
    unsigned      int request,
    caddr_t       arg);
```

Description

ioctl() performs some of the UNIX-like I/O control functions.

fd	is the path to the device that is affected by the function. (Input)
request	is the function to perform. (Input)
arg	has various meanings depending on the function to be performed. (Input)

The two functions that are supported are:

- FIONREAD — Return the number of bytes available for reading from the path. The number of bytes is returned at the `unsigned long` pointed to by `arg`. FIONREAD works with RBF, SCF, Pipe, and Sockman.
- FIONBIO — Set the I/O blocking status on the path. `arg` is a pointer to 0 (blocking I/O) or 1 (non-blocking I/O). FIONBIO works with Sockman (only `path_opts` block type it gets and checks for).

ioctl() returns zero on success for most requests. Some specialized requests may return non-zero values on success. See the specific description for the request. On failure, ioctl() returns -1 and sets `errno` to indicate an error.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

unix.1

inw()

Get Word from I/O Address

Syntax

```
#include <regs.h>
#include <types.h>
u_int16 inw(void *address);
```

Library

cpu.l

See Also

```
inc()
inl()
outc()
outl()
outw()
```

irq_change()

Set IRQ Level

Syntax

```
#include <regs.h>
#include <types.h>
status_code irq_change(status_code newval);
```

Description

`irq_change()` sets the interrupt level to `newval` (input). The next time `irq_change()` is called, it returns the previous interrupt level.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

cpu.l

See Also

[irq_disable\(\)](#)
[irq_enable\(\)](#)
[irq_maskget\(\)](#)
[irq_restore\(\)](#)
[irq_save\(\)](#)

Syntax

```
#include <regs.h>
void irq_disable(void);
```

Description

`irq_disable()` masks interrupts to the highest interrupt level available on the CPU. It is a system-state-only system call.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

cpu.l

See Also

[irq_change\(\)](#)
[irq_enable\(\)](#)
[irq_maskget\(\)](#)
[irq_restore\(\)](#)
[irq_save\(\)](#)

irq_enable()

Unmask Interrupts

Syntax

```
#include <regs.h>
void irq_enable(void);
```

Description

irq_enable() unmasks all interrupts to the lowest interrupt level available on the CPU.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

cpu.l

See Also

[irq_change\(\)](#)
[irq_disable\(\)](#)
[irq_maskget\(\)](#)
[irq_restore\(\)](#)
[irq_save\(\)](#)

irq_maskget()

Mask Interrupts; Return Previous Level

Syntax

```
#include <regs.h>
status_code irq_maskget (void);
```

Description

`irq_maskget()` masks all interrupts to the highest level available on the CPU. `irq_maskget()` returns the interrupt level before masking.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

cpu.l

See Also

[irq_change\(\)](#)
[irq_disable\(\)](#)
[irq_enable\(\)](#)
[irq_restore\(\)](#)
[irq_save\(\)](#)

irq_restore()

Restore Interrupt Level

Syntax

```
#include <regs.h>
void irq_restore(status_code stat_reg);
```

Description

irq_restore() sets the interrupt level.

stat_code specifies the value to put in the processor status register. (Input)

Attributes

Operating System:	OS-9
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

cpu.l

See Also

[irq_change\(\)](#)
[irq_disable\(\)](#)
[irq_enable\(\)](#)
[irq_maskget\(\)](#)
[irq_save\(\)](#)

irq_save()

Return Current Interrupt Level

Syntax

```
#include <regs.h>
status_code irq_save(void);
```

Description

`irq_save()` returns the current value of the status register.

Attributes

Operating System:	OS-9
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

cpu.l

See Also

[irq_change\(\)](#)
[irq_disable\(\)](#)
[irq_enable\(\)](#)
[irq_maskget\(\)](#)
[irq_restore\(\)](#)

isalnum()

See If Parameter Is Alphanumeric

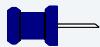
Syntax

```
#include <ctype.h>
int isalnum(int c);
```

Description

`isalnum()` returns a non-zero value for any character for which `isalpha()` or `isdigit()` is true.

c is the character to test. (Input)



Note

This routine, available as a function or a macro in `ctype.h`, does not operate well with values outside the range of an `unsigned char`.

The function checks for invalid input (values that are not representable as an `unsigned char` and not equal to the value of the macro `EOF`) and returns 0 for such input. The macro does not check for invalid input, which makes it faster than the function. The macro is the default. To use the function you must explicitly use the `undef` preprocessor directive or ensure that the macro expansion does not occur by placing the function name within parentheses.

For example: `b = (isalnum)(c);`

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

clib.l

See Also

[isalpha\(\)](#)
[iscntrl\(\)](#),
[isdigit\(\)](#)
[isgraph\(\)](#)
[islower\(\)](#)
[isprint\(\)](#)
[ispunct\(\)](#)
[isspace\(\)](#)
[isupper\(\)](#)
[isxdigit\(\)](#)

isalpha()

See If Parameter Is Alphabetic

Syntax

```
#include <ctype.h>
int isalpha(int c);
```

Description

`isalpha()` returns a non-zero value for any character for which `isupper()` or `islower()` is non-zero. `isalpha()` returns non-zero only for the characters for which `isupper()` or `islower()` is non-zero.

c is the character to test. (Input)



Note

This routine, available as a function or a macro in `ctype.h`, does not operate well with values outside the range of an `unsigned char`.

The function checks for invalid input (values that are not representable as an `unsigned char` and not equal to the value of the macro `EOF`) and returns 0 for such input. The macro does not check for invalid input, which makes it faster than the function. The macro is the default. To use the function you must explicitly use the `undef` preprocessor directive or ensure that the macro expansion does not occur by placing the function name within parentheses.

For example: `b = (isalpha)(c);`

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

clib.l

See Also

[isalnum\(\)](#)
[iscntrl\(\)](#)
[isdigit\(\)](#)
[isgraph\(\)](#)
[islower\(\)](#)
[isprint\(\)](#)
[ispunct\(\)](#)
[isspace\(\)](#)
[isupper\(\)](#)
[isxdigit\(\)](#)

_isascii(), isascii()

See If Parameter Is ASCII

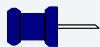
Syntax

```
#include <ctype.h>
int _isascii(int c);
int isascii(int c);
```

Description

`_isascii()` returns a non-zero value if `c` is an ASCII character (the value is in the range `0x00` through `0x7f`). Otherwise, it returns zero. Only `_isascii()` is available in strictly conforming ANSI mode. Otherwise, both are available. `_isascii()` and `isascii()` are macro definitions.

c is the character to test. (Input)



Note

This routine, available as a function or a macro in `ctype.h`, does not operate well with values outside the range of an `unsigned char`.

The function checks for invalid input (values that are not representable as an `unsigned char` and not equal to the value of the macro `EOF`) and returns 0 for such input. The macro does not check for invalid input, which makes it faster than the function. The macro is the default. To use the function you must explicitly use the `undef` preprocessor directive or ensure that the macro expansion does not occur by placing the function name within parentheses.

For example: `b = (isascii)(c);`

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

ctype.h

See Also

[isalnum\(\)](#)
[isalpha\(\)](#)
[iscntrl\(\)](#)
[isdigit\(\)](#)
[isgraph\(\)](#)
[islower\(\)](#)
[isprint\(\)](#)
[ispunct\(\)](#)
[isspace\(\)](#)
[isupper\(\)](#)
[isxdigit\(\)](#)

isatty()

Find Terminal Name

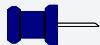
Syntax

```
#include <UNIX/os9def.h>
#include <setsys.h>
int isatty(int fd);
```

Description

`isatty()` returns 1 if `fd` is associated with a SCF device. Otherwise, it returns zero.

fd is the integer to test. (Input)



Note

This routine, available as a function or a macro in `ctype.h`, does not operate well with values outside the range of an `unsigned char`. The function checks for invalid input (values that are not representable as an `unsigned char` and not equal to the value of the macro `EOF`) and returns 0 for such input. The macro does not check for invalid input, which makes it faster than the function. The macro is the default. To use the function you must explicitly use the `undef` preprocessor directive or ensure that the macro expansion does not occur by placing the function name within parentheses.

For example: `b = (isatty)(c);`

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library

unix.l

iscntrl()

See If Parameter Is Control Code

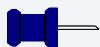
Syntax

```
#include <ctype.h>
int iscntrl(int c);
```

Description

`iscntrl()` is a C macro that returns a non-zero value for ASCII characters with values from 0 to 0x1f and 0x7f.

c is the character to test. (Input)



Note

This routine, available as a function or a macro in `ctype.h`, does not operate well with values outside the range of an `unsigned char`.

The function checks for invalid input (values that are not representable as an `unsigned char` and not equal to the value of the macro `EOF`) and returns 0 for such input. The macro does not check for invalid input, which makes it faster than the function. The macro is the default. To use the function you must explicitly use the `undef` preprocessor directive or ensure that the macro expansion does not occur by placing the function name within parentheses.

For example: `b = (iscntrl)(c);`

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

ctype.h

See Also

[isalnum\(\)](#)
[isalpha\(\)](#)
[isdigit\(\)](#)
[isgraph\(\)](#)
[islower\(\)](#)
[isprint\(\)](#)
[ispunct\(\)](#)
[isspace\(\)](#)
[isupper\(\)](#)
[isxdigit\(\)](#)

isdigit()

See If Parameter Is Digit

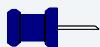
Syntax

```
#include <ctype.h>
int isdigit(int c);
```

Description

`isdigit()` is a C macro that returns a non-zero value for any decimal-digit character.

c is the character to test. (Input)



Note

This routine, available as a function or a macro in `ctype.h`, does not operate well with values outside the range of an `unsigned char`.

The function checks for invalid input (values that are not representable as an `unsigned char` and not equal to the value of the macro `EOF`) and returns 0 for such input. The macro does not check for invalid input, which makes it faster than the function. The macro is the default. To use the function you must explicitly use the `undef` preprocessor directive or ensure that the macro expansion does not occur by placing the function name within parentheses.

For example: `b = (isdigit)(c);`

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

ctype.h

See Also

[isalnum\(\)](#)
[isalpha\(\)](#)
[iscntrl\(\)](#)
[isgraph\(\)](#)
[islower\(\)](#)
[isprint\(\)](#)
[ispunct\(\)](#)
[isspace\(\)](#)
[isupper\(\)](#)
[isxdigit\(\)](#)

_iseuc_kana()

See If Parameter Is the First Byte of a Half Width Kaka-Kana Character

Syntax

```
#include <ctype.h>
int _iseuc_kana(int c);
```

Description

`_iseuc_kana()` is a C macro that returns a non-zero value if `c` is the first byte of a half-width Kata-Kana character. Otherwise, it returns zero. The first byte of a half-width Kata-Kana character is `0X8E`.

`c` is the character to test. (Input)



Note

This routine, available as a function or a macro in `ctype.h`, does not operate well with values outside the range of an `unsigned char`.

The function checks for invalid input (values that are not representable as an `unsigned char` and not equal to the value of the macro `EOF`) and returns 0 for such input. The macro does not check for invalid input, which makes it faster than the function. The macro is the default. To use the function you must explicitly use the `undef` preprocessor directive or ensure that the macro expansion does not occur by placing the function name within parentheses.

For example: `b = (iseuc_kana)(c);`

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

ctype.h

See Also

[isalnum\(\)](#)
[isalpha\(\)](#)
[iscntrl\(\)](#)
[isdigit\(\)](#)
[isgraph\(\)](#)
[islower\(\)](#)
[isprint\(\)](#)
[ispunct\(\)](#)
[isspace\(\)](#)
[isupper\(\)](#)
[isxdigit\(\)](#)

_iseuc1(), _iseuc2()

See If Parameter Is EUC Character

Syntax

```
#include <ctype.h>
int _iseuc1();
```

Description

`_iseuc1()` is a C macro that returns a non-zero value if `C` is the first byte of an EUC packed format JIS x 0208-1990 character. Otherwise, it returns zero. The valid range is between `0xa1` and `0xfe`.

`_iseuc2()` is the same thing for the second byte of the character and has the same valid range.

`c` is the character to test. (Input)



Note

This routine, available as a function or a macro in `ctype.h`, does not operate well with values outside the range of an `unsigned char`.

The function checks for invalid input (values that are not representable as an `unsigned char` and not equal to the value of the macro `EOF`) and returns 0 for such input. The macro does not check for invalid input, which makes it faster than the function. The macro is the default. To use the function you must explicitly use the `undef` preprocessor directive or ensure that the macro expansion does not occur by placing the function name within parentheses.

For example: `b = (iseuc1)(c);`

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

ctype.h

See Also

[isalnum\(\)](#)
[isalpha\(\)](#)
[iscntrl\(\)](#)
[isdigit\(\)](#)
[isgraph\(\)](#)
[islower\(\)](#)
[isprint\(\)](#)
[ispunct\(\)](#)
[isspace\(\)](#)
[isupper\(\)](#)
[isxdigit\(\)](#)

See If Parameter Is Printing Character

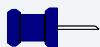
Syntax

```
#include <ctype.h>
int isgraph(int c);
```

Description

`isgraph()` is a C macro that returns a non-zero value if `c` is any printing character except a space. Otherwise, it returns zero.

`c` is the character to test. (Input)



Note

This routine, available as a function or a macro in `ctype.h`, does not operate well with values outside the range of an `unsigned char`.

The function checks for invalid input (values that are not representable as an `unsigned char` and not equal to the value of the macro `EOF`) and returns 0 for such input. The macro does not check for invalid input, which makes it faster than the function. The macro is the default. To use the function you must explicitly use the `undef` preprocessor directive or ensure that the macro expansion does not occur by placing the function name within parentheses.

For example: `b = (isgraph)(c);`

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

ctype.h

See Also

[isalnum\(\)](#)
[isalpha\(\)](#)
[iscntrl\(\)](#)
[isdigit\(\)](#)
[islower\(\)](#)
[isprint\(\)](#)
[ispunct\(\)](#)
[isspace\(\)](#)
[isupper\(\)](#)
[isxdigit\(\)](#)

_isjis_kana()

See If Parameter Is Hankaku-kana

Syntax

```
#include <ctype.h>
int _isjis_kana(int c);
```

Description

`_isjis_kana()` is a C macro that returns a non-zero value if `c` is a Hankaku-Kana. Otherwise, it returns zero. The valid range is between `0xa1` and `0xdf`.

`c` is the character to test. (Input)



Note

This routine, available as a function or a macro in `ctype.h`, does not operate well with values outside the range of an `unsigned char`.

The function checks for invalid input (values that are not representable as an `unsigned char` and not equal to the value of the macro `EOF`) and returns 0 for such input. The macro does not check for invalid input, which makes it faster than the function. The macro is the default. To use the function you must explicitly use the `undef` preprocessor directive or ensure that the macro expansion does not occur by placing the function name within parentheses.

For example: `b = (isjis_kana)(c);`

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

ctype.h

See Also

[isalnum\(\)](#)
[isalpha\(\)](#)
[iscntrl\(\)](#)
[isdigit\(\)](#)
[isgraph\(\)](#)
[islower\(\)](#)
[isprint\(\)](#)
[ispunct\(\)](#)
[isspace\(\)](#)
[isupper\(\)](#)
[isxdigit\(\)](#)

islower()

See If Parameter Is Lowercase

Syntax

```
#include <ctype.h>
int islower(int c);
```

Description

`islower()` is a C macro that tests for lowercase letters. `islower()` returns a non-zero value for ASCII characters from `0x61` (`a`) to `0x7a` (`z`).

c is the character to test. (Input)



Note

This routine, available as a function or a macro in `ctype.h`, does not operate well with values outside the range of an `unsigned char`. The function checks for invalid input (values that are not representable as an `unsigned char` and not equal to the value of the macro `EOF`) and returns 0 for such input. The macro does not check for invalid input, which makes it faster than the function. The macro is the default. To use the function you must explicitly use the `undef` preprocessor directive or ensure that the macro expansion does not occur by placing the function name within parentheses.

For example: `b = (islower)(c);`

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

ctype.h

See Also

[isalnum\(\)](#)
[isalpha\(\)](#)
[iscntrl\(\)](#)
[isdigit\(\)](#)
[isgraph\(\)](#)
[isprint\(\)](#)
[ispunct\(\)](#)
[isspace\(\)](#)
[isupper\(\)](#)
[isxdigit\(\)](#)

isprint()

See If Parameter Is Printable Character

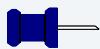
Syntax

```
#include <ctype.h>
int isprint(int c);
```

Description

`isprint()` is a C macro that tests for printing characters. It returns a non-zero value for ASCII characters with values from 0x20 to 0x7e.

c is the character to test. (Input)



Note

This routine, available as a function or a macro in `ctype.h`, does not operate well with values outside the range of an `unsigned char`.

The function checks for invalid input (values that are not representable as an `unsigned char` and not equal to the value of the macro `EOF`) and returns 0 for such input. The macro does not check for invalid input, which makes it faster than the function. The macro is the default. To use the function you must explicitly use the `undef` preprocessor directive or ensure that the macro expansion does not occur by placing the function name within parentheses.

For example: `b = (isprint)(c);`

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

ctype.h

See Also

[isalnum\(\)](#)
[isalpha\(\)](#)
[iscntrl\(\)](#)
[isdigit\(\)](#)
[isgraph\(\)](#)
[islower\(\)](#)
[ispunct\(\)](#)
[isspace\(\)](#)
[isupper\(\)](#)
[isxdigit\(\)](#)

ispunct()

See If Parameter Is Punctuation Character

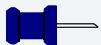
Syntax

```
#include <ctype.h>
int ispunct(int c);
```

Description

`ispunct()` is a C macro that returns a non-zero value for any printing character that is neither a space nor a character for which `isalnum()` is non-zero.

c is the character to test. (Input)



Note

This routine, available as a function or a macro in `ctype.h`, does not operate well with values outside the range of an `unsigned char`. The function checks for invalid input (values that are not representable as an `unsigned char` and not equal to the value of the macro `EOF`) and returns 0 for such input. The macro does not check for invalid input, which makes it faster than the function. The macro is the default. To use the function you must explicitly use the `undef` preprocessor directive or ensure that the macro expansion does not occur by placing the function name within parentheses.

For example: `b = (ispunct)(c);`

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

ctype.h

See Also

[isalnum\(\)](#)
[isalpha\(\)](#)
[iscntrl\(\)](#)
[isdigit\(\)](#)
[isgraph\(\)](#)
[islower\(\)](#)
[isprint\(\)](#)
[isspace\(\)](#)
[isupper\(\)](#)
[isxdigit\(\)](#)

_issjis1()

See If Parameter Is First Byte of SJIS Kanji

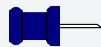
Syntax

```
#include <ctype.h>
int _issjis1(int c);
```

Description

`_issjis1()` is a C macro that returns a non-zero value if `c` is the first byte of a Shift-JIS (Japanese Industrial Standard) Kanji character. Otherwise, it returns zero. The valid ranges for Shift-JIS first bytes are between `0x81` and `0x9f` or between `0xe0` and `0xfc`.

`c` is the character to test. (Input)



Note

This routine, available as a function or a macro in `ctype.h`, does not operate well with values outside the range of an `unsigned char`.

The function checks for invalid input (values that are not representable as an `unsigned char` and not equal to the value of the macro `EOF`) and returns 0 for such input. The macro does not check for invalid input, which makes it faster than the function. The macro is the default. To use the function you must explicitly use the `undef` preprocessor directive or ensure that the macro expansion does not occur by placing the function name within parentheses.

For example: `b = (issjis1)(c);`

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

ctype.h

See Also

[isalnum\(\)](#)
[isalpha\(\)](#)
[iscntrl\(\)](#)
[isdigit\(\)](#)
[isgraph\(\)](#)
[islower\(\)](#)
[isprint\(\)](#)
[ispunct\(\)](#)
[_issjis2\(\)](#)
[isspace\(\)](#)
[isupper\(\)](#)
[isxdigit\(\)](#)

See If Parameter Is Second Byte of SJIS Kanji

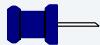
Syntax

```
#include <ctype.h>
int _issjis2(int c);
```

Description

`_issjis2()` is a C macro that returns a non-zero value if `c` is the second byte of a Shift-JIS (Japanese Industrial Standard) Kanji character. Otherwise, it returns zero. The valid range for Shift-JIS second bytes are between `0x40` and `0xfc`.

`c` is the character to test. (Input)
`c` cannot equal `0x7f`.



Note

This routine, available as a function or a macro in `ctype.h`, does not operate well with values outside the range of an `unsigned char`. The function checks for invalid input (values that are not representable as an `unsigned char` and not equal to the value of the macro `EOF`) and returns 0 for such input. The macro does not check for invalid input, which makes it faster than the function. The macro is the default. To use the function you must explicitly use the `undef` preprocessor directive or ensure that the macro expansion does not occur by placing the function name within parentheses.

For example: `b = (issjis2)(c);`

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

ctype.h

See Also

[isalnum\(\)](#)
[isalpha\(\)](#)
[iscntrl\(\)](#)
[isdigit\(\)](#)
[isgraph\(\)](#)
[islower\(\)](#)
[isprint\(\)](#)
[ispunct\(\)](#)
[_issjis1\(\)](#)
[isspace\(\)](#)
[isupper\(\)](#)
[isxdigit\(\)](#)

isspace()

See If Parameter Is White Space

Syntax

```
#include <ctype.h>
int isspace(int c);
```

Description

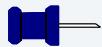
isspace() is a C macro that returns a non-zero value for standard white space characters.

c is the character to test. (Input)

The standard white space characters are:

Table 2-14 isspace() White Space Characters

Description	Character
Space	
Form feed	\f
Newline	\n
Carriage return	\r
Horizontal tab	\t
Vertical tab	\v



Note

This routine, available as a function or a macro in `ctype.h`, does not operate well with values outside the range of an `unsigned char`.

The function checks for invalid input (values that are not representable as an `unsigned char` and not equal to the value of the macro `EOF`) and returns 0 for such input. The macro does not check for invalid input, which makes it faster than the function. The macro is the default. To use the function you must explicitly use the `undef` preprocessor directive or ensure that the macro expansion does not occur by placing the function name within parentheses.

For example: `b = (isspace)(c);`

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`ctype.h`

See Also

[isalnum\(\)](#)
[isalpha\(\)](#)
[iscntrl\(\)](#)
[isdigit\(\)](#)
[isgraph\(\)](#)
[islower\(\)](#)
[isprint\(\)](#)
[ispunct\(\)](#)
[isupper\(\)](#)
[isxdigit\(\)](#)

isupper()

See If Parameter Is Uppercase

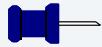
Syntax

```
#include <ctype.h>
int isupper(int c);
```

Description

`isupper()` is a C macro that returns a non-zero value for uppercase letters. `isupper()` returns a non-zero value for ASCII characters with values in the range from `0x41` (`A`) to `0x5a` (`Z`).

c is the character to test. (Input)



Note

This routine, available as a function or a macro in `ctype.h`, does not operate well with values outside the range of an `unsigned char`. The function checks for invalid input (values that are not representable as an `unsigned char` and not equal to the value of the macro `EOF`) and returns 0 for such input. The macro does not check for invalid input, which makes it faster than the function. The macro is the default. To use the function you must explicitly use the `undef` preprocessor directive or ensure that the macro expansion does not occur by placing the function name within parentheses.

For example: `b = (isupper)(c);`

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

ctype.h

See Also

[isalnum\(\)](#)
[isalpha\(\)](#)
[iscntrl\(\)](#)
[isdigit\(\)](#)
[isgraph\(\)](#)
[islower\(\)](#)
[isprint\(\)](#)
[ispunct\(\)](#)
[isspace\(\)](#)
[isxdigit\(\)](#)

isxdigit()

See If Parameter Is Hexadecimal

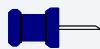
Syntax

```
#include <ctype.h>
int isxdigit(int c);
```

Description

`isxdigit()` is a C macro that returns a non-zero value for any hexadecimal-digit character.

c is the character to test. (Input)



Note

This routine, available as a function or a macro in `ctype.h`, does not operate well with values outside the range of an `unsigned char`.

The function checks for invalid input (values that are not representable as an `unsigned char` and not equal to the value of the macro `EOF`) and returns 0 for such input. The macro does not check for invalid input, which makes it faster than the function. The macro is the default. To use the function you must explicitly use the `undef` preprocessor directive or ensure that the macro expansion does not occur by placing the function name within parentheses.

For example: `b = (isxdigit)(c);`

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

ctype.h

See Also

[isalnum\(\)](#)
[isalpha\(\)](#)
[iscntrl\(\)](#)
[isdigit\(\)](#)
[isgraph\(\)](#)
[islower\(\)](#)
[isprint\(\)](#)
[ispunct\(\)](#)
[isspace\(\)](#)
[isupper\(\)](#)

Syntax

```
#include <time.h>
int _julian(
    int      *time,
    int      *date);
```

Description

`_julian()` converts the time and date from Gregorian format to the Julian equivalents.

`time` and `date` are pointers to the Gregorian format time and date values. (Input/Output)

The following format is assumed:

Figure 2-5 `_julian()` Input Date and Time Formats

Time:	0	Hour (0-23)	Minute	Second
Date:	Year (2-bytes)	Month	Day	

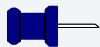
Byte 0 Byte 1 Byte 2 Byte 3

`_julian()` modifies the objects to which its parameters point as follows:

Figure 2-6 `_julian()` Output Date and Time Formats

Time:	Seconds since midnight (0-86399)
Date:	Julian Day Number

If an error occurs, -1 is returned and the appropriate error code is placed in the global variable `errno`.



Note

Be certain to pass pointers for `date` and `time` values.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

Example

```
main()
{
    int date,time,tick;
    short day;
    _sysdate(0,&time,&date,&day,&tick);
    _julian(&time,&date);
    printf("The Julian date is %d. \
           Cinderella needs to be home in %d
seconds!\n",date,86400-time);
}
```

See Also

[`_sysdate\(\)`](#) (OS-9 for 68K)
[`_os_gregorian\(\)`](#) (OS-9 for 68K)
[`_os_julian\(\)`](#) (OS-9 for 68K)

F\$Time
F\$Julian
F_TIME

OS-9 for 68K Technical Manual
OS-9 for 68K Technical Manual
OS-9 Technical Manual

Syntax

```
#include <signal.h>
int kill(
    int      pid,
    int      sigcode);
```

Description

`kill()` sends a signal to a process. Both the sending and receiving process must have the same user number unless the sending process' user number is that of the super user (0).

<code>pid</code>	is the process ID of the process to kill. (Input)
<code>sigcode</code>	is the signal code to send. Only the lower 16 bits of <code>sigcode</code> are used. (Input)
	The value in <code>sigcode</code> is sent as a signal to the process specified by <code>pid</code> . You can pass any value in <code>sigcode</code> . The conventional code numbers are defined in the <code>signal.h</code> header file.

If an error occurs, -1 is returned and the appropriate error code is placed in the global variable `errno`.



Note

The super user can send a signal to all processes running on the system if the `pid` is zero.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

sys_clib.l

See Also

[_os_send\(\)](#)
[F\\$Send](#)
[F_SEND](#)
[kill command](#)

OS-9 for 68K Technical Manual or
OS-9 Technical Manual
Using OS-9 for 68K or *Using OS-9*

Syntax

```
#include <stdlib.h>
long int labs(long int num);
```

Description

`labs()` returns the absolute value of `num`. `num` and the returned value each have type `long int`.

`num` is the value to use. (Input)

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

See Also

[abs\(\)](#)

Allocate Storage for Array (Low-Overhead)

Syntax

```
#include <stdlib.h>
void *_lalloc(
                unsigned long      nel,
                unsigned long      elsize);
```

Description

`_lalloc()` allocates space for an array. The allocated memory is cleared to zeroes.

<code>nel</code>	specifies the number of elements in the array. (Input)
<code>elsize</code>	specifies the size of each element. (Input)

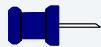
`_lalloc()` calls `_lmalloc()` to allocate memory.

- If the allocation is successful, `_lalloc()` returns a pointer to the area.
- If the allocation fails, `_lalloc()` returns NULL.



Note

Use of the low-overhead allocation functions (`_lalloc()`, `_lmalloc()`, and `_lrealloc()`) instead of the general allocation functions (`calloc()`, `malloc()`, and `realloc()`) saves 8 bytes per allocation because the low-overhead functions do not save the allocation size or the 4-byte check value.



Note

Use extreme care to ensure that you only access the assigned memory. Modifying addresses immediately above or below the assigned memory causes unpredictable program results.

The low-overhead functions require that you keep track of the sizes of allocated spaces in memory. Note the `_lfree()` and `_lrealloc()` parameters.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

`clib.l`

See Also

`calloc()`
`free()`
`_lfree()`
`_lmalloc()`
`_lrealloc()`
`malloc()`
`_mallocmin()`
`_os9_srqmem()`
`realloc()`

ldexp()

Multiply Float by Exponent of 2

Syntax

```
#include <math.h>
double ldexp(
    double   fp,
    int      exp);
```

Description

ldexp() multiplies a floating point value by an integral power of two.
ldexp() returns the value equal to $fp * 2^{\text{exp}}$.

fp	is the floating point value. (Input)
exp	is the integral power of two. (Input)

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

clib.lib

Syntax

```
#include <stdlib.h>
ldiv_t ldiv(
    long int    numer,
    long int    denom);
```

Description

`ldiv()` returns the quotient and remainder of the division of the numerator by the denominator. The parameters and member of the returned structure are all of type `long int`.

<code>numer</code>	is the numerator. (Input)
<code>denom</code>	is the denominator. (Input)

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

See Also

[div\(\)](#)

_lfree()

Return Memory (Low-Overhead)

Syntax

```
#include <stdlib.h>
void _lfree(
    void             *ptr,
    unsigned long    size);
```

Description

`_lfree()` returns a block of memory granted by `_lalloc()` or `_lmalloc()`. The memory is returned to a pool of memory for later re-use by `calloc()`, `_lalloc()`, `_lmalloc()`, or `malloc()`.

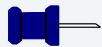
`ptr` is a pointer to the memory block. (Input)
`size` specifies the size of the memory block. (Input)

`_lfree()` never returns an error.



Note

Use of the low-overhead allocation functions (`_lalloc()`, `_lmalloc()`, and `_lrealloc()`) instead of the general allocation functions (`calloc()`, `malloc()`, and `realloc()`) saves 8 bytes per allocation because the low-overhead functions do not save the allocation size or the 4-byte check value.



Note

If `_lfree()` is used with something other than a pointer returned by `_lalloc()` or `_lmalloc()`, the memory lists maintained by `_lmalloc()` and `malloc()` are corrupted and programs behave unpredictably.

The low-overhead functions require that you keep track of the sizes of allocated spaces in memory.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

`clib.l`

See Also

`calloc()`
`free()`
`_freemin()`
`_lalloc()`
`_lmalloc()`
`_lrealloc()`
`malloc()`
`_os_srtmem()`
`realloc()`

Syntax

```
#include <UNIX/os9def.h>
void lftocr(
    char    *buf ,
    int      n) ;
```

Description

`lftocr()` converts all linefeed characters to carriage-return characters in `buf` for a span of `n` characters.

<code>buf</code>	is a pointer to a buffer. (Input/Output).
<code>n</code>	is the number of characters spanned. (Input)

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`unix.l`

_Imalloc()

Allocate Memory from Arena (Low-Overhead)

Syntax

```
#include <stdlib.h>
void *_lmalloc(unsigned long size);
```

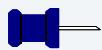
Description

`_lmalloc()` returns a pointer to a memory block. The pointer is suitably aligned for storing data of any type.

size specifies the size of the memory block in bytes. (Input)

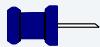
`_lmalloc()` maintains an amount of memory, called an arena, from which it grants memory requests. `_lmalloc()` searches its arena for a block of free memory large enough for the request and, in the process, joins adjacent blocks of free space returned by either `_lfree()` or `free()`. If sufficient memory is not available in the arena, `_lmalloc()` calls `_srgmem()` to get more memory from the system.

`_lmalloc()` returns `NULL` if no memory is available or if the arena is detected to be corrupted.



Note

Use of the low-overhead allocation functions (`_lcalloc()`, `_lmalloc()`, and `_lrealloc()`) instead of the general allocation functions (`calloc()`, `malloc()`, and `realloc()`) saves 8 bytes per allocation because the low-overhead functions do not save the allocation size or the 4-byte check value.



Note

Use extreme care to ensure that you only access the memory assigned by `_lmalloc()`. Modifying addresses immediately above or below the assigned memory or passing `_lfree()` a value not assigned by `_lmalloc()` causes unpredictable program results.

The low-overhead functions require that you keep track of the sizes of allocated spaces in memory. Note the `_lfree()` and `_lrealloc()` parameters.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

`clib.l`

See Also

[`_lalloc\(\)`](#)
[`_lfree\(\)`](#)
[`_lrealloc\(\)`](#)
[`malloc\(\)`](#)
[`_mallocmin\(\)`](#)
[`calloc\(\)`](#)
[`free\(\)`](#)
[`_os_srqmem\(\)`](#)
[`realloc\(\)`](#)

Syntax

```
#include <locale.h>
struct lconv *localeconv(void);
```

Description

`localeconv()` sets the components of an object with type `struct lconv` with values appropriate for formatting numeric quantities according to the rules of the current locale.

The members of the structure with type `char *` are pointers to strings, any of which (except `decimal_point`) can point to a NULL value to indicate that the value is not available in the current locale or is of zero length. The members with type `char` are non-negative numbers, any of which can be `CHAR_MAX` to indicate that the value is not available in the current locale. The members include the following:

`char *decimal_point`

The decimal point character used to format non-monetary quantities.

`char *thousands_sep`

The character used to separate groups of digits before the decimal point character in formatted non-monetary quantities.

char *grouping

A string whose elements indicate the size of each group of digits in formatted, non-monetary quantities. The string may contain the following:

CHAR_MAX	No further grouping is to be performed.
0	The previous element is to be repeatedly used for the remaining digits.
other	The integer value is the number of digits that comprise the current group. The next element is examined to determine the size of the next group of digits before the current group.

char *int_curr_symbol

The international currency symbol that applies to the current locale. The first three characters contain the alphabetic international currency symbol. The fourth character (immediately preceding the null character) is the character used to separate the international currency symbol from the monetary quantity.

char *currency_symbol

The local currency symbol applicable to the current locale.

char *mon_decimal_point

The decimal point used to format monetary quantities.

char *mon_thousands_sep

The separator for groups of digits before the decimal point in formatted monetary quantities.

`char *mon_grouping`

A string whose elements indicate the size of each group of digits in formatted monetary quantities. The string may contain the following:

<code>CHAR_MAX</code>	No further grouping is to be performed.
<code>0</code>	The previous element is to be repeatedly used for the remaining digits.
<code>other</code>	The integer value is the number of digits that comprise the current group. The next element is examined to determine the size of the next group of digits before the current group.

`char *positive_sign`

The string used to indicate a non-negative-valued formatted monetary quantity.

`char *negative_sign`

The string used to indicate a negative-valued formatted monetary quantity.

`char int_frac_digits`

The number of fractional digits (those after the decimal point) to display in an internationally formatted monetary quantity.

`char p_cs_precedes`

Set to one or zero if the `currency_symbol` respectively precedes or succeeds the value for a non-negative formatted monetary quantity.

`char p_sep_by_space`

Set to one or zero if the `currency_symbol` respectively is or is not separated by a space from the value for a non-negative formatted monetary quantity.

`char n_cs_precedes`

Set to one or zero if the `currency_symbol` respectively precedes or succeeds the value for a negative formatted monetary quantity.

`char n_sep_by_space`

Set to one or zero if the `currency_symbol` respectively is or is not separated by a space from the value for a negative formatted monetary quantity.

`char p_sign_posn`

Set to a value indicating the positioning of the `positive_sign` for a non-negative formatted monetary quantity. The value may be:

- 0 Parentheses surround the quantity and currency symbol.
- 1 The sign string precedes the quantity and currency symbol.
- 2 The sign string succeeds the quantity and currency symbol.
- 3 The sign string immediately precedes the currency symbol.
- 4 The sign string immediately succeeds the currency symbol.

`char n_sign_posn`

Set to a value indicating the positioning of the `negative_sign` for a negative formatted monetary quantity. The values are the same as for `p_sign_posn`.

`localeconv()` returns a pointer to the filled in object. You cannot modify the structure pointed to by the return value, but a subsequent call to `localeconv()` can overwrite the structure. In addition, calls to `setlocale()` with categories `LC_ALL`, `LC_MONETARY`, or `LC_NUMERIC` may overwrite the contents of the structure.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`clib.lib`

localtime()

Convert Calendar Time to Local Time

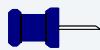
Syntax

```
#include <time.h>
struct tm *localtime(time_t *timer);
```

Description

`localtime()` converts a calendar time into a broken down time, expressed as local time.

timer is a pointer to the calendar time. (Input)



Note

`localtime()` returns a pointer to a static area which may be overwritten. To ensure data integrity, use the value or save it immediately.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

clib.l

See Also

`mktime()` (for more information on the time structure `tm`) `time()`

Syntax

```
#include <math.h>
double log(double x);
```

Description

`log()` returns the natural logarithm of `x` (input). A domain error occurs, `EDOM` stored in `errno`, if `x` is negative. A range error, `ERANGE` stored in `errno`, occurs if `x` is zero. `log()` returns `HUGE_VAL` for both errors.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

Possible Errors

`EDOM`

`ERANGE`

Syntax

```
#include <math.h>
double log10(double x);
```

Description

`log10()` returns the base-ten logarithm of `x` (input). A domain error, `EDOM` stored in `errno`, occurs if `x` is negative. A range error, `ERANGE` stored in `errno`, occurs if `x` is zero. `log10()` returns `HUGE_VAL` for both errors.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

Possible Errors

`EDOM`
`ERANGE`

Syntax

```
#include <setjmp.h>
void longjmp(
    jmp_buf    env,
    int        val);
```

Description

`longjmp()` and `setjmp()` allow program control to return directly to a higher level function. They are most useful when dealing with errors and signals encountered in a low-level routine.

`env` is the program environment buffer.
(Input)

`val` is the error status value. (Input)

`longjmp()` restores the environment saved by the most recent call to `setjmp()` in the same execution of the program, with the corresponding `jmp_buf` parameter. If there has been no such call or if the function containing the `setjmp()` call has returned in the interim, the behavior is undefined.

All accessible objects have values as of the time `longjmp()` was called, except that the value of objects of automatic storage duration that are local to the function containing the call to the corresponding `setjmp()` call and `longjmp()` call are indeterminate.

Because it bypasses the usual function call and return mechanisms, `longjmp()` executes correctly in contexts of interrupts, signals, and any of their associated functions. However, if `longjmp()` is called from a nested signal handler (that is, from a function called as a result of a signal raised during the handling of another signal), the behavior is undefined.

After `longjmp()` is completed, program execution continues as if the corresponding call to `setjmp()` had just returned the value specified by `val`. `longjmp()` cannot cause `setjmp()` to return the value zero. If `val` is zero, `setjmp()` returns the value 1.



WARNING

Do not call `longjmp()` before `env` is initialized by `setjmp()` or if the function calling `setjmp()` has already returned.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.lib`

See Also

[setjmp\(\)](#)

_lrealloc()

Resize Block of Memory (Low-Overhead)

Syntax

```
#include <stdlib.h>
void *_lrealloc(
    void *oldptr,
    unsigned long newsize,
    unsigned long oldsize);
```

Description

`_lrealloc()` resizes a block of memory.

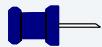
<code>oldptr</code>	is a pointer to the memory block. (Input) <code>oldptr</code> should be a value returned by a previous call to <code>_lmalloc()</code> , <code>_lcalloc()</code> , or <code>_lrealloc()</code> .
<code>newsize</code>	specifies the size to use for the memory block. (Input)
<code>oldsize</code>	is the size of the old memory block. (Input)

`_lrealloc()` returns a pointer to a new memory block of size `newsize`. The pointer is aligned to store data of any type.

If `newsize` is smaller than `oldsize`, the contents of the old block are truncated and placed in the new block. Otherwise, all of the old block's contents begin the new block.

The results of `_lrealloc(NULL, 10, 0)` and `_lmalloc(10)` are the same.

`_lrealloc()` returns `NULL` if the requested memory is not available or if `newsize` is specified as zero.



Note

Use of the low-overhead allocation functions (`_lalloc()`, `_lmalloc()`, and `_lrealloc()`) instead of the general allocation functions (`calloc()`, `malloc()`, and `realloc()`) saves 8 bytes per allocation because the low-overhead functions do not save the allocation size or the 4-byte check value. The low-overhead functions require that you keep track of the sizes of allocated spaces in memory.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

`clib.l`

See Also

`calloc()`
`free()`
`_freemin()`
`_lalloc()`
`_lfree()`
`_lmalloc()`
`malloc()`
`_mallocmin()`
`_os_srqmem()`
`_os_srtmem()`
`realloc()`

Syntax

```
#include <modes.h>
long lseek(
    int      path,
    long    position,
    int      place);
```

Description

`lseek()` repositions the file pointer for the file open on `path` to the byte offset given in `position`.

`path` is the path number of the open file. (Input)

`position` is the new file position offset. (Input)

`place` determines from which file position the offset is based. (Input)

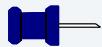
0 — From the beginning of the file.

1 — From the current position.

2 — From the end of the file.

Seeking to a location beyond end-of-file for a file open for writing and then writing to it creates a hole in the file. The hole contains data with no particular value, usually garbage remaining on the disk.

The returned value is the resulting position in the file. If an error occurs, -1 is returned and the appropriate error code is placed in the global variable `errno`.



Note

On OS-9, the file pointer is a full unsigned 32-bits.

Do not use `lseek()` on a buffered file because the buffering routines use `fseek()` to keep track of the file pointer.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

[`fseek\(\)`](#)
[`_os_seek\(\)`](#)

`I$Seek`
`I_SEEK`

OS-9 for 68K Technical Manual
OS-9 Technical Manual

Syntax

```
#include <modes.h>
int makdir(
            const char    *name,
            int             mode,
            int             perm[ ,
            int             size));
```

Description

`makdir()` creates a directory file.

`name` is a pointer to the name of the directory. (Input)

`mode` specifies the file access mode. (Input)

Only the lower 16 bits of `mode` are used. (Input)

`perm` specifies the file attribute permissions. (Input)

Only the lower 16 bits of `perm` are used. If the `FAM_SIZE` (or `S_ISIZE`) bit is set in `mode`, the initial size of the directory is set to `size`.

`size` is the size of the directory. (Input)

If successful, `makdir()` returns zero. Otherwise, -1 is returned and the appropriate error code is placed in the global variable `errno`.

Attributes

Operating System: OS-9 and OS-9 for 68K

State: User and System

Threads: Safe

Re-entrant: Yes

Library

`sys_clib.l`

See Also

[_os_mkdir\(\)](#)

[I\\$MakDir](#)

[I_MAKDIR](#)

OS-9 for 68K Technical Manual

OS-9 Technical Manual

make_gdesc()

Make Global Descriptor Table Entry

Syntax

```
#include <regs.h>
#include <types.h>
void *make_gdesc(
    void *tbl_entry_addr,
    u_int32 offset,
    u_int16 segment_attr);
```

Description

make_gdesc() makes an entry in the global descriptor table. It returns a pointer to the next entry.

tbl_entry_addr	is a pointer to the address of the local descriptor table or the global descriptor table. (Input)
offset	is the offset into the global descriptor table. (Input)
segment_attr	is the segment attributes. (Input)



Note

make_gdesc() is only supported by the 80x86 family of processors version of the compiler.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

cpu.l

make_idesc()

Make Interrupt Descriptor Table Entry

Syntax

```
#include <regs.h>
#include <types.h>

void *make_idesc(
                void          *tbl_entry_addr,
                u_int32       offset,
                u_int16       segment_attr,
                u_int16       selector);
```

Description

make_idesc() makes an entry in the interrupt descriptor table. It returns a pointer to the next entry.

tbl_entry_addr	is a pointer to the interrupt descriptor table. (Input)
offset	is an offset into the table. (Input)
segment_attr	specifies the segment attributes. (Input)
selector	specifies the selector entry value. (Input)



Note

make_idesc() is only supported by the 80x86 family of processors version of the compiler.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

cpu.l

make_module()

Create Module

Syntax

```
#include <memory.h>
#include <module.h>
mh_com *make_module(
    const char      *name,
    int             size,
    int             attr,
    int             perm,
    int             typelang,
    int             color);
```

Description

`make_module()` creates a memory module with the specified name and attributes in the specified memory. `make_module()` is similar to `_mkdata_module()`. However, `make_module()` allows you to specify the type/language of the module and color of memory in which to make the module.

mod_exec	is defined in the <code>module.h</code> header file.
name	is a pointer to the name of the module. (Input)
size	specifies the desired memory size of the module in bytes. (Input) The size value does not include the module header and CRC bytes.
size	is the amount of memory available for actual use. (Input) The memory in the module is initially cleared to zeroes.

<code>attr</code>	specifies the module's attribute and revision level. (Input)
<code>perm</code>	Only the lower 16 bits of <code>attr</code> are used.
<code>typelang</code>	specifies the module permissions. (Input)
<code>Only the lower 16 bits of perm are used.</code>	Only the lower 16 bits of <code>typelang</code> are used.
<code>color</code>	specifies the type and language of the module. (Input)
	Only the lower 16 bits of <code>typelang</code> are used.
<code>color</code>	indicates the specific memory type in which to load the module. (Input)
	The <code>memory.h</code> header file contains definitions of the memory types that you can specify:

Table 2-15 `make_module()` Color Memory Types

Memory Type	Description
<code>SYSRAM</code> or <code>MEM_SYS</code>	System RAM memory
<code>VIDEO1</code>	Video memory for plane A
<code>VIDEO2</code>	Video memory for plane B
<code>MEM_ANY</code>	No specific memory type

If `color` is `MEM_ANY`, the module is made in whatever memory the system allocates.

`make_module()` returns a pointer to the beginning of the module header. If the module cannot be created, `-1` is returned and the appropriate error code is placed in the global variable `errno`.



Note

The name of the module created by `make_module()` always begins at offset \$34 within the module. This implies that program modules, trap handlers, file managers, device drivers, and device descriptors cannot be made conveniently by this call.

Attributes

Operating System:	OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

[_os_datmod\(\)](#)

F\$DatMod service request

OS-9 for 68K Technical Manual

Syntax

```
#include <stdlib.h>
void *malloc(size_t size);
```

Description

`malloc()` allocates space for an object. The value of the object is indeterminate.

size is the size of the memory block to allocate. (Input)

If size is zero, NULL is returned.

`malloc()` returns either a null pointer (if sufficient free memory is not available) or a pointer to the allocated space.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

clib.l

See Also

```
calloc()
free()
_os_srqmem()
_os9_srqmem()
realloc()

F$SRqMem
F_SROMEM
```

OS-9 for 68K Technical Manual

OS-9 Technical Manual

_mallocmin()

Set Minimum Allocation Size

Syntax

```
#include <stdlib.h>
void _mallocmin(unsigned size);
```

Description

`_mallocmin()` sets the minimum amount of memory that allocation functions may request from the system.

`size` is the minimum allocation size in bytes.
(Input)

`size` cannot be less than the system memory block size. Size must be greater than or equal to the larger of the system memory block size or 4K. If a smaller size is requested, `size` is automatically set to the system memory block size.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

clib.l

See Also

[calloc\(\)](#)
[_lalloc\(\)](#)
[_lmalloc\(\)](#)
[_lrealloc\(\)](#)
[malloc\(\)](#)
[realloc\(\)](#)

Determine Number of Bytes in Multibyte Character

Syntax

```
#include <stdlib.h>
int mblen(
    const char *str,
    size_t num);
```

Description

`mblen()` determines the number of bytes contained in the multibyte character pointed to by `str`, if `str` is not a null pointer. `mblen()` is similar to the following; however, the shift state of `mbtowc()` is unaffected:

```
mbtowc( (wchar_t *)0, s, n);
```

`str` is a pointer to a multibyte character.
(Input)

num is the number of bytes to search. (Input)

If `str` is a null pointer, `mblen()` returns a non-zero value (if the multibyte character encodings have state-dependent encodings) or returns a zero value.

If `str` is not a null pointer, `mblen()` returns:

- 0 if `str` points to the null character.
 - The number of bytes contained in the multibyte character if the next `num` or fewer bytes form a valid multibyte character.
 - `-1` if they do not form a valid multibyte character.

`mblen()` is supported for both the C and JAPAN locales.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

clib.l

See Also

[mbtowc\(\)](#)

Syntax

```
#include <stdlib.h>

size_t mbstowcs(
    wchar_t *pwcs,
    const char *str,
    size_t num);
```

Description

`mbstowcs()` converts a sequence of multibyte characters that begins in the initial shift state into a sequence of corresponding codes and stores not more than `num` codes in an array. No multibyte characters that follow a null character are examined or converted. A null character is converted into a code with value zero. Each multibyte character is converted as if by a call to `mbtowc()`, except that the shift state of `mbtowc()` is unaffected.

<code>pwcs</code>	is a pointer to the array into which to store the converted values. (Output)
<code>str</code>	is a pointer to the original sequence of multibyte characters. (Input)
<code>num</code>	specifies the maximum number of elements to modify. (Input)

No more than `num` elements are modified in `pwcs`. If copying takes place between overlapping objects, the behavior is undefined.

If an invalid multibyte character is encountered, `mbstowc()` returns `(size_t)-1`. Otherwise, `mbstowcs()` returns the number of array elements modified, not including a terminating zero code, if any.

`mbstowcs()` is supported for both the C and JAPAN locales.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

clib.l

See Also

[wcstombs\(\)](#)

Syntax

```
#include <stdlib.h>
int mbtowc(
    wchar_t           *pwc,
    const char       *mbchar,
    size_t            maxnum);
```

Description

`mbtowc()` determines the number of bytes contained in the multibyte character pointed to by `mbchar`, if `mbchar` is not a null pointer. It then determines the code for the value of type `wchar_t` that corresponds to that multibyte character. The value of the code corresponding to the null character is zero. If the multibyte character is valid and `pwc` is not a null pointer, `mbtowc()` stores the code in `pwc`. At most `maxnum` bytes of `mbchar` are examined.

`pwc` is a pointer to the array in which to store the code. (Output)

`mbchar` is a pointer to a multibyte character. (Input)

If `mbchar` is a null pointer, `mbtowc()` returns a non-zero value if multibyte character encodings have state-dependent encodings or returns a zero value.

If `mbchar` is not a null pointer, `mbtowc()` either returns:

- 0 if `mbchar` is a pointer to the null character.
- The number of bytes contained in the converted multibyte character if the next `maxnum` or fewer bytes form a valid multibyte character.
- -1 if they do not form a valid multibyte character.

maxnum specifies the maximum number of bytes to examine. (Input)

The returned value is never greater than maxnum or MB_CUR_MAX.

`mbtowcs()` is supported for both the C and JAPAN locales.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

See Also

[wctomb\(\)](#)

Syntax

```
#include <string.h>
void *memchr(
    const void    *mem_area,
    int          chr,
    size_t        size);
```

Description

`memchr()` locates the first occurrence of `chr`, converted to an `unsigned char`, in the initial `size` characters, each interpreted as `unsigned char`, of `mem_area`.

<code>mem_area</code>	is a pointer to the memory to search. (Input)
<code>chr</code>	is the character for which to search. (Input)
<code>size</code>	is the size of the memory area to search. (Input)

`memchr()` returns a pointer to the located character, or a null pointer if the character does not occur in the object.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

Syntax

```
#include <string.h>
int memcmp(
            const void    *string1,
            const void    *string2,
            size_t        size);
```

Description

`memcmp()` compares the first `size` characters of `string1` to the first `size` characters of `string2`.

<code>string1</code>	is a pointer to a string to compare. (Input)
<code>string2</code>	is a pointer to a string to compare. (Input)
<code>size</code>	is the number of characters to compare. (Input)

`memcmp()` returns an integer that is:

- Positive, if `string1` is greater than `string2`.
- Zero, if `string1` equals `string2`.
- Negative, if `string1` is less than `string2`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.lib`

Syntax

```
#include <string.h>
void *memcpy(
    void          *dest,
    const void    *source,
    size_t        num);
```

Description

`memcpy()` copies characters from one location to another. If copying takes place between overlapping objects, the behavior is undefined. `memcpy()` returns the value of `dest`.

<code>dest</code>	is a pointer to the destination in memory. (Output)
<code>source</code>	is a pointer to the memory to copy into <code>dest</code> . (Input)
<code>num</code>	is the number of characters to copy. (Input)

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

memmove()

Move Memory

Syntax

```
#include <string.h>
void *memmove(
    void          *dest,
    const void   *source,
    size_t        num);
```

Description

memmove() copies characters from one location to another. Copying takes place as if the characters from the original location are first copied into a temporary array of `num` characters that does not overlap `dest` and `source`, and then the `num` characters from the temporary array are copied into `dest`.

dest	is a pointer to the destination in memory. (Output)
source	is a pointer to the memory to copy into dest. (Input)
num	is the number of characters to copy. (Input)

memmove() returns the value of `dest`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

clib.l

Syntax

```
#include <string.h>
void *memset(
    void          *mem,
    int           fill,
    size_t        size);
```

Description

`memset()` copies the value of `fill`, converted to an `unsigned char`, into each of the first `size` characters of `mem`. `memset()` returns the value of `mem`.

<code>mem</code>	is a pointer to the memory to fill. (Output)
<code>fill</code>	is the value with which to fill an area of memory. (Input)
<code>size</code>	is the number of characters to fill with <code>fill</code> . (Input)

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.lib`

_mkdata_module()

Create Data Memory Module

Syntax

```
#include <module.h>
char *_mkdata_module(
    const char *name,
    unsigned size,
    int attr,
    int perm);
```

Description

`_mkdata_module()` creates a data memory module. Other processes on the system can then access the data module by `modlink()`. The memory in the data module is initially cleared to zeroes.

name	is a pointer to the desired module name. (Input)
size	specifies the size of the module in bytes. (Input)



Note

The `size` value does not include the module header and CRC bytes. The specified size is the amount of memory available for actual data storage.

`attr` specifies the module's attribute and revision level. Only the lower 16 bits of `attr` are used.

`perm` specifies the module access permissions. Only the lower 16 bits of `perm` are used.

`_mkdata_module()` returns a pointer to the beginning of the module header.

If the data module cannot be created, -1 is returned and the appropriate error code is placed in the global variable `errno`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.lib`

See Also

[_os_datmod\(\)](#)

`F$DatMod`

`F_DATMOD`

OS-9 for 68K Technical Manual

OS-9 Technical Manual

Syntax

```
#include <modes.h>
int mknod(
            const char      *name,
            int              perm);
```

Description

`mknod()` creates a directory file.

`name` is a pointer to the name of the directory.
(Input)

`perm` specifies the access permissions for the
directory. (Input)

Only the lower 16 bits of `perm` are used.

`mknod()` returns zero if the directory was successfully created. If the creation failed, -1 is returned and the appropriate error code is placed in the global variable `errno`.



Note

`mknod()` does not make UNIX-style special files as there are no such files on OS-9 for 68K and OS-9. This is a historical function and is likely to be removed in a future release. Use `mkdir()` in all new code.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

sys_clib.l

See Also

[makdir\(\)](#)
[_os_mkdir\(\)](#)
[I\\$MakDir](#)
[I_MAKDIR](#)

OS-9 for 68K Technical Manual
OS-9 Technical Manual

Syntax

```
#include <stdio.h>
char *mktemp(const char *name);
```

Description

`mktemp()` ensures that the name of a temporary file is unique in the system and does not clash with any other file name.

`name` is a pointer to the template string. (Input)

A pointer to a template string is passed to `mktemp()`. The template string should look like a filename with six trailing x's. `mktemp()` replaces the x's with a letter and the current process ID. The letter is chosen so that the resulting name does not conflict with an existing file.

`mktemp()` returns a pointer to the file name or `NULL` if more than 26 `mktemp()` files exist.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

[tmpnam\(\)](#)
[tmpfile\(\)](#)

Syntax

```
#include <time.h>
time_t mkttime(struct tm *tp);
```

Description

`mkttime()` converts the broken-down time, expressed as local time, into a calendar time value with the same encoding as the values returned by `time()`. The original values of the `tm_wday` and `tm_yday` fields of the structure are ignored, and the original values of the other fields are not restricted to the indicated ranges.

`tp` is a pointer to the broken-down time structure. (Input)

On successful completion, the `tm_wday` and `tm_yday` fields of the structure are set appropriately. The other fields are set to represent the specified calendar time, but with their values forced to the indicated ranges. The final value of `tm_mday` is not set until `tm_mon` and `tm_year` are determined.

`mkttime()` returns the specified calendar time encoded as a value of type `time_t`. If the calendar time cannot be represented, `mkttime()` returns the value `(time_t)-1`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`clib.lib`

See Also

[asctime\(\)](#)
[getenv\(\)](#)
[gmtime\(\)](#)
[localtime\(\)](#)
[time\(\)](#)

Syntax

```
#include <memory.h>
#include <module.h>
mh_com *modcload(
    const char      *modname,
    int             mode,
    int             memtype);
```

Description

`modcload()` opens the file specified by the pathlist. It reads one or more memory modules from the file into memory until an error or end of file is reached. `modname` is considered a pathlist and all modules in the specified file are loaded. A link is made to the first module loaded from the file and a pointer to it is returned.

<code>modname</code>	is a pointer to the path to the module. (Input)
<code>mode</code>	is the mode which opens the file to load. (Input) If any access <code>mode</code> is acceptable, specify zero for <code>mode</code> .

`memtype` indicates the specific memory type in which to load the module. (Input)
The `memory.h` header file contains definitions of the memory types that may be specified:

Table 2-16 modload() Memory Types

Memory Type	Description
SYSRAM or MEM_SYS	System RAM memory
VIDEO1	Video memory for plane A
VIDEO2	Video memory for plane B.
MEM_ANY	No specific memory type.

If `memtype` is `MEM_ANY`, the module may be loaded in whatever memory the system allocates.

If the load is successful, `modload()` returns a pointer to the module. If the load fails, `-1` is returned and the appropriate error code is placed in the global variable `errno`.

Attributes

Operating System:	OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

Syntax

```
#include <math.h>
double modf(
    double      value,
    double      *num);
```

Description

`modf()` breaks `value` into integral and fractional parts, each of which has the same sign as the parameter. It stores the integral part as a `double` in `num`.

`value` is a real number. (Input)

`num` is a pointer to the integer portion of `value`. (Output)

`modf()` returns the signed fractional part of `value`.

Attributes

Operating System: OS-9 and OS-9 for 68K

State: User and System

Threads: Safe

Re-entrant: Yes

Standards: ANSI

Library

`clib.l`

Syntax

```
#include <module.h>
mh_com *modlink(
    const char    *modname,
    int           typelang);
```

Description

`modlink()` searches the module directory for a module with the same name as that pointed to by `modname` and links to it, provided that `typelang` matches the respective value of Type/Language in the module. If the module is found, the module's link count is incremented by one.

<code>modname</code>	is a pointer to the name of the module. (Input)
<code>typelang</code>	is the type and language value of the module. (Input)
	If any module type or language is acceptable, specify zero for <code>typelang</code> . Only the lower 16 bits of <code>typelang</code> are used.

The `module.h` header file contains a structure appropriate for accessing the elements of system-defined memory modules.

If the link is successful, `modlink()` returns a pointer to the module. If the link fails, `-1` is returned and the appropriate error code is placed in the global variable `errno`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

sys_clib.l

See Also

[_os_link\(\)](#)

[_os_load\(\)](#)

F\$Link

F_LINKM

OS-9 for 68K Technical Manual

OS-9 Technical Manual

modload()

Load and Link to Memory Module

Syntax

```
#include <module.h>
mh_com *modload(
    const char      *modname,
    int             accessmode);
```

Description

`modload()` loads all modules in the file specified by the path `modname`.

`modname` is a pointer to the name of the module.
(Input)

accessmode specifies the module's access mode.
(Input)

If any module access mode is acceptable, specify zero for `accessmode`. Only the lower 16 bits of `accessmode` are used.

A link is made to the first module loaded from the file and a pointer to it is returned.

The `module.h` header file contains a structure appropriate for accessing the elements of system-defined memory modules.

If the load is successful, `modload()` returns a pointer to the module. If the load fails, `-1` is returned and the appropriate error code is placed in the global variable `errno`.

To load a module using the shell PATH variable list, see `_os_loadp()` or `modloadp()`.

Attributes

Operating System: OS-9 and OS-9 for 68K

State: User and System

Threads: Safe

Re-entrant: Yes

Library

sys_clib.l

See Also

[_os_link\(\)](#)
[_os_load\(\)](#)
[_os_loadp\(\)](#)

F\$Load

F_LOAD

OS-9 for 68K Technical Manual

OS-9 Technical Manual

modloadp()

Load and Link to Memory Module Using PATH

Syntax

```
#include <module.h>
mh_com *modloadp(
    const char      *modname,
    int             accessmode,
    char            *namebuffer);
```

Description

modloadp() loads all modules in the file specified by the path modname. The PATH environment variable determines alternate directories to search for the named module.

PATH can contain a list of directories to search if the program is not found in the module directory or the execution directory. PATH is set to any number of directory pathlists separated by colons:

/h0/cmdu : /d0/cmdu : /n0/droid/h0/special_stuff/cmdu	
modname	is a pointer to the name of the module. (Input)
accessmode	is the access mode for the module. (Input)
	Only the lower 16 bits of accessmode are used.
namebuffer	is a pointer to an array containing the pathlist of the successfully loaded module. (Input)
	Any errors other than EOS_PNNF leave the path list used for the load attempt intact. namebuffer can be NULL if access to the path string is not required.

modloadp() returns a pointer to the module if the load is successful. If the load is unsuccessful, -1 is returned and the appropriate error code is placed in errno.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

sys_clib.l

See Also

[_os_loadp\(\)](#)
[modload\(\)](#)
[_os_load\(\)](#)

Syntax

```
#include <mqueue.h>
int mq_close(mqd_t mqdes);
```

Description

`mq_close()` removes the association between the message queue descriptor and its message queue. The results of using this descriptor after successful return from `mq_close()`—and until the return of this descriptor from a subsequent `mq_open()`—are undefined.

`mqdes` is the message queue descriptor. (Input)
If the process has successfully attached a notification request to the message queue via `mqdes`, the attachment is removed and the message queue is available for another process to attach for notification.

Upon successful completion `mq_close()` returns a value of 0. Otherwise, the function returns -1 and sets `errno` to indicate the error.

Errors

EBADF	The <code>mqdes</code> argument is not a valid message queue descriptor.
ENOSYS	<code>mq_close()</code> not supported.

Attributes

Operating System:	OS-9
State:	User
Threads:	Safe
Re-entrant:	No
Standards:	POSIX

Library

[mq.1](#)

See Also

[mq_open](#)

[mq_unlink](#)

mq_getattr

Get Message Queue Attributes

Syntax

```
#include <mqueue.h>
int mq_getattr(mqd_t mqdes, struct mq_attr *mqstat);
```

Description

`mqdes` specifies a message queue descriptor.
(Input)

`mq_getattr()` gets status information and attributes of the message queue and the open message queue description associated with the message queue descriptor. The results are returned in the `mq_attr` structure referenced by the `mqstat` argument.

`mqstat` is the argument that references the `mq_attr` structure. (Output)

Upon return, the following members shall have the values associated with the open message queue description as set when the message queue was opened and as modified by subsequent `mq_setattr()` calls: `mq_flags`.

The following attributes of the message queue shall be returned as set at message queue creation: `mq_maxmsg`, `mq_msgsize`.

Upon return, the following members within the `mq_attr` structure referenced by the `mqstat` argument shall be set according to the current state of the message queue: `mq_curmsgs`—the number of messages currently on the queue.

OS-9 specifies the following user-defined flag for use with `mq_getattr()` and `mq_setattr()`: `_MQ_O_NOTIFY_IMMEDIATE`. When this flag is specified by the `mq_setattr()` function (`mq_flags` field of `mqstat` parameter) then calls to either `_mq_notify_write()` or `mq_notify()` will notify immediately if appropriate. (Message queue nonfull for `_mq_notify_write()`, and message queue non-empty for `mq_notify()`).

This extension provides message queue notification consistency with other OS-9 services such the "send signal on data ready" concept.

Upon successful completion `mq_getattr()` returns 0. Otherwise, the function returns -1 and sets `errno` to indicate the error.

Errors

EBADF	The <code>mqdes</code> argument is not a valid message queue descriptor.
ENOSYS	<code>mq_getattr()</code> not supported.

Attributes

Operating System:	OS-9
State:	User
Threads:	Safe
Re-entrant:	No
Standards:	POSIX

Library

`mq.l`

See Also

[mq_open](#)
[mq_send](#)
[mq_setattr](#)

Notify Process That a Message is Available on a Queue

Syntax

```
#include <mqueue.h>
int mq_notify(
    mqd_t           mqdes,
    const struct sigevent *notification);
```

Description

If the argument notification is not NULL, this function registers the calling process to be notified of message arrival at an empty message queue associated with mqdes the specified message queue descriptor.

mqdes is the specified message queue descriptor. (Input)

The notification specified by the notification argument, notification, is sent to the process when the message queue transitions from empty to nonempty. At any time, only one process can be registered for notification by a message queue. If the calling process or any other process has already registered for notification of message arrival at the specified message queue, subsequent attempts to register for that message queue fail.

notification is the notification argument. (Input)

If notification is NULL and the process is currently registered for notification by the specified message queue, the existing registration is removed.

When the notification is sent to the registered process, its registration is removed. The message queue is then available for registration.

If a process has registered for notification of message arrival at a message queue, and some thread is blocked in `mq_receive()` waiting to receive a message when a message arrives at the queue, the arriving message satisfies the appropriate `mq_receive()`. The resulting behavior is as if the message queue remains empty, and no notification is sent.

Upon successful completion, `mq_notify()` returns a value of 0. Otherwise, the function returns -1 and sets `errno` to indicate the error.

Errors

EBADF	The <code>mqdes</code> argument is not a valid message queue descriptor.
EBUSY	A process is already registered for notification by the message queue.
ENOSYS	<code>mq_notify()</code> not supported.

Attributes

Operating System:	OS-9
State:	User
Threads:	Safe
Re-entrant:	No
Standards:	POSIX

Library

`mq.1`

See Also

[mq_open](#)
[mq_send](#)
[mq_setattr](#)
[_MQ_0_NOTIFY_IMMEDIATE](#)

_mq_notify_write

Notify Process That a Space in the Message Queue is Available

Syntax

```
#include <mqueue.h>
int _mq_notify_write(
    mqd_t mqdes,
    const struct sigevent *notification);
```

Description

If the argument notification is not NULL, this function registers the calling process to be notified of message removal at a full message queue associated with the specified message queue descriptor, mqdes.

mqdes is the specified message queue descriptor. (Input)

The notification specified by the notification argument, notification, is sent to the process when the message queue transitions from full to nonfull. At any time, only one process may be registered for notification by a message queue. If the calling process or any other process has already registered for notification of the specified message queue, subsequent attempts to register for that message queue will fail.

If notification is NULL and the process is currently registered for notification by the specified message queue, the existing registration is removed. The message queue is then available for registration.

notification is the notification argument. (Input)

If a process has registered for notification of a message queue and some thread is blocked in `mq_send()` waiting to send a message when the message queue transitions to nonfull, the arriving message will satisfy the appropriate `mq_send()`. The resulting behavior is as if the message queue remains full, and no notification is sent.

Errors

EBADF	The mqdes argument is not a valid message queue descriptor.
EBUSY	A process is already registered for notification by the message queue.
ENOSYS	mq_notify() not supported.

Attributes

Operating System:	OS-9
State:	User
Threads:	Safe
Re-entrant:	No
Standards:	POSIX

Library

mq.1

See Also

[mq_open](#)
[mq_send](#)
[mq_setattr](#)
[_MQ_0_NOTIFY_IMMEDIATE](#)

mq_open

Open a Message Queue

Syntax

```
#include <mqueue.h>
mqd_t mq_open(const char *name, int oflag, ...);
```

Description

`mq_open()` establishes the connection between a process and a message queue with a message queue descriptor. It creates an open message queue description that refers to the message queue, and it creates a message queue descriptor that refers to that open message queue description.

name	points to a string naming a message queue. (Input)
	<p>It is unspecified whether the name appears in the file system and is visible to other functions that take pathnames as arguments. The name argument conforms to the construction rules for a pathname. If name begins with the slash character, then processes calling <code>mq_open()</code> with the same value of name shall refer to the same message queue object, as long as that name has not been removed. 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.</p>
	<p>If the name argument is not the name of an existing message queue and creation is not requested, <code>mq_open()</code> fails and returns an error.</p>

<code>oflag</code>	requests the desired receive and/or send access to the message queue. (Input) The requested access permission to receive messages or send messages is granted if the calling process would be granted read or write access, respectively, to an equivalently protected file.
The value of <code>oflag</code> is the bitwise inclusive OR of values from the following list. Applications shall specify exactly one of the first three values (access modes) below in the value of <code>oflag</code> :	
<code>O_RDONLY</code>	Open the message queue for receiving messages. The process can use the returned message queue descriptor with <code>mq_receive()</code> , but not <code>mq_send()</code> . A message queue may be open multiple times in the same or different processes for receiving messages.
<code>O_WRONLY</code>	Open the queue for sending messages. The process can use the returned message queue descriptor with <code>mq_send()</code> but not <code>mq_receive()</code> . A message queue may be open multiple times in the same or different processes for sending messages.
<code>O_RDWR</code>	Open the queue for both receiving and sending messages. The process can use any of the functions allowed for <code>O_RDONLY</code> and <code>O_WRONLY</code> . A message queue may be open multiple times in the same or different processes for sending messages.

Any combination of the remaining flags may be specified in the value of oflag:

O_CREAT

This option is used to create a message queue, and it requires two additional arguments: mode, which is of type mode_t, and attr, which is a pointer to a mq_attr structure. If the pathname, name, has already been used to create a message queue that still exists, then this flag has no effect, except as noted under O_EXCL. Otherwise, a message queue is created without any messages in it. The user ID of the message queue is set to the effective user ID of the process, and the group ID of the message queue is set to the effective group ID of the process. The "file permission bits" is set to the value of mode. When bits in mode other than file permission bits are set, the effect is implementation defined. If attr is NULL, the message queue is created with implementation-defined default message queue attributes. If attr is non-NUL and the calling process has the appropriate privilege on name, the message queue mq_maxmsg and mq_msgsize attributes are set to the values of the corresponding members in the mq_attr structure referred to by attr. If attr is non-NUL, but the calling process does not have the appropriate privilege on name, mq_open() fails and returns an error without creating the message queue.

O_EXCL	If O_EXCL and O_CREAT are set, mq_open() fails if the message queue name exists. The check for the existence of the message queue and the creation of the message queue if it does not exist is atomic with respect to other processes executing mq_open() naming the same name with O_EXCL and O_CREAT set. If O_EXCL is set and O_CREAT is not set, the result is undefined.
O_NONBLOCK	The setting of this flag is associated with the open message queue description and determines whether a mq_send() or mq_receive() shall wait for resources or messages that are not currently available, or fail with errno set to [EAGAIN]. See mq_send() and mq_receive() for details.

mq_open() does not add or remove messages from the queue.

Upon successful completion, mq_open() returns a message queue descriptor. Otherwise, the function returns (mqd_t) -1 and sets errno to indicate the error.

Errors

EACCES	The message queue exists and the permissions specified by oflag are denied, or the message queue does not exist and permission to create the message queue is denied.
EEXIST	O_CREAT and O_EXCL are set and the named message queue already exists.
EINTR	The mq_open() operation was interrupted by a signal.

EINVAL	The <code>mq_open()</code> operation is not supported for the given name. The implementation shall document under what circumstances this error may be returned.
EMFILE	<code>O_CREAT</code> was specified in <code>oflag</code> , the value of <code>attr</code> is not <code>NULL</code> , and either <code>mq_maxmsg</code> or <code>mq_msgsiz</code> was less than or equal to zero.
ENAMETOOLONG	Too many message queue descriptors or file descriptors are currently in use by this process.
ENFILE	The length of the name string exceeds <code>{PATH_MAX}</code> , or a pathname component is longer than <code>{NAME_MAX}</code> while <code>{_POSIX_NO_TRUNC}</code> is in effect.
ENOENT	Too many message queues are currently open in the system.
ENOSPC	<code>O_CREAT</code> is not set and the named message queue does not exist.
ENOSYS	There is insufficient space for the creation of the new message queue.
	<code>mq_open()</code> not supported.

Attributes

Operating System:	OS-9
State:	User
Threads:	Safe
Re-entrant:	No
Standards:	POSIX

Library

`mq.l`

See Also

[mq_close](#)
[mq_receive](#)
[mq_send](#)
[mq_setattr](#)
[mq_getattr](#)
[mq_unlink](#)

mq_receive

Receive a Message From a Message Queue

Syntax

```
#include <mqueue.h>
size_t mq_receive(
    mqd_t          mqdes,
    char           *msg_ptr,
    size_t          msg_len,
    unsigned int   *msg_prio);
```

Description

`mq_receive()` is used to receive the oldest of the highest priority message(s) from the message queue specified by `mqdes`. If the size of the buffer in bytes, specified by `msg_len`, is less than the `mq_msgsize` attribute of the message queue, the function fails and returns an error. Otherwise, the selected message is removed from the queue and copied to the buffer pointed to by `msg_ptr`.

<code>mqdes</code>	specifies the message queue. (Input)
<code>msg_ptr</code>	points to the buffer. (Output)
<code>msg_len</code>	specifies the size of the buffer in bytes. (Input)
<code>msg_prio</code>	If <code>msg_prio</code> is not NULL, the priority of the selected message is stored in the location referenced by <code>msg_prio</code> . (Input)

If the specified message queue is empty and `O_NONBLOCK` is not set in the message queue description associated with `mqdes`, `mq_receive()` blocks until a message is enqueued on the message queue or until `mq_receive()` is interrupted by a signal. If more than one thread is waiting to receive a message when a message arrives at an empty queue and the Process Scheduling option is supported, then the thread of highest priority that has been waiting the longest is selected to receive the message. Otherwise, it is unspecified which waiting thread receives the message. If the specified message queue is empty and `O_NONBLOCK` is set in the message queue description associated with `mqdes`, no message is removed from the queue, and `mq_receive()` returns an error.

Upon successful completion, `mq_receive()` returns the length of the selected message in bytes and the message is removed from the queue. Otherwise, no message is removed from the queue, the function returns a -1, and sets `errno` to indicate the error.

Errors

EAGAIN	<code>O_NONBLOCK</code> was set in the message description associated with <code>mqdes</code> , and the specified message queue is empty.
EBADF	The <code>mqdes</code> argument is not a valid message queue descriptor open for reading.
EMSGSIZE	The specified message buffer size, <code>msg_len</code> , is less than the message size attribute of the message queue.
EINTR	The <code>mq_receive()</code> operation was interrupted by a signal.
ENOSYS	The <code>mq_receive()</code> function is not supported by this implementation.
EBADMSG	A data corruption problem with the message has been detected.

Attributes

Operating System:	OS-9
State:	User
Threads:	Safe
Re-entrant:	No
Standards:	POSIX

Library

`mq.l`

See Also

[mq_send](#)

Syntax

```
#include <mqueue.h>
int mq_send(
    mqd_t          mqdes,
    const char     *msg_ptr,
    size_t          msg_len,
    unsigned int    msg_prio);
```

Description

`mq_send()` adds the message pointed to by the argument `msg_ptr` to the message queue specified by `mqdes`. `msg_len` specifies the length of the message in bytes pointed to by `msg_ptr`. The value of `msg_len` is less than or equal to the `mq_msgsize` attribute of the message queue, otherwise `mq_send()` fails.

If the specified message queue is not full, `mq_send()` behaves as if the message is inserted into the message queue at the position indicated by the `msg_prio` argument. A message with a larger numeric value of `msg_prio` is inserted before messages with lower values of `msg_prio`. A message is inserted after other messages in the queue, if any, with equal `msg_prio`. The value of `msg_prio` is less than `{MQ_PRIO_MAX}`.

<code>mqdes</code>	specifies the message queue. (Output)
<code>msg_ptr</code>	points to the buffer. (Input)
<code>msg_len</code>	specifies the size of the buffer in bytes. (Input)
<code>msg_prio</code>	If <code>msg_prio</code> is not NULL, the priority of the selected message is stored in the location referenced by <code>msg_prio</code> . (Input)

If the specified message queue is full and `O_NONBLOCK` is not set in the message queue description associated with `mqdes`, `mq_send()` blocks until space becomes available to enqueue the message, or until `mq_send()` is interrupted by a signal. If more than one thread is waiting to send when space becomes available in the message queue and the Process Scheduling option is supported, then the thread of the highest priority that has been waiting the longest is unblocked to send its message. Otherwise, it is unspecified which waiting thread is unblocked. If the specified message queue is full and `O_NONBLOCK` is set in the message queue description associated with `mqdes`, the message is not queued and `mq_send()` returns an error.

Upon successful completion, `mq_send()` returns a value of 0. Otherwise, no message is enqueued, the function returns -1, and `errno` is set to indicate the error.

Errors

EAGAIN	The <code>O_NONBLOCK</code> flag is set in the message queue description associated with <code>mqdes</code> , and the specified message queue is full.
EBADF	The <code>mqdes</code> argument is not a valid message queue descriptor open for writing.
EINTR	A signal interrupted the call to <code>mq_send()</code> .
EINVAL	The value of <code>msg_prio</code> was outside the valid range.
EMSGSIZE	The specified message length, <code>msg_len</code> , exceeds the message size attribute of the message queue.
ENOSYS	<code>mq_send()</code> not supported.

Attributes

Operating System:	OS-9
State:	User
Threads:	Safe
Re-entrant:	No
Standards:	POSIX

Library

`mq.h`

See Also

`mq_receive`

`mq_setattr`

mq_setattr

Set Message Queue Attributes

Syntax

```
#include <mqueue.h>
int mq_setattr(
    mqd_t          mqdes,
    const struct   mq_attr *mqstat,
    struct mq_attr *omqstat);
```

Description

`mq_setattr()` is used to set attributes associated with the open message queue description referenced by the message queue descriptor specified by `mqdes`.

The message queue attributes corresponding to the following members defined in the `mq_attr` structure are set to the specified values upon successful completion of `mq_setattr()`:

<code>mq_flags</code>	The value of this member is the bitwise logical OR of zero or more of <code>O_NONBLOCK</code> and any implementation-defined flags.
<code>mqdes</code>	specifies the message queue. (Input)
<code>omqstat</code>	If <code>omqstat</code> is non-NULL, the function <code>mq_setattr()</code> stores, in the location referenced by <code>omqstat</code> , the previous message queue attributes and the current queue status. (Output)
	These values are the same as would be returned by a call to <code>mq_getattr()</code> at that point.

The values of the `mq_maxmsg`, `mq_msgsize`, and `mq_curmsgs` members of the `mq_attr` structure are ignored by `mq_setattr()`.

OS-9 specifies the following user-defined flag for use with `mq_getattr()` and `mq_setattr()`: `_MQ_O_NOTIFY_IMMEDIATE`. When this flag is specified by the `mq_setattr()` function (`mq_flags` field of `mqstat` (input) parameter) then calls to either `_mq_notify_write()` or `mq_notify()` will notify immediately if appropriate.

(Message queue nonfull for `_mq_notify_write()`, and message queue non-empty for `mq_notify()`) This extension provides message queue notification consistency with other OS-9 services such the "send signal on data ready" concept.

Upon successful completion, `mq_setattr()` returns a value of 0 and the attributes of the message queue are changed as specified. Otherwise, the message queue attributes are unchanged, and the function returns a value of -1 and sets `errno` to indicate the error.

Errors

<code>EBADF</code>	The <code>mqdes</code> argument is not a valid message queue descriptor.
<code>ENOSYS</code>	<code>mq_setattr()</code> not supported.

Attributes

Operating System:	OS-9
State:	User
Threads:	Safe
Re-entrant:	No
Standards:	POSIX

Library

`mq.l`

See Also

[mq_open](#)
[mq_send](#)

mq_unlink

Remove a Message Queue

Syntax

```
#include <mqueue.h>
int mq_unlink(const char *name);
```

Description

`mq_unlink()` removes the message queue named by the pathname `name`.

`name` is the pathname. (Input)

After a successful call to `mq_unlink()` with `name`, a call to `mq_open()` with `name` fails if the flag `O_CREAT` is not set in `flags`. If one or more processes have the message queue open when `mq_unlink()` is called, destruction of the message queue is postponed until all references to the message queue have been closed. Calls to `mq_open()` to re-create the message queue may fail until the message queue is actually removed. However, the `mq_unlink()` call need not block until all references have been closed; it may return immediately.

Upon successful completion, `mq_unlink()` returns a value of 0. Otherwise, the named message queue is not changed by this function call, and the function returns a value of -1 and sets `errno` to indicate the error.

Errors

EACCES	Permission is denied to unlink the named message queue.
ENAMETOOLONG	The length of the name string exceeds {NAME_MAX} while {_POSIX_NO_TRUNC} is in effect.
ENOENT	The named message queue does not exist.
ENOSYS	<code>mq_unlink()</code> not supported.

Attributes

Operating System:	OS-9
State:	User
Threads:	Safe
Re-entrant:	No
Standards:	POSIX

Library

`mq.l`

See Also

[mq_close](#)

[mq_open](#)

munlink()

Unlink from Module

Syntax

```
#include <module.h>
int munlink(mh_com *module);
```

Description

`munlink()` informs the system that the process no longer requires the specified module. The module's link count is decremented, and the module is removed from memory if the link count reaches zero.

module is a pointer to the module. (Input)

It must have been a pointer returned by modcload(), modload(), modloadp(), modlink(), _os_link(), _os_load(), or _os_loadp(). On error, -1 is returned and the appropriate error code is placed in the global variable errno.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

sys lib.l

See Also

[modcload\(\)](#)
[modlink\(\)](#)
[modload\(\)](#)
[modloadp\(\)](#)
[_os_link\(\)](#)
[_os_load\(\)](#)
[_os_loadp\(\)](#)
[_os_unlink\(\)](#)
[F\\$Unlink](#)
[F_UNLINK](#)

OS-9 for 68K Technical Manual
OS-9 Technical Manual

Syntax

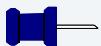
```
#include <module.h>
int munload(
            const char *name,
            int         typelang);
```

Description

`munload()` informs the system that the process no longer requires the module name which has a type/language of `typelang`. The module's link count is decremented. The module is removed from memory when the link count reaches zero.

<code>name</code>	is a pointer to the name of the module. (Input)
<code>typelang</code>	specifies the type and language of the module. (Input) If <code>typelang</code> is zero, any module may be unloaded. Only the lower 16 bits of <code>typelang</code> are used.

On error, `-1` is returned and the appropriate error code is placed in the global variable `errno`.



Note

`munload()` differs from `munlink()` in that it unlinks by module name rather than module pointer. Attempting to unlink a module by performing a link to a module by name to determine its address, then unlinking it twice does not work because the first unlink removes the module from the process' address space.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

sys_clib.l

See Also

[munlink\(\)](#)
[_os_link\(\)](#)
[_os_load\(\)](#)
[_os_unload\(\)](#)
F\$UnLoad
F_UNLOAD

OS-9 for 68K Technical Manual

OS-9 Technical Manual

Expand Value to Integral Constant Expression

Syntax

```
#include <stddef.h>
int offsetof(type, mbr_desig);
```

Description

`offsetof()` expands to an integral constant expression. The integral constant expression has a type of `size_t`, the value of which is the offset in bytes to the structure member from the beginning of the structure.

`type` specifies the type of the structure. (Input)

`mbr_desig` specifies the member name of the structure for which you want to compute the offset. If the member is a bit field, the behavior is undefined.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

Syntax

```
#include <modes.h>
int open(
    const char      *name,
    int              mode);
```

Description

`open()` opens an existing file.

`name` is a pointer to the name of the file. (Input)

`mode` specifies the access mode for the file.
(Input)

The values for `mode` are defined in the `modes.h` header file. Only the lower 16 bits of `mode` are used.

`open()` returns a path number identifying the file when I/O is performed. If the open fails, -1 is returned and the appropriate error code is placed in the global variable `errno`.



Note

The thread enabled version of this function contains a cancel point. For more information see ***Using OS-9 Threads***.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

sys_clib.l

See Also

[close\(\)](#)
[creat\(\)](#)
[_os_create\(\)](#)
[_os_open\(\)](#)
[I\\$Open](#)
[_os_open\(\)](#)

OS-9 for 6K Technical Manual

OS-9 Technical Manual

Syntax

```
#include <dir.h>
DIR *opendir(const char *filename);
```

Description

`opendir()` opens the specified directory and associates a directory stream with it. `opendir()` returns a pointer identifying the directory stream in subsequent operations. The pointer `NULL` is returned if `filename` cannot be accessed or if it cannot allocate enough memory to hold the entire `DIR` structure.

<code>filename</code>	is a pointer to the name of the directory. (Input)
-----------------------	---

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

`sys_clib.l`

See Also

[closedir\(\)](#)
[readdir\(\)](#)
[rewinddir\(\)](#)
[seekdir\(\)](#)
[telldir\(\)](#)

_os_acqlk()

Acquire Ownership of Resource Lock

Syntax

```
#include <lock.h>
#include <types.h>
error_code _os_acqlk(
    lk_desc      *lock,
    signal_code *signal);
```

Description

`_os_acqlk()` acquires ownership of a resource lock (that is, it attempts to gain exclusive access to a resource).

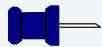
`lock` is a pointer to the `lk_desc` structure of the lock that you are trying to acquire. (Input)

`signal` is a pointer to the location where `_os_acqlk()` stores the signal that prematurely terminated the acquisition of the lock, if applicable. (Output)

If the lock is not owned by another process, the calling process is granted ownership and the call returns without error.

If the lock is already owned, the calling process is suspended and inserted into a waiting queue for the resource based on relative scheduling priority.

When ownership of the lock is released, the next process in the queue is granted ownership and is activated. The activated process returns from the `_os_acqlk()` call without error. If a process received a signal while waiting on a lock, the process is activated without gaining ownership of the lock. The process returns from the `_os_acqlk()` call with an `EOS_SIGNAL` error code and the signal code returned in the `signal` pointer.



Note

If a waiting process receives an `S_WAKEUP` signal, the signal code does not register and is zero.

Attributes

Operating System:	OS-9
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

`lock.l`

See Also

<code>F_ACQLK</code>	<i>OS-9 Technical Manual</i>
<code>F_CAQLK</code>	<i>OS-9 Technical Manual</i>
<code>F_CRLK</code>	<i>OS-9 Technical Manual</i>
<code>F_DDLK</code>	<i>OS-9 Technical Manual</i>
<code>F_DELLK</code>	<i>OS-9 Technical Manual</i>
<code>F_RELlk</code>	<i>OS-9 Technical Manual</i>
<code>F_WAITLK</code>	<i>OS-9 Technical Manual</i>

_os9_alarm_atdate()

Send Signal at Gregorian Date/Time

Syntax

```
#include <alarm.h>
#include <types.h>
error_code _os9_alarm_atdate(
                           alarm_id      *alarm_id,
                           signal_code   signal,
                           u_int32       time,
                           u_int32       date);
```

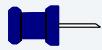
OS-9 Attributes

Compatibility	State	Threads Compatibility	Reentrant
OS-9 for 68K	User	Safe	Yes

Description

`_os9_alarm_atdate()` sends a signal to the caller at a specific date and time. The alarm signal is sent anytime the system date/time becomes greater than or equal to the alarm time.

alarm_id	is a pointer to the location where <code>_os9_alarm_atdate()</code> stores the alarm ID. (Output)
signal	specifies the signal code. (Input)
date	is in the format yyyyymmdd. (Input)
time	is in the format 00hhmmss. (Input)



Note

`_os9_alarm_atdate()` only allows you to specify the time to the nearest second. However, it does adjust if the system's date and time have changed (using `_os9_setime()`).

Attributes

Operating System:	OS-9 for 68K
State:	User
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`F$Alarm`

OS-9 for 68K Technical Manual

_os_alarm_atime()

Send Signal at Specified Time

Syntax

```
#include <alarm.h>
#include <types.h>
error_code _os_alarm_atime(
    alarm_id          *alarm_id,
    signal_code       signal,
    u_int32           time);
```

Description

`_os_alarm_atime()` sends a signal when the system time reaches or passes the specified time. The time value is considered to be an absolute value in seconds since 1 January 1970 Greenwich Mean Time.

alarm_id	is a pointer to the location where <code>_os_alarm_atime()</code> stores the alarm ID. (Output)
signal	is the signal code of the signal to send. (Input)
time	specifies the time. (Input)

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

[_os_alarm_set\(\)](#)
[_os9_alarm_atdate\(\)](#)
[_os9_alarm_atjul\(\)](#)

F\$Alarm
F_ALARM

OS-9 for 68K Technical Manual
OS-9 Technical Manual

_os9_alarm_atjul()

Send Signal at Julian Date/Time

Syntax

```
#include <alarm.h>
#include <types.h>
error_code _os9_alarm_atjul(
    alarm_id          *alarm_id,
    signal_code       signal,
    u_int32           time,
    u_int32           date);
```

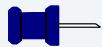
OS-9 Attributes

Compatibility	State	Threads Compatibility	Re-entrant
OS-9 for 68K	User	Safe	Yes

Description

`_os9_alarm_atjul()` sends a signal to the caller at a specific Julian date and time. The alarm signal is sent anytime the system date/time becomes greater than or equal to the alarm time.

alarm_id	is a pointer to the location where <code>_os9_alarm_atjul()</code> stores the alarm ID. (Output)
signal	specifies the signal code. (Input)
time	is the seconds after midnight. (Input)
date	is the Julian day number. (Input)



Note

`_os9_alarm_atjul()` only allows you to specify the time to the nearest second. However, it does adjust if the system's date and time have changed (using `_os9_setime()`).

Attributes

Operating System:	OS-9 for 68K
State:	User
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`F$Alarm`

OS-9 for 68K Technical Manual

_os_alarm_cycle()

Send Signal at Specified Time Intervals

Syntax

```
#include <alarm.h>
#include <types.h>
error_code _os_alarm_cycle(
    alarm_id      *alarm_id,
    signal_code   signal,
    u_int32       time);
```

Description

`_os_alarm_cycle()` sends a signal after the specified time interval has elapsed and then resets the alarm. This provides a recurring periodic signal. The time interval may be specified in system clock ticks; or, if the high order bit is set, the low 31 bits are considered a time in 256ths of a second. The minimum time interval allowed is 2 system clock ticks.

alarm_id	is a pointer to the location where <code>_os_alarm_cycle()</code> stores the alarm ID. (Output)
signal	is the signal code of the signal to send. (Input)
time	specifies the time interval between signals. (Input)

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

[_os_alarm_set\(\)](#)

F\$Alarm

F_ALARM, A_CYCLE

OS-9 for 68K Technical Manual

OS-9 Technical Manual

_os_alarm_delete()

Remove Pending Alarm Request

Syntax

```
#include <alarm.h>
#include <types.h>

error_code _os_alarm_delete(
                           alarm_id     alrm_id);
```

Description

`_os_alarm_delete()` removes a cyclic alarm or any alarm that has not expired.

`alrm_id` specifies the alarm to remove. (Input)
If `alrm_id` is zero, all pending alarm requests for the owner are removed.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`F$Alarm` ***OS-9 for 68K Technical Manual***
`F_ALARM, F_DELETE` ***OS-9 Technical Manual***

_os_alarm_reset()

Reset Existing Alarm Request

Syntax

```
#include <alarm.h>
#include <types.h>

error_code _os_alarm_reset(
                           alarm_id      alarm_id,
                           signal_code   signal,
                           u_int32       time);
```

Description

`_os_alarm_reset()` sets an existing alarm to send after the specified time interval has elapsed.

alarm_id	is the alarm ID to reset. (Input)
signal	is the signal code of the signal to send. (Input)
time	may be specified in system clock ticks; or, if the high order bit is set, the low 31 bits are considered a time in 256ths of a second. (Input)
	Usually, the minimum time interval allowed is 2 clock ticks.

Attributes

Operating System:	OS-9
State:	User
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

[_os_alarm_set\(\)](#)

F\$Alarm

F_ALARM, A_RESET

OS-9 for 68K Technical Manual
OS-9 Technical Manual

_os_alarm_set()

Send Signal After Specified Time Interval

Syntax

```
#include <alarm.h>
#include <types.h>
error_code _os_alarm_set(
    alarm_id          *alrm_id,
    signal_code       signal,
    u_int32           time);
```

Description

`_os_alarm_set()` sends one signal after the specified time interval has elapsed.

<code>alrm_id</code>	is a pointer to the location where <code>_os_alarm_set()</code> stores the alarm ID. (Output)
<code>signal</code>	is the signal code of the signal to send. (Input)
<code>time</code>	may be specified in system clock ticks; or, if the high order bit is set, the low 31 bits are considered a time in 256ths of a second. (Input)
	Usually, the minimum time interval allowed is 2 system clock ticks.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

F\$Alarm

F_ALARM, A_SET

OS-9 for 68K Technical Manual

OS-9 Technical Manual

Syntax

```
#include <io.h>
#include <types.h>
error_code _os_alias(
    const char    *alias_name,
    const char    *real_name);
```

Description

`_os_alias()` creates an alternate name for a device pathlist. Processes can then reference a specific device pathlist with a shorter or more convenient name.

alias_name	is a pointer to the alternate name. (Input)
real_name	is a pointer to the actual device name; it must exist. (Input)
	OS-9 does not validate its existence.



WARNING

Do not use a real device name as `alias_name`.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

I_ALIAS

OS-9 Technical Manual

Syntax

```
#include <memory.h>
#include <types.h>
error_code _os9_allbit(
    u_int32      bit_number,
    u_int32      count,
    void        *address);
```

Description

`_os9_allbit()` sets bits in a buffer that represents a bit map. Bit numbers range from 0 to $n - 1$, where n is the number of bits in the bit map.

bit_number	specifies the bit number of the first bit to set. (Input)
	Only the lower 16 bits of <code>bit_number</code> are used.
count	specifies the number of bits to set. (Input)
	Only the lower 16 bits of <code>count</code> are used.
address	is a pointer to the base address of a bit map. (Input/Output)

Attributes

Operating System:	OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

[_os9_schbit\(\)](#)
[_os9_delbit\(\)](#)
[F\\$AllBit](#)

OS-9 for 68K Technical Manual

_os9_allpd()

Allocate Fixed-Length Block of Memory

Syntax

```
#include <process.h>
#include <types.h>
error_code _os9_allpd(
    void          *table,
    u_int16       *number,
    void          **ptr);
```

Description

`_os9_allpd()` allocates fixed-length blocks of system memory. It allocates and initializes (to zeros) a block of storage and returns its address.

table	is a pointer to the process/path table. (Input)
number	is a pointer to the location where <code>_os9_allpd()</code> stores the process/path number. (Output)
ptr	is a pointer to the location where <code>_os9_allpd()</code> stores the pointer to the process/path descriptor. (Output)

Attributes

Operating System:	OS-9 for 68K
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

[_os9_findpd\(\)](#)
[_os9_retpd\(\)](#)
F\$AllPD

OS-9 for 68K Technical Manual

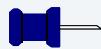
Syntax

```
#include <process.h>
#include <types.h>
error_code _os_alltsk(pr_desc *proc_desc);
```

Description

`_os_alltsk()` initializes the protection hardware for a newly activated process.

`proc_desc` is a pointer to the process descriptor.
(Input)



Note

`_os_alltsk()` only performs a useful function on MMU/SSM systems. On non-MMU/SSM systems, `_os_alltsk()` simply returns without initializing the protection hardware.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`F$AllTsk` ***OS-9 for 68K Technical Manual***
`F_ALLTSK` ***OS-9 Technical Manual***

_os_alocproc()

Allocate Process Descriptor

Syntax

```
#include <process.h>
#include <types.h>

error_code _os_alocproc(
    process_id      *proc_id,
    pr_desc         **proc_desc);
```

Description

`_os_alocproc()` allocates and initializes a process descriptor. The descriptor's address is stored in the process descriptor table.

<code>proc_id</code>	is a pointer to the location where <code>_os_alocproc()</code> stores the process ID. (Output)
<code>proc_desc</code>	is a pointer to the location where <code>_os_alocproc()</code> stores the pointer to the process descriptor. (Output)

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`F$AllPrc`

OS-9 for 68K Technical Manual

_os_altmdir()

Set Alternate Working Module Directory

Syntax

```
#include <moddir.h>
#include <types.h>
error_code _os_altmdir(const char *name);
```

Description

`_os_altmdir()` establishes an alternate working module directory for a process.

name	specifies the name of the module directory. (Input)
------	---

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`F_ALTMDIR` ***OS-9 Technical Manual***

_os_aproc()

Insert Process in Active Process Queue

Syntax

```
#include <process.h>
#include <types.h>
error_code _os_aproc(process_id proc_id);
```

Description

`_os_aproc()` inserts the process specified by `proc_id` (input) into the active process queue so that it may be scheduled for execution.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

[_os_nproc\(\)](#)

`F$APROC`

`F_ALLPRC`

OS-9 for 68K Technical Manual

OS-9 Technical Manual

_os9_aproc()

Insert Process in Active Process Queue

Syntax

```
#include <process.h>
#include <types.h>

error_code _os9_aproc(pr_desc *proc_desc);
```

Description

`_os9_aproc()` inserts a process into the active process queue so that it may be scheduled for execution.

<code>proc_desc</code>	is a pointer to the process descriptor. (Input)
------------------------	--

Attributes

Operating System:	OS-9 for 68K
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_nproc()`
`F$AProc`

OS-9 for 68K Technical Manual

_os_attach()

Attach New Device to System

Syntax

```
#include <io.h>
#include <modes.h>
#include <types.h>
error_code _os_attach(
                      const char      *name,
                      u_int32         mode,
                      dev_list        **dev_tbl);
```

Description

`_os_attach()` causes a new I/O device to become known to the system or verifies that the device is already attached.

name	is a pointer to the I/O device's name. (Input)
mode	is the access mode. It can be <code>FAM_READ</code> , <code>FAM_WRITE</code> , <code>S_IREAD</code> , or <code>S_IWRITE</code> . <code>mode</code> may be used to verify that subsequent read and/or write operations are permitted. (Input) Only the lower 16 bits of <code>mode</code> are used.



Note

All available access modes are defined in the following location:

`MWOS\<OS>\SRC\DEFS\modes.h`

For More Information

For a list of all available access modes, refer to the section, “Access Modes and Permissions,” in Chapter 3 of the ***OS-9 Technical Manual***.

`dev_tbl`

is a pointer to the location where `_os_attach()` stores the pointer to the device list entry. (Output)

If the descriptor name is found, `_os_attach()` links to the I/O device’s file manager and device driver.

If the device was already attached, `_os_attach()` returns a pointer to the pointer to the device table entry containing the address of the I/O device’s file manager and device driver in `dev_tbl`.

If the device is not already attached, `_os_attach()` creates a new device table entry and initializes the device. It returns a pointer to the pointer to the entry in `dev_tbl`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

[`_os_detach\(\)`](#)

`I$Attach`

OS-9 for 68K Technical Manual

`I_ATTACH`

OS-9 Technical Manual

Syntax

For OS-9 for 68K:

```
#include <cache.h>* for OS-9 for 68K systems  
#include <regs.h>  
#include <types.h>  
  
error_code _os_cache(u_int32 control);
```

For OS-9:

```
#include <regs.h>  
#include <types.h>  
  
error_code _os_cache(  
    u_int32      control,  
    void        *addr,  
    u_int32      size);
```

Description

`_os_cache()` performs operations on the system instruction and/or data caches, if there are any.

control	If <code>control</code> is zero, the system instruction and data caches are flushed. (Input)
	Only non-super-group, user-state processes may perform this operation.

Any program that builds or changes executable code in memory must flush the instruction cache with `_os_cache()` before executing the new code. This is necessary because the hardware instruction cache is not updated by data (write) accesses and may contain the unchanged instruction(s). For example, if a subroutine builds a system call on its stack, you must execute the `_os_cache()` call to flush the instruction cache before executing the temporary instructions.



For More Information

Refer to your operating system technical manual for a description of the possible values for `control`.

`addr` specifies the target address for the flush operation. (Input)

`size` indicates the size of the target memory area to be flushed. (Input)



Note

The cache should be flushed before trying to disable the data cache.

Only super-group processes may perform precise operations of `F$CCT1`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

[_os_scache\(\)](#)

F\$CCT1

F_CCTL

OS-9 for 68K Technical Manual
OS-9 Technical Manual

Syntax

```
#include <clock.h>
#include <types.h>
error_code _os_caqlk(lk_desc *lock);
```

Description

`_os_caqlk()` is used to conditionally acquire ownership of a resource lock.

`lock` is a pointer to the lock. (Input)

If another process does not own the lock, the calling process is granted ownership and the call returns without error.

If the lock is already owned, the calling process is not suspended. Instead, it returns from the `_os_caqlk()` call with an `EOS_NOLOCK` error and is not granted ownership of the resource lock.

Attributes

Operating System:	OS-9
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

`lock.l`

See Also

<code>F_ACQLK</code>	<i>OS-9 Technical Manual</i>
<code>F_CAQLK</code>	<i>OS-9 Technical Manual</i>
<code>F_CRLK</code>	<i>OS-9 Technical Manual</i>
<code>F_DDLK</code>	<i>OS-9 Technical Manual</i>
<code>F_DELLK</code>	<i>OS-9 Technical Manual</i>
<code>F_RELlk</code>	<i>OS-9 Technical Manual</i>
<code>F_WAITLK</code>	<i>OS-9 Technical Manual</i>

_os_chain()

Execute New Primary Module

Syntax

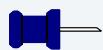
```
#include <process.h>
#include <module.h>
#include <types.h>
error_code _os_chain(
    u_int32      priority,
    u_int32      path_cnt,
    const char   *mod_name,
    const void   *params,
    u_int32      param_size,
    u_int32      mem_size,
    u_int32      type_lang);
```

Description

`_os_chain()` executes a new program without the overhead of creating a new process. It is functionally similar to a Fork command followed by an Exit. `_os_chain()` effectively resets the calling process' program and data memory areas and begins executing a new primary module. Open paths are not closed or otherwise affected.

priority	is the priority of the process. (Input) Only the lower 16 bits of <code>priority</code> are used.
path_cnt	specifies the number of I/O paths to copy. (Input) Only the lower 16 bits of <code>path_cnt</code> are used.
mod_name	is a pointer to the new program to execute. (Input)
params	is a pointer to the parameter block. (Input)

<code>param_size</code>	specifies the size of the parameter block. (Input)
<code>mem_size</code>	specifies the additional memory size in bytes. (Input)
<code>type_lang</code>	specifies the desired module type/language. (Input)
	<code>type_lang</code> must be either program/object (MT_PROGRAM, ML_OBJECT) or zero (for any). Only the lower 16 bits of <code>type_lang</code> are used.



Note

On OS-9 for 68K, `_os_chain()` never returns to the calling process.

On OS-9, an error is returned only if there is not enough memory to hold the parameters. If an error occurs during the `chain`, it is returned as an exit status to the parent of the process performing the `chain`.

On OS-9, a threaded process may only chain from the `main()` thread and may not chain if multiple threads are active in the process.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

[_os_fork\(\)](#)

[_os_load\(\)](#)

F\$Chain

F_CHAIN

OS-9 for 68K Technical Manual

OS-9 Technical Manual

_os_chainm()

Execute New Primary Module Given Pointer to Module

Syntax

```
#include <process.h>
#include <types.h>
error_code _os_chainm(
    u_int32      priority,
    u_int32      path_cnt,
    mh_com       *mod_head,
    void         *params,
    u_int32      param_size,
    u_int32      mem_size);
```

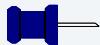
Description

`_os_chainm()` executes a new program without the overhead of creating a new process. It is functionally similar to a fork command followed by an exit. `_os_chainm()` effectively resets the calling process' program and data memory areas and begins executing a new primary module. Open paths are not closed or otherwise affected.

`_os_chainm()` is similar to `_os_chain()`. However, it is passed a pointer to the module in memory to chain to instead of the module name.

<code>priority</code>	is the priority of the process. (Input) Only the lower 16 bits of <code>priority</code> are used.
<code>path_cnt</code>	is the number of I/O paths to copy. (Input) Only the lower 16 bits of <code>path_cnt</code> are used.
<code>mod_head</code>	is a pointer to the module header. (Input)
<code>params</code>	is a pointer to the parameter block. (Input)

param_size	specifies the size of the parameter block. (Input)
mem_size	specifies the additional memory size in bytes. (Input)



Note

An error is returned only if there is not enough memory to hold the parameters. If an error occurs during the `_os_chainm()`, it is returned as an exit status to the parent of the process performing the `_os_chainm()`.

A threaded process may only chain from the `main()` thread and may not chain if multiple threads are active in the process.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

F_CHAIN	<i>OS-9 Technical Manual</i>
F_CHAINM	<i>OS-9 Technical Manual</i>
F_EXIT	<i>OS-9 Technical Manual</i>
F_FORK	<i>OS-9 Technical Manual</i>
F_LOAD	<i>OS-9 Technical Manual</i>

Syntax

```
#include <modes.h>
#include <types.h>
error_code _os_chdir(
    const char    *name,
    u_int32       mode);
```

Description

`_os_chdir()` changes a process' working directory to the directory file specified by the pathlist. You may change the execution or the data directory (or both), depending on the access mode you specify. The specified file must be a directory file, and the caller must have access permission for the specified mode.

name	is a pointer to the pathlist. (Input)
mode	specifies the access mode. (Input) Only the lower 16 bits of mode are used.

Note

All available access modes are defined in the following location:
MWOS\<OS>\SRC\DEFS\modes.h



For More Information

For a list of all available access modes, refer to the section, “Access Modes and Permissions,” in Chapter 3 of the **OS-9 Technical Manual**.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

`I$ChgDir` ***OS-9 for 68K Technical Manual***
`I_CHDIR` ***OS-9 Technical Manual***

_os_chkmem()

Check Memory Block's Accessibility

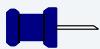
Syntax

```
#include <process.h>
#include <types.h>
error_code _os_chkmem(
    u_int32      size,
    u_int32      mode,
    void        *mem_ptr,
    pr_desc     *proc_desc);
```

Description

`_os_chkmem()` determines if a process has access to a specified memory block.

size	specifies the size of the memory area. (Input)
mode	specifies the permissions to check. (Input)
	Only the lower 16 bits of mode are used.
mem_ptr	is a pointer to the beginning of the memory to check. (Input)
proc_desc	is a pointer to the process descriptor of the target process. (Input)



Note

`_os_chkmem()` only performs a useful function on MMU systems. On non-MMU systems, `_os_chkmem()` simply returns without checking the memory block's accessibility.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

F\$ChkMem
F_CHKMEM

OS-9 for 68K Technical Manual
OS-9 Technical Manual

_os_chmdir()

Change Process' Current Module Directory

Syntax

```
#include <moddir.h>
#include <types.h>
error_code _os_chmdir(const char *name);
```

Description

`_os_chmdir()` changes a process' current module directory to the directory specified by `name`.

`name` is a pointer to a full pathlist or a pathlist relative to the current module directory.
(Input)
To change to the system's root module directory, specify a slash (/) for `name`.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_close()

Close Path to File/Device

Syntax

```
#include <modes.h>
#include <types.h>
error_code _os_close(path_id path);
```

Description

`_os_close()` terminates the I/O path specified by `path`.

`path` is the path ID returned from a previous call to `_os_open()`, `_os_create()`, or `_os_dup()`. (Input)

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_create()`
`_os_detach()`
`_os_dup()`
`_os_exit()`
`_os_open()`
`I$Close`
`I_CLOSE`

OS-9 for 68K Technical Manual
OS-9 Technical Manual

Syntax

```
#include <process.h>
#include <types.h>
error_code _os_clrsigs(process_id proc_id);
```

Description

`_os_clrsigs()` removes any queued signals that have been sent to the target process.

`proc_id` specifies the target process ID number.
(Input)

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

[_os_sigmask\(\)](#)

_os_cmdperm()

Change Permissions of Module Directory

Syntax

```
#include <moddir.h>
#include <types.h>
error_code _os_cmdperm(
    const char      *name,
    u_int32         perm);
```

Description

`_os_cmdperm()` changes the access permissions of an existing module directory. This makes it possible to restrict access to a particular module directory.

name	is a pointer to the name of the existing module directory. (Input)
perm	specifies the new permissions. (Input) Only the lower 16 bits of <code>perm</code> are used.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

Syntax

```
#include <strings.h>
#include <types.h>
error_code _os_cmpnam(
    const char      *pattern,
    const char      *string,
    u_int32          length,
    int32            *cmp_result);
```

Description

`_os_cmpnam()` compares a target name to a source pattern to determine if they are equal. `_os_cmpnam()` is not case-sensitive; it does not differentiate between upper and lower case characters.

pattern	is a pointer to the pattern string. (Input)
	Two wildcard characters are recognized in the pattern string:
	<ul style="list-style-type: none">• A question mark (?) matches any single character.• An asterisk (*) matches any string.
string	is a pointer to the target name string. (Input)
	The target name must be terminated by a null byte.
length	specifies the length of the pattern string. (Input)

`cmp_result` is a pointer to the location where `_os_cmpnam()` stores the lexicographic result of the comparison. (Output)
`_os_cmpnam()` returns zero if the target string is the same as the pattern string.

OS-9 for 68K: If the error `EOS_PNNF` is returned, the strings differ and the integer at the address passed in `cmp_result` is not changed.

OS-9: If the error `EOS_DIFFER` is returned, the strings differ and the integer at `cmp_result` has the following meaning:

Table 2-17 `_os_cmpnam()` Parameter `cmp_result` Values

Integer	Description
positive	The target name is less than the pattern string.
negative	The target string is greater than the pattern string.



Note

On OS-9 for 68K, you never receive an `EOS_DIFFER` error. On OS-9, you never receive an `EOS_PNNF` error.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

F\$CmpNam

F_CMPNAM

OS-9 for 68K Technical Manual

OS-9000 Technical Manual

_os_config()

Configure an Element

Syntax

```
#include <sysglob.h>
error_code _os_config(
    u_int32      code,
    void        *param);
```

Description

`os_config()` is a wildcard call that configures elements of the operating system which may or may not be process specific. It dynamically reconfigures operating system resources at runtime. The target resources may be system-wide resources or they may be process-specific, depending on the nature of the configuration call being made.

code	is the configuration command code. (Input)
	Refer to the OS-9 Technical Manual for a list of valid codes for your version of the operating system.
param	is a pointer to any additional parameters required by the specified configuration function. (Input)

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

See Also

[_os_iocfg\(\)](#)

Syntax

```
#include <process.h>
#include <types.h>
error_code _os_cpy_ioproc(
    process_id          proc_id,
    void                *buffer,
    u_int32              count);
```

Description

`_os_cpy_ioproc()` copies the I/O process descriptor for the specified process into a buffer.

process_id	is the process ID. (Input)
buffer	is a pointer to the buffer in which to copy the process descriptor. (Output)
count	specifies the number of bytes to copy. (Input)

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_cpymem()

Copy External Memory

Syntax

```
#include <process.h>
#include <types.h>
error_code _os_cpymem(
    process_id      proc_id,
    void            *from,
    void            *to,
    u_int32         count);
```

Description

`_os_cpymem()` copies external memory into your buffer for inspection.

`proc_id` specifies the process ID of the owner of the external memory. (Input) Using a process ID of 0 allows access to the system memory.

`from` is a pointer to the external process' memory to copy. (Input)

`to` is a pointer to the caller's destination buffer. (Output)

`count` is the number of bytes to copy. (Input)

`_os_cpymem()` can copy portions of the system's address space. This is especially helpful in examining modules. Any memory in the system may be viewed in this way.

Attributes

Operating System: OS-9 and OS-9 for 68K

State: User

Threads: Safe

Re-entrant: Yes

Library

os_lib.l

See Also

[_os_move\(\)](#)

Syntax

```
#include <module.h>
#include <types.h>
error_code _os_crc(
    void      *start,
    u_int32   count,
    int32     *accum);
```

Description

`_os_crc()` generates or checks the CRC (cyclic redundancy check) values of sections of memory. Compilers, assemblers, and other module generators use `_os_crc()` to generate a valid module CRC.

start	is a pointer to the section of memory. (Input)
count	specifies the byte count for the section of memory. (Input)
accum	is a returned value. It is a pointer to the CRC accumulator. (Output)

If the CRC of a new module is to be generated, the CRC is accumulated over the module (excluding the CRC). The accumulated CRC is complemented and stored in the correct position in the module.

The CRC is calculated over a specified number of bytes starting at the source address. It is not necessary to cover an entire module in one call, because the CRC may be accumulated over several calls. The CRC accumulator must be initialized to -1 before the first `_os_crc` call for any particular module.

To generate the CRC of an existing module, the calculation must be performed on the entire module, including the module CRC. The CRC accumulator contains the CRC constant bytes if the module CRC is correct. The CRC constant is defined in `module.h` as `CRCCON`. The value is `0x00800fe3`.

To generate the CRC for a module:

1. Initialize the accumulator to -1.
2. Perform the CRC over the module.
3. Call `F_CRC` with a NULL value for `start`.
4. Complement the CRC accumulator.
5. Write the contents of the accumulator to the module.

The CRC value is three bytes long, in a four-byte field. To generate a valid module CRC, you must include the byte preceding the CRC in the check. You must initialize this byte to zero. For convenience, if a data pointer of zero is passed, the CRC is updated with one zero data byte. `_os_crc` always returns 0xff in the most significant byte of the `accum` parameter, so `accum` may be directly stored (after complement) in the last 4 bytes of a module as the correct CRC.



For More Information

Refer to your operating system technical manual for more information on how your operating system uses CRCs for modules.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`F$CRC`
`F_CRC`

OS-9 for 68K Technical Manual
OS-9 Technical Manual

_os_create()

Create Path to New File

Syntax

```
#include <modes.h>
#include <types.h>
error_code _os_create(
    const char    *name,
    u_int32       mode,
    path_id       *path,
    u_int32       perm, ...);
```

Description

`_os_create()` creates a new file.

- On multifile devices, the new file is entered in the directory structure.
- On non-multifile devices, `create` is synonymous with `open`.

`name` is a pointer to the path name of the new file. (Input)

`mode` specifies the access mode. (Input)

If data is to be written to the file, `mode` must have the `FAM_WRITE` or `S_IWRITE` bit set. Only the lower 16 bits of `mode` are used.

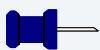


Note

All available access modes are defined in the following location:

`MWOS\<OS>\SRC\DEFS\modes.h`

path	is a pointer to the location where <code>_os_create()</code> stores the path number that identifies the file in subsequent I/O service requests until the file is closed. (Output)
perm	<p>specifies the attributes to use for the new file. (Input)</p> <p>Only the lower 16 bits of <code>perm</code> are used.</p> <p>If either the <code>FAM_SIZE</code> bit or the <code>S_ISIZE</code> (initial file size) bit is set, you can also pass an initial file size estimate as a variable argument.</p> <p>If either the <code>FAM_NOCREATE</code> bit or the <code>S_IEXCL</code> (exclusive) bit is set, an error is returned if <code>name</code> already exists. If neither <code>FAM_NOCREATE</code> nor <code>S_EXCL</code> are set, any existing file called <code>name</code> is deleted.</p>



Note

All available permissions are defined in the following location:

`MWOS\<OS>\SRC\DEFS\modes.h`



For More Information

For a list of all available access modes and permissions, refer to the section, “Access Modes and Permissions,” in Chapter 3 of the **OS-9 Technical Manual**.

By default, if the pathlist specifies a file name that already exists, `_os_create()` recreates the file.



Note

On OS-9 for 68K, the binding for this call exists for compatibility. If speed is an issue, use `_os9_create()`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_attach()`
`_os_open()`
`_os_close()`
`_os_mkdir()`
`I$Create`
`I_CREATE`

OS-9 Technical Manual
OS-9 Technical Manual

Syntax

```
#include <modes.h>
#include <types.h>
error_code _os9_create(
    const char      *name,
    u_int32         mode,
    path_id         *path,
    u_int32         perm, ...);
```

Description

`_os9_create()` creates a new file.

- On multifile devices, the new file name is entered in the directory structure.
 - On non-multifile devices, `create` is synonymous with `open`.

name is a pointer to the path name. (Input)

mode specifies the access mode. (Input)

If data is to be written to the file, mode must have the write bit set. Only the lower 16 bits of mode are used.



Note

All available access modes are defined in the following location:

MWOS\<OS>\SRC\DEFS\modes.h

path

is a pointer to the location where
`_os9_create()` stores the path
number. (Output)

perm

specifies the attributes to use for the new file. (Input)

Only the lower 16 bits of perm are used.

If either the FAM_SIZE bit or the S_ISIZE (initial file size) bit is set, you can also pass an initial file size estimate as a variable argument.



Note

All available permissions are defined in the following location:

MWOS\<OS>\SRC\DEFS\modes.h



For More Information

For a list of all available access modes and permissions, refer to the section, “Access Modes and Permissions,” in Chapter 3 of the **OS-9 Technical Manual**.

By default, if the pathlist specifies an existing file name, `_os9_create()` returns an error.

Attributes

Operating System:	OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

[_os_attach\(\)](#)
[_os_close\(\)](#)
[_os_mkdir\(\)](#)
[_os_open\(\)](#)
[I\\$Create](#)

OS-9 Technical Manual

Syntax

```
#include <lock.h>
#include <types.h>
error_code _os_crlk(lk_desc **lock);
```

Description

`_os_crlk()` creates a new resource lock descriptor. A resource lock descriptor is allocated and initialized to a free state (that is, not currently owned).

<code>lock</code>	<code>lock</code> is a pointer to the location where <code>_os_crlk()</code> stores the pointer to the lock descriptor. (Output)
	<code>lock</code> is used as a handle to perform further operations on the lock.

Attributes

Operating System:	OS-9
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

`lock.l`

See Also

<code>F_ACQLK</code>	<i>OS-9 Technical Manual</i>
<code>F_CASLK</code>	<i>OS-9 Technical Manual</i>
<code>F_CRLK</code>	<i>OS-9 Technical Manual</i>
<code>F_DDLK</code>	<i>OS-9 Technical Manual</i>
<code>F_DELLK</code>	<i>OS-9 Technical Manual</i>
<code>F_RELlk</code>	<i>OS-9 Technical Manual</i>
<code>F_WAITLK</code>	<i>OS-9 Technical Manual</i>

Syntax

```
#include <module.h>
#include <types.h>
error_code _os_datmod(
    const char      *mod_name,
    u_int32         size,
    u_int16         *attr_rev,
    u_int16         *type_lang,
    u_int32         perm,
    void            **mod_exec,
    mh_data         **mod_head);
```

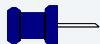
Description

`_os_datmod()` creates a data module with the specified attribute/revision and clears the data portion of the module. The module is created and entered into the system module directory.

mod_name	is a pointer to the module name string. (Input)
size	is the size of the data portion. (Input)
attr_rev	The module's desired attribute and revision are passed at the location pointed to by <code>attr_rev</code> . (Input/Output) <code>_os_datmod()</code> also stores the module's attribute and revision at the location pointed to be <code>attr_rev</code> .
type_lang	The module's desired type and language are passed at the location pointed to by <code>type_lang</code> . (Input/Output) <code>_os_datmod()</code> also stores the module's actual type and language at the location pointed to by <code>type_lang</code> .

perm	specifies the access permissions for the module. (Input) Only the lower 16 bits of <code>perm</code> are used.
mod_exec	is a pointer to the location where <code>_os_datmod()</code> stores the pointer to the module data. (Output)
mod_head	is a pointer to the location where <code>_os_datmod()</code> stores the pointer to the module header. (Output)

Be careful not to alter the data module's header or name string to prevent the module from becoming unknown to the system.



Note

The created module contains at least `size` usable data bytes but may be somewhat larger. The module itself is larger by at least the size of the module header, CRC, and name string; and it is rounded up to the nearest system memory allocation boundary.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_setcrc()`
`_os_move()`

F\$DatMod
`F_DATMOD`

OS-9 Technical Manual
OS-9 Technical Manual

Syntax

```
#include <process.h>
#include <types.h>
error_code _os_ddlk(process_id proc_id);
```

Description

`_os_ddlk()` checks for a deadlock situation between processes. A search for the current process (calling process) in the linked list of potential conflictors is begun from the process specified by `proc_id`.

`proc_id` specifies the process with which to begin the search. (Input)

If the calling process is already in the linked list of processes, an `EOS_DEADLK` error is returned to the caller.

If the process is not in the linked list, the current process is added to the list associated with `proc_id`.

`_os_ddlk()` is useful for preventing a deadlock situation when protecting multiple resources from simultaneous accesses. The deadlock list usually represents the list of processes waiting to acquire access to an associated resource.

Attributes

Operating System:	OS-9
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

`lock.l`

See Also

F_ACQLK
F_CAQLK
F_CRLK
F_DDLK
F_DELLK
F_RELlk
F_WAITLK

OS-9 Technical Manual
OS-9 Technical Manual

Syntax

```
#include <memory.h>
#include <types.h>
error_code _os9_delbit(
    u_int32      bit_number,
    u_int32      count,
    void        *address);
```

Description

`_os9_delbit()` clears bits in a buffer that represents a bit map. Bit numbers range from 0 to $n - 1$, where n is the number of bits in the bitmap.

bit_number	specifies the bit number of the first bit to clear. (Input)
	Only the lower 16 bits of <code>bit_number</code> are used.
count	specifies the number of bits to clear. (Input)
	Only the lower 16 bits of <code>count</code> are used.
address	is a pointer to the base address of a bitmap. (Output)

Attributes

Operating System:	OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

[_os9_allbit\(\)](#)
[_os_cpymem\(\)](#)
[_os9_schbit\(\)](#)
[F\\$DelBit](#)

OS-9 Technical Manual

Syntax

```
#include <modes.h>
#include <types.h>
error_code _os_delete(
    const char      *name,
    u_int32         mode);
```

Description

`_os_delete()` deletes a file. The caller must have non-sharable write access to the file (the file may not already be open) or an error results.

`name` is a pointer to the file to delete. (Input)

`mode` specifies the access mode. (Input)

Possible values for `mode` include:

FAM_READ FAM_WRITE FAM_EXEC
S_IREAD S_IWRITE S_IEXEC

Attempts to delete from non-multifile devices result in an error. The access mode specifies the data or execution directory (but not both) in the absence of a full pathlist. Only the lower 16 bits of `mode` are used.



Note

All available access modes are defined in the following location:

MWOS\<OS>\SRC\DEFS\modes.h



For More Information

For a list of all available access modes, refer to the section, “Access Modes and Permissions,” in Chapter 3 of the ***OS-9 Technical Manual***.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

[_os_attach\(\)](#)
[_os_create\(\)](#)
[_os_detach\(\)](#)
[_os_open\(\)](#)

I\$Delete
I_DELETE

OS-9 Technical Manual
OS-9 Technical Manual

Syntax

```
#include <clock.h>
#include <types.h>
error_code _os_dellk(lk_desc *lock);
```

Description

`_os_dellk()` deletes an existing lock descriptor.

`lock` is a pointer to the `lk_desc` structure for the lock to delete. (Input)

`_os_dellk()` does not check for suspended processes still waiting to acquire the lock; an implementation using locks must release all processes waiting on a resource lock before deleting it. Failure to release suspended processes prior to deletion could result in system corruption.

Attributes

Operating System:	OS-9
State:	User
Threads:	Safe
Re-entrant:	Yes

Library

`lock.l`

See Also

<code>F_ACQLK</code>	<i>OS-9 Technical Manual</i>
<code>F_CAQLK</code>	<i>OS-9 Technical Manual</i>
<code>F_CRLK</code>	<i>OS-9 Technical Manual</i>
<code>F_DDLK</code>	<i>OS-9 Technical Manual</i>
<code>F_DELLK</code>	<i>OS-9 Technical Manual</i>
<code>F_RELINK</code>	<i>OS-9 Technical Manual</i>
<code>F_WAITLK</code>	<i>OS-9 Technical Manual</i>

_os_delmdir()

Delete Existing Module Directory

Syntax

```
#include <moddir.h>
#include <types.h>
error_code _os_delmdir(const char *name);
```

Description

`_os_delmdir()` deletes an existing module directory. The directory must be empty.

name	is a pointer to the module directory. (Input)
------	--

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

F_DELMDIR ***OS-9 Technical Manual***

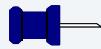
Syntax

```
#include <process.h>
#include <types.h>
error_code _os_deltsk(pr_desc *proc_desc);
```

Description

`_os_deltsk()` returns the process' protection resources when the process terminates.

`proc_desc` is a pointer to the process descriptor.
(Input)



Note

`_os_deltsk()` only performs a useful function on MMU/SSM systems. On non-MMU/SSM systems, `_os_deltsk()` simply returns without deallocating the process descriptor.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

[_os_alltsk\(\)](#)
[_os_chkmem\(\)](#)
[_os_protect\(\)](#)
[_os_permit\(\)](#)
[F\\$DelTsk](#)
[F_DELTSK](#)

OS-9 for 68K Technical Manual

OS-9 Technical Manual

_os_detach()

Remove Device from System

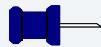
Syntax

```
#include <io.h>
#include <types.h>
error_code _os_detach(dev_list *dev_tbl);
```

Description

`_os_detach()` removes a device from the system device table if no other process is using the device.

`dev_tbl` is a pointer to the address of the device table entry. (Input)



Note

On OS-9 for 68K, if an invalid address is passed in `dev_tbl`, the system may crash or undergo severe damage.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_attach()`
`_os_close()`

`I$Detach`
`I_DETACH`

OS-9 for 68K Technical Manual
OS-9 Technical Manual

Syntax

```
#include <dexec.h>
#include <types.h>
error_code _os_dexec(
    process_id          proc_id,
    dexec_mode          mode,
    void                *params,
    u_int32              *num_instr,
    u_int32              num_bpts,
    void                **brk_pts,
    u_int32              *tot_instr,
    dexec_status         *status,
    u_int32              *except,
    void                **addr,
    error_code           *exit_status);
```

Description

OS-9 for 68K:

`_os_dexec()` controls the execution of a suspended child process created by either `_os9_dfork()` or by `_os_dfork()`. OS-9 for 68K uses `_os9_dfork()` and OS-9 uses `_os_dfork()`. The process performing `_os_dexec()` is suspended, and its debugged child process is executed instead. This process terminates and control returns to the parent after the specified number of instructions have been executed, a breakpoint is reached, or an unexpected exception occurs. The parent and the child processes are never active at the same time.

proc_id	is the process ID of the child to execute. (Input)
mode	specifies the debug mode. (Input)
params	is the parameter list pointer. (Input)

<code>num_instr</code>	points to the location where the number of instructions to execute is passed. (Input/Output)
	<code>_os_dexec()</code> also stores the number of instructions not executed at the location pointed to by <code>num_instr</code> . If the integer at <code>num_instr</code> is zero, commands are executed continuously.
<code>num_bpts</code>	specifies the number of breakpoints in the list. (Input)
	Maximum of 16 break points allowed.
<code>brk_pts</code>	is a pointer to the pointer to the breakpoint list. (Input)
<code>tot_instr</code>	points to where the number of instructions executed so far is passed. (Input/Output)
	<code>_os_dexec()</code> also stores the updated total number of instructions at the location pointed to by <code>tot_instr</code> .
<code>status</code>	is a pointer to the location where <code>_os_dexec()</code> stores the return status. (Output)
<code>except</code>	is a pointer to the location where <code>_os_dexec()</code> stores the exception that the child received. (Output)
<code>addr</code>	is a pointer to the location where <code>_os_dexec()</code> stores the violation address. (Output)
<code>exit_status</code>	is a pointer to the location where <code>_os_dexec()</code> stores the exit status of the child. (Output)

An `_os_dexit()` call must be made for the resources (memory) of the debugged process to be returned.



For More Information

Refer to your operating system technical manual for system-specific information.



Note

Tracing is allowed through subroutine libraries, intercept routines, and OS-9 for 68K user-state trap handlers. This is not a problem, but may seem strange at times.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

[_os_dfork\(\)](#)
[_os9_dfork\(\)](#)
[_os_dexit\(\)](#)

F\$DExec
F_DEXEC
F_DFORK

OS-9 for 68K Technical Manual
OS-9 Technical Manual
OS-9 Technical Manual

Syntax

```
#include <dexec.h>
#include <types.h>
error_code _os_dexit(process_id proc_id);
```

Description

`_os_dexit()` terminates a suspended child process created by `_os_dfork()`. To allow examination of the child after its termination, normal termination by the child process does not release any resources.

`proc_id` is the process ID of the child to terminate. (Input)

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

[_os_exit\(\)](#)
[_os_dfork\(\)](#)
[_os9_dfork\(\)](#)
[_os_dexec\(\)](#)

`F$DExit`
`F_DEXIT`

OS-9 Technical Manual
OS-9 Technical Manual

_os_dfork()

Fork Process Under Control of Debugger

Syntax

```
#include <dexec.h>
#include <types.h>

error_code _os_dfork(
                    u_int32      priority,
                    u_int32      path_cnt,
                    const char  *mod_name,
                    const void   *params,
                    u_int32      param_size,
                    u_int32      mem_size,
                    process_id  *proc_id,
                    u_int32      type_lang,
                    regs        *reg_stack,
                    fregs       *fregstack);
```

Description

`_os_dfork()` creates a new process which becomes a child of the caller. It sets up the process' memory, MPU registers, and standard I/O paths. In addition, `_os_dfork()` allows a debugger utility to create a process whose execution can be closely controlled. The created process is not placed in the active queue, but is left in a suspended state. This allows the debugger to control its execution through the `_os_dexec()` and `_os_dexit()` calls.

<code>priority</code>	is the priority of the new process. (Input) Only the lower 16 bits of <code>priority</code> are used.
<code>path_cnt</code>	is the number of I/O paths for the child to inherit. (Input) Only the lower 16 bits of <code>path_cnt</code> are used.
<code>mod_name</code>	is a pointer to the module name. (Input)

params	is a pointer to the parameter block. (Input)
param_size	specifies the size of the parameter block. (Input)
mem_size	specifies any additional stack space to allocate. (Input)
	mem_size is specified in bytes.
proc_id	is a pointer to the location where _os_dfork() stores the child process ID. (Output)
type_lang	specifies the desired type and language. (Input)
	Only the lower 16 bits of type_lang are used.
reg_stack	is a pointer to the register buffer. (Input)
freg_stack	is a pointer to the floating point register buffer. (Input)

Attributes

Operating System:	OS-9
State:	User
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

[_os_dexit\(\)](#)
[_os_fork\(\)](#)
[_os9_dfork\(\)](#)
[_os_dexec\(\)](#)
F\$DExit
F_DFORK

OS-9 Technical Manual
OS-9 Technical Manual

_os9_dfork()

Fork Process Under Control of Debugger

Syntax

```
#include <dexec.h>
#include <types.h>
error_code _os9_dfork(
    u_int32      priority,
    u_int32      path_cnt,
    const char   *mod_name,
    const void   *params,
    u_int32      param_size,
    u_int32      mem_size,
    process_id   *proc_id,
    u_int32      type_lang,
    void         *reg_buf);
```

Description

`_os9_dfork()` creates a new process which becomes a child of the caller. It sets up the new process' memory, MPU registers, and standard I/O paths. In addition, `_os9_dfork()` allows a debugger utility to create a process whose execution can be closely controlled. The process is not placed in the active queue, but is left in a suspended state. This allows the debugger to control its execution through the `_os_dexec()` and `_os_dexit()` system calls.

priority	is the priority of the new process. (Input) Only the lower 16 bits of <code>priority</code> are used.
path_cnt	is the number of the I/O paths for a child to inherit. (Input) Only the lower 16 bits of <code>path_cnt</code> are used.
mod_name	is the module name pointer. (Input)

params	is a pointer to an additional parameter block. (Input)
param_size	is the size of the parameter block. (Input)
mem_size	specifies any additional stack space to allocate. (Input)
mem_size	is specified in bytes.
proc_id	is a pointer to the location where _os9_dfork() stores the child process ID. (Output)
type_lang	specifies the desired type and language. (Input)
type_lang	Only the lower 16 bits of type_lang are used.
reg_buf	is a pointer to a copy of all child registers. (Input)

Attributes

Operating System:	OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

[_os_dexec\(\)](#)
[_os_dexit\(\)](#)
[_os_fork\(\)](#)
 F\$DFork

OS-9 for 68K Technical Manual

Syntax

```
#include <dexec.h>
#include <types.h>
error_code _os_dforkm(
    u_int32      priority,
    u_int32      path_cnt,
    mh_com       *mod_head,
    void         *params,
    u_int32      param_size,
    u_int32      mem_size,
    process_id   *proc_id,
    regs         *reg_stack,
    fregs        *fregstack);
```

Description

`_os_dforkm()` creates a new process which becomes a child of the caller. It sets up the process' memory, MPU registers, and standard I/O paths. In addition, `_os_dforkm()` allows a debugger utility to create a process whose execution can be closely controlled. The created process is not placed in the active queue, but is left in a suspended state. This allows the debugger to control its execution through the `_os_dexec()` and `_os_dexit()` calls. `_os_dforkm()` is similar to `_os_dfork()`. However, `_os_dforkm()` is passed a pointer to the module to fork rather than the module name.

<code>priority</code>	is the priority of the new process. (Input)
	Only the lower 16 bits of <code>priority</code> are used.
<code>path_cnt</code>	is the number of I/O paths for the child to inherit. (Input)
	Only the lower 16 bits of <code>path_cnt</code> are used.
<code>mod_head</code>	is a pointer to the module header. (Input)

params	is a pointer to the parameter block. (Input)
param_size	specifies the size of the parameter block. (Input)
mem_size	specifies any additional stack space to allocate. (Input)
proc_id	is a pointer to the location where _os_dforkm() stores the child process ID. (Output)
reg_stack	is a pointer to the register buffer. (Input)
fregstack	is a pointer to the floating point register buffer. (Input)

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

[_os_fork\(\)](#)
[_os_dexec\(\)](#)
[_os_dexit\(\)](#)
[_os_dfork\(\)](#)

Syntax

```
#include <modes.h>
#include <types.h>
error_code _os_dup(
    path_id    dup_path,
    path_id    *new_path);
```

Description

`_os_dup()` returns a synonymous path number for an existing file or device.

`dup_path` is the path number of the path to duplicate. (Input)

`new_path` is a pointer to the location where `_os_dup()` stores the new path number for the same path. (Output)

`_os_dup()` always uses the lowest available path number. For example, if you perform an `_os_close()` on path #0 and an `_os_dup()` on path #4, path #0 is returned as the new path number. Therefore, service requests using either the old or new path numbers operate on the same file or device.



Note

It is usually not a good idea for more than one process to be performing I/O on the same path concurrently.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

I\$Dup

I_DUP

OS-9 for 68K Technical Manual

OS-9 Technical Manual

_os_ev_allclr()

Wait for All Bits Defined by Mask to Become Clear

Syntax

```
#include <events.h>
#include <types.h>
error_code _os_ev_allclr(
    event_id      ev_id,
    int32         *value,
    signal_code   *signal,
    u_int32       mask);
```

Description

`_os_ev_allclr()` waits for an event to occur. That is, it waits until an `_os_ev_signal()` occurs that clears all of the bits corresponding to the set bits in the mask.

ev_id	identifies the event. (Input)
value	is a pointer to the location where <code>_os_ev_allclr()</code> stores the actual event value. (Output)
signal	is a pointer to the location where <code>_os_ev_allclr()</code> stores the signal code. (Output)
	If the process receives a signal while in the event queue, it is activated even though the event has not actually occurred.
mask	specifies the activation mask. (Input)

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

F_EVENT

F_EVENT, EV_ALLCR

F_EVENT, EV_SIGNAL

OS-9 Technical Manual

OS-9 Technical Manual

OS-9 Technical Manual

_os_ev_allset()

Wait for Event to Occur

Syntax

```
#include <events.h>
#include <types.h>
error_code _os_ev_allset(
    event_id          ev_id,
    int32             *value,
    signal_code       *signal,
    u_int32           mask);
```

Description

`_os_ev_allset()` waits for an event to occur. That is, it waits until an `_os_ev_signal()` occurs that sets all of the bits corresponding to the set bits in the mask.

ev_id	identifies the event. (Input)
value	is a pointer to the location where <code>_os_ev_allset()</code> stores the actual event value. (Output)
signal	is a pointer to the location where <code>_os_ev_allset()</code> stores the signal code. (Output)
	If the process receives a signal while in the event queue, it is activated even though the event has not actually occurred.
mask	specifies the activation mask.(Input)

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

F_EVENT

F_EVENT, EV_ALLSET

F_EVENT, EV_SIGNAL

OS-9 Technical Manual

OS-9 Technical Manual

OS-9 Technical Manual

_os_ev_anycclr()

Wait for Event to Occur

Syntax

```
#include <events.h>
#include <types.h>
error_code _os_ev_anycclr(
    event_id           ev_id,
    int32              *value,
    signal_code        *signal,
    u_int32            mask);
```

Description

`_os_ev_anycclr()` waits for an event to occur. It waits until an `_os_ev_signal()` occurs that clears any of the bits corresponding to the set bits in the mask.

ev_id	identifies the event. (Input)
value	is a pointer to the location where <code>_os_ev_anycclr()</code> stores the actual event value. (Output)
signal	is a pointer to the location where <code>_os_ev_anycclr()</code> stores the signal code. (Output)
	If the process receives a signal while in the event queue, it is activated even though the event has not actually occurred.
mask	specifies the activation mask. (Input)

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

F_EVENT

F_EVENT, EV_ANYCLR

F_EVENT, EV_SIGNL

OS-9 Technical Manual

OS-9 Technical Manual

OS-9 Technical Manual

_os_ev_anyset()

Wait for Event to Occur

Syntax

```
#include <events.h>
#include <types.h>
error_code _os_ev_anyset(
    event_id          ev_id,
    int32             *value,
    signal_code       *signal,
    u_int32           mask);
```

Description

`_os_ev_anyset()` waits for an event to occur. That is, it waits until an `_os_ev_signal()` occurs that sets any of the bits corresponding to the set bits in the mask.

ev_id	identifies the event. (Input)
value	is a pointer to the location where <code>_os_ev_anyset()</code> stores the actual event value. (Output)
signal	is a pointer to the location where <code>_os_ev_anyset()</code> stores the signal code. (Output)
	If the process receives a signal while in the event queue, it is activated even though the event has not actually occurred.
mask	specifies the activation mask. (Input)

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

F_EVENT

F_EVENT, EV_ANYSET

F_EVENT, EV_SIGNAL

OS-9 Technical Manual

OS-9 Technical Manual

OS-9 Technical Manual

_os_ev_change()

Wait for Event to Occur

Syntax

```
#include <events.h>
#include <types.h>
error_code _os_ev_change(
    event_id          ev_id,
    int32             *value,
    signal_code       *signal,
    u_int32           mask,
    u_int32           pattern);
```

Description

`_os_ev_change()` waits for an event to occur. That is, it waits until an `_os_ev_signal()` occurs that changes any of the bits corresponding to the set bits in `mask`.

ev_id	identifies the event. (Input)
value	is a pointer to the location where <code>_os_ev_change()</code> stores the actual event value. (Output)
signal	is a pointer to the location where <code>_os_ev_change()</code> stores the signal code. (Output)
	If the process receives a signal while in the event queue, it is activated even though the event has not actually occurred.
mask	specifies the activation mask. (Input)
pattern	specifies the wait pattern. (Input)

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

F_EVENT
F_EVENT , EV_CHANGE
F_EVENT , EV_SIGNL

OS-9 Technical Manual

OS-9 Technical Manual

OS-9 Technical Manual

_os_ev_creat()

Create New Event

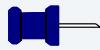
Syntax

```
#include <events.h>
#include <types.h>
error_code _os_ev_creat(
    int32          winc,
    int32          sinc,
    u_int32        perm,
    event_id       *ev_id,
    const char     *name,
    int32          value,
    u_int32        color);
```

Description

`_os_ev_creat()` allows you to create events dynamically as needed. When an event is created, an initial value is specified, as well as increments to apply each time the event is waited for or an event signal occurs. Subsequent `_os_event` calls use the returned ID number to refer to the created event.

winc	specifies the auto-increment value for <code>_os_ev_wait()</code> . (Input)
sinc	Only the lower 16 bits of <code>winc</code> are used. (Input)
perm	specifies the auto-increment value for <code>_os_ev_signal()</code> . (Input)
perm	Only the lower 16 bits of <code>sinc</code> are used. specifies the access permissions. (Input)
perm	Only the lower 16 bits of <code>perm</code> are used.



Note

OS-9 for 68K ignores perm.

ev_id	is a pointer to the location where _os_ev_creat() stores the event identifier. (Output)
name	is a pointer to the event name string. (Input)
value	specifies the initial event variable value. (Input)
color	specifies the memory type for the event structure. (Input) Only the lower 16 bits of color are used.



Note

OS-9 for 68K ignores color.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

[_os_ev_delete\(\)](#)
[_os_ev_signal\(\)](#)
[_os_ev_wait\(\)](#)
[_os9_ev_wait\(\)](#)
[F\\$Event](#)
[F_EVENT](#)
[F_EVENT, EV_CREAT](#)
[F_EVENT, EV_SIGNL](#)

OS-9 for 68K Technical Manual or
OS-9 Technical Manual
OS-9 Technical Manual
OS-9 Technical Manual

_os_ev_delete()

Delete Existing Event

Syntax

```
#include <events.h>
#include <types.h>
error_code _os_ev_delete(const char *name);
```

Description

`_os_ev_delete()` removes an event from the system event table, freeing the entry for use by another event. An event may not be deleted unless its use count is zero.

`name` is a pointer to the event's name string.
(Input)

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

_os_ev_creat()

F\$Event

F EVENT

F_EVENT, EV_DELETE

F_EVENT, EV_SIGNAL

OS-9 for 68K Technical Manual

OS-9 Technical Manual

OS-9 Technical Manual

OS-9 Technical Manual

_os_ev_info()

Return Event Information

Syntax

```
#include <events.h>
#include <types.h>
```

OS-9 for 68K:

```
error_code _os_ev_info(
    event_id      ev_id,
    u_int32        size,
    ev_infostr    *buffer);
```

OS-9:

```
error_code _os_ev_info(
    event_id      ev_id,
    u_int32        size,
    void          *buffer);
```

Description

`_os_ev_info()` returns event information in the caller's buffer. The returned information is defined by the `ev_infostr` event information structure defined in the `events.h` header file.

<code>ev_id</code>	specifies the event index to use to begin the search. (Input)
<code>size</code>	specifies the buffer size. (Input)
<code>buffer</code>	is a pointer to the buffer containing the event information. (Output)

On OS-9, the name of the event is appended to the end of the information structure; `buffer` and `size` must be large enough to accommodate the name of the target event.

`_os_ev_info()` returns the event information block for the first active event whose index is greater than or equal to this index. If no such event exists, an error is returned.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

F\$Event

F_EVENT

F_EVENT, EV_INFO

F_EVENT, EV_SIGNL

OS-9 for 68K Technical Manual

OS-9 Technical Manual

OS-9 for 68K Technical Manual

OS-9 for 68K Technical Manual

OS-9 for 68K Technical Manual

_os9_ev_info()

Return Event Information

Syntax

```
include <events.h>
#include <types.h>
error_code _os9_ev_info(
    u_int32      *index,
    ev_infostr   *buffer);
```

Description

`os9_ev_info()` returns event information in the caller's buffer. The returned information is defined by the `ev_infostr` event information structure defined in `events.h`.

index	The event index to begin the search for a valid event is passed at the location pointed to by <code>index</code> . (Input) <code>_os9_ev_info()</code> also stores the event index of the valid event found (if any) at the location pointed to by <code>index</code> .
buffer	is a pointer to a buffer large enough to hold the event information. (Output)

Attributes

Operating System:	OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`os_lib.l`

See Also

`F$Event`

OS-9 for 68K Technical Manual

Syntax

```
#include <events.h>
#include <types.h>
error_code _os_ev_link(
    const char      *name,
    event_id        *ev_id);
```

Description

`_os_ev_link()` determines the ID number of an existing event. To keep the use count synchronized properly, you must perform an `_os_ev_unlink()` call when the event is longer be used.

name	is a pointer to the event name string. (Input)
ev_id	is a pointer to the location where <code>_os_ev_link()</code> stores the event identifier. (Output)

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

[_os_ev_unlink\(\)](#)

F\$Event

F_EVENT

F_EVENT, EV_LNK

F_EVENT, EV_UNLNK

OS-9 for 68K Technical Manual

OS-9 Technical Manual

OS-9 Technical Manual

OS-9 Technical Manual

_os_ev_pulse()

Signal Event Occurrence

Syntax

```
#include <events.h>
#include <types.h>
error_code _os_ev_pulse(
    event_id      ev_id,
    int32         *value,
    u_int32       actv_flag);
```

Description

`_os_ev_pulse()` signals an event occurrence. The event value is set to what is passed at the location `value`, and the signal auto-increment is not applied. Then, the `_os_ev_signal()` search routine is executed and the original event value is restored.

ev_id	identifies the event. (Input)
value	The value to pulse the event to is passed at the location pointed to by <code>value</code> . (Input/Output) <code>_os_ev_pulse()</code> also stores the event value prior to the pulse at the location pointed to by <code>value</code> .
actv_flag	specifies which process(es) to activate. (Input) <ul style="list-style-type: none">• If <code>actv_flag</code> is one, all processes in range are activated.• If <code>actv_flag</code> is zero, the first process in the event queue waiting for that range is activated.

Attributes

Operating System:
State:
Threads:
Re-entrant:

OS-9 and OS-9 for 68K
User and System
Safe
Yes

Library

os_lib.l

See Also

[_os_ev_signal\(\)](#)
[F\\$Event](#)
[F_EVENT](#)
[F_EVENT, EV_PULSE](#)
[F_EVENT, EV_SIGNL](#)

OS-9 for 68K Technical Manual
OS-9 Technical Manual
OS-9 Technical Manual
OS-9 Technical Manual

_os_ev_read()

Read Event Value Without Waiting

Syntax

```
#include <events.h>
#include <types.h>
error_code _os_ev_read(
    event_id      ev_id,
    int32         *value);
```

Description

`_os_ev_read()` reads the value of an event without waiting for or affecting the event variable.

ev_id	identifies the event. (Input)
value	is a pointer to the location where <code>_os_ev_read()</code> stores the current event value. (Output)

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

F\$Event	<i>OS-9 for 68K Technical Manual</i>
F_EVENT	<i>OS-9 Technical Manual</i>
F_EVENT, EV_READ	<i>OS-9 Technical Manual</i>
F_EVENT, EV_SIGNAL	<i>OS-9 Technical Manual</i>

_os_ev_set()

Set Event Variable and Signal Event Occurrence

Syntax

```
#include <events.h>
#include <types.h>
error_code _os_ev_set(
    event_id      ev_id,
    int32         *value,
    u_int32        actv_flag);
```

Description

`_os_ev_set()` signals that an event has occurred. The current event variable is set to the value passed at `value`. Then, the event queue is searched for the first process waiting for that event.

<code>ev_id</code>	identifies the event. (Input)
<code>value</code>	is a pointer to the location where <code>_os_ev_set()</code> stores the event value prior to the set operation. (Output)
<code>actv_flag</code>	specifies which process(es) to activate. (Input) <ul style="list-style-type: none">• If <code>actv_flag</code> is one, all processes in range are activated.• If <code>actv_flag</code> is zero, the first process in the event queue waiting for that range is activated.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

[_os_ev_signal\(\)](#)

F\$Event

F_EVENT

F_EVENT, EV_SET

F_EVENT, EV_SIGNL

OS-9 for 68K Technical Manual

OS-9 Technical Manual

OS-9 Technical Manual

OS-9 Technical Manual

Syntax

```
#include <events.h>
#include <types.h>
error_code _os_ev_setand(
    event_id      ev_id,
    int32         *value,
    u_int32       mask,
    u_int32       actv_flag)
```

Description

`_os_ev_setand()` signals that an event has occurred. The current event variable is ANDed with the value passed in `mask`. Then, the event queue is searched for the first process waiting for that event value.

ev_id	identifies the event. (Input)
value	is a pointer to the location where <code>_os_ev_setand()</code> stores the event value prior to the logical operation. (Output)
mask	is the event mask. (Input)
actv_flag	specifies which process(es) to activate. (Input) <ul style="list-style-type: none">• If <code>actv_flag</code> is one, all processes in range are activated.• If <code>actv_flag</code> is zero, the first process in the event queue waiting for that range is activated.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

F_EVENT	<i>OS-9 Technical Manual</i>
F_EVENT , EV_SETAND	<i>OS-9 Technical Manual</i>
F_EVENT , EV_SIGNL	<i>OS-9 Technical Manual</i>

Syntax

```
#include <events.h>
#include <types.h>
error_code _os_ev_setor(
    event_id      ev_id,
    int32         *value,
    u_int32       mask,
    u_int32       actv_flag);
```

Description

`_os_ev_setor()` signals that an event has occurred. The current event variable is ORed with the value passed in `mask`. Then, the event queue is searched for the first process waiting for that event value.

ev_id	identifies the event. (Input)
value	is a pointer to the location where <code>_os_ev_setor()</code> stores the event value prior to the logical operation. (Output)
mask	is the event mask. (Input)
actv_flag	specifies which process(es) to activate. (Input) <ul style="list-style-type: none">• If <code>actv_flag</code> is one, all processes in range are activated.• If <code>actv_flag</code> is zero, the first process in the event queue waiting for that range is activated.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

F_EVENT	<i>OS-9 Technical Manual</i>
F_EVENT , EV_SETOR	<i>OS-9 Technical Manual</i>
F_EVENT , EV_SIGNL	<i>OS-9 Technical Manual</i>

Syntax

```
#include <events.h>
#include <types.h>
error_code _os_ev_setr(
    event_id      ev_id,
    int32         *value,
    u_int32       actv_flag);
```

Description

`_os_ev_setr()` signals that an event has occurred. The current event value is incremented by `value` (output). Then, the event queue is searched for the first process waiting for that event value. Arithmetic underflows or overflows are set to 0x80000000 (minimum integer) or 0x7fffffff (maximum integer), respectively.

`ev_id` identifies the event. (Input)

The amount to add to the event value is passed at the location pointed to by `value`.

- **OS-9 for 68K:** `_os_ev_setr()` stores the event value **before** the increment at `value`.
- **OS-9:** `_os_ev_setr()` stores the event value **after** the increment at `value`.

`actv_flag` specifies which process(es) to activate. (Input)

- If `actv_flag` is one, all processes in range are activated.
- If `actv_flag` is zero, the first process in the event queue waiting for that range is activated.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

[_os_ev_set\(\)](#)
[_os_ev_signal\(\)](#)

F\$Event
F_EVENT, EV_SETR
F_EVENT, EV_SIGNL

OS-9 for 68K Technical Manual

OS-9 Technical Manual

OS-9 Technical Manual

Syntax

```
#include <events.h>
#include <types.h>
error_code _os_ev_setxor(
    event_id      ev_id,
    int32         *value,
    u_int32        mask,
    u_int32        actv_flag);
```

Description

`_os_ev_setxor()` signals that an event has occurred. The current event value is **EXCLUSIVE-ORed** with `mask`. Then, the event queue is searched for the first process waiting for that event value.

ev_id	identifies the event. (Input)
value	is a pointer to the location where <code>_os_ev_setxor()</code> stores the event value prior to the logical operation. (Output)
mask	specifies the event mask. (Input)
actv_flag	specifies which process(es) to activate. (Input) <ul style="list-style-type: none">• If <code>actv_flag</code> is one, all processes in range are activated.• If <code>actv_flag</code> is zero, the first process in the event queue waiting for that range is activated.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

F_EVENT	<i>OS-9 Technical Manual</i>
F_EVENT , EV_SETXOR	<i>OS-9 Technical Manual</i>
F_EVENT , EV_SIGNL	<i>OS-9 Technical Manual</i>

Syntax

```
#include <events.h>
#include <types.h>
error_code _os_ev_signal(
    event_id      ev_id,
    int32         *value,
    u_int32        actv_flag);
```

Description

`_os_ev_signal()` signals that an event has occurred. The current event variable is updated with the signal auto-increment that was specified when the event was created. Then, the event queue is searched for the first process waiting for that event value.

ev_id	identifies the event that has occurred. (Input)
value	is a pointer to the location where <code>_os_ev_signal()</code> stores the event value prior to the signal operation. (Output)
actv_flag	specifies which process(es) to activate. (Input) <ul style="list-style-type: none">• If <code>actv_flag</code> is one, all processes in the event queue with a value in range are activated.• If <code>actv_flag</code> is zero, the first process in the event queue waiting for that range is activated.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

F\$Event
F_EVENT
F_EVENT, EV_SIGNL

OS-9 for 68K Technical Manual
OS-9 Technical Manual
OS-9 Technical Manual

Syntax

```
#include <events.h>
#include <types.h>
error_code _os_ev_tstset(
    event_id          ev_id,
    int32             *value,
    signal_code       *signal,
    u_int32           mask);
```

Description

`_os_ev_tstset()` waits for an event to occur. The event variable is ANDed with the value in `mask`. If the result is not zero, the calling process is suspended in a FIFO event queue until an `_os_ev_signal()` occurs that clears all of the bits corresponding to the set bits in the mask. Then, the bits corresponding to the set bits in the mask are set.

<code>ev_id</code>	identifies the event. (Input)
<code>value</code>	is a pointer to the location where <code>_os_ev_tstset()</code> stores the actual event value. (Output)
<code>signal</code>	is a pointer to the location where <code>_os_ev_tstset()</code> stores the signal code. (Output)
<code>mask</code>	If a process in the event queue receives a signal, it is activated even though the event has not actually occurred. specifies the activation mask. (Input)

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

F_EVENT	<i>OS-9 Technical Manual</i>
F_EVENT , EV_TSTSET	<i>OS-9 Technical Manual</i>
F_EVENT , EV_SIGNL	<i>OS-9 Technical Manual</i>

Syntax

```
#include <events.h>
#include <types.h>
error_code _os_ev_unlink(event_id ev_id);
```

Description

`_os_ev_unlink()` decrements the use count of an event. When the use count reaches zero, you can delete the event using `_os_ev_delete()`.

`ev_id` identifies the event. (Input)

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

[`_os_ev_delete\(\)`](#)
[`_os_ev_link\(\)`](#)

`F$Event`
`F_EVENT`
`F_EVENT, EV_UNLINK`
`F_EVENT, EV_DELETE`

OS-9 for 68K Technical Manual
OS-9 Technical Manual
OS-9 Technical Manual
OS-9 Technical Manual

_os_ev_wait()

Wait for Event to Occur

Syntax

```
#include <events.h>
#include <types.h>
error_code _os_ev_wait(
    event_id      ev_id,
    int32         *value,
    signal_code   *signal,
    int32         min_val,
    int32         max_val);
```

Description

`_os_ev_wait()` waits for an event to occur. It waits until an event call places the value in the range between the minimum and maximum activation values. Then, the wait auto-increment (specified at creation) is added to the event variable.

ev_id	identifies the event. (Input)
value	is a pointer to the location where <code>_os_ev_wait()</code> stores the actual event value. (Output)
signal	is a pointer to the location where <code>_os_ev_wait()</code> stores the signal code. (Output)
	If a process in the event queue receives a signal, it is activated even though the event has not actually occurred.
min_val	is the minimum activation value. (Input)
max_val	is the maximum activation value. (Input)

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

[_os_ev_signal\(\)](#)
[_os_ev_waitr\(\)](#)
[_os9_ev_waitr\(\)](#)
[F_EVENT](#)
[F_EVENT , EV_WAIT](#)
[F_EVENT , EV_SIGNL](#)

OS-9 Technical Manual

OS-9 Technical Manual

OS-9 Technical Manual

_os9_ev_wait()

Wait for Event to Occur

Syntax

```
#include <events.h>
#include <types.h>
error_code _os9_ev_wait(
    event_id      ev_id,
    int32         *value,
    int32         min_val,
    int32         max_val);
```

Description

`_os9_ev_wait()` waits for an event to occur. It waits until an event call places the value in range between the minimum activation value and the maximum activation value. Then, the wait auto-increment (specified at creation) is added to the event variable.

ev_id	specifies the event. (Input)
value	is a pointer to the location where <code>_os9_ev_wait()</code> stores the actual event value. (Output)
min_val	specifies the minimum activation value. (Input)
max_val	specifies the maximum activation value. (Input)

If a process in the event queue receives a signal, it is activated even though the event has not actually occurred.

Attributes

Operating System:	OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

F\$Event

OS-9 for 68K Technical Manual

_os_ev_waitr()

Wait for Relative Event to Occur

Syntax

```
#include <events.h>
#include <types.h>
error_code _os_ev_waitr(
    event_id      ev_id,
    int32         *value,
    signal_code   *signal,
    int32         *min_val,
    int32         *max_val);
```

Description

`_os_ev_waitr()` waits for an event to occur. It waits until an event call places the value in the range between the minimum and maximum activation values, where the integers at locations `min_val` and `max_val` are relative to the current event value. Then, the wait auto-increment (specified at creation) is added to the event variable.

ev_id	identifies the event. (Input)
value	is a pointer to the location where <code>_os_ev_waitr()</code> stores the actual event value. (Output)
signal	is a pointer to the location where <code>_os_ev_waitr()</code> stores the signal code if a signal activates the process. (Output)
min_val	<p>is a pointer to location where <code>_os_ev_waitr()</code> stores the minimum absolute activation value. (Input/Output)</p> <p>The minimum relative activation value is passed at the location pointed to by <code>min_val</code>.</p>

`max_val` is a pointer to the location where `_os_ev_waitr()` stores the maximum absolute activation value. (Input/Output)
The maximum relative activation value is passed at the location pointed to by `max_val`.

The event value is added to `min_val` and `max_val`, and the actual values are returned to the caller. If an underflow or overflow occurs on the addition, the values 0x80000000 (minimum integer) and 0x7fffffff (maximum integer) are used, respectively.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

[`_os_ev_signal\(\)`](#)
[`_os_ev_wait\(\)`](#)
[`_os9_ev_waitr\(\)`](#)
`F_EVENT`
`F_EVENT, EV_SETOR`
`F_EVENT, EV_SIGNL`

OS-9 Technical Manual
OS-9 Technical Manual
OS-9 Technical Manual

_os9_ev_waitr()

Wait for Relative Event to Occur

Syntax

```
#include <events.h>
#include <types.h>
error_code _os9_ev_waitr(
    event_id      ev_id,
    int32         *value,
    int32         *min_val,
    int32         *max_val);
```

Description

`_os9_ev_waitr()` waits for an event to occur. It waits until an event call places the value in the range between the minimum and maximum activation values, where `min_val` and `max_val` are relative to the current event value. Then, the wait auto-increment (specified at creation) is added to the event variable.

<code>ev_id</code>	specifies the event. (Input)
<code>value</code>	is a pointer to the location where <code>_os9_ev_waitr()</code> stores the actual event value. (Output)
<code>min_val</code>	is a pointer to the location where <code>_os9_ev_waitr()</code> stores the minimum absolute value. (Input/Output) The minimum relative activation value is passed at the location pointed to by <code>min_val</code> .
<code>max_val</code>	is the pointer to the location where <code>_os9_ev_waitr()</code> stores the maximum absolute value. (Input/Output) The maximum relative activation value is passed at the location pointed to by <code>max_val</code> .

The event value is added to `min_val` and `max_val` respectively, and the actual values are returned to the caller. If an underflow or overflow occurs on the addition, the values \$80000000 (minimum integer) and \$7fffffff (maximum integer) are used, respectively.

Attributes

Operating System:	OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

[_os9_ev_wait\(\)](#)
`F$Event`

OS-9 for 68K Technical Manual

_os_exec()

Start Child Process

Syntax

```
#include <process.h>
#include < dexec.h> /* used when _os_dfork(),
                     /* _os9_dfork(), or _os_dforkm()
                     /* is specified as func */

#include <types.h>
error_code _os_exec(
    u_int32      (*func)(),
    u_int32      priority,
    u_int32      pathcnt,
    void        *modname,
    char        **argv,
    char        **envp,
    u_int32      datasize, ...);
```

Description

`_os_exec()` prepares the parameter and environment list before creating a process. The function to create a new process passes information to the new process as binary data specified by a pointer and size. It is up to the forked process to interpret the data.

<code>func</code>	is a pointer to one of the following functions that create a new process. (Input)
	<ul style="list-style-type: none"> • OS-9 for 68K: <code>_os_fork()</code>, <code>_os9_dfork()</code>, or <code>_os_chain()</code>
	<ul style="list-style-type: none"> • OS-9: <code>_os_fork()</code>, <code>_os_forkm()</code>, <code>_os_chain()</code>, <code>_os_chainnm()</code>, <code>_os_dfork()</code>, or <code>_os_dforkm()</code>

<code>priority</code>	is the priority value at which the new process is to run. (Input)
<code>pathcnt</code>	Only the lower 16 bits of <code>priority</code> are used. If <code>priority</code> is zero, the new process receives the priority of the calling process.
<code>modname</code>	is the number of open paths to pass to the new process. (Input)
<code>argv</code>	Normally, <code>pathcnt</code> is three, causing the three standard paths to be passed. A value of zero passes no open paths. Only the lower 16 bits of <code>pathcnt</code> are used.
<code>envp</code>	is a pointer to a string naming the new primary module or a pointer to the module in the case of the functions ending in <code>_os_forkm()</code> , <code>_os_chainm()</code> , and <code>_os_dforkm()</code> . (Input)
<code>envp</code>	is a pointer to the parameter pointer list that the new process is to receive. (Input)
<code>envp</code>	By convention, <code>argv[0]</code> is the name of the new module. The end of the <code>argv</code> list is marked by a null pointer.
<code>envp</code>	is a pointer to the environment pointer list containing environment variables that the new process is to receive. (Input)
<code>envp</code>	The end of the <code>envp</code> list is also marked by a null pointer.

datasize	is the additional memory (in bytes) to allocate to the new process' stack memory. (Input)
	A datasize of zero indicates the default stack size required by the process.
The first seven parameters are identical for <code>_os_fork()</code> , <code>_os_forkm()</code> , <code>_os_chain()</code> , <code>_os_chainm()</code> , <code>_os_dfork()</code> , or <code>_os_dforkm()</code> . Four additional parameters may be required (add1, add2, add3, and add4) and have different meaning depending on the function specified by func:	

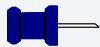
Table 2-18 `_os_exec()` Additional Parameters

Call	Parameter	Definition
<code>_os_fork()</code>	add1	Child process ID returns to the location of the add1 pointer
	add2	Type/language of the new process
	add3	*Orphan: 0 = Normal child process 1 = Process not associated with parent
<code>_os_forkm()</code>	add1	The child process ID returns to the location of the add1 pointer
	add2	*Orphan: 0 = Normal child process 1 = Process not associated with parent
<code>_os_chain()</code>	add1	Type/language of the new process
<code>_os_chainm()</code>		Does not use the extra parameters

Table 2-18 _os_exec() Additional Parameters (continued)

Call	Parameter	Definition
_os_dfork()	add1	Child process ID returns to the location of the add1 pointer
	add2	Type/language of the new process
	add3	Pointer to register stack for child process
	add4	Pointer to floating point register stack for child process
_os9_dfork()	add1	Child process ID returns to the location of the add1 pointer
	add2	Type/language of the new process
	add3	Pointer to register stack for child process
_os_dforkm()	add1	Child process ID returns to the location of the add1 pointer
	add2	Pointer to register stack for child process
	add3	Pointer to floating point register stack for child process

*On OS-9 for 68K, the orphan flag must be set to zero.



Note

If you are using `_os_exec()` when `func` indicates a chain function, you must provide a full pathlist for the module or verify that the module is already in memory.

For OS-9, a threaded process may only chain from the `main()` thread and may not chain if multiple threads are active in the process.

`_os_exec()` returns the value of `(*func)()`, which returns 0 if successful. Otherwise, it returns an error number.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

[`_os_chain\(\)`](#)
[`_os_chainm\(\)`](#)
[`_os_dfork\(\)`](#)
[`_os_dforkm\(\)`](#)
[`_os_fork\(\)`](#)
[`_os_forkm\(\)`](#)
[`os9fork\(\)`, `os9forkc\(\)`](#)
[`os9exec\(\)`](#)
[`os9kexec\(\)`](#)

Example

```
#include <process.h>
#include <dexec.h>
#include <errno.h>
#include <cglob.h>
#include <types.h>
                           /* Use the environment pointer given */
                           /* to you by the shell.          */
char *argblk[] =
{
    0,0,0,0,0,0,           /* Create an array of pointers to */
};                         /* become argv[] of child process. */
main()
{
    u_int16 prior = 0;      /* Priority of Child process */
                           /* 0 is same as parent */
    u_int16 paths = 3;      /* number of paths to inherit */
                           /* stdin, stdout, stderr */
    u_int32 edata = 0;      /* No Extra Data Space */
    process_id child_id;   /* child id number */
    char orphan = 0;        /* make this child a normal */
                           /* child not an orphan */
    u_int16 type_lang;     /* type language of child */

    type_lang = mktypelang(MT_PROGRAM, ML_OBJECT);

    argblk[0] = "dir";      /* Child process is "dir" */
    argblk[1] = "-e";       /* With a -e option */
    argblk[2] = "-a";       /* And a -a option */
    argblk[3] = 0;          /* Make sure end of options is a null ptr */

    if ((errno = _os_exec(_os_fork, prior, paths,
                          argblk[0], argblk, _environ,
                          edata, &child_id, type_lang,
                          orphan)) != SUCCESS)
        _os_exit(_errmsg(errno,
                           "Problem Creating Child %s\n",
                           argblk[0]));

/*If we get this far, the _os_exec() was successful*/
    _os_wait(0);           /* Wait for child process to die */
/* die*/
    printf("The Child's ID number was %d\n",
           child_id);
}
```

Syntax

```
include <process.h>

int os9exec(
    int             (*func)(),
    const char     *modname,
    char           **argv,
    char           **envp,
    unsigned int   stacksize,
    int            priority,
    int            pathcnt);
```

Description

`os9exec()` is a high-level fork/chain interface that prepares the parameter and environment list before a process is created.

`os9forkc()` and `chainc()` are C-level hooks to the system calls. By themselves, they provide no real parameter information. This leaves the interpretation of a raw parameter string up to the `cstart` portion of the C program. Problems can occur when handling quotes and control characters.

<code>os9exec()</code>	provides more precise control over the parameter strings and enables the environment variables to be easily inherited from the parent process. (Input)
	<code>os9exec()</code> closely resembles the UNIX <code>execve()</code> system call.

func	is a pointer to a function that creates a new process. (Input) This function is normally <code>chain()</code> , <code>chainc()</code> , <code>os9fork()</code> , or <code>os9forkc()</code> . func passes information to a new process as binary data specified by a pointer and size. It is up to the forked process to interpret the data.
modname	is a pointer to a string naming the new primary module. (Input)
argv	is a pointer to the parameter pointer list that the new process receives. (Input) By convention, <code>argv[0]</code> is the name of the new module and a null pointer marks the end of the <code>argv</code> list.
envp	is a pointer to the environment pointer list that the new process receives. (Input) A null pointer marks the end of the <code>envp</code> list. You can set <code>envp</code> to point to the environment list of the current process to allow the child to inherit that environment.
stacksize	specifies the additional memory (in bytes) to allocate to the new process' stack memory. (Input) A <code>stacksize</code> of zero indicates the default stack size the process requires.
priority	is the priority at which the new process runs. (Input) If <code>priority</code> is zero, the new process receives the priority of the calling process.
pathcnt	is the number of open paths to pass to the new process. (Input)

If `func` is `os9forkc()` or `os9fork()` and the initial attempt to create the child process fails due to EOS_PNNF (path name not found), `os9exec()` calls `modloadp()` with `modname` to load the module from disk and attempts the fork again. This action is the same as the shell handling of executable files.



Note

If you use `os9exec()` when `func` is `chain()` or `chained()`, you must either:

- Ensure that the module is in memory or in your current execution directory. (See `modlink()`, `modload()`, and `modloadp()`.)
- Provide a full pathlist to the module.

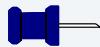
For OS-9, a threaded process may only chain from the `main()` thread and may not chain if multiple threads are active in the process.

`os9exec()` returns the value of `(*func)()`. This is the ID of the child process for `os9fork()`. `-1` is returned on error.



Note

Any program that executes a C program must do so using the `os9exec()` interface. The `cstart` module can handle parameter strings passed by `os9fork()` and `chain()`, but no environment is available and the `argv` pointer list is separated by white space.



Note

`os9exec()` is not automatically available if you are using ANSI or extended-ANSI modes. To use `os9exec()` in one of these modes, link with `sys_clib.l`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`



For More Information

The primary purpose of the `sys_clib.l` library is to provide a library for compatibility with the Microware K&R C compiler. For additional information, refer to [Appendix A: Prototyping sys_clib.l Functions](#).

See Also

`chainc()`, `chain()`
`modlink()`
`modload()`
`modloadp()`
`_os_exec()`
`os9fork()`, `os9forkc()`

Example

The following code fragment is an example call to `os9exec()`.

The `argblk` array contains pointers to the parameters in the order that the new program is to receive them. The new process receives a copy of the parameters, not the addresses of the parameters.

This example passes the current environment by naming the global variable `_environ`.

```
extern int os9forkc();
extern char **_environ;

char *argblk[] = {
    "rename",
    "-x",
    "oldname",
    "newname",
    0,
};

main ()
{
    int pid;
    if ((pid = os9exec(os9forkc,argblk[0],argblk,
        _environ,0,0,3)) > 0) wait(0);
    else printf ("can't fork %s",argblk[0]);
}
```

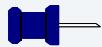
Syntax

```
#include <process.h>
#include <types.h>
error_code _os_exit(status_code status);
```

Description

`_os_exit()` allows a process to terminate itself. Its data memory area is deallocated and its primary module is unlinked. All open paths are automatically closed.

status is the status code returned to the parent process. (Input)



Note

The parent **must** perform an `_os_wait()` or an `_os_exit()` before the process descriptor is returned to free memory.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

[_os_aproc\(\)](#)
[_os_close\(\)](#)
[_os_findpd\(\)](#)
[_os_fork\(\)](#)
[_os9_retpd\(\)](#)
[_os_srtmem\(\)](#)
[_os_unlink\(\)](#)
[_os_wait\(\)](#)

[F\\$Exit](#)
[F_EXIT](#)

OS-9 for 68K Technical Manual
OS-9 Technical Manual

_os_findpd()

Find Process Descriptor

Syntax

```
#include <process.h>
#include <types.h>
error_code _os_findpd(
    process_id      proc_id,
    pr_desc         **proc_desc);
```

Description

`_os_findpd()` converts a process number to the absolute address of its process descriptor data structure.

<code>proc_id</code>	specifies the process ID. (Input)
<code>proc_desc</code>	is a pointer to the location where <code>_os_findpd()</code> stores the pointer to the process descriptor. (Output)

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

[`_os_aallocproc\(\)`](#)
[`_os_rtnprc\(\)`](#)
`F$FindPD`
`F_FINDPD`

OS-9 for 68K Technical Manual
OS-9 Technical Manual

_os9_findpd()

Find Fixed Block of Memory

Syntax

```
#include <process.h>
#include <types.h>
error_code _os9_findpd(
    u_int32      number,
    void         **address,
    void         *table);
```

Description

`_os9_findpd()` converts a table index to the absolute address of its data structure.

number	is the number of the table element. (Input)
	Only the lower 16 bits of <code>number</code> are used.
address	is a pointer to the location where <code>_os9_findpd()</code> stores the table entry pointer. (Output)
table	is a pointer to the table to use. (Input)

Attributes

Operating System:	OS-9
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

[_os9_allpd\(\)](#)
[_os9_retpd\(\)](#)
F\$FindPD

OS-9 for 68K Technical Manual

Add or Remove Device from Fast IRQ Table (System-State Only)

Syntax

```
#include <regs.h>
error_code _os_firq(
    u_int32      vector,
    u_int32      priority,
    void        *irq_entry,
    void        *statics);
```

Description

`_os_firq()` installs an IRQ service routine into or removes one from the system's fast IRQ system.

vector	specifies the vector number. (Input)
priority	specifies the priority. (Input) <code>priority</code> is reserved and must be zero.
irq_entry	is a pointer to the IRQ service routine's entry point. (Input) If <code>irq_entry</code> is zero, the IRQ service routine is deleted.
statics	is a pointer to the global static storage. (Input)

This fast IRQ system provides a faster interrupt response context than the normal IRQ vector polling scheme (provided through `_os9_irq()`).

- To place a routine into the fast IRQ system, set `irq_entry` to a non-zero value.
- To remove a routine from the fast IRQ system, set `irq_entry` to zero.

Only one `_os_firq()` routine can be active at a time per vector. An attempt to install a second routine on a vector using `_os_firq()` causes an `EOS_VCTBSY` error. If additional devices are required to be on the same vector as an `_os_firq()` device, use `_os9_irq()` to install them.

The interrupt stack provided is a small “default” IRQ stack. The service routine must ensure that this stack is not overflowed. The `init` module’s “IRQ stack” value does **not** affect this stack; the `init` module value is used to control the `_os9_irq()` polling system stack.

Attributes

Operating System:	OS-9 for 68K
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

[`_os_irq\(\)`](#)
`F$FIRQ`

OS-9 for 68K Technical Manual

_os_fmod()

Find Module Directory Entry

Syntax

```
#include <module.h>
#include <types.h>
error_code _os_fmod(
    u_int32      *type_lang,
    mod_dir      **moddir_entry,
    char         *mod_name);
```

Description

`_os_fmod()` searches the module directory for a module whose name, type, and language match the parameters. If found, a pointer to the module directory entry is returned in `moddir_entry`.

<code>type_lang</code>	specifies the type and language of the module. (Input)
<code>moddir_entry</code>	Only the lower 16 bits of <code>type_lang</code> are used.
<code>mod_name</code>	is a pointer to the location where <code>_os_fmod()</code> stores the pointer to the module directory entry. (Output)

Attributes

Operating System:	OS-9
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

Syntax

```
#include <process.h>
#include <types.h>
error_code _os_fork(
    u_int32      priority,
    u_int32      path_cnt,
    const char   *mod_name,
    const void   *params,
    u_int32      param_size,
    u_int32      mem_size,
    process_id   *proc_id,
    u_int32      type_lang,
    u_int32      orphan);
```

Description

`_os_fork()` creates a new process which becomes a child of the caller. It sets up the new process' memory, MPU registers, and standard I/O paths.

priority	specifies the priority of the new process. (Input)
	Only the lower 16 bits of <code>priority</code> are used. If <code>priority</code> is zero, the new process inherits the same priority as the calling process.
path_cnt	specifies the number of I/O paths for the child to inherit. (Input)
	Only the lower 16 bits of <code>path_cnt</code> are used.
mod_name	is the module name pointer. (Input)
params	is a pointer to the parameter block. (Input)

param_size	specifies the size of the parameter block. (Input)
mem_size	specifies any additional stack space to allocate. (Input)
proc_id	is a pointer to the location where <code>_os_fork()</code> stores the child process ID. (Output)
type_lang	specifies the desired type and language. (Input)
	Only the lower 16 bits of <code>type_lang</code> are used. If <code>type_lang</code> is zero, any module may be loaded regardless of type and language.

If the `orphan` flag is non-zero, the new process executes without a parent. If the `orphan` flag is zero, the new process is the child of the calling process. On OS-9 for 68K if the `orphan` flag is set to a non-zero value, the `EOS_PARAM` error is returned.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

[`_os_wait\(\)`](#)
[`_os_exit\(\)`](#)
[`_os_chain\(\)`](#)

`F$Fork`
`F_FORK`

OS-9 Technical Manual
OS-9 Technical Manual

Syntax

```
#include <process.h>
#include <types.h>

int os9fork(
            const char      *modname,
            int              parmsize,
            const char      *parmptr,
            int              type,
            int              lang,
            int              datasize,
            int              prior);

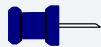
int os9forkc(
            const char      *modname,
            int              parmsize,
            const char      *parmptr,
            int              type,
            int              lang,
            int              datasize,
            int              prior,
            int              pathcnt);
```

Description

`os9forkc()` creates a process that runs concurrently with the calling process. When the new process terminates, its exit status is available to the forking (parent) process. The new process inherits the standard paths (0, 1, and 2) plus any additional paths indicated by `pathcnt`.

<code>modname</code>	is a pointer to a null-terminated module name. (Input)
----------------------	--

<code>parmsize</code>	specifies the size of the parameter list. (Input)
<code>parmprt</code>	Generally, you must set <code>parmsize</code> equal to <code>strlen(parmptr)+1</code> .
<code>type</code>	is a pointer to a null-terminated string to pass to the new process. (Input)
<code>type</code>	specifies the type of module. (Input)
<code>lang</code>	If <code>type</code> is zero, any module may be forked, regardless of type.
<code>lang</code>	specifies the module language. (Input)
<code>lang</code>	If <code>lang</code> is zero, any module may be forked, regardless of language.
<code>datasize</code>	gives extra memory to the new program. (Input)
<code>datasize</code>	If no extra memory is required, set <code>datasize</code> to zero.
<code>prior</code>	is the new priority at which to run the program. (Input)
<code>prior</code>	Indicate zero if no priority change is desired.
<code>pathcnt</code>	is the number of the parent's open paths for the new process to inherit. (Input)
<code>os9fork()</code>	If the fork is unsuccessful, -1 is returned and the appropriate error code is placed in the global variable <code>errno</code> . If the fork is successful, the process ID number of the new process is returned.
<code>os9fork()</code>	<code>os9fork()</code> is the same as <code>os9forkc()</code> without a <code>pathcnt</code> parameter.
<code>_os_exec()</code>	<code>_os_exec()</code> is the preferred method to fork C programs.



Note

Beware of forking to a system module. It only makes sense to fork to a program of type object module. `os9fork()` is a historical function and is likely to be removed in a future release. Use `os9forkc()` instead.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`



For More Information

The primary purpose of the `sys_clib.l` library is to provide a library for compatibility with the Microware K&R C compiler. For additional information, refer to [Appendix A: Prototyping sys_clib.l Functions](#).

See Also

[_os_chain\(\)](#)
[_os_exec\(\)](#)
[_os_fork\(\)](#)

`F$Chain`
`F$Fork`
`F_CHAIN`
`F_FORK`

OS-9 for 68K Technical Manual
OS-9 for 68K Technical Manual
OS-9 Technical Manual
OS-9 Technical Manual

_os_forkm()

Create New Process by Module Pointer

Syntax

```
#include <process.h>
#include <types.h>
error_code _os_forkm(
    u_int32 priority,
    u_int32 path_cnt,
    mh_com *mod_head,
    void *params,
    u_int32 param_size,
    u_int32 mem_size,
    process_id *proc_id,
    u_int32 orphan);
```

Description

`_os_forkm()` creates a new process which becomes a child of the caller. It sets up the new process' memory, MPU registers, and standard I/O paths. The new process is forked by a module pointer.

`_os_forkm()` assumes that the module pointer is the primary module pointer for the new process.

priority specifies the priority of the new process.
(Input)

Only the lower 16 bits of priority are used. If priority is zero, the new process inherits the same priority as the calling process.

`path_cnt` specifies the number of I/O paths for the child to inherit. (Input)

Only the lower 16 bits of `path_cnt` are used.

`mod_head` is the module name pointer. (Input)

params	is a pointer to the parameter block. (Input)
param_size	specifies the size of the parameter block. (Input)
mem_size	specifies any additional stack space to allocate. (Input)
proc_id	is a pointer to the location where <code>_os_forkm()</code> stores the child process ID. (Output)

If the `orphan` flag is non-zero, the new process executes without a parent. If the `orphan` flag is zero, the new process is the child of the calling process. Only the lower 16 bits of `orphan` are used.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

[`_os_fork\(\)`](#)

Syntax

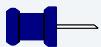
```
#include <memory.h>
#include <types.h>
error_code _os_get_blkmap(
    void *start,
    void *buffer,
    u_int32 size,
    u_int32 *min_alloc,
    u_int32 *num_segs,
    u_int32 *tot_mem,
    u_int32 *free_mem);
```

Description

`_os_get_blkmap()` copies the address and size of the system's free RAM blocks into your buffer for inspection. It also returns information concerning the free RAM as noted by the parameters. A series of structures showing the address and size of free RAM blocks is returned in your buffer.

start	is a pointer to the address to begin reporting the segments. (Input)
buffer	is a pointer to the buffer to use. (Input)
size	specifies the buffer size in bytes. (Input)
min_alloc	is a pointer to the location where <code>_os_get_blkmap()</code> stores the minimum memory allocation size for the system. (Output)
num_segs	is a pointer to the location where <code>_os_get_blkmap()</code> stores the number of memory fragments in the system. (Output)

tot_mem	is a pointer to the location where <code>_os_get_blkmap()</code> stores the total RAM found by the system at startup. (Output)
free_mem	is a pointer to the location where <code>_os_get_blkmap()</code> stores the current total free RAM available. (Output)



Note

Although `_os_get_blkmap()` returns the address and size of the system's free memory blocks, you cannot assume that these blocks are free to use. Use `_os_srqmem()` to request free memory blocks.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

[`_os_mem\(\)`](#)
[`_os_srqmem\(\)`](#)
[`F\$GBlkMap\(\)`](#)
[`F_GBLKMP`](#)

OS-9 for 68K Technical Manual
OS-9 Technical Manual

_os_getdl()

Get System I/O Device List Head Pointer

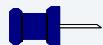
Syntax

```
#include <io.h>
#include <types.h>
error_code _os_getdl(dev_list **dev_list_ptr);
```

Description

`_os_getdl()` returns a pointer to the first entry in the system's I/O device list.

<code>dev_list_ptr</code>	is a pointer to the location where <code>_os_getdl()</code> stores the pointer to the first entry in the device list. (Output)
---------------------------	--



Note

This pointer should never be accessed directly in user state. You should use `_os_cpymem()` to get a copy of the device list entry.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_gettime()

Get System Date/Time

Syntax

```
#include <time.h>
#include <types.h>
error_code _os_gettime(
    time_t      *time,
    u_int32     *ticks);
```

Description

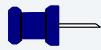
`_os_gettime()` returns the current system time in the number of seconds since a date specified in Greenwich Mean Time. To determine the specified date, refer to the applicable manual, **OS-9 for 68K Technical Manual** or **OS-9 Technical Manual**.

time	is a pointer to the location where <code>_os_gettime()</code> stores the current time. (Output)
ticks	is a pointer to the location where <code>_os_gettime()</code> stores the clock tick rate in ticks per second in the most significant word and the current tick in the least significant word. (Output)



Note

On OS-9 for 68K, the binding for this call performs conversions which allow compatibility with OS-9. If speed is a consideration, see `_os9_gettime()`.



Note

`_os_gettime()` returns a date and time of zero (with no error) if no previous call to `_os_setime()` has been made. A tick rate of zero indicates the clock is not running.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

[`_os_setime\(\)`](#)
[`_os9_gettime\(\)`](#)

`F$Time`

`F_TIME`

OS-9 for 68K Technical Manual

OS-9 Technical Manual

_os9_gettime()

Get System Date/Time

Syntax

```
#include <time.h>
#include <types.h>
error_code _os9_gettime(
    u_int32      format,
    u_int32      *time,
    u_int32      *date,
    u_int16      *day,
    u_int32      *ticks);
```

Description

`_os9_gettime()` returns the current system date and time. In the normal (Gregorian) format, `time` is expressed as 00hhmmss and `date` as yyyyymmdd. The Julian format expresses `time` as seconds since midnight and `date` as the Julian day number.

format	specifies the format of the returned date and time:.. (Input)
--------	---

Table 2-19 `_os9_gettime()` Format Values

Format	The Date and Time Are Returned In
0	Gregorian format
1	Julian format
2	Gregorian format with ticks
3	Julian format with ticks

time	is a pointer to the location where <code>_os9_gettime()</code> stores the current system time. (Output)
date	is a pointer to the location where <code>_os9_gettime()</code> stores the current system date. (Output)
day	is a pointer to the location where <code>_os9_gettime()</code> stores the day of the week, where 0 is Sunday and 6 is Saturday. (Output)
ticks	is a pointer to the location where <code>_os9_gettime()</code> stores the clock tick data. (Output)
	If <code>ticks</code> are requested, the clock tick rate in ticks per second is returned in the most significant word at the location pointed to by <code>ticks</code> . The least significant word contains the current tick.

The following figures illustrate values returned at the pointers.

Figure 2-7 Gregorian Formats

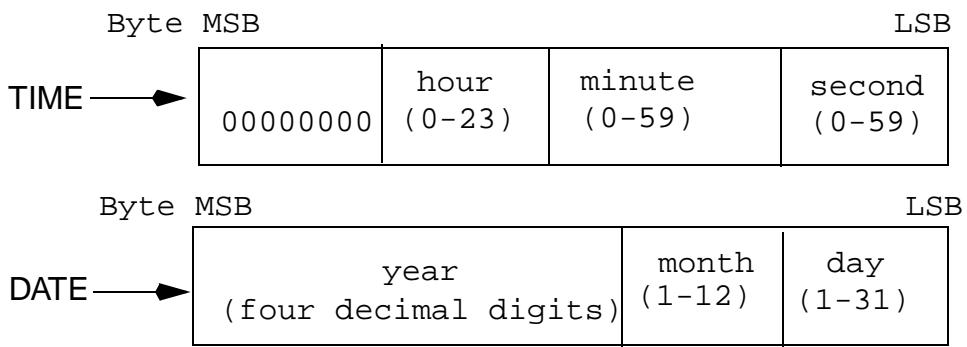
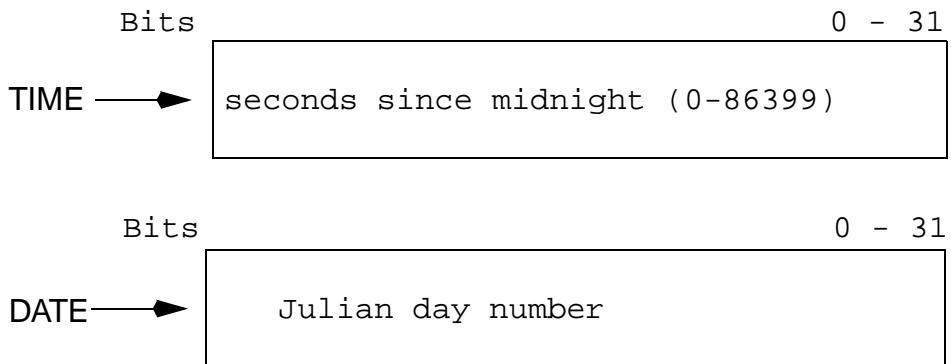


Figure 2-8 Julian Formats



Note

`_os9_gettime()` returns a date and time of zero (with no error) if no previous call to `_os9_setime()` is made after the system start-up. A tick rate of zero indicates the clock is not running.

Attributes

Operating System:	OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os9_setime()`
`_os_julian()`
`_os_gettime()`

`F$Time`

OS-9 for 68K Technical Manual

_os_get_ioproc()

Get Pointer to I/O Process Descriptor

Syntax

```
#include <io.h>
#include <types.h>
error_code _os_get_ioproc(
    process_id      proc_id,
    io_proc         **proc);
```

Description

`_os_get_ioproc()` returns a pointer to the I/O process descriptor for the specified process.

<code>proc_id</code>	specifies the process ID of the process. (Input)
<code>proc</code>	is a pointer to the location where <code>_os_get_ioproc()</code> stores the pointer to the I/O process descriptor. (Output)

Attributes

Operating System:	OS-9
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_get_mdp()

Get Current and Alternate Module Directory Pathlists

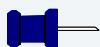
Syntax

```
#include <moddir.h>
#include <types.h>
error_code _os_get_mdp(
    char      *current,
    char      *alternate);
```

Description

`_os_get_mdp()` returns pathlists to the current and alternate module directories.

current	is a pointer to the location where <code>_os_get_mdp()</code> stores the pathlist of the current module directory. (Output)
alternate	is a pointer to the location where <code>_os_get_mdp()</code> stores the pathlist of the alternate module directory. (Output)



Note

`current` and `alternate` should point to buffers of at least 64 bytes to accommodate the directory names.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

[_os_get_moddir\(\)](#)

Get Copy of Module Directory

Syntax

For OS-9 for 68K:

```
#include <module.h>
#include <types.h>
error_code _os_get_moddir(
                           void      *buffer,
                           u_int32   *count);
```

For OS-9:

```
#include <moddir.h>
#include <types.h>
error_code _os_get_moddir(
                           void      *buffer,
                           u_int32   *count);
```

Description

`_os_get_moddir()` copies the process' current module directory into your buffer for inspection.

`buffer` is a pointer to the buffer. (Output)



For More Information

For more information on the structure of the buffer (`mod_dir`) refer to the appropriate header file. (For OS-9 for 68K, the header file is `module.h`. For OS-9, the header file is `moddir.h`.)

count

The maximum number of bytes to copy is passed at the location pointed to by count. (Input/Output)

`_os_get_moddir()` also stores the actual number of bytes copied at the location pointed to by count.

Although the module directory contains pointers to each module in the system, never access the modules directly. Use `_os_cpymem()` to copy portions of the system's address space for inspection.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_cpymem()`
`_os_move()`

Syntax

```
#include <io.h>
#include <types.h>
error_code _os_getpd(
    path_id      path,
    pd_com       **pd);
```

Description

`_os_getpd()` converts the path number to the absolute address of its path descriptor data structure.

path	specifies the path number. (Input)
pd	is a pointer to the location where <code>_os_getpd()</code> stores the address of the path descriptor. (Output)

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

[_os9_allpd\(\)](#)
[_os9_retpd\(\)](#)

_os_get_prdesc()

Get State Descriptor Copy

Syntax

```
#include <process.h>
#include <types.h>
error_code _os_get_prdesc(
    process_id      procid,
    void           *buffer,
    u_int32         *count);
```

Description

`_os_get_prdesc()` copies a process' or thread's state descriptor into a buffer for inspection. The format of the returned data is a `pr_desc` structure.

procid	is the requested process or thread ID. (Input)
buffer	is a pointer to the buffer to use. (Output)
count	The maximum number of bytes to copy is passed at the location pointed to by count. (Input/Output) <code>_os_get_prdesc()</code> also stores the actual number of bytes copied at the location pointed to by count.



Note

The preprocessor macro `_USE_V3_0_PROCDESC` must be defined to correctly use this function.

Attributes

Operating System:	OS-9
State:	User
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

[_os_get_prsrc\(\)](#)
[_os_gprdsc\(\)](#)

_os_get_prsrc()

Get Resource Descriptor Copy

Syntax

```
#include <process.h>
#include <types.h>
error_code _os_get_prsrc(
    process_id   procid,
    void         *buffer,
    u_int32      *count);
```

Description

`_os_get_prsrc()` copies a process' resource descriptor into a buffer for inspection. All the threads within a process share the same resource descriptor. The format of the returned data is a `pr_src` structure.

procid	is the requested process or thread ID. (Input)
buffer	is a pointer to the buffer to use. (Output)
count	The maximum number of bytes to copy is passed at the location pointed to by count. (Input/Output) <code>_os_get_prsrc()</code> also stores the actual number of bytes copied at the location pointed to by count.



Note

The preprocessor macro `_USE_V3_0_PROCDESC` must be defined to correctly use this function.

Attributes

Operating System:	OS-9
State:	User
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

[_os_get_prdesc\(\)](#)
[_os_gprdsc\(\)](#)

_os_get_prtbl()

Get Copy of Process Descriptor Block Table

Syntax

```
#include <process.h>
#include <types.h>
error_code _os_get_prtbl(
    void *buffer,
    u_int32 *count);
```

Description

`_os_get_prtbl()` copies the process descriptor block table into the caller's buffer for inspection.

`buffer` is a pointer to the buffer. (Output)

count The maximum number of bytes to copy is passed at the location pointed to by count. (Input/Output)

`_os_get_prtbl()` also stores the actual number of bytes copied at the location pointed to by `count`.

Although `_os_get_prtbl()` returns pointers to all process descriptors, never access the process descriptors directly. Instead, use `_os_gprdsc()` to inspect specific process descriptors.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

[_os_aallocproc\(\)](#)

[_os_gprdsc\(\)](#)

_os_getstat()

Get File/Device Status

Syntax

```
#include <sg_codes.h>
#include <types.h>
error_code _os_getstat(
    path_id      path,
    u_int32      code,
    void        *pb);
```

Description

`_os_getstat()` is a wildcard call used to handle individual device parameters that are not uniform on all devices or are highly hardware dependent.

path	is the path number. (Input)
code	is the GetStat code. (Input)
pb	is a pointer to the parameter block. (Input)

The exact operation of this call depends on the device driver and file manager associated with the path.

The mnemonics for the status codes are located in the `sg_codes.h` header file.

The status codes that are currently defined by Microware and the functions that they perform are described in this manual. The functions have an `_os_gs_` prefix.



Note

This function is available within OS-9 for 68K, where it resides in the `conv_lib.l` library. However, it can be used only with DPIO file managers and drivers, such as SoftStax.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

_os_getsys()

Examine System Global Variable

Syntax

```
#include <sysglob.h>
#include <types.h>
error_code _os_getsys(
    u_int32      offset,
    u_int32      size,
    glob_buff   *sysvar);
```

Description

`_os_getsys()` enables a process to examine a system global variable. These variables have a `d_` prefix in the system header file `sysglob.h`. Consult the files in the `DEFS` directory for a description of the system global variables.

offset	is the variable's offset in the system globals. (Input)
size	specifies the size of the variable (1, 2, or 4 bytes). (Input)
sysvar	is a pointer to the location where <code>_os_getsys()</code> stores the value of the requested system global. (Output)

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

Example

This example gets the current value of the tick counter:

```
#include <stddef.h>
if ((err = _os_getsys(offsetof(sysglobs, d_ticks),
        4, &buf)) != 0)
    exit(err);
```

See Also

[_os_setsys\(\)](#)

_os_gprdsc()

Get Process Descriptor Copy

Syntax

```
#include <process.h>
#include <types.h>
error_code _os_gprdsc(
    process_id      procid,
    void            *buffer,
    u_int32         *count);
```

Description

`_os_gprdsc()` copies a process descriptor into your buffer for inspection.

procid	is the requested process ID. (Input)
buffer	is a pointer to the buffer to use. (Output)
count	The maximum number of bytes to copy is passed at the location pointed to by count. (Input/Output)
	<code>_os_gprdsc()</code> also stores the actual number of bytes copied at the location pointed to by count.



Note

`_os_gprdsc()` only gets process descriptors in versions prior to 3.0.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

[_os_get_prtbl\(\)](#)

_os_gregorian()

Get Gregorian Date

Syntax

```
#include <time.h>
#include <types.h>
error_code _os_gregorian(
    u_int32      *time,
    u_int32      *date);
```

Description

`_os_gregorian()` converts Julian dates to Gregorian dates. Gregorian dates are considered the normal calendar dates.

`time` is a pointer to the location where the time in seconds since midnight. (Input/Output)

`_os_gregorian()` stores the time in the form 00hhmmss in the location pointed to by `time`.

`date` is a pointer to the location where the Julian date is passed. (Input/Output)

`_os_gregorian()` stores the date in the form yyyyymmdd in the location pointed to by `date`. Refer to your operating system technical manual for more information.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

[_os_julian\(\)](#)

[_os_gettime\(\)](#)

_os9_gs_cdfd()

Return File Descriptor

Syntax

```
#include <cdi.h>
#include <types.h>
error_code _os9_gs_cdfd(
    path_id      path,
    u_int32      count,
    void        *fdbuf );
```

Description

`_os9_gs_cdfd()` reads the file descriptor describing the path number. The file descriptor may be read for informational purposes only, as there are no user-changeable parameters.

path	specifies the path number. (Input)
count	is the number of bytes to copy. (Input)
	Only the lower 16 bits of count are used.
fdbuf	is a pointer to the buffer area for the file descriptor. (Output)

Attributes

Operating System:	OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

_os_gs_cpypd()

Copy Contents of Path Descriptor

Syntax

```
#include <sg_codes.h>
#include <types.h>
error_code _os_gs_cpypd(
    path_id      path,
    u_int32      *size,
    void         *path_desc);
```

Description

`_os_gs_cpypd()` copies the contents of the specified path's path descriptor to your buffer.

path	is the path number. (Input)
size	The number of bytes to copy from the path descriptor is passed at the location pointed to by <code>size</code> . (Input/Output) <code>_os_gs_cpypd()</code> also stores the actual number of bytes copied at the location pointed to by <code>size</code> .
path_desc	is a pointer to the buffer for the path descriptor data. (Output)

If the `size` value is greater than the size of the target path descriptor, the actual size of the path descriptor is returned.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_gs_cstats()

Get Cache Status Information

Syntax

```
#include <rbf.h>
#include <types.h>
error_code _os_gs_cstats(
    path_id      path,
    cachestats   *cache_inf );
```

Description

`_os_gs_cstats()` returns a copy of the current `cachestats` structure.

<code>path</code>	specifies the number of a path open on the device. (Input)
<code>cache_inf</code>	is a pointer to the location where <code>_os_gs_cstats()</code> stores the structure containing information about RBF caching. (Output)

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_gs_devnm()

Return Device Name

Syntax

```
#include <sg_codes.h>
#include <types.h>
error_code _os_gs_devnm(
    path_id      path,
    char        *name);
```

Description

`_os_gs_devnm()` returns the name of the device associated with the specified path.

`path` specifies the path number. (Input)

`name` is a pointer to the location where `_os_gs_devnm()` stores the device name. (Output)

Some networked devices using NFM may require a buffer size of 128 or 256 bytes for the device name.



Note

On OS-9, the buffer pointed to by `name` should be at least 64 bytes to accommodate a device name. On OS-9 for 68K, the buffer pointed to by `name` is a fixed 32 bytes.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

Syntax

```
#include <sg_codes.h>
#include <types.h>
error_code _os_gs_devtyp(
    path_id      path,
    u_int16      *type,
    u_int16      *class);
```

Description

`_os_gs_devtyp()` returns the type and class of the device associated with the specified path number.

path	specifies the path number. (Input)
type	is a pointer to the location where <code>_os_gs_devtyp()</code> stores the device type. (Output)
class	is a pointer to the location where <code>_os_gs_devtyp()</code> stores the device class. (Output)

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_gs_diskfree()

Return Information About RBF Disk Free Space

Syntax

```
error_code _os_gs_diskfree(path_id  path,
                           u_int32   *totblocks,
                           u_int32   *blksize,
                           u_int32   *avail,
                           u_int32   *contig);
```

Description

_os_gs_diskfree() returns information about RBF disk free space.

path	specifies the drive's path number. (Input)
totblocks	is set to the total number of blocks on the disk. (Output)
blksize	is set to the size of blocks used on the disk (256, 512, etc.) (Output)
avail	is set to the total number of free blocks on the disk. (Output)
contig	is set to the number of blocks in the largest contiguous area. (Output)

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

Syntax

```
#include <sg_codes.h>
#include <types.h>
error_code _os_gs_dopt(
    path_id      path,
    u_int32      *size,
    void         *dopts);
```

Description

`_os_gs_dopt()` copies the initial (default) device path options into your buffer. These options are used for initializing new paths to the device.

path	specifies the path number. (Input)
size	The size of the buffer is passed at the location pointed to by <code>size</code> . (Input/Output)
	<code>_os_gs_dopt()</code> also stores the actual number of bytes copied at the location pointed to by <code>size</code> .
dopts	is a pointer to a buffer for the device path options. (Output)

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_gs_dsize()

Get Number of Blocks on Media

Syntax

```
#include <rbf.h>
#include <types.h>
error_code _os_gs_dsize(
    path_id      path,
    u_int32     *totblocks,
    u_int32     *blocksize);
```

Description

`_os_gs_dsize()` gets information about the number of blocks on the media.

path	specifies the drive's path number. (Input)
totblocks	is a pointer to the location where <code>_os_gs_dsize()</code> stores the total number of blocks on the device. (Output)
blocksize	is a pointer to the location where <code>_os_gs_dsize()</code> stores the size of a disk block in bytes. (Output)

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

_os_gs_edt()

Get I/O Interface Edition Number

Syntax

```
#include <ioedt.h>
#include <types.h>
error_code _os_gs_edt(
    path_id      path,
    u_int32     *edition);
```

Description

`_os_gs_edt()` returns the I/O interface edition number of the driver and validates the compatibility of drivers and file managers.

`path` specifies the driver's path number.
(Input)

`edition` is a pointer to the location where
`_os_gs_edt()` stores the driver I/O
interface edition number. (Output)

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

[_os_gs_eof\(\)](#)

Test for End of File

Syntax

```
#include <sg_codes.h>
#include <types.h>
error_code _os_gs_eof(path_id path);
```

Description

`_os_gs_eof()` returns the `EOS_EOF` error if the current position of the file pointer associated with the specified path is at the end-of-file. SCF never returns `EOS_EOF` with this GetStat.

path specifies the driver's path number.
(Input)

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

_os_gs_fd()

Read File Descriptor Sector

Syntax

```
#include <rbf.h>
#include <types.h>
error_code _os_gs_fd(
    path_id          path,
    u_int32          size,
    fd_stats         *fdbuf );
```

Description

`_os_gs_fd()` returns a copy of the file descriptor sector for the file associated with the specified path.

path	specifies the driver's path number. (Input)
size	is the number of bytes of the file descriptor to copy. (Input)
fdbuf	is a pointer to the buffer for the file descriptor sector. (Output)

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_gs_fdaddr()

Get File Descriptor Block Address for Open File

Syntax

```
#include <rbf.h>
#include <types.h>
error_code _os_gs_fdaddr(
    path_id    path,
    u_int32   *addr);
```

Description

`_os_gs_fdaddr()` returns the file descriptor block number associated with the specified path number.

`path` specifies the driver's path number.
(Input)

`addr` is a pointer to the location where
`_os_gs_fdaddr()` stores the block
address of the file descriptor. (Output)

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

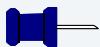
Syntax

```
#include <rbf.h>
#include <types.h>
error_code _os_gs_fdinf(
    path_id      path,
    u_int32      size,
    u_int32      fdblk,
    fd_stats     *fdbuf );
```

Description

`_os_gs_fdinf()` returns a copy of the specified file descriptor sector for the file associated with the specified file descriptor sector.

path	specifies an open path on a particular device. (Input)
size	specifies the number of bytes of the file descriptor block to copy. (Input)
fdblk	specifies the file descriptor sector number to get. (Input)
fdbuf	is a pointer to the buffer for the file descriptor block. (Output)



Note

Typically, `_os_gs_fdinf()` is used to rapidly scan a directory on a device. You do not need to specify the path number of the file for which you want the file descriptor in `path`. However, `path` must be an open path on the same device as the file. `path` is typically the path number of the directory you are currently scanning.



Note

Only super users may use this call.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

_os9_gs_free()

Return Amount of Free Space on Device

Syntax

```
#include <sg_codes.h>
#include <types.h>
error_code _os9_gs_free(
    path_id      path,
    u_int32     *free);
```

Description

`_os9_gs_free()` returns the amount of free space on a device.

`path` specifies the path number of the device.
(Input)

`free` is a pointer to the location where
`_os9_gs_free()` stores the amount of
free space. (Output)

The amount of free space is specified in
bytes.



Note

`_os9_gs_free()` is only supported by the NRF file manager.

Attributes

Operating System:	OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_gs_luopt()

Read Logical Unit Options

Syntax

```
#include <sg_codes.h>
#include <types.h>
error_code _os_gs_luopt(
    path_id      path,
    u_int32      *size,
    void         *luopts);
```

Description

`_os_gs_luopt()` copies the contents of the logical unit options for a path into the buffer pointed to by `luopts`.

`path` specifies the driver's path number.
(Input)

`size` The size of the buffer is passed at the location pointed to by `size`.
(Input/Output)

`_os_gs_luopt()` also stores the actual number of bytes copied at the location pointed to by `size`. `size` may not be less than the size of the file manager's logical unit option section.

`luopts` is a pointer to your buffer. (Output)

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_gs_parity()

Calculate Parity of File Descriptor

Syntax

```
#include <rbf.h>
#include <types.h>
error_code _os_gs_parity(
    path_id      path,
    fd_stats     *fdbuf,
    u_int32       *parity);
```

Description

`_os_gs_parity()` uses a 32-bit vertical parity to validate file descriptor structures.

path	specifies an open path to the file with which the file descriptor is associated. (Input)
fdbuf	is a pointer to the file descriptor block. (Input)
parity	is a pointer to the location where <code>_os_gs_parity()</code> stores the resulting parity. (Output)

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_gs_popt()

Read Path Descriptor Option Section

Syntax

```
#include <sg_codes.h>
#include <types.h>
error_code _os_gs_popt(
    path_id      path,
    u_int32      *size,
    void        *user_opts);
```

Description

`_os_gs_popt()` copies the option section of the path descriptor into the variable-sized area pointed to by `user_opts`. You must include `rbf.h`, `sbf.h`, and/or `scf.h` for the corresponding file managers and declare `size` according to the size of the `rbf_opts`, `sbf_opts`, or `scf_opts`.

path	specifies the driver's path number. (Input)
size	The requested number of bytes is passed at the location pointed to by <code>size</code> . (Input/Output) <code>_os_gs_popt()</code> also stores the actual number of bytes copied at the location pointed to by <code>size</code> .
user_opts	is a pointer to your buffer. (Output)



Note

On OS-9 for 68K, 128 bytes are always allocated by the binding, although only the specified number of bytes are passed back to you.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

_os_gs_pos()

Get Current File Position

Syntax

```
#include <sg_codes.h>
#include <types.h>
error_code _os_gs_pos(
    path_id      path,
    u_int32     *position);
```

Description

`_os_gs_pos()` returns the current position of the file pointer associated with the specified path.

`path` specifies the driver's path number.
(Input)

`position` is a pointer to the location where
`_os_gs_pos()` stores the file position.
(Output)

- If the device is opened in block mode (`S_IBLKMODE`), the `position` parameter returns position in terms of blocks.
- If the device is opened in block-mode (`S_IBLKMODE`), the `position` parameter to `_os_gs_pos()` returns in terms of blocks (rather than bytes, which is the behavior when in non-block mode).

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

_os9_gspump()

Return Data for Specified Process' Memory Map

Syntax

```
#include <process.h>
#include <types.h>
error_code _os9_gspump(
    process_id      proc_id,
    u_int32         size,
    void            *buffer);
```

Description

`_os9_gspump()` returns data about a specified process' memory map for debugging purposes.

<code>proc_id</code>	specifies the process ID. (Input)
<code>size</code>	specifies the size of the buffer. (Input)
<code>buffer</code>	is a pointer to the storage buffer. (Output)

One word in this format is returned for each memory block in the system. This information is taken from the process' SSM data structure. If the specified address space is not large enough, only the information that fits in the area is returned.

Attributes

Operating System:	OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`F$GSPUMP`

OS-9 for 68K Technical Manual

Syntax

```
#include <sg_codes.h>
#include <types.h>
error_code _os_gs_ready(
    path_id      path,
    u_int32     *incount);
```

Description

`_os_gs_ready()` checks for data available to be read on the specified path. RBF devices do not return the `EOS_NOTRDY` error because there is always information available to be read on RBF devices. `_os_gs_ready()` returns the number of bytes left in `incount`.

`path` specifies the driver's path number.
(Input)

`incount` is a pointer to the location where
`_os_gs_ready()` stores the number of
characters available to be read. (Output)

If the device is opened in block mode (`S_IBLKMODE`), `_os_gs_ready` returns the number of blocks available.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_gs_size()

Return Current File Size

Syntax

```
#include <sg_codes.h>
#include <types.h>
error_code _os_gs_size(
    path_id      path,
    u_int32     *size);
```

Description

`_os_gs_size()` returns the current size of the file associated with the specified path.

path	specifies the driver's path number. (Input)
size	is a pointer to the location where <code>_os_gs_size()</code> stores the file size. (Output)

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

Syntax

```
#include <process.h>
#include <types.h>
error_code _os_id(
    process_id      *proc_id,
    u_int16         *priority,
    u_int16         *age,
    int32           *schedule,
    u_int16         *group,
    u_int16         *user);
```

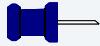
Description

`_os_id()` returns the caller's process ID number, current process priority and age, scheduling constant, and owner ID.

proc_id	is a pointer to the location where <code>_os_id()</code> stores the current process ID number. (Output)
priority	is a pointer to the location where <code>_os_id()</code> stores the priority of the current process. (Output)
age	is a pointer to the location where <code>_os_id()</code> stores the age of the current process. (Output)
schedule	is a pointer to the location where <code>_os_id()</code> stores the scheduling constant of the current process. (Output)
group	is a pointer to the location where <code>_os_id()</code> stores the group number of the current process. (Output)

user

is a pointer to the location where
`_os_id()` stores the user number of the
current process. (Output)



Note

On OS-9 for 68K, the binding for this call, performs conversions which allow compatibility with OS-9. If speed is a consideration, see `_os9_id()`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System (OS-9); User (OS-9 for 68K)
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

[`_os9_id\(\)`](#)

Syntax

```
#include <process.h>
#include <types.h>
error_code _os9_id(
    process_id      *proc_id,
    u_int16          *priority,
    u_int16          *group,
    u_int16          *user);
```

Description

`_os9_id()` returns the caller's process ID number, group and user ID, and current process priority.

proc_id	is a pointer to the location where <code>_os9_id()</code> stores the process ID. (Output)
priority	is a pointer to the location where <code>_os9_id()</code> stores the current process priority. (Output)
group	is a pointer to the location where <code>_os9_id()</code> stores the group ID. (Output)
user	is a pointer to the location where <code>_os9_id()</code> stores the user ID. (Output)

Attributes

Operating System:	OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

_os_initdata()

Initialize Static Storage from Module

Syntax

```
#include <module.h>
#include <types.h>
error_code _os_initdata(
    mh_com      *mod_head,
    void        *statics);
```

Description

`_os_initdata()` clears the uninitialized data area, copies the module header's initialized data to the specified data area, and clears the remote data area (if it exists). It then adjusts the code and data offsets.

mod_head	is a pointer to the module header. (Input)
statics	is a pointer to the data area. (Input)

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

Syntax

```
#include <signal.h>
#include <types.h>
#include <cglob.h>
error_code _os_intercept(
                        void      (*func)(),
                        void      *data_ptr);
```

Description

`_os_intercept()` installs a signal intercept routine. The intercept routine is entered asynchronously because a signal may be sent at any time, similar to an interrupt. The signal code is passed as a parameter to the intercept routine.

`func` is a pointer to the intercept routine's address. (Input)

`data_ptr` is a pointer to the intercept routine's global storage. (Input)

It usually contains the address of the program's data area.

`_os_rte()` must be used to exit from the intercept routine (`func`). If the routine fails to use `_os_rte()` at every possible exit, unpredictable results occur.



Note

The intercept routine should be short and fast, such as setting a flag in the process' data area. Avoid complicated system calls (such as I/O).

Signal handlers should be careful about the following:

- Calling any function in the library that may be working with static storage structures.

- Calling any function that is not re-entrant.
- Modifying any static storage structure other than a variable of type `volatile sig_atomic_t`

In OS-9 for 68K, `data_ptr` **must** be `_glob_data` to provide compatibility between OS-9 for 68K and OS-9. Also, each time the intercept routine is called, the state of the processor (such as its registers) is pushed on to the user stack.

`signal()`, `intercept()`, and the POSIX threading library use `_os_intercept()` internally; do not use `_os_intercept()` in combination with any of these.

Only the global data pointer is set up for the signal handler, the proper constant pointer, if applicable, is not automatically set up.

`intercept()` and `signal()` properly initialize both the data and constant pointers before calling the signal handling function.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe (see Note)
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`intercept()`
`_os_rte()`
`_os_send()`
`signal()`

`F$ICPT`
`F_ICPT`

OS-9 for 68K Technical Manual
OS-9 Technical Manual

_os_ioconfig()

Configure an Element of Process/System I/O

Syntax

```
#include <modes.h>
error_code _os_ioconfig(
    u_int32      code,
    void        *param);
```

Description

`_os_ioconfig()` is a wildcard call used to configure elements of the I/O subsystem which may or may not be associated with an existing path. It is intended to be used to dynamically reconfigure system I/O resources at runtime. The target I/O resources may be system-wide resources or they may be process- or path-specific, depending on the nature of the configuration call being made.

code	is the IO configuration command code. (Input)
	Refer to the OS-9 Technical Manual for a list of valid command codes for your version of the operating system.
param	is a pointer to any additional parameters required by the specified configuration function. (Input)

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

See Also

[_os_config\(\)](#)

_os_iodel()

Check for Use of I/O Module

Syntax

```
#include <module.h>
#include <types.h>
error_code _os_iodel(mh_com *mod_head);
```

Description

`_os_iodel()` is executed when the kernel unlinks a file manager, device driver, or device descriptor module. It is used to determine if the I/O system is still using the module.

`mod_head` is a pointer to the module header. (Input)

Attributes

Operating System:	OS-9
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

Syntax

```
#include <process.h>
#include <types.h>
error_code _os_ioexit(
    process_id      proc_id,
    u_int32         path_cnt);
```

Description

`_os_ioexit()` is executed when the kernel terminates or chains to a process.

`proc_id` specifies the process ID. (Input)

`path_cnt` specifies the number of I/O paths. (Input)

If the most significant bit of `path_cnt` is cleared, the process' default data and execution directory paths and all other open paths in the path translation table are closed. The I/O process descriptor is also deallocated.

If the most significant bit of `path_cnt` is set, the remaining bits specify the number of paths to leave open. The default directory paths are not closed, and the I/O process descriptor is not de-allocated.

Attributes

Operating System:	OS-9
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_iofork()

Set Up I/O for New Process

Syntax

```
#include <process.h>
#include <types.h>
error_code _os_iofork(
    process_id      par_proc_id,
    process_id      new_proc_id,
    u_int32         path_cnt);
```

Description

`_os_iofork()` is executed when the kernel creates a new process. `_os_iofork()` creates an I/O process descriptor for the new process.

`par_proc_id` is the parent's process ID. (Input)

`new_proc_id` is the process ID of the new process. (Input)

`path_cnt` is the number of I/O paths. (Input)

Attributes

Operating System:	OS-9
State:	User
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

Syntax

```
#include <process.h>
#include <types.h>
error_code _os9_iqueue(process_id proc_id);
```

Description

`_os9_iqueue()` links the calling process into the I/O queue and performs an untimed sleep.

`proc_id` specifies the process ID of the process currently in control of the device. (Input)

Attributes

Operating System:	OS-9 for 68K
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_findpd()`
`_os_send()`
`_os_sleep()`
`_os9_sleep()`
F\$IOQu

OS-9 for 68K Technical Manual

Add or Remove Device from IRQ Table

Syntax

```
#include <regs.h>
#include <types.h>
error_code _os_irq(
    u_int32          vector,
    u_int32          priority,
    const void      *irq_entry,
    void            *statics);
```

Description

`_os_irq()` installs an IRQ service routine into the system polling table.

<code>vector</code>	specifies the vector number. (Input) Only the lower 16 bits of <code>vector</code> are used.
<code>priority</code>	specifies the priority. (Input) Only the lower 16 bits of <code>priority</code> are used. <ul style="list-style-type: none"> • If <code>priority</code> is 0, the routine is polled first on the vector. • If <code>priority</code> is 1, the routine is polled first and no other device can have a priority of 1 on the vector. • If <code>priority</code> is 255, the routine is polled last on the vector.
<code>irq_entry</code>	is a pointer to the IRQ service routine entry point. (Input) If <code>irq_entry</code> is 0, the call deletes the IRQ service routine.

`statics`

is a pointer to the global static storage.
`statics` must be unique to the device.
(Input)



WARNING

The called `irq` function must not be compiled with `stackcheck` because the stack is out of range at this time.



For More Information

Refer to the *OS-9 Technical I/O Manual* for more information.

Attributes

Operating System:

OS-9

State:

System

Threads:

Safe

Re-entrant:

Yes

Library

`os_lib.l`

See Also

`F_IRQ`

OS-9 Technical Manual

Add or Remove Device from IRQ Table

Syntax

```
#include <regs.h>
#include <types.h>
error_code _os9_irq(
    u_int32      vector,
    u_int32      priority,
    const void   *irq_entry,
    const void   *statics,
    const void   *port);
```

Description

`_os9_irq()` installs an IRQ service routine into the system polling table.

<code>vector</code>	specifies the vector number. (Input) Only the lower 16 bits of <code>vector</code> are used.
<code>priority</code>	specifies the priority. (Input) Only the lower 16 bits of <code>priority</code> are used. <ul style="list-style-type: none"> • If <code>priority</code> is 0, the routine is polled first on the vector. • If <code>priority</code> is 255, the routine is polled last on the vector.
<code>irq_entry</code>	is a pointer to the IRQ service routine entry point. (Input) If <code>irq_entry</code> is 0, the call deletes the IRQ service routine.
<code>statics</code>	is a pointer to the global static storage. (Input) <code>statics</code> must be unique to the device.



port

is the port address. (Input)

For More Information

Refer to the ***OS-9 for 68K Technical I/O Manual*** for more information.

Attributes

Operating System:	OS-9 for 68K
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

F\$IRQ

OS-9 for 68K Technical Manual

_os_julian()

Get Julian Date

Syntax

```
#include <time.h>
#include <types.h>
error_code _os_julian(
    u_int32          *time,
    u_int32          *date);
```

Description

`_os_julian()` converts Gregorian dates to Julian dates.

`time` is a pointer to the location where the time in the form `00hhmmss` is passed.
(Input/Output)

`_os_julian()` stores the time in seconds since midnight in the location pointed to by `time`.

`date` is a pointer to the location where the date in the form `yyyyymmdd` is passed.
(Input/Output)

`_os_julian()` stores the Julian date in the location pointed to by `date`.



For More Information

Refer to your operating system technical manual for more information.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library

os_lib.l

os9kexec()

OS-9 Create Process

Syntax

```
#include <process.h>
#include <types.h>
int os9kexec(
    u_int32 (*func)(),
    priority,
    pathcnt,
    *modname,
    **argv,
    **envp,
    datasize,
    *extra1,
    *extra2,
    *extra3,
    *extra4);
```

Description

`os9kexec()` is a high-level fork/chain interface that prepares the parameter and environment list before a process is created.

`os9kexec()` closely resembles the UNIX `execve()` system call.

`func` is a pointer to one of the following functions that create a new process:
`_os_fork()`, `_os_forkm()`,
`_os_chain()`, `_os_chainm()`,
`_os_dfork()`, and `_os_dforkm()`
(Input)

func passes information to a new process as binary data specified by a pointer and size. It is up to the forked process to interpret the data.

priority	is the priority at which the new process is to run. (Input) If priority is zero, the new process receives the priority of the calling process.
pathcnt	is the number of open paths to pass to the new process. (Input) Normally, pathcnt is 3, causing the 3 standard paths to be passed. A pathcnt of zero passes no open paths.
modname	is a pointer to a string naming the new primary module or to a module in the case of the functions ending in m (_os_forkm(), _os_chainm(), and _os_dforkm()). (Input)
argv	is a pointer to the parameter pointer list that the new process receives. (Input) By convention, argv[0] is the name of the new module. A null pointer marks the end of the argv list.
envp	is a pointer to the environment pointer list that the new process receives. (Input) It points to environment variables. A null pointer marks the end of the envp list.
datasize	specifies the additional memory (in bytes) to allocate to the new process' stack memory.(Input) If datasize is zero, the default stack size required by the process is used.

The first 7 parameters are identical for _os_fork(), _os_forkm(), _os_chain(), _os_chainm(), _os_dfork(), and _os_dforkm().

The remaining four parameters have different meanings, depending on the function specified by `func`:

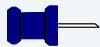
Table 2-20 os9kexec() Additional Parameters

Function	Parameter	Definition
<code>_os_fork()</code>	<code>extra1</code> (Input/Output)	The child process ID returns to where <code>extra1</code> points
	<code>extra2</code> (Input)	Type/language of the new process
	<code>extra3</code> (Input)	Orphan: 0 = Normal child process 1 = Process not associated with parent
<code>_os_forkm()</code>	<code>extra1</code> (Input/Output)	The child process ID returns to where <code>extra1</code> points
	<code>extra2</code> (Input)	Orphan: 0 = Normal child process 1 = Process not associated with parent
<code>_os_chain()</code>	<code>extra1</code> (Input/Output)	Type/language of the new process
<code>_os_chainm()</code>		Does not use the extra parameters

Table 2-20 os9kexec() Additional Parameters (continued)

Function	Parameter	Definition
<code>_os_dfork()</code>	<code>extra1</code> (Input/Output)	The child process ID returns to where <code>extra1</code> points
	<code>extra2</code> (Input)	Type/language of the new process
	<code>extra3</code> (Input)	Pointer to register stack for child process
	<code>extra4</code> (Input)	Pointer to floating point register stack for child process
<code>_os_dforkm()</code>	<code>extra1</code> (Input/Output)	The child process ID returns to where <code>extra1</code> points
	<code>extra2</code> (Input)	Pointer to register stack for child process
	<code>extra3</code> (Input)	Pointer to floating point register stack for child process

If `func` indicates a fork function and the initial attempt to create the child process fails due to EOS_PNNF (path name not found), `os9kexec()` calls `modloadp()` with `modname` to load the module from disk and attempts the fork again. This action is the same as the shell handling of executable files.



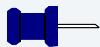
Note

If you use `os9kexec()` when `func` is `_os_chain()` or `_os_chainc()`, you must either:

- Ensure that the module is in memory or in your current execution directory (see `modlink()`, `modload()`, and `modloadp()`).
- Provide a full pathlist to the module.

A threaded process may only chain from the `main()` thread and may not chain if multiple threads are active in the process.

`os9kexec()` returns the value of `(*func)()`, which returns 0 if successful; otherwise, it returns an error number.



Note

Any program that executes a C program should do so using the `os9kexec()` interface. The `cstart` module can handle parameter strings passed by `os9fork()` and `chain()`, but no environment is available and the `argv` pointer list is separated by white space.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

For More Information

The primary purpose of the `sys_clib.l` library is to provide a library for compatibility with the Microware K&R C compiler. For additional information, refer to [Appendix A: Prototyping sys_clib.l Functions](#).

See Also

[`_os_exec\(\)`](#)
[`_os_fork\(\)`](#)
[`_os_forkm\(\)`](#)
[`_os_chain\(\)`](#)
[`_os_chainm\(\)`](#)
[`_os_dfork\(\)`](#)
[`_os_dforkm\(\)`](#)

Example

```
#include <module.h>
#include <types.h>
#include <const.h>
extern int _os_fork();
extern char **_environ;

char * argblk[] = {
    "rename",
    "oldname",
    "newname",
    "-x",
    0,
}

main()
{
    u_int16 type_lang = mktypelang (MT_PROGRAM,
                                    ML_OBJECT);
    process_id pid;

    If ((errno = os9keexec (_os_fork,
        0,           * no change in priority
        3,           * pass stdin, stdout, and stderr
        angblk[0],   * pointer to module name
        argblk,      * the argument list
        _environ,    * environment list for new process
        0,           * no extra memory
        &pid         * pointer to the process ID
        type_lang,
        0))        * normal child process

        != SUCCESS)
    printf ("can't fork %s\n", argblk[0]);
    else
        wait(0)
}
```

Syntax

```
#include <module.h>
#include <types.h>
error_code _os_link(
    char      **mod_name,
    mh_com    **mod_head,
    void      **mod_exec,
    u_int16   *type_lang,
    u_int16   *attr_rev);
```

Description

`_os_link()` searches the current module directory (and an alternate module directory in OS-9) for a module whose name, type, and language match the parameters.

The module name is passed at the location pointed to by `mod_name`. On OS-9, if `mod_name` is an explicit module directory pathlist (for example, `/usr/pers1/prog`), `_os_link()` also stores the pointer to the module that was successfully linked (for example, `prog`) at the location pointed to by `mod_name` (input/output).

<code>mod_head</code>	is a pointer to the location where <code>_os_link()</code> stores the address of the module's header. (Output)
<code>mod_exec</code>	is a pointer to the location where <code>_os_link()</code> stores the absolute address of the module's execution entry point. (Output)

`type_lang`

The type and language of the module are passed at the location pointed to by `type_lang`. (Input/Output)

`_os_link()` also stores the actual type and language at the location pointed to by `type_lang`. If `type_lang` is zero (specifying any), any module can be linked to regardless of the type and language.

`attr_rev`

is a pointer to the location where `_os_link()` stores the attribute and revision level of the module. (Output)

Attributes

Operating System:
State:
Threads:
Re-entrant:

OS-9 and OS-9 for 68K
User and System
Safe
Yes

Library

`os_lib.l`

See Also

`_os_load()`
`_os_unlink()`
`_os_unload()`

F\$Link

F_LINK

OS-9 for 68K Technical Manual

OS-9 Technical Manual

_os_linkm()

Link to Memory Module by Module Pointer

Syntax

```
#include <module.h>
#include <types.h>
error_code _os_linkm(
    mh_com      *mod_head,
    void        **mod_exec,
    u_int16     *type_lang,
    u_int16     *attr_rev);
```

Description

`_os_linkm()` causes OS-9 to link to the module specified by `mod_head`.

<code>mod_head</code>	is a pointer to the module. (Input)
<code>mod_exec</code>	is a pointer to the location where <code>_os_linkm()</code> stores the absolute address of the module's execution entry point. (Output)
<code>attr_rev</code>	is a pointer to the location where <code>_os_linkm()</code> stores the attribute and revision level of the module. (Output)

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

[_os_load\(\)](#)
[_os_unlink\(\)](#)
[_os_unload\(\)](#)

Syntax

```
#include <module.h>
#include <modes.h>
#include <memory.h>
#include <types.h>
error_code _os_load(
    const char      *mod_name,
    mh_com          **mod_head,
    void             **mod_exec,
    u_int32          mode,
    u_int16          *type_lang,
    u_int16          *attr_rev,
    u_int32          color);
```

Description

`_os_load()` opens a file specified by the pathlist. It reads one or more memory modules from the file into memory until an error or end of file is reached. Then, it closes the file. Modules are loaded into the highest physical memory available of the specified `color` type.

mod_name	is a pointer to the module name. (Input)
mod_head	is a pointer to the location where <code>_os_load()</code> stores the pointer to the module. (Output)
mod_exec	is a pointer to the location where <code>_os_load()</code> stores the pointer to the module execution entry point. (Output)
mode	specifies the access mode. (Input) Only the lower 16 bits of <code>mode</code> are used.



Note

All available access modes are defined in the following location:
MWOS\<OS>\SRC\DEFS\modes.h



For More Information

For a list of all available access modes, refer to the section, “Access Modes and Permissions,” in Chapter 3 of the ***OS-9 Technical Manual***.

`type_lang`

The type and language of the module are passed at the location pointed to by `type_lang`. (Input/Output)

`_os_load()` also stores the actual type and language at the location pointed to by `type_lang`. If `type_lang` is zero, any module can be linked to regardless of the type and language.

`attr_rev`

is a pointer to the location where `_os_load()` stores the attribute and revision level of the module. (Output)

When a module is loaded, its name is added to the system module directory, and the first module read is linked. The returned parameters are the same as those returned by a link call and apply only to the first module loaded.



Note

`_os_load()` does not work on SCF devices.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

F\$Load

F_LOAD

OS-9 for 68K Technical Manual

OS-9 Technical Manual

_os_loadp()

Load and Link to Memory Module Using PATH

Syntax

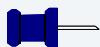
```
#include <module.h>
#include <modes.h>
#include <types.h>
error_code _os_loadp(
    const char      *name,
    u_int32         mode,
    char            *nameptr,
    mh_com          **mod_head);
```

Description

`_os_loadp()` opens a file specified by the pathlist. It reads one or more memory modules from the file into memory until an error or end of file is reached; then it closes the file.

The `PATH` environment variable determines the alternate directories to search for the named file.

name	is a pointer to the file name. (Input)
mode	specifies the access mode. (Input) Only the lower 16 bits of <code>mode</code> are used.



Note

All available access modes are defined in the following location:
`MWOS\<OS>\SRC\DEFS\modes.h`



For More Information

For a list of all available access modes, refer to the section, “Access Modes and Permissions,” in Chapter 3 of the **OS-9 Technical Manual**.

nameptr

is a pointer to a buffer for the pathlist of the successfully loaded file. (Input)

Use NULL to indicate that no complete pathlist is required.

mod_head

is a pointer to the location where `_os_loadp()` stores the pointer to the module. (Output)



For More Information

Refer to your operating system technical manual for information regarding the PATH environment variable.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_load()`
`_os_exec()`

_os_mkdir()

Make New Directory

Syntax

```
#include <modes.h>
#include <types.h>
error_code _os_mkdir(
    const char      *name,
    u_int32         mode,
    u_int32         perm, ...);
```

Description

`_os_mkdir()` creates and initializes a new directory as specified by the pathlist. `_os_mkdir()` is the only way to create a new directory file. `_os_mkdir()` fails on non-multifile devices.

name	is a pointer to the pathlist. (Input)
mode	specifies the access mode. (Input)
	Only the lower 16 bits of mode are used.
perm	specifies the access permissions. (Input)



Note

All available access modes and permissions are defined in the following location: MWOS\<OS>\SRC\DEFS\modes.h



For More Information

For a list of all available access modes and permissions, refer to the section, “Access Modes and Permissions,” in Chapter 3 of the **OS-9 Technical Manual**.

A variable parameter may be included to specify the initial allocation size.

`_os_mkdir()` does not return a path number. You should use `_os_open()` to open a directory. The new directory automatically has its directory bit set in the access permission attributes. The remaining attributes are specified by the bytes passed in `mode` and `perm`.



For More Information

Refer to your operating system technical manual for the available modes and attributes.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_makmdir()

Create New Module Directory

Syntax

```
#include <moddir.h>
#include <types.h>
error_code _os_makmdir(
                      const char      *name,
                      u_int32         perm);
```

Description

`_os_makmdir()` creates a new module directory. The name of the new module directory is relative to the current module directory.

name	is a pointer to the name of the new module directory. (Input)
perm	specifies the access permissions for the new module directory. (Input) Only the lower 16 bits of <code>perm</code> are used.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

Syntax

```
#include <memory.h>
#include <types.h>
error_code _os_mem(
    u_int32      *size,
    void        **mem_ptr);
```

Description

`_os_mem()` contracts or expands the process' data memory area. The requested size is rounded up to an even memory allocation block. Additional memory is allocated contiguously upward (towards higher addresses) or deallocated downward from the old highest address.

`size`

The desired memory size in bytes is passed at the location pointed to by `size`. (Input/Output)

`_os_mem()` also stores the actual size of the memory at the location pointed to by `size`. If the integer pointed to by `size` is zero, `_os_mem()` treats the call as a request for information and returns the current upper bound in `mem_ptr` and the amount of free memory in `size`.

`mem_ptr`

is a pointer to the location where `_os_mem()` stores the pointer to the new end of data segment plus 1. (Output)

Attributes

Operating System:

OS-9 and OS-9 for 68K

State:

User and System

Threads:

Safe

Re-entrant:

Yes

Library

os_lib.l

See Also

F\$Mem

F_MEM

OS-9 for 68K Technical Manual

OS-9 Technical Manual

_os_mkmodule()

Create Data Module of Specified Color Type

Syntax

```
#include <module.h>
#include <memory.h>
#include <types.h>
error_code _os_mkmodule(
                        const char      *mod_name,
                        u_int32          size,
                        u_int16          *attr_rev,
                        u_int16          *type_lang,
                        u_int32          perm,
                        void             **mod_exec,
                        mh_com           **mod_head,
                        u_int32          color);
```

Description

`_os_mkmodule()` creates a memory module with the specified name and attributes in the specified memory. The module is created with a valid CRC and entered into the system module directory. You can specify the type/language of the module and color of memory in which to make the module.

mod_name	is a pointer to the name of the data module. (Input)
size	specifies the size of the data portion to clear. (Input)
attr_rev	is a pointer to the location where <code>_os_mkmodule()</code> stores the module's attribute and revision. (Output)

type_lang	The type and language for the module are passed at the location pointed to by type_lang. (Input/Output) _os_mkmodule() also stores the actual type and language at the location pointed to by type_lang.
perm	specifies the access permissions for the module. (Input) Only the lower 16 bits of perm are used.
mod_exec	is a pointer to the location where _os_mkmodule() stores the pointer to the module data. (Output)
mod_head	is a pointer to the location where _os_mkmodule() stores the pointer to the module header. (Output)
color	is the memory color type: SYSRAM, VIDEO1, or VIDEO2. (Input) If color is zero, no memory type is specified. Consequently, the module is made in whatever memory the system allocates.



Note

The created module always begins at offset \$34 within the module. This implies that this call cannot conveniently make program modules, trap handlers, file managers, device drivers, and device descriptors.

Attributes

Operating System:	OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

[_os_datmod\(\)](#)
[_os_setcrc\(\)](#)
[_os_move\(\)](#)

_os_modaddr()

Find Module Given Pointer

Syntax

```
#include <module.h>
#include <types.h>
error_code _os_modaddr(
    void      *mem_ptr,
    mh_com   **mod_head);
```

Description

`_os_modaddr()` locates a module given a pointer to any position within the module and returns a pointer to the module's header.

mem_ptr	is a pointer to any position within the module. (Input)
mod_head	is a pointer to the location where <code>_os_modaddr()</code> stores the pointer to the module header. (Output)

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

Syntax

```
#include <memory.h>
#include <types.h>
error_code _os_move(
    void      *from,
    void      *to,
    u_int32   count);
```

Description

`_os_move()` is a fast “block-move” subroutine which copies data bytes from one address space to another, usually from system to user or vice versa. This call allows the source and destination buffers to overlap.

from	is a pointer to the source data. (Input)
to	is a pointer to the destination data. (Output)
count	is the byte count to copy. (Input)

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

_os_nproc()

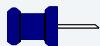
Start Next Process

Syntax

```
#include <process.h>
#include <types.h>
error_code _os_nproc(void);
```

Description

`_os_nproc()` removes the next process from the active process queue and initiates its execution. `_os_nproc()` does not return to the caller.



Note

The process calling `_os_nproc()` should already be in one of the system's process queues. If not, the process becomes unknown to the system. This occurs even though the process descriptor still exists and is printed out by a `procs` command.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

[`_os_aproc\(\)`](#)

Syntax

```
#include <modes.h>
#include <types.h>
error_code _os_open(
    const char    *name,
    u_int32       mode,
    path_id       *path);
```

Description

`_os_open()` opens a path to an existing file or device as specified by the pathlist. `_os_open()` returns a path number which is used in subsequent service requests to identify the path. If the file does not exist, an error is returned.

name	is a pointer to the path name of the existing file or device. (Input)
mode	specifies which subsequent read and/or write operations are permitted. (Input) Only the lower 16 bits of mode are used.



Note

All available access modes are defined in the following location:

MWOS\<OS>\SRC\DEFS\modes.h

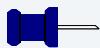


For More Information

For a list of all available access modes, refer to the section, “Access Modes and Permissions,” in Chapter 3 of the ***OS-9 Technical Manual***.

path

is a pointer to the location where `_os_open()` stores the resulting path number. (Output)



Note

A non-directory file may be opened with no mode bits set. This allows you to examine characteristics such as the attributes and size by the `_os_getstat()` system requests, but does not permit any actual I/O on the path.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_attach()`
`_os_close()`
`_os_create()`

`I$Open`
`I_OPEN`

OS-9 for 68K Technical Manual
OS-9 Technical Manual

Syntax

```
#include <process.h>
#include <types.h>
error_code _os9_panic(
    u_int32      panic_code);
```

Description

`_os9_panic()` is called when the kernel detects a disastrous, but not necessarily fatal, system condition. Ordinarily, `_os9_panic()` is undefined and the system dies.

`panic_code` specifies the panic code to output.
(Input)

`_os9_panic()` is called only when the kernel believes there are no processes remaining to be executed. Although the system is probably dead at this point, it may not be. Interrupt service routines or system-state alarms could cause the system to become active.

Attributes

Operating System:	OS-9 for 68K
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

[_os9_setsvc\(\)](#)
`F$Panic`

OS-9 for 68K Technical Manual

_os_permit()

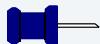
Allow Access to Memory Block

Syntax

```
#include <process.h>
#include <types.h>
#include <modes.h>
error_code _os_permit(
                      void          *mem_ptr,
                      u_int32       size,
                      u_int32       perm,
                      process_id   pid);
```

Description

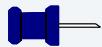
`_os_permit()` allows a process to access a block of memory. The `_os_permit()` service routine must update SSM (System Security Module) data structures to show that a process may access the specified memory in the requested mode.



Note

Only super-group users may use the `_os_permit()` function.

mem_ptr	is a pointer to the beginning of the memory area to grant access permissions. (Input)
size	is the size of the memory area. (Input)
perm	is the permissions to grant. (Input) Only the lower 16 bits of <code>perm</code> are used.
pid	is the process identifier of the target process (not used for OS-9 for 68K). (Input)



Note

`_os_permit()` only performs a useful function on MMU/SSM systems. On non-MMU/SSM systems, `_os_permit()` simply returns without allowing access to the memory block.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`F$Permit`
`F_PERMIT`

OS-9 for 68K Technical Manual
OS-9 Technical Manual

_os_perr()

Print Error Message

Syntax

```
#include <errno.h>
#include <types.h>
error_code _os_perr(
    path_id      path,
    u_int32      errnum);
```

Description

`_os_perr()` is the system's error reporting facility. It writes an error message to the standard error path. Most OS-9 for 68K systems print ERROR #mmm.nnn. Error numbers 000:000 to 063:255 are reserved for the operating system.

`path` is the path to the error message file. (Input)

If `path` is zero, standard format is specified.

`errnum` is the error number. (Input)

Only the lower 16 bits of `errnum` are used.

If an error path number is specified, the path is searched for a text description of the error encountered. The error message path contains an ASCII file of error messages. Each line may be up to 80 characters long. If the error number matches the first 7 characters in a line (that is, 000:215), the rest of the line is printed along with the error number.

Error messages may be continued on several lines by beginning each continuation line with a space. An example error file might contain lines like this:

000:214 (E\$FNA) File not accessible.

An attempt to open a file failed. The file was found, but is inaccessible to you in the requested mode. Check the file's owner ID and access attributes.

000:215 (E\$BPNam) Bad pathlist specified.

The specified pathlist is syntactically incorrect.

Attributes

Operating System:	OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

_os_protect()

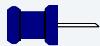
Prevent Access to Memory Block

Syntax

```
#include <process.h>
#include <types.h>
error_code _os_protect(
    void          *mem_ptr,
    u_int32       size,
    u_int32       perm,
    process_id   pid);
```

Description

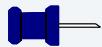
`_os_protect()` removes a process' permission to access a block of memory.



Note

Only super-group users may use the `_os_protect` function.

mem_ptr	is a pointer to the beginning of the memory area to remove access permissions. (Input)
size	specifies the size of memory. (Input)
perm	specifies the permissions to remove. (Input)
	Only the lower 16 bits of <code>perm</code> are used.
pid	specifies the process identifier for the target process. (Input)



Note

`_os_protect()` only performs a useful function on MMU/SSM systems. On non-MMU/SSM systems, `_os_protect()` simply returns without preventing access to the memory block.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`F$Protect`
`F_PROTECT`

OS-9 for 68K Technical Manual
OS-9 Technical Manual

_os_prsnam()

Parse Path Name

Syntax

```
#include <strings.h>
#include <types.h>
error_code _os_prsnam(
    const char    *name,
    u_int32       *length,
    char          *delimiter,
    char          **updated);
```

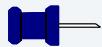
Description

`_os_prsnam()` parses a string for a valid pathlist element and returns its size. This call parses one element in a pathlist, not the entire pathlist. A valid pathlist element may contain the following characters:

A -	zUpper case letters .	Periods
a -	zLower case letters _	Underscores
0 -	9Numbers \$	Dollar signs

Other characters terminate the name and are returned as the pathlist delimiter.

name	is a pointer to the pathlist string. (Input)
length	is a pointer to the location where <code>_os_prsnam()</code> stores the length of the pathlist element. (Output)
delimiter	is a pointer to the location where <code>_os_prsnam()</code> stores the pathlist delimiter. (Output)
updated	is a pointer to the location where <code>_os_prsnam()</code> stores the pointer to the first character of name. (Output)



Note

Several `_os_prsnam()` calls are needed to process a pathlist with more than one name. `_os_prsnam()` terminates a name when it reaches a delimiter character. Usually, pathlists must be terminated with a null byte.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_cmpnam()`

Syntax

```
#include <io.h>
#include <types.h>
error_code _os_rdalst(
    char      *buffer,
    u_int32   *count);
```

Description

`_os_rdalst()` copies the system alias list to the caller's buffer. At most, `count` bytes are copied to the buffer. Each alias entry is null terminated.

buffer	is a pointer to the buffer into which to copy the alias list. (Output)
count	The number of total bytes to copy is passed at the location pointed to by <code>count</code> . (Input/Output)
	<code>_os_rdalst()</code> also stores the actual number of bytes copied at the location pointed to by <code>count</code> .

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_read()

Read Data from File or Device

Syntax

```
#include <modes.h>
#include <types.h>
error_code _os_read(
    path_id      path,
    void         *buffer,
    u_int32       *count);
```

Description

`_os_read()` reads the number of bytes from the specified path. The path must previously have been opened in read or update mode. The data is returned exactly as read from the file or device without additional processing or editing such as backspace and line delete. If not enough data is in the file to satisfy the read request, fewer bytes are read than requested, but an end-of-file error is not returned until the next `_os_read()` service request returns an end-of-file error.

path	specifies the path number. (Input)
buffer	is a pointer to the caller's data buffer. (Output)
count	The number of bytes requested to be read is passed at the location pointed to by count. (Input/Output) <code>_os_read()</code> also stores the number of bytes actually read at the location pointed to by count.

If the device is opened in block mode (`S_IBLKMODE`), the `count` parameter is interpreted as the number of blocks to read, whereas in non-block mode it is interpreted as the number of bytes to read.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

[_os_readln\(\)](#)

[I\\$Read](#)

[I_READ](#)

OS-9 for 68K Technical Manual

OS-9 Technical Manual

Syntax

```
#include <modes.h>
#include <types.h>
error_code _os_readln(
    path_id      path,
    void         *buffer,
    u_int32      *count);
```

Description

`_os_readln()` reads the specified number of bytes from the input file or device until an end-of-line character is encountered. On SCF-type devices, `_os_readln()` also causes line editing such as backspacing, line delete, echo, and automatic line feed to occur.

path	specifies the path number. (Input)
buffer	is a pointer to the location where the bytes to be read are placed. (Output)
count	The number of bytes requested is passed at the location pointed to by count. (Input/Output) <code>_os_readln()</code> also stores the number of bytes actually read at the location pointed to by count.

If the device is opened in block mode (`S_IBLKMODE`), the `count` parameter is interpreted as the number of blocks to read, whereas in non-block mode it is interpreted as the number of bytes to read.

`_os_readln()` does not look for a carriage return.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

[_os_read\(\)](#)

I\$ReadLn

I_READLN

OS-9 for 68K Technical Manual

OS-9 Technical Manual

_os_rellk()

Release Ownership of Resource Lock

Syntax

```
#include <lock.h>
#include <types.h>
error_code _os_rellk(
    lk_desc *lock);
```

Description

`_os_rellk()` releases ownership of a resource lock and activates the next process waiting to acquire the lock. The next process in the lock's queue is activated and granted exclusive ownership of the resource lock. If no other process is waiting on the lock, the lock is simply marked free for acquisition.

lock is a pointer to the lk_desc structure of the lock to release. (Input)

Attributes

Operating System:	OS-9
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

lock.1

See Also

```
_os_acqlk()
_os_caqlk()
_os_crlk()
_os_ddlk()
_os_dellk()
    os_waitlk()
```

F RELIJK

OS-9 Technical Manual

_os9_retpd()

Return Fixed Block of Memory

Syntax

```
#include <process.h>
#include <types.h>
error_code _os9_retpd(
    u_int32    number,
    void      *table);
```

Description

`_os9_retpd()` deallocates a fixed block of memory.

number is the number of the table element.
(Input)

Only the lower 16 bits of number are used. (Input)

`table` is a pointer to the address of the table to use.

Attributes

Operating System:	OS-9 for 68K
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

os lib.l

See Also

`_os9_allpd()`
`_os_findpd()`

_os_rte()

Return from Signal Interrupt Routine

Syntax

```
#include <signal.h>
#include <types.h>
error_code _os_rte(void);
```

Description

`_os_rte()` terminates a process' signal intercept routine and continues executing the main program.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

[_os_intercept\(\)](#)

F\$RTE

F_RTE

OS-9 for 68K Technical Manual

OS-9 Technical Manual

_os_rtnprc()

Deallocate Process Descriptor

Syntax

```
#include <process.h>
#include <types.h>
error_code _os_rtnprc(
    process_id    proc_id);
```

Description

`_os_rtnprc()` deallocated a process descriptor previously allocated by `_os_alocproc()`. You must ensure that the process' system resources are returned before calling `_os_rtnprc()`.

`proc_id` identifies the process descriptor. (Input)

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

[`_os_alocproc\(\)`](#)
[`_os_findpd\(\)`](#)

_os9_salarm_atdate()

Execute System-State Subroutine at Gregorian Date/Time

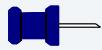
Syntax

```
#include <regs.h>
#include <alarm.h>
#include <types.h>
error_code _os9_salarm_atdate(
                                alarm_id      *alm_id,
                                u_int32        time,
                                u_int32        date,
                                REGISTERS     *regs);
```

Description

`_os9_salarm_atdate()` executes a system-state subroutine at a specific date and time. The alarm subroutine executes anytime the system date/time becomes greater than or equal to the alarm time.

alm_id	is a pointer to the location where <code>_os9_salarm_atdate()</code> stores the alarm ID. (Output)
time	is the time for the alarm to go off in the form 00hhmmss. (Input)
date	is the date for the alarm to go off in the form yyymmdd. (Input)
regs	is a pointer to the register image to pass to the function. (Input)
	The address of the system-state subroutine to execute should be set in <code>regs->pc</code> .



Note

`_os9_salarm_atdate()` only allows you to specify the time to the nearest second. However, it does adjust if the system's date and time have changed (using `_os9_setime()`).

Attributes

Operating System:	OS-9 for 68K
State:	User
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os9_setime()`

`F_ALARM`

OS-9 for 68K Technical Manual

_os_salarm_atime()

Execute System-State Subroutine at Specified Time

Syntax

```
#include <alarm.h>
error_code _os_salarm_atime(
    alarm_id      *alarm_id,
    u_int32        time,
    u_int32        flags,
    u_int32        (*func)(),
    void          *func_pb);
```

Description

`_os_salarm_atime()` executes the specified subroutine after the specified time interval has elapsed. The alarm time is an absolute time, not a relative time. The time value is considered to be a value in seconds since a date specified in Greenwich Mean Time. To determine the specified date, reference the applicable manual, ***OS-9 for 68K Technical Manual*** or ***OS-9 Technical Manual***.

<code>alarm_id</code>	is a pointer to the location where <code>_os_salarm_atime()</code> stores the alarm ID. (Output)
<code>time</code>	specifies the time when the alarm expires. (Input)
<code>flags</code>	specifies the alarm flags. (Input)
<code>func</code>	is a pointer to a function to execute. (Input)
<code>func_pb</code>	is a pointer to the function's parameter block. (Input)

Attributes

Operating System:	OS-9
State:	User
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

[_os_salarm_set\(\)](#)

F_ALARM

OS-9 Technical Manual

_os9_salarm_atjul()

Execute System-State Subroutine at Julian Date/Time

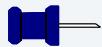
Syntax

```
#include <regs.h>
#include <alarm.h>
#include <types.h>
error_code _os9_salarm_atjul(
                           alarm_id      *alarm_id,
                           u_int32        time,
                           u_int32        date,
                           REGISTERS     *regs);
```

Description

`_os9_salarm_atjul()` executes a system-state subroutine at a specific Julian date and time. The alarm subroutine is executed anytime the system date/time becomes greater than or equal to the alarm time.

alarm_id	is a pointer to the location where <code>_os9_salarm_atjul()</code> stores the alarm ID. (Output)
time	is the time for the alarm to go off in seconds after midnight. (Input)
date	is the Julian day number for the alarm to go off. (Input)
regs	is a pointer to the register image to pass to the function. (Input)
	The address of the system-state subroutine to execute should be set in <code>regs->pc</code> .



Note

`_os9_salarm_atjul()` only allows the time to be specified to the nearest second. However, it does adjust if the system's date and time have changed (using `_os9_setime()`).

Attributes

Operating System:	OS-9 for 68K
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os9_setime()`

`F$Alarm`

OS-9 for 68K Technical Manual

_os_salarm_cycle()

Set Cyclic Alarm Clock

Syntax

```
#include <alarm.h>
#include <types.h>
error_code _os_salarm_cycle(
    alarm_id      *alrm_id,
    u_int32       time,
    u_int32       flags,
    u_int32       (*func)(),
    void          *func_pb);
```

Description

`_os_salarm_cycle()` executes the specified subroutine after the specified time interval has elapsed. Then the alarm is reset to provide recurring periodic execution of the specified subroutine.

<code>alrm_id</code>	is a pointer to the location where <code>_os_salarm_cycle()</code> stores the alarm ID. (Input)
<code>time</code>	specifies the alarm's time interval. (Input)
	time may be specified in system clock ticks; or, if the high order bit is set, the low 31 bits are considered a time in 256ths of a second. Usually, the minimum time interval allowed is 2 system clock ticks.
<code>flags</code>	specifies the alarm flags. (Input)
<code>func</code>	is a pointer to a function to execute. (Input)
<code>func_pb</code>	is a pointer to the function's parameter block. (Input)

Attributes

Operating System: OS-9
State: System
Threads: Safe
Re-entrant: Yes

Library

os_lib.l

See Also

[_os_salarm_set\(\)](#)

[F_ALARM](#)

OS-9 Technical Manual

_os9_salarm_cycle()

Execute System-State Subroutine Every N Ticks/Seconds

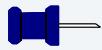
Syntax

```
#include <regs.h>
#include <alarm.h>
#include <types.>h
error_code _os9_salarm_cycle(
                                alarm_id      *alarm_id,
                                u_int32        time,
                                REGISTERS     *regs);
```

Description

`_os9_salarm_cycle()` executes a system-state subroutine after the specified time interval has elapsed. Then the alarm is reset to provide recurring periodic execution of the system-state subroutine.

<code>alarm_id</code>	is a pointer to the location where <code>_os9_salarm_cycle()</code> stores the alarm ID. (Output)
<code>time</code>	specifies the alarm's time interval. (Input)
	time may be specified in system clock ticks; or, if the high order bit is set, the low 31 bits are considered a time in 256ths of a second. Usually, the minimum time interval allowed is 2 system clock ticks.
<code>regs</code>	is a pointer to the register image to pass to the function. (Input)
	The address of the system-state subroutine to execute should be set in <code>regs->pc</code> .



Note

Keep cyclic system-state alarms as fast as possible. Schedule them with as long a cycle as possible to avoid consuming a large portion of available CPU time.

Attributes

Operating System:	OS-9 for 68K
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

[_os9_salarm_set\(\)](#)

F\$Alarm

OS-9 for 68K Technical Manual

_os_salarm_delete()

Remove Pending Alarm Request

Syntax

```
#include <alarm.h>
#include <types.h>
error_code _os_salarm_delete(alarm_id alarm_id);
```

Description

`_os_salarm_delete()` removes a cyclic alarm or any alarm that has not expired.

alarm_id	specifies the alarm ID. If <code>alarm_id</code> is 0 (zero), all pending alarm requests are removed, which are owned by the system process (forked with <code>TH_SPOWN</code>). (Input)
----------	--

In a threaded environment, the `alarm_id` is checked to determine the owner of the thread. If a 0 is passed for the `alarm_id`, all pending alarm requests for the system will be removed, which are owned by the system process (forked with `TH_SPOWN`).

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

F\$Alarm	<i>OS-9 for 68K Technical Manual</i>
F_ALARM	<i>OS-9 Technical Manual</i>

_os_salarm_delete_sp()

Delete Installed Alarms

Syntax

```
#include <alarm.h>
#include <types.h>
error_code _os_salarm_delete_sp(alrm_id, flags);
```

Description

`_os_salarm_delete_sp()` takes a flag value to allow correct deletion of installed alarms, thus providing a workaround for a historic OS-9 bug in `_os_salarm_delete`.

<code>alrm_id</code>	specifies the alarm ID. (Input) If <code>alrm_id</code> is zero, all pending alarm requests are removed.
<code>flags</code>	specifies that the alarm(s) to be deleted are owned by either the system process (<code>TH_SPOWN</code>) or by the current user process (<code>flags=0</code>). (Input)

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`F$Alarm`
`F_ALARM`

OS-9 for 68K Technical Manual
OS-9 Technical Manual

_os_salarm_reset()

Reset Alarm Clock

Syntax

```
#include <alarm.h>
#include <types.h>
error_code _os_salarm_reset(
    alarm_id      alrm_id,
    u_int32        time,
    u_int32        flags,
    u_int32        (*func)(),
    void          *func_pb);
```

Description

`_os_salarm_reset()` allows you to change the parameters on an existing alarm.

alrm_id	specifies the alarm ID to reset. (Input)
time	specifies the alarm's time interval or a time. (Input)
	A time interval may be specified in system clock ticks or 256ths of a second. The minimum time interval allowed is 2 system clock ticks.
flags	specifies the alarm flags. (Input)
func	is a pointer to a function to execute. (Input)
func_pb	is a pointer to the function's parameter block. (Input)

Attributes

Operating System:	OS-9
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

[_os_salarm_set\(\)](#)

F_ALARM

OS-9 Technical Manual

_os_salarm_set()

Set Alarm Clock

Syntax

```
#include <alarm.h>
#include <types.h>
error_code _os_salarm_set(
    alarm_id      *alrm_id,
    u_int32        time,
    u_int32        flags,
    u_int32        (*func)(),
    void          *func_pb);
```

Description

`_os_salarm_set()` executes the specified subroutine after the specified time interval has elapsed.

alrm_id	is a pointer to the location where <code>_os_salarm_set()</code> stores the alarm ID. (Output)
time	specifies the alarm's time interval or a time. (Input) <code>time</code> may be specified in system clock ticks; or, if the high order bit is set, the low 31 bits are considered a time in 256ths of a second. Usually, the minimum time interval allowed is 2 system clock ticks.
flags	specifies the alarm flags. (Input)
func	is a pointer to the function to execute. (Input)
func_pb	is a pointer to the function's parameter block. (Input)

Attributes

Operating System:	OS-9K
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

F_ALARM

OS-9 Technical Manual

_os9_salarm_set()

Execute System-State Subroutine after Specified Time Interval

Syntax

```
#include <regs.h>
#include <alarm.h>
#include <types.h>
error_code _os9_salarm_set(
                           alarm_id      *alarm_id,
                           u_int32        time,
                           REGISTERS     *regs);
```

Description

`_os9_salarm_set()` executes a system-state subroutine after the specified time interval has elapsed.

alarm_id	is a pointer to the location where <code>_os9_salarm_set()</code> stores the alarm ID. (Output)
time	specifies the time interval. (Input) The time interval, <code>time</code> , may be specified in system clock ticks or 256ths of a second. The minimum time interval allowed is 2 system clock ticks.
regs	is a pointer to the register image to pass to the function. (Input) The address of the system-state subroutine to execute should be set in <code>regs->pc</code> .

Attributes

Operating System:	OS-9 for 68K
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

F\$Alarm

OS-9 for 68K Technical Manual

_os_scache() Cache Control

Syntax

```
#include <regs.h>
#include <cache.h>
#include <types.h>
error_code _os_scache(
    u_int32      control,
    void        (**cctl)(),
    **cctl_data,
    void        *addr,
    u_int32      size);
```

Description

`_os_scache()` performs operations on the system instruction and/or data caches, if there are any.

control	specifies the cache operation. (Input)
	If <code>control</code> is zero, the system instruction and data caches are flushed. Non-super group, user-state processes may perform this operation.

The following table shows the bits that are defined in the `control` parameter for precise operation. Only system-state and super-group processes can use the `control` parameter for precise operation.

Table 2-21 control Parameter Bits Defined for `_os_scache()`

Bit	Name	Description
Bit 0	C_ENDATA	If set, enables the data cache.
Bit 1	C_DISDATA	If set, disables the data cache.

Table 2-21 control Parameter Bits Defined for _os_scache()

Bit	Name	Description
Bit 2	C_FLDATA	If set, flushes the data cache.
Bit 3	C_INVDATA	If set, invalidates the data cache (if supported by hardware).
Bit 4	C_ENINST	If set, enables the instruction cache.
Bit 5	C_DISINST	If set, disables the instruction cache
Bit 6	C_FLINST	If set, flushes the instruction cache.
Bit 7	C_INVINNINST	If set, invalidates the instruction cache.
Bit 8	C_ADDR	If set, flags a target address for flush operation.
Bits 9-14		Reserved for future use by Microware.
Bit 15	C_GETCCTL	If set, returns a pointer to the cache control routine and cache static global data.

Table 2-21 control Parameter Bits Defined for _os_scache()

Bit	Name	Description
Bit 16	C_STODATA	If set, stores the data cache (if supported by hardware).
Bits 17-31		Reserved for future use by Microware.



Note

All other bits are reserved. If any reserved bit is set, an EOS_PARAM error is returned.

By OS-9 convention, the terms flush, invalidate, and store are defined as follows:

Flush	Operation by which a cache line is written to main memory and marked as invalid.
Invalidate	Operation by which a cache line is marked as invalid. No write of cache line to main memory is performed.
Store	Operation by which a cache line is written to main memory. The cache line is not marked as invalid.
cctl	is a pointer to the location where _os_scache() stores the cache routine's address. (Output)

These specific operations may or may not be supported by a particular hardware configuration. The Flush operation is considered the base operation and will be supported by most processors.

cctl_data	is a pointer to the location where <code>_os_scache()</code> stores the cache routine's static storage address. (Output)
addr	is a pointer that specifies the address for the flush operation. (Input)
	This parameter is used when the C_ADDR bit of <code>control</code> is set. Set <code>addr</code> to 0 when C_ADDR is not set.
size	indicates the size of the target memory area to be flushed. (Input)

Any program that builds or changes executable code in memory should flush the instruction cache with `_os_scache()` before executing new code. This is necessary because the hardware instruction cache is not updated by data (write) accesses and may contain the unchanged instruction(s). For example, if a subroutine builds a system call on its stack, you must use the `_os_scache()` system call to flush the instruction cache before executing the temporary instructions.

If the C_GETCCTL bit of `control` is set, `_os_scache()` returns a pointer to the cache control routine in the cache extension module and a pointer to that routine's static global data. This enables drivers and file managers to call the cache routine directly rather than making a possibly time-consuming `_os_scache()` request. The prototype for this cache control routine is as follows:

```
error_code (*cache_cctl) (u_int32 control,
                         void *addr, u_int32 size);
```

The `control`, `addr`, and `size` parameters are as defined earlier. The only exception is the C_GETCCTL bit in the `control` parameter. This obviously has no meaning in this context.

Attributes

Operating System:	OS-9
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

I_SETSTAT, SS_CACHE	<i>OS-9 Technical Manual</i>
F_CCTL (System State)	<i>OS-9 Technical Manual</i>

Syntax

```
#include <memory.h>
#include <types.h>
error_code _os9_schbit(
    u_int16          *bit_number,
    u_int16          *count,
    void             *address,
    void             *bitmap_end);
```

Description

`_os9_schbit()` searches the specified buffer which represents a bit map for a free block (cleared bits) of the required length, starting at the beginning bit number. `_os9_schbit()` returns the offset of the first block found of the specified length.

bit_number	The beginning bit number to search is passed at the location pointed to by <code>bit_number</code> . (Input/Output)
	<code>_os9_schbit()</code> also stores the beginning bit number found at the location pointed to by <code>bit_number</code> .
count	The number of bits needed is passed at the location pointed to by <code>count</code> . (Input/Output)
	<code>_os9_schbit()</code> also stores the number of bits found at the location pointed to by <code>count</code> .
address	is a pointer to the base address of the bitmap. (Input)
bitmap_end	is a pointer to the end of the bitmap plus one. (Input)

If `_os9_schbit()` returns `SUCCESS`, you should check `count` to make sure that you received the requested number of bits. If `count` is less than the requested amount, `_os9_schbit()` returned the largest number of bits available, starting at `bit_number`.

Attributes

Operating System:	OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

[`_os9_allbit\(\)`](#)
[`_os9_delbit\(\)`](#)

Syntax

```
#include <modes.h>
#include <types.h>
error_code _os_seek(
    path_id      path,
    u_int32      position);
```

Description

`_os_seek()` repositions the path's file pointer. The file pointer is the 32-bit address of the next byte in the file to read or write. A `seek` may be performed to any value, even if the file is not large enough. Subsequent `write` requests automatically expand the file to the required size, if possible. `read` requests return an end-of-file condition.

`path` specifies the path number. (Input)

`position` specifies the new position in the file.
(Input)

If the device is opened in block-mode (`S_IBLKMODE`), the `position` parameter to `_os_seek()` is interpreted by the function as a block to seek to (rather than as a byte to seek to, which is the behavior when in non-block mode).

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`I$Seek`

`I_SEEK`

OS-9 for 68K Technical Manual

OS-9 Technical Manual

_os_sema_init()

Initialize Semaphore Data Structure for Use

Syntax

```
#include <semaphore.h>
error_code _os_sema_init(semaphore *sema);
```

Description

OS-9 for 68K: `_os_sema_init()` is valid in version 3.0 and greater.

`_os_sema_init()` initializes a semaphore data structure for subsequent use. The application should call `_os_sema_init()` before the first use of the semaphore. The semaphore may be part of a data module or it may be part of any data structure within the application.

`sema` is a pointer to the semaphore data structure to initialize. (Input)



Note

It is necessary to clear the shared memory area before making the `_os_sema_init()` call. In some instances, such as with driver applications or global variables, initializing the structure to zero will clear the shared memory area automatically.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

[_os_sema_p\(\)](#)
[_os_sema_term\(\)](#)
[_os_sema_v\(\)](#)

_os_sema_p()

Reserve Semaphore (Acquire Exclusive Access)

Syntax

```
#include <semaphore.h>
error_code _os_sema_p(semaphore *sema);
```

Description

OS-9 for 68K: `_os_sema_p()` is valid in version 3.0 and greater.

`_os_sema_p()` attempts to reserve the semaphore. If the semaphore is busy, the caller is suspended until the current holder releases the semaphore.

`sema` is a pointer to the semaphore to reserve.
 (Input)

`_os_sema_p()` returns an error code if a signal arrives while a process is waiting for a semaphore (i.e., it signals that the process is resuming execution--because of the incoming signal--without obtaining the semaphore).

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

[_os_sema_init\(\)](#)
[_os_sema_term\(\)](#)
[_os_sema_v\(\)](#)

_os_sema_term()

Terminate Use of Semaphore Data Structure

Syntax

```
#include <semaphore.h>
error_code _os_sema_term(semaphore *sema);
```

Description

OS-9 for 68K: `_os_sema_term()` is valid in version 3.0 and greater.

`_os_sema_term()` terminates the use of the semaphore. The application should call `_os_sema_term()` before terminating the application.

`sema` is a pointer to the semaphore data structure to terminate. (Input)



WARNING

Within OS-9 and OS-9 for 68K, each call to `_os_sema_term()` invalidates the sync code of the semaphore. Therefore, the last process to use the semaphore should be the only one to terminate it.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

[_os_sema_init\(\)](#)
[_os_sema_p\(\)](#)
[_os_sema_v\(\)](#)

_os_sema_v()

Release Semaphore (Release Exclusive Access)

Syntax

```
#include <semaphore.h>
error_code _os_sema_v( semaphore *sema );
```

Description

OS-9 for 68K: `_os_sema_v()` is valid in version 3.0 and greater.

`_os_sema_v()` releases the reservation of a semaphore. If a process is waiting on the semaphore, it is activated, allowing it to attempt to acquire the semaphore.

`sema` is a pointer to the semaphore to release.
(Input)

The `_os_sema_v()` call releases the semaphore, but does not give it to the next process waiting for it. Instead, if the process that issued the `_os_sema_v()` call issues a `_os_sema_p()` call before its slice is up, then it re-obtains the semaphore (i.e., the waiting process if running at the same priority was unable to obtain it).

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

```
_os_sema_init()  
_os_sema_p()  
os sema term()
```

_os_send()

Send Signal to Another Process

Syntax

```
#include <signal.h>
#include <types.h>
error_code _os_send(
                    process_id      proc_id,
                    signal_code     signal);
```

Description

`_os_send()` sends a signal to the specified process. A process may send the same signal to all processes of the same group/user ID by passing zero as the receiving process' ID number.

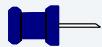
If the receiving process does not have a signal handler installed, the signal will cause the receiving process to be killed. If the receiving process has a signal handler and is blocked on I/O, then a variety of error or result codes may be returned to the mainline program depending on the I/O system in use. For example, SCF in OS-9 for 68K would return the signal number sent by `_os_send()`. SPF would return an error code of 000:233.

<code>proc_id</code>	is the process ID number for the intended receiver. (Input)
<code>signal</code>	specifies the signal code to send. (Input)



Note

On OS-9 for 68K, the lower 16 bits of the signal value are used. The upper 16 bits are ignored.



Note

There are times in which the kernel sends a signal to a threaded process ID. When this occurs, the kernel looks for the first thread containing a signal handler and delivers the signal to it.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

[_os_wait\(\)](#)
[_os_intercept\(\)](#)
[_os_sleep\(\)](#)

F\$Send
F_SEND

OS-9 for 68K Technical Manual
OS-9 Technical Manual

_os_setcrc()

Generate Valid CRC in Module

Syntax

```
#include <module.h>
#include <types.h>
error_code _os_setcrc(mh_com *mod_head);
```

Description

`_os_setcrc()` updates the header parity and CRC of a module in memory. The module must have the correct size and sync bytes; other parts of the module are not checked.

`mod_head`

is a pointer to the module. (Input)

The module image must start on a longword address or an exception may occur.



WARNING

Although `_os_setcrc()` and `_setcrc()` functions update the header and `crc` of a module in memory, the copy of the module header in the module directory is not affected. Therefore, you have an invalid module directory entry for the memory module with the new CRC; and an error 236 results if links to the module are attempted.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

[_os_crc\(\)](#)

_os_setime()

Set System Time

Syntax

```
#include <time.h>
#include <types.h>
error_code _os_setime(u_int32 time);
```

Description

`_os_setime()` sets the current system time and starts the system real-time clock to produce time slice interrupts.



Note

Only super users may set the time.

time

is stored as the number of seconds since a specific date in Greenwich Mean Time. (Input)

time is the desired current time in Greenwich Mean Time.



For More Information

To determine the specified date, reference the applicable manual, **OS-9 for 68K Technical Manual** or **OS-9 Technical Manual**.

On OS-9 for 68K, the binding for this call performs conversions which allow compatibility with OS-9. If speed is a consideration, refer to [_os9_setime\(\)](#).

The time is not validated in any way. If time is zero on systems with a battery-backed clock, the actual time is read and set from the real-time clock.

In order to get the system time/date from the real time clock or the battery-backed clock, the month field in the date parameter must be [0](#).

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

[gmtime\(\)](#)
[_os_link\(\)](#)
[_os_gettime\(\)](#)
[_os9_setime\(\)](#)

F\$STime
F_STIME

OS-9 for 68K Technical Manual
OS-9 Technical Manual

_os9_setime()

Set System Date/Time

Syntax

```
#include <time.h>
#include <types.h>
error_code _os9_setime(
    u_int32 time,
    u_int32 date);
```

Description

`_os9_setime()` sets the current system date/time and starts the system real-time clock to produce time slice interrupts. Only super users may set the time.

time is in the format 00hhmmss. (Input)

date is in the format yyyy-mm-dd. (Input)

Each value is one byte in length, except for the year which is 2 bytes long. Values must be in hexadecimal format.

The date and time are not checked for validity. On systems with a battery-backed clock, it is usually only necessary to supply the year to the `_os9_setime()` call. The actual date and time are read from the real-time clock.

In order to get the system time/date from the real-time clock or the battery-backed clock, the month field in the date parameter must be 0.

Attributes

Operating System:	OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

[_os_link\(\)](#)
[_os9_gettime\(\)](#)
[F\\$STime](#)

OS-9 for 68K Technical Manual

_os_setpr()

Set Process Priority

Syntax

```
#include <process.h>
#include <types.h>
error_code _os_setpr(
    process_id      proc_id,
    u_int32         priority);
```

Description

`_os_setpr()` changes the process priority to the value specified by priority. A super user (group ID zero) may change any process' priority. A non-super user can only change the priorities of processes with the same user ID.

proc_id	is the process ID. (Input)
priority	specifies the new priority. (Input)
	65535 is the highest priority; 0 is the lowest.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

F\$SPrior	<i>OS-9 for 68K Technical Manual</i>
F_SPRIOR	<i>OS-9 Technical Manual</i>

Syntax

```
#include <sg_codes.h>
#include <types.h>
error_code _os_setstat(
    path_id      path,
    u_int32      code,
    void        *pb);
```

Description

`_os_setstat()` is a wildcard call used to handle individual device parameters that are not uniform on all devices or are highly hardware dependent.

`path` is the path number. (Input)
`code` is the SetStat code. (Input)
Only the lower 16 bits of code are used.
`pb` is a pointer to the parameter block.

The exact operation of this call depends on the device driver and file manager associated with the path. The mnemonics for the status codes are located in the `sg_codes.h` header file. This manual describes the status codes currently defined by Microware and the functions that they perform. These calls have an `_os_ss_` prefix.



Note

This function is available within OS-9 for 68K, where it resides in the `conv_lib.l` library. However, it can be used only with DPIO file managers and drivers, such as SoftStax.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

Syntax

```
#include <svctbl.h>
#include <types.h>
error_code _os_setsvc(
    u_int32      count,
    u_int32      state_flag,
    void        *init_tbl,
    void        *params);
```

Description

`_os_setsvc()` adds or replaces a system service in OS-9's user and privileged system service request tables.

`count` is a count of the entries in the initialization table. (Input)

`state_flag` specifies whether user or system state tables, or both, are updated. (Input)

- If `state_flag` is `USER_STATE` (1), only the user table is updated.
- If `state_flag` is `SYSTEM_STATE` (2), only the system table is updated.
- If `state_flag` is `USER_STATE` OR'ed with `SYSTEM_STATE` (3), both tables are updated.

init_tbl

is a pointer to the initialization table.
(Input)

An example initialization table might look like this:

```
error_code printmsg(), service();  
  
svctbl syscalls[] =  
{  
    {F_PRINT, printmsg, 0, 1, 1},  
    {F_SERVICE, service, 0, 1, 1}  
};
```

params

may be a pointer to anything, but is intended to be a pointer to global static storage. (Input)

When a system call is used, the params data pointer is used as the global static storage pointer for the function.



Note

Use `get_static()` from `cpu.l` to acquire your current static storage pointer.

Valid service request codes range from 0 to 255.

The available service request codes are located in the `funcs.h` file.

Attributes

Operating System:	OS-9
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os9_setsvc()

Service Request Table Initialization

Syntax

```
#include <funcs.h>
#include <types.h>
error_code _os9_setsvc(
    void      *init_tbl,
    void      *params);
```

Description

`_os9_setsvc()` adds or replaces function requests in OS-9 for 68K's user and privileged system service request tables. To install a system-state-only system call, `SYSTRAP` is added to the function code number. Otherwise, both the system- and user-state dispatch tables are updated. `SYSTRAP` is defined in the `funcs.h` header file.

`init_tbl` is a pointer to the initialization table.
(Input)

The following is an example initialization table:

```

struct svc
{
    u_int16 func_code;      * service call function code number
    u_int16 offset;         * difference between the address of
                           the
                           * next table entry and the address of
                           the
                           * routine.*/
} svctbl[] =
{
    {F_SERVICE,(u_int32)routine -(u_int32)&svctbl[1]},
        {F_SERVICE+SYSTRAP,
            (u_int32)sys_routine -(u_int32)&svctbl[2]};
    {-1,-1}
};

```

Valid service request codes range from 0 to 255.

If the most significant bit of the function code word is set, only the system table is updated. Otherwise, both the system and user tables are updated.

`params`

is intended to be a pointer to the global static storage. (Input)

This allows you to associate a global data pointer with each installed system call. When the system call is used, the data pointer is automatically passed.

Whatever `params` points to is passed to the system call; `params` may be a pointer to anything.



Note

The OS-9 for 68K service request codes are located in
`/MWOS/OS9/SRC/DEFS.`

Attributes

Operating System:	OS-9 for 68K
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

Syntax

```
#include <sysglob.h>
#include <types.h>
error_code _os_setsys(
    u_int32      offset,
    u_int32      size,
    glob_buff   sysvar);
```

Description

`_os_setsys()` changes a system global variable. These variables have a `d_` prefix and are located in the system library `sysglob.h`. Consult the files in the `DEFS` directory for a description of the system global variables. Only super users may change system variables. Super users must be extremely careful when changing system global variables.

`offset` is the offset to the system global variables. (Input)

`size` specifies the size of the target variable. (Input)

`sysvar` is the new value for the system global variable. (Input)



For More Information

Refer to your operating system technical manual for the system global variables that may be changed.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

Syntax

```
#include <process.h>
#include <types.h>
error_code _os_setuid(owner_id user_id);
```

Description

`_os_setuid()` changes the current user ID to `user_id`.

`user_id` is the desired group/user ID number.
(Input)

The following restrictions apply to `_os_setuid()`:

- Users with group ID zero may change their IDs to anything without restriction.
- A primary module owned by a group zero user may change its ID to anything without restriction.
- Any primary module may change its user ID to match the module's owner.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_sgetstat()

GetStat Call Using System Path Number

Syntax

```
#include <sg_codes.h>
#include <types.h>
error_code _os_sgetstat(
    path_id      spath,
    u_int32      code,
    void        *pb);
```

Description

`_os_sgetstat()` makes a specific GetStat or SetStat call.

`spath` is the system path number. (Input)

`code` is the GetStat code. (Input)

`pb` is a pointer to the parameter block corresponding to the function being performed. (Input)

If the function does not have a parameter block (for example, `_os_ss_reset()`), `pb` should be NULL.

Attributes

Operating System: OS-9

State: User and System

Threads: Safe

Re-entrant: Yes

Library

`os_lib.l`

_os_sgs_devnm()

Return Device Name

Syntax

```
#include <sg_codes.h>
#include <types.h>
error_code _os_sgs_devnm(
    path_id      path,
    char        *namebuf );
```

Description

`_os_sgs_devnm()` returns the name of the device associated with the specified system path number.

path	is the system path number. (Input)
namebuf	is a pointer to the location where <code>_os_sgs_devnm()</code> stores the device name. (Output)

Attributes

Operating System:	OS-9K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_siglngj()

Set Signal Mask Value and Return on Specified Stack Image

Syntax

```
#include <signal.h>
#include <types.h>
error_code _os_siglngj(
    void      *usp,
    u_int16   siglvl);
```

Description

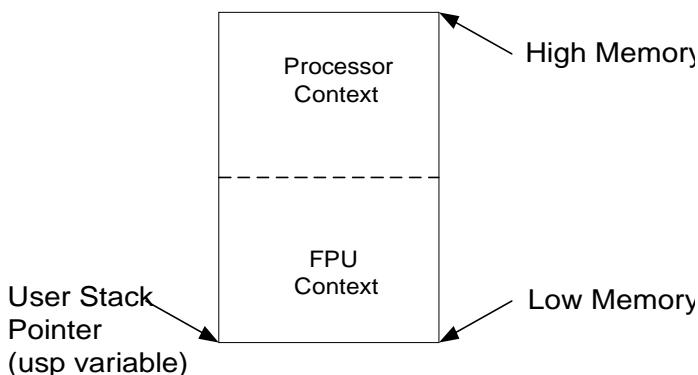
F_SIGLNGJ allows processes to perform `long jump()` operations from their signal intercept routines and unmask signals in one operation.

- | | |
|--------|--|
| usp | is a pointer to the new process stack image. |
| siglvl | is the new signal level value. |
- This call is usually used by nested intercept routines to resume execution in the process at a different location from where the process was interrupted by the original signal. When this call is made, the operating system performs the following functions:
- Validates and copies the target process stack image from the memory buffer pointed to by the `usp` variable to the process' system-state stack
 - Sets the process' signal mask to the value specified in the `siglvl` variable
 - Returns to the process restoring the context copied from the user state process stack image

The operating system takes appropriate precautions to verify that the memory location pointed to by the `usp` variable is accessible to the process and to ensure that the process does not attempt to make a state change.

The stack image pointed to by the `usp` variable must have the format shown in [Figure 2-9](#).

Figure 2-9 `_os_siglngj()` Required Stack Image



The specific format of the processor context is defined by the `longstk` structure definition found in the `reg<CPU Family>.h` file for the associated processor. The format of the floating-point context varies depending on whether the target system has a hardware floating-point unit versus floating-point emulation software.

For floating-point hardware, the stack image is the same as that defined by the `fregs` structure definition found in the associated `reg<CPU Family>.h` header file.

For floating-point emulation, the floating-point context differs from the hardware implementation context in that it may contain additional context information specific to the FPU module performing the emulation. The definition for the floating-point context as used by the FPU module is the `fpu_context` structure which is also defined in the associated `reg<CPU Family>.h` header file for the target processor.

If a particular application needs to access the contents of the process context, it may use the size of these structures for indexing.

Alternatively, the application can determine the size of the FPU context at runtime by accessing the kernel globals field, d_fpusize, which contains the size of the FPU context.

Attributes

Operating System:	OS-9
State:	User
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

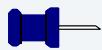
[_os_sigmask\(\)](#)
[_os_send\(\)](#)

_os_sigmask()

Mask/Unmask Signals During Critical Code

Syntax

```
#include <signal.h>
#include <types.h>
error_code _os_sigmask(int32 mode);
```



Note

The only function that `_os_sigmask` provides in system-state processes is to wake up that process. It does not interrupt code that is currently executing.

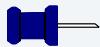
Description

`_os_sigmask()` enables signals to reach the calling process or disables signals from reaching the calling process. If a signal is sent to a process whose mask is disabled, the signal is queued until the process mask becomes enabled.

mode is the process signal level. (Input)

Table 2-22 _os_sigmask() Parameter Mode Values

If Mode Is	The Mask Level Is
0	Cleared
1	Set or incremented
-1	Decrement



Note

Signals are analogous to hardware interrupts and should be masked sparingly.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

[_os_sleep\(\)](#)
[_os_wait\(\)](#)

Example

To ensure that an entire structure can be cleared without the reception of a signal one might use:

```
_os_sigmask(1);
memset(sptr, 0, sizeof(*sptr));
_os_sigmask(-1);
```

The state of the signal mask is preserved using this method.

To avoid a race condition:

```
_os_sigmask(1);
sig_received = FALSE;
sleep(0);           * sleep does an implied
                   * _os_sigmask(0);
```

Syntax

```
#include <signal.h>
error_code _os_sigreset(void);
```

Description

`_os_sigreset()` resets the context stack for a signal intercept. Normally, the operating system keeps the state of the process on the system stack while a signal intercept routine executes. If signals are unmasked during the intercept routine and another signal occurs before the intercept routine finishes, another state is pushed on the system stack.

Similarly, if the intercept routine does a `longjmp()` and then unmasks signals, more states are placed on the system stack by subsequent signals.

To prevent the system stack from overflowing, whenever a program uses `longjmp()` calls out of an intercept routine or unmasks signals in an interrupt service routine with the intent of never using `_os_rte()` to return, it should use `_os_sigreset()`:

```
if(setjmp(x) != 0) {
    _os_sigreset();
    _os_sigmask(-1);
}
```

If the code actually expects to return through the nested intercept routines with multiple `_os_rte()` calls, the states must be left where they are and `_os_sigreset()` should not be called.

Attributes

Operating System:	OS-9 for 68K
State:	User
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

[_os_rte\(\)](#)

_os_sigrs()

Resize Current Process' Signal Queue Block

Syntax

```
#include <signal.h>
#include <types.h>
error_code _os_sigrs(u_int32 signals);
```

Description

`_os_sigrs()` allows a process to change the maximum number of signals that are queued on its behalf.

signals is the new maximum number of signals.
(Input)

`_os_sigrs()` can be used to increase or decrease the number of signals queued. An error is returned (EOS_PARAM) if a request is made to reduce the number of queued signals while there are signals pending. The initial default for the system is specified in the system `Init` module.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os lib.l

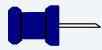
Syntax

```
#include <signal.h>
#include <types.h>
error_code _os_sleep(
    u_int32          *ticks,
    signal_code      *signal);
```

Description

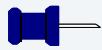
`_os_sleep()` deactivates the calling process until the requested number of ticks have elapsed.

<code>ticks</code>	<p>The length of time to sleep in ticks/second is passed at the location pointed to by <code>ticks</code>. (Input)</p> <ul style="list-style-type: none"> • If <code>ticks</code> is zero, the process sleeps indefinitely. • If <code>ticks</code> is one, the process gives up a time slice but does not necessarily sleep for one tick. • <code>_os_sleep()</code> stores the number of ticks left to sleep when awakened prematurely at the location pointed to by <code>ticks</code>. If <code>_os_sleep()</code> completes without interruption, <code>ticks</code> contains the value 0. <p><code>time</code> may be specified in system clock ticks; or, if the high order bit is set, the low 31 bits are considered a time in 256ths of a second.</p>
<code>signal</code>	<p>is a pointer to the location where <code>_os_sleep()</code> stores the last signal the process received. (Output)</p>



Note

`_os_sleep()` cannot be used to time more accurately than ± 1 tick because it is not known when the `_os_sleep()` request was made during the current tick.



Note

The system clock must be running to perform a timed sleep. The system clock is not required to perform an indefinite sleep or to give up a time slice.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

[`_os_send\(\)`](#)
[`_os_wait\(\)`](#)

_os9_sleep()

Put Calling Process to Sleep

Syntax

```
#include <signal.h>
#include <types.h>
error_code _os9_sleep(u_int32 *ticks);
```

Description

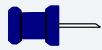
`_os9_sleep()` deactivates the calling process until the requested number of ticks have elapsed.

`ticks`

The requested number of ticks is passed at the location pointed to by `ticks`.
(Input/Output)

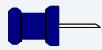
- If `ticks` is zero, the process sleeps indefinitely.
- If `ticks` is one, the process gives up a time slice but does not necessarily sleep for one tick.
- `_os9_sleep()` stores the number of ticks left to sleep when awakened prematurely at the location pointed to by `ticks`. If `_os9_sleep()` completes without interruption, `ticks` contains the value 0.

`time` may be specified in system clock ticks; or, if the high order bit is set, the low 31 bits are considered a time in 256ths of a second.



Note

`_os9_sleep()` cannot be used to time more accurately than ± 1 tick because it is not known when the `_os9_sleep()` request was made during the current tick.



Note

The system clock must be running to perform a timed sleep. The system clock is not required to perform an indefinite sleep or to give up a time slice.

Attributes

Operating System:	OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

[`_os_send\(\)`](#)
[`_os_wait\(\)`](#)

Syntax

```
#include <module.h>
#include <types.h>
error_code _os_slink(
    u_int32      sub_num,
    const char   *mod_name,
    void         **lib_exec,
    void         **mem_ptr,
    mh_com       *mod_head);
```

Description

`_os_slink()` attempts to link or load the named module. It returns a pointer to the execution entry point and a pointer to the library's static data area for subsequent calls to the subroutine. The calling program must store and maintain the subroutine's entry point and data pointer. The calling program must also set the subroutine library's data pointer and dispatch to the correct subroutine.

sub_num	is the subroutine number. (Input)
mod_name	Only the lower 16 bits of <code>sub_num</code> are used.
lib_exec	is a pointer to the location where <code>_os_slink()</code> stores the pointer to the subroutine entry point. (Output)
mem_ptr	is a pointer to the location where <code>_os_slink()</code> stores the pointer to the subroutine static memory. (Output)
mod_head	is a pointer to the location where <code>_os_slink()</code> stores the pointer to the module header. (Output)

You can remove a subroutine by passing a null pointer for the name of the module and specifying the subroutine number.

A process can link to a maximum of 16 subroutine libraries, numbered from 0 to 15.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

[_os_tlink\(\)](#)

_os_slinkm()

Install User Subroutine Library Module by Module Pointer

Syntax

```
#include <module.h>
#include <types.h>
error_code _os_slinkm(
    u_int32      sub_num,
    mh_com       *mod_head,
    void         **lib_exec,
    void         **mem_ptr);
```

Description

`_os_slinkm()` is passed a pointer to the subroutine library module to install. It returns a pointer to the execution entry point and a pointer to the library's static data area for subsequent calls to the subroutine. The calling program must store and maintain the subroutine's entry point and data pointer. The calling program must also set the subroutine library's data pointer and dispatch to the correct subroutine.

sub_num	is the subroutine number. (Input)
mod_head	Only the lower 16 bits of <code>sub_num</code> are used.
lib_exec	is a pointer to the module header. (Input)
mem_ptr	is a pointer to the location where <code>_os_slinkm()</code> stores the pointer to the subroutine library entry point. (Output)
	is a pointer to the location where <code>_os_slinkm()</code> stores the pointer to the subroutine library static memory. (Output)

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

[_os_slink\(\)](#)
[_os_tlink\(\)](#)

_os_srqmem()

System Memory Request

Syntax

```
#include <memory.h>
#include <types.h>
error_code _os_srqmem(
    u_int32      *size,
    void         **mem_ptr,
    u_int32      color);
```

Description

`_os_srqmem()` allocates a block of a specific memory type.

`size` The byte count of the requested memory is passed at the location pointed to by `size`. (Input/Output)

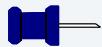
`_os_srqmem()` also stores the actual number of bytes allocated at the location pointed to by `size`. If `size` is 0xFFFFFFFF, the largest block of free memory of the specified type is allocated to the calling process.

`mem_ptr` is a pointer to the location where `_os_srqmem()` stores the pointer to the allocated memory block. (Output)

`color` specifies the memory type. (Input)

Only the lower 16 bits of `color` are used.

- If `color` is non-zero, the search is restricted to memory areas of that color. The area with the highest priority is searched first.
- If `color` is zero, the search is based only on priority.



Note

The byte count of allocated memory and the pointer to the allocated block must be saved if the memory is ever to be returned to the system.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

[_os9_srqmem\(\)](#)
[_os_srtmem\(\)](#)
[_os_mem\(\)](#)

F\$SRqCMem
F_SRQMEM

OS-9 for 68K Technical Manual
OS-9 Technical Manual

_os9_srqmem()

System Memory Request

Syntax

```
#include <memory.h>
#include <types.h>
error_code _os9_srqmem(
    u_int32      *size,
    void        **mem_ptr);
```

Description

`_os9_srqmem()` allocates a memory block from the top of available RAM.

`size` The size of the memory block is passed at the location pointed to by `size`.
(Input/Output)

`_os9_srqmem()` also stores the actual number of bytes allocated at the location pointed to by `size`. If `size` is 0xFFFFFFFF, the largest block of free memory is allocated to the calling process.

`mem_ptr` is a pointer to the location where `_os9_srqmem()` stores the pointer to the allocated memory block. (Output)



Note

The byte count of allocated memory and the pointer to the allocated block must be saved if the memory is ever to be returned to the system.

Attributes

Operating System:	OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

[_os_srtmem\(\)](#)
[_os_mem\(\)](#)
[_os_srqmem\(\)](#)
F\$SRqMem

OS-9 for 68K Technical Manual

_os_srtmem()

Return System Memory

Syntax

```
#include <memory.h>
#include <types.h>
error_code _os_srtmem(
    u_int32      size,
    void        *mem_ptr);
```

Description

`_os_srtmem()` deallocates memory when it is no longer needed.

`size` specifies the byte count of the returned memory. (Input)

`mem_ptr` is a pointer to the memory block to return. (Input)

Attributes

Operating System: OS-9 and OS-9 for 68K

State: User and System

Threads: Safe

Re-entrant: Yes

Library

`os_lib.l`

See Also

[`_os_srqmem\(\)`](#)

[`_os_mem\(\)`](#)

Syntax

```
#include <sg_codes.h>
#include <types.h>
error_code _os_ss_attr(
    path_id      path,
    u_int32      attr);
```

Description

`_os_ss_attr()` changes a file's attributes to the new value, if possible. You cannot set the directory bit of a non-directory file or clear the directory bit of a non-empty directory.

path	specifies the path number of the file. (Input)
attr	specifies the new values for the file attributes. (Input)

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

_os_ss_break()

Send Break Conditional to Serial Line

Syntax

```
#include <sg_codes.h>
#include <types.h>
error_code _os_ss_break(path_id path);
```

Description

`_os_ss_break()` sends a break conditional to the serial line.

`path` specifies the path number. (Input)

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`I_SETSTAT`, `SS_BREAK`

OS-9 Technical Manual

_os_ss_cache()

Enable/Disable RBF Caching

Syntax

```
#include <rbf.h>
#include <types.h>
error_code _os_ss_cache(
    path_id      path,
    u_int32      enblflag,
    u_int32      drvcsize);
```

Description

`_os_ss_cache()` enables and disables RBF caching on an RBF device.

path	specifies the path number. (Input)
enblflag	is the cache enable/disable flag. (Input) <ul style="list-style-type: none">• If <code>enblflag</code> is zero, caching is disabled.• If <code>enblflag</code> is non-zero, caching is enabled.
drvcsize	is the memory size for the cache. (Input)

Attributes

Operating System:	OS-9
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os9_ss_close()

Notify Driver that Path Has Been Closed

Syntax

```
#include <sg_codes.h>
#include <types.h>
error_code _os9_ss_close(path_id path);
```

Description

`_os9_ss_close()` notifies a driver that a path has been closed.

path specifies the path number that was closed. (Input)

Attributes

Operating System:	OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

_os_ss_dcoff()

Set Signal to Be Sent when DCDL Goes False

Syntax

```
#include <scf.h>
#include <types.h>
error_code _os_ss_dcoff(
    path_id      path,
    signal_code  signal);
```

Description

`_os_ss_dcoff()` sets the signal code to be sent when the Data Carrier Detect line becomes false.

path	specifies the path number. (Input)
signal	is the signal code to send. (Input)

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

[`_os_ss_dcon\(\)`](#)

_os_ss_dcon()

Set Signal to Be Sent when DCDL Goes True

Syntax

```
#include <scf.h>
#include <types.h>
error_code _os_ss_dcon(
    path_id      path,
    signal_code  signal);
```

Description

`_os_ss_dcon()` sets the signal code to be sent when the Data Carrier Detect line becomes true.

path	specifies the path number. (Input)
signal	is the signal code to send. (Input)

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

[_os_ss_dcoff\(\)](#)

_os_ss_dopt()

Set Device Path Options

Syntax

```
#include <sg_codes.h>
#include <types.h>
error_code _os_ss_dopt(
    path_id      path,
    u_int32      size,
    void        *dopts);
```

Description

`_os_ss_dopt()` sets the initial (default) device path options (`dopts`). These options initialize new paths to the device.

path	specifies the path number. (Input)
size	specifies the size of the options area. (Input)
dopts	is a pointer to the default options for the device. (Input)

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_ss_dsrts()

Disable RTS Line

Syntax

```
#include <scf.h>
#include <types.h>
error_code _os_ss_dsrts(path_id path);
```

Description

`_os_ss_dsrts()` disables the RTS line on serial devices.

path specifies the path number. (Input)

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

I\$SetStt
I_SETSTAT, SS_DSRTS

OS-9 for 68K Technical Manual

OS-9 Technical Manual

Syntax

```
#include <scf.h>
#include <types.h>
error_code _os_ss_enrts(path_id path);
```

Description

`_os_ss_enrts()` enables the RTS line on serial devices.

path specifies the path number. (Input)

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

I\$SetStt

I_SETSTAT, SS_ENRTS

OS-9 for 68K Technical Manual

OS-9 Technical Manual

_os_ss_erase()

Erase Tape

Syntax

```
#include <sbf.h>
#include <types.h>
error_code _os_ss_erase(
    path_id      path,
    u_int32      blocks);
```

Description

`_os_ss_erase()` erases a portion of the tape.

path	specifies the tape drive's path number. (Input)
blocks	specifies the number of blocks to erase. (Input) <ul style="list-style-type: none">• If <code>blocks</code> is 0xffffffff, SBF erases until the end-of-tape is reached.• If <code>blocks</code> is positive, SBF erases the amount of tape equivalent to that number of blocks.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

Syntax

```
#include <rbf.h>
#include <types.h>
error_code _os_ss_fd(
    path_id      path,
    fd_stats     *fdinfo);
```

Description

`_os_ss_fd()` changes the file descriptor sector data. The path must be open for write. To change the file descriptor sector data, you must first read the sector, make the changes, and then write the sector.

path	specifies the path number. (Input)
fdinfo	is a pointer to the file descriptor's buffer. (Input)

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

_os_ss_fillbuff()

Fill Path Buffer with Data

Syntax

```
#include <scf.h>
#include <types.h>
error_code _os_ss_fillbuff(
    path_id      path,
    char         *buffer,
    u_int32       size);
```

Description

`_os_ss_fillbuff()` fills the input path buffer with the data in buffer.

path	specifies the path number. (Input)
buffer	is a pointer to the location of your buffer. (Input)
size	specifies the size of your buffer. (Input)

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_ss_flushmap()

Flush Cached Bit Map Information for RBF Device

Syntax

```
#include <rbf.h>
#include <types.h>
error_code _os_ss_flushmap(
                           path_id      path);
```

Description

`_os_ss_flushmap()` flushes the cached bit map information for an RBF device. This is normally only performed after the bit map on the disk is changed by a utility such as format.

path	is the path number for any file on the device or the entire device (for example, a path open to /h0@). (Input)
------	--

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_ss_hdlink()

Make Hard Link to Existing File

Syntax

```
#include <rbf.h>
#include <types.h>
error_code _os_ss_hdlink(
    path_id      path,
    char        **link_path);
```

Description

`_os_ss_hdlink()` creates a new directory entry. This directory entry points to the file descriptor block of an open file. `_os_ss_hdlink()` updates the pathlist pointer.

path	is the path number of an existing file. (Input)
link_path	The new name for the directory entry is passed at the location pointed to by link_path. (Input/Output) <code>_os_ss_hdlink()</code> also stores the updated pathlist pointer at the location pointed to by link_path.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

[_os_ss_link_adj\(\)](#)

Set Link Count of a File

Syntax

```
#include <sg_codes.h>
#include <types.h>
error_code _os_ss_link_adj(
    path_id      path,
    u_in32       fd_addr,
    int32        link_adj)
```

Description

This `I_SETSTAT` call sets the link count of a file. The link count of a file indicates the number of hard links in the file system that point to a particular file. This call can be used to correct the link count of files under corrupt file system conditions. File link counts can only be adjusted by super-user processes.

path	Input
fd_addr	Input
link_adj	Input

Attributes

Operating System:	OS-9
State:	User; System; I/O; IRQ
Threads:	Safe
Re-entrant:	Yes

Library

oslib.l

See Also

[_os_ss_hdlink\(\)](#)
[_os_setstat\(\)](#)

_os_ss_lock()

Lock Out Record

Syntax

```
#include <rbf.h>
#include <types.h>
error_code _os_ss_lock(
    path_id      path,
    u_int32      size);
```

Description

`_os_ss_lock()` locks out a section of the file from the current file pointer position up to the specified number of bytes.

`path` is the file's path number.

`size` is the size of the section to lockout.

- If `size` is zero, all locks are removed (record lock, EOF lock, and file lock).
- If `size` is `xffffffff`, the entire file is locked out regardless of the file pointer's location. This is a special type of file lock that remains in effect until released by an `_os_ss_lock()` with `size` set to zero, a read or write of zero bytes, or the file is closed.

Attributes

Operating System: OS-9 and OS-9 for 68K

State: User and System

Threads: Safe

Re-entrant: Yes

Library

`os_lib.l`

Syntax

```
#include <sg_codes.h>
#include <types.h>
error_code _os_ss_luopt(
    path_id      path,
    u_int32      size,
    void        *luopts);
```

Description

`_os_ss_luopt()` writes the logical unit options for a path to a buffer.

path	is the path number. (Input)
size	specifies the buffer size of the logical unit options area. (Input)
luopts	is a pointer to the logical unit options. (Input)

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os9_ss_open()

Notify Driver that Path Has Been Opened

Syntax

```
#include <modes.h>
#include <types.h>
error_code _os9_ss_open(path_id path);
```

Description

`_os9_ss_open()` is an internal call for drivers.

`path` specifies the path number.

Attributes

Operating System:	OS-9
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_ss_popt()

Write Option Section of Path Descriptor

Syntax

```
#include <sg_codes.h>
#include <types.h>
error_code _os_ss_popt(
    path_id      path,
    u_int32      size,
    void        *user_opts);
```

Description

`_os_ss_popt()` writes the option section of the path descriptor from the options buffer pointed to by `user_opts`.

path	is the path number. (Input)
size	specifies the buffer size. (Input)
user_opts	is a pointer to the options buffer. (Input)

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

[_os_gs_popt\(\)](#)

`I$SetStt`

`I_SETSTAT`

OS-9 for 68K Technical Manual
OS-9 Technical Manual

_os_ss_relea()

Release Device

Syntax

```
#include <sg_codes.h>
#include <types.h>
error_code _os_ss_relea(path_id path);
```

Description

`_os_ss_relea()` releases the device from any `_os_ss_sendsig()`, `_os_ss_dcon()`, or `_os_ss_dcoff()` request made by the calling process.

path is the path number of the device to release. (Input)

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

```
_os_ss_sendsig()
_os_ss_dcoff()
_os_ss_dcon()
```

Syntax

```
#include <sg_codes.h>
#include <types.h>
error_code _os_ss_rename(
    path_id      path,
    const char   *newname);
```

Description

`_os_ss_rename()` changes the file name in the file's associated directory entry to `newname`.

path	is the file's path number. (Input)
newname	is a pointer to the file's new name. (Input)

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_ss_reset()

Restore Head to Track Zero

Syntax

```
#include <sg_codes.h>
#include <types.h>
error_code _os_ss_reset(
                        path_id      path);
```

Description

For RBF, `_os_ss_reset()` directs the disk head to track zero. It is used for formatting and error recovery. For SBF, `_os_ss_reset()` rewinds the tape.

path is the path number of the device. (Input)

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_ss_reten()

Retention Pass on Tape Drive

Syntax

```
#include <sbf.h>
#include <types.h>
error_code _os_ss_reten(path_id path);
```

Description

`_os_ss_reten()` performs a retention pass on the tape drive.

`path` is the path number of the tape drive.
(Input)

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_ss_rfm()

Skip Tape Marks

Syntax

```
#include <sbf.h>
#include <types.h>
error_code _os_ss_rfm(
    path_id      path,
    int32        marks);
```

Description

`_os_ss_rfm()` skips the number of tape marks specified in `marks`.

`path` is the tape drive's path number. (Input)
`marks` specifies the number of tape marks to skip. (Input)

If `marks` is negative, the tape is rewound the specified number of marks.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_ss_sendsig()

Send Signal on Data Ready

Syntax

```
#include <sg_codes.h>
#include <types.h>
error_code _os_ss_sendsig(
    path_id      path,
    signal_code  signal);
```

Description

`_os_ss_sendsig()` sets up a signal to send to a process when an interactive device or pipe has data ready. `_os_ss_sendsig()` must be reset each time the signal is sent.

path specifies the path number of the process receiving the signal. (Input)

signal specifies the signal to send. (Input)

Attributes

Operating System: OS-9 and OS-9 for 68K

State: User and System

Threads: Safe

Re-entrant: Yes

Library

os_lib.l

See Also

_os_ss_relea()

_os_ss_size()

Set File Size

Syntax

```
#include <sg_codes.h>
#include <types.h>
error_code _os_ss_size(
    path_id      path,
    u_int32      size);
```

Description

`_os_ss_size()` sets the size of the file associated with `path` to the specified size.

<code>path</code>	is the file's path number. (Input)
<code>size</code>	is the size of the file in bytes. (Input)

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

Syntax

```
#include <sbf.h>
#include <types.h>
error_code _os_ss_skip(
    path_id      path,
    int32        blocks);
```

Description

`_os_ss_skip()` skips the specified number of blocks.

`path` is the path number of the tape drive.
(Input)

`blocks` specifies the number of blocks to skip.
(Input)

If `blocks` is negative, the tape is
rewound the specified number of blocks.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_ss_skipend()

Skip to End of Tape

Syntax

```
#include <sbf.h>
#include <types.h>
error_code _os_ss_skipend(path_id path);
```

Description

`_os_ss_skipend()` skips the tape to the end of data. This enables you to append data to tapes on cartridge-type tape drives.

path is the path number of the tape drive.
(Input)

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

_os_ss_ticks()

Wait Specified Number of Ticks for Record Release

Syntax

```
#include <rbf.h>
#include <types.h>
error_code _os_ss_ticks(
                        path_id      path,
                        u_int32      delay);
```

Description

`_os_ss_ticks()` may be used to cause an error (`EOS_LOCK`) to return to the user program if a file lock conflict still exists after the specified number of ticks have elapsed.

- | | |
|-------|---|
| path | is the path number. (Input) |
| delay | specifies the delay interval. (Input) <ul style="list-style-type: none">• A delay of zero (RBF's default) causes sleep until the record is released.• A delay of one indicates that if the record is not released immediately, an error is returned. If the high order bit is set, the lower 31 bits are converted from 256ths of a second to ticks before sleeping. |

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_ss_wfm()

Write Tape Marks

Syntax

```
#include <sbf.h>
#include <types.h>
error_code _os_ss_wfm(
    path_id      path,
    u_int32      marks);
```

Description

`_os_ss_wfm()` writes the specified number of tape marks.

path	specifies the tape drive's path number. (Input)
marks	specifies the number of tape marks to write. (Input)

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

Syntax

```
#include <rbf.h>
#include <types.h>
error_code _os_ss_wtrack(
    path_id          path,
    const void      *trkbuf,
    const void      *ilvtbl,
    u_int32          track,
    u_int32          head,
    u_int32          interleave);
```

Description

`_os_ss_wtrack()` causes a format track operation (used with most floppy disks) to occur. For hard or floppy disks with a “format entire disk” command, this formats the entire media only when the track number and side number are both zero.

path	is the path number. (Input)
trkbuf	is a pointer to the track buffer. (Input)
ilvtbl	is a pointer to the interleave table. (Input) The interleave table contains byte entries of LBNs ordered to match the requested interleave offset.
track	is the track number. (Input)
head	is the side number. (Input)
interleave	is the interleave value. (Input)

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

I\$SetStt

I_SETSTAT

OS-9 for 68K Technical Manual

OS-9 Technical Manual

Syntax

```
#include <settrap.h>
#include <types.h>
#include <regs.h>
error_code _os_strap(
                      u_int32      *excpt_stack,
                      strap        *init_tbl);
```

Description

`_os_strap()` enables user programs to catch “process local” program exceptions such as illegal instructions and divide-by-zeroes.

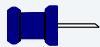
`excpt_stack` is a pointer to the stack to use if an exception occurs. (Input)

If `excpt_stack` is zero, `_os_strap()` uses the current stack.

`init_tbl` is a pointer to the service request initialization table. (Input)

If a user routine is not provided and one of these exceptions occurs, the program is aborted. An initialization table for a 68K platform might appear as follows:

```
strap errtab[ ] = {
    {T_BUSERR, errtrap},
    {T_ADDERR, errtrap},
    {-1, NULL}
};
```



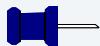
Note

The previous example is for the 68K platform only. The `T_BUSERR` and `T_ADDERR` are defined only for that platform, not for the PPC or X86 platforms.

When a user's exception routine is executed, it is passed the following:

```
void errtrap(vector_errno, badpc, badaddr)
    u_int32 vector_errno, * error number of the vector
                                * PC where exception occurred
    badpc,                  * address where exception
    badaddr;                * occurred
```

You can disable an error exception handler by calling `_os_strap()` with an initialization table that specifies zero as the offset to the routine(s) to remove.



Note

On OS-9 for 68K, the binding for this call performs conversions which allow compatibility with OS-9. If speed is a consideration, see `_os9_strap()`.



Note

Beware of exceptions in exception handling routines. They are usually not re-entrant.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

F\$STrap
F_STRAP

OS-9 for 68K Technical Manual
OS-9 Technical Manual

_os9_strap()

Set Error Trap Handler

Syntax

```
#include <settrap.h>
#include <regs.h>
#include <types.h>
error_code _os9_strap(
                      u_int32      *excpt_stack,
                      os9strap     *init_tbl);
```

Description

`_os9_strap()` enables user programs to catch “process local” program exceptions such as illegal instructions and divide by zeroes.

`excpt_stack` is a pointer to the stack to use if an exception occurs. (Input)

If `excpt_stack` is zero, `_os9_strap()` uses the current stack.

`init_tbl`

is a pointer to the service request initialization table. (Input)

The first field of the `init_tbl` should be the vector `OFFSET` of the exception vector on which to install the handler. The second field is the `OFFSET` from the next entry in `init_tbl` to the exception handling function for this vector.

`init_tbl` may look like the following:

```
void ovferror();  
void chkerror();  
extern void os9_init_tbl();  
  
os9strap *init_tbl = (os9strap  
*)os9_init_tbl;  
  
_asm("os9strap_init_tbl: dc.w %0,  
      ovferror-*-4", T_TRAPV << 2);  
_asm(" dc.w %0, chkerror-*-4", T_CHK  
<< 2);  
_asm(" dc.w -1, -1");
```

Note

To access the contents of the initialization table, `os9strap_init_tbl` in this example, you must define `os9_init_tbl` as if it were a function.

For example, `os9strap *os9_init_tbl` would not generate PC-relative references, while `extern void os9_init_tbl()` forces the compiler to generate references into the code area.

If a user routine is not provided and one of these exceptions occur, the program is aborted.

You can disable an error exception handler by calling `_os9_strap()` with an initialization table that specifies zero as the offset to the routine(s) to remove.



Note

Beware of exceptions in exception handling routines. They are usually not re-entrant.

Attributes

Operating System:	OS-9 for 68K
State:	User
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

F\$STrap

OS-9 for 68K Technical Manual

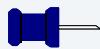
Syntax

```
#include <process.h>
#include <types.h>
error_code _os_suspend(process_id proc_id);
```

Description

`_os_suspend()` temporarily suspends a process. A super user (group ID zero) may suspend any process. A non-super user gets an `E_PERMIT` message returned.

`proc_id` identifies the target process. (Input)



Note

`_os_suspend()` is currently not supported on OS-9 for 68K.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_sysdbg()

Call System Debugger

Syntax

```
#include <process.h>
#include <types.h>
error_code _os_sysdbg(
    void      *param1,
    void      *param2);
```

Description

`_os_sysdbg()` calls the system level debugger, if one exists.

`param1` and `param2` are parameters passed to the debugger.
These are currently not used. (Input)

When the system level debugger is active, it runs in system state and effectively stops timesharing. When used in user state, `_os_sysdbg()` can only be called by users in group zero. When used in system state, anyone may use `_os_sysdbg()`. Never use this call when other users are on the system.

You must enable the system debugger before installing breakpoints or attempting to trace instructions. If no system debugger is available, the system is reset. The system debugger handles some of the exception vectors directly. This makes it impossible to use the user debugger when the system debugger is enabled.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_sysglobs()

Get pointer to system globals

Syntax

```
#include <regs.h>
void    *get_sysglobs(void);
```

Description

Returns a pointer to the system globals.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	System
Threads:	Safe
Re-entrant:	Yes

Library:

cpu.l

See Also

[sysglob.h](#)
[_os_getsys\(\)](#)
[_os_setsys\(\)](#)

Return System Identification Service Request

Syntax

```
#include <regs.h>
#include <types.h>
error_code _os_sysid(
    u_int32      *oem,
    u_int32      *serial,
    u_int32      *mpu_type,
    u_int32      *os_type,
    u_int32      *fpu_type,
    int32        *time_zone,
    u_int32      *kernel_ver,
    u_int32      *resv2,
    char         *sys_ident,
    char         *copyright,
    char         *author);
```

Description

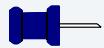
`_os_sysid()` returns the following information about the system:

Table 2-23 `_os_sysid()` Return Information

Pointer	Information Stored by <code>_os_sysid()</code>
<code>oem</code> (Output)	OEM identification number
<code>serial</code> (Output)	Copy serial number
<code>mpu_type</code> (Output)	Processor identifier (such as 68020/68030/68040/80386)

Table 2-23 _os_sysid() Return Information (continued)

Pointer	Information Stored by _os_sysid()
os_type (Output)	Kernel (OS) MPU configuration
fpu_type (Output)	Floating-point processor identifier (such as 68881 and 80387)
time_zone (Output)	System time zone in minutes offset from Greenwich Mean Time (GMT)
kernel_ver (Output)	The kernel's version number for OS-9 for 68K
resv2 (Output)	A reserved pointer
sys_ident (Output)	A pointer to an application buffer to copy the system identification message into. The buffer must be at least 80 bytes in length.
copyright (Output)	A pointer to an application buffer to copy the copyright message into. The buffer must be at least 80 bytes in length.
author (Output)	For OS-9: A pointer to an application buffer to copy the author message for OS-9. The buffer must be at least 80 bytes in length. For OS-9 for 68K: This parameter should be NULL.



Note

On OS-9 for 68K, `time_zone` is always returned as -1.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

Syntax

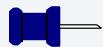
```
#include <threads.h>
error_code _os_thexit(error_code status);
```

Description

`_os_thexit()` causes the calling thread to exit. If the calling program is not multi-threaded, the `EOS_PERMIT` error is returned.

`status` is the value to be returned to any thread waiting for this thread via an `_os_waitid()` call. (Input)

If successful, `_os_thexit` does not return to the caller.



Note

Threads created via `pthread_create()` should not use this call. Doing so will result in instability and loss of resources for the process.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`, `mt_os_lib.l`

See Also

[_os_thfork\(\)](#)

_os_thfork()

Fork a Thread

Syntax

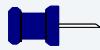
```
#include <threads.h>
error_code _os_thfork(
    thread_t          *thread,
    thread_attr_t     *attr,
    void              **stack_top,
    u_int32           *stack_size,
    void              *func,
    void              *arg,
    void              *data);
```

Description

`_os_thfork()` forks a new thread of control in the current process.

thread	is a pointer to a <code>thread_t</code> object which is used to return the newly created thread's ID. (Input)
attr	is a pointer to a thread attribute structure. (Input) The thread attribute structure is declared in <code>threads.h</code> .
stack_top	is a pointer to the top of user stack to be used by this thread— <code>stack_top</code> will be the initial stack pointer for the thread. (Input)
stack_size	If the value pointed to by <code>stack_size</code> is non-zero, the thread's user stack will be allocated by the system and a pointer to the allocated memory will be returned in the location pointed to by <code>stack_top</code> . The size of the actual stack allocated will be returned at <code>stack_size</code> .

	If <code>stack_size</code> points to zero, the value pointed to by <code>stack_top</code> will be used as the thread's stack. (Input/Output)
<code>func</code>	is the address of the initial function to be executed by the forked thread. (Input)
<code>arg</code>	is passed as the only argument to the function specified by <code>func</code> . (Input)
<code>data</code>	is the address of thread's thread-specific data. (Input)
	The pointer to void value pointed to by <code>data</code> is maintained on behalf of the thread. (Input)



Note

Threads created with `_os_thfork()` or `F_THFORK` are not permitted to use C library calls that have threading issues. Create threads with `pthread_create()`.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os.lib.l`, `mt_os.lib.l`

See Also

[`_os_thexit\(\)`](#)
[`_os_thread\(\)`](#)

_os_thread()

Set Thread Parameters

Syntax

```
#include <funcs.h>
#include <threads.h>
error_code _os_thread(
    u_int32    code,
    thread_t   thread_id,
    void       *arg);
```

Description

`_os_thread()` sets thread parameters for the thread specified by `thread_id` (input).

code	is the subfunction to perform. (Input)
	The legal values for <code>code</code> are <code>TH_ORPHAN</code> to orphan the thread and <code>TH_TSDATA</code> to set the thread specific data for the thread. Attempting to orphan an already orphaned thread will result in the error <code>EOS_THREAD_INVLD</code> .
arg	If <code>code</code> is <code>TH_TSDATA</code> , <code>arg</code> is the thread specific data pointer. (Input)



Note

Threads created via `pthread_create()` should not use this call.
Doing so will result in instability and loss of resources for the process.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os.lib.l, mt_os.lib.l`

See Also

[_os_thexit\(\)](#)
[_os_thfork\(\)](#)

_os_tlink()

Install User/System State Trap Handling Module

Syntax

```
#include <module.h>
#include <types.h>
error_code _os_tlink(
    u_int32      trap_num,
    const char   *mod_name,
    void         **lib_exec,
    mh_trap      **mod_head,
    void         *params,
    u_int32      mem_size);
```

Description

`_os_tlink()` attempts to link or load the specified module. If a trap module already exists for the specified trap number, an error is returned. If static storage is required for the trap handler, it is allocated and initialized.

trap_num	specifies the user trap number. (Input)
mod_name	On OS-9 for 68K valid trap numbers are in the range 1 through 15. On OS-9 valid trap numbers vary from processor to processor, but are generally in the range 0 through 15. See the include file in <code>/MWOS/OS9000/proc/DEFS</code> for the macros <code>TLINK_MIN</code> and <code>TLINK_MAX</code> .
lib_exec	is a pointer to the name of the trap module. (Input) If <code>mod_name</code> is zero or points to a null string, the trap handler is unlinked.
	is a pointer to the location where <code>_os_tlink()</code> stores the pointer to the trap execution entry point. (Output)

mod_head	is a pointer to the location where <code>_os_tlink()</code> stores the pointer to the trap module. (Output)
params	is a pointer to any additional parameters. (Input)
mem_size	Currently, this is not implemented. specifies the additional memory size. (Input)



Note

OS-9 only allows trap handlers to run in system state. OS-9 for 68K allows trap handlers to run in either system state or user state.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

_os_tlinkm()

Install User Trap Handling Module by Module Pointer

Syntax

```
#include <module.h>
#include <types.h>
error_code _os_tlinkm(
    u_int32      trap_num,
    mh_com       *mod_head,
    void         **lib_exec,
    void         *params,
    u_int32      mem_size);
```

Description

`_os_tlinkm()` is passed a pointer to the module to install. If a trap module already exists for the specified trap number, an error is returned. If static storage is required for the trap handler, it is allocated and initialized.

trap_num	specifies the user trap number (1 through 15). (Input)
mod_head	is a pointer to the module header. (Input)
lib_exec	is a pointer to the location where <code>_os_tlinkm()</code> stores the pointer to the trap execution entry point. (Output)
params	is a pointer to the additional parameters, if necessary. (Input)
mem_size	specifies the additional memory size. (Input)

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

_os_tranpn()

Translate User Path to System Path

Syntax

```
#include <io.h>
#include <types.h>
error_code _os_tranpn(
    process_id      proc_id,
    path_id         user_path,
    path_id         *sys_path);
```

Description

`_os_tranpn()` translates a user path number to a system path number. System-state processes use this call to access the user paths (standard I/O paths).

proc_id	specifies the process ID. (Input)
user_path	specifies the user path to translate. (Input)
sys_path	is a pointer to the location where <code>_os_tranpn()</code> stores the mapped system path. (Output)

Attributes

Operating System:	OS-9
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

Syntax

```
#include <virtual.h>
#include <types.h>
error_code _os_transadd(
    u_int32      *size,
    u_int32      mode,
    void        **blk_addr,
    void        *reserved);
```

Description

`_os_transadd()` translates an address to or from its external bus address.

size

The size of the block to translate is passed at the location pointed to by `size`. (Input/Output)

`_os_transadd()` also stores the size of the block translated at the location pointed to by `size`.

mode

specifies the type of translation. (Input)

- If `mode` is zero, the local CPU address is translated to the external bus address.
- If `mode` is one, the external bus address is translated to the local CPU address.

blk_addr	The address of the block to translate is passed at the location pointed to by blk_addr. (Input/Output) _os_transadd() also stores the block that was translated at the location pointed to by blk_addr. _os_transadd() is used when the external bus address must be passed to hardware devices, such as DMA-type controllers. If the specified source block is nonlinear with respect to its destination mapping, _os_transadd() returns the maximum number of bytes accessible at the translated address.
reserved	is a pointer reserved for future use. (Input)

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

Syntax

```
#include <memory.h>
#include <types.h>
error_code _os9_translate(
    u_int32      *size,
    u_int32      mode,
    void        **blk_addr);
```

Description

`_os9_translate()` translates an address to or from its external bus address.

size	The size of the block to translate is passed at the location pointed to by <code>size</code> . (Input/Output) <code>_os9_translate()</code> also stores the size of the block translated at the location pointed to by <code>size</code> .
mode	specifies the type of translation. (Input) <ul style="list-style-type: none">• If <code>mode</code> is zero, the local CPU address is translated to the external bus address.• If <code>mode</code> is one, the external bus address is translated to the local CPU address.
blk_addr	The address of the block to translate is passed at the location pointed to by <code>blk_addr</code> . (Output) <code>_os9_translate()</code> also stores the block that was translated at the location pointed to by <code>blk_addr</code> .

`_os9_translate()` is used when the external bus address must be passed to hardware devices, such as DMA-type controllers.

If the specified source block is nonlinear with respect to its destination mapping, `_os9_translate()` returns the maximum number of bytes accessible at the translated address.



Note

`_os9_translate()` is only available if colored memory is enabled.

Attributes

Operating System:	OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

Syntax

```
#include <process.h>
#include <types.h>
error_code _os_uacct(
    u_int32          acct_code,
    pr_desc          *proc_desc);
```

Description

`_os_uacct()` provides a means for users to set up an accounting system. `_os_uacct()` is called by the kernel whenever it forks or exits a process. This provides a mechanism for users to keep track of system operators.

To install a handler for this service request, the `_os_setsvc()` system call must be used to add the user's accounting routine to the system's service request dispatch table. This is usually done in an OS9P2 module.

You may perform your own system accounting by calling `_os_uacct()` with a user defined `acct_code` identifying the operation to perform. For example, when the kernel forks a process it identifies the operation by passing the `F_FORK` code to the accounting routine.

`acct_code` is the operation identifier. `reserved` is a pointer reserved for future use. (Input)

This is usually a system call function code. Only the lower 16 bits of `acct_code` are used.

`proc_desc` is a pointer to the current process descriptor. `reserved` is a pointer reserved for future use. (Input)

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

Syntax

```
#include <module.h>
#include <types.h>
error_code _os_unlink(mh_com *mod_head);
```

Description

`_os_unlink()` informs the system that the calling process no longer needs the module and decrements the module's link count.

`mod_head` is a pointer to the module header.
`reserved` is a pointer reserved for future use. (Input)

Note

Some modules, such as device drivers in use and all modules included in the bootfile, cannot be unlinked.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`os_lib.l`

See Also

`_os_unload()`
F\$UnLink
F_UNLINK

OS-9 for 68K Technical Manual
OS-9 Technical Manual

_os_unload()

Unlink Module by Name

Syntax

```
#include <module.h>
#include <types.h>
error_code _os_unload(
    const char      *mod_name,
    u_int32         type_lang);
```

Description

`_os_unload()` locates the module specified by `mod_name` in the module directory and decrements its link count.

<code>mod_name</code>	is a pointer to the module name. (Input)
<code>type_lang</code>	specifies the module's type and language. (Input)
	Only the lower 16 bits of <code>type_lang</code> are used.

Library

`os_lib.l`

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

See Also

[_os_unlink\(\)](#)

[F\\$UnLoad](#)

[F_UNLOAD](#)

OS-9 for 68K Technical Manual

OS-9 Technical Manual

Syntax

```
#include <module.h>
#include <types.h>
error_code _os_vmodul(
    mh_com           *mod_head,
    mh_com           *mod_block,
    u_int32          block_size);
```

Description

`_os_vmodul()` checks the module header parity and CRC bytes of an OS-9 module. If the header values are valid, the module is entered into the module directory. The current module directory is searched for another module with the same name. If a module with the same name and type exists, the module with the highest revision level is retained in the module directory. Ties are broken in favor of the established module.

mod_head	is a pointer to the address of the module. (Input)
mod_block	is a pointer to the memory block containing the module. (Input)
block_size	specifies the size of the memory block containing the module. (Input)

Attributes

Operating System:	OS-9
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

[_os_crc\(\)](#)
[_os_load\(\)](#)

Syntax

```
#include <module.h>
#include <types.h>
error_code _os9_vmodul(
    u_int32      mod_group,
    u_int32      size,
    mh_com       *mod_head,
    mod_dir     **dir_entry);
```

Description

`_os9_vmodul()` checks the module header parity and CRC bytes of an OS-9 module. If the header values are valid, the module is entered into the module directory, and a pointer to the directory entry is returned. The current module directory is searched for another module with the same name. If a module with the same name and type exists, the module with the highest revision level is retained in the module directory. Ties are broken in favor of the established module.

mod_group	specifies the beginning of the module group (ID). (Input)
size	is the size of the module. (Input)
mod_head	is a pointer to the module. (Input)
dir_entry	is a pointer to the location where <code>_os9_vmodul()</code> stores the pointer to the directory entry. (Output)

Attributes

Operating System:	OS-9 for 68K
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

[_os_crc\(\)](#)
[_os_load\(\)](#)

Syntax

```
#include <process.h>
#include <types.h>
error_code _os_wait(
    process_id      *child_id,
    status_code     *status);
```

Description

`_os_wait()` deactivates the calling process until a child process terminates by executing an `_os_exit()` system call or is otherwise terminated. The child's ID number and exit status are returned to the parent. If the child process died due to a signal, status contains the signal code.

<code>child_id</code>	is a pointer to the location where <code>_os_wait()</code> stores the process ID of the terminating child. (Output)
<code>status</code>	is a pointer to the location where <code>_os_wait()</code> stores the child process' exit status code. (Output)



Note

If a signal is received by a process waiting for children to terminate, the process is activated. In this case, `child_id` contains zero, because no child process has terminated.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

[_os_exit\(\)](#)
[_os_fork\(\)](#)
[_os_send\(\)](#)

F\$Wait
F_WAIT

OS-9 for 68K Technical Manual
OS-9 Technical Manual

Syntax

```
#include <process.h>
error_code _os_waitid(
    process_id      *child_id,
    error_code      *status,
    signal_code     *signal,
    u_int32         wait_flag);
```

Description

`_os_waitid()` has two primary functions:

- waiting for a child process or sibling thread
- controlling a signal for the death of a child process or sibling thread

Waiting for a Child Process or Sibling Thread

To specify a wait related activity, `wait_flag` should be 0.

<code>child_id</code>	The value pointed by <code>child_id</code> specifies the process or thread to wait for. If the value of <code>child_id</code> is the ID of a thread, the caller must be a thread in the same process as <code>child_id</code> . Otherwise an <code>EOS_IPRCID</code> error is returned. (Input)
<code>status</code>	If the call is successful, the exit code of <code>child_id</code> is returned in the location pointed to by <code>status</code> . (Output)
<code>signal</code>	If the wait is interrupted by a signal, a value of <code>EOS_BSIG</code> is returned by <code>_os_waitid()</code> and the value of the signal that caused the interruption is stored in the location pointed to by <code>signal</code> . (Output)

Controlling a Signal for the Death of a Child Process or Sibling Thread

`wait_flag`

To specify a signal related activity, `wait_flag` should be non-zero. The valid values for `wait_flag` are `WT_SIGNAL` and `WT_RELEASE`. (Input)

When `wait_flag` is `WT_SIGNAL` it specifies that the caller wants to receive a signal when the process or thread specified by `child_id` terminates. The value of the signal to be sent is pointed to by `signal`. If the process or thread specified by `child_id` has already terminated, the signal is sent immediately.

When `wait_flag` is `WT_RELEASE` it specifies that the caller is no longer interested in getting a signal on the termination of the process or thread specified by `child_id`. The value pointed to by `signal` is irrelevant in this case.

The value pointed to by `status` is not modified when `wait_flag` is non-zero.

`_os_waitid()` returns immediately when `wait_flag` is non-zero; it never blocks, regardless of the state of the child.

Attributes

Operating System:

OS-9

State:

User and System

Threads:

Safe

Re-entrant:

Yes

Library

`os_lib.l`, `mt_os_lib.l`

See Also

[_os_exit\(\)](#)
[_os_thexit\(\)](#)
[_os_wait\(\)](#)

_os_waitlk()

Activate Next Process Waiting to Acquire Lock

Syntax

```
#include <lock.h>
#include <types.h>
error_code _os_waitlk(
    lk_desc      *lock,
    signal_code  *signal);
```

Description

`_os_waitlk()` activates the next process waiting to acquire the lock. The next process in the lock's queue is activated and granted exclusive ownership of the resource lock. If no other process is waiting on the lock, the lock is simply marked free for acquisition. The calling process in either case is suspended and inserted into a waiting queue for the resource based on relative scheduling priority.

`lock` is a pointer to the lock on which to wait.
(Input)

`signal` is a pointer to the location where
`_os_waitlk()` stores the received
signal. (Output)

If a process receives a signal while waiting on a lock, the process is activated without gaining ownership of the lock.

The process returns from the wait lock call with an `EOS_SIGNAL` error code and the signal code is returned through the signal pointer.



Note

If an `S_WAKEUP` signal is received by a waiting process, the signal code does not register and is zero.

Attributes

Operating System:	OS-9
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

lock.l

See Also

[_os_acqlk\(\)](#)
[_os_caqlk\(\)](#)
[_os_crlk\(\)](#)
[_os_ddlk\(\)](#)
[_os_dellk\(\)](#)
[_os_rellk\(\)](#)
[F_WAITLK](#)

OS-9 Technical Manual

_os_write()

Write Data to File or Device

Syntax

```
#include <modes.h>
#include <types.h>
error_code _os_write(
    path_id      path,
    void         *buffer,
    u_int32      *count);
```

Description

`_os_write()` outputs bytes to a file or device associated with the specified path number. The path must have been opened or created in the write or update access modes.

path	is the specified path number for the file or device. (Input)
buffer	is a pointer to the data buffer. (Output)
count	The number of bytes to write is passed at the location pointed to by count. (Input/Output)
<code>_os_write()</code> also stores the number of bytes written at the location pointed to by count.	

Data is written to the file or device without processing or editing. If data is written past the present end-of-file, the file is automatically expanded.



Note

On RBF devices, any record that was locked is released.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

[_os_open\(\)](#)
[_os_create\(\)](#)
[_os9_create\(\)](#)
[_os writeln\(\)](#)

_os_writeln()

Write Line of Text with Editing

Syntax

```
#include <modes.h>
#include <types.h>
error_code _os_writeln(
    path_id      path,
    void         *buffer,
    u_int32      *count);
```

Description

`_os_writeln()` outputs bytes to a file or device associated with the specified path number. The path must have been opened or created in write or update access modes. `_os_writeln()` writes data until it encounters a carriage return character or `count` bytes. Line editing is also activated for character-oriented devices such as terminals and printers.

path	is the path number of the file or device. (Input)
buffer	is a pointer to the location where the bytes to be written are placed. (Output)
count	The number of bytes to write is passed at the location pointed to by <code>count</code> . (Input/Output)
	<code>_os_writeln()</code> also stores the number of bytes written at the location pointed to by <code>count</code> .



Note

On RBF devices, any record that was locked is released.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

[_os_open\(\)](#)
[_os9_create\(\)](#)
[_os_create\(\)](#)
[_os_write\(\)](#)

_os_yield()

Yield the Processor

Syntax

```
#include <process.h>
error_code _os_yield(void);
```

Description

`_os_yield()` causes the calling process or thread to be placed back into the active queue. The active queue contents are aged and the highest aged process is given control of the processor. In other words, `_os_yield()` causes the operating system to advance to the next executable process or thread. It is possible that the next executable process or thread will be the one that called `_os_yield()`. The status of the process' or thread's signal mask remains unchanged during this system call. `_os_yield()` is much like `_os_sleep()` with a tick count of 1, except that signals are not implicitly unmasked.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

os_lib.l

See Also

[_os_sleep\(\)](#)

Syntax

```
#include <regs.h>
#include <types.h>
void outc(
    void      *address,
    u_int32   data);
```

Description

outc() writes a character to an I/O address on the 80x86 processors or memory address on all other processors.

data	specifies the character to write. (Input)
	Only the lower 8 bits are used.
address	specifies the I/O or memory address. (Input)

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

cpu.l

See Also

[outl\(\)](#)
[outw\(\)](#)
[inc\(\)](#)
[inl\(\)](#)
[inw\(\)](#)

outl()

Write Integer to I/O Address

Syntax

```
#include <regs.h>
#include <types.h>
void outl(
    void      *address,
    u_int32  data);
```

Description

outl() writes a long value to the I/O address on the 80x86 or the memory address on all other processors.

data	specifies the long value to write. (Input)
address	specifies the I/O or memory address. (Input)

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

cpu.l

See Also

[outc\(\)](#)
[outw\(\)](#)
[inc\(\)](#)
[inl\(\)](#)
[inw\(\)](#)

Syntax

```
#include <regs.h>
#include <types.h>
void outw(
    void      *address,
    u_int32   data);
```

Description

`outw()` writes a word to the I/O address on the 80x86 or the memory address on all other processors.

data specifies the word to write. (Input)

 Only the lower 16 bits are used.

address specifies the I/O or memory address.
(Input)

Attributes

Operating System: OS-9

State: User and System

Threads: Safe

Re-entrant: Yes

Library

cpu.l

See Also

[outc\(\)](#)

[outl\(\)](#)

[inc\(\)](#)

[inl\(\)](#)

[inw\(\)](#)

_parsepath()

Parse Disk File (RBF) Pathlist

Syntax

```
#include <strings.h>
int _parsepath(const char *string);
```

Description

`_parsepath()` parses a disk file (RBF) pathlist. `_parsepath()` is useful for programs that must determine the validity of a pathlist name without actually creating or opening a file.

`string` is a pointer to the pathlist to parse.
(Input)

`_parsepath()` returns the number of bytes in the valid pathlist. If RBF does not accept the pathlist as valid, `_parsepath()` returns -1.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

sys_clib.l

See Also

_os_prsnam()

F\$PrsNam

F_PRSNAM

OS-9 for 68K Technical Manual

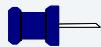
OS-9 Technical Manual

Syntax

```
#include <signal.h>
int pause(void);
```

Description

`pause()` suspends a task until a signal is received. `pause()` always returns -1 unless the received signal caused the process to terminate. In this case, `pause()` never returns.



Note

The thread enabled version of this function contains a cancel point. For more information see ***Using OS-9 Threads***.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`kill()`
`_os_sleep()`
`_os9_sleep()`
`sleep()`
`F$Sleep`
`F_SLEEP`

OS-9 for 68K Technical Manual
OS-9 Technical Manual

pclose()

Close a Pipe from/to a Process

Syntax

```
#include <UNIX/os9def.h>
#include <stdio.h>
int pclose(FILE *stream)
```

Description

`pclose()` closes a pipe from or to a process that was opened by `popen()`. `pclose()` waits for the associated process to terminate and returns the exit status of the command.

`stream` is a pointer to the file. (Input)

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`unix.l`

See Also

[popen\(\)](#)

Syntax

```
#include <stdio.h>
void perror(const char *chptr);
```

Description

`perror()` maps the error number in `errno` to an error message.

`chptr` is a pointer to a character. (Input)

If `chptr` is not a null pointer and the character pointed to by `chptr` is not a null character, `perror()` writes a sequence of characters to the standard error stream. The sequence written contains `chptr`, a colon (:), a space, and an appropriate error message string followed by a newline character.

The contents of the error message strings are the same as those returned by `strerror()` with parameter `errno`.

Attributes

Operating System: OS-9 and OS-9 for 68K

State: User and System

Threads: Safe

Re-entrant: Yes

Library

`clib.lib`

See Also

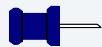
[strerror\(\)](#)

Syntax

```
int pffinit(void);
```

Description

`pffinit()` was used on the 6809 C compiler to allow `printf()` to perform float and double conversions. A dummy function named `pffinit()` exists here for 6809/68000 portability.



Note

`pffinit()` is historical and may be removed in a future release.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

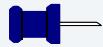
`sys_clib.l`

Syntax

```
int pflinit(void);
```

Description

`pflinit()` was used on the 6809 C compiler to allow `printf()` to perform long int conversions. A dummy function named `pflinit()` exists here for 6809/68000 portability.



Note

`pflinit()` is historical and may be removed in a future release.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

Create an Interprocess Communication Channel

Syntax

```
#include <UNIX/os9def.h>
int pipe(unsigned int fd[2]);
```

Description

pipe() creates an I/O mechanism called a pipe and returns 2 path numbers, fd[0] and fd[1].

fd[0] and fd[1]

These paths are open for read and write access. (Output)

When the pipe is written, up to 4 KB of data are buffered before the writing process is blocked. Data read from one of the paths accesses data written to either of the paths on a FIFO (first-in-first-out) basis.

The standard programming model is that after the pipe has been set up, two (or more) cooperating processes (created by subsequent _os_exec calls) passes data through the pipe using read and write I/O calls.

If an error occurs, the pipe() returns -1 and sets the global variable errno to indicate the error.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

unix.l

See Also

[pclose\(\)](#)
[popen\(\)](#)

popen()

Open a Pipe From/To a Process

Syntax

```
#include <UNIX/os9def.h>
#include <stdio.h>
FILE *popen(
            const char      *command,
            const char      *type);
```

Description

popen() opens a pipe from or to a process. The arguments to popen() are pointers to null-terminated strings containing, respectively, a shell command line and an I/O mode, either r for reading or w for writing. popen() creates a pipe between the calling process and the command to be executed. The value returned is a stream pointer such that you can write to the standard input of the command, if the I/O mode is w, by writing to the file stream. If the I/O mode was r, you can read from the standard output of the command by reading from the file stream. A stream opened by popen() should be closed with pclose() which waits for the associated process to terminate and returns the exit status of the command.

command	is the command written to. (Input)
type	is the character mode. (Input)

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

unix.l

See Also

[pclose\(\)](#)

[pipe\(\)](#)

[popen\(\)](#)

Syntax

```
#include <math.h>
double pow(
    double      x,
    double      y);
```

Description

pow() returns x (input) raised to the y (input) power. A domain error occurs, EDOM stored in errno, if x is negative. A domain error also occurs if both x and y are zero. A range error, ERANGE stored in errno, occurs on underflow and overflow. pow() returns HUGE_VAL for both errors.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

clib.l

Possible Errors

EDOM
ERANGE

_prealloc_rand()

Return No Error in Calls to Rand

Syntax

```
#include stdlib.h  
error_code _prealloc_rand(void);
```

Description

`_prealloc_rand` can be called to ensure that future calls to `rand` do not return an error.

`_prealloc_rand` returns `SUCCESS` if successful. Otherwise it returns an error number.

Attributes

Operating System:	OS-9
State:	User
Threads:	Safe
Re-entrant:	No

Library

`clib.l`

Syntax

```
#include <stdio.h>
int prerr(
            int      path,
            int      errnum);
```

Description

`prerr()` prints an error message corresponding to an error number on the standard error path.

path	is the path number of the error message file. (Input)
errnum	is the error number. (Input) It is divided into two numbers.

- **OS-9 for 68K:** Error numbers are 2 bytes wide. The most significant byte is the error's group number, and the least significant byte is the specific error within that group.
- **OS-9:** Error numbers are 4 bytes wide. The most significant 2 bytes are the error's group number, and the least significant 2 bytes are the specific error within that group.

`prer()` prints a message in the form “Error #ggg:sss” where `ggg` is the group number of the error and `sss` is the specific error number within the group. If `path` is non-zero, the opened error message file is searched for further text related to the error. If more text is available, it is printed.

Example

```
prerr(0, 216)                                /* function call */
Error #000:216                                 /* output */
prerr(errmsg_path, 216)                         /* function call */
Error #000:216 (E$PNNF) File not found.       /* output line 1 */
The pathlist does not lead to any known file.  /* output line 2 */
```

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

sys_clib.l

See Also

[_os_perr\(\)](#)

F\$PErr

OS-9 for 68K Technical Manual

_prgname()

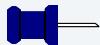
Get Module Name

Syntax

```
#include <module.h>
char *_prgname(void);
```

Description

`_prgname()` returns a pointer to the name of the calling module. Normally, the `argv[0]` string indicates the name of the program as called by the parent process with `os9exec()`. You can use `_prgname()` to determine the actual name of the module as it appears in the module directory.



Note

If the code that calls `_prgname()` is executing in a trap handler, the name of the trap handler module is returned.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

[_errormsg\(\)](#)
[os9exec\(\)](#)

Syntax

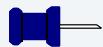
```
#include <stdio.h>
int printf(const char *format, ...);
```

Description

`printf()` is a standard C library function that performs formatted output to `stdout`. It converts, formats, and prints any parameters as indicated by the control string.

`format` is a pointer to a control string. (Input)

`printf()` returns the number of characters transmitted, or a negative value if an output error occurred.



Note

The `sclib.l` and `sclib.il` libraries contain a smaller version of the `printf()` function. Refer to the **Using Ultra C/C++** manual for more information about the small library functions.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`clib.l`

See Also

[fprintf\(\)](#)
[scanf\(\)](#)
[sprintf\(\)](#)

Syntax

```
#include <strings.h>
int _prsnam(const char *string);
```

Description

`_prsnam()` parses a path name segment.

`string` is a pointer to the path name segment.
(Input)

`_prsnam()` returns the number of characters in a valid path name segment. If the path name segment is invalid, -1 is returned. Use `_parsenam()` to determine if a path is a valid disk file (RBF) pathlist.

You can use successive calls to `_prsnam()` to parse a complete path name.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

[_os_prsnam\(\)](#)
`F$PrsNam`
`F_PRSNAM`

OS-9 for 68K Technical Manual
OS-9 Technical Manual

pthread_attr_destroy()

Free Thread Attribute Object

Syntax

```
#include <pthread.h>
int pthread_attr_destroy(pthread_attr_t *attr);
```

Description

`pthread_attr_destroy()` tells the library that a `pthread` attribute, `attr`, will no longer be used. The attribute, in effect, becomes uninitialized.

`attr` is a pointer to an initialized `pthread` attribute object. (Input)

If successful, returns a value of 0; otherwise, returns an error.

Attributes

Operating System:	OS-9
State:	User
Compatibility:	POSIX

Library

`mt_clib.l`

Possible Errors

<code>EINVAL</code>	an invalid <code>pthread_attr_t</code> pointer was passed
---------------------	---

See Also

[pthread_attr_init\(\)](#)
[pthread_create\(\)](#)

Example

```
err = pthread_attr_destroy(&attr);
if (err != 0)
    fprintf(stderr, "error destroying attribute - %s\n", strerror(err));
```

pthread_attr_getdetachstate()

Get Detach State Attribute

Syntax

```
#include <pthread.h>
int pthread_attr_getdetachstate(
    const pthread_attr_t *attr,
    int                  *detachstate);
```

Description

`pthread_attr_getdetachstate()` gets the detach state attribute, `attr`, in the attribute object. The integer pointed to by `detachstate` (input) will be written with `PTHREAD_CREATE_DETACHED` or `PTHREAD_CREATE_JOINABLE`.

`attr` is a pointer to an initialized `pthread` attribute object. (Input)

If successful, returns a value of 0; otherwise, returns an error.

Attributes

Operating System:	OS-9
State:	User
Compatibility:	POSIX

Library

`mt_clib.l`

Possible Errors

<code>EINVAL</code>	<code>attr</code> or <code>detachstate</code> is invalid or the object pointed to by <code>attr</code> is not properly initialized.
---------------------	---

See Also

[pthread_attr_init\(\)](#)
[pthread_attr_setdetachstate\(\)](#)
[pthread_create\(\)](#)
[pthread_detach\(\)](#)
[pthread_join\(\)](#)

Example

```
err = pthread_attr_getdetachstate(&attr, &state);
if (err != 0)
    fprintf(stderr, "error getting detach state - %s\n",
strerror(err));
```

[_pthread_attr_getinitfunction\(\)](#)

Get Initialization Function Attribute

Syntax

```
#include <pthread.h>
int _pthread_attr_getinitfunction(
    const pthread_attr_t *attr,
    int (**initfunc)(void *),
    void **initfunc_arg,
    void **initfunc_gp,
    void **initfunc_cp);
```

Description

`_pthread_attr_getinitfunction()` returns the initialization function pointer and initialization function argument fields from an attribute objects.

attr	is a pointer to an initialized pthread attribute object. (Input)
initfunc	points to a place to store the initialization function pointer. (Output)
initfunc_arg	points to a place to store the initialization function argument. (Output)
initfunc_gp	points to a place to store the initialization function global pointer. (Output)
initfunc_cp	points to a place to store the initialization function constant pointer. (Output)



For More Information

Refer to [_pthread_attr_setinitfunction\(\)](#) for more information about these fields.

Attributes

Operating System: OS-9
State: User

Library

mt_clib.l

Possible Errors

EINVAL attr does not refer to an initialized attributes object. initfunc or initfunc_arg is invalid.

See Also

[pthread_attr_init\(\)](#)
[_pthread_attr_setinitfunction\(\)](#)
[pthread_create\(\)](#)

Example

```
err = _pthread_attr_getinitfunction(&attr, &initfunc, &initfunc_arg, &gp, &cp);
if (err != 0)
    fprintf(stderr, "error getting initialization function - %s\n",
            strerror(err));
```

_pthread_attr_getpriority()

Get Priority Attribute

Syntax

```
#include <pthread.h>
int _pthread_attr_getpriority(
    const pthread_attr_t *attr,
    u_int32           *priority);
```

Description

`_pthread_attr_getpriority()` sets the `u_int32` pointed to by `priority` with the current priority setting from the specified `pthread` attribute object pointed to by `attr`. A value of 0 indicates that threads created with the specified attribute object will adopt the priority of the creating thread. A non-zero value indicates the desired priority for the created thread.

<code>attr</code>	is a pointer to an initialized <code>pthread</code> attribute object. (Input)
<code>priority</code>	is a pointer to the unsigned interger. (Input)

If successful, returns a value of 0; otherwise, returns an error.

Attributes

Operating System:	OS-9
State:	User

Library

`mt_clib.l`

Possible Errors

<code>EINVAL</code>	<code>attr</code> or <code>priority</code> is invalid or the object pointed to by <code>attr</code> is not properly initialized.
---------------------	--

See Also

[_pthread_attr_init\(\)](#)
[_pthread_attr_setpriority\(\)](#)
[_pthread_create\(\)](#)

Example

```
err = _pthread_attr_getpriority(&attr, &pr);
if (err != 0)
    fprintf(stderr, "error getting priority - %s\n",
strerror(err));
```

pthread_attr_getstackaddr()

Get Stack Address Attribute

Syntax

```
#include <pthread.h>
int pthread_attr_getstackaddr(
    const pthread_attr_t *attr,
    void                **stackaddr);
```

Description

`pthread_attr_getstackaddr()` gets the thread stack address attribute, `attr` (input), in the attribute object.

`pthread_attr_getstackaddr()` stores the thread stack address attribute value in `stackaddr` (output) if successful.

If successful, returns a value of 0; otherwise, returns an error.

Attributes

Operating System:	OS-9
State:	User
Compatibility:	POSIX

Library

`mt_clib.l`

Possible Errors

<code>EINVAL</code>	<code>attr</code> or <code>stackaddr</code> is invalid or the object pointed to by <code>attr</code> is not properly initialized.
---------------------	---

See Also

[pthread_attr_init\(\)](#)
[pthread_attr_setstackaddr\(\)](#)
[pthread_create\(\)](#)

Example

```
err = pthread_attr_getstackaddr(&attr, &stack);
if (err != 0)
    fprintf(stderr, "error getting stack address - %s\n",
strerror(err));
printf("Highest stack address is 0x%x\n", stack);
```

pthread_attr_getstacksize()

Get Stack Size Attribute

Syntax

```
#include <pthread.h>
int pthread_attr_getstacksize(
    const pthread_attr_t *attr,
    size_t *stacksize);
```

Description

`pthread_attr_getstacksize()` gets the thread stack size attribute, `attr` (input), in the attribute object.

`pthread_attr_getstacksize()` stores the thread stack size attribute value in `stacksize` (output) if successful.

If successful, returns a value of 0; otherwise, returns an error.

Attributes

Operating System:	OS-9
State:	User
Compatibility:	POSIX

Library

`mt_clib.l`

Possible Errors

<code>EINVAL</code>	<code>attr</code> or <code>stacksize</code> is invalid or the object pointed to by <code>attr</code> is not properly initialized.
---------------------	---

See Also

[pthread_attr_init\(\)](#)
[pthread_attr_setstacksize\(\)](#)
[pthread_create\(\)](#)

Example

```
err = pthread_attr_getstacksize(&attr, &size);
if (err != 0)
    fprintf(stderr, "error getting stack size - %s\n",
strerror(err));
printf("Stack size will be %u\n", size);
```

pthread_attr_init()

Allocate Thread Creation Attribute Object

Syntax

```
#include <pthread.h>
int pthread_attr_init(pthread_attr_t *attr);
```

Description

`pthread_attr_init()` sets default values into the `pthread` creation attribute object. The default values for a thread creation attribute object are shown in [Table 2-24](#):

Table 2-24 Default values for thread creation attribute

Attribute	Default Value
Stack Size	<code>PTHREAD_STACK_MIN</code>
Stack Address	<code>NULL</code> (system allocated stack)
Detach State	<code>PTHREAD_CREATE_JOINABLE</code>
Priority	0 (priority of creator)
Initialization Function	<code>NULL</code> (none)

If successful, returns a value of 0; otherwise, returns an error.

Attributes

Operating System: OS-9
State: User
Compatibility: POSIX

Library

`mt_clib.l`

Possible Errors

ENOMEM	Insufficient memory exists to initialize the attribute.
EINVAL	attr is invalid.

See Also

[pthread_create\(\)](#)

Example

```
err = pthread_attr_init(&attr);
if (err != 0)
    fprintf(stderr, "error initializing attribute - %s\n", strerror(err));
```

pthread_attr_setdetachstate()

Set Detached State Attribute

Syntax

```
#include <pthread.h>
int pthread_attr_setdetachstate(
    pthread_attr_t *attr,
    int             detachstate);
```

Description

`pthread_attr_setdetachstate()` sets the detach state attribute of the specified attribute object. Valid values for `detachstate` are `PTHREAD_CREATE_DETACHED` or `PTHREAD_CREATE_JOINABLE`.

Threads created as joinable retain information upon exit so that status can be returned when `pthread_join()` is used, unless `pthread_detach()` is used to detach the thread.

Threads created as detached automatically free all resources upon exit and cannot be used with `pthread_join()`. These type of threads are forked as “orphan” OS-9 threads.

If successful, returns a value of 0; otherwise, returns an error.

`attr` is a pointer to an initialized `pthread_attr_t` attribute object. (Output)

`detachstate` is the detach state attribute of the specified attribute object. (Input)

Attributes

Operating System:	OS-9
State:	User
Compatibility:	POSIX

Library

`mt_clib.l`

Possible Errors

EINVAL

attr or detachstate is not valid or attr is not properly initialized.

See Also

```
pthread_attr_init()  
pthread_attr_getdetachstate()  
pthread_create()  
pthread_detach()  
pthread_join()
```

Example

```
err = pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_DETACHED);
if (err != 0)
    fprintf(stderr, "error setting to detached state - %s\n", strerror(err));
```

_pthread_attr_setinitfunction()

Set Initialization Function Attribute

Syntax

```
#include <pthread.h>
int _pthread_attr_setinitfunction(
    const pthread_attr_t *attr,
    int (*initfunc)(void *),
    void *initfunc_arg,
    void *gp,
    void *cp);
```

Description

`_pthread_attr_setinitfunction()` sets the initialization function address, argument, global data and constant pointer fields of an attribute object.

attr	is a pointer to an initialized pthread attribute object. (Input)
initfunc	points to the initialization function. (Input)
initfunc_arg	is the argument to pass as the initialization functions sole argument. (Input)
gp	specifies the global data pointer that should be in place when calling initfunc. (Input)
cp	specifies the constant pointer that should be in place when calling initfunc. (Input)

If a constant pointer is not applicable for a particular processor or the code is compiled in such a way that a constant pointer is not needed, the value of `cp` may be `NULL`. Passing `NULL` as the `initfunc` parameter disables the calling of an initialization function. Passing `NULL` as the `initfunc_arg` parameter simply specifies that the value of the initialization function parameter should be `NULL`.

The initialization function has the following prototype:

```
int initfunc(void *initfunc_arg);
```

The value of the argument is the value of the `initfunc_arg` parameter in the thread's creation attributes object. If the initialization function returns a non-zero value, that value is converted to a pointer to void and passed to `pthread_exit()`, thus terminating the created thread without ever calling the intended start function.

The initialization function is called in the context of the created thread before the call to `pthread_create()` returns to its caller. The function may perform application specific thread initialization as necessary. The function could be useful in eliminating any race conditions that may exist between the creating thread and created thread since it is known that the initialization code will run in the created thread prior to the return from `pthread_create()`.

Although `pthread_self()` will function correctly, the value it returns should not be communicated to any other threads. The created thread has, technically, not finished its initialization, thus it is not ready to handle all thread operations. The initialization function should not interact with any other threads.

The initialization function runs at the priority of the creating thread, instead of the priority specified for the created thread. That is, if a high priority thread is creating a low priority thread with an initialization function, the initialization function will execute at high priority in the context of the low priority thread.

Attributes

Operating System:	OS-9
State:	User

Library

`mt_clib.l`

Possible Errors

<code>EINVAL</code>	<code>attr</code> does not refer to an initialized attributes object.
---------------------	---

See Also

[_pthread_attr_init\(\)](#)
[_pthread_attr_getinitfunction\(\)](#)
[_pthread_create\(\)](#)
[_pthread_exit\(\)](#)
[get_static\(\)](#)
[get_const\(\)](#)

Example

```
err = _pthread_attr_setinitfunction(&attr, thread_startup, &sema);
if (err != 0)
    fprintf(stderr, "error setting initialization function - %s\n",
            strerror(err));
```

_pthread_attr_setpriority()

Set Priority Attribute

Syntax

```
#include <pthread.h>
int _pthread_attr_setpriority(
                           pthread_attr_t *attr,
                           u_int32      priority);
```

Description

`_pthread_attr_setpriority()` sets the priority attribute of the pthread attribute object pointed to by `attr` to `priority`. A priority of 0 indicates that created threads should adopt the priority of the creating thread. A non-zero value specifies the desired priority for the created thread.

<code>attr</code>	is a pointer to an initialized pthread attribute object. (Input)
<code>priority</code>	is a pointer to the unsigned interger. (Input)

If successful, returns a value of 0; otherwise, returns an error.

Attributes

Operating System:	OS-9
State:	User

Library

`mt_clib.l`

Possible Errors

<code>EINVAL</code>	If <code>attr</code> is invalid or the object pointed to by <code>attr</code> is not properly initialized or the value of <code>priority</code> is out of range for a thread priority (0 to 65535).
---------------------	---

See Also

[_pthread_attr_init\(\)](#)
[_pthread_attr_getpriority\(\)](#)
[_pthread_create\(\)](#)
[_pthread_setpr\(\)](#)

Example

```
err = _pthread_attr_setpriority(&attr, 255);
if (err != 0)
    fprintf(stderr, "error setting priority - %s\n",
strerror(err));
```

pthread_attr_setstackaddr()

Set Stack Address Attribute

Syntax

```
#include <pthread.h>
int pthread_attr_setstackaddr(
    pthread_attr_t *attr,
    void           *stackaddr);
```

Description

`pthread_attr_setstackaddr()` allows a thread to specify a particular pre-allocated thread stack. The address specified is the desired stack pointer for the created thread. The specified stack must be at least `PTHREAD_STACK_MIN` in size.

`attr` is a pointer to an initialized `pthread_attr_t` attribute object. (Input)

`stackaddr` is a pointer to the pre-allocated thread stack. (Input)



Note

The `stackaddr` parameter is rounded down to an eight-byte boundary. To get the actual stack address used for created threads with a given attribute object. Use `pthread_attr_getstackaddr()` to get the actual address passed to the next created thread.

There is a matrix of possibilities for the two functions `pthread_attr_setstacksize()` and `pthread_attr_setstackaddr()`. Either one can be called independent of the other one being called. The behavior depends upon the following matrix, shown in **Table 2-25**.

Table 2-25 Function Behavior

Setstackaddr()	Setstacksize()	Resultant behavior
Not called	Not called	A system-allocated stack, of size <code>PTHREAD_STACK_MIN</code> , will be given to created threads.
Called	Not called	The specified stack address will be passed to created threads. The size will be assumed to be <code>PTHREAD_STACK_MIN</code> .
Not called	Called	A system-allocated stack of the size specified will be passed to created threads.
Called	Called	The specified stack address will be passed to created threads. The size will be assumed to be the size set by <code>pthread_attr_setstacksize()</code> .

Be aware of the following requirements when setting the stack address explicitly:

- The address passed to this function will be passed directly to threads created with this attribute object. Make sure that the top of the stack is passed (the highest RAM address of the stack).
- Do not create more than one thread with a given stack address.
- The stack should be "pre-loaded" with a NULL link pointer to ensure proper stack back-tracing.

If successful, returns a value of 0; otherwise, returns an error.

Attributes

Operating System:	OS-9
State:	User
Compatibility:	POSIX

Library

mt_clib.l

Possible Errors

EINVAL	attr is invalid or attr is not properly initialized.
--------	--

See Also

[pthread_attr_init\(\)](#)
[pthread_attr_getstackaddr\(\)](#)
[pthread_create\(\)](#)

Example

```
stack = malloc(PTHREAD_STACK_MIN);
if (stack == NULL)
    fprintf(stderr, "error allocating stack - %s\n", strerror(errno));
memset(stack, 0, PTHREAD_STACK_MIN);
err = pthread_attr_setstackaddr(&attr, stack + stacksize);
if (err != 0)
    fprintf(stderr, "error setting stack address - %s\n", strerror(err));
```

pthread_attr_setstacksize()

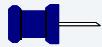
Set Stack Size Attribute

Syntax

```
#include <pthread.h>
int pthread_attr_setstacksize(
                    pthread_attr_t *attr,
                    size_t           stacksize);
```

Description

`pthread_attr_setstacksize()` sets the stack size that will be allocated for threads that are created with the specified attribute object.



Note

The `stacksize` parameter is rounded down to an eight-byte boundary. To get the actual stack size used for created threads with a given attribute object. Use `pthread_attr_getstacksize()` to get the actual stack size attribute used to create threads.

There is a matrix of possibilities for the two functions `pthread_attr_setstacksize()` and `pthread_attr_setstackaddr()`. Either one can be called independent of the other one being called. The behavior depends upon the following matrix, shown in **Table 2-26**.

Table 2-26 Function Behavior

Setstackaddr()	Setstacksize()	Resultant behavior
Not called	Not called	A system-allocated stack, of size <code>PTHREAD_STACK_MIN</code> , will be given to created threads.
Called	Not called	The specified stack address will be passed to created threads. The size will be assumed to be <code>PTHREAD_STACK_MIN</code> .
Not called	Called	A system-allocated stack of the size specified will be passed to created threads.
Called	Called	The specified stack address will be passed to created threads. The size will be assumed to be the size set by <code>pthread_attr_setstacksize()</code> .

If successful, returns a value of 0; otherwise, returns an error.

Attributes

Operating System:	OS-9
State:	User
Compatibility:	POSIX

Library

`mt_clib.l`

Possible Errors

EINVAL

`attr` is invalid, `attr` is not properly initialized or `stacksize` is less than PTHREAD_STACK_MIN.

See Also

[pthread_attr_init\(\)](#)
[pthread_attr_getstacksize\(\)](#)
[pthread_create\(\)](#)

Example

```
err = pthread_attr_setstacksize(&attr, 4096);
if (err != 0)
    fprintf(stderr, "error setting stack size - %s\n", strerror(err));
```

pthread_cancel()

Cancel Target Thread

Syntax

```
#include <pthread.h>
int pthread_cancel(pthread_t thread);
```

Description

`pthread_cancel()` cancels the target thread unless it is not currently cancelable. If the thread is not cancelable, the request is held pending until it reaches a cancellation point. The call to `pthread_cancel()` returns immediately regardless of the cancelability of the target thread.

If the specified thread has asynchronous cancels enabled it will terminate immediately without doing any sort of cleanup.

When a thread processes a deferred cancel the cleanup routines are called, thread specific data destructors are called, and the thread is terminated with the exit status `PTHREAD_CANCELED`.

`thread` is the target thread. (Input)



Note

Cancelling an asynchronous cancel type thread is guaranteed to cause a loss of resources. For example, the memory allocated to implement thread safety for C library functions will be lost. Use deferred cancellation whenever possible.

In addition, cancelling an asynchronous cancel type thread that is in a queue waiting for a resource will most likely cause the process to exit with an exception.

Attributes

Operating System:	OS-9
State:	User
Compatibility:	POSIX

Library

mt_clib.l

Possible Errors

ESRCH	No thread could be found corresponding to that specified by the given thread ID.
-------	--

See Also

[pthread_cond_timedwait\(\)](#)
[pthread_cond_wait\(\)](#)
[pthread_exit\(\)](#)
[pthread_join\(\)](#)
[pthread_setcancelstate\(\)](#)
[pthread_setcanceltype\(\)](#)

Example

```
err = pthread_cancel(worker);
if (err != 0)
    fprintf(stderr, "error cancelling worker - %s\n", strerror(err));
```

pthread_cleanup_pop()

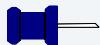
Pop Cleanup Routine

Syntax

```
#include <pthread.h>
void pthread_cleanup_pop(int execute);
```

Description

`pthread_cleanup_pop()` removes the routine at the top of the cancellation cleanup stack of the calling thread and invokes the popped thread if `execute` (input) is nonzero.



Note

`pthread_cleanup_pop()` and `pthread_cleanup_push()` have to be in the same lexical scope.

Attributes

Operating System:	OS-9
State:	User
Compatibility:	POSIX

Library

`mt_clib.l`

Possible Errors

None

See Also

[pthread_cancel\(\)](#)
[pthread_cleanup_push\(\)](#)
[pthread_setcancelstate\(\)](#)
[pthread_setcanceltype\(\)](#)

Example

```
pthread_cleanup_pop(1);      /* pop and call top cleanup function */
```

pthread_cleanup_push()

Push Cleanup Routine

Syntax

```
#include <pthread.h>
void pthread_cleanup_push(
    void (*routine)(void *),
    void *arg);
```

Description

`pthread_cleanup_push()` is similar to the ANSI `atexit()` function. It allows a thread to push a series of routines that should be called if the thread is terminated by `pthread_cancel()` or `pthread_exit()`. The routines are called in the reverse order that they were pushed onto the cleanup stack. That is, the most recently pushed routine is called first, followed by the next most recent, and so on.

Each `pthread_cleanup_push()` invocation must have an associated `pthread_cleanup_pop()` invocation in the same lexical scope. This is strictly enforced by having the `pthread_cleanup_push()` macro begin with an open brace ({}) and the `pthread_cleanup_pop()` macro end with a close brace (}).

Attributes

Operating System:	OS-9
State:	User
Compatibility:	POSIX

Library

`mt_clib.l`

Possible Errors

None

See Also

[pthread_cancel\(\)](#)
[pthread_cleanup_pop\(\)](#)
[pthread_setcancelstate\(\)](#)
[pthread_setcanceltype\(\)](#)

Example

```
err = pthread_mutex_lock(mutex);
if (err != 0)
    fprintf(stderr, "error locking mutex - %s\n", strerror(err));
pthread_cleanup_push(pthread_mutex_unlock, mutex);
err = pthread_cond_wait(condvar, mutex); /* cancellation point */
if (err != 0)
    fprintf(stderr, "error during cond_wait - %s\n", strerror(err));
pthread_cleanup_pop(1); /* unlock mutex */
```

pthread_cond_broadcast()

Release Threads Waiting for Condition Variable

Syntax

```
#include <pthread.h>
int pthread_cond_broadcast(pthread_cond_t *cond);
```

Description

`pthread_cond_broadcast()` releases every thread waiting on the specified condition variable.

If more than one thread is blocked on a condition variable, the OS-9 scheduler determines the order in which threads are activated. When each thread is unblocked it returns from its call to

`pthread_cond_wait()` or `pthread_cond_timedwait()`. The thread owns the mutex with which it called `pthread_cond_wait()` or `pthread_cond_timedwait()`. The thread(s) that are unblocked contend for the mutex in the normal fashion, as if each had called `pthread_mutex_lock()`.

`pthread_cond_broadcast()` may be called by a thread whether or not that thread currently owns the mutex that threads calling `pthread_cond_wait()` or `pthread_cond_timedwait()` have associated with the condition variable during their waits. However, if predictable scheduling behavior is required, then that mutex should be locked by the thread calling `pthread_cond_broadcast()`.

`pthread_cond_broadcast()` has no effect if there are no threads currently blocked on `cond`.

If successful, returns a value of 0; otherwise, returns an error.

cond	Input
------	-------

Attributes

Operating System:	OS-9
State:	User
Compatibility:	POSIX

Library

mt_clib.l

Possible Errors

EINVAL

The value cond does not refer to an initialized condition variable.

See Also

[pthread_cond_init\(\)](#)
[pthread_cond_timedwait\(\)](#)
[pthread_cond_wait\(\)](#)
[pthread_cond_signal\(\)](#)

Example

```
err = pthread_cond_broadcast(cond);
if (err != 0)
    fprintf(stderr, "failed to signal readers - %s\n", strerror(err));
err = pthread_mutex_unlock(data_lock);
if (err != 0)
    fprintf(stderr, "failed to unlock data lock - %s\n", strerror(err));
```

pthread_cond_destroy()

Free Condition Variable Object

Syntax

```
#include <pthread.h>
int pthread_cond_destroy(pthread_cond_t *cond);
```

Description

The function `pthread_cond_destroy()` destroys the given condition variable specified by `cond`; the object becomes, in effect, uninitialized.

If successful, returns a value of 0; otherwise, returns an error.

<code>cond</code>	is a pointer to the given condition variable. (Input)
-------------------	---

Attributes

Operating System:	OS-9
State:	User
Compatibility:	POSIX

Library

`mt_clib.l`

Possible Errors

<code>EBUSY</code>	An attempt to destroy the object referenced by <code>cond</code> while it is in use by another thread. For example, while being used in a <code>pthread_cond_wait()</code> or a <code>pthread_cond_timedwait()</code> .
<code>EINVAL</code>	The value specified by <code>cond</code> is invalid.

See Also

[pthread_cond_broadcast\(\)](#)
[pthread_cond_init\(\)](#)
[pthread_cond_signal\(\)](#)
[pthread_cond_timedwait\(\)](#)
[pthread_cond_wait\(\)](#)

Example

```
err = pthread_cond_destroy(&cond);
if (err != 0)
    fprintf(stderr, "failed to destroy condvar - %s\n", strerror(err));
```

pthread_cond_init()

Allocate Condition Variable Object

Syntax

```
#include <pthread.h>
int pthread_cond_init(
    pthread_cond_t           *cond,
    const pthread_condattr_t *attr);
```

Description

The function `pthread_cond_init()` initializes the condition variable referenced by `cond` with attributes referenced by `attr`. If `attr` is `NULL`, the default condition variable attributes are used; the effect is the same as passing the address of a default condition variable attributes object. Upon successful initialization, the state of the condition variable becomes initialized.

<code>attr</code>	is a pointer to an initialized <code>pthread_attr</code> attribute object. (Input)
<code>cond</code>	is a pointer to the condition variable. (Input)

If successful, returns a value of 0; otherwise, returns an error.

Attributes

Operating System:	OS-9
State:	User
Compatibility:	POSIX

Library

`mt_clib.l`

Possible Errors

EAGAIN	The system lacked the necessary resources (other than memory) to initialize another condition variable.
ENOMEM	Insufficient memory exists to initialize the condition variable.
EBUSY	An attempt to reinitialize the object referenced by <code>cond</code> (a previously initialized, but not yet destroyed, condition variable) has been detected.
EINVAL	The value specified by <code>cond</code> or <code>attr</code> is invalid.

See Also

[pthread_cond_broadcast\(\)](#)
[pthread_cond_destroy\(\)](#)
[pthread_cond_signal\(\)](#)
[pthread_cond_timedwait\(\)](#)
[pthread_cond_wait\(\)](#)
[pthread_condattr_init\(\)](#)

Example

```
err = pthread_cond_init(&cond, NULL);
if (err != 0)
    fprintf(stderr, "failed to initialize condvar - %s\n", strerror(err));
```

pthread_cond_signal()

Release Thread Waiting for Condition Variable

Syntax

```
#include <pthread.h>
int pthread_cond_signal(pthread_cond_t *cond);
```

Description

`pthread_cond_signal()` releases one thread waiting on the specified condition variable.

When the thread is unblocked it returns from its call to `pthread_cond_wait()` or `pthread_cond_timedwait()`. The thread owns the mutex with which it called `pthread_cond_wait()` or `pthread_cond_timedwait()`. The thread that is unblocked contends for the mutex in the normal fashion, as if it had called `pthread_mutex_lock()`.

`pthread_cond_signal()` may be called by a thread whether or not that thread currently owns the mutex that threads calling `pthread_cond_wait()` or `pthread_cond_timedwait()` have associated with the condition variable during their waits. However, if predictable scheduling behavior is required, then that mutex should be locked by the thread calling `pthread_cond_signal()`.

`pthread_cond_signal()` has no effect if there are no threads currently blocked on `cond`.

If successful, returns a value of 0; otherwise, returns an error.

`cond` is a pointer to the condition variable.
(Input)

Attributes

Operating System:	OS-9
State:	User
Compatibility:	POSIX

Library

mt_clib.l

Possible Errors

EINVAL

The value cond does not refer to an initialized condition variable.

See Also

[pthread_cond_init\(\)](#)
[pthread_cond_timedwait\(\)](#)
[pthread_cond_wait\(\)](#)
[pthread_cond_broadcast\(\)](#)

Example

```
err = pthread_cond_signal(cond);
if (err != 0)
    fprintf(stderr, "failed to signal worker - %s\n", strerror(err));
err = pthread_mutex_unlock(work_que_lock);
if (err != 0)
    fprintf(stderr, "failed to unlock work queue - %s\n", strerror(err));
```

pthread_cond_timedwait()

Wait on Condition Variable for Specified Interval

Syntax

```
#include <pthread.h>
int pthread_cond_timedwait(
    pthread_cond_t      *cond,
    pthread_mutex_t     *mutex,
    const struct timespec *abstime);
```

Description

`pthread_cond_timedwait()` is used to block on a condition variable until an absolute time is reached. It must be called with `mutex` locked by the calling thread or `EINVAL` will be returned.

`mutex` is a pointer to the mutex. (Input)

This function releases the mutex and causes the calling thread to block on the condition variable `cond`. If another thread is able to acquire the mutex after the about-to-block thread has released it, then a subsequent call to `pthread_cond_signal()` or `pthread_cond_broadcast()` in that thread behaves as if it were issued after the about-to-block thread has blocked.

`cond` is a pointer to the condition variable. (Input)

Upon return, the mutex is locked and is owned by the calling thread. When using condition variables, there is always a boolean predicate involving shared variables associated with each condition wait that is true if the thread should proceed. Spurious wakeups from the `pthread_cond_timedwait()` may occur. Since the return from `pthread_cond_timedwait()` does not imply anything about the value of this predicate, the predicate should be re-evaluated upon each return.

The effect of using more than one mutex for concurrent `pthread_cond_wait()` or `pthread_cond_timedwait()` operations on the same condition variable will result in `EINVAL` errors

being returned. That is, a condition variable becomes bound to a unique mutex when a thread waits on the condition variable, and this dynamic binding ends when the last concurrent wait returns.

A condition wait is a cancellation point. When the cancelability enable state of a thread is set to PTHREAD_CANCEL_DEFERRED, a side effect of acting upon a cancellation request while in a condition wait is that the mutex is re-acquired before calling the first cancellation cleanup handler. The effect is as if the thread were unblocked, allowed to execute up to the point of returning from the call to `pthread_cond_timedwait()`, but at that point notices the cancellation request and instead of returning to the caller of `pthread_cond_timedwait()`, starts the thread cancellation activities, which includes calling cancellation cleanup handlers.

A thread that has been unblocked because it has been canceled while blocked in a call to `pthread_cond_timedwait()` does not consume any condition signal that may be directed concurrently at the condition variable if there are other threads blocked on the condition variable.

The `timespec` pointed to by `abstime` specifies an absolute time in GMT that the call should return if the thread is not awakened by a `pthread_cond_signal()` or `pthread_cond_broadcast()`.

`abstime` is a pointer to the absolute time. (Input)

The Microware Pthread implementation supports the concept of interruption as it relates to condition variable waits. If a thread has a pending interruption or is interrupted while blocked, `pthread_cond_timedwait()` will return `EINTR`. The mutex will be re-acquired prior to return.

If successful, returns a value of 0; otherwise, returns an error.



Note

This function contains a cancel point.

Attributes

Operating System:	OS-9
State:	User
Compatibility:	POSIX

Library

mt_clib.l

Possible Errors

ETIMEDOUT	The time specified by abstime to pthread_cond_timedwait() has passed.
EINVAL	The value specified by cond, mutex, or abstime is invalid. Different mutexes are supplied for concurrent pthread_cond_wait() or pthread_cond_timedwait() operations on the same condition variable. The mutex is not owned by the current thread at the time of the call.

Additional Error

EINTR	_pthread_interrupt() was called with this thread as the target prior to or during this call.
-------	--

See Also

[pthread_cond_broadcast\(\)](#)
[pthread_cond_signal\(\)](#)
[pthread_cond_wait\(\)](#)

Example

```
err = _os_gettime(&tspec->tv_sec, &ticks);
if (err != SUCCESS)
    fprintf(stderr, "error getting GMT - %s\n", strerror(err));
tspec->tv_sec += 5; /* give up after 5 seconds */
tspec->tv_nsec = 0;
err = pthread_cond_timedwait(cond, mutex, tspec);
switch (err) {
case EINTR:
    fputs("timed wait interrupted\n", stderr);
    break;
case ETIMEDOUT:
    fputs("timed wait timed out\n", stderr);
    break;
default:
    fprintf(stderr, "error on timed wait - %s\n",
strerror(err));
    break;
}
```

pthread_cond_wait()

Wait on Condition Variable

Syntax

```
#include <pthread.h>
int pthread_cond_wait(
    pthread_cond_t      *cond,
    pthread_mutex_t     *mutex);
```

Description

`pthread_cond_wait()` is used to block on a condition variable. It must be called with `mutex` locked by the calling thread or `EINVAL` will be returned.

`mutex` is a pointer to the mutex. (Input)

This function releases the mutex and causes the calling thread to block on the condition variable `cond`. If another thread is able to acquire the mutex after the about-to-block thread has released it, then a subsequent call to `pthread_cond_signal()` or `pthread_cond_broadcast()` in that thread behaves as if it were issued after the about-to-block thread has blocked.

`cond` is a pointer to the condition variable. (Input)

Upon return, the mutex is locked and is owned by the calling thread. When using condition variables, there is always a boolean predicate involving shared variables associated with each condition wait that is true if the thread should proceed. Spurious wakeups from the `pthread_cond_wait()` may occur. Since the return from `pthread_cond_wait()` does not imply anything about the value of this predicate, the predicate should be re-evaluated upon each return.

The effect of using more than one mutex for concurrent `pthread_cond_wait()` or `pthread_cond_timedwait()` operations on the same condition variable will result in `EINVAL` errors being returned. That is, a condition variable becomes bound to a unique mutex when a thread waits on the condition variable, and this dynamic binding ends when the last concurrent wait returns.

A condition wait is a cancellation point. When the cancelability enable state of a thread is set to PTHREAD_CANCEL_DEFERRED, a side effect of acting upon a cancellation request while in a condition wait is that the mutex is re-acquired before calling the first cancellation cleanup handler. The effect is as if the thread were unblocked, allowed to execute up to the point of returning from the call to `pthread_cond_wait()`, but at that point notices the cancellation request and instead of returning to the caller of `pthread_cond_wait()`, starts the thread cancellation activities, which includes calling cancellation cleanup handlers.

A thread that has been unblocked because it has been canceled while blocked in a call to `pthread_cond_wait()` does not consume any condition signal that may be directed concurrently at the condition variable if there are other threads blocked on the condition variable.

The Microware Pthread implementation supports the concept of interruption as it relates to condition variable waits. If a thread has a pending interruption or is interrupted while blocked, `pthread_cond_wait()` will return `EINTR`. The mutex will be re-acquired prior to return.

If successful, returns a value of 0; otherwise, returns an error.



Note

This function contains a cancel point.

Attributes

Operating System:	OS-9
State:	User
Compatibility:	POSIX

Library

`mt_clib.l`

Possible Errors

EINVAL

The value specified by `cond` or `mutex` is invalid. Different mutexes are supplied for concurrent `pthread_cond_wait()` or `pthread_cond_timedwait()` operations on the same condition variable. The mutex is not owned by the current thread at the time of the call.

Additional Error

EINTR

`_pthread_interrupt()` was called with this thread as the target prior to or during this call.

See Also

[pthread_cond_broadcast\(\)](#)
[pthread_cond_signal\(\)](#)
[pthread_cond_timedwait\(\)](#)

Example

```
err = pthread_cond_wait(cond, mutx);
if (err != 0)
    fprintf(stderr, "failed to wait on condvar\n", strerror(err));
```

pthread_condattr_destroy()

Free Condition Variable Attributes Object

Syntax

```
#include <pthread.h>
int pthread_condattr_destroy(
    pthread_condattr_t *attr);
```

Description

`pthread_condattr_destroy()` destroys a condition variable attributes object; the object becomes, in effect, uninitialized.

If successful, returns a value of 0; otherwise, returns an error.

<code>attr</code>	is a pointer to an initialized <code>pthread_attr_t</code> attribute object. (Input)
-------------------	--

Attributes

Operating System:	OS-9
State:	User
Compatibility:	POSIX

Library

`mt_clib.l`

Possible Errors

<code>EINVAL</code>	The value specified by <code>attr</code> is invalid.
---------------------	--

See Also

[pthread_cond_init\(\)](#)
[pthread_condattr_init\(\)](#)

Example

```
err = pthread_condattr_destroy(attr);
if (err != 0)
    fprintf(stderr, "failed to destroy condattr - %s\n", strerror(err));
```

pthread_condattr_getpshared()

Get Condition Variable Process-Shared Attribute

Syntax

```
#include <pthread.h>
int pthread_condattr_getpshared(
    const pthread_condattr_t *attr,
    int                      *pshared);
```

Description

`pthread_condattr_getpshared()` obtains the value of the process-shared attribute from the attributes object referenced by `attr`.

If successful, returns 0 and stores the value of the process-shared attribute of `attr` into the object referenced by the `pshared` parameter. Otherwise, an error number is returned.

<code>attr</code>	is a pointer to an initialized <code>pthread_attr_t</code> attribute object. (Input)
<code>pshared</code>	is a pointer to the process-shared attribute. (Output)



Note

This facility is not currently supported in Microware's Pthreads implementation. The process-shared attribute can be changed, but both values behave like `PTHREAD_PROCESS_PRIVATE`.

`_POSIX_THREAD_PROCESS_SHARED` is not currently defined.

Attributes

Operating System:	OS-9
State:	User
Compatibility:	POSIX

Library

mt_clib.l

Possible Errors

EINVAL

The value specified by attr is invalid.

See Also

[pthread_condattr_init\(\)](#)
[pthread_condattr_setpshared\(\)](#)

Example

```
err = pthread_condattr_getpshared(attr, &pshare);
if (err != 0)
    fprintf(stderr, "failed to get pshared attribute - %s\n", strerror(err));
```

pthread_condattr_init()

Allocate Condition Variable Attributes Object

Syntax

```
#include <pthread.h>
int pthread_condattr_init(pthread_condattr_t *attr);
```

Description

`pthread_condattr_init()` initializes a condition variable attributes object `attr` with the default value for all of the attributes.

attr	is a pointer to an initialized <code>pthread</code> attribute object. (Output)
------	--

The default values of the attributes are shown in [Table 2-27](#).

Table 2-27 Default attribute values for condition variable attribute object

Attribute	Default Value
Process-shared	PTHREAD_PROCESS_PRIVATE

If successful, returns a value of 0; otherwise, returns an error.

Attributes

Operating System:	OS-9
State:	User
Compatibility:	POSIX

Library

`mt_clib.l`

Possible Errors

ENOMEM	Insufficient memory exists to initialize the condition variable attributes object.
EINVAL	<code>attr</code> is an invalid value.

See Also

[pthread_cond_init\(\)](#)
[pthread_condattr_destroy\(\)](#)

Example

```
err = pthread_condattr_init(&condattr);
if (err != 0)
    fprintf(stderr, "failed to init condattr - %s\n", strerror(err));
```

pthread_condattr_setpshared()

Set Condition Variable Process-Shared Attribute

Syntax

```
#include <pthread.h>
int pthread_condattr_setpshared(
                    pthread_condattr_t *attr,
                    int                  pshared);
```

Description

`pthread_condattr_setpshared()` sets the process-shared attribute in an initialized attributes object referenced by `attr`. Valid values for `pshared` are `PTHREAD_PROCESS_SHARED` or `PTHREAD_PROCESS_PRIVATE`.

If successful, returns a value of 0; otherwise, returns an error.

<code>attr</code>	is a pointer to an initialized <code>pthread_attr_t</code> attribute object. (Input)
<code>pshared</code>	is a pointer to the process-shared attribute. (Input)



Note

This facility is not currently supported in the Microware Pthreads implementation. The process-shared attribute can be changed, but `PTHREAD_PROCESS_SHARED` behaves exactly like `PTHREAD_PROCESS_PRIVATE`. `_POSIX_THREAD_PROCESS_SHARED` is not currently defined.

Attributes

Operating System:	OS-9
State:	User
Compatibility:	POSIX

Library

mt_clib.l

Possible Errors

EINVAL

The value specified by attr is invalid.
The new value specified for the attribute
is outside the range of legal values for
that attribute.

See Also

[pthread_condattr_init\(\)](#)
[pthread_condattr_getpshared\(\)](#)

Example

```
err = pthread_condattr_setpshared(&condattr, PTHREAD_PROCESS_PRIVATE);
if (err != 0)
    fprintf(stderr, "failed to set to private - %s\n", strerror(err));
```

pthread_create()

Create New Thread

Syntax

```
#include <pthread.h>
int pthread_create(
    pthread_t             *thread,
    const pthread_attr_t  *attr,
    void                  *(*start_routine) (void *),
    void                  *arg);
```

Description

`pthread_create()` is used to create a new thread, with attributes specified by `attr`, within a process. If `attr` is `NULL`, the default attributes are used. If the attributes specified by `attr` are modified later, the attributes of the thread are not affected. Upon successful completion, `pthread_create()` stores the ID of the created thread in the location referenced by `thread`.

`attr` is a pointer to an initialized `pthread_attr_t` attribute object. (Input)

`thread` is a pointer to the target thread. (Output)

The thread starts by executing `start_routine` with `arg` as its sole argument. If the `start_routine` returns, the effect is as if there was an implicit call to `pthread_exit()` using the return value of `start_routine` as the exit status.

`start_routine` is a pointer to the start thread routine. (Input)

`arg` is the sole argument for execution. (Input)

The thread in which `main()` was originally invoked differs from this. When this thread returns from `main()`, the effect is as if there was an implicit call to `exit()` using the return value of `main()` as the exit status.

If `pthread_create()` fails, no new thread is created, and the contents of the location referenced by `thread` are undefined.

The `pthread_attr` structure is used when threads are created.

If successful, returns a value of 0; otherwise, returns an error.

Attributes

Operating System:	OS-9
State:	User
Compatibility:	POSIX

Library

`mt_clib.l`

Possible Errors

EAGAIN	The system lacked the necessary resources to create another thread, or the system-imposed limit on the total number of threads in a process <code>PTHREAD_THREADS_MAX</code> would be exceeded.
EINVAL	The value specified by <code>attr</code> is invalid.

See Also

[_os_thfork\(\)](#)
[pthread_exit\(\)](#)
[pthread_join\(\)](#)
[pthread_detach\(\)](#)

Example

```
err = pthread_create(&tid, &worker_attr, worker_loop, NULL);
if (err != 0)
    fprintf(stderr, "error creating worker - %s\n", strerror(err));
```

pthread_detach()

Orphan Target Thread

Syntax

```
#include <pthread.h>
int pthread_detach(pthread_t thread);
```

Description

`pthread_detach()` orphans the designated thread. Any thread within the caller's process can be detached unless it is already in detached state.

The `pthread_detach()` function is used to indicate that storage for the thread can be reclaimed when that thread terminates. If thread has not terminated, `pthread_detach()` does not cause it to terminate. Multiple `pthread_detach()` calls on the same target thread result in `EINVAL` being returned.

If successful, returns a value of 0; otherwise, returns an error.

`thread` is the target thread. (Input)

Attributes

Operating System:	OS-9
State:	User
Compatibility:	POSIX

Library

`mt_clib.l`

Possible Errors

<code>EINVAL</code>	The value specified by <code>thread</code> does not refer to a thread that can be joined.
<code>ESRCH</code>	No thread could be found corresponding to that specified by the given thread ID.

See Also

[pthread_join\(\)](#)

Example

```
err = pthread_detach(io_thread);
if (err != 0)
    fprintf(stderr, "error detaching I/O thread - %s\n", strerror(err));
```

pthread_equal()

Compare Thread Identifiers

Syntax

```
#include <pthread.h>
int pthread_equal(pthread_t t1, pthread_t t2);
```

Description

`pthread_equal()` tests whether two thread IDs are the same.

`pthread_equal()` returns a nonzero value if the two thread IDs are equal; otherwise 0 is returned.

`t1` and `t2` are two thread IDs. (Input)

Attributes

Operating System:	OS-9
State:	User
Compatibility:	POSIX

Library

`mt_clib.l`

Possible Errors

None

See Also

[pthread_self\(\)](#)

Example

```
if (pthread_equal(worker[0], dead))
    fputs("worker #0 died", stderr);
```

Syntax

```
#include <pthread.h>
void pthread_exit(void *value_ptr);
```

Description

`pthread_exit()` terminates the calling thread. If any thread is waiting on a join on this thread, they are released and passed `value_ptr` as the exit status.

`value_ptr` is a pointer to the value. (Input)

`pthread_exit()` terminates the calling thread and makes the value `value_ptr` available to any successful join with the terminating thread. Any cancellation cleanup handlers that have been pushed and not yet popped, shall be popped in the reverse order that they were pushed and then executed. After all cancellation cleanup handlers have been executed, if the thread has any thread-specific data, appropriate destructor functions are called in an unspecified order. Thread termination does not release any application visible process resources (e.g. allocated memory, open paths, etc.). Nor does it perform any process level cleanup actions like calling any `atexit()` routines that may exist.

An implicit call to `pthread_exit()` is made when a thread other than the thread in which `main()` was first invoked returns from the start routine that was used to create it. The return value of the function serves as the exit status of the thread.

`pthread_exit()` returns immediately without doing anything if called from a cancellation cleanup handler or destructor function that was invoked as a result of either an implicit or explicit call to `pthread_exit()`.

After a thread has terminated, the result of access to local (auto) variables of the thread is undefined. Thus, references to local variables of the exiting thread should not be used for the `pthread_exit()` `value_ptr` parameter value.

The process exits with an exit status of 0 after the last thread has been terminated. The behavior is as if the implementation called `exit()` with a zero argument at the time of thread termination.

Calling `pthread_exit` from the thread in which `main` was first invoked does not necessarily cause the process to exit. The process will continue to run until all threads have terminated or an exit call is made.

Attributes

Operating System:	OS-9
State:	User
Compatibility:	POSIX

Library

`mt_clib.l`

Possible Errors

None

See Also

`exit()`
`pthread_create()`
`pthread_join()`
`pthread_detach()`

Example

```
pthread_exit((void *)SUCCESS);
```

pthread_getspecific()

Get Thread-Specific Data Pointer

Syntax

```
#include <pthread.h>
void *pthread_getspecific(pthread_key_t key);
```

Description

`pthread_getspecific()` returns the value currently bound to the specified key on behalf of the calling thread.

`pthread_getspecific()` returns the thread-specific data value associated with the given key. If no thread-specific data value is currently associated with key, then the value `NULL` is returned.

`t key` is the key associated with the calling thread. (Input)

Attributes

Operating System:	OS-9
State:	User
Compatibility:	POSIX

Library

`mt_clib.l`

Possible Errors

None

See Also

`pthread_key_create()`
`pthread_setspecific()`

Example

```
thread_data = pthread_getspecific(thread_data_key);
if (thread_data == NULL) {
    thread_data = malloc(sizeof(thread_data_t));
    if (thread_data == NULL)
        fprintf(stderr, "memory allocation error - %s\n",
strerror(errno));
    pthread_setspecific(thread_data_key, thread_data);
}
```

_pthread_getstatus()

Get Thread Status Information

Syntax

```
#include <pthread.h>
int _pthread_getstatus(
    pthread_t           thread,
    _pthread_status_t   *status);
```

Description

`_pthread_getstatus()` returns various pieces of information related to the target `thread` in the structure pointed to by `status`.

`thread` is the target thread. (Input)

`status` is a pointer to the status. (Output)

The following table describes the fields of the `_pthread_status_t` structure:

Table 2-28 `_pthread_status_t` Structure Fields

Type	Name	Description
u_int32 (bit masks follow)	<code>status</code>	Bits for various boolean pieces of information:
	<code>_PT_DETACHED</code>	
	0 = joinable thread	
	1 = detached thread	
	<code>_PT_EXIT</code>	
	0 = thread has terminated	
	1 = thread has not yet terminated	
	<code>_PT_CSTATE</code>	

Table 2-28 _pthread_status_t Structure Fields (continued)

Type	Name	Description
	_PT_CANCEL	0 = cancels enabled 1 = cancels disabled
	_PT_CTYPE	0 = deferred cancels 1 = asynchronous cancels
	_PT_CPENDING	0 = no cancel request pending 1 = cancel request pending
	_PT_SSTATE	0 = suspendable 1 = unsuspendable
	_PT_SPENDING	0 = no suspend request pending 1 = suspend request pending
	_PT_SFLAG	0 = not suspended 1 = suspended
	_PT_BOOSTED	0 = not priority boosted 1 = priority boosted
	_PT_IPENDING	

Table 2-28 _pthread_status_t Structure Fields (continued)

Type	Name	Description
0 = no interruption pending 1 = interruption pending		
thread_t	tid	OS-9 thread identifier of thread
thread_t	creator	OS-9 thread identifier of thread's creator
void *	stack	stack base (highest address)
size_t	stack_siz e	Stack size in bytes
u_int16	priority	Thread's priority
u_int16	bpriority	Thread's boosted priority
u_int32 [2]	resv	Reserved space for future additional status information

If successful, returns a value of 0; otherwise, returns an error.

Attributes

Operating System: OS-9
State: User

Library

mt_clib.l

Possible Errors

EINVAL	The passed thread or status pointer is NULL.
ESRCH	The specified target thread is not valid.

Example

```
err = _pthread_getstatus(child, &stats);
if (err != 0)
    fprintf(stderr, "failed to get status for child - %s",
strerror(err));
printf("child's OS-9 thread ID is %u\n", stats.tid);
```

_pthread_global_slot()

Read or Write Global Variables

Syntax

```
u_int32      * _pthread_global_slot(  
                  int32 slot)
```

Description

This function is used when reading or writing global versions of “core” C run-time variables. `_mainid`, the process ID of a thread’s host process, is the only example of such a variable.

`_pthread_global_slot()` is used to get the address of the global version of `_mainid`.

`slot` is the slot number. (Input)

Slot numbers are defined in MWOS/SRC/DEFS/pthread.h. Once a slot number has been assigned to a variable, it will not change in a subsequent release.

`_pthread_global_slot()` returns the address of the storage for a specific slot number. This makes it equally easy to read or write the variable. In addition, `_pthread_global_slot()` automatically saves and restores any modified registers except the return value. This makes it easier to call from assembly language.

A module’s global data pointer and `_pthread` value must be valid prior to calling `_pthread_global_slot()`

Attributes

Operating System:	OS-9
State:	User

Library

`mt_clib.l`

_pthread_interrupt()

Interrupt Target Thread

Syntax

```
#include <pthread.h>
int _pthread_interrupt(pthread_t thread);
```

Description

`_pthread_interrupt()` interrupts any `pthread_cond_wait()` or `pthread_cond_timedwait()` being done by the specified thread. If the thread is not currently blocked in `pthread_cond_wait()` or `pthread_cond_timedwait()`, `_pthread_interrupt()` makes the interruption pending.

`_pthread_interrupt()` is implemented as if the target thread can atomically check for a pending interrupt and then block in `pthread_cond_timedwait()` or `pthread_cond_wait()` if none is pending. That is, there is no window between when a thread checks for a pending interrupt and when the thread actually blocks where an interruption request could be missed.

If successful, returns a value of 0; otherwise, returns an error.

thread is the target thread. (Input)

Attributes

Operating System: OS-9
State: User

Library

mt_clib.l

Possible Errors

EINVAL thread is invalid.

ESRCH thread is not a valid thread.

See Also

[_pthread_cond_timedwait\(\)](#)
[_pthread_cond_wait\(\)](#)
[_pthread_getstatus\(\)](#)
[_pthread_interrupt_clear\(\)](#)

Example

```
err = _pthread_interrupt(wait_thread);
if (err != 0)
    fprintf(stderr, "failed to interrupt waiter - %s\n", strerror(err));
```

_pthread_interrupt_clear()

Clear Interrupt Request for Target Thread

Syntax

```
#include <pthread.h>
int _pthread_interrupt_clear(
                    pthread_t      thread,
                    int            *old_status);
```

Description

`_pthread_interrupt_clear()` clears any pending interrupt for the specified thread. This function might be useful if other interruptible operations are defined for a particular application. Refer to `_pthread_getstatus()` for more information on determining if a particular thread has an interruption pending.

thread	is the target thread. (Input)
old_status	The value of the interruption status for the target thread is returned at the integer pointed to by <code>old_status</code> . (Output)
	If the old status is not required, <code>NULL</code> may be passed for <code>old_status</code> .

If successful, returns a value of 0; otherwise, returns an error.

Attributes

Operating System:	OS-9
State:	User

Library

`mt_clib.l`

Possible Errors

EINVAL	thread is invalid.
ESRCH	thread is not a valid thread.

See Also

[pthread_cond_timedwait\(\)](#)
[pthread_cond_wait\(\)](#)
[_pthread_interrupt\(\)](#)
[_pthread_getstatus\(\)](#)

Example

```
err = _pthread_interrupt_clear(pthread_self());
if (err != 0)
    fprintf(stderr, "failed to clear interruption - %s\n", strerror(err));
```

pthread_join()

Wait for Target Thread to Terminate

Syntax

```
#include <pthread.h>
int pthread_join(
    pthread_t thread,
    void **value_ptr);
```

Description

The `pthread_join()` function suspends execution of the calling thread until the target thread terminates, unless the target thread has already terminated. On return from a successful `pthread_join()` call with a non-NULL `value_ptr` (output) argument, the value passed to `pthread_exit()` by the terminating thread (input) is stored in the location referenced by `value_ptr`.

When a `pthread_join()` returns successfully, the target thread has been terminated. Multiple simultaneous calls to `pthread_join()` specifying the same target thread results in one thread successfully getting the exit status and the remainder getting `EOS_NOCHLD` as the result of `pthread_join()`.

Exited but remaining unjoined threads count against the maximum number of threads a process may have, `PTHREAD_THREADS_MAX`.

If successful, returns a value of 0; otherwise, returns an error.



Note

This function contains a cancel point.

Attributes

Operating System:	OS-9
State:	User
Compatibility:	POSIX

Library

mt_clib.l

Possible Errors

EINVAL	The value specified by <code>thread</code> does not refer to a thread that can be joined.
ESRCH	No thread could be found corresponding to that specified by the given thread ID.
EDEADLK	A deadlock was detected, or the value of <code>thread</code> specifies the calling thread.

See Also

[pthread_create\(\)](#)
[pthread_detach\(\)](#)
[pthread_exit\(\)](#)

Example

```
err = pthread_join(child, &status);
if (err != 0)
    fprintf(stderr, "error waiting for child - %s\n", strerror(err));
printf("Child's exit status was %u\n", status);
```

_pthread_local_slot

Read or Write Thread-Specific Variables

Syntax

```
u_int32 *_pthread_local_slot(  
    int32 slot)
```

Description

This function is used when reading or writing thread-specific versions of “core” C run-time variables. `errno` is an example of a local slot. For threaded applications, there exists one `errno` per thread.

`_pthread_local_slot()` is used to get the address of the calling thread's version of `errno`.

slot is the slot number. (Input)

Slot numbers are defined in MWOS / SRC / DEFS / pthread.h. Once a slot number has been assigned to a variable, it will not change in a subsequent release.

`_pthread_local_slot()` returns the address of the storage for a specific slot number. This makes it equally easy to read or write the variable. In addition, `_pthread_local_slot()` automatically saves and restores any modified registers except the return value. This makes it easier to call from assembly language.

This might be used in the dispatcher during function exit to copy the version of `errno` generated by the code within a subroutine module back to the application's version of `errno`.

A module's global data pointer and `_pthread` value must be valid prior to calling `_pthread_local_slot()`.

Attributes

Operating System: OS
State: User

Library

mt_clib.l

pthread_key_create()

Create Thread-Specific Data Key

Syntax

```
#include <pthread.h>
int pthread_key_create(
    pthread_key_t *key,
    void (*destructor) (void *));
```

Description

`pthread_key_create()` creates a thread-specific data key visible to all threads in the process. Key values provided by `pthread_key_create()` are opaque objects used to locate thread-specific data. Although the same key value may be used by different threads, the values bound to the key by `pthread_setspecific()` are maintained on a per-thread basis and persist for the life of the calling thread.

If successful, `pthread_key_create()` stores the newly created key value at `*key` and returns 0. Otherwise, an error number is returned.

Attributes

Operating System:	OS-9
State:	User
Compatibility:	POSIX

Library

`mt_clib.l`

Possible Errors

EAGAIN	The system lacked the necessary resources to create another thread-specific data key, or the limit on the total number of keys per process, PTHREAD_KEYS_MAX, has been exceeded.
EINVAL	The key value is invalid.
ENOMEM	Insufficient memory exists to create the key.

See Also

[pthread_getspecific\(\)](#)
[pthread_key_delete\(\)](#)
[pthread_setspecific\(\)](#)

Example

```
err = pthread_key_create(&thread_data_key, free_thread_data);
if (err != 0)
    fprintf(stderr, "failed to create key - %s\n", strerror(err));
```

pthread_key_delete()

Delete Thread-Specific Data Key

Syntax

```
#include <pthread.h>
int pthread_key_delete(pthread_key_t key);
```

Description

`pthread_key_delete()` deletes a thread-specific data key (input) previously returned by `pthread_key_create()`. The thread-specific data values associated with key need not be `NULL` at the time `pthread_key_delete()` is called. It is the responsibility of the application to free any application storage or perform any cleanup actions for data structures related to the deleted key or associated thread-specific data in any threads; this cleanup can be done either before or after `pthread_key_delete()` is called.

If successful, returns a value of 0; otherwise, returns an error.

Attributes

Operating System:	OS-9
State:	User
Compatibility:	POSIX

Library

`mt_clib.l`

Possible Errors

<code>EINVAL</code>	The key value is invalid.
---------------------	---------------------------

See Also

[pthread_key_create\(\)](#)
[pthread_getspecific\(\)](#)
[pthread_setspecific\(\)](#)

Example

```
err = pthread_key_delete(thread_data_key);
if (err != 0)
    fprintf(stderr, "error deleting key - %s\n", strerror(err));
```

pthread_mutex_destroy()

Free Mutex Object

Syntax

```
#include <pthread.h>
int pthread_mutex_destroy(pthread_mutex_t *mutex);
```

Description

`pthread_mutex_destroy()` destroys the mutex object referenced by `mutex` (input); the mutex object becomes, in effect, uninitialized.

If successful, returns a value of 0; otherwise, returns an error.

Attributes

Operating System:	OS-9
State:	User
Compatibility:	POSIX

Library

mt_clib.l

Possible Errors

EBUSY	Attempt to destroy the object referenced by <code>mutex</code> while it is locked or referenced. For example, while being used in a <code>pthread_cond_wait()</code> or <code>pthread_cond_timedwait()</code> by another thread.
EINVAL	The value specified by <code>mutex</code> is invalid.

See Also

[pthread_mutex_init\(\)](#)

Example

```
err = pthread_mutex_destroy(mutex);
if (err != 0)
    fprintf(stderr, "error destroying mutex - %s\n",
strerror(err));
```

pthread_mutex_getprioceiling()

Get Mutex Priority Ceiling

Syntax

```
#include <pthread.h>
int pthread_mutex_getprioceiling(
    const pthread_mutex_t      *mutex,
    int                         *prioceiling);
```

Description

`pthread_mutex_getprioceiling()` obtains the value of the priority ceiling value from the mutex object referenced by `mutex` (input).

The value stored at `prioceiling` (output) will be the current value of the priority ceiling for the mutex. Valid priority ceilings are in the range 0 to 65535 (0xffff).

If successful, returns 0 and stores the value of the priority ceiling of `mutex` into the integer referenced by the `prioceiling` parameter. Otherwise, returns an error number.

Attributes

Operating System:	OS-9
State:	User
Compatibility:	POSIX

Library

`mt_clib.l`

Possible Errors

<code>EINVAL</code>	The value specified by <code>mutex</code> is invalid.
---------------------	---

See Also

[pthread_mutexattr_init\(\)](#)
[pthread_mutex_setprioceiling\(\)](#)

Example

```
err = pthread_mutex_getprioceiling(mutex, &pc);
if (err != 0)
    fprintf(stderr, "error getting priority ceiling - %s\n",
strerror(err));
```

pthread_mutex_init()

Allocate Mutex Object

Syntax

```
#include <pthread.h>
int pthread_mutex_init(
    pthread_mutex_t           *mutex,
    const pthread_mutexattr_t *attr);
```

Description

The `pthread_mutex_init()` function initializes the mutex referenced by `mutex` with attributes specified by `attr`.

`mutex` is a pointer to the mutex. (Input)

`attr` is a pointer to an initialized pthread attribute object. (Input)

If successful, returns a value of 0; otherwise, returns an error.

Attributes

Operating System: OS-9

State: User

Compatibility: POSIX

Library

`mt_clib.l`

Possible Errors

EAGAIN	The system lacked the necessary resources (other than memory) to initialize another mutex.
EBUSY	An attempt to reinitialize the object referenced by <code>mutex</code> (a previously initialized, but not yet destroyed, mutex).
EINVAL	The value specified by <code>attr</code> or <code>mutex</code> is invalid.

See Also

[pthread_mutex_lock\(\)](#)
[pthread_mutex_trylock\(\)](#)
[pthread_mutex_unlock\(\)](#)
[pthread_mutex_destroy\(\)](#)

Example

```
err = pthread_mutex_init(&glob_mutex, NULL);
if (err != 0)
    fprintf(stderr, "error initializing mutex - %s\n", strerror(err));
```

pthread_mutex_lock()

Lock Mutex Object

Syntax

```
#include <pthread.h>
int pthread_mutex_lock(pthread_mutex_t *mutex);
```

Description

The mutex object referenced by `mutex` (input) is locked by calling `pthread_mutex_lock()`. If the mutex is already locked, the calling thread blocks until the mutex becomes available. This operation returns with the mutex object referenced by `mutex` in the locked state with the calling thread as its owner. An attempt by the current owner of a mutex to relock the mutex results in an EDEADLK error.

If a signal is delivered to a thread waiting for a mutex, upon return from the signal handler the thread resumes waiting for the mutex as if it was not interrupted.

If priority inheritance is enabled for the specified mutex and a thread with a lower priority already owns the mutex then the owning thread's priority will be raised to the level of calling thread.

After the lock is acquired, if priority protection is enabled for the specified mutex and the specified ceiling priority is greater than the thread's current priority the thread's priority will be raised.

If successful, returns a value of 0; otherwise, returns an error.

Attributes

Operating System:	OS-9
State:	User
Compatibility:	POSIX

Library

`mt_clib.l`

Possible Errors

EINVAL	The value specified by <code>mutex</code> does not refer to an initialized mutex object.
EDEADLK	The current thread already owns the mutex.

See Also

[pthread_mutex_trylock\(\)](#)
[pthread_mutex_unlock\(\)](#)

Example

```
err = pthread_mutex_lock(&glob_mutex);
if (err != 0)
    fprintf(stderr, "error locking mutex - %s\n", strerror(err));
```

pthread_mutex_setprioceiling()

Set Mutex Priority Ceiling

Syntax

```
#include <pthread.h>
int pthread_mutex_setprioceiling(
    pthread_mutex_t      *attr,
    int                  ceiling);
```

Description

`pthread_mutex_setprioceiling()` is used to set the priority ceiling value in an initialized mutex object referenced by `mutex`.

If successful, returns a value of 0; otherwise, returns an error.

Attributes

Operating System:	OS-9
State:	User
Compatibility:	POSIX

Library

`mt_clib.l`

Possible Errors

EINVAL	The value specified by <code>mutex</code> is invalid. The new value specified for the attribute is outside the range of legal values for that attribute.
--------	--

See Also

[pthread_mutexattr_init\(\)](#)
[pthread_getprioceiling\(\)](#)

Example

```
err = pthread_mutex_setprioceiling(mutex, 255);
if (err != 0)
    fprintf(stderr, "error setting priority ceiling - %s\n", strerror(err));
```

pthread_mutex_trylock()

Lock Mutex Object (Non-Blocking)

Syntax

```
#include <pthread.h>
int pthread_mutex_trylock(pthread_mutex_t *mutex);
```

Description

`pthread_mutex_trylock()` is a non-blocking mutex lock operation. If `mutex` (input) is currently unowned, the calling thread is made the owner. If `mutex` is currently owned (by any thread, including the calling thread), `EBUSY` is returned.

Returns 0 if a lock on the mutex object referenced by `mutex` is acquired; otherwise, returns an error number.

Attributes

Operating System:	OS-9
State:	User
Compatibility:	POSIX

Library

`mt_clib.l`

Possible Errors

<code>EBUSY</code>	The mutex could not be acquired because it was already locked.
<code>EINVAL</code>	The value specified by <code>mutex</code> does not refer to an initialized mutex object.

See Also

[pthread_mutex_lock\(\)](#)
[pthread_mutex_unlock\(\)](#)

Example

```
err = pthread_mutex_trylock(&glob_mutex);
if (err != 0 && err != EBUSY)
    fprintf(stderr, "error trying to lock glob_mutex - %s\n",
strerror(err));
```

pthread_mutex_unlock()

Unlock Mutex Object

Syntax

```
#include <pthread.h>
int pthread_mutex_unlock(pthread_mutex_t *mutex);
```

Description

`pthread_mutex_unlock()` is called by the owner of the mutex object referenced by `mutex` (input) to release it. A `pthread_mutex_unlock()` call by a thread that is not the owner of the mutex results in an `EPERM` error. Calling `pthread_mutex_unlock()` when the mutex object is unlocked also results in an `EPERM` error.

If there are threads blocked on the mutex object referenced by `mutex` when `pthread_mutex_unlock()` is called, the mutex becomes available, and is given to the next waiting thread.

If successful, returns a value of 0; otherwise, returns an error.

Attributes

Operating System:	OS-9
State:	User
Compatibility:	POSIX

Library

`mt_clib.l`

Possible Errors

<code>EINVAL</code>	The value specified by <code>mutex</code> does not refer to an initialized mutex object.
<code>EPERM</code>	The current thread does not own the mutex.

See Also

[pthread_mutex_lock\(\)](#)
[pthread_mutex_trylock\(\)](#)

Example

```
err = pthread_mutex_unlock(&glob_mutex);
if (err != 0)
    fprintf(stderr, "error unlocking glob_mutex - %s\n", strerror(err));
```

pthread_mutexattr_destroy()

Free Mutex Attributes Object

Syntax

```
#include <pthread.h>
int pthread_mutexattr_destroy(
    pthread_mutexattr_t *attr);
```

Description

`pthread_mutexattr_destroy()` destroys a mutex attributes object; the object becomes, in effect, uninitialized.

If successful, returns a value of 0; otherwise, returns an error.

Attributes

Operating System:	OS-9
State:	User
Compatibility:	POSIX

Library

mt_clib.l

Possible Errors

EINVAL The value specified by `attr` is invalid.

See Also

[pthread_mutex_init\(\)](#)
[pthread_mutexattr_init\(\)](#)

Example

```
err = pthread_mutexattr_destroy(mutex_attr);
if (err != 0)
    fprintf(stderr, "error destroying attr - %s\n", strerror(err));
```

pthread_mutexattr_getprioceiling()

Get Priority Ceiling Attribute

Syntax

```
#include <pthread.h>
int pthread_mutexattr_getprioceiling(
    const pthread_mutexattr_t *attr,
    int                      *prioceiling);
```

Description

`pthread_mutexattr_getprioceiling()` obtains the value of the priority ceiling attribute from the mutex attributes object referenced by `attr` (input).

The value stored at `prioceiling` (output) will be the current value of the priority ceiling attribute. Valid priority ceilings are in the range 0 to 65535 (0xffff).

If successful, returns 0 and stores the value of the priority ceiling attribute of `attr` into the integer referenced by the `prioceiling` parameter. Otherwise, returns an error number.

Attributes

Operating System:	OS-9
State:	User
Compatibility:	POSIX

Library

`mt_clib.l`

Possible Errors

<code>EINVAL</code>	The value specified by <code>attr</code> is invalid.
---------------------	--

See Also

[pthread_mutexattr_init\(\)](#)
[pthread_mutexattr_setprioceiling\(\)](#)

Example

```
err = pthread_mutexattr_getprioceiling(mutex_attr, &pc);
if (err != 0)
    fprintf(stderr, "error getting priority ceiling - %s\n",
strerror(err));
```

pthread_mutexattr_getprotocol()

Get Protocol Attribute

Syntax

```
#include <pthread.h>
int pthread_mutexattr_getprotocol(
    const pthread_mutexattr_t *attr,
    int *protocol);
```

Description

`pthread_mutexattr_getprotocol()` obtains the value of the protocol attribute from the mutex attributes object referenced by `attr` (input).

The value stored at `protocol` will be one of `PTHREAD_PRIO_NONE`, `PTHREAD_PRIO_INHERIT`, or `PTHREAD_PRIO_PROTECT`.

If successful, returns 0 and stores the value of the protocol attribute of `attr` into the integer referenced by the `protocol` (output) parameter. Otherwise, returns an error number.

Attributes

Operating System:	OS-9
State:	User
Compatibility:	POSIX

Library

`mt_clib.l`

Possible Errors

<code>EINVAL</code>	The value specified by <code>attr</code> is invalid.
---------------------	--

See Also

[pthread_mutexattr_init\(\)](#)
[pthread_mutexattr_setprotocol\(\)](#)

Example

```
err = pthread_mutexattr_getprotocol(mutex_attr, &prot);
if (err != 0)
    fprintf(stderr, "error getting protocol - %s\n",
strerror(err));
```

pthread_mutexattr_getpshared()

Get Mutex Process-Shared Attribute

Syntax

```
#include <pthread.h>
int pthread_mutexattr_getpshared(
    const pthread_mutexattr_t *attr,
    int                      *pshared);
```

Description

`pthread_mutexattr_getpshared()` obtains the value of the process-shared attribute from the attributes object referenced by `attr` (input).

The value stored at `pshared` will be either `PTHREAD_PROCESS_SHARED` or `PTHREAD_PROCESS_PRIVATE`.

If successful, returns 0 and stores the value of the process-shared attribute of `attr` into the object referenced by the `pshared` (output) parameter. Otherwise, returns an error number.



Note

This facility is not currently supported in Microware's Pthreads implementation. The process-shared attribute can be changed, but both values behave like `PTHREAD_PROCESS_PRIVATE`.
`_POSIX_THREAD_PROCESS_SHARED` is not currently defined.

Attributes

Operating System:	OS-9
State:	User
Compatibility:	POSIX

Library

`mt_clib.l`

Possible Errors

EINVAL

The value specified by attr is invalid.

See Also

[pthread_mutexattr_init\(\)](#)
[pthread_mutexattr_setpshared\(\)](#)

Example

```
err = pthread_mutexattr_getpshared(mutex_attr, &pshared);
if (err != 0)
    fprintf(stderr, "error getting pshared - %s\n", strerror(err));
```

pthread_mutexattr_init()

Allocate Mutex Attributes Object

Syntax

```
#include <pthread.h>
int pthread_mutexattr_init(
    pthread_mutexattr_t *attr);
```

Description

`pthread_mutexattr_init()` initializes a mutex attributes object `attr` with a default value for all of the attributes.

`attr` is a pointer to an initialized `pthread_attr_t` attribute object. (Input)

The default values for the attributes are shown in [Table 2-29](#).

Table 2-29 Default attribute values for mutex attribute object

Attribute	Default Value
Process-shared	PTHREAD_PROCESS_PRIVATE
Protocol	PTHREAD_PRIO_NONE
Priority Ceiling	<none>

Returns 0 if successful or an error code if unsuccessful.

Attributes

Operating System: OS-9
State: User
Compatibility: POSIX

Library

`mt_clib.l`

Possible Errors

EINVAL	The key value is invalid.
ENOMEM	Insufficient memory exists to initialize the mutex attributes object.

See Also

[pthread_mutex_init\(\)](#)
[pthread_mutexattr_destroy\(\)](#)

Example

```
err = pthread_mutexattr_init(mutex_attr);
if (err != 0)
    fprintf(stderr, "error initializing attr - %s\n", strerror(err));
```

pthread_mutexattr_setprioceiling()

Set Priority Ceiling Attribute

Syntax

```
#include <pthread.h>
int pthread_mutexattr_setprioceiling(
    pthread_mutexattr_t *attr,
    int                 ceiling);
```

Description

`pthread_mutexattr_setprioceiling()` is used to set the priority ceiling attribute in an initialized attributes object referenced by `attr` (input).

`ceiling` (output) must be a valid OS-9 priority value; it must be in the range 0 to 65535 (0xffff).

If successful, returns a value of 0; otherwise, returns an error.

Attributes

Operating System:	OS-9
State:	User
Compatibility:	POSIX

Library

`mt_clib.l`

Possible Errors

<code>EINVAL</code>	The value specified by <code>attr</code> is invalid. The new value specified for the attribute is outside the range of legal values for that attribute.
---------------------	--

See Also

[pthread_mutexattr_init\(\)](#)
[pthread_mutexattr_getprioceiling\(\)](#)

Example

```
err = pthread_mutexattr_setprioceiling(mutex_attr, 255);
if (err != 0)
    fprintf(stderr, "error setting priority ceiling - %s\n",
strerror(err));
```

pthread_mutexattr_setprotocol()

Set Protocol Attribute

Syntax

```
#include <pthread.h>
int pthread_mutexattr_setprotocol(
    pthread_mutexattr_t *attr,
    int                  protocol);
```

Description

`pthread_mutexattr_setprotocol()` is used to set the protocol attribute in an initialized attributes object referenced by `attr` (input).

`protocol` (output) must be either `PTHREAD_PRIO_NONE`, `PTHREAD_PRIO_INHERIT`, or `PTHREAD_PRIO_PROTECT`.

If successful, returns a value of 0; otherwise, returns an error.

Attributes

Operating System:	OS-9
State:	User
Compatibility:	POSIX

Library

`mt_clib.l`

Possible Errors

`EINVAL`

The value specified by `attr` is invalid.
The new value specified for the attribute
is outside the range of legal values for
that attribute.

See Also

[pthread_mutexattr_init\(\)](#)
[pthread_mutexattr_getprotocol\(\)](#)

Example

```
err = pthread_mutexattr_setprotocol(mutex_attr, param);
if (err != 0)
    fprintf(stderr, "error setting protocol attr - %s\n",
strerror(err));
```

pthread_mutexattr_setpshared()

Set Mutex Process-Shared Attribute

Syntax

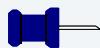
```
#include <pthread.h>
int pthread_mutexattr_setpshared(
    pthread_mutexattr_t *attr,
    int                  pshared);
```

Description

`pthread_mutexattr_setpshared()` is used to set the process-shared attribute in an initialized attributes object referenced by `attr` (input).

`pshared` (input) must be either `PTHREAD_PROCESS_SHARED` or `PTHREAD_PROCESS_PRIVATE`.

If successful, returns a value of 0; otherwise, returns an error.



Note

This facility is not currently supported in Microware's Pthreads implementation. The process-shared attribute can be changed, but both values behave like `PTHREAD_PROCESS_PRIVATE`.
`_POSIX_THREAD_PROCESS_SHARED` is not currently defined.

Attributes

Operating System:	OS-9
State:	User
Compatibility:	POSIX

Library

`mt_clib.l`

Possible Errors

EINVAL

The value specified by `attr` is invalid.
The new value specified for the attribute
is outside the range of legal values for
that attribute.

See Also

[pthread_mutexattr_init\(\)](#)
[pthread_mutexattr_getpshared\(\)](#)

Example

```
err = pthread_mutexattr_setpshared(mutex_attr, PTHREAD_PROCESS_PRIVATE);
if (err != 0)
    fprintf(stderr, "error setting pshared attr - %s\n", strerror(err));
```

pthread_once()

Execute Routine Once per Process

Syntax

```
#include <pthread.h>
int pthread_once(
    pthread_once_t *once_control,
    void           (*init_routine) (void));
```

Description

The first call to `pthread_once()` by any thread in a process with a given `once_control` (input) calls the `init_routine()` with no arguments. Subsequent calls of `pthread_once()` (input) with the same `once_control` will not call the `init_routine()`. On return from `pthread_once()` by any thread, it is guaranteed that `init_routine()` has completed. The `once_control` parameter is used to determine whether the associated initialization routine has been called.

`pthread_once()` is not a cancellation point. However, if `init_routine()` is a cancellation point and is canceled, the effect on `once_control` is as if `pthread_once()` was never called.

The behavior of `pthread_once()` is undefined if `once_control` has automatic storage duration or is not initialized by `PTHREAD_ONCE_INIT`.

If successful, returns a value of 0; otherwise, returns an error.

Attributes

Operating System:	OS-9
State:	User
Compatibility:	POSIX

Library

`mt_clib.l`

Possible Errors

EINVAL

once is an invalid pointer to a pthread_once_t object. once does not point to an initialized object. init_routine is an invalid address.

Example

```
err = pthread_once(get_key_once, create_data_key);
if (err != 0)
    fprintf(stderr, "error creating data key - %s\n", strerror(err));
```

_pthread_resume()

Decrement Suspension Counter

Syntax

```
#include <LIB/pthread.h>
int _pthread_resume(
                    pthread_t thread,
                    int *status);
```

Description

`_pthread_resume()` decrements the suspension counter for the specified target thread. The suspension status of the target thread is returned at the `int` pointed to by `status`. The `int` is as follows:

- 0 if the target thread was not suspended
- 1 if the target thread went from suspended to not suspended
- > 1 if the target thread remained suspended

A suspension counter is used to support multiple suspension requests with the same target thread. An equal number of resume requests must be made for the thread to continue execution.

If successful, returns a value of 0; otherwise, returns an error.

thread	is the target thread. (Input)
status	is a pointer to the suspension status of the target thread. (Output)

Attributes

Operating System:	OS-9
State:	User

Library

`mt_clib.l`

Possible Errors

EINVAL	thread or status is NULL.
ESRCH	thread is not a valid thread.

See Also

[_pthread_setsuspendable\(\)](#)
[_pthread_setunsuspendable\(\)](#)
[_pthread_suspend\(\)](#)

Example

```
err = _pthread_resume(worker, &level);
if (err != 0)
    fprintf(stderr, "error resuming worker - %s\n", strerror(err));
```

Syntax

```
#include <pthread.h>
pthread_t pthread_self(void);
```

Description

`pthread_self()` returns the calling thread's thread ID.

Attributes

Operating System:	OS-9
State:	User
Compatibility:	POSIX

Library

`mt_clib.l`

Possible Errors

None

See Also

[pthread_equal\(\)](#)

Example

```
if (pthread_self() == worker[0])
    fputs("thread is worker #0\n", stdout);
```

pthread_setcancelstate()

Set Cancel State

Syntax

```
#include <pthread.h>
int pthread_setcancelstate(
    int state,
    int *oldstate);
```

Description

`pthread_setcancelstate()` sets a thread's cancel state. `state` (input) can be either `PTHREAD_CANCEL_ENABLE` or `PTHREAD_CANCEL_DISABLE`. The previous value of the thread's cancel state is returned at `oldstate` (output).

Any cancel requests made against a thread while its state is `PTHREAD_CANCEL_DISABLE` will be held pending until the state is changed to `PTHREAD_CANCEL_ENABLE`.

If successful, returns a value of 0; otherwise, returns an error.

Attributes

Operating System:	OS-9
State:	User
Compatibility:	POSIX

Library

`mt_clib.l`

Possible Errors

<code>EINVAL</code>	<code>state</code> is neither <code>PTHREAD_CANCEL_ENABLE</code> nor <code>PTHREAD_CANCEL_DISABLE</code> . <code>oldstate</code> is an invalid address.
---------------------	--

See Also

[pthread_setcanceltype\(\)](#)
[pthread_cancel\(\)](#)
[pthread_cleanup_push\(\)](#)
[pthread_cleanup_pop\(\)](#)
[pthread_testcancel\(\)](#)

Example

```
err = pthread_setcancelstate(PTHREAD_CANCEL_DISABLE, &oldstate);
if (err != 0)
    fprintf(stderr, "error setting cancel state - %s\n", strerror(err));
```

pthread_setcanceltype()

Set Cancel Type

Syntax

```
#include <pthread.h>
int pthread_setcanceltype(
    int type,
    int *oldtype);
```

Description

`pthread_setcanceltype()` sets a thread's cancel type. `type` (input) can be either `PTHREAD_CANCEL_DEFERRED` or `PTHREAD_CANCEL_ASYNCHRONOUS`. The previous value of the thread's cancel type is returned at `oldtype` (output).

When a thread's cancel type is `PTHREAD_CANCEL_DEFERRED` cancel requests against it wait to take effect until the next call to `pthread_testcancel()`.

When a thread's cancel type is `PTHREAD_CANCEL_ASYNCHRONOUS` cancel requests are acted upon when they are made. That is, when a thread calls `pthread_cancel()` with a target thread that has cancellation enabled and asynchronous, the target thread will immediately cancel.

If successful, returns a value of 0; otherwise, returns an error.



Note

Cancelling an asynchronous cancel type thread is guaranteed to cause a loss of resources. For example, the memory allocated to implement thread safety for C library functions will be lost. Use deferred cancellation whenever possible.

Attributes

Operating System: OS-9
State: User
Compatibility: POSIX

Library

mt_clib.l

Possible Errors

EINVAL

`type` is neither
PTHREAD_CANCEL_DEFERRED nor
PTHREAD_CANCEL_ASYNCHRONOUS.

See Also

```
pthread_setcancelstate()  
pthread_cancel()  
pthread_cleanup_push()  
pthread_cleanup_pop()  
pthread_testcancel()
```

Example

```
err = pthread_setcanceltype(PTHREAD_CANCEL_DEFERRED, &oldtype);
if (err != 0)
    fprintf(stderr, "error setting cancel type - %s\n", strerror(err));
```

_pthread_setpr()

Set Priority for Target Thread

Syntax

```
#include <pthread.h>
int _pthread_setpr(
                    pthread_t thread,
                    u_int32 priority);
```

Description

`_pthread_setpr()` sets the OS-9 priority of `thread` (input) to `priority` (input). This call must be used by threaded applications instead of `_os_setpr()` to ensure that priority inversion avoidance is properly supported for mutexes. Calling `_os_setpr()` directly results in undefined behavior as it relates to priority inversion.

Use `_pthread_getstatus()` to determine the priority of a thread.

If successful, returns a value of 0; otherwise, returns an error.

Attributes

Operating System:	OS-9
State:	User

Library

`mt_clib.l`

Possible Errors

EINVAL	thread is invalid. priority is out of range. Valid range of priority is 0-65538.
ESRCH	thread is an invalid thread ID.

See Also

[_pthread_getstatus\(\)](#)
[pthread_mutex_destroy\(\)](#)
[pthread_mutex_destroy\(\)](#)
[_pthread_attr_setpriority\(\)](#)
[_pthread_attr_getpriority\(\)](#)

Example

```
err = _pthread_setpr(reactor_shutdown, HIGH_PRIORITY);
if (err != 0)
    fprintf(stderr, "failed to set priority - %s", strerror(err));
```

_pthread_setsignalrange()

Set Range of Signal Values

Syntax

```
#include <pthread.h>
int _pthread_setsignalrange(
    signal_code low,
    signal_code high);
```

Description

`_pthread_setsignalrange()` is used to specify the set of signal values that the Pthread library uses internally. Using this function will cause the Pthread library to use signals in the range `low` to (`high - 1`).

`low` and `high` are the low and high ranges of signals used in the Pthread library. (Input)

Use this function if your application uses the same set of signal values as the Pthread library. By default, the Pthread library will use signals in the range 40,000 to 49,999 inclusive.

A minimum of 1000 signal values must be specified. The Pthreads library uses about 5 signals per thread as well as 1 per timed condition variable wait.

The new set of signals may not overlap the current set of signal values. This is to ensure integrity of any already allocated signal numbers.

`_pthread_setsignalrange()` returns 0 if successful or an error code if not.

Attributes

Operating System:	OS-9
State:	User

Library

`mt_clib.l`

Possible Errors

EINVAL

If less than 1000 signal values are in the range or high is less than low or the specified range overlaps with the signal range currently in use.

Example

```
err = _pthread_setsignalrange(2000, 3500);
if (err != SUCCESS)
    fprintf(stderr, "error setting signal range - %s\n", strerror(err));
```

pthread_setspecific()

Set Thread-Specific Data Pointer

Syntax

```
#include <pthread.h>
int pthread_setspecific(
                    pthread_key_t key,
                    const void *value);
```

Description

`pthread_setspecific()` function associates a thread-specific value with a key (input) obtained via a previous call to `pthread_key_create()`. Different threads may bind different values to the same key. These values are typically pointers to blocks of dynamically allocated memory that have been reserved for use by the calling thread. `value` is an input.

If successful, returns a value of 0; otherwise, returns an error.

Attributes

Operating System:	OS-9
State:	User
Compatibility:	POSIX

Library

`mt_clib.l`

Possible Errors

<code>ENOMEM</code>	Insufficient memory exists to associate the value with the key.
<code>EINVAL</code>	The key value is invalid.

See Also

[pthread_key_create\(\)](#)
[pthread_getspecific\(\)](#)

Example

```
err = pthread_setspecific(thread_data_key, thread_data);
if (err != 0)
    fprintf(stderr, "error setting thread data - %s\n", strerror(err));
```

_pthread_setsuspendable()

Decrement Suspendability Counter

Syntax

```
#include <pthread.h>
int _pthread_setsuspendable(void);
```

Description

`_pthread_setsuspendable()` decrements the suspendability counter for the calling thread. When this counter is at 0, the thread is suspendable.

This call would be used by applications that contain thread suspension and resource locking. Before taking a common lock a thread would set itself unsuspendable. This prevents the thread from holding a common lock while it is in the suspended state. Holding a common lock while suspended could cause deadlock for the remaining unsuspended threads. After unlocking the common lock the thread would call this function to return itself to the suspendable state.

Calling this function from a suspendable thread yields no change in state.

Attributes

Operating System:	OS-9
State:	User

Library

`mt_clib.l`

Possible Errors

None

See Also

[_pthread_resume\(\)](#)
[_pthread_setunsuspendable\(\)](#)
[_pthread_suspend\(\)](#)

Example

The following example illustrates how to execute a semaphore protected critical section. Using the mechanisms shown below, a thread calling `_pthread_suspend()` can be assured that `glob_lock` will not be claimed by the suspended thread.

```
_pthread_setunsuspendable();

ec = _os_sema_p(&glob_lock);
if (ec != SUCCESS) {
    fprintf(stderr, "failed to get semaphore\n");
    pthread_exit((void *)ec);
}

/* critical section code */

ec = _os_sema_v(&glob_lock);
if (ec != SUCCESS) {
    fprintf(stderr, "failed to release semaphore\n");
    pthread_exit((void *)ec);
}

(pthread_setsuspendable());
```

_pthread_setunsuspendable()

Increment Suspendability Counter

Syntax

```
#include <pthread.h>
int _pthread_setunsuspendable(void);
```

Description

`_pthread_setunsuspendable()` increments the suspendability counter for the calling thread. When this counter is greater than 0, the thread is unsuspendable. This call does not return until the unsuspendable state is achieved.

This call is used by applications that contain thread suspension and resource locking. Before taking a common lock a thread would use this call to set itself unsuspendable. This prevents the thread from holding a common lock while it is in the suspended state. After unlocking the common lock the thread would call `_pthread_setsuspendable()` to return itself to the normal suspendable state.

Calling this function from a unsuspendable thread simply increases the suspendability counter. It is expected that each `_pthread_setunsuspendable()` call has a matching `_pthread_setsuspendable()` call.

Calling this function more than `0xffffffff` times without any intervening `_pthread_setsuspendable()` calls results in undefined behavior. Fewer `_pthread_setsuspendable()` calls than `_pthread_setunsuspendable()` calls will be required to return to the normal suspendable state.

Attributes

Operating System:	OS-9
State:	User

Library

`mt_clib.l`

Possible Errors

Errors from memory allocation and getting a process descriptor if called when signals are masked.

See Also

[_pthread_resume\(\)](#)
[_pthread_setsuspendable\(\)](#)
[_pthread_suspend\(\)](#)

Example

See the example provided for [_pthread_setunsuspendable\(\)](#).

_pthread_suspend()

Increment Suspension Counter

Syntax

```
#include <pthread.h>
int _pthread_suspend(
    pthread_t thread,
    unsigned int *count);
```

Description

`_pthread_suspend()` increments the suspension counter for the target thread specified by `thread` (input). The target thread's suspension counter prior to the suspension request is returned at the unsigned integer pointed to by `count` (output). A counter is used to support multiple suspension requests on the same target thread. An equal number of resume requests must be made before the target thread will resume execution.

This call does not return until the target thread has been successfully suspended. That is, if the target thread has set itself unsuspendable then this call will poll until the target sets itself back to suspendable.

Refer to the section on Thread Suspension for more information on what services are guaranteed while threads are suspended.

Returns 0 if the thread's suspension counter was successfully incremented or an error number if not.

Attributes

Operating System:	OS-9
State:	User

Library

`mt_clib.l`

Possible Errors

EINVAL	The specified thread or count pointer is NULL.
ESRCH	The specified thread is invalid or has terminated.
EDEADLK	The specified thread is the calling thread and there is only one thread in the process.

See Also

[_pthread_resume\(\)](#)
[_pthread_setsuspendable\(\)](#)
[_pthread_setunsuspendable\(\)](#)

Example

```
err = _pthread_suspend(child, &count);
if (err != 0) {
    fprintf(stderr, "failed to suspend child\n");
    pthread_exit((void *)err);
}

/* do some activity with child suspended */

err = _pthread_resume(child, &status);
if (err != 0) {
    fprintf(stderr, "failed to resume child\n");
    pthread_exit((void *)err);
}
```

pthread_testcancel()

Test for Pending Cancel

Syntax

```
#include <pthread.h>
void pthread_testcancel(void);
```

Description

`pthread_testcancel()` checks for a pending, deferred cancel request. If there is one, cancellation cleanup handlers are called in the reverse order in which they were pushed, thread specific data destructors are called in an unspecified order, and the thread is terminated with `PTHREAD_CANCELED` as its status.

If the cancel state of the thread is `PTHREAD_CANCEL_DISABLE`, this call has no effect.

`pthread_testcancel()` does not return if a cancel is pending.

Attributes

Operating System:	OS-9
State:	User
Compatibility:	POSIX

Library

`mt_clib.l`

Possible Errors

None

See Also

[pthread_cancel\(\)](#)
[pthread_setcancelstate\(\)](#)
[pthread_setcanceltype\(\)](#)

Example

```
pthread_testcancel();
```

Syntax

```
#include <stdio.h>
int putc(
    int      chr,
    FILE    *stream);
    int      putchar(int chr);
```

Description

putc() converts a character to an `unsigned char`. It writes the character to the output stream at the position indicated by the associated file position indicator for the stream, if defined, and advances the indicator appropriately. If the stream cannot support positioning requests or if the stream was opened with append mode, the character is appended to the output stream.

If it is implemented as a macro, it may evaluate the stream more than once. Therefore, the parameter should never be an expression with side effects.

putchar() is similar to putc() with the second parameter `stdout`.

`chr` is the character to write. (Input)

Only the lower 16 bits of `chr` are used.

`stream` is a pointer to the C I/O `FILE` structure. (Input)

putc() returns the character written. If a write error occurs, the error indicator for the stream is set and putc() returns EOF.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

clib.l

See Also

[fopen\(\)](#)

[fputc\(\)](#)

[getc\(\), getchar\(\)](#)

Syntax

```
#include <UNIX/os9def.h>
int putenv(const char *string)
```

Description

`putenv()` sets the value of an environment variable.

`string`

is in the form “`name=value`” where `name` is the name of the environment variable to set and `value` is the new value for the environment variable.
(Input)

If the variable named exists, its value is changed to `value`. If the variable name does not exist, a new environment list entry is created and set to the value passed. `putenv()` makes a copy of `string` so subsequent modifications to `string` do not affect the environment.

`putenv()` returns non-zero if it was unable to obtain enough space using `malloc` for an expanded environment, otherwise zero.

Note

`putenv()` manipulates the environment pointed to by `_environ`, and can be used in conjunction with `getenv()`. However, `envp` (the third argument to `main`) is not changed. This routine uses `malloc` to enlarge the environment.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

unix.l

Syntax

```
#include <stdio.h>
int puts(const char *strptr);
```

Description

`puts()` writes `strptr` to the stream pointed to by `stdout` and adds a newline character to the output. The terminating null character is not written.

`strptr` is a pointer to a string to write to `stdout`. (Input)

`puts()` returns `EOF` if an error occurs. Otherwise, it returns a non-negative value.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`clib.l`

See Also

[fgets\(\)](#)
[gets\(\)](#)

Syntax

```
#include <stdio.h>
int putw(
    int      num,
    FILE    *stream);
```

Description

`putw()` writes an integer, converted to a word, to a file. `putw()` neither assumes nor causes any special alignment in the file.

`num` is the integer to write to the structure `FILE`. (Input)

`stream` is a pointer to the file. (Input)

If successful, `putw()` returns the value it has written. Otherwise, it returns `-1`.



Note

`putw()`, like `getw()`, is machine-dependent because the size of the integer it outputs varies with the integer size of the machine on which it resides. This compiler defines `int` values as 4-byte quantities, but `putw()` actually outputs 2 bytes.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

sys_clib.l

See Also

[getw\(\)](#)

Syntax

```
#include <stdlib.h>
void qsort(
            void          *base,
            size_t        nmemb,
            size_t        size,
            int          (*compar)
                (const void *, const void *));
```

Description

qsort() sorts an array.

base	is a pointer to the initial array element. (Input)
nmemb	is the number of elements in the array. (Input)
size	is the size of each array element. (Input)
compar	is a pointer to a comparison function. (Output)

The contents of the array are sorted into ascending order according to compar. compar is called with two parameters that point to the objects being compared. compar returns an integer that is:

- Negative, if the first parameter is less than the second.
- Zero, if the first parameter is equal to the second.
- Positive, if the first parameter is greater than the second.

If two elements compare as equal, their order in the sorted array is unspecified. `qsort()` returns no value.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

raise()

Send Signal To Program

Syntax

```
#include <signal.h>
int raise(int sig);
```

Description

`raise()` sends a signal to the executing program.

sig specifies the signal to send. (Input)

`raise()` returns zero if successful. Otherwise, it returns a non-zero value.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

clib.l

Syntax

```
#include <stdlib.h>
int rand(void);
```

Description

`rand()` computes a sequence of pseudo-random integers in the range zero to `RAND_MAX` and returns a pseudo-random integer.

`rand()` uses the following algorithm:

```
static unsigned long int next = 1;
int rand(void)          * RAND_MAX assumed to be 32767
{
    next = next * 1103515245 + 12345;
    return (unsigned int) (next/65536) % 32768;
}
```

`next` contains the current internal value for calculating pseudo-random numbers. Because the value of `next` is divided by 65536 before returning the value, this algorithm avoids the common problem of non-random, low-order bits.

By storing the value in an `int`, this algorithm also allows the period of the sequence to be greater than `RAND_MAX`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`clib.l`

Syntax

```
#include <modes.h>
int read(
    int         path,
    char        *buffer,
    unsigned     count);
```

Description

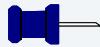
read() reads bytes from a path.

path	specifies the path number of the path from which to read. (Input)
	The path number is an integer indicating one of the standard path numbers (0, 1, or 2) or a path number returned from a successful call to open(), creat() , create() , or dup() .
buffer	is a pointer to a space with at least count bytes of memory into which read() puts the data read from the path. (Output)
count	is the minimum buffer size. (Input)

It is guaranteed that at most, count bytes are read, but often less are read, either because the path serves a terminal and input stops at the end of a line or the end-of-file has been reached.

read() essentially performs a **raw** read; that is, the read is performed without translating characters. The characters are passed to the program as read.

`read()` returns the number of bytes actually read (0 indicates the end of the file). If an error occurs, -1 is returned and the appropriate error code is placed in the global variable `errno`.



Note

End-of-file is returned as zero bytes being read, not an error indication.



Note

The thread enabled version of this function contains a cancel point. For more information see *Using OS-9 Threads*.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`creat()`
`create()`
`dup()`
`open()`
`read()`
`readln()`

`I$Read`
`I_READ`

OS-9 for 68K Technical Manual
OS-9 Technical Manual

Syntax

```
#include <dir.h>
struct direct *readdir(DIR *dirp);
```

Description

`readdir()` returns a pointer to a structure containing the next directory entry.

`dirp` is a pointer to the directory. (Input)

`readdir()` returns `NULL` upon reaching the end of the directory or detecting an invalid `seekdir()` operation.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

`sys_clib.l`

See Also

[closedir\(\)](#)
[opendir\(\)](#)
[rewinddir\(\)](#)
[seekdir\(\)](#)
[telldir\(\)](#)

Syntax

```
#include <modes.h>
int readln(
    int          path,
    char        *buffer,
    unsigned     count);
```

Description

`readln()` reads bytes from a path.

path specifies the path number of the file from which to read. (Input)

The path number is an integer indicating one of the standard path numbers (0, 1, or 2) or a path number returned from a successful call to `open()`, `creat()`, `create()`, or `dup()`.

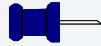
buffer is a pointer to a space with at least `count` bytes of memory into which `read()` puts the data read from the path. (Output)

count is the minimum buffer size. (Input)

It is guaranteed that at most, `count` bytes are read. Often less are read, either because the path serves a terminal and input stops at the end of a line or the end-of-file has been reached.

`readln()` causes line-editing, such as echoing, to take place and returns once a `\n` is read in the input or the number of bytes requested has been read. `readln()` is the preferred call for reading from the terminal.

`readln()` returns the number of bytes actually read (0 indicating end of file). If an error occurs, -1 is returned and the appropriate error code is placed in the global variable `errno`.



Note

End-of-file is returned as zero bytes being read, not an error indication.



Note

The thread enabled version of this function contains a cancel point. For more information see ***Using OS-9 Threads***.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

[creat\(\)](#)
[create\(\)](#)
[dup\(\)](#)
[open\(\)](#)
[_os_readln\(\)](#)
[read\(\)](#)
`I$ReadLn`
`I_READLN`

OS-9 for 68K Technical Manual
OS-9 Technical Manual

Syntax

```
#include <UNIX/os9def.h>
int readv(
    unsigned int      fd,
    struct iovec     *iov,
    unsigned int      iovcnt);
```

Description

`readv()` reads input, the same action as `read()`. The difference is that `readv()` scatters the input data into the `iovcnt` buffers specified by the members of the `iov` array:

`iov[0], iov[1], ...iov[iovcnt-1]`.

If `iovcnt` is zero, the value of the `iov_len` member in an `iovec` structure is less than zero, or all the `iov_len` members of the `iovec` structures are zero, `readv()` returns -1 and sets the global variable `errno`.

For `readv()`, the `iovec` structure is defined as:

```
struct iovec{
    caddr_t iov_base;
    int    iov_len
};
```

Each `iovec` entry specifies the base address and length of an area in memory where data should be placed. `readv()` always fills an area completely before proceeding to the next.

<code>fd</code>	Input
<code>iov</code>	Output
<code>iovcnt</code>	Input

Upon successful completion, `readv()` returns the number of bytes actually read and placed in the buffers.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

unix.l

Syntax

```
#include <stdlib.h>
void *realloc(
    void        *ptr,
    size_t      size);
```

Description

`realloc()` changes the size of a memory block. The contents of the object are unchanged up to the lesser of the new and old sizes. If the new size is larger, the value of the newly allocated portion of the object is indeterminate.

`ptr` is a pointer to the old block of memory.
(Input)

If `ptr` is a null pointer, `realloc()` behaves like `malloc()` for the specified size. Otherwise, if `ptr` does not match a pointer earlier returned by `calloc()`, `malloc()`, or `realloc()`, or if the space has been deallocated by a call to `free()`, `_lfree()`, `realloc()`, or `_lrealloc()`, the behavior is undefined.

`size` is the size of the new block of memory.
(Input)

If the space cannot be allocated, `ptr` is unchanged.

If `size` is zero and `ptr` is not a null pointer, the object `ptr` points to is freed.

`realloc()` returns either a null pointer or a pointer to the possibly moved allocated space.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

clib.l

See Also

[calloc\(\)](#)
[free\(\)](#)
[_lalloc\(\)](#)
[_lfree\(\)](#)
[_lrealloc\(\)](#)
[malloc\(\)](#)
[realloc\(\)](#)

Syntax

```
#include <UNIX/regexp.h>
regexp *regcomp(const char *str);
```

Description

`str` `regcomp()` compiles a string (`str`) into an internal form suitable for pattern matching with `regex()`. `regex()` matches a compiled regular expression against a string. (Input)

`regcomp()` returns a pointer to the compiled regular expression. Otherwise a string containing an error message is returned. If an error occurs, the function `regerror()` is called with a character string for the error and `NULL` is returned.

When the compiled regular expression is no longer needed, it can be passed to `free()`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

unix.l

See Also

[regerror\(\)](#)
[regex\(\), regexec\(\)](#)

reerror()

Regular Expression Error Handler

Syntax

```
#include <UNIX/regexp.h>
void regerror (const char *msg);
```

Description

`msg` `regerror()` is called by the functions `regex()` and `regcomp()` when various errors occur. `regerror()` prints the string "regexp: " and the string pointed to by `msg`. (Input)

If printing a message is inappropriate for the application using regular expressions, the programmer may define a `regerror()` function to override the `unix.l` version.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

unix.l

See Also

`regcomp()`
`regex()`, `regexec()`

Syntax

```
#include <UNIX/regexp.h>
int regex(
    regexp *re,
    char const *str);
```

Description

`re` and `str` `regex()` compares a string (`str`) with a compiled regular expression (`re`). (Input)

`regex()` returns one if the strings match and zero if the strings don't match. If an error occurs, `reerror()` is called with the appropriate error message and zero is returned to indicate the strings did not match.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: No

Library

unix.l

See Also

`regcomp()`
`regerror()`

Syntax

```
#include <stdio.h>
int remove(const char *filename);
```

Description

`remove()` deletes a file. Subsequent attempts to open a file using that name fail, unless the file is recreated.

`filename` is a pointer to the file to delete. (Input)
If the file is open, `remove()` returns `-1` with `errno` set to `EOS_SHARE`.

`remove()` returns zero if the operation succeeds. Otherwise, it returns a non-zero value.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`clib.l`

Syntax

```
#include <stdio.h>
int rename(
    const char *old,
    const char *new);
```

Description

`rename()` renames a file.

old	is a pointer to the name of an existing file. (Input)
new	is a pointer to the new name for the file. (Input)

The file named `old` is no longer accessible by that name. If a file named `new` exists prior to the call to `rename()`, it is overwritten.

`rename()` returns zero if the operation succeeds. Otherwise, it returns a non-zero value and the file name does not change.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`clib.l`

Syntax

```
#include <stdio.h>
void rewind(FILE *stream);
```

Description

`rewind()` sets the file position indicator for the stream pointed to by `stream` to the beginning of the file. It is equivalent to the following except that the error indicator for the stream is also cleared:

```
(void)fseek(stream, 0L, SEEK_SET)
;
stream           is a pointer to the C I/O file structure.
                  (Input)
rewind() returns no value.
```

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

clib.l

See Also

[fseek\(\)](#)

Syntax

```
#include <dir.h>
void rewinddir(DIR *dirp);
```

Description

`rewinddir()` resets the position of the named directory stream to the beginning of the directory.

`dirp` is a pointer to the directory stream.
(Input)

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

[closedir\(\)](#)
[opendir\(\)](#)
[readdir\(\)](#)
[seekdir\(\)](#)
[telldir\(\)](#)

rindex()

Search for Character in String

Syntax

```
#include <strings.h>
char *rindex(
    const char *ptr,
    int ch);
```

Description

`rindex()` returns a pointer to the last occurrence of a character in a string. If the character is not found, `rindex()` returns `NULL`.

<code>ptr</code>	s a pointer to the string to search. (Input)
<code>ch</code>	is the character for which to search. (Input)
	Only the lower 8 bits of <code>ch</code> are used.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

sys_clib.l

Example

The following example code fragment looks for a period (.) in the string and sets `ptr` to point to it. The search starts from the end of the string and progresses to the front of the string. Note that `ch` is a character, not a pointer to a character:

```
func( )
{
    char *ptr,*string;
    if((ptr = rindex(string,'.')) {
        process(ptr);
    } else {
        printf("No '.' found!\n");
    }
}
```

See Also

[index\(\)](#)

sbrk()

Extend Data Memory Segment

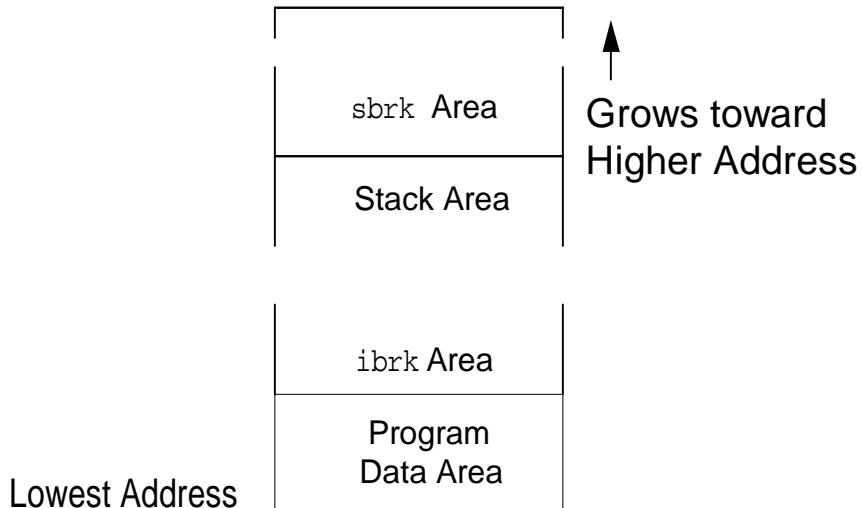
Syntax

```
#include <stdlib.h>
char *sbrk(unsigned size);
```

Description

`sbrk()` allocates memory from the top of the data area upwards.

Table 2-30 sbrk() Memory Allocation Table



size specifies the size of the memory block.
(Input)

`sbrk()` grants memory requests by calling the `F$MEM` (for OS-9) or `F_MEM` (for OS-9) system call. This method resizes the data area to a larger size. The new memory granted is contiguous with the end of the previous data memory.

On systems without an MMU, `sbrk()` fails quickly because it may keep growing in size until the data area reaches other allocated memory. At this point, it is impossible to increase in size and an error is returned. A program may increase its data size by only 20K, even if 200K is available elsewhere.

If an error occurs, -1 is returned and the appropriate error code is placed in the global variable `errno`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

`sys_clib.l`

See Also

[_os_mem\(\)](#)
[_os_srqmem\(\)](#)
`F$Mem`
`F_MEMORY`

OS-9 for 68K Technical Manual
OS-9 Technical Manual

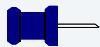
Syntax

```
#include <stdio.h>
int scanf(const char *format, ...);
```

Description

`scanf()` reads input from `stdin`.

format	is a pointer to the control string. (Input)
	<code>scanf()</code> returns <code>EOF</code> if an input failure occurs before any conversion.
	Otherwise, <code>scanf()</code> returns the number of input items assigned, which can be fewer than provided for, or zero in the event of an early matching failure.



Note

The `sclib.l` and `sclib.il` libraries contain a smaller version of the `scanf()` function. Refer to the ***Using Ultra C/C++*** manual for more information about the small library functions.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

clib.l

See Also

[fscanf\(\)](#)

[sscanf\(\)](#)

sched_yield()

Yield Processor

Syntax

```
#include <sched.h>
int sched_yield(void);
```

Description

`sched_yield()` gives up a thread's timeslice, allowing the kernel to reschedule the active threads. If there are other active threads at the same priority as the calling thread, one of them will be given the processor. The standard aging process occurs at every `sched_yield()` call.

`sched_yield()` works much like `_os_sleep()` with a `ticks` value of 1, except that it does not implicitly unmask signals.

If successful, 0 is returned, otherwise an error number is returned.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`_os_sleep()`
`tsleep()`

Example

```
while (not_ready)
    sched_yield();/* allow other threads to run */
```

Syntax

```
#include <dir.h>
void seekdir(
    DIR      *dirp,
    long     loc);
```

Description

`seekdir()` sets the position of the next `readdir()` operation on the directory stream to `loc`.

`dirp` is a pointer to the directory. (Input)

`loc` is the location for the next `readdir()` operation. (Input)

`loc` is a specific position in the directory stream returned by a previous call to `telldir()`.

Values returned by `telldir()` are valid only for the lifetime of the associated `dirp` pointer. If the directory is closed and then reopened, the `telldir()` value may no longer be valid. It is safe to use a previous `telldir()` value immediately after a call to `opendir()` and before any calls to `readdir()`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

[closedir\(\)](#)
[opendir\(\)](#)
[readdir\(\)](#)
[rewinddir\(\)](#)
[telldir\(\)](#)

Syntax

```
#include <UNIX/os9types.h>
#include <types.h>
int select (
    unsigned int      width,
    fd_set            *readfds,
    fd_set            *writefds,
    fd_set            *exceptfds,
    struct timeval    *timeout);
```

Description

`select()` examines the path sets whose addresses are passed in `readfds`, `writefds`, and `exceptfds` to see if some of their paths are ready for reading, ready for writing, or have an exceptional condition pending.

`width` is the number of bits to be checked in each bit mask that represent a path. The paths from 0 through `width-1` in the descriptor sets are examined. `width` is in the range `1-FD_SETSIZE`. The total number of ready paths in all the sets is returned. The macros `FD_SET`, `FD_CLR`, `FD_ISSET`, and `FD_ZERO` have been defined in `os9types.h` to manipulate these `fd_set` structures.

`timeout`

If `timeout` is not a NULL pointer, it specifies a maximum interval to wait for the selection to complete. If `timeout` is a NULL pointer, the `select` blocks indefinitely. To effect a poll, the `timeout` argument should be a non-NUL pointer, pointing to a zero-valued `timeval` structure.

readfds, writefds, exceptfds

Any readfds, writefds, and exceptfds may be given as NULL pointers if no paths are of interest.
(Input)

select() returns a non-negative value on success. A positive value indicates the number of ready paths in the path sets. Zero indicates that the time limit referred to by timeout has expired. On failure, select() returns -1, sets errno to indicate the error, and the path sets are not changed.



Note

User applications using select() can not use the _os_intercept() function.



Note

No paths in the exceptfds are ever returned since OS-9 for 68K does not support the concept of “exceptional condition.”

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe
Re-entrant:	No

Library

unix.1

set<name>()

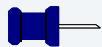
Set Value of SPR, DCR, PREG, or TB

Syntax

```
#include <getset.h>
#include <types.h>
void _set_<name>(u_int32 val);
```

Description

The `_set_<name>()` class of functions can be used to set the value of Special Purpose Registers (SPRs), Device Control Registers (DCRs), Processor Registers (PREG), and Time-Based Registers (TBs).



Note

All functions are provided. It is the programmer's responsibility to avoid using system-state only registers from user-state and accessing registers that do not exist on the target processor. All registers are accessible from system-state. Only those noted are available from user-state. Attempting to use system state registers from user state causes unpredictable results.

Table 2-31 `_set_<name>()` Valid Name Values

<name>	Register Type	Processor(s)				User	Readable
		403	505	601	603		
bar	SPR			•			•
besr	DCR		•				•

Table 2-31 _set_<name>() Valid Name Values (continued)

<name>	Register Type	Processor(s)				User	Readable
		403	505	601	603		
br0	DCR	•					•
br1	DCR	•					•
br2	DCR	•					•
br3	DCR	•					•
br4	DCR	•					•
br5	DCR	•					•
br6	DCR	•					•
br7	DCR	•					•
cmpa	SPR		•				•
cmpb	SPR		•				•
cmpc	SPR		•				•
cmpd	SPR		•				•
cmpe	SPR		•				•
cmpf	SPR		•				•
cmpg	SPR		•				•
cmph	SPR		•				•
counta	SPR		•				•

Table 2-31 _set_<name>() Valid Name Values (continued)

<name>	Register Type	Processor(s)				User	Readable
		403	505	601	603		
countb	SPR		•				•
cr	PREG	•	•	•	•	•	•
ctr	SPR	•	•	•	•	•	•
dabr	SPR			•			•
dac1	SPR	•					•
dac2	SPR	•					•
dar	SPR		•	•	•		•
dbat01	SPR				•		•
dbat0u	SPR				•		•
dbat11	SPR				•		•
dbat1u	SPR				•		•
dbat21	SPR				•		•
dbat2u	SPR				•		•
dbat31	SPR				•		•
dbat3u	SPR				•		•
dbcrr	SPR	•					•
dbsr	SPR	•					•

Table 2-31 _set_<name>() Valid Name Values (continued)

<name>	Register Type	Processor(s)				User	Readable
		403	505	601	603		
dccr	SPR	•					•
dcmp	SPR			•			•
dec	SPR		•	•	•		•
der	SPR		•				•
dmacc0	DCR	•					•
dmacr0	DCR	•					•
dmacr1	DCR	•					•
dmacr2	DCR	•					•
dmacr3	DCR	•					•
dmact0	DCR	•					•
dmact1	DCR	•					•
dmact2	DCR	•					•
dmact3	DCR	•					•
dmada0	DCR	•					•
dmada1	DCR	•					•
dmada2	DCR	•					•
dmada3	DCR	•					•

Table 2-31 _set_<name>() Valid Name Values (continued)

<name>	Register Type	Processor(s)				User	Readable
		403	505	601	603		
dmasa0	DCR	•					•
dmasa1	DCR	•					•
dmasa2	DCR	•					•
dmasa3	DCR	•					•
dmasr	DCR	•					•
dmiss	SPR				•		•
dpdr	SPR		•				•
dsisr	SPR		•	•	•		•
ear	SPR			•	•		•
ecr	SPR		•				•
eid	SPR		•				•
eie	SPR		•				•
esr	SPR	•					•
evpr	SPR	•					•
exier	DCR	•					•
exisr	DCR	•					•
fpecr	SPR		•				•

Table 2-31 _set_<name>() Valid Name Values (continued)

<name>	Register Type	Processor(s)				User	Readable
		403	505	601	603		
fpscscr	PREG	•	•	•	•	•	•
hash1	SPR				•		•
hash2	SPR				•		•
hid0	SPR			•	•		•
hid1	SPR			•			•
hid15	SPR			•			•
hid2	SPR			•			•
hid5	SPR			•			•
iabr	SPR			•	•		•
iac1	SPR	•					•
iac2	SPR	•					•
ibat01	SPR			•	•		•
ibat0u	SPR			•	•		•
ibat11	SPR			•	•		•
ibat1u	SPR			•	•		•
ibat21	SPR			•	•		•
ibat2u	SPR			•	•		•

Table 2-31 _set_<name>() Valid Name Values (continued)

<name>	Register Type	Processor(s)				User	Readable
		403	505	601	603		
ibat31	SPR			•	•		•
ibat3u	SPR			•	•		•
icadr	SPR		•				•
iccr	SPR	•					•
iccst	SPR			•			•
icdat	SPR		•				•
icmp	SPR				•	•	•
ictrl	SPR		•				•
imiss	SPR				•		•
iocr	DCR	•					•
lctrl1	SPR		•				•
lctrl2	SPR		•				•
lr	SPR	•	•	•	•	•	•
mq	SPR			•		•	•
nri	SPR		•				•
msr	PREG	•	•	•	•		•
pbl1	SPR	•					•

Table 2-31 _set_<name>() Valid Name Values (continued)

<name>	Register Type	Processor(s)				User	Readable
		403	505	601	603		
pbl2	SPR	•					•
pbu1	SPR	•					•
pbu2	SPR	•					•
pir	SPR			•			•
pit	SPR	•					•
rpa	SPR				•		•
rtcl	SPR			•			•
rtcu	SPR			•			•
sdr1	SPR			•	•		•
sprg0	SPR	•		•	•		•
sprg1	SPR	•		•	•		•
sprg2	SPR	•		•	•		•
sprg3	SPR	•		•	•		•
sr0	PREG			•	•		•
sr1	PREG			•	•		•
sr2	PREG			•	•		•
sr3	PREG			•	•		•

Table 2-31 _set_<name>() Valid Name Values (continued)

<name>	Register Type	Processor(s)				User	Readable
		403	505	601	603		
sr4	PREG			•	•		•
sr5	PREG			•	•		•
sr6	PREG			•	•		•
sr7	PREG			•	•		•
sr8	PREG			•	•		•
sr9	PREG			•	•		•
sr10	PREG			•	•		•
sr11	PREG			•	•		•
sr12	PREG			•	•		•
sr13	PREG			•	•		•
sr14	PREG			•	•		•
sr15	PREG			•	•		•
srr0	SPR	•	•	•	•		•
srr1	SPR	•	•	•	•		•
srr2	SPR	•					•
srr3	SPR	•					•
tbhi	SPR	•					•

Table 2-31 _set_<name>() Valid Name Values (continued)

<name>	Register Type	Processor(s)				User	Readable
		403	505	601	603		
tbl	SPR			•		•	
tblo	SPR		•				•
tbu	SPR			•		•	
tcr	SPR		•				•
tsr	SPR		•				•
xer	SPR	•		•	•	•	•

Attributes

Operating System: OS-9
 State: User and System
 Threads: Safe
 Re-entrant: Yes

Library

cpu.l

See Also

[_get_<name>\(\)](#)

Syntax

```
#include <stdio.h>
void setbuf(
    FILE      *stream,
    char      *buf );
```

Description

`setbuf()` sets up buffering for an I/O stream. It may be used only after the stream has been associated with an open file and before any other operation is performed on the stream. It is similar to `setvbuf()` called with `_IOFBF` for mode and `BUFSIZ` for size, or, if `buf` is a null pointer, with `_IONBF` for mode.

stream	is a pointer to the C I/O FILE structure. (Input)
buf	is a pointer to the file's buffer. (Input)
<code>setbuf()</code>	returns no value. (Input)

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

clib.l

See Also

[fopen\(\)](#)
[getc\(\), getchar\(\)](#)
[putc\(\), putchar\(\)](#)
[setvbuf\(\)](#)

Syntax

```
#include <module.h>
int _setcrc(mh_com *module);
```

Description

`_setcrc()` updates the header parity and CRC of a module in memory. The module must have the correct size and sync bytes; other parts of the module are not checked.

`module` is a pointer to the module to update.
(Input)

If an error occurs, -1 is returned and the appropriate error code is placed in the global variable `errno`.



WARNING

Although `_os_setcrc()` and `_setcrc()` functions do update the header and CRC of a module in memory, the copy of the module header in the module directory is not affected. Therefore, you have an invalid module directory entry for the memory module with the new CRC and an error 236 results if links to the module are attempted.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

[_os_setcrc\(\)](#)

F\$SetCRC

F_SETCRC

OS-9 for 68K Technical Manual

OS-9 Technical Manual

set_excpt_base()

Set Exception Table Base Address

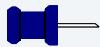
Syntax

```
#include <regs.h>
void set_excpt_base(
    u_int32    *dsreq);
```

Description

`set_excpt_base()` sets the exception table base address. It is a system state only system call.

`dsreg` is a pointer to the base address of the exception table. (Input)



Note

`set_excpt_base()` is supported only on the 80x86 family of processors version of the compiler.

Attributes

Operating System:	OS-9
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

cpu.1

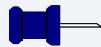
Syntax

```
#include <regs.h>
void set_gdtr(
    u_int32      *base);
```

Description

`set_gdtr()` sets the global descriptor pointer. It is a system state only system call.

`base` is a pointer to the global descriptor.
(Input)



Note

`set_gdtr()` is supported only on the 80x86 family of processors version of the compiler.

Attributes

Operating System:	OS-9
State:	System
Threads:	Safe
Re-entrant:	Yes

Library

cpu.l

setgid()

Set Global Descriptor Pointer

Syntax

```
#include <UNIX/os9def.h>
int setgid(
            unsigned    int gid);
```

Description

`setgid()` sets the user's group identification of the current process as specified by `gid` (input). `setgid()` returns -1 on error and sets the global variable `errno` to indicate the error.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

unix.l

Syntax

```
#include <time.h>
int setime(struct sgtbuf *timebuf);
```

Description

`setime()` sets the system time from a time buffer. The time units are defined in the `time.h` header file. This function uses the year 1900 as a base, and the year passed to the function must be an offset of 1900. For example, to use the `setime()` function and set to the year 2011, you need to pass 111 in the year member of the `tm` structure.

`timebuf` is a pointer to the time buffer. (Input)

If successful, `setime()` returns a pointer to a buffer. Otherwise, -1 is returned and the appropriate error code is placed in the global variable `errno`.



Note

Only super users may set the time.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

[getime\(\)](#)
[_os_setime\(\)](#)
[_os9_setime\(\)](#)

[F\\$STime](#)
[F_STIME](#)

OS-9 for 68K Technical Manual
OS-9Technical Manual

Syntax

```
#include <setjmp.h>
int setjmp(jmp_buf env);
```

Description

`setjmp()` is a macro which saves its calling environment in its `env` parameter for later use by a call to `longjmp()`.

`env` is a program environment structure.
(Input)

If the return is from a direct call, `setjmp()` returns zero.

If the return is from a call to `longjmp()`, `setjmp()` returns a non-zero value.

`setjmp()` may only be used as follows:

- The entire expression of an expression statement. This may be cast to `void`. For example:

```
(void) setjmp(env);
```

- The entire controlling expression of a selection or iteration statement. For example:

```
while(setjmp(env))
```

- One expression of a relational or equality operator with the operand an integral constant expression, with the resulting expression being the entire controlling expression of a selection or iteration statement. For example:

```
while(1<=setjmp(env))
```

- The operand of the unary operator (!) with the resulting expression being the entire controlling expression of a selection or iteration statement. For example:

```
while(!setjmp(env))
```

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

clib.l

See Also

[longjmp\(\)](#)

Syntax

```
#include <locale.h>
char *setlocale(
    int          category,
    const char  *locale);
```

Description

`setlocale()` selects the appropriate portion of the program's locale as specified by `category` and `locale`. You can use `setlocale()` to change or query the program's entire current locale or portions of it.

`category` can be one of the values shown in **Table 2-32.** (Input)

Table 2-32 setlocale() Category Values

Name	Description
LC_ALL	Names the program's entire locale. The other values for <code>category</code> name only a portion of the program's locale.
LC_COLLATE	Affects the behavior of <code>strcoll()</code> and <code>strxfrm()</code>
LC_CTYPE	Affects the behavior of the character handling functions and the multibyte functions
LC_MONETARY	Affects the monetary formatting information returned by <code>localeconv()</code>
LC_NUMERIC	Affects the decimal point character for the formatted I/O functions and the string conversion functions, as well as the non-monetary formatting information returned by <code>localeconv()</code>
LC_TIME	Affects the behavior of <code>strftime()</code>

Currently, the C locale is fully supported.

If `locale` (input) is C, the minimal environment for C translation is specified.

If `locale` is "", the value of the corresponding environment variable, `LC_TIME` for category `LC_TIME`, and so forth, is checked for validity. If the environment variable contains a valid locale string, `locale` is set accordingly. If it does not contain a valid string, `setlocale()` behaves as if passed an invalid locale string.

If the environment variable is not set or is set to the empty string, the existence and validity of the environment variable `LANG` is checked and `locale` is set from these. If `LANG` does not exist, `locale` is set to C.

At program startup, the equivalent of the following is executed:

```
setlocale(LC_ALL, "C");
```

If `locale` is a pointer to a string and the selection can be honored, `setlocale()` returns a pointer to the string associated with the specified category for the new locale. If the selection cannot be honored, `setlocale()` returns a null pointer and the program's locale is not changed.

A null pointer for `locale()` causes `setlocale()` to return a pointer to the string associated with the category for the program's current locale; the program's locale is not changed.

The pointer to the string returned by `setlocale()` is such that a subsequent call with that string value and its associated category restores that part of the program's locale. The program does not modify the string, but it may be overwritten by a subsequent call to `setlocale()`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

clib.l

See Also

[atof\(\)](#)
[atoi\(\)](#)
[atol\(\)](#)
[fprintf\(\)](#)
[fscanf\(\)](#)
[isalnum\(\)](#)
[isalpha\(\)](#)
[iscntrl\(\)](#)
[isdigit\(\)](#)
[isgraph\(\)](#)
[islower\(\)](#)
[isprint\(\)](#)
[ispunct\(\)](#)
[isspace\(\)](#)
[isupper\(\)](#)
[isxdigit\(\)](#)
[_issjis1\(\)](#)
[_issjis2\(\)](#)
[_isjis_kana\(\)](#)
[mblen\(\)](#)
[mbstowcs\(\)](#)
[mbtowc\(\)](#)
[printf\(\)](#)
[scanf\(\)](#)
[sprintf\(\)](#)
[sscanf\(\)](#)
[vfprintf\(\)](#)
[vprintf\(\)](#)
[vsprintf\(\)](#)
[wctomb\(\)](#)
[wcstombs\(\)](#)
[strtod\(\)](#)
[strtol\(\)](#)
[strtoul\(\)](#)

strcoll()
strftime()
strxfrm()

Syntax

```
#include <process.h>
int setpr(
    int      pid,
    int      prior);
```

Description

`setpr()` sets the priority of a process. The lowest priority is zero; the highest is 65535.

<code>pid</code>	is the process ID. (Input)
<code>prior</code>	specifies the priority. (Input)
	Only the lower 16 bits of <code>prior</code> are used.

If an error occurs, such as the process not having the same user ID as the caller, -1 is returned and the appropriate error code is placed in the global variable `errno`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

setpwent()

Reset Password File

Syntax

```
#include <UNIX/pwd.h>
int setpwent(void);
```

Description

setpwent() has the effect of rewinding the password file to allow for repeated searches.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

unix.l

See Also

[endpwent\(\)](#)
[fgetpwent\(\)](#)
[getpwent\(\)](#)
[getpwnam\(\)](#)
[getpwuid\(\)](#)

Syntax

```
#include <sg_codes.h>
#include <sgstat.h>
int setstat(
    int      code,
    int      path,
    char    *buffer); /* code = 0* /
int setstat(
    int      code,
    int      path,
    long     size); /* code = 2* /
```

Description

`setstat()` sets the path options or the file size of the file open on `path`.

If `code` is 0, the buffer is copied to the path descriptor options section. The `sgstat.h` header file contains the definitions for the path options.

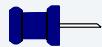
If `code` is 2, `size` should be an `int` specifying the new file size. Only the lower 16 bits of `code` are used.

`path` is the path number of the open file.
(Input)

`buffer` is a pointer to the buffer containing the path descriptor options. (Input)

`size` is the size of the new file. (Input)

If an error occurs, both forms of the call return -1 and place the appropriate error code in the global variable `errno`.



Note

This call exists for 6809 portability. The `_os_ss` functions are the preferred versions of these function calls.

Attributes

Operating System:	OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

[getstat\(\)](#)
[_os_ss_attr\(\)](#)
[_os9_ss_close\(\)](#)
[_os_ss_dcon\(\)](#)
[_os_ss_dcoff\(\)](#)
[_os_ss_dsrts\(\)](#)
[_os_ss_enrts\(\)](#)
[_os_ss_erase\(\)](#)
[_os_ss_fd\(\)](#)
[_os_ss_lock\(\)](#)
[_os_ss_popt\(\)](#)
[_os_ss_relea\(\)](#)
[_os_ss_reset\(\)](#)
[_os_ss_rfm\(\)](#)
[_os_ss_sendsig\(\)](#)
[_os_ss_size\(\)](#)
[_os_ss_skip\(\)](#)
[_os_ss_ticks\(\)](#)
[_os_ss_wfm\(\)](#)
[_os_ss_wtrack\(\)](#)
[_os9_ss_open\(\)](#)

I\$SetStt

OS-9 for 68K Technical Manual

_setsys()

Set/Examine System Global Variables

Syntax

```
#include <sysglob.h>
#include <setsys.h>
int _setsys(
    int      glob,
    int      size[ ,
    int      value]);
```

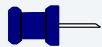
Description

`_setsys()` changes or examines a system global variable. These variables are defined in `sys.l`. The same values are defined in the `setsys.h` header file for C program use. These variables begin with a `D_` prefix.

<code>glob</code>	is the offset to the intended variable. (Input)
	Only the lower 16 bits of <code>glob</code> are used.
<code>size</code>	is the size of the variable. (Input)
	<code>size 0x80000000</code> is used to examine the variable.
<code>value</code>	is an optional parameter used only when changing a variable. (Input)

`_setsys()` returns the value of the variable on an examination request. On a change request, `_setsys()` returns the value of the variable before the change.

If the examine or change request fails, `-1` is returned and the appropriate error code is placed in the global variable `errno`.



Note

Use `_getsys()` to examine variables without the possibility of accidental change.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

[`_getsys\(\)`](#)
[`_os_getsys\(\)`](#)
[`_os_setsys\(\)`](#)
`F$SetSys`

OS-9 for 68K Technical Manual

Syntax

```
#include <process.h>
int setuid(int uid);
```

Description

`setuid()` sets the group/user ID of the process to `uid`. The following restrictions govern the use of `setuid()`:

- User number 0 . 0 may change his/her ID to anything without restriction.
- A primary module owned by user 0 . 0 may change the process' user ID to anything without restriction.
- Any primary module may change the process' user ID to match the module's owner.

If the call fails, -1 is returned and the appropriate error code is placed in the global variable `errno`.

`uid` is the user identification. (Input)

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.lib`

See Also

[getuid\(\)](#)

Syntax

```
#include <stdio.h>
int setvbuf(
    FILE      *stream,
    char      *buf,
    int       mode,
    size_t    size);
```

Description

`setvbuf()` sets up buffering for an I/O stream. It may be used only after the `stream` has been associated with an open file and before any other operation is performed on the stream.

<code>stream</code>	is a pointer to the C I/O <code>FILE</code> structure. (Input)
<code>buf</code>	is a pointer to a buffer. (Input) If <code>buf</code> is not a null pointer, the library may use the array it points to instead of a buffer allocated by <code>setvbuf()</code> .
<code>mode</code>	determines how <code>stream</code> is buffered. (Input)

Table 2-33 `setvbug()` Parameter Mode Values

Mode	Description
<code>_IOFBF</code>	I/O is fully buffered
<code>_IOLBF</code>	I/O is line buffered
<code>_IONBF</code>	I/O is unbuffered

size specifies the array size. (Input)
The contents of the array at any time are indeterminate.

`setvbuf()` returns zero if successful. If an invalid value is given for mode or if the request cannot be honored, `setvbuf()` returns a non-zero value.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`clib.l`

Syntax

```
#include <signal.h>
int sigmask(int level);
```

Description

`sigmask()` controls the process' signal mask. A signal mask is an internal variable in each process descriptor which determines the signal handling for the process.

level	specifies the signal level. (Input)
	• 0 specifies clear
	• 1 specifies increment
	• -1 specifies decrement

The process' signal level is not decremented below zero.

If a signal is received by a process whose signal mask is zero, normal program flow is interrupted and the signal is processed by executing the program's intercept routine.

If a signal is received by a process whose signal mask is non-zero, the signal is placed in a queue of signals waiting to be processed. The queued signals become active only when the process' signal mask becomes zero.

Regardless of the signal mask value, a process wakes up from event waits and I/O operation when signals arrive, but the signal handling routine is not executed until the signals are unmasked.

The process' signal mask is automatically incremented while the intercept routine is executing. This prevents the intercept routine from being accidentally re-entered if a new signal arrives. The process may use `sigmask()` within its intercept routine to allow re-entrant signals or to force the signal mask to remain non-zero when normal program execution resumes.

When a process makes an F\$Sleep or F\$Wait (for OS-9 for 68K) or F_SLEEP or F_WAIT (for OS-9) system call, its signal mask is automatically cleared. If any signals are pending, the process returns to the intercept routine without sleeping.

The S_KILL and S_WAKE signals ignore the state of the signal mask and are never queued. S_KILL terminates the receiving process, and S_WAKE ensures that the receiving process is active.

If an error occurs, -1 is returned and the appropriate error code is placed in the global variable errno. If errors do not occur, sigmask() returns zero.



Note

I/O operations using the `cio` library should not be performed by both the main program and the intercept routine.

If an intercept routine is exited with `longjmp()`, the signal mask is still set to one. Generally, the destination of the long jump should unmask signals immediately.

OS-9 for 68K: The depth to which signals may be queued is limited only by available memory.

OS-9: There is a set limit.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

[_os_intercept\(\)](#)

[_os_send\(\)](#)

[_os_sigmask\(\)](#)

[F\\$SigMask](#)

[F_SIGMASK](#)

OS-9 for 68K Technical Manual

OS-9 Technical Manual

Syntax

```
#include <signal.h>
void (*signal(int sig, void (*func)(int)))(int);
```

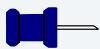
Description

`signal()` chooses one of three ways in which receipt of the signal number `sig` is handled.

- If `func` is `SIG_DFL`, default handling for that signal occurs.
- If `func` is `SIG_IGN`, the signal is ignored.
- Otherwise, `func` is a pointer to a function to call when that signal occurs. Such a function is called a **signal handler**.

When a signal occurs, signals are masked and `signal(sig, SIG_DFL)` is executed. Then, the specified type of handling is executed. Signals are blocked for the duration of the signal handler, unless the program takes some action that unmasks the signal.

There are times in which a thread receives a signal it is not expecting. When this occurs, the thread forwards the signal to an arbitrary thread that is expecting it.



Note

A program will not receive an abort or quit signal from the keyboard (usually ^C and ^E) unless the program performs output to the terminal. This is because the operating system sends the abort/quit signals to the last process to perform I/O to the terminal. If you run the program from the shell and type ^E before the program performs I/O to the terminal, the shell receives the signal and kills the running program. If a program requires immediate control of the terminal, perform some I/O to one of the standard paths such as printing a program banner or getting the terminal options with `_gs_opt()`.



For More Information

Refer to your operating system technical manual for more information.



Note

For each signal recognized by `signal()`, the default handling is program termination (with an exit status related to the exception or the signal number for non-exception related signals) and the program startup condition is `SIG_DFL`.

The default handling is reset on all signals.

Default Handling

If you specify default handling, the process terminates and the termination status is set to the signal number that occurred. This occurs if no `signal()` has been called for the signal or `signal()` was called with `SIG_DFL` as func.

If the program has used `intercept()` the signal handler assigned to the call is called and execution resumes at the point of interruption instead of terminating.



Note

Do not use `_os_intercept()` with `signal()`.

Ignore Handling

If a signal occurs that is to be ignored, the program resumes at the point of interruption.

If the occurring signal is exception-related, the process terminates with the appropriate vector as the exit status. Effectively, exception-related signals cannot be ignored.

Available Signals

The set of signals for the signal function varies with the operating system and the processor. A set of signals is also available regardless of the operating system.

ANSI “Core” Signals

Table 2-34 ANSI Core Signals

Signal	Description
SIGABRT	Abnormal termination, such as initiated by <code>abort()</code> .
SIGFPE	An erroneous arithmetic operation, such as zero divide or an operation resulting in overflow. This signal is generated on processor-specific conditions.
SIGILL	Detection of an invalid function image, such as illegal instruction. This signal is generated on processor-specific conditions.
SIGINT	Receipt of an interactive attention signal generated by a <code><control>C</code> on OS-9 for 68K and OS-9.
SIGSEGV	An invalid access to storage. This signal is generated on processor-specific conditions.
SIGTERM	A termination request sent to the program.

OS-9 for 68K/OS-9 “Core” Signals

Table 2-35 OS-9 for 68K and OS-9 Core Signals

Signal	Description
SIGKILL	A death request sent to the program. This signal cannot be ignored or handled.
SIGWAKE	A wakeup request sent to the program. If a process is active when this signal arrives, it is ignored. Otherwise, the process is activated, but the signal handling function is not called.
SIGQUIT	Receipt of an interactive abort signal generated by a <control>E on OS-9 for 68K and OS-9.
SIGHUP	Modem hangup signal. This signal is sent to a process when a modem connection is lost.
SIGALRM	Not automatically sent. Programs could use this signal for an alarm call.
SIGPIPE	Not automatically sent.
SIGUSR1	Condition is user defined.
SIGUSR2	Condition is user defined.

680x0: OS-9 for 68K, OS-9

Table 2-36 OS-9 for 68K and OS-9 Core Signals for 680x0

Signal	Description
SIGADDR	Address error
SIGCHK	CHK instruction
SIGTRAPV	TRAPV instruction
SIGPRIV	Privilege violation
SIGTRACE	Trace exception
SIG1010	1010 line-A exception
SIG1111	1111 line-F exception

680x0: OS-9

Table 2-37 OS-9 Core Signals for 680x0

Signal	Description
SIGCOPRCV	Coprocessor protocol violation
SIGFMTERR	Format error
SIGUNIRQ	Uninitialized interrupt

80x86: OS-9

Table 2-38 OS-9 Core Signals for 80x86

Signal	Description
SIGGPROT	General protection
SIGSTACK	Stack exception
SIGSEGNP	Segment not present
SIGINVTSS	Invalid TSS
SIGDBLFLT	Double fault
SIGBNDCHK	Boundary check
SIGBRKPT	Breakpoint
SIGNMI	Non-maskable interrupt
SIGDBG	Debug exception

PowerPC: OS-9

Table 2-39 OS-9 Core Signals for PowerPC

Signal	Description
SIGALIGN	Alignment
SIGCHECK	Machine check
SIGINST	Instruction access
SIGPRIV	Privilege violation

Function Handling

If a signal which has an associated function occurs, the equivalent of `(*func)(sig)` is executed. `func` may terminate by executing a `return` statement or by calling `abort()`, `exit()`, or `longjmp()`. If `func` executes a `return` statement and `sig` was exception-related, the program exits with the appropriate vector for a status. Otherwise, the program resumes execution at the point it was interrupted.

Asynchronous Signals

Signals that occur by means other than `abort()` or `raise()` are called **asynchronous**. Signal handlers dealing with asynchronous signals should be careful about the following:

Calling any function in the library (except `signal()` with the occurring signal) that may be working with static storage structures.

Calling any function that is not re-entrant.

Modifying any static storage structure other than a variable of type `volatile sig_atomic_t`.

If the `signal()` call can be honored, it returns `func` for the most recent call to `signal()` for the specified signal `sig`. Otherwise, `SIG_ERR` is returned and a positive value is stored in `errno`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

See Also

[abort\(\)](#)
[exit\(\)](#)
[intercept\(\)](#)
[_os_intercept\(\)](#)
[raise\(\)](#)

Syntax

```
#include <math.h>
double sin(double x);
```

Description

`sin()` returns the sine of `x` (input) as a double float. `x` is in radians.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

Syntax

```
#include <math.h>
double sinh(double x);
```

Description

`sinh()` returns the hyperbolic sine of x (input) as a double float. x is in radians. A range error occurs, `ERANGE` stored in `errno`, if the magnitude of x is too large. In this case, the appropriately signed `HUGE_VAL` is returned.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

Possible Error

`ERANGE`

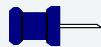
Syntax

```
#include <signal.h>
int sleep(unsigned seconds);
```

Description

`sleep()` suspends the calling process for the specified time. A sleep time of zero seconds sleeps indefinitely. `sleep()` returns the number of ticks remaining to sleep if awakened prematurely by a signal.

seconds	specifies the length of time for the process to sleep. (Input)
---------	--



Note

The thread enabled version of this function contains a cancel point. For more information see ***Using OS-9 Threads***.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`_os_sleep()`
`_os9_sleep()`
F\$Sleep
F_SLEEP

OS-9 for 68K Technical Manual
OS-9 Technical Manual

Syntax

```
#include <module.h>
mh_com *_sliblink(
    int             sllib_num,
    const char     *sllib_name);
```

Description

`_sliblink()` links or loads the named module. Then, it places the subroutine module's execution entry pointer and static storage data pointer in the global arrays `_sublibs` and `_submems`, respectively. You can use `_subcall()` to make subsequent calls to the subroutine library.

`sllib_num` specifies the subroutine number. (Input)

`sllib_name` is a pointer to the subroutine name string. (Input)



For More Information

For more information on creating, installing, and calling subroutine library modules, refer to the ***OS-9 Technical Manual***.

You can remove a subroutine module by passing a null pointer for the name of the module and specifying the subroutine number. A process can link to a maximum 16 subroutine libraries, numbered from 0 to 15.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

See Also

[_subcall\(\)](#)
[F_SLINK](#)

OS-9 Technical Manual

Syntax

```
#include <stdio.h>
int sprintf(
            char          *str,
            const char    *format, ...);
```

Description

`sprintf()` is a standard C library function that performs formatted output to the array specified by `str`. `sprintf()` converts, formats, and prints any parameters as indicated by the control string. A null character is written at the end of the characters written; it is not counted as part of the returned sum. If copying takes place between overlapping objects, the behavior is undefined.

`str` is a pointer to an array into which to write the generated output. (Output)

`format` is a pointer to a control string. (Input)

`sprintf()` returns the number of characters written in the array, not counting the terminating null character.



For More Information

The `sclib.l` and `sclib.il` libraries contain a smaller version of the `sprintf()` function. Refer to the ***Using Ultra C/C++*** manual for more information about the small library functions.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

clib.l

See Also

[printf\(\)](#)
[fprintf\(\)](#)

Syntax

```
#include <math.h>
double sqrt(double x);
```

Description

`sqrt()` returns the square root of `x` (input). `x` must not be negative. If the parameter is negative, `sqrt()` sets a domain error `EDOM` in `errno` and returns `HUGE_VAL`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

Possible Error

`EDOM`

Syntax

```
#include <stdlib.h>
void srand(unsigned int seed);
```

Description

`srand()` uses the parameter as a seed for a new sequence of pseudo-random numbers to be returned by subsequent calls to `rand()`. If `srand()` is then called with the same seed value, the sequence of pseudo-random numbers is repeated. If `rand()` is called before any calls to `srand()` have been made, the same sequence is generated as when `srand()` is first called with a seed value of one.

`seed` is used as a seed for a new sequence of pseudo-random numbers. (Input)

`srand()` does not return a value.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

clib.l

See Also

[rand\(\)](#)

Syntax

```
#include <memory.h>
char *srqcmem(
                int      bytecnt,
                int      memtype);
```

Description

`srqcmem()` is a direct hook to the `F$SRqCMem` (for OS-9 for 68K) or `_os_srqcmem()` (for OS-9000) system call.

bytecnt	is rounded to a system-defined block size. (Input)
	The size of the allocated block is stored in the global integer variable <code>_srgcsiz</code> . If <code>bytecnt</code> is <code>0xffffffff</code> , the largest contiguous block of free memory in the system is allocated.
memtype	indicates the specific type of memory to allocate. (Input) <code>memory.h</code> contains definitions of the memory types that you may specify:

Table 2-40 srqcmem() Memory Types

Memory Type	Description
SYSRAM or MEM_SYS	System RAM memory
VIDEO1	Video memory for plane A

Table 2-40 srqcmem() Memory Types

Memory Type	Description
VIDEO2	Video memory for plane B
MEM_ANY	No memory type specified

If `memtype` is `MEM_ANY`, any available system memory may be allocated.

If successful, a pointer to the memory granted is returned. The returned pointer always begins on an even byte boundary. If the request was not granted, `-1` is returned and the appropriate error code is placed in the global variable `errno`.

Note

`srqcmem()` is identical to `_srqmem()` with the exception of the additional color parameter.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

[free\(\)](#)
[malloc\(\)](#)
[_os_srqmem\(\)](#)
[_os9_srqmem\(\)](#)
[_os_srtmem\(\)](#)
[F\\$SRqCMem](#)
[F_SRQMEM](#)

OS-9 for 68K Technical Manual
OS-9 Technical Manual

Syntax

```
#include <memory.h>
char *_srqmem(unsigned size);
```

Description

`_srqmem()` and the complementary function `_srtmem()` are provided to request and return system memory when tight control over memory allocation is required.

`_srqmem()` is a direct hook into the `F$SRqMem` (for OS-9 for 68K) or `F_SRQMEM` (for OS-9) system call.

`size` specifies the number of bytes to request.
(Input)

The specified `size` is rounded to a system-defined block size. A `size` of `0xffffffff` obtains the largest contiguous block of free memory in the system. The global unsigned variable `_srqsiz` may be examined to determine the actual size of the allocated block.

If successful, a pointer to the memory granted is returned. If the request was not granted, `-1` is returned and the appropriate error code is placed in the global variable `errno`.

The returned pointer always begins on an even byte boundary. Care should be taken to preserve the value of the pointer if the memory is to be returned with `_srtmem()` or `_os_srtmem()`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

sys_clib.l

See Also

[_os_srqmem\(\)](#)
[_os9_srqmem\(\)](#)
[_os_srtmem\(\)](#)

F\$SRqMem
F_SRQMEM

OS-9 for 68K Technical Manual
OS-9 Technical Manual

Syntax

```
#include <memory.h>
int _srtmem(
    unsigned    size,
    char       *ptr);
```

Description

`_srtmem()` is a direct hook into the F\$SRtMem (**OS-9 for 68K**) or `_os_srtmem()` (**OS-9**) system call. It is used to return memory granted by `_srqmem()` or `_os_srqmem()`. Care should be taken to ensure that `size` and `ptr` are the same as those returned by `_srqmem()` or `_os_srqmem()`.

<code>size</code>	specifies the number of bytes to return. (Input)
<code>ptr</code>	is a pointer to the memory to return. (Input)

If an error occurs, -1 is returned and the appropriate error code is placed in the global variable `errno`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

[_os_srqmem\(\)](#)
[_os9_srqmem\(\)](#)
[_os_srtmem\(\)](#)

F\$SRtMem
F_SRTMEM

OS-9 for 68K Technical Manual
OS-9 Technical Manual

Syntax

```
#include <sg_codes.h>
int _ss_attr(
    int      path,
    int      attr);
```

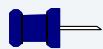
Description

`_ss_attr()` changes a disk file's attributes. `_os_gs_fd()` determines the current attributes of a file. Only the owner of the file or the super user can change a file's attributes.

`path` is the path number of a disk file. (Input)
`attr` specifies the file attributes to set. (Input)
Only the lower 16 bits of `attr` are used.

The attributes selected in `attr` are set in the file open on `path`. The `modes.h` header file defines the valid mode values.

If an error occurs, -1 is returned and the appropriate error code is placed in the global variable `errno`.



Note

`_ss_attr()` is effective even if the owner or super user does not have write permission to the path. You cannot set the `directory` bit of a non-directory file or clear the `directory` bit of a directory that is not empty.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

sys_clib.l

See Also

[_os_gs_fd\(\)](#)
[_os_ss_attr\(\)](#)
[_os_ss_fd\(\)](#)
[I\\$SetStt](#)
[I_SETSTAT, SS_ATTR](#)

OS-9 for 68K Technical Manual
OS-9 Technical Manual

Syntax

```
#include <stdio.h>
int sscanf(
    const char *input,
    const char *format, ...);
```

Description

`sscanf()` reads input from the string specified by `input`. Reaching the end of the string is equivalent to encountering end-of-file for `fscanf()`. If copying takes place between overlapping objects, the behavior is undefined.

<code>input</code>	is a pointer to the input string. (Input)
<code>format</code>	is a pointer to the control string. (Input)

`sscanf()` returns `EOF` if an input failure occurs before any conversion. Otherwise, `sscanf()` returns the number of input items assigned. This can be fewer than provided for or zero, in the event of an early matching failure.



For More Information

The `sclib.l` and `sclib.il` libraries contain a smaller version of the `sscanf()` function. Refer to the ***Using Ultra C/C++*** manual for more information about the small library functions.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

clib.l

See Also

[scanf\(\)](#)

[fscanf\(\)](#)

[printf\(\)](#)

_ss_dcoff()

Send Data Carrier Lost Signal to Process

Syntax

```
#include <scf.h>
int _ss_dcoff(
    int      path,
    int      sigcode);
```

Description

`_ss_dcoff()` sends a signal to the calling process when the “data carrier detect line” associated with the device is lost.

path	is the path number of the device. (Input)
sigcode	is the signal code to send. (Input)
	Only the lower 16 bits of <code>sigcode</code> are used on OS-9 for 68K.

If `_ss_dcoff()` cannot register the signal, -1 is returned and the appropriate error code is placed in the global variable `errno`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

[`_os_ss_dcoff\(\)`](#)

`I$SetStt`

`I_SETSTAT, SS_DCOFF`

OS-9 for 68K Technical Manual

OS-9 Technical Manual

_ss_dcon()

Send Data Carrier Present Signal to Process

Syntax

```
#include <scf.h>
int _ss_dcon(
    int      path,
    int      sigcode);
```

Description

`_ss_dcon()` sends a signal to the calling process when the “data carrier detect line” associated with the device is present.

path	is the path number of the device. (Input)
sigcode	is the signal code to send. (Input) Only the lower 16 bits of <code>sigcode</code> are used on OS-9 for 68K.

If `_ss_dcon()` cannot register the signal, -1 is returned and the appropriate error code is placed in the global variable `errno`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

[_os_ss_dcon\(\)](#)

I\$SetStt

I_SETSTAT, SS_DCON

OS-9 for 68K Technical Manual

OS-9 Technical Manual

Syntax

```
#include <scf.h>
int _ss_dsrts(int path);
```

Description

`_ss_dsrts()` disables the RTS line for an open device.

`path` is the path number of the device. (Input)

If `_ss_dsrts()` cannot disable the RTS line, -1 is returned and the appropriate error code is placed in the global variable `errno`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

[`_os_ss_dsrts\(\)`](#)

`I$SetStt`

`I_SETSTAT, SS_DSRTS`

OS-9 for 68K Technical Manual

OS-9 Technical Manual

_ss_enrts()

Enable RTS Line

Syntax

```
#include <scf.h>
int _ss_enrts(int path);
```

Description

`_ss_enrts()` enables the RTS line for an open device.

`path` is the path number of the device. (Input)

If `_ss_enrts()` cannot enable the RTS line, -1 is returned and the appropriate error code is placed in the global variable `errno`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

[_os_ss_enrts\(\)](#)

[I\\$SetStt](#)

[I_SETSTAT, SS_ENRTS](#)

OS-9 for 68K Technical Manual

OS-9 Technical Manual

Syntax

```
#include <rbf.h>
int _ss_lock(
    int          path,
    unsigned     locksize);
```

Description

`_ss_lock()` locks out a section of the file open on `path` from the current file position up to the number of bytes specified by `locksize`.

`path` is the path number of the file. (Input)

`locksize` specifies the number of bytes in the file to lock. (Input)

If `locksize` is zero, all locks (record-lock, EOF lock, and file lock) are removed.

If `locksize` is `0xffffffff`, the entire file is locked out regardless of the file pointer's location. This is a special type of file lock that remains in effect until released by `_ss_lock(path, 0)`, a read or write of zero bytes, or the file is closed.

If an error occurs, `-1` is returned and the appropriate error code is placed in the global variable `errno`.

For More Information

Refer to your operating system technical manual for information on RBF record locking.



Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

sys_clib.l

See Also

[_os_ss_lock\(\)](#)
[I\\$SetStt](#)
[I_SETSTAT, SS_LOCK](#)

OS-9 for 68K Technical Manual
OS-9 Technical Manual

Syntax

```
#include <sg_codes.h>
#include <sgstat.h>
int _ss_opt(
    int             path,
    struct sgbuf   *buffer);
```

Description

`_ss_opt()` copies a buffer into the options section of the path descriptor open on `path`.

`path` specifies the path number of the path descriptor. (Input)

`buffer` is a pointer to a buffer containing the path descriptor information. (Input)

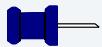
Generally, a program gets the options with `_gs_opt()`, changes the appropriate values, and updates the path options with `_ss_opt()`. The structure `sgbuf` declared in the `sgstat.h` header file provides a convenient means to access the individual option values.

If the path was invalid, `-1` is returned and the appropriate error code is placed in the global variable `errno`.



Note

It is common practice to preserve a copy of the original options so the program can restore them before exiting. The option changes take effect on the currently open path (and any paths created with `I$Dup` or `I_DUP` to the same).



Note

If you use `_ss_popt()` on OS-9, you must use the OS-9 for 68K header file. `_ss_luopt()` is provided on OS-9 solely for compatibility with OS-9. If the OS-9 for 68K header file is unavailable, change the source code to use `I_SETSTAT`, `SS_PATHOPT` or `I_SETSTAT`, `SS_LUOPT`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`_os_gs_popt()`
`_os_ss_popt()`

`I$SetStt`
`I_SETSTAT`
`tmode`

OS-9 for 68K Technical Manual
OS-9 Technical Manual
Using OS-9 for 68K or Using OS-9

Syntax

```
#include <rbf.h>
#include <direct.h>
int _ss_pfd(
    int             path,
    struct fd      *buffer);
```

Description

`_ss_pfd()` copies certain fields from the file descriptor information pointed to by `buffer` into the file descriptor sector of the file open on `path`. The buffer is usually obtained from `_gs_gfd()`. Only the owner ID, the modification date, and creation date fields are changed.

`path` is the path number of the file. (Input)

`buffer` is a pointer to the buffer containing the file descriptor information. (Input)

The structure `fd` declared in the `direct.h` header file provides a convenient means to access the file descriptor information.

If an error occurs, `-1` is returned and the appropriate error code is placed in the global variable `errno`.



Note

The buffer must be at least 32 bytes long or unanticipated data could be written into the file descriptor sector. Only the super user can change the owner ID field. You cannot change the file attributes with this call. Instead, use `_ss_attr()`.



For More Information

Refer to your operating system technical manual for information on the RBF File Manager.



Note

You must use the OS-9 for 68K `direct.h` header file. This call is provided on OS-9 solely for compatibility with OS-9 for 68K. If speed is an issue on OS-9, use `_os_ss_fd`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`_os_gs_fd()`
`_os_ss_fd()`
`I$SetStt`
`I_SETSTAT`

OS-9 for 68K Technical Manual
OS-9 Technical Manual

Syntax

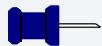
```
#include <sg_codes.h>
int _ss_rel(int path);
```

Description

`_ss_rel()` cancels the signal to send from a device on data ready.

`path` is the device's path number. (Input)

If an error occurs, `-1` is returned and the appropriate error code is placed in the global variable `errno`.



Note

The signal request is also cancelled when the issuing process dies or closes the path to the device. This feature exists only on SCF and PIPEMAN devices.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`_os_ss_relea()`
`_os_ss_sendsig()`

`I$SetStt`
`I_SETSTAT, SS_RELEASE`

OS-9 for 68K Technical Manual
OS-9 Technical Manual

Syntax

```
#include <sg_codes.h>
int _ss_rest(int path);
```

Description

`_ss_rest()` causes an RBF device to restore the disk head to track zero. It is usually used for disk formatting and error recovery.

`path` is the device's path number. (Input)

If an error occurs, -1 is returned and the appropriate error code is placed in the global variable `errno`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

[_os_ss_reset\(\)](#)

I\$SetStt

I_SETSTAT, SS_RESET

OS-9 for 68K Technical Manual

OS-9 Technical Manual

Syntax

```
#include <sg_codes.h>
int _ss_size(
    int      path,
    int      size);
```

Description

`_ss_size()` changes the size of an open file. This call is effective only on RBF devices. The size change is immediate. If the file size is decreased, the freed sectors are returned to the system. If the file size is increased, sectors with undefined contents are added to the file.

path	is the file's path number. (Input)
size	specifies the new file size in bytes. (Input)

If the path is invalid or the device is not an RBF device, -1 is returned and the appropriate error code is placed in the global variable `errno`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

[_os_ss_size\(\)](#)

`I$SetStt`

`I_SETSTAT, SS_SIZE`

OS-9 for 68K Technical Manual
OS-9 Technical Manual

_ss_ssig()

Send Signal on Data Ready

Syntax

```
#include <sg_codes.h>
int _ss_ssig(
    int    path,
    int    sigcode);
```

Description

`_ss_ssig()` sets up a signal to be sent to the calling process when an interactive device has data ready. This feature exists only on SCF devices and pipes. When data is received on the device indicated by `path`, a signal is sent to the calling process.

path	is the device's path number. (Input)
sigcode	specifies the signal code to send. (Input) Only the lower 16 bits of <code>sigcode</code> are used on OS-9 for 68K.

`_ss_ssig()` must be called each time the signal is sent if it is to be used again.

The device is considered busy and returns an error if any read requests arrive before the signal is sent. Write requests to the device are allowed while in this state.

If an error occurs, -1 is returned and the appropriate error value is placed in the global variable `errno`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

[_os_ss_relea\(\)](#)

[_os_ss_sendsig\(\)](#)

I\$SetStt

I_SETSTAT, SS_SENDSIG

OS-9 for 68K Technical Manual

OS-9 Technical Manual

Syntax

```
#include <rbf.h>
int _ss_tiks(
    int      path,
    int      tickcnt);
```

Description

If a read or write request is issued for a part of a file that is locked out by another user, RBF normally sleeps indefinitely until the conflict is removed. This feature exists only on RBF devices. `_ss_tiks()` may be used to return the error `EOS_LOCK` to the program if the conflict still exists after a specified number of ticks have elapsed.

path	specifies the file's path number. (Input)
tickcnt	specifies the number of ticks to wait if a record-lock conflict occurs while the file is open on <code>path</code> . (Input)
	<ul style="list-style-type: none">• A <code>tickcnt</code> of zero (RBF's default) causes a sleep until the record is released.• A <code>tickcnt</code> of one returns an error if the record is not released immediately.

If an error occurs, `-1` is returned and the appropriate error value is placed in the global variable `errno`.



For More Information

Refer to your operating system technical manual for information on RBF record locking.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library

sys_clib.l

See Also

[_os_ss_ticks\(\)](#)
[I\\$SetStt](#)
[I_SETSTAT, SS_TICKS](#)

OS-9 for 68K Technical Manual
OS-9 Technical Manual

_ss_wtrk()

Write Track

Syntax

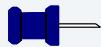
```
#include <rbf.h>
int _ss_wtrk(
    int          path,
    int          trkno,
    int          siden,
    int          ilvf,
    const char   *trkbuf,
    const char   *ilvptr);
```

Description

`_ss_wtrk()` performs a write-track operation on a disk drive. It is essentially a direct hook into the driver's write-track entry point.

path	is the path on which the device is open. (Input)
trkno	is the track number on which to write. (Input)
siden	is the side of the track to write. (Input)
ilvf	is the interleave factor. (Input)
trkbuf	is a pointer to the track buffer image.(Input)
ilvptr	is a pointer to the interleave table. (Input)

If an error occurs, -1 is returned and the appropriate error value is placed in the global variable `errno`.



Note

This feature exists only on RBF devices. You can obtain additional information on actual use of this call by examining the `format` utility and/or a device driver.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`_os_ss_wtrack()`
`I$SetStt`
`I_SETSTAT`, `SS_WTRACK`
`format`

OS-9 for 68K Technical Manual

OS-9 Technical Manual

Using OS-9 for 68K or *Using OS-9*

stacksiz(), _stacksiz()

Get Size of Stack Used

Syntax

```
#include <stdlib.h>
int stacksiz(void);
int _stacksiz(void);
```

Description

`stacksiz()` returns the maximum number of bytes of stack used at the time of the call if stack checking code is in effect. You can use `stacksiz()` to determine the stack size a program requires. `stacksiz()` is not available when compiling in strictly conforming ANSI mode.



Note

`stacksiz()` is historical and may be removed in a future release.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

cstart.r

Syntax

```
#include <UNIX/stat.h>
int stat(
    const char      *path,
    struct stat     *buf );
```

Description

`stat()` obtains information about the file named by `path` (input). Read, write, or execute permission of the named file is not required, but all directories leading to the file must be searchable.

`buf` is a pointer to a `stat` structure into which information about the file is placed. (Output)

A `stat` structure includes the following members:

```
unsigned short  st_mode;      /* attributes */
unsigned short  st_nlink;     /* nhard links */
unsigned short  st_uid;       /* user ID of owner */
unsigned short  st_gid;       /* group ID of owner */
unsigned long   st_size;      /* file size */
time_t          st_atime;     /* last access time */
time_t          st_mtime;     /* last modify time */
time_t          st_ctime;     /* last status change time */
long            st_ino;        /* undefined */
long            st_dev;        /* undefined */
long            st_rdev;       /* undefined */
```

If an error occurs, `stat()` returns -1 and sets the global variable `errno` to indicate the error.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

unix.l

See Also

[fstat\(\)](#)

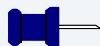
Syntax

```
#include <strings.h>
void _strass(
    void      *to,
    void      *from,
    int       count);
```

Description

`_strass()` is available to maintain source code compatibility. `count` bytes are copied from the memory pointed to by `from` to the memory pointed to by `to`, regardless of contents.

<code>to</code>	is a pointer to the destination for the copy. (Output)
<code>from</code>	is a pointer to the structure to copy. (Input)
<code>count</code>	is the number of bytes to copy. (Input)



Note

`_strass()` can move at most 65536 bytes. No regard is given to overlapping moves.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

strcat()

String Catenation

Syntax

```
#include <string.h>
char *strcat(
    char          *orig,
    const char   *append);
```

Description

`strcat()` appends a copy of `append`, including the terminating null character, to the end of `orig`. The initial character of `append` overwrites the null character at the end of `orig`. If copying takes place between overlapping objects, the behavior is undefined.

`orig` is a pointer to the original string.
(Input/Output)

append is a pointer to the string to append to orig. (Input)

`strcat()` returns the value of `orig`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

clib.l

Syntax

```
#include <string.h>
char *strchr(
    const char    *src,
    int           chr);
```

Description

`strchr()` locates the first occurrence of `chr`, converted to a `char`, in string `src`. The terminating null character is considered part of the string.

`src` is a pointer to the string to search.
(Input)

`chr` specifies the character for which to search. (Input)

`strchr()` returns a pointer to the located character, or a null pointer if the character does not occur in the string.

Attributes

Operating System: OS-9 and OS-9 for 68K

State: User and System

Threads: Safe

Re-entrant: Yes

Standards: ANSI

Library

`clib.l`

strcmp()

String Comparison

Syntax

```
#include <string.h>
int strcmp(
    const char *string1,
    const char *string2);
```

Description

`strcmp()` compares `string1` to `string2`.

`string1` is a pointer to a string to compare.
(Input)

`string2` is a pointer to a string to compare.
(Input)

`strcmp()` returns an integer that is:

- Positive, if `string1` is greater than `string2`
 - Zero, if `string1` equals `string2`
 - Negative, if `string1` is less than `string2`

Attributes

Operating System: OS-9 and OS-9 for 68K

State: User and System

Threads: Safe

Re-entrant: Yes

Standards: ANSI

Library

c lib. l

Syntax

```
#include <string.h>
int strcoll(
    const char      *string1,
    const char      *string2);
```

Description

`strcoll()` compares two strings, both interpreted as appropriate to the `LC_COLLATE` category of the current locale.

`string1` is a pointer to a string to compare.
(line 1)

(Input)

`string2` is a pointer to a string to compare.

(Input)

When both `string1` and `string2` are interpreted as appropriate to the current locale, `strcoll()` returns an integer that is:

- Positive, if `string1` is greater than `string2`.
 - Zero, if `string1` equals `string2`.
 - Negative, if `string1` is less than `string2`.

Attributes

Operating System: OS-9 and OS-9 for 68K

State: User and System

Threads: Safe

Re-entrant: Yes

Standards: ANSI

Library

clib.l

Syntax

```
#include <string.h>
char *strncpy(
    char      *dest,
    const char *src);
```

Description

`strncpy()` copies a string, including the terminating null character, into an array. If copying takes place between overlapping objects, the behavior is undefined.

`dest` is a pointer to the destination array.
(Output)

`src` is a pointer to the string to copy into
`dest`. (Input)

`strncpy()` returns the value of `dest`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

Syntax

```
#include <string.h>
size_t strcspn(
    const char    *src,
    const char    *delim);
```

Description

`strcspn()` returns the length of the segment of `src` up to the first occurrence of any character from `delim`.

<code>src</code>	is a pointer to a source string. (Input)
<code>delim</code>	is a pointer to a string of characters. (Input)

`strcspn()` returns the length of the segment.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`clib.l`

strerror()

Map Error Message String

Syntax

```
#include <string.h>
char *strerror(int errnum);
```

Description

`strerror()` maps an error number to an error message string.

`errnum` specifies the error number. (Input)

`strerror()` returns a pointer to the string. The program cannot modify the array, but a subsequent call to `strerror()` overwrites the array.

Error messages are generally in the form:

`<major>:<minor> <detail>`

`<major>` is the family of error.

`<minor>` is the error number within the family.

`<detail>` is the information from the system `errmsg` file, if available.

The system `errmsg` file (`/dd/SYS/errmsg`, `/h0/SYS/errmsg`, or `/d0/SYS/errmsg`) contains the error messages generated by `perror()`.

Attributes

Operating System: OS-9 and OS-9 for 68K

State: User and System

Threads: Safe

Re-entrant: No

Standards: ANSI

Library

clib.l

See Also

[perror\(\)](#)

Syntax

```
#include <time.h>
size_t strftime(
    char          *dest,
    size_t         maxsize,
    const char    *format,
    const struct tm *timeptr);
```

Description

strftime() places characters into an array as controlled by format. The format is a multibyte character sequence, beginning and ending in its initial shift state.

dest	is a pointer to the array into which to place the characters. (Output)
maxsize	specifies the maximum number of characters to place in the array. (Input)
format	is a pointer to a control string. (Input)
timeptr	format consists of zero or more conversion specifiers and ordinary multibyte characters. A conversion specifier consists of a percent (%) sign followed by a character that determines the behavior of the conversion specifier. is a pointer to a time structure. (Input)

All ordinary multibyte characters, including the terminating null character, are copied unchanged into the array. If copying takes place between overlapping objects, the behavior is undefined. No more than maxsize characters are placed in the array.

Each conversion specifier is replaced by appropriate characters as described in the following list. These characters are determined by the LC_TIME category of the current locale and by the values contained in the structure pointed to by `timeptr`.

Table 2-41 `strftime()` Conversion Values

Value	Replaced By
%a	The locale's abbreviated weekday name.
%A	The locale's full weekday name.
%b	The locale's abbreviated month name.
%B	The locale's full month name.
%c	The locale's appropriate date and time representation.
%d	The day of the month as a decimal number (01 - 31).
%H	The hour (24-hour clock) as a decimal number (00 - 23).
%I	The hour (12-hour clock) as a decimal number (01 - 12).
%j	The day of the year as a decimal number (001 - 366).
%m	The month as a decimal number (01 - 12).
%M	The minute as a decimal number (00 - 59).
%p	The locale's equivalent of the AM/PM designations associated with a 12-hour clock.
%S	The second as a decimal number (00 - 61).
%U	The week number of the year (the first Sunday as the first day of week 1) as a decimal number (00 - 53).
%w	The weekday as a decimal number (0 - 6), where Sunday is 0.

Table 2-41 strftime() Conversion Values (continued)

Value	Replaced By
%W	The week number of the year (the first Monday as the first day of week 1) as a decimal number (00 - 53).
%x	The locale's appropriate date representation.
%X	The locale's appropriate time representation.
%Y	The year without century as a decimal number (00 - 99).
%y	The year with century as a decimal number.
%z	The time zone name or abbreviation, or by no characters if no time zone is determinable.
%%	%.

If a conversion specifier is not one of the above, the behavior is undefined.

If the total number of resulting characters including the terminating null character is not more than `maxsize`, `strftime()` returns the number of characters placed into array `dest` not including the terminating null character. Otherwise, zero is returned and the contents of the array are indeterminate.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`clib.lib`

Syntax

```
#include <strings.h>
char *strhcpy(
    char          *dest,
    const char   *src);
```

Description

`strhcpy()` makes a copy of `src` in `dest`.

`dest` is a pointer to the destination string.
(Output)

`src` is a pointer to the source string to copy.
(Input)

`src` is assumed to have the high-order bit set on the character byte indicating the last character.

`strhcpy()` copies the bytes from `src` to `dest` until it reaches the null terminator. `strhcpy()` then removes the high-bit from the last character and appends a null byte to the output string.

`strhcpy()` is used primarily for copying file names from the directory entries on RBF disks. `strhcpy()` returns its first parameter.



Note

`strhcpy()` assumes that `dest` contains adequate space for the copy. The calling routine is responsible for verifying the space.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

sys_clib.l

Syntax

```
#include <string.h>
size_t strlen(const char *str);
```

Description

strlen() determines the length of str.

str is a pointer to a string. (Input)

strlen() returns the number of characters that precede the terminating null character.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

clib.l

strncat()

String Catenation

Syntax

```
#include <string.h>
char *strncat(
    char          *orig,
    const char   *append,
    size_t        count);
```

Description

`strncat()` appends characters (a null character and characters that follow it are not appended) from `append` to the end of `orig`. The initial character of `append` overwrites the null character at the end of `orig`. A terminating null character is always appended to the result. If copying takes place between overlapping objects, the behavior is undefined.

`orig` is a pointer to the original string.
(Input/Output)

append is a pointer to the string to append to orig. (Input)

count specifies the maximum number of characters to append. (Input)

`strncat()` returns the value of `orig`.

Attributes

Operating System: OS-9 and OS-9 for 68K

State: User and System

Threads: Safe

Re-entrant: Yes

Standards: ANSI

Library

c lib.1

Syntax

```
#include <string.h>
int strcmp(
            const char      *string1,
            const char      *string2,
            size_t          count);
```

Description

strcmp() compares not more than count characters from string1 to string2. Characters following a null character are not compared. The strings do not need to be null terminated.

string1	is a pointer to a string to compare. (Input)
string2	is a pointer to a string to compare. (Input)
count	specifies the maximum number of characters to compare. (Input)

strcmp() returns an integer that is:

- Positive, if string1 is greater than string2.
- Zero, if string1 equals string2.
- Negative, if string1 is less than string2.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

clib.l

Syntax

```
#include <string.h>
char *strncpy(
    char      *dest,
    const char *src,
    size_t     count);
```

Description

`strncpy()` copies characters from `src` to `dest`. Characters following a null character are not copied. If copying takes place between overlapping objects, the behavior is undefined.

`dest` is a pointer to the destination string.
(Output)

`src` is a pointer to the source string to copy.
(Input)

`count` specifies the maximum number of characters to copy. (Input)

If `src` is shorter than `count` characters, null characters are appended to the copy in `dest`, until `count` characters in all have been written.

`strncpy()` returns the value of `dest`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.lib`

strupr()

Locate First Occurrence of String

Syntax

```
#include <string.h>
char *strupr(
            const char    *src,
            const char    *delim);
```

Description

`strupr()` locates the first occurrence in `src` of any character from `delim`.

`src` is a pointer to a string in which to search for any character from `delim`. (Input)

`delim` is a pointer to a string containing characters to locate in `src`. (Input)

`strupr()` returns a pointer to the character or a null pointer, if no character from `delim` occurs in `src`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`clib.l`

strrchr()

Locate Last Occurrence of String

Syntax

```
#include <string.h>
char *strrchr(
                const char    *src,
                int            chr);
```

Description

`strrchr()` locates the last occurrence of `chr`, converted to `char`, in `src`. The terminating null character is considered part of the string.

`src` is a pointer to a string. (Input)

`chr` is the character to search for in `src`. (Input)

`strrchr()` returns a pointer to the character or a null pointer, if `chr` does not occur in the string.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

strspn()

Compute String Length

Syntax

```
#include <string.h>
char *strspn(
    const char    *src,
    const char    *pattern);
```

Description

`strspn()` returns the length of the maximum initial segment of `src` which consists entirely of characters from `pattern`.

`src` is a pointer to a string. (Input)

pattern is a pointer to a string. (Input)

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

clib.l

Syntax

```
#include <string.h>
char *strstr(
    const char    *src,
    const char    *pattern);
```

Description

`strstr()` locates the first occurrence in `src` of the sequence of characters (excluding the terminating null character) in `pattern`.

`src` is a pointer to a string. (Input)

`pattern` is a pointer to a string containing a sequence of characters to search for in `src`. (Input)

`strstr()` returns a pointer to the located string or a null pointer, if the string is not found. If `pattern` is a pointer to a string with zero length, `strstr()` returns `src`.

Attributes

Operating System: OS-9 and OS-9 for 68K

State: User and System

Threads: Safe

Re-entrant: Yes

Standards: ANSI

Library

clib.l

Syntax

```
#include <stdlib.h>
double strtod(
    const char *begin,
    char       **end);
```

Description

`strtod()` converts the initial portion of `begin` to double representation. `strtod()` separates the input string, attempts to convert the subject sequence to a floating point number, and then returns the result.

begin	is a pointer to the beginning of a string. (Input)
end	is a pointer to the pointer to the first character after the converted string. (Input)

The input string is separated into three parts:

1. An initial, possibly empty, sequence of white space characters (as specified by `isspace()`).
2. A subject sequence resembling a floating point constant. The subject sequence is the longest initial subsequence of the input string starting with the first non-white space character having the expected form. The subject sequence contains no characters if the input string is empty or consists entirely of white space, or if the first non-white space character is other than a sign, a digit, or a decimal point character.

3. A final string of one or more unrecognized characters, including the terminating null character of the input string.

`strtod()` expects the subject sequence to have the following form:

- An optional plus or minus sign.
- A non-empty sequence of digits optionally containing a decimal point character.
- An optional exponent part, but no floating suffix.

If the subject sequence has the expected form, the character sequence starting with the first digit or the decimal point character (whichever occurs first) is interpreted as a floating constant. Currently, only the C locale is supported. Therefore, the decimal point character is a period. If neither an exponent part nor a decimal point character appears, a decimal point is assumed to follow the last digit in the string. If the subject sequence begins with a minus sign, the value resulting from the conversion is negated. A pointer to the final string is stored in `end`, unless `end` is a null pointer.

If the subject sequence is empty or does not have the expected form, no conversion is performed. `begin` is stored in `end`, unless `end` is a null pointer.

`strtod()` returns the converted value, if any. If no conversion could be performed, zero is returned. If the correct value is outside the range of representable values, plus or minus `HUGE_VAL` is returned, according to the sign of the value, and `ERANGE` is stored in `errno`. If the correct value would cause underflow, zero is returned and `ERANGE` is stored in `errno`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.lib`

See Also

[isspace\(\)](#)
[strtol\(\)](#)
[strtoul\(\)](#)

Possible Error

[ERANGE](#)

Syntax

```
#include <string.h>
char *strtok(
    char          *src,
    const char    *delims);
```

Description

`strtok()` breaks `src` into a series of tokens when a sequence of calls is made to `strtok()`. A character from `delims` separates each token. When the first call is made, `src` is the first parameter. A null pointer is the first parameter for subsequent calls.

<code>src</code>	is a pointer to the string to break into tokens. (Input)
<code>delims</code>	is a pointer to a string containing characters to separate each token. (Input)
	<code>delims</code> may be different from call to call.

The first call in the sequence searches `src` for the first character that is not contained in the current `delims`. If no such character is found, then there are no tokens in `src`, and `strtok()` returns a null pointer. If a character is found, it starts the first token.

`strtok()` searches from that location for a character that is contained in the current `delims`. If the character is not found, the current token extends to the end of `src`, and subsequent searches for a token return a null pointer. If the character is found, it is overwritten by a null character to terminate the current token.

The search for the next token starts with the next character. `strtok()` saves a pointer to this location. Each subsequent call behaves in the same manner.

`strtok()` returns a pointer to the first character of a token, or a null pointer if there is no token.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`clib.lib`

Example

```
#include <string.h>
static char str[] = "?a???b,,,#c";
char *t;

t = strtok(str, "?");           /* t points to the token "a" */
t = strtok(NULL, ",");         /* t points to the token "?b" */
t = strtok(NULL, "#");         /* t points to the token "c" */
t = strtok(NULL, "?");         /* t is a null pointer */
```

Syntax

```
#include <stdlib.h>
long strtol(
            const char    *begin,
            char          **end,
            int           base);
```

Description

`strtol()` converts the initial portion of `begin` to `long int` representation.

`begin` is a pointer to the beginning of the string. (Input)

`end` is a pointer to the pointer to the first character after the converted string. (Input)

`base` is the base of the number string. (Input)

`strtol()` separates the input string into three parts:

1. An initial, possibly empty, sequence of white space characters (as specified by `isspace()`).
2. A subject sequence resembling an integer represented in some radix determined by `base`. The subject sequence is the longest initial subsequence of the input string, starting with the first non-white space character of the expected form. The subject sequence contains no characters if the input string is empty or consists entirely of white space, or if the first non-white space character is other than a sign or a permissible letter or digit.
3. A final string of one or more unrecognized characters, including the terminating null character of the input string.

`strtol()` then attempts to convert the subject sequence to an integer and returns the result.

- If `base` is zero, the subject sequence is expected to be an integer constant, optionally preceded by a plus or minus sign. An integer suffix is not included. If the subject sequence is an integer constant, the character sequence starting with the first digit is interpreted as an integer constant.
- If `base` is between 2 and 36, the subject sequence is expected to be a sequence of letters and digits representing an integer. `base` specifies the radix, optionally preceded by a plus or minus sign. An integer suffix is not included. The letters from `a` through `z` (regardless of case) are assigned the values 10 to 35. Only letters with assigned values less than `base` are permitted. If `base` is 16, the characters `0x` or `0X` may optionally precede the sequence, following the sign if present.

If the subject sequence has the expected form, it is used as the base for conversion, ascribing to each letter its value.

If the subject sequence begins with a minus sign, the value resulting from the conversion is negated. A pointer to the final string is stored in `end`, unless `end` is a null pointer.

If the subject sequence is empty or does not have the expected form, no conversion is performed. `begin` is stored in `end`, unless `end` is a null pointer.

`strtol()` returns the converted value, if any. If no conversion could be performed, zero is returned. If the correct value is outside the range of representable values, `LONG_MAX` or `LONG_MIN` is returned (according to the sign of the value) and `ERANGE` is stored in `errno`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

clib.l

See Also

[isupper\(\)](#)
[strtod\(\)](#)
[strtoul\(\)](#)

Possible Error

ERANGE

Syntax

```
#include <stdlib.h>
#include <limits.h>
unsigned long strtoul(
    char *begin,
    char **end,
    int base);
```

Description

`strtoul()` converts the initial portion of `begin` to `unsigned long` int representation.

`begin` is a pointer to the beginning of the string.
(Input)

`end` is a pointer to the pointer to the first character after the converted string.
(Input)

`base` is the base of the number string. (Input)

`strtoul()` separates the input string into three parts:

1. An initial, possibly empty, sequence of white space characters (as specified by `isspace()`).
2. A subject sequence resembling an unsigned integer represented in some radix determined by `base`. The subject sequence is the longest initial subsequence of the input string, starting with the first non-white space character of the expected form. The subject sequence contains no characters if the input string is empty or consists entirely of white space, or if the first non-white space character is other than a sign or a permissible letter or digit.
3. A final string of one or more unrecognized characters, including the terminating null character of the input string.

`strtoul()` then attempts to convert the subject sequence to an unsigned integer and returns the result.

- If `base` is zero, `strtoul()` expects the subject sequence to be an integer constant, optionally preceded by a plus or minus sign. An integer suffix is not included.

If the subject sequence is an integer constant, the character sequence starting with the first digit is interpreted as an integer constant.

- If `base` is between 2 and 36, the subject sequence is expected to be a sequence of letters and digits representing an integer. `base` specifies the radix, optionally preceded by a plus or minus sign. It does not include an integer suffix. The letters `a` through `z` (regardless of case) are assigned the values 10 to 35. Only letters whose assigned values are less than `base` are permitted. If `base` is 16, the characters `0x` or `0X` may optionally precede the sequence of letters and digits, following the sign if present.

If the subject sequence has the expected form, it is used as the base for conversion, ascribing to each letter its value.

If the subject sequence begins with a minus sign, the value resulting from the conversion is negated. A pointer to the final string is stored in `end`, unless `end` is a null pointer.

If the subject sequence is empty or does not have the expected form, no conversion is performed. `begin` is stored in `end`, unless `end` is a null pointer.

`strtoul()` returns the converted value, if any. If no conversion could be performed, zero is returned. If the correct value is outside the range of representable values, `ULONG_MAX` is returned and `ERANGE` is stored in `errno`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

clib.l

See Also

[isspace\(\)](#)
[strtod\(\)](#)
[strtol\(\)](#)

Possible Error

[ERANGE](#)

Syntax

```
#include <string.h>
size_t strxfrm(
    char          *dest,
    const char     *trans,
    size_t          count);
```

Description

`strxfrm()` transforms `trans` and places the result in `dest`. You can then use `strcmp()` to compare two transformed strings. The result of the `strcmp()` on the transformed strings is the same as the result of `strcoll()` applied to the two original strings.

<code>dest</code>	is a pointer to the destination string.
<code>trans</code>	is a pointer to the string to transform.
<code>count</code>	specifies the maximum number of characters to place in <code>dest</code> .
	<code>count</code> includes the terminating null character. If <code>count</code> is zero, <code>dest</code> can be a null pointer.

If copying takes place between overlapping objects, the behavior is undefined.

`strxfrm()` returns the length of the transformed string, not including the terminating null character. If the value returned is `count` or more, the contents of `dest` are indeterminate.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

clib.l

See Also

[strcmp\(\)](#)

[strcoll\(\)](#)

Syntax

```
#include <module.h>
void _subcall(void);
```

Description

`_subcall()` calls a subroutine module previously linked by `_sliblink()`. `_subcall()` expects the first two longwords in the code area following it to contain, respectively:

- The subroutine library number.
- The function number to call.

`_subcall()` retrieves these identifiers, adjusts the program stack, and dispatches to the subroutine library entry point with the correct global static storage configuration.



For More Information

For more information, consult the ***OS-9 Technical Manual***.



Note

A process can link to a maximum of sixteen subroutine libraries, numbered from 0 to 15.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

clib.l

See Also

[_sliblink\(\)](#)

sys_mktime()

Convert Local Time to Greenwich Mean Time

Syntax

```
#include <time.h>
time_t sys_mktime(struct tm *tp);
```

Description

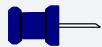
`sys_mktime()` converts the local time values in a broken-down time structure into a Universal Coordinated Time value.

`tp` is a pointer to a broken-down time structure. (Input)

Broken-down time has the following format:

```
struct tm {  
    int tm_sec;          /* seconds after the minute: [0,59] */  
    int tm_min;          /* minutes after the hour: [0,59] */  
    int tm_hour;         /* hours after midnight: [0,23] */  
    int tm_mday;         /* day of the month: [1,31] */  
    int tm_mon;          /* months after January: [0,11] */  
    int tm_year;         /* years after 1900 */  
    int tm_wday;         /* days after Sunday: [0,6] */  
    int tm_yday;         /* days after January 1: [0,365] */  
    int tm_isdst;        /* Daylight Savings Time (DST) flag */  
                        /* 0 = no, 1 = yes, -1 = unknown */  
};
```

`sys_mktime()` ignores the input values of `tm_wday` and `tm_yday`. It does not require the other fields of `tm` to be within the ranges specified in the structure. Instead, it **normalizes** the fields and sets `tm_wday`, `tm_yday`, and `tm_isdst`. This allows you to easily do temporal arithmetic without having to worry about mixed-base operations.



Note

If `(time_t)-1` returns to indicate an error, there is a one second interval that appears to be non-representable. The encoding of times for `time_t` only represents times ranging from approximately 1902 to 2038.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

`sys_clib.l`

See Also

`mktime()`

_sysdate()

Get Current System Date/Time

Syntax

```
#include <time.h>
int _sysdate(
    int      format,
    int      *time,
    int      *date,
    short    *day,
    int      *tick);
```

Description

`_sysdate()` obtains the current time, date, day of week, and clock tick from the system. Note that all the parameters except `format` are pointers to the receiving locations.

<code>format</code>	specifies the format of the date and time to return. (Input) format can be any of the following: <ul style="list-style-type: none">• 0 = Gregorian• 2 = Gregorian with ticks• 1 = Julian• 3 = Julian with ticks
<code>time</code>	is a pointer to the location where <code>_sysdate()</code> stores the time value. (Output)
<code>date</code>	is a pointer to the location where <code>_sysdate()</code> stores the date value. (Output)
<code>day</code>	is a pointer to the location where <code>_sysdate()</code> stores the day of week value. (Output)

`tick`

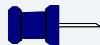
is a pointer to the location where
`_sysdate()` stores the tick value.
(Output)

The values are returned in the following format:

Figure 2-10 `_sysdate()` Return Format

	Byte 0	Byte 1	Byte 2	Byte 3	
Time:	0	Hour (0-23)	Minute	Second	Gregorian
	Seconds Since Midnight (0-86399)				
Date:	Year (2-bytes)	Month	Day		Gregorian
	Julian Day Number				
Day:	Day of Week	0 = Sunday, 1 = Monday, ...			
Tick:	No. of Ticks/Second	Current Clock Tick			

If an error occurs, -1 is returned and the appropriate error code is placed in the global variable `errno`.



Note

Be careful to pass pointers to the `date`, `time`, and `day` values. Also, be sure `day` is declared to be a `short` or the value appears as `day + 65536`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

sys_clib.l

Example

```
main()
{
    int date,time,tick;
    short day;

    .
    .
    .
    _sysdate(0,&time,&date,&day,&tick);
    .
    :
}
```

See Also

[_os_gettime\(\)](#)
[_os_julian\(\)](#)
[F\\$Julian](#)
[F\\$Time](#)
[F_TIME](#)

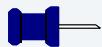
OS-9 for 68K Technical Manual
OS-9 for 68K Technical Manual
OS-9 Technical Manual

Syntax

```
#include <process.h>
int _sysdbg(void);
```

Description

`_sysdbg()` starts the system-level debugger, if one exists. This allows you to debug system state routines, such as drivers. The system-level debugger runs in system state and effectively stops timesharing whenever it is active. Only the super user can make this call.



Note

If the system does not have a system-level debugger, the system is reset.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

sys_clib.l

See Also

[_os_sysdbg\(\)](#)

F\$SysDbg

F_SYSDBG

OS-9 for 68K Technical Manual
OS-9 Technical Manual

Syntax

```
#include <stdlib.h>
int system(const char *string);
```

Description

`system()` passes a string to the shell for execution. A null pointer may be used for the string to inquire whether a command processor exists.

`string` is a pointer to the string to pass to the shell. (Input)

The string may be any valid shell command line.

- If `string` is a null pointer, `system()` returns non-zero only if a command processor is available.
- If `string` is not a null pointer, `system()` returns the exit status of the forked shell.

The shell used to execute the string is either the current value of the `SHELL` environment variable or `shell`, if `SHELL` is unavailable.



Note

The thread enabled version of this function contains a cancel point. For more information see ***Using OS-9 Threads***.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

clib.l

Syntax

```
#include <regs.h>
void *system_addr(void *addr);
```

Description

`system_addr()` returns the system-state version of the specified address. If the passed address is already in the system-state address space or address translation is not applicable on the host processor, it is returned untouched.

`addr` is the address to be converted to its system-state address (Input).

`system_addr()` returns the system-state version of the specified address.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`cpu.l`

See Also

[user_addr\(\)](#)

Syntax

```
#include <math.h>
double tan(double x);
```

Description

`tan()` returns the tangent of `x` (input). `x` is presumed to be in radians.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

Syntax

```
#include <math.h>
double tanh(double x);
```

Description

`tanh()` returns the hyperbolic tangent of `x` (input).

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

Syntax

```
#include <dir.h>
long telldir(DIR *dirp);
```

Description

`telldir()` returns the current location associated with the named directory stream.

<code>dirp</code>	is a pointer to the directory stream. (Input)
-------------------	--

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

[closedir\(\)](#)
[opendir\(\)](#)
[readdir\(\)](#)
[rewinddir\(\)](#)
[seekdir\(\)](#)

Syntax

```
#include <UNIX/os9def.h>
char *tempnam(
    char    *dir,
    char    *pfx);
```

Description

`tempnam()` generates names that can safely be used for temporary files. `tempnam()` allows the user to control the choice of a directory.

`dir`

points to the name of the directory in which the file is to be created. (Input)

If `dir` is NULL or points to a string which is not a name for an appropriate directory, “.” is used. This entire sequence can be preempted by setting the environment variable `TMPDIR`, whose value is the name of the desired temporary-file directory.

`pfx`

Many applications prefer their temporary files to have certain favorite initial letter sequences in their names. Use the `pfx` argument for this. (Input)

This argument may be NULL or point to a string to be used as the first few characters of the temporary-file name.

`tempnam()` uses `malloc()` to get space for the constructed file name, and returns a pointer to this area. Thus any pointer value returned from `tempnam()` may serve as an argument to `free()`. If `tempnam()` cannot return the expected result for any reason, that is `malloc()` failed, or none of the above mentioned attempts to find an appropriate directory was successful, a NULL pointer is returned.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

unix.1

Syntax

```
#include <termcap.h>
extern char *BC;
extern char *UP;
int tgetent(
    char      *bufptr,
    const char *name);
```

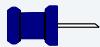
Description

`tgetent()` extracts the entry for the specified terminal type. The entry is placed in a buffer. The buffer size must be at least 1,024 characters and must remain intact for all subsequent calls to `tgetnum()`, `tgetflag()`, and `tgetstr()`.

<code>bufptr</code>	is a pointer to the buffer in which to place the entry. (Output)
<code>name</code>	is a pointer to the name of the terminal type. (Input)

If the `termcap` file cannot be opened, -1 is returned. If the terminal name cannot be found, 0 is returned. If the terminal name is found, 1 is returned and the data is placed in the buffer.

By defining the shell environment variables `TERMCAP` and `TERM`, you can modify the behavior of `tgetent()`. If `TERMCAP` is found in the environment and its value string begins with a slash (/), that string is used as the path to the file to use instead of the default `termcap` file. If the value string does not begin with a slash and `name` is the same as the environment string for `TERM`, `tgetent()` uses this file to search for the terminal entry. This mode is useful for testing or for custom `termcap` definitions. If the `TERMCAP` value string does not begin with a slash, the string is used as the `termcap` string instead of reading a file.



Note

`tgetent()` must be called before any of the other `termcap` library functions.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

`termlib.l`

See Also

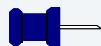
[tgetflag\(\)](#)
[tgetnum\(\)](#)
[tgetstr\(\)](#)

Syntax

```
#include <termcap.h>
extern char *BC;
extern char *UP;
int tgetflag(const char *id);
```

Description

`tgetflag()` returns 1 if the capability ID was specified for the terminal type and 0 if not specified for the terminal type.



Note

`tgetent()` must be called before any of the other `termcap` library functions.

<code>id</code>	is a pointer to the capability ID. (Input)
-----------------	--

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

`termlib.l`

Syntax

```
#include <termcap.h>
extern char *BC;
extern char *UP;
int tgetnum(const char *id);
```

Description

`tgetnum()` returns the numeric value given for the capability ID.



Note

`tgetent()` must be called before any of the other `termcap` library functions.

`id`

is a pointer to the capability ID. (Input)

If `id` was not specified for the terminal type, -1 is returned.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

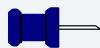
`termlib.l`

Syntax

```
#include <termcap.h>
extern char *BC;
extern char *UP;
char *tgetstr(
    const char      *id,
    char            **strptr);
```

Description

`tgetstr()` places the string given for the capability ID into a buffer.



Note

`tgetent()` must be called before any of the other `termcap` library functions.

<code>id</code>	is a pointer to the capability ID. (Input)
<code>strptr</code>	is a pointer to the pointer to the buffer for the capability string. (Output)
	<code>strptr</code> is advanced past the last character of the returned data.

`tgetstr()` returns null or 0 upon error.

Cursor motion and padding information are interpreted when the string is actually output with `tgoto()` and `tputs()`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

termlib.l

See Also

[tgoto\(\)](#)

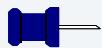
[tputs\(\)](#)

Syntax

```
#include <termcap.h>
extern char *BC;
extern char *UP;
char *tgoto(
            const char *motion_string,
            int         column,
            int         line);
```

Description

`tgoto()` returns a string suitable for positioning the cursor on the terminal.



Note

`tgetent()` must be called before any of the other `termcap` library functions.

`motion_string`

is a pointer to the string given by the `cm` capability. (Input)

`column`

specifies the column destination for the cursor. (Input)

`line`

specifies the line destination for the cursor. (Input)

The `extern` variables `UP` and `BC` (set to the `up` and `bc` capability strings, respectively) are used to avoid placing a null or newline character into the output string.

`tgoto()` returns a pointer to the translated motion string if the motion string was successfully created. Otherwise, the string "OOPS" is returned.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

termlib.l

Syntax

```
#include <time.h>
time_t time(time_t *timer);
```

Description

`time()` determines the current calendar time. The encoding of the value is unspecified.

`timer` is a returned value. It is a pointer to the calendar time. (Output)

`time()` returns the implementation's best approximation to the current calendar time. The value (`time_t`) -1 is returned if the calendar time is unavailable. If `timer` is not a null pointer, the return value is also assigned to the object to which it points.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

See Also

[getenv\(\)](#)
[_os_gettime\(\)](#)
[_os9_gettime\(\)](#)

Syntax

```
#include <UNIX/times.h>
int times(struct tms *buffer);
```

Description

`times()` returns time-accounting information for the current process and for the terminated child processes of the current process. All times are in 1/60 seconds.

`buffer` (output) points to the following structure:

```
struct tms {
    clock_t    tms_utime;    /* user time */
    clock_t    tms_stime;    /* system time */
    clock_t    tms_cutime;   /* user time, children */
    clock_t    tms_cstime;   /* system time, children */
};
```

This information comes from the calling process.

`tms_utime` is the CPU time used while executing instructions in the user space of the calling process.

`tms_stime` is the CPU time used by the system on behalf of the calling process.

`tms_cutime` and `tms_cstime` are not implemented and set to zero by `times()`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

unix.l

Syntax

```
#include <stdio.h>
FILE *tmpfile(void);
```

Description

`tmpfile()` creates a temporary binary file that is automatically removed when it is closed or at program termination. If the program terminates abnormally, temporary files are not removed. The file is opened for update with "wb+" mode.

`tmpfile()` returns a pointer to the stream of the file that it created. If the file cannot be created, `tmpfile()` returns a null pointer.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

clib.l

See Also

[fopen\(\)](#)

Syntax

```
#include <stdio.h>
char *tmpnam(char *filename);
```

Description

`tmpnam()` generates a string. The string is a valid file name and is not the same as the name of an existing file. `tmpnam()` generates a different string each time it is called up to `TMP_MAX` times. If it is called more than `TMP_MAX` times, it returns a null pointer.

filename	is a pointer to the generated file name. (Output)
	<ul style="list-style-type: none">• If <code>filename</code> is a null pointer, <code>tmpnam()</code> leaves its result in an internal static object and returns a pointer to that object. Subsequent calls to <code>tmpnam()</code> modify the same object.• If <code>filename</code> is not a null pointer, it is assumed to point to an array of at least <code>L_tmpnam</code> chars. <code>tmpnam()</code> writes its result in that array and returns the parameter as its value.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

clib.1

Syntax

```
#include <ctype.h>
int toascii(int c);
int toascii(int c);
```

Description

`_toascii()` returns a character with all bits that are not part of a standard ASCII character turned off. It is used for compatibility with other systems. Only `_toascii()` is available in strictly conforming ANSI mode.

c is the returned character. (Input)

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

clib.l

See Also

`getc()`, `getchar()`

_tolower()

Convert Character to Lowercase

Syntax

```
#include <ctype.h>
int _tolower(int c);
```

Description

`_tolower()` is a macro that changes the uppercase parameter to lowercase. The parameter **must** be uppercase or the results are unpredictable and useless. Use `tolower()` if the parameter is not guaranteed to be uppercase.

`c` is the character to convert. (Input)

Only the lower 8 bits of `c` are used.

`_tolower()` is hard-coded to the ASCII character set. Changing the locale does not cause `_tolower()` to recognize a non-ASCII character. You must use `tolower()` if you are not using ASCII characters.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

ctype.h.

See Also

```
tolower()  
toupper()  
isascii(), isascii()
```

Syntax

```
#include <ctype.h>
int tolower(int c);
```

Description

`tolower()` converts an uppercase letter to the corresponding lowercase letter.

`c` is the character to convert. (Input)

If the parameter `c` is such that `islower(c)` returns a non-zero value (is uppercase), `tolower()` returns the corresponding character. Otherwise, `c` is returned unchanged.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

See Also

[islower\(\)](#)

_toupper()

Convert Character to Uppercase

Syntax

```
#include <ctype.h>
int _toupper(int c);
```

Description

`_toupper()` is a macro that changes the lowercase parameter to uppercase. The parameter *must* be lowercase or the results are unpredictable and useless. Use `toupper()` if the parameter is not guaranteed to be lowercase.

`c` is the character to convert. (Input)

Only the lower 8 bits of `c` are used.

`_toupper()` is hard-coded to the ASCII character set. Changing the locale does not cause `_toupper()` to recognize a non-ASCII character. You must use `toupper()` if you are not using ASCII characters.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`ctype.h`

See Also

`tolower()`
`toupper()`
`_isascii()`, `isascii()`

toupper()

Convert Character to Uppercase

Syntax

```
#include <ctype.h>
int toupper(int c);
```

Description

`toupper()` converts a lowercase letter to the corresponding uppercase letter.

`c` is the character to convert. (Input)

If the parameter `c` is such that `isupper(c)` returns a non-zero value (is uppercase), `toupper()` returns the corresponding character. Otherwise, `c` is returned unchanged.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.l`

See Also

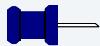
[isupper\(\)](#)

Syntax

```
#include <termcap.h>
extern char *BC;
extern char *UP;
void tputs(
            const char    *str,
            int           lines_affected,
            int           (*outfunc)(char));
```

Description

`tputs()` is used to output the capability strings to the terminal. `tputs()` decodes leading padding information from the `str` string.



Note

`tgetent()` must be called before any of the other `termcap` library functions.

<code>str</code>	is a pointer to the string. (Input)
<code>lines_affected</code>	is the number of lines the operation affects. (Input) This should be set to 1 if not applicable.
<code>outfunc</code>	is a pointer to a function called by <code>tputs()</code> to output each successive character of the <code>str</code> string. (Input)

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

termlib.l

See Also

[_gs_opt\(\)](#)

Syntax

```
#include <signal.h>
int tsleep(unsigned svalue);
```

Description

`tsleep()` deactivates the calling process for a specified interval.

`svalue` specifies the interval below. (Input)

- If `svalue` is 0, the process sleeps indefinitely.
- If `svalue` is 1, the process gives up the current time slice.

For values greater than one, `svalue` is considered a tick count to sleep, if the high bit is clear. If the high bit of `svalue` is set, the remaining 31 bits are considered the number of 256ths of a second to sleep.

If the sleeping process is awakened prematurely by a signal, `tsleep()` returns the number of ticks remaining to sleep.



Note

The thread enabled version of this function contains a cancel point. For more information see [**Using OS-9 Threads**](#).

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

sys_clib.l

See Also

[_os_sleep\(\)](#)

[_os9_sleep\(\)](#)

F\$Sleep

F_SLEEP

OS-9 for 68K Technical Manual

OS-9 Technical Manual

Syntax

```
#include <stdio.h>
int ungetc(
    int      c,
    FILE    *stream);
```

Description

`ungetc()` pushes a character, converted to an `unsigned char`, back onto the input stream. The pushed-back character is returned by subsequent reads on that stream in the reverse order of their pushing. A successful intervening call with the stream pointed to by `stream` to a file positioning function (`fseek()`, `fsetpos()`, or `rewind()`) discards any pushed-back characters for the stream. The external storage corresponding to the stream is unchanged.

`c` is the character to push back. (Input)
If `c` is `EOF`, the operation fails and the input stream is unchanged.
`stream` is a pointer to the C I/O `FILE` structure. (Input)

One character of pushback is guaranteed. If `ungetc()` is called too many times on the same stream without an intervening read or file positioning operation on that stream, the operation may fail.

A successful call to `ungetc()` clears the end-of-file indicator for the stream. The value of the file position indicator for the stream after reading or discarding all pushed-back characters is the same as it was before the characters were pushed back. For a text stream, the value of the file position indicator after a successful call to `ungetc()` is unspecified until all pushed-back characters are read or discarded. For a binary stream, the file position indicator is decremented by each successful call to `ungetc()`. If the value was zero before a call, it is indeterminate after the call.

`ungetc()` returns the character pushed back after conversion. If the conversion fails, `EOF` is returned.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

`clib.l`

See Also

[getc\(\)](#), [getchar\(\)](#)
[fopen\(\)](#)
[fseek\(\)](#)
[fsetpos\(\)](#)
[rewind\(\)](#)
[setbuf\(\)](#)

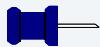
Syntax

```
#include <modes.h>
int unlink(const char *name);
```

Description

unlink() decrements the link count of a file. When the link count reaches zero (0), the space occupied by the file on the disk is freed and the file no longer exists.

name	is a pointer to the name of the file. (Input)
	If successful, unlink() returns 0. Otherwise, -1 is returned and the appropriate error code is placed in the global variable errno.



Note

OS-9 for 68K does not yet support multiple links to a file. unlink() always causes the file to be removed from the disk. Attempting to delete an open file by the calling (or another) process results in an EOS_SHARE error.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

sys_clib.l

See Also

[_os_delete\(\)](#)

[I\\$Delete](#)

[I_DELETE](#)

OS-9 for 68K Technical Manual

OS-9 Technical Manual

unlinkx()

Unlink (Delete) File

Syntax

```
#include <modes.h>
int unlinkx(
    const char      *name,
    int              mode);
```

Description

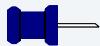
`unlinkx()` decrements the link count of a file.

name is a pointer to the name of the file.(Input)

mode specifies the file's access mode. (Input)

If the execution bit (FAM_EXEC or S_IEXEC) in the mode is set, the path is assumed to be based in the current execution directory. The header file modes.h defines the legal mode values. When the link count reaches zero, the space occupied by the file on the disk is freed and the file no longer exists. Only the lower 16 bits of mode are used.

If successful, `unlinkx()` returns 0. Otherwise, -1 is returned and the appropriate error code is placed in the global variable `errno`.



Note

OS-9 for 68K does not support multiple links to a file. `unlinkx()` always causes the file to be removed from the disk. Attempting to delete an open file by the calling (or another) process results in an `EOS_SHARE` error.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library

sys_clib.l

See Also

[_os_delete\(\)](#)
[I\\$Delete](#)
[I_DELETE](#)

OS-9 for 68K Technical Manual
OS-9 Technical Manual

user_addr()

Address Translation Function

Syntax

```
#include <regs.h>
void *user_addr(void *addr);
```

Description

`user_addr()` returns the user-state version of the specified address. If the passed address is already in the user-state address space or address translation is not applicable on the host processor, it is returned untouched.

addr is the address to be converted to its user-state address. (Input)

`user_addr()` returns the user-state version of the specified address.

Attributes

Operating System:	OS-9
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

cpu.l

See Also

system_addr()

Syntax

```
#include <stdarg.h>
type va_arg(va_list ap, type);
```

Description

`va_arg()` is a macro associated with the `stdarg.h` header file. `stdarg.h` provides a method for stepping through a list of function parameters whose length and type are unknown.

`ap` is an object representing the next unnamed parameter. (Input)

`va_arg()` expands into an expression. The expression returns a value that has the type and value of the next unnamed parameter. It also modifies `ap` such that the next call to `va_arg()` returns the next parameter. `ap` must have been previously initialized using `va_start()`.

If there is no actual next parameter or if `type` is not compatible with the type of the actual next parameter (as promoted according to the default parameter promotions), the behavior is undefined.

Attributes

Operating System: OS-9 and OS-9 for 68K

State: User and System

Threads: Safe

Re-entrant: Yes

Standards: ANSI

Library

`clib.l`

See Also

[va_start\(\)](#)

va_end()

End References to Current Variable Parameter List

Syntax

```
#include <stdarg.h>
void va_end(va_list ap);
```

Description

va_end() is a macro associated with the stdarg.h header file. stdarg.h provides a method for stepping through a list of function parameters whose length and type are unknown.

ap is an object representing the next unnamed parameter. (Input)

va_end() expands into an expression. The expression facilitates a normal return from the function whose variable parameter list was referred to by the expansion of the va_start() call that initialized ap.

va_end() may modify ap so that you can no longer use it without an intervening call to va_start(). If there is no corresponding call to va_start() or if va_end() is not called before the return, the behavior is undefined. va_end() returns no value.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

clib.l

See Also

[va_start\(\)](#)

Syntax

```
#include <stdarg.h>
void va_start(va_list ap, parmN);
```

Description

`va_start()` is a macro associated with the `stdarg.h` header file. `stdarg.h` provides a method for stepping through a list of function parameters whose length and type are unknown.

ap	is an object representing the next unnamed parameter. (Input)
parmN	identifies the right most parameter in the variable parameter list in the function definition (the parameter just before the <code>, ...</code>). (Input)
	If <code>parmN</code> is declared with one of the following types, the behavior is undefined: <ul style="list-style-type: none">• The register storage class• A function type• An array type• A type that is not compatible with the type that results after applying the default parameter promotions. For example, <code>char</code>, <code>short</code>, and <code>float</code>.

`va_start()` returns no value.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

clib.l

Syntax

```
#include <stdarg.h>
#include <stdio.h>
int vfprintf(
    FILE           *stream,
    const char     *format,
    va_list        arg);
```

Description

`vfprintf()` performs formatted output to a stream. It converts, formats, and prints any parameters as indicated by the control string. `vfprintf()` is similar to `fprintf()`. However, `arg` replaces the variable parameter list.

stream	is a pointer to the C I/O <code>FILE</code> structure. (Input)
format	is a pointer to a control string. (Input)
arg	is initialized by <code>va_start()</code> (and possibly subsequent <code>va_arg()</code> calls). (Input)

`vfprintf()` does not call `va_end()`. `vfprintf()` returns the number of characters transmitted. If an output error occurs, a negative value is returned.



For More Information

The `sclib.l` and `sclib.il` libraries contain a smaller version of the `vfprintf()` function. Refer to the ***Using Ultra C/C++*** manual for more information about the small library functions.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

clib.l

Example

The following shows the use of vfprintf() in a general error-reporting routine.

```
#include <stdarg.h>
#include <stdio.h>
void error(char *function_name, char *format, ...)
{
    va_list args;
    va_start(args, format); /* print out name of function */
                           /* causing error */
    fprintf(stderr, "ERROR in %s ", function_name);
                           /* print out remainder of message */
    vfprintf(stderr, format, args);
    va_end(args);
}
```

See Also

[fprintf\(\)](#)
[va_arg\(\)](#)
[va_end\(\)](#)
[va_start\(\)](#)

Syntax

```
#include <stdarg.h>
#include <stdio.h>
int vprintf(
            const char    *format,
            va_list       arg);
```

Description

`vprintf()` performs formatted output to `stdout`. It converts, formats, and prints any parameters as indicated by the control string. `vprintf()` is similar to `printf()`. However, `arg` replaces the variable parameter list.

<code>format</code>	is a pointer to a control string. (Input)
<code>arg</code>	is initialized by <code>va_start()</code> (and possibly subsequent <code>va_arg()</code> calls). (Input)

`vprintf()` does not call `va_end()`.

`vprintf()` returns the number of characters transmitted. If an output error occurs, a negative value is returned.



For More Information

The `sclib.l` and `sclib.il` libraries contain a smaller version of the `vprintf()` function. Refer to the ***Using Ultra C/C++*** manual for more information about the small library functions.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

clib.l

See Also

[fprintf\(\)](#)
[printf\(\)](#)
[va_arg\(\)](#)
[va_end\(\)](#)
[va_start\(\)](#)

Syntax

```
#include <stdarg.h>
#include <stdio.h>
int vsprintf(
    char          *str,
    const char    *format,
    va_list       arg);
```

Description

`vsprintf()` performs formatted output to a string. It converts, formats, and prints any parameters as indicated by the control string.

`vsprintf()` is similar to `sprintf()`. However, `arg` replaces the variable parameter list.

<code>str</code>	is a pointer to a string. (Output)
<code>format</code>	is a pointer to a control string. (Input)
<code>arg</code>	is initialized by <code>va_start()</code> (and possibly subsequent <code>va_arg()</code> calls). (Input)

`vsprintf()` does not call `va_end()`. If copying takes place between overlapping objects, the behavior is undefined.

`vsprintf()` returns the number of characters written in the array, not counting the terminating null character.



For More Information

The `sclib.l` and `sclib.il` libraries contain a smaller version of the `vsprintf()` function. Refer to the ***Using Ultra C/C++*** manual for more information about the small library functions.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No
Standards:	ANSI

Library

clib.l

See Also

[fprintf\(\)](#)
[sprintf\(\)](#)
[va_arg\(\)](#)
[va_end\(\)](#)
[va_start\(\)](#)

Syntax

```
#include <process.h>
int wait(unsigned int *status);
int wait(0);
```

Description

`wait()` suspends the current process until a child process has terminated.

`status` is a pointer to a storage location where the exit status of the terminating child process is stored. (Output)

`wait()` returns the process ID of the terminating child process and places the exit status of that process in the `unsigned int` pointed to by `status`. If `status` is passed as zero, no child status is available to the caller.

The lower 16 bits of the `status` value contains the parameter of the `exit()` or `_exit()` call as executed by the child process, or the signal number if it was interrupted with a signal. A normally terminating C program with no explicit call to `exit()` in `main()` returns a status equal to the returned value of `main()`.

`wait()` returns:

- -1 if there is no child process for which to wait.
- 0 if a signal was received before a child process terminated.



Note

The status codes used in the operating system may not be compatible with other operating systems. A `wait` must be executed for each child process created.



Note

The thread enabled version of this function contains a cancel point. For more information see ***Using OS-9 Threads***.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

sys_clib.l

See Also

[exit\(\)](#)
[_exit\(\)](#)
[_os_wait\(\)](#)
[_os_fork\(\)](#)
[F\\$Wait](#)
[F_WAIT](#)

OS-9 for 68K Technical Manual
OS-9 Technical Manual

Syntax

```
#include <sys/wait.h>
pid_t waitpid(
    pid_t      pid,
    int       *stat_loc,
    int       options);
```

Description

`waitpid()` allows a thread to wait for a process and get its exit status.

`pid` and `options`

The `pid` and `options` arguments determine how the wait is done. (Input)

For values of `pid` less than or equal to 0, the next available child process' status is returned. For values of `pid` greater than 1 (1 is an invalid process ID) the exit status of the specified process is returned. The `options` parameter can either be 0 to indicate a blocking wait or `WNOHANG` to indicate a non-blocking wait.

If or when a child's exit status is available `waitpid()` returns the process ID and, if `stat_loc` (output) is non-NULL, writes the exit status at `stat_loc`. If a blocking `waitpid()` is interrupted by a signal it sets `errno` to `EINTR` and returns -1. If `options` is `WNOHANG` and the calling process has children, but none of the specified children has available exit status, 0 is returned. Otherwise `errno` is set to the error number and -1 is returned.

Attributes

Operating System:

OS-9 and OS-9 for 68K

State:

User

Threads:

Safe

Re-entrant:

Yes

Library

sys_clib.l

Possible Errors

ECHLD	The calling process has no children or not the specific child requested.
EINTR	A blocking <code>waitpid()</code> was interrupted by a signal.
EINVAL	The value of <code>options</code> is invalid.

Example

```
while (waitpid(child, &status, 0) == -1 && errno == EINTR)  
;
```

See Also

[wait\(\)](#)
[pthread_join\(\)](#)
[pthread_exit\(\)](#)

Syntax

```
#include <stdlib.h>
size_t wcstombs(
    char          *str,
    const wchar_t *pwcs,
    size_t         maxnum);
```

Description

`wcstombs()` converts a sequence of codes that correspond to multibyte characters from an array into a sequence of multibyte characters that begins in the initial shift state. `wcstombs()` stores these multibyte characters, stopping if a multibyte character would exceed the limit of `maxnum` total bytes or if a null character is stored. Each code is converted as if by a call to `wctomb()`, except that the shift state of `wctomb()` is not affected.

<code>str</code>	is a pointer to the string in which to store the multibyte characters. (Output)
<code>pwcs</code>	is a pointer to the array from which to read. (Input)
<code>maxnum</code>	specifies the maximum number of bytes to convert. (Input)

No more than `maxnum` bytes are modified in `str`. If copying takes place between overlapping objects, the behavior is undefined.

If a code is encountered that does not correspond to a valid multibyte character, `wcstombs()` returns `(size_t)-1`. Otherwise, `wcstombs()` returns the number of bytes modified, not including a terminating null character, if any.

`wcstombs()` is supported for both the C and JAPAN locales.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

clib.l

See Also

[wctomb\(\)](#)

Syntax

```
#include <stdlib.h>
int wctomb(
    char      *str,
    wchar_t   wchar);
```

Description

`wctomb()` determines the number of bytes needed to represent the multibyte character corresponding to the code whose value is `wchar`. This includes any change in shift state. `wctomb()` stores the multibyte character representation in `str`, if `str` is not a null pointer. At most `MB_CUR_MAX` characters are stored.

<code>str</code>	is a pointer to the string in which to store the multibyte character representation. (Output)
<code>wchar</code>	is the multibyte character to be represented. (Input) If <code>wchar</code> is zero, <code>wctomb()</code> is left in the initial shift state.

If `str` is a null pointer, `wctomb()` returns the following:

- A non-zero value if multibyte character encodings have state-dependent encodings.
- Zero if multibyte character encodings do not have state-dependent encodings.

If `str` is not a null pointer, `wctomb()` returns the following:

- -1 if the value of `wchar` does not correspond to a valid multibyte character
- The number of bytes contained in the multibyte character corresponding to the value of `wchar`.

The returned value is never greater than `MB_CUR_MAX`.

`wctomb()` is supported for both the `C` and `JAPAN` locales.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes
Standards:	ANSI

Library

`clib.lib`

Syntax

```
#include <modes.h>
int write(
    int             path,
    const char     *buffer,
    unsigned        count);
```

Description

`write()` writes bytes to a path.

`path`

is the path number. (Input)

The path number is an integer which specifies one of the standard path numbers (0, 1, or 2) or a path number returned from a successful call to `open()`, `creat()`, `create()`, or `dup()`.

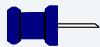
`buffer`

is a pointer to space with at least `count` (input) bytes of memory from which `write()` obtains the data to write to the path. (Output)

It is guaranteed that at most `count` bytes are written. If fewer bytes are written, an error has occurred.

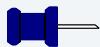
`write()` essentially performs a **raw write**; that is, the `write` is performed without translating characters. The characters are passed to the file (or device) as transmitted by the program.

`write()` returns the number of bytes actually written. If an error occurred, `-1` is returned and the appropriate error code is placed in the global variable `errno`.



Note

`write()` performs a **raw** write to the terminal; no linefeeds after returns are transmitted.



Note

The thread enabled version of this function contains a cancel point. For more information see ***Using OS-9 Threads***.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`creat()`
`create()`
`dup()`
`open()`
`_os_write()`
`writeln()`
`I$Write`
`I_WRITE`

OS-9 for 68K Technical Manual
OS-9 Technical Manual

Syntax

```
#include <modes.h>
int writeln(
    int             path,
    const char     *buffer,
    unsigned        count);
```

Description

`writeln()` writes bytes to a path.

`path` is the path number. (Input)

The path number is an integer which specifies one of the standard path numbers (0, 1, or 2) or a path number returned from a successful call to `open()`, `creat()`, `create()`, or `dup()`.

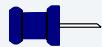
`buffer` is a pointer to space with at least `count` (input) bytes of memory from which `writeln()` obtains the data to write to the path. (Output)

It is guaranteed that at most `count` bytes are written.

`writeln()` causes output-filtering to take place such as outputting a linefeed after a carriage return or handling the page pause facility.

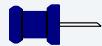
`writeln()` writes, at most, one line of data. A carriage return indicates the end of a line. `writeln()` is the preferred call for writing to the terminal.

`writeln()` returns the number of bytes actually written. If an error occurred, -1 is returned and the appropriate error code is placed in the global variable `errno`.



Note

`writeln()` stops when the carriage return is written, even if the byte count has not been exhausted.



Note

The thread enabled version of this function contains a cancel point. For more information see ***Using OS-9 Threads***.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	Yes

Library

`sys_clib.l`

See Also

`creat()`
`create()`
`dup()`
`open()`
`_os writeln()`
`write()`
`I$WriteLn`
`I_WRITELN`

OS-9 for 68K Technical Manual
OS-9 Technical Manual

Syntax

```
#include <UNIX/os9def.h>
int writev(
            unsigned int      fd,
            struct iovec     *iov,
            unsigned int      iovcnt);
```

Description

`writev()` performs the same action as `write()`, but gathers the output data from the `iovcnt` buffers specified by the members of the `iov` array: `iov[0]`, `iov[1]`, ..., `iov[iovcnt-1]`. If `iovcnt` is zero, any `iov_len` member of a passed `iovec` structure is less than zero, or all the `iov_len` members of the passed `iovec` structures are zero, `writev()` returns -1 and sets the global variable `errno`.

For `writev()`, the `iovec` structure is defined as follows.

```
struct iovec {
    caddr_t      iov_base;
    int iov_len;
};
```

Each `iovec` entry specifies the base address and length of an area in memory from which data should be written. `writev()` always writes a complete area before proceeding to the next.

Attributes

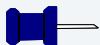
Operating System:	OS-9 and OS-9 for 68K
State:	User and System
Threads:	Safe
Re-entrant:	No

Library

`unix.l`

Appendix A: Prototyping sys_clib.l Functions

Before this topic is discussed in depth, it is important to note that the primary purpose of `sys_clib.l` is to provide a library for compatibility with the Microware K&R C compiler. This allows people who wrote code for that compiler to use the Ultra C/C++ ANSI C compiler without altering their original code.



Note

It is strongly recommended that when writing new code, a programmer not use the `sys_clib` library functions.

Table 1-38 provides a table to look up equivalent functions for the `sys_clib` functions.

Function Declarations and K&R Compiler vs. Prototypes and ANSI C

In the old days when K&R C ruled the earth, the concept of a function prototype did not exist. The closest equivalent concept was a "function declaration". The purpose of the function declaration was to "pre-define" what the return type of a function was so that some degree of type checking could be done on its use. An example of a function declaration is:

```
struct direct *readdir();
```

This shows that the function "readdir" returns a pointer to a structure of the type "direct" (found in dir.h). NOTE: There is nothing in the declaration which mentions what type, or even how many, arguments the readdir function takes. As a result, no type checking was ever done on these functions parameters. The "standard promotion" of arguments was always assumed to have occurred.

The "standard promotion" rules are quite simple, all types being passed in that are "narrower" than an int type, are "widened" to an int type, and all arguments that are of floating point type are "widened" to a double precision. As a natural consequence, all functions which are compiled in K&R mode (for instance, all the sys_clib.l functions) assume that the calling function has already promoted the function arguments to their widened forms.

ANSI C prototypes are somewhat different. An example of a prototype is:

```
int os9fork(char *, int, char *, short, short, int short);
```

This shows that the function "os9fork" returns an int and accepts the parameters specified. Of particular interest is the three "short" parameters. In ANSI mode, the compiler assumes that the prototype was visible to the calling program, and as a result, the values were NOT widened to int. If the prototype was not visible to the calling function, the values would have been widened. This could have potentially caused a problem (especially if the parameters were passed on the

stack and NOT in registers). As a result, the Ultra C compiler has an option (-cw) which warns the user if a function is being called without a prototype for that function being visible.

"ANSI"fyng Code and Function Prototypes

It has been a common practice as of late to take code and switch it over to ANSI-compliant code. This often entails prototyping many of the functions which used to be compiled with K&R C compilers. If this were done strictly as in the prototype from the "os9fork" function mentioned above, and the os9fork function was not recompiled with an ANSI compiler, things would not work since the function itself would be expecting the short arguments to be widened to ints and the calling program, after having seen the prototype, would send shorts.

There is really only one way to handle this in a manner that causes everything to still work and that is to prototype the functions to their widened types, as follows:

```
int os9fork(char *, int, char *, int, int, int, int);
```

Unfortunately, this has the undesired side effect of implying that the fourth, fifth and seventh arguments are valid integers, when indeed only their lower 16 bits are valid. As a result, Microware has decided to optionally prototype these functions in system header files.

sys_clib.l functions affected by this include:

```
os9exec()  
os9fork(), os9forkc()  
os9kexec()
```

Index

Symbols

_aton() 42
_atou() 42
_atou(alpha)
 to unsigned conversion 158
_cleareof() 51, 110, 184
_cmpnam() 71, 110, 191
_cpymem() 67, 110, 195
_dtoa() 42
_environ 107
_errmsg() 71, 109, 219
_ERROR 156
_ev_create() 57, 109, 221
_ev_delete() 57, 109, 223
_ev_info() 57, 109, 224
_ev_link() 58, 109, 226
_ev_pulse() 58, 109, 227
_ev_read() 58, 109, 229
_ev_set() 58, 110, 231
_ev_setr() 58, 110, 233
_ev_signal() 58, 110, 235
_ev_unlink() 58, 110, 237
_ev_wait() 58, 110, 239
_ev_waitr() 58, 110, 241
_exit() 39, 110, 253
_fileno() 52, 271
_freemem() 67, 294
_freemin() 296
get<name>() 315
_get_module_dir() 74, 111, 336
_get_process_desc() 111, 343
_get_process_table() 82, 111, 345
_getsys() 79, 111, 360
_gregorian() 93, 112, 368

_gs_devn() 43, 112, 370
_gs_eof() 44, 112, 372
_gs_gfd() 44, 112, 374
_gs_opt() 44, 112, 376
_gs_pos() 44, 112, 378
_gs_rdy() 44, 112, 380
_gs_size() 44, 112, 382
_INTEGER 156
_IOFBF 1183
_IOLBF 1183
_IONBF 1183
_isascii() 40, 406
_isjis_kana() 41, 414, 416, 420
_issjis1() 41, 428
_issjis2() 41, 430
_julian() 93, 113, 439
_lalloc() 67, 444
_lfree() 67, 448
_lmalloc() 67, 451
_lrealloc() 67, 462
_mallocmin() 68, 476
_mkdata_module() 74, 111, 488
_os_acqlk() 85, 536
_os_alarm_atime() 59, 540
_os_alarm_cycle() 59, 544
_os_alarm_delete() 59, 546
_os_alarm_reset() 59, 547
_os_alarm_set() 59, 549
_os_alias() 53, 551
_os_alltsk() 68, 557
_os_alocproc() 82, 558
_os_altmdir() 75, 559
_os_aproc() 82, 560
_os_attach() 53, 562
_os_cache() 71, 564
_os_caqlk() 85, 567
_os_chain() 83, 568
_os_chainm() 75, 571
_os_chdir() 53, 573
_os_chkmem() 575
_os_chmdir() 75, 577

_os_close() 53, 578
_os_clrsigs() 59, 579
_os_cmdperm() 75, 580
_os_cmpnam() 71, 581
_os_config() 584
 configure 584
_os_cpy_ioproc() 53, 585
_os_cpymem() 68, 586
_os_crc() 75, 588
_os_create() 53, 590
_os_crlk() 85, 596
_os_datmod() 75, 597
_os_ddlk() 85, 599
_os_delete() 53, 603
_os_dellk() 85, 605
_os_delmdir() 75, 606
_os_deltsk() 68, 607
_os_detach() 53, 609
_os_dexec() 72, 610
_os_dexit() 72, 613
_os_dfork() 83, 614
_os_dforkm() 83, 618
_os_dup() 53, 620
_os_ev_allclr() 59, 622
_os_ev_allset() 59, 624
_os_ev_anyclr() 59, 626
_os_ev_anyset() 59, 628
_os_ev_change() 59, 630
_os_ev_creat() 59, 632
_os_ev_delete() 60, 635
_os_ev_info() 60, 636
_os_ev_link() 60, 639
_os_ev_pulse() 60, 640
_os_ev_read() 60, 642
_os_ev_set() 60, 643
_os_ev_setand() 60, 645
_os_ev_setor() 60, 647
_os_ev_setr() 60, 649
_os_ev_setxor() 60, 651
_os_ev_signal() 60, 653
_os_ev_tstset() 61, 655

_os_ev_unlink() 61, 657
_os_ev_wait() 61, 658
_os_ev_waitr() 61, 662
_os_exec() 83, 666
_os_exit() 40, 677
_os_findpd() 83, 679
_os_firq() 64, 682
_os_fmod() 75, 684
_os_fork() 83, 685
_os_forkm() 83, 690
_os_get_blkmap() 693
_os_get_ioproc() 53, 701
_os_get_mdp() 76, 702
_os_get_moddir() 76, 703
_os_get_module() 703
_os_get_prtbl() 83, 710
_os_getdl() 53, 695
_os_gettime() 94, 696
_os_getpd() 72, 705
_os_getstat() 44, 83, 712
_os_getsys() 44, 83, 714
_os_gprdsc() 83, 716
_os_gregorian() 94, 718
_os_gs_cpypd() 44, 721
_os_gs_cstats() 44, 722
_os_gs_devnm() 44, 723
_os_gs_devtyp() 44, 725
_os_gs_dopt() 44, 727
_os_gs_dsize() 44, 728
_os_gs_edt() 45, 729
_os_gs_eof() 45, 730
_os_gs_fd() 45, 731
_os_gs_fdaddr() 45, 732
_os_gs_fdinf() 45, 733
_os_gs_luopt() 45, 736
_os_gs_parity() 45, 737
_os_gs_popt() 45, 738
_os_gs_pos() 45, 740
_os_gs_ready() 45, 743
_os_gs_size() 45, 744
_os_id() 84, 745

_os_initdata() 76, 748
_os_intercept() 61, 749
_os_iocfg 751
 configure 751
_os_iodel() 53, 54, 752
_os_ioexit() 54, 753
_os_iofork() 54, 754
_os_irq() 64, 756
_os_julian() 94, 760
_os_link() 76, 769
_os_linkm() 76, 771
_os_load() 76, 773
_os_loadp() 76, 776
_os_mkdir() 54, 778
_os_makmdir() 76, 780
_os_mem() 68, 781
_os_mkmodule() 783
_os_modaddr() 76, 786
_os_move() 787
_os_nproc() 84, 788
_os_open() 54, 789
_os_permit() 69, 792
_os_perr() 72, 794
_os_protect() 69, 796
_os_prsnam() 72, 798
_os_rdalst() 54, 801
_os_read() 54, 802
_os_readln() 54, 804
_os_rellk() 86, 806
_os_rte() 64, 808
_os_rtnprc() 84, 809
_os_salarm_atime() 61, 812
_os_salarm_cycle() 61, 816
_os_salarm_delete() 62, 820, 821
_os_salarm_reset() 62, 822
_os_salarm_set() 62, 824
_os_scache() 72, 828
_os_seek() 43, 835
_os_sema_init() 62, 836
_os_sema_p() 62, 838
_os_sema_term() 62

_os_sema_v() 62, 841
_os_send() 62, 842
_os_setcrc() 76, 844
_os_setime() 94, 846
_os_setpr() 84, 850
_os_setstat() 45, 851
_os_setsvc() 72, 853
_os_setsys() 46, 857
_os_setuid() 72, 859
_os_sgetstat() 46, 860
_os_sgs_devnm() 46, 861
_os_sigmask() 63, 865
_os_sigreset() 63, 867
_os_sigrs() 63, 869
_os_sleep() 84, 870
_os_slink() 76, 874
_os_slinkm() 77, 876
_os_srqmem() 69, 878
_os_srtmem() 69, 882
_os_ss_attr() 46, 883
_os_ss_break() 46, 884
_os_ss_cache() 46, 885
_os_ss_close() 46, 886
_os_ss_dcoff() 46, 887
_os_ss_dcon() 46, 888
_os_ss_dopt() 46, 889
_os_ss_dsrts() 46, 890
_os_ss_enrts() 46, 891
_os_ss_erase() 46, 892
_os_ss_fd() 47, 893
_os_ss_fillbuff() 47
_os_ss_flushmap() 47, 895
_os_ss_hdlink() 47, 896
_os_ss_lock() 47, 898
_os_ss_luopt() 47, 899
_os_ss_popt() 47, 901
_os_ss_relea() 47, 902
_os_ss_rename() 47, 903
_os_ss_reset() 47, 904
_os_ss_reten() 905
_os_ss_rfm() 47

_os_ss_sendsig() 48, 907
_os_ss_size() 48, 908
_os_ss_skip() 48, 909
_os_ss_skipend() 48, 910
_os_ss_ticks() 48, 911
_os_ss_wfm() 48, 912
_os_ss_wtrack() 48, 913
_os_strap() 72, 915
_os_suspend() 84, 921
_os_sysdbg() 79, 922
_os_sysid() 73, 924
_os_tlink() 77, 932
_os_tlinkm() 77, 934
_os_uacct() 73, 941
_os_unlink() 77, 943
_os_unload() 77, 944
_os_vmodul() 77, 945
_os_wait() 84, 949
_os_waitlk() 86, 954
_os_write() 54, 956
_os_writeln() 54, 958
_os9_alarm_atdate() 58, 538
_os9_alarm_atjul() 58, 542
_os9_allbit() 71, 553
_os9_allpd() 82, 555
_os9_aproc() 82, 561
_os9_create() 53, 593
_os9_delbit() 72, 601
_os9_dfork() 83, 616
_os9_ev_wait() 61, 660
_os9_ev_waitr() 61, 664
_os9_findpd() 72, 680
_os9_gettime() 94, 698
_os9_gs_cdfd() 44, 720
_os9_gs_free() 45, 735
_os9_gspump() 83, 742
_os9_id() 84, 747
_os9_iqueue() 54, 755
_os9_irq() 64, 758
_os9_panic() 72, 791
_os9_retpd() 72, 807

_os9_salarm_atdate() 61, 810
_os9_salarm_atjul() 61, 814
_os9_salarm_cycle() 62, 818
_os9_salarm_set() 62, 826
_os9_schbit() 72, 833
_os9_setime() 94, 848
_os9_setsvc() 72, 855
_os9_sleep() 84, 872
_os9_srqmem() 69, 880
_os9_ss_open() 47, 900
_os9_strap() 73, 918
_os9_translate() 69, 939
_os9_vmodul() 77, 947
_parsepath() 73, 112, 964
_prgname() 73, 113, 978
_prsnam() 73, 113, 981
_pthread_attr_getpriority() 987
_pthread_attr_setpriority() 1000
_pthread_getstatus() 1045
_pthread_interrupt() 1050
_pthread_interrupt_clear() 1052
_pthread_resume() 1093
_pthread_setpr() 1100
_pthread_setsignalrange() 1102
_pthread_setsuspendable() 1106
_pthread_setunsuspendable() 1108
_pthread_suspend() 1110
_REAL 156
set<name>() 1151
_setcrc() 77, 114, 1162
_setsys() 79, 114, 1180
_srqcsiz 1206
_srqmem() 69, 115, 1209
_srqsiz 1209
_srtmem() 69, 115, 1211
_ss_attr() 48, 115, 1213
_ss_dcoff() 48, 115, 1217
_ss_dcon() 48, 115, 1218
_ss_dsrts() 48, 116, 1219
_ss_enrts() 48, 113, 1220
_ss_lock() 48, 113, 1221

_ss_opt() 48, 113, 1223
_ss_pfd() 49, 113, 1225
_ss_rel() 49, 114, 1227
_ss_rest() 49, 114, 1228
_ss_size() 49, 114, 1229
_ss_ssig() 49, 114, 1230
_ss_tiks() 49, 114, 1232
_ss_wtrk() 49, 114, 1234
_stacksiz() 73, 1236
_strass() 73, 114, 1239
_subcall() 77, 1274
_sysdate() 94, 115, 1278–??, 1279–1280
_sysdbg() 79, 115, 1281
_toascii() 42, 1303
_tolower() 42, 1304
_toupper() 42, 1306

Numerics

80x86-specific functions

get
 current task segment 95, 329
 global descriptor pointer 95, 334
get_current_tss() 95, 329
get_excpt_base() 95, 333
get_gdtr() 95, 334
list of 95
make
 global descriptor table entry 95, 468
 interrupt descriptor table entry 95, 470
make_gdesc() 95, 468
make_idesc() 95, 470
return exception table base address 95, 333
set
 exception table base address 95, 1164
 global descriptor pointer 95, 1165
set_excpt_base() 95, 1164
set_gdtr() 95, 1165

abort() 39, 107, 130
 abnormal program termination 130
abs() 65, 131
absolute value
 abs 131
 compute 443
 fabs() 257
 floating 257
 labs() 443
 return 443
access permissions
 change for module directory 580
access() 108, 132
accounting 941
acos() 65, 133
acquire
 ownership of resource lock 536
 conditional 567
activate next process waiting to acquire lock 954
active process queue
 _os_aproc 560
 _os9_aproc() 561
 insert process 560, 561
add
 device to IRQ table 756, 758
add filename to root directory variable
 buildpath() 165
address translation function
 system_addr() 1285
 user_addr() 1318
Address, Get Stack 989
alarm() 134
alarm.h 15
alarms
 _os_alarm_atime() 540
 _os_alarm_cycle() 544
 _os_alarm_delete() 546
 _os_alarm_reset() 547
 _os_alarm_set() 549

_os_salarm_atime() 812
_os_salarm_cycle() 816
_os_salarm_delete() 820, 821
_os_salarm_reset() 822
_os_salarm_set() 824
_os9_alarm_atdate() 538
_os9_alarm_atjul() 542
_os9_salarm_atdate() 810
_os9_salarm_atjul() 814
_os9_salarm_cycle() 818
_os9_salarm_set() 826
alm_atdate() 136
alm_atjul() 138
alm_cycle() 140
alm_delete() 142
alm_set() 143
delete 142
execute system-state subroutine 818
remove
 cyclic alarm 546
 pending request 142, 546, 820, 821
reset
 alarm clock 822
 existing request 547
send
 after specified interval 826
 at Gregorian date/time 810
 at Julian date/time 814
 cyclic 818
 signal after specified interval 143, 549
 signal after specified seconds 134
 signal at Gregorian date/time 136, 538
 signal at Julian date/time 138, 542
 signal at specific time 140, 540, 544, 812
 system-state subroutine 810, 814, 826
 set alarm clock 816, 824
alias
 _os_rdalst() 801
 copy system alias list 801
alloca() 135
allocate

_lalloc() 444
colored memory 1206
fixed-length memory block 555
low-overhead 444
memory from arena 451, 475
process descriptor 558
resource lock descriptor 596
storage for array 167, 444
task 557
allocate bytes of space in stack frame 135
alm_atdate() 57, 108, 136
alm_atjul() 57, 109, 138
alm_cycle() 57, 109, 140
alm_delete() 57, 109, 142
alm_set() 57, 109, 143
alpha
 to floating conversion 153
 to integer conversion 154
 to long conversion 155
 to numeric translation 156
alphanumeric 402
append mode
 file position indicator 105
arc
 cosine 133
 sine 147
 tangent 150, 151
array
 allocate storage 167
 search 163
asctime() 93, 145
asin() 65, 147
assert() 71, 97, 148
assert.h 15
Associate a Thread Value with a Key 1104
AST 90
at Exit, Execute Specified Function 1012
at Top of Stack, Remove Routine 1010
atan() 65, 150
atan2() 65, 151
atexit() 39, 152

atof() 42, 153
atoi 154
atoi() 42
atol() 42, 155
aton() 156
aton.h 16
attach
 device 159, 562
 path to file pointer 260
attach() 50, 109, 159
Attribute Block, Discontinue 982
Attribute Priority, Get 987
Attribute, Get Detach State 983
Attribute, Get Value of 1064, 1077, 1079, 1081
Attribute, Set Process-Shared 1034, 1070, 1085, 1087, 1089

B

base-ten logarithm 459
bcmpl() 161
bcopy() 162
binary file
 create temporary 1301
 tmpfile() 1301
bit index find
 ffs() 265
bit map
 _os9_allbit() 553
 _os9_delbit() 601
 _os9_schbit() 833
 deallocate 601
 flush cached information 895
 search for free area 833
 set bits 553
Block on a Condition Variable 1026
Block, Discontinue Attribute 982
break
 serial connection 884
 string into tokens 1264
broken-down time
 convert to

calendar time 493
string format 145
structure 1276
bsearch() 86, 163
buildpath() 165
bypass macro replacement 8
bzero 166

C

cache
control 564, 828
status
 _os_gs_cstats() 722
 get information 722
cache.h 16
calculate
 module CRC 197
 parity of file descriptor 737
calendar time 367
call
 register function 152
 subroutine module 1274
 system debugger 922, 1281
Calling Thread, Set OS-9 Priority for 1100
Calling Thread, Terminate 1041
calloc() 67, 107, 167
Cancel State, Set 1096
Cancel Target Thread 1008
Cancel Type, Set 1098
Cancel, Test for Pending 1112
capability ID
 check 1293
 get 1294, 1295
 tgetflag() 1293
 tgetnum() 1294
 tgetstr() 1295
capability string
 output 1308
 tputs() 1308
cbrt() 169

cdi.h 16
CDT 90
ceil() 65, 170
ceiling function 170
CET 90
chain() 171
chainc() 82, 109, 171
change
 current constant data pointer 174
 current data directory 176
 current execution directory 182
 current static storage pointer 175
 file access permissions 178
 module directory 577
 owner of file 180
 permissions of module directory 580
 system global variable 1180
 working directory 573
change_static() 71, 175
CHAR_MAX 453, 454, 455
character
 get from file 266
 get from I/O address 386
 get next from file/stdin 326
 multibyte 477, 479, 481, 1333, 1335
 output to file 289
 push back 1312
 put next to file/stdout 1113
 rindex() 1140
 search for 387, 1140
 standard white space 432
 unget 1312
 ungetc() 1312
 write to I/O address 961
character classification
 _isascii() 40, 406
 _isjis_kana() 41, 414, 416, 420
 _issjis1() 41, 428
 _issjis2() 41, 430
 isalnum() 40, 402
 isalph() 404

isalpha() 40
isascii() 406
iscntrl() 40, 410
isdigit() 40, 412
isgraph() 41, 418
islower() 41, 422
isprint() 41, 424
ispunct() 41, 426
isspace() 41, 432
isupper() 41, 435
isxdigit() 41, 437
list of functions 40
test for
 alphabetic 40, 404
 alphanumeric 40, 402
 ASCII 40, 406
 control code 40, 410
 digit 40, 412
 Hankaku-kana 41, 414, 416, 420
 hexadecimal 41, 437
 Kanji 41, 428, 430
 lowercase 41, 422
 printable character 41, 424
 printing character 41, 418
 punctuation character 41, 426
 uppercase 41, 435
 white space 41, 432
character conversion
 _aton() 42, 156
 _atou() 42, 158
 dtoa() 42, 211
 _toascii() 42, 1303
 _tolower() 42, 1304
 _toupper() 42, 1306
 alpha to
 floating 42, 153
 integer 42, 154
 long 42, 155
 numeric 42, 156
 unsigned 42, 158
atof() 42, 153

atoi() 42, 154
atol() 42, 155
convert
 double to ASCII 42
 to lowercase 42, 1304, 1305
 to uppercase 42, 1306, 1307
double to ASCII 211
list of functions 42
toascii() 1303
tolower() 42, 1305
toupper() 42, 1307
translate 42, 1303
chdir() 109, 176
check
 buffered file for
 end of file 262
 error condition 263
for
 deadlock situation 599
 end of file 372
 use of I/O module 752
memory block's accessibility 575
terminal capability presence 1293
chmod() 109, 178
chown() 109, 180
chxdir() 50, 110, 182
clear
 end of file condition 184
 error condition 186
 process signal queue 579
Clear Interrupt Request 1052
cleareof() 184
clearerr() 51, 186
clib.l 10, 108
clock() 93, 187
 era for 108
CLOCKS_PER_SEC 140, 143, 187
close
 file 258
 named directory stream 190
 path 188, 578, 886

close() 51, 110, 188
closedir() 110, 190
colored memory
 allocate 1206
 load module into 495
 modcload() 495
 srqcmem() 1206
compare
 memory 484
 names 581
 strings 191, 1242, 1243, 1255
Compare Thread IDs 1040
compute
 absolute value 443
 floating point remainder 277
 ldiv() 447
 quotient 210, 447
 remainder 210, 447
 string length 1259
Condition Variable, Block on a 1026
Condition Variable, Release Threads Waiting for 1014
constant data
 change current pointer 174
control
 locale 1171
 process' signal handling 1185
convert
 dtoa() 211
 alpha to
 floating 153
 integer 154
 long 155
 numeric 156
 unsigned 158
 broken-down time 145
 broken-down to calendar time 493
 calendar to
 Greenwich Mean Time 367
 local time 457
 string format 206
 character to

lowercase 1304, 1305
 uppercase 1306, 1307
 date/time to
 Gregorian value 368
 Julian value 439
 double to ASCII 211
 input string 301
 local to Greenwich Mean Time 1276
 localtime() 457
 mbstowcs() 479
 mktme() 493
 sequence of multibyte characters 479
 string to
 double 1261
 long 1266, 1269
 wide to multibyte characters 1333, 1335
convert characters
 carriage return to linefeed 205
 linefeed to carriage return 450
copy
 _os_rdalst() 801
 contents of path descriptor 721
 external memory 195, 586
 memory 485
 old OS-9 strings 1251
 string 1244, 1256
 system alias list 801
cos() 65, 193
cosh() 65, 194
cosine 193
 arc 133
 hyperbolic 194
Counter, Decrement Suspendability 1106
Counter, Decrement Suspension 1093
Counter, Increment Suspendability 1108
Counter, Increment Suspension 1110
cpu.l 11
CRC
 _os_crc() 588
 _os_setcrc() 844
 calculate 197

check value 588
crc() 197
generate 588, 844
crc() 74, 110, 197
CRCCON 588
creat() 51, 110, 200
create
 data module 488, 597, 783
 device alias 551
 directory 466, 490, 778
 event 632
 file 200, 202, 593
 hard link 896
 module 472
 module directory 780
 path to new file 590
 process 685, 687, 690
 resource lock descriptor 596
 temporary binary file 1301
 unique file name 492
Create New Thread 1036
create temporary file names 1289
Create Thread-Specific Data Key 1058
create() 51, 108, 202
crtolf() 205
CST 90
cstart.r 10
ctime() 93, 206
ctype.h 16
cube root
 cbrt() 169
cursor movement
 get capability 1297
 tgoto() 1297

D

data
 _gs_rdy() 380
 _os_move() 787
 carrier detect 887, 888

move [787](#)
test for available [380](#)
Data Key, Create Thread-Specific [1058](#)
Data Key, Delete Thread-Specific [1060](#)
data module
 [_os_datmod\(\)](#) [597](#)
 [_os_mkmodule\(\)](#) [783](#)
 create [597](#), [783](#)
 specify color type [783](#)
deallocate
 bits in bit map [601](#)
 memory [293](#)
 process descriptor [607](#), [809](#)
debugged program [610](#)
 [_os_dexec\(\)](#) [610](#)
 [_os_dexit\(\)](#) [613](#)
 [_os_dfork\(\)](#) [614](#)
 [_os9_dfork\(\)](#) [616](#)
 execute [610](#)
 exit [613](#)
 fork [614](#), [616](#)
 terminate [613](#)
debugger control
 [_os_dforkm\(\)](#) [618](#)
 fork process [618](#)
Decrement Suspendability Counter [1106](#)
Decrement Suspension Counter [1093](#)
delete
 existing event [635](#)
 existing lock descriptor [605](#)
 existing module directory [606](#)
 file [603](#), [1136](#), [1314](#), [1316](#)
Delete Thread-Specific Data Key [1060](#)
Designated Thread, Orphan [1038](#)
detach device [208](#), [609](#)
Detach State Attribute, Get [983](#)
detach() [51](#), [108](#), [208](#)
Detached, Set Forked Threads [995](#)
determine
 end of file [372](#)
 file accessibility [132](#)

number of bytes in multibyte character 481
parent process ID 342
path number from file 271
process ID 341
size of unused stack area 294
string length 1253
user ID 364
device 562, 712
 _gs_devn() 370
 _os_alias() 551
 _os_close() 578
 _os_detach() 609
 _os_getstat() 712
 _os_gs_devnm() 723
 _os_gs_devtyp() 725
 _os_gs_dopt() 727
 _os_gs_dsize() 728
 _os_irq() 756
 _os_open() 789
 _os_read() 802
 _os_setstat() 851
 _os_sgs_devnm() 861
 _os_ss_dopt() 889
 _os_ss_relea() 902
 _os_write() 956
 _os9_gs_free() 735
 _os9_irq() 758
 _ss_enrts() 1220
 _ss_rel() 1227
 _ss_rest() 1228
add to IRQ table 756, 758
attach to system 159, 562
close path 578
create alias 551
detach 609
detach() 208
disable RTS line 1219
enable RTS line 1220
get name 370
get size of SCSI device 728
open 789

read data from 802
read path options 727
release 902, 1227
remove 208, 609
 from IRQ table 756, 758
restore 1228
return
 amount of free space 735
 name 723, 861
 type 725
set
 path options 889
 status 851
 write data 956
device controller
 ioctl() 393
difftime() 93, 209
dir.h 17
direct.h 17
directory
 _os_altmdir() 559
 _os_chdir() 573
 _os_chmdir() 577
 _os_cmdperm() 580
 _os_delmdir() 606
 _os_fmod() 684
 _os_get_mdp() 702
 _os_get_moddir() 703
 _os_mkdir() 778
 _os_makmdir() 780
change
 current data 176
 current execution 182
 current module directory 577
 permissions on module directory 580
 working directory 573
chdir() 176
chxdir() 182
close named stream 190
closedir() 190
create 466, 490, 778

create module directory 780
delete module directory 606
establish alternate module directory 559
find module directory entry 684
get
 alternate module directory pathlist 702
 copy of module directory 703
 current module directory pathlist 702
get current working 366
initialize 778
mkdir() 466
make 778
mknod() 490
open 535
opendir() 535
readdir() 1126
reset position of directory stream 1139
return
 current location 1288
 pointer to 1126
rewinddir() 1139
seekdir() 1147
set
 alternate working module directory 559
 position of next readdir 1147
telldir() 1288
disable
 RBF caching 885
 RTS line 890, 1219
Discontinue Attribute Block 982
div() 65, 210
double to ASCII conversion 211
driver
 _os_ss_close 886
 _os9_ss_open() 900
 notify of open path 900
 notify that path has closed 886
dtoa() 211
dup() 109, 213
dup2() 215
duplicate path 213, 215, 620

ebrk() 67, 109, 216
EDT 90
EET 90
enable
 RBF caching 885
 RTS line 891, 1220
end references to current variable parameter list 1320
endpwent() 218
enter
 I/O queue 755
 process in active process queue 561
environment
 change or add value 1115
environment functions
 _exit() 39, 253
 _os_exit() 40, 677
 abnormal program termination 39, 130
 abort() 39, 130
 atexit() 39, 152
 exit() 39, 254
 get value for environment name 40, 330
 getenv() 40, 330
 listed 39
 register function to call at normal program termination 39,
 152
 shell command execution 40, 1283
 system() 40, 1283
 task termination 39, 253, 254
 terminate calling process 40, 677
environment name
 get value for 330
environment names
 common set of 107
environment variables
 TERM 1291
 TERMCAP 1291
 TZ 90, 108
ERANGE 98
erase tape 892

errno 98
 value to which macro is set on failure 106
errno.h 17
error
 _errmsg() 219
 _os_perr() 794
 _os_strap() 915
 _os9_strap() 918
 check buffered file for error condition 263
 clear 186
 display message 219
 ferror() 263
 format of messages 1246
 map number 967
 perror() 967
 prerr() 976
 print message 219, 794, 976
 set trap handler 915, 918
error message strings
 contents of 107
error messages
 format of 106
EST 90
Euclidean distance function 383
events
 _ev_creat() 221
 _ev_delete() 223
 _ev_info() 224
 _ev_link() 226
 _ev_pulse() 227
 _ev_read() 229
 _ev_set() 231
 _ev_setr() 233
 _ev_signal() 235
 _ev_unlink() 237
 _ev_wait() 239
 _ev_waitr() 241
 _os_ev_allclr() 622
 _os_ev_allset() 624
 _os_ev_anyclr() 626
 _os_ev_anyset() 628

_os_ev_change() 630
_os_ev_creat() 632
_os_ev_delete() 635
_os_ev_info() 636
_os_ev_link() 639
_os_ev_pulse() 640
_os_ev_read() 642
_os_ev_set() 643
_os_ev_setand() 645
_os_ev_setor() 647
_os_ev_setr() 649
_os_ev_setxor() 651
_os_ev_signal() 653
_os_ev_tstset() 655
_os_ev_unlink() 657
_os_ev_wait() 658
_os_ev_waitr() 662
_os9_ev_wait() 660
_os9_ev_waitr() 664
create 221, 632
decrement link count 237
delete 223, 635
get information 224
link to existing event 226, 639
read event value 642
read without waiting 229
return event information 636
set event variable 231, 643, 645, 647, 651
 relative 233, 649
signal event occurrence 227, 231, 233, 235, 640, 643,
 645, 647, 649, 651, 653
unlink 237, 657
wait for
 bits to clear 622
 event to occur 239, 622, 624, 626, 628, 630, 655,
 658, 660
 relative event 241, 662, 664
events.h 18
examine system global variable 360, 714, 1180
example
 error file 795

for generating CRC 199
initialization table 854, 855, 915
of _errmsg() 220
of _get_process_desc() 343
of _julian() 440
of _os_exec() 671
of _os_getsys() 715
of _sysdate() 1280
of fopen() 281
of index() 388
of intercept() 392
of os9exec() 676
of os9kexec() 768
of rindex() 1141
of strtok() 1265
of vfprintf() 1324
exception table
 set base address 1164
 set_excpt_base() 1164
exec interfaces
 execl() 243
 execle() 245
 execv() 247, 249
 execvp() 251
 execl() 243
 execle() 245
 execute 610
 module 568, 571
 new module 171
 shell command 1283
 system-state subroutine 826
 at Gregorian date/time 810
 at Julian date/time 814
Execute Routine Once per Process 1091
Execute Specified Function at Exit 1012
execv() 247, 249
execvp() 251
exit debugged program 613
Exit Status, Store Thread 1054
exit() 39, 254
 status returned by 107

Exit, Execute Specified Function at 1012
exp() 65, 256
expand
 value to integral constant expression 532
exponential function 256
extend data memory segment 1142
external memory 586

F

fabs() 65, 257
fclose() 51, 258
fdopen 110
fdopen() 51, 110, 260
feof() 51, 262
ferror() 51, 263
fflush() 51, 264
ffs() 265
fgetc() 51, 266
fgetpos() 43, 106, 267
fgetpwent() 268
fgets() 51, 269
file 1113
 _fileno() 271
 _gs_pos() 378
 _gs_size() 382
 _os_close() 578
 _os_create() 590
 _os_delete() 603
 _os_getstat() 712
 _os_gs_fdaddr() 732
 _os_gs_pos() 740
 _os_gs_size() 744
 _os_open() 789
 _os_read() 802
 _os_setstat() 851
 _os_ss_attr() 883
 _os_ss_hdlink() 896
 _os_ss_rename() 903
 _os_ss_size() 908
 _os_write() 956

_os9_create() 593
_ss_attr() 1213
_ss_size() 1229
affect of opening multiple times 106
attach path to file pointer 260
change
 access permissions 178
 attributes 1213
 name 903, 1137
 owner 180
 size 1229
chmod() 178
chown() 180
close 258
close path 578
creat() 200
create 200, 202, 590, 593
 temporary binary file 1301
 unique name 492
create() 202
delete 603, 1136, 1314, 1316
determine
 accessibility 132
 path number 271
 size 382
fclose() 258
fdopen() 260
fflush() 264
fgetc() 266
fgetpos() 267
fgets() 269
fileno() 271
fix buffer 1161
flush buffer 264
fopen() 278
fputc() 289
fputs() 290
fread() 291
freopen() 298
fseek() 307
fsetpos() 309

ftell() 312
fwrite() 313
generate unique name 1302
get
 character from file 266
 current position 267, 378, 740
 current size 382
 file descriptor block address 732
 status 356, 712
 string from 269, 354
gets() 354
getstat() 356
getw() 365
load module from 773
lseek() 464
make hard link to 896
mktemp() 492
open 278, 533, 789
open() 533
output
 character to 289
 string to 290, 1117
 word to 1118
position pointer 464
print to 1323, 1325, 1327
read
 characters from 354
 data from 291, 802
 word 365
remove 1136
remove() 1136
rename 903, 1137
rename() 1137
re-open 298
report pointer position 312
reposition pointer 307
return
 current size 744
 path number 200, 202
 pointer to zero 1138
rewind() 1138

set
 attributes 883, 1213
 current position 309
 current size 1229
 size 908
 status 851, 1177
setbuf() 1161
setstat() 1177
tmpfile() 1301
tmpnam() 1302
unlink 1314, 1316
unlink() 1314
unlinkx() 1316
valid access modes 260, 278
vfprintf() 1323
vprintf() 1325
vsprintf() 1327
write
 data 313, 956
 string to 290
 unwritten data 264

file descriptor
 _gs_gfd() 374
 _os_gs_fd() 731
 _os_gs_fdaddr() 732
 _os_gs_fdin() 733
 _os_gs_parity() 737
 _os_ss_fd() 893
 _os9_gs_cdfd() 720
 _ss_pfd() 1225
 calculate parity 737
 get block address 732
 get sector 374, 733
 put sector 1225
 read sector 731
 return 720
 write sector 893

file names
 composing valid 105

file pointer
 report position 312

reposition 307, 464
logical 835
return to zero 1138
file positioning functions
 `_os_seek()` 43, 835
 `fgetpos()` 43, 267
 `fseek()` 43, 307
 `fsetpos()` 43, 309
 `ftell()` 43, 312
 get current position in file 43, 267
 list of 43
 report file pointer position 43, 312
 reposition
 file pointer 43, 307
 logical file pointer 43, 835
 return file pointer to zero 43, 1138
 `rewind()` 43, 1138
 set current file position 43, 309
`fileno()` 271
fill memory 487
find
 fixed block of memory 680
 module 786
 module directory entry 684
 path descriptor 705
 process descriptor 679
 temporal difference 209
 terminal name
 `isatty()` 408
find first set bit
 `ffs()` 265
`findnstr()` 87, 110, 272
`findstr()` 87, 110, 274
fix file buffer 1161
`float.h` 18
floating absolute value 257
floor function 276
`floor()` 65, 276
flush
 cached bit map information 895
 file buffer 264

fmod() 65, 98, 277
fopen() 52, 278
fork process under control of debugger 614, 616, 618
Forked Threads Detached, Set 995
format of
 error messages 106
 Gregorian value 368
 time and date values 368
formatted output 282, 979, 1202
fprintf() 52, 106, 282
fputc() 52, 289
fputs() 52, 290
fread() 52, 291
free() 67, 293
freemem() 294
freopen() 52, 298
frexp() 66, 300
from/to process
 create communication pipe 970
fscanf() 52, 106, 301
fseek() 43, 307
fsetpos() 43, 309
fstat() 311
ftell() 43, 106, 312
funcs.h 18
function
 defined 8
Function at Exit, Execute Specified 1012
fwrite() 52, 313

G

generate
 CRC 197, 588
 unique valid file name 1302
 valid CRC 844
get
 alternate module directory pathlist 702
 cache status information 722
 calendar time 1299
 character from

file 266
I/O address 386
copy of
 module directory 703
 process descriptor block table 710
current
 constant data pointer 328
 file position 378, 740
 file size 382
 module directory pathlist 702
 position in file 267
 static storage pointer 359
 task segment 329
cursor movement capability 1297
device
 name 370
 status 712
event information 224
exception table base address 333
external memory 216
file descriptor
 block address 732
 sector 374, 733
file status 356, 712
free memory block map 693
global descriptor pointer 334
Gregorian date 718
I/O interface edition number 729
integer from I/O address 389
Julian date 760
module directory entry 336
module name 978
next character from file/stdin 326
ownership of resource lock 536
parameter in variable parameter list 1319
path options 376
pointer to I/O process descriptor 585, 701
process descriptor copy 343, 716
process ID 745, 747
process table entry 345
processor time 187

size of
 SCSI devices 728
 stack used 1236
status 712
string
 from file 269
 length 1245
system
 date 696, 698
 global variables 360
 I/O device list head pointer 695
 time 335, 696, 698
Termcap entry 1291
terminal capability 1295
terminal capability ID 1294
user ID 364, 745, 747
value for environment name 330
value of registers 315
word from I/O address 395
Get Attribute Priority 987
Get Detach State Attribute 983
get file status
 fstat() 311
 stat() 1237
Get Stack Address 989
Get Stack Size 991
Get Value of Attribute 1064, 1077, 1079, 1081
get_const() 328
get_current_tss() 95, 329
get_excpt_base() 95, 333
get_gdtr() 95, 334
get_static() 71, 359
getc() 52, 326
getchar() 326
getenv() 40, 107, 330
getgid() 332
getime() 93, 111, 335
getnextpwnam 337
 getopt() 338
getpid() 82, 111, 341
getppid() 342

getpw() 346
getpwent() 347
getpwnam() 350
getpwuid() 352
gets() 52, 354
GetStat
 _gs_devn() 43, 370
 _gs_eof() 44, 372
 _gs_gfd() 44, 374
 _gs_opt() 44, 376
 _gs_pos() 44, 378
 _gs_rdy() 44, 380
 _gs_size() 44, 382
 _os_getstat() 44, 712
 _os_getsys() 44, 714
 _os_gs_cpypd() 44, 721
 _os_gs_cstats() 44, 722
 _os_gs_devnm() 44, 723
 _os_gs_devtyp() 44, 725
 _os_gs_dopt() 44, 727
 _os_gs_dsize() 44, 728
 _os_gs_edt() 45, 729
 _os_gs_eof() 45, 730
 _os_gs_fd() 45, 731
 _os_gs_fdaddr() 45, 732
 _os_gs_fdinf() 45, 733
 _os_gs_luopt() 45, 736
 _os_gs_parity() 45, 737
 _os_gs_popt() 45, 738
 _os_gs_pos() 45, 740
 _os_gs_ready() 45, 743
 _os_gs_size() 45, 744
 _os_sgetstat() 46, 860
 _os_sgs_devnm() 46
 _os9_gs_cdfd() 44, 720
 _os9_gs_free() 45, 735
calculate parity of file descriptor 45, 737
check for end of file 44, 372
copy contents of path descriptor 44, 721
examine system global variable 44, 714
get

cache status information 44, 722
current file position 44, 45, 378, 740
current file size 44, 382
device name 43, 370
device status 44, 712
file
 descriptor block address for open file 45, 732
 descriptor sector 44, 45, 374
 status 44, 356, 712
I/O interface edition number 45, 729
path options 44, 376
size of SCSI device 44, 728
specified file descriptor sector 733
GetStat call using system path number 46, 860
getstat() 356
list of functions 43
read
 device path options 44, 727
 file descriptor sector 45, 731
 logical unit options 45, 736
 path descriptor options section 45, 738
return
 amount of free space on device 45, 735
 current file size 45, 744
 device name 44, 46, 723
 device type 44, 725
 file descriptor 44, 720
test
 for data available 44, 380
 for data ready 45, 743
 for end of file 45, 730
getstat() 111, 356
gettmeofday() 362
getuid() 71, 111, 364
getw() 52, 111, 365
getwd() 366
global descriptor
 make entry in table 468
 make_gdesc() 468
 set pointer 1165
 set_gdtr() 1165

globals 12
GMT 90
gmtime() 93, 367
Greenwich Mean Time 367, 1276
Gregorian
 _gregorian() 368
 _os_gregorian() 718
 convert to 368
group identification
 getgid() 332

H

hard link
 _os_ss_hdlink() 896
create 896
header files 15
 alarm.h 15
 assert.h 15
 aton.h 16
 cache.h 16
 cdi.h 16
 ctype.h 16
 dir.h 17
 direct.h 17
 errno.h 17
 events.h 18
 float.h 18
 funcs.h 18
 io.h 19
 ioctl.h 19
 ioedt.h 19
 limits.h 19
 locale.h 19
 lock.h 20
 math.h 20
 memory.h 21
 moddir.h 21
 modes.h 22
 module.h 23, 588
 os9def.h 24

os9time.h 24
process.h 25
procid.h 26, 28
rbf.h 29
regexp.h 29
regs.h 30
sbf.h 30
scf.h 31
semaphore.h 31
setjmp.h 31
setsys.h 31
settrap.h 32
sg_codes.h 32
sgstat.h 32
signal.h 33
stdarg.h 33
stddef.h 33
stdio.h 34
stdlib.h 35
string.h 36
strings.h 36
svctbl.h 36
sysglob.h 37
termcap.h 37
termio.h 37
time.h 38
un.h 38
hyperbolic
cosine 194
sine 1198
tangent 1287
hypot() 66, 112, 383

I/O address
get
character from 386
integer from 389
word from 395
inc() 386

inl() 389
inw() 395
outc() 961
outl() 962
outw() 963
write
 character to 961
 integer to 962
 word to 963
I/O device
 _os_getdl() 695
 attach 562
 get system pointer for list 695
I/O device functions
 get from I/O address
 character 49, 386
 integer 49, 389
 word 49, 395
 inc() 49, 386
 inl() 49, 389
 inw() 49, 395
 list of 49
 outc() 50, 961
 outl() 50, 962
 outw() 50, 963
 write to I/O address
 character 50, 961
 integer 50, 962
 word 50, 963
I/O functions
 _cleareof() 51, 184
 _fileno() 52, 271
 _os_alias() 53, 551
 _os_attach() 53, 562
 _os_chdir() 53, 573
 _os_close() 53, 578
 _os_cpy_ioproc() 53, 585
 _os_create() 53, 590
 _os_delete() 53, 603
 _os_detach() 53, 609
 _os_dup() 53, 620

_os_get_ioproc() 53, 701
_os_getdl() 53, 695
_os_iodel() 53, 54, 752
_os_ioexit() 54, 753
_os_ifork() 54, 754
_os_mkdir() 54, 778
_os_open() 54, 789
_os_rdalst() 54, 801
_os_read() 54, 802
_os_readln() 54, 804
_os_write() 54, 956
_os writeln() 54, 958
_os9_create() 53, 593
_os9_iqueue() 54, 755
attach
 new device to system 53
 path to file pointer 51, 260
 to device 50, 159, 562
attach() 50, 159
change
 current execution directory 50, 182
 working directory 53, 573
check
 buffered file for
 end of file 51, 262
 error condition 51, 263
 for use of I/O module 53, 54, 752
chkdir() 50, 182
clear
 end of file condition 51, 184
 error condition 51, 186
cleareof() 184
clearerr() 51, 186
close
 file 51, 258
 path 51, 53, 188, 578
close() 51, 188
copy system alias list 54, 801
creat() 51, 200
create
 device alias 53, 551

directory 52, 466, 490
file 51, 200, 202
path to new file 53, 590, 593
temporary binary file 55, 1301
unique file name 52, 492
create() 51, 202
delete file 53, 603
detach device 51, 208
detach() 51, 208
determine path number from file 52, 271
duplicate path 53, 620
enter I/O queue 54, 755
fclose() 51, 258
fdopen() 51, 260
feof() 51, 262
ferror() 51, 263
fflush() 51, 264
fgetc() 51, 266
fgets() 51, 269
fileno() 271
fix file buffer 55, 1161
flush file's buffer 51, 264
fopen() 52, 278
formatted output 52, 54, 55, 282, 979, 1202
fprintf() 52, 282
fputc() 52, 289
fputs() 52, 290
fread() 52, 291
freopen() 52, 298
fscanf() 52, 301
fwrite() 52, 313
get
 character 266, 326
 character from
 file 51
 pointer to I/O process descriptor 53, 585, 701
 string from file 51, 52, 269, 354
 system I/O device list head pointer 53, 695
get character from
 file 52
get pointer to I/O process descriptor 53

getc() 52, 326
getchar() 326
gets() 52, 354
getw() 52, 365
initialize for
 float output 54
 longs output 54
input string conversion 52, 55, 301, 1144, 1215
input strings conversion 55
list of 50
mkdir() 52, 466
make new directory 54, 778
mknod() 52, 490
mktemp() 52, 492
open
 directory 53, 535
 file 52, 53, 278, 533
 path to file or device 54, 789
open() 53, 533
opendir() 53, 535
output to file
 character 52, 289
 string 52, 55, 290, 1117
 word 55, 1118
pffinit() 54, 968
pflinit() 54, 969
print to
 file 56, 1323, 1325, 1327
 standard output 56
 string 56
printf() 54, 979
put next character 55, 1113
putc() 55, 1113
putchar() 1113
puts() 55, 1117
putw() 55, 1118
read
 bytes from path 55, 1124, 1127
 data from device 54, 802
 data from file 52, 54, 291, 802
 text line with editing 54, 804

word from file 52, 365
read() 55, 1124
readln() 55, 1127
remove
 device from system 53, 609
 file 55, 1136
remove() 55, 1136
rename file 55
rename() 55
re-open file 52, 298
scanf() 55, 1144
set up
 buffer for I/O stream 55, 1183
 I/O for new process 54, 754
setbuf() 55, 1161
setvbuf() 55, 1183
sprintf() 55, 1202
sscanf() 55, 1215
terminate I/O for exiting process 54, 753
tmpfile() 55, 1301
unget character 56, 1312
ungetc() 56, 1312
unlink file 56, 1314, 1316
unlink() 56, 1314
unlinkx() 56, 1316
vfprintf() 56, 1323
vprintf() 56, 1325
vsprintf() 56, 1327
write
 bytes to path 56, 1337, 1339
 data to device 54, 956
 data to file 52, 54, 313, 956
 line of text with editing 54, 958
write() 56, 1337
writeln() 56, 1339
I/O interface edition number
 _os_gs_edt() 729
 get 729
I/O module
 _os_iodel() 752
check for use of 752

I/O process descriptor
 `_os_cpy_ioproc()` 585
 `_os_get_ioproc()` 701
 get pointer to 585, 701

I/O queue
 `_os9_iqueue()` 755
 enter 755

I/O stream
 set buffer for 1183
 `setvbuf()` 1183

`ibrk()` 67, 112, 384

IDs, Compare Thread 1040

`inc()` 49, 386

Increment Suspendability Counter 1108

Increment Suspension Counter 1110

`index()` 87, 113, 387

indicate event occurrence 227, 231, 233, 235, 647

initialize
 directory 778
 for float output 968
 for longs output 969
 process descriptor 557, 558
 resource lock descriptor 596
 service request table 855
 static storage 748
 variable parameter list 1321

`inl()` 49, 389

input
 failures
 defined 304
 read from stream 301
 string conversion 301, 1144, 1215

insert process in active process queue 560

install
 system state trap handling module 932
 user subroutine library module 876
 user subroutine module 874
 user trap handling module 934

integer
 return random 1123
 write to I/O address 962

integerget from I/O address 389
intercept trap
 _os_intercept() 749
 set up 749
intercept() 58, 113, 390
interprocess communication
 _ev_create() 57, 221
 _ev_delete() 57, 223
 _ev_info() 57, 224
 _ev_link() 58, 226
 _ev_pulse() 58, 227
 _ev_read() 58, 229
 _ev_set() 58, 231
 _ev_setr() 58, 233
 _ev_signal() 58, 235
 _ev_unlink() 58, 237
 _ev_wait() 58, 239
 _ev_waitr() 58, 241
 _os_alarm_atime() 59, 540
 _os_alarm_cycle() 59, 544
 _os_alarm_delete() 59, 546
 _os_alarm_reset() 59, 547
 _os_alarm_set() 59, 549
 _os_clrsigs() 59, 579
 _os_ev_allclr() 59, 622
 _os_ev_allset() 59, 624
 _os_ev_anyclr() 59, 626
 _os_ev_anyset() 59, 628
 _os_ev_change() 59, 630
 _os_ev_create() 59, 632
 _os_ev_delete() 60, 635
 _os_ev_info() 60, 636
 _os_ev_link() 60, 639
 _os_ev_pulse() 60, 640
 _os_ev_read() 60, 642
 _os_ev_set() 60, 643
 _os_ev_setand() 60, 645
 _os_ev_setor() 60, 647
 _os_ev_setr() 60, 649
 _os_ev_setxor() 60, 651
 _os_ev_signal() 60, 653

_os_ev_tstset() 61, 655
_os_ev_unlink() 61, 657
_os_ev_wait() 61, 658
_os_ev_waitr() 61, 662
_os_intercept() 61, 749
_os_salarm_atime() 61, 812
_os_salarm_cycle() 61, 816
_os_salarm_delete() 62, 820, 821
_os_salarm_reset() 62, 822
_os_salarm_set() 62, 824
_os_sema_init() 62, 836
_os_sema_p() 62, 838
_os_sema_term() 62
_os_sema_v() 62, 841
_os_send 842
_os_send() 62
_os_sigmask() 63, 865
_os_sigrs() 63, 869
_os9_alarm_atdate() 58, 538
_os9_alarm_atjul() 58, 542
_os9_ev_wait() 61, 660
_os9_ev_waitr() 61, 664
_os9_salarm_atdate() 61, 810
_os9_salarm_atjul() 61, 814
_os9_salarm_cycle() 62, 818
_os9_salarm_set() 62, 826
alarm() 134
alm_atdate() 57, 136
alm_atjul() 57, 138
alm_cycle() 57, 140
alm_delete() 57, 142
alm_set() 57, 143
clear process signal queue 59, 579
control process' signal handling 63, 1185
create event 57, 59, 221, 632
delete event 57, 60, 223, 635
execute system-state subroutine
 after specified time interval 62, 826
 at Gregorian date/time 61, 810
 at Julian date/time 61, 814
 at specified time interval 61

every n ticks/second 62, 818
get event information 57, 224
initialize semaphore data structure 62
intercept() 58, 390
link to existing event 58, 60, 226, 639
list of functions 57
mask signals during critical code 63, 865
pause() 63, 965
raise() 63, 1122
read event without waiting 58, 60, 229, 642
release semaphore 62
remove pending alarm request 57, 59, 62, 142, 546, 820, 821
reserve semaphore 62
reset alarm
 clock 62, 822
 request 59, 547
resize current process' signal queue block 63, 869
return event information 60, 636
send signal
 after specified time interval 57, 59, 143, 549
 at Gregorian date/time 57, 58, 136, 538
 at Julian date/time 57, 58, 138, 542
 at seconds 134
 at specified time 59, 540, 812
 at specified time intervals 57, 59, 140, 544
 to another process 62, 842
 to program 63, 1122
set
 alarm clock 62, 824
 cyclic alarm clock 61, 816
 event variable 58, 60, 231, 643, 645, 647, 651
 process signal handler 58, 390, 749
 relative event variable 58, 60, 233, 649
 signal intercept trap 61
sigmask() 63, 1185
signal event occurrence 58, 60, 227, 231, 233, 235, 640, 643, 645, 647, 649, 651, 653
signal() 63, 1188
specify signal handling 63, 1188
terminate use of semaphore data structure 62

unlink event 58, 61, 237, 657
unmask signals 63, 865
wait for
 all bits defined by mask to become clear 59, 622
 event to occur 58, 59, 61, 239, 624, 626, 628, 630, 655, 658, 660
 relative event to occur 58, 61, 241, 662, 664
 signal 63, 965
interrupt descriptor table
 make entry 470
 make_idesc() 470
interrupt exception
 _os_rte() 808
 return from 808
interrupt level
 irq_change() 396
 irq_disable() 397
 irq_enable() 398
 irq_maskget() 399
 irq_restore() 400
 irq_save() 401
 mask 397, 399
 restore 400
 return current 401
 return to previous 399
 set 396
 unmask 398
interrupt manipulation
 _os_firq() 64
 _os_irq() 64, 756
 _os_rte() 64, 808
 _os_sigreset() 63
 _os9_irq() 64, 758
 add device to fast IRQ table 64
 add device to IRQ table 64, 756, 758
 irq_change() 64, 396
 irq_disable() 64, 397
 irq_enable() 64, 398
 irq_maskget() 64, 399
 irq_restore() 64, 400
 irq_save() 64, 401

list of functions 64
mask interrupts 64, 397, 399
remove device from fast IRQ table 64
remove device from IRQ table 64, 756, 758
reset signal intercept context stack 63
restore interrupt level 64, 400
return
 current interrupt level 64, 401
 from interrupt exception 64, 808
set IRQ level 64, 396
unmask interrupts 64, 398
Interrupt Request, Clear 1052
Interrupt Thread Process 1050
Interval, Wait for Specified 1022
inw() 49, 395
io.h 19
ioctl() 393
ioctl.h 19
ioedt.h 19
IRQ
 table
 _os_firq() 682
 add device
 fast 682
 remove device
 fast 682
IRQ level
 irq_change() 396
 mask interrupts 397, 399
 restore interrupt level 400
 return current interrupt level 401
 set 396
 unmask interrupts 398
IRQ table
 _os_irq() 756, 758
 add device 756, 758
 remove table 756, 758
irq_change() 64, 396
irq_disable() 64, 397
irq_enable() 64, 398
irq_maskget() 64, 399

irq_restore() 64, 400
irq_save() 64, 401
isalnum() 40, 97, 402
isalpha() 40, 404
isascii() 406
isatty() 408
iscntrl() 40, 97, 410
isdigit() 40, 412
isgraph() 41, 418
islower() 41, 97, 422
isprint() 41, 97, 424
ispunct() 41, 426
isspace() 41, 432
isupper() 41, 435
isxdigit() 41, 437

J

Japanese Industrial Standard 428, 430
JST 90
Julian date and time 439
Julian day defined 138

K

Key Value, Return Specified 1043
Key, Associate a Thread Value with a 1104
Key, Create Thread-Specific Data 1058
Key, Delete Thread-Specific Data 1060
kill() 82, 113, 441

L

labs() 66, 443
LC_ALL 456, 1171
LC_COLLATE 1171, 1243
LC_CTYPE 1171
LC_MONETARY 456, 1171
LC_NUMERIC 456, 1171
LC_TIME 1171, 1172, 1249

ldexp() 66, 446
ldiv() 66, 447
lftocr() 450
limits.h 19
link 769
 to existing event 226, 639
 to memory module 498, 500, 502, 769, 771, 776
load
 memory module 171, 500, 502, 773, 776
 module into colored memory 495
locale control 1171
locale.h 19
localeconv() 71, 453
localtime() 94, 457
locate
 first occurrence of string 1257, 1260
 last occurrence of string 1258
 string 1241
lock
 _os_acqlk() 536
 _os_caqlk() 567
 _os_crlk() 596
 _os_ddlk() 599
 _os_dellk() 605
 _os_rellk() 806
 _os_waitlk() 954
 acquire 536
 activate next 954
 conditionally acquire ownership of 567
 create 596
 deadlock situation 599
 delete descriptor 605
 get ownership 536
 out record 898, 1221
 release ownership 806
Lock Mutex Object 1068
Lock, Set Non-Blocking 1072
lock.h 20
log() 66, 458
log10() 66, 459
logarithm

base-ten 459
natural 458
logical file pointer
 `_os_seek()` 835
 reposition 835
logical unit
 `_os_gs_luopt()` 736
 `_os_ss_luopt()` 899
 read options 736
 write options 899
`longjmp()` 71, 460
`Iseek()` 113, 464

M

`m_tmzone` 93, 108
macro
 bypass replacement 8
 defined 8
`mkdir()` 52, 111, 466
make
 directory 778
 global descriptor table entry 468
 interrupt descriptor table entry 470
`make_gdesc()` 95, 468
`make_idesc()` 95, 470
`make_module()` 74, 111, 472
`malloc()` 68, 107, 475
map
 error number 967
mask
 interrupts 397, 399
 signals 865
matching failures
 defined 304
`math.h` 20
mathematical functions
 `abs()` 65, 131
 `acos()` 65, 133
 arc
 cosine 65, 133

sine 65, 147
tangent 65, 150, 151
asin() 65, 147
atan() 65, 150
atan2() 65, 151
base-ten logarithm 66, 459
ceil() 65, 170
ceiling function 65, 170
compute
 absolute value 66, 443
 floating point remainder 65, 277
 quotient and remainder 65, 66, 210, 447
cos() 65, 193
cosh() 65, 194
cosine 65, 193
 arc 65, 133
 hyperbolic 65, 194
div() 65, 210
Euclidean distance function 66, 383
exp() 65, 256
exponential function 65, 256
fabs() 65, 257
floating absolute value 65, 257
floor function 65, 276
floor() 65, 276
fmod() 65, 277
frexp() 66, 300
hyperbolic
 cosine 65, 194
 sine 66, 1198
 tangent 66, 1287
hypot() 66, 383
integer absolute value 65, 131
labs() 66, 443
ldexp() 66, 446
ldiv() 66, 447
list of 65
log() 66, 458
log10() 66, 459
modf() 66, 497
multiply float by exponent of 2 66, 446

natural logarithm 66, 458
pow() 66, 974
power function 66, 974
rand() 66, 1123
return
 parts of real number 66, 497
 portions of floating point number 66, 300
 random integer 66, 1123
set seed for random number generator 66, 1205
sin() 66, 1197
sine 66, 1197
 arc 65, 147
 hyperbolic 66, 1198
sinh() 66, 1198
sqrt() 66, 1204
square root 66, 1204
srand() 66, 1205
tan() 66, 1286
tangent 66, 1286
 arc 150, 151
 hyperbolic 66, 1287
tanh() 66, 1287
mblen() 78, 477
mbstowcs() 479
mbstowsc() 78
mbtowc() 78, 481
MDT 90
MEM_ANY 473, 496, 1207
MEM_SYS 473, 496, 1206
Member, Set Priority 1000
memchr() 68, 483
memcmp() 68, 484
memcpy() 68, 485
memmove() 68, 486
memory
 _mkdata_module() 488
 _os_get_blkmap() 693
 _os_link() 769
 _os_linkm() 771
 _os_loadp() 776
 _os9_allpd() 555

_os9_findpd() 680
_os9_retpd() 807
_setcrc() 1162
allocate fixed-length block 555
allocate from arena 451, 475
allow access to 792
arena 451
change size of block 1131
check accessibility 575
compare 484
copy 485, 486
copy external 586
create data memory module 488
create module 472
fill 487
find fixed block 680
get free block map 693
link to module 498, 500, 502, 769, 771, 776
load module 500, 502, 773, 776
 into colored memory 495
make_module() 472
modcload() 495
modlink() 498
modload() 500
modloadap() 502
prevent access to block 796
request internal 384
resize block 1131
resize block of 462
resize data area 781
return 293, 448
return fixed block 807
return pointer to memory block 384
re-validate module CRC 1162
search 483
set minimum allocation size 476
set reclamation bound 296
translate address 937, 939
update
 CRC 1162
 header parity 1162

memory area
 clear
 bzero() 166
 compare
 bcmpl() 161
 copy
 bcopy() 162
memory arena 475
memory management
 _cpymem() 67, 195
 _freemem() 67, 294
 _lalloc() 67, 444
 _lfree() 67, 448
 _lmalloc() 67, 451
 _lrealloc() 67, 462
 _mallocmin() 68, 476
 _os_alltsk() 68, 557
 _os_chkmem() 575
 _os_cpymem() 68, 586
 _os_deltsk() 68, 607
 _os_mem() 68, 781
 _os_permit() 69, 792
 _os_protect() 69, 796
 _os_srqmem() 69, 878
 _os_srtmem() 69, 882
 _os9_srqmem() 69, 880
 _os9_translate() 69, 939
 _srqmem() 69, 1209
 _srtmem() 69, 1211
allocate
 colored memory 69, 1206
 from arena 67, 68, 451, 475
 storage for array 67, 167, 444
 task 68, 557
allow access to 69, 792
calloc() 67, 167
change size of block 1131
check accessibility 575
compare 68, 484
contract data area 781
copy 68, 485

external memory 67, 68, 195, 586
deallocate 293
 process descriptor 68, 607
determine size of unused stack area 67, 294
ebrk() 67, 216
expand data area 781
extend data segment 69, 1142
fill 68, 487
free() 67, 293
freemem() 294
get external memory 67, 216
ibrk() 67, 384
list of functions 67
low-overhead 444, 448, 451, 462
malloc() 68, 475
memchr() 68, 483
memcmp() 68, 484
memcpy() 68, 485
memmove() 68, 486
memset() 68, 487
move 68, 486
prevent access to 69, 796
realloc() 69, 1131
request 880
request internal memory 67, 384
resize
 block 67, 69, 462, 1131
 data area 68, 781
return 293, 882
 memory 67, 448
return system 69
sbrk() 69, 1142
search 68, 483
set minimum allocation size 68, 476
srqcmem() 69, 1206
system
 request 69, 878, 1209
 return 69, 1211
 translate address 69, 939
memory map
_os9_gspump() 742

return data for 742
memory.h 21
memset() 68, 487
miscellaneous functions
 list of 71
mknod() 52, 111, 490
mktemp() 52, 111, 492
mktime() 94, 493
modcload() 74, 111, 495
moddir.h 21
modes.h 22
modf() 66, 497
modlink() 74, 111, 498
modload() 75, 111, 500
modloadp() 75, 111, 502
module 703, 769
 _os_altmdir() 559
 _os_chain() 568
 _os_chainm() 571
 _os_chmdir() 577
 _os_fmod() 684
 _os_iodel() 752
 _os_link() 769
 _os_linkm() 771
 _os_load() 773
 _os_loadp() 776
 _os_makmdir() 780
 _os_modaddr() 786
 _os_slink() 874
 _os_slinkm() 876
 _os_tlink() 932
 _os_tlinkm() 934
 _os_unlink() 943
 _os_unload() 944
 _os_vmodul() 945
 _os9_vmodul() 947
_prgname() 978
_setcrc() 1162
_subcall() 1274
calculate CRC 197
call subroutine module 1274

chain() 171
chainc() 171
change module directory 577
check
 CRC 945, 947
 for use of I/O module 752
 parity 945, 947
crc() 197
create 472
 module directory 780
establish alternate working directory 559
execute 171, 568, 571
find 786
 directory entry 684
get
 copy of module directory 703
 name 978
initialize static storage 748
install
 system state trap handler 932
 user subroutine library module 876
 user subroutine module 874
 user trap handler 934
link to 771, 776
load 171, 776
 from file 773
locate 786
make_module() 472
munlink() 528
munload() 530
re-validate CRC 1162
set alternate working directory 559
unlink
 by address 943
 by name 944
 from 528
unload 530
update
 CRC 1162
 header parity 1162
verify 945, 947

module directory
 _get_module_dir() 336
 _os_cmdperm() 580
 _os_delmdir() 606
 _os_fmod() 684
 _os_get_mdp() 702
 _os_get_moddir() 703
 _os_makmdir() 780
change permissions 580
change process' current 577
create 780
delete 606
find entry 684
get
 alternate pathlist 702
 copy of 703
 current pathlist 702
 entry 336

module manipulation
 _get_module_dir() 74, 336
 _mkdata_module() 74, 488
 _os_altdir() 75, 559
 _os_chainm() 75, 571
 _os_chmdir() 75, 577
 _os_cmdperm() 75, 580
 _os_crc() 75, 588
 _os_datmod() 75, 597
 _os_delmdir() 75, 606
 _os_fmod() 75, 684
 _os_get_mdp() 76, 702
 _os_get_moddir() 76, 703
 _os_initdata() 76, 748
 _os_link() 76, 769
 _os_linkm() 76, 771
 _os_load() 76, 773
 _os_loadp() 76, 776
 _os_makmdir() 76, 780
 _os_modaddr() 76, 786
 _os_setcrc() 76, 844
 _os_slink() 76, 874
 _os_slinkm() 77, 876

_os_tlink() 77, 932
_os_tlinkm() 77, 934
_os_unlink() 77, 943
_os_unload() 77, 944
_os_vmodul() 77, 945
_os9_vmodul() 77, 947
_setcrc() 77, 1162
_subcall() 77, 1274
calculate module CRC 74, 197
call subroutine module 77, 1274
change
 permissions of module directory 75, 580
 process' current module directory 75, 577
crc() 74, 197
create
 data memory module 74, 488
 data module 75, 597
 module 74, 472
 new module directory 76, 780
delete existing module directory 75, 606
execute new primary module 75, 571
find
 module directory entry 75, 684
 module given pointer 76, 786
generate CRC 75, 76, 588, 844
get
 copy of module directory 76, 703
 current and alternate module directory pathlists 76, 702
 module directory entry 74, 336
initialize static storage from module 76, 748
install
 system state trap handling module 77, 932
 user subroutine library module 77, 876
 user subroutine module 874
 user trap handling module 77, 934
link to memory module 74, 76, 498, 500, 502, 769, 771, 776
load
 memory module 75, 76, 500, 502, 776
 module into colored memory 74, 495
 module(s) from file 76, 773

make_module() 74, 472
modcload() 74, 495
modlink() 74, 498
modload() 75, 500
modloadap() 75, 502
munlink() 75, 528
munload() 75, 530
re-validate module CRC 77, 1162
set alternate working module directory 75, 559
unlink
 from module 75, 528
 module 77, 943, 944
unload module 75, 530
validate module 77, 945, 947
module.h 23, 588
move
 data 787
 memory 486
mq.l 11
MST 90
multibyte character
 convert sequence 78, 479
 convert to from wide 78, 1335
 convert to from wide characters 78, 1333
 determine number of bytes in 78, 477, 481
 list of functions 78
 mblen() 78, 477
 mbstowcs() 78, 479
 mbtowc() 481
 wcstombs() 78, 1333
 wctomb() 78, 1335
multiply float by exponent of 2 446
munlink() 75, 112, 528
munload() 75, 112, 530
Mutex Object, Lock 1068

N

natural logarithm 458
New Thread, Create 1036
Non-Blocking Lock, Set 1072

non-local goto 460, 1169
notify driver that path has been opened 900
numeric formatting convention inquire 453

O

Object, Lock Mutex 1068
offsetof() 532
Once per Process, Execute Routine 1091
open
 directory 535
 file 278, 533
 path 789
 re-open file 298
open() 53, 112, 533
opendir() 53, 112, 535
Orphan Designated Thread 1038
os_lib.l 11, 108
os_ss_fillbuff() 894
os_ss_rfm() 906
OS-9 Priority for Calling Thread, Set 1100
os9def.h 24
os9exec() 84, 112, 672–676
os9fork() 84, 112, 687–690
os9forkc() 687–690
os9kexec() 84, 112, 762–768
os9time.h 24
outc() 50, 961
outl() 50, 962
output
 capability string 1308
 character to file 289, 1113
 character to standard out 1113
 string to file 290, 1117
 word to file 1118
outw() 50, 963

P

parameter

_isascii() 406
_isjis_kana() 414, 416, 420
_issjis1() 428
_issjis2() 430
is it alphabetic 404
is it alphanumeric 402
is it ASCII 406
is it control code 410
is it digit 412
is it Hankaku-kana 414, 416, 420
is it hexadecimal 437
is it Kanji 428, 430
is it lower case 422
is it printable 424
is it printing character 418
is it punctuation 426
is it upper case 435
is it white space 432
isalpha() 404
isascii() 406
iscntrl() 410
isdigit() 412
isgraph() 418
islower() 422
isprint() 424
ispunct() 426
isspace() 432
isupper() 435
isxdigit() 437
parent process ID
 determine 342
parse
 disk file pathlist 964
 path name 798
 segment 981
password file
 close
 endpwent() 218
 get entry 347
 get file entry
 fgetpwent() 268

get name 346, 350
get next entry for name
 getnextpwnam() 337
get user identification 352
reset for searching 1176
path
 _gs_opt() 376
 _os_ss_fillbuff() 894
 _os_dup() 620
 _os_prsnam() 798
 _os_ss_close() 886
 _os9_ss_open() 900
 _prsnam() 981
 _ss_opt() 1223
close 188, 578
copy options 376
create to new file 590, 593
dup() 213
dup2() 215
duplicate 213, 215, 620
fill buffer with data 894
get options 376
notify driver 886
 of open path 900
open 789
parse name 798
 segment 981
read bytes from 1124, 1127
read() 1124
readln() 1127
set options 1223
valid access modes 260
write bytes to 1337, 1339
write() 1337
writeln() 1339
path descriptor
 _os_getpd() 705
 _os_gs_cpypd() 721
 _os_gs_popt() 738
 _os_ss_popt() 901
copy contents of 721

find 705
read option section 738
write option section 901
pattern
 search string for 272, 274
pause() 63, 112, 965
PDT 90
Pending Cancel, Test for 1112
per Process, Execute Routine Once 1091
 perror() 73, 106, 967
pffinit() 54, 113, 968
pflinit() 54, 113, 969
pipe() 970
place
 diagnostics in programs 148
 formatted time in buffer 1248, 1249
position file pointer 464
pow() 66, 974
power function 974
PowerPC-specific functions
 list of 96
prerr() 73, 113, 976
print
 error message 219, 794, 976
 to file 1323, 1325, 1327
printf() 54, 979
priority
 set for process 1175
 setpr() 1175
Priority for Calling Thread, Set OS-9 1100
Priority Member, Set 1000
Priority, Get Attribute 987
process
 _os_aproc() 560
 _os_clrsigs() 579
 _os_exit() 677
 _os_fork() 685
 _os_forkm() 690
 _os_ioexit() 753
 _os_iofork() 754
 _os_nproc() 788

_os_setpr() 850
_os_sigrs() 869
_os_sleep() 870
_os_suspend() 921
_os_wait() 949
_os9_aproc() 561
_os9_gspump() 742
_os9_sleep() 872
clear signal queue 579
create 685, 687, 690
end 677
insert in active process queue 560, 561
intercept() 390
kill() 441
os9fork() 687
os9forkc() 687
put to sleep 870, 872
resize current signal queue block 869
return data for memory map 742
send signal to 441
set priority 850, 1175
set up
 I/O 754
 signal handler 390
setpr() 1175
sleep 1310
start 788
suspend 921, 1329
terminate 677
 I/O 753
wait for child to terminate 949, 1329
wait() 1329
process descriptor
 _get_process_desc() 343
 _os_alocproc() 558
 _os_deltsk() 607
 _os_findpd() 679
 _os_get_ioproc() 701
 _os_get_prtbl() 710
 _os_gprdsc() 716
 _os_rtnprc() 809

_os_setpr() 84
allocate 558
deallocate 607, 809
find 679
get copy 343, 716
 of block table 710
initialize 558
set process priority 84
process ID
 _os_id() 745
 _os9_id() 747
 determine 341
 get 745, 747
process manipulation
 _get_process_desc() 343
 _get_process_table() 82, 345
 _os_allocproc() 82, 558
 _os_aproc() 82, 560
 _os_chain() 83, 568
 _os_dfork() 83, 614
 _os_dforkm() 83, 618
 _os_exec() 83, 666
 _os_findpd() 83, 679
 _os_fork() 83, 685
 _os_forkm() 83, 690
 _os_get_prtbl() 83, 710
 _os_getstat() 83, 712
 _os_getsys() 83, 714
 _os_gprdsc() 83, 716
 _os_id() 84, 745
 _os_nproc() 84, 788
 _os_rtnprc() 84, 809
 _os_setpr() 850
 _os_sleep() 84, 870
 _os_suspend() 84, 921
 _os_wait() 84, 949
 _os9_allpd() 82, 555
 _os9_aproc() 82, 561
 _os9_dfork() 83, 616
 _os9_gspump() 83, 742
 _os9_id() 84, 747

_os9_sleep() 84, 872
allocate
 fixed-length block of memory 82, 555
 process descriptor 82, 558
chain() 171
chainc() 82, 171
create process 83, 84, 685, 687, 690
deallocate process descriptor 84, 809
determine parent process ID number 342
determine process ID number 82, 341
enter process in active process queue 82, 561
examine system global variable 83, 714
execute new module 83, 568
find process descriptor 83, 679
fork process under control of debugger 83, 614, 616, 618
get
 copy of process descriptor block 83
 copy of process descriptor block table 710
 file/device status 83, 712
 process descriptor copy 83, 343, 716
 process ID/user ID 84, 745, 747
 process table entry 82, 345
getpid() 82, 341
getppid() 342
insert process in active process queue 82, 560
kill() 82, 441
list of functions 82
load and execute new module 171
OS-9 system call processing 84, 672
OS-9000 system call processing 84, 762
os9exec() 84, 672
os9fork() 84, 687
os9forkc() 687
os9kexec() 84, 762
put calling process to sleep 84, 870, 872
return data for specified process' memory map 83, 742
send signal to process 82, 441
set process priority 84, 850, 1175
setpr() 84, 1175
sleep for specified interval 85, 1310
sleep() 85

start child process 83
start next process 84, 788
suspend
 execution for specified time 85
 process 84, 921
system call processing 666
tsleep() 85, 1310
wait for process termination 84, 85, 949, 1329
wait() 85, 1329
process table
 _get_process_table() 345
 get entry 345
process times
 get 1300
Process, Execute Routine Once per 1091
Process, Interrupt Thread 1050
process.h 25
Process-Shared Attribute, Set 1034, 1070, 1085, 1087, 1089
procid.h 26, 28
program
 _exit() 253
 abnormal termination 130
 diagnostics 148
 end 253, 254
 exit() 254
 termination 152
program globals 12
 class 1 12
 class 2 13
protection hardware
 initialize 557
PST 90
pthread_attr_destroy() 982
pthread_attr_getdetachstate() 983
pthread_attr_getstackaddr() 989
pthread_attr_getstacksize() 991
pthread_attr_init() 993
pthread_attr_setdetachstate() 995
pthread_attr_setstackaddr() 1002
pthread_attr_setstacksize() 1005
pthread_cancel() 1008

pthread_cleanup_pop() 1010
pthread_cleanup_push() 1012
pthread_cond_broadcast() 1014
pthread_cond_destroy() 1016
pthread_cond_init() 1018
pthread_cond_signal() 1020
pthread_cond_timedwait() 1022
pthread_cond_wait() 1026
pthread_condattr_destroy() 1029
pthread_condattr_getpshared() 1030
pthread_condattr_init() 1032
pthread_condattr_setpshared() 1034
pthread_create() 1036
pthread_detach() 1038
pthread_equal() 1040
pthread_exit() 1041
pthread_getspecific() 1043
pthread_join() 1054
pthread_key_create() 1058
pthread_key_delete() 1060
pthread_mutex_destroy() 1062
pthread_mutex_init() 1066
pthread_mutex_lock() 1068
pthread_mutex_trylock() 1072
pthread_mutex_unlock() 1074
pthread_mutexattr_destroy() 1076
pthread_mutexattr_getpshared() 1064, 1077, 1079, 1081
pthread_mutexattr_init() 1083
pthread_mutexattr_setpshared() 1070, 1085, 1089
pthread_once() 1091
pthread_self() 1095
pthread_setcancelstate() 1096
pthread_setcanceltype() 1098
pthread_setspecific() 1104
pthread_testcancel() 1112
put
 calling process to sleep 870, 872
 file descriptor sector 1225
 next character to file/standard out 1113
putc() 55, 1113
putchar() 1113

putenv() 1115
puts() 55, 1117
putw() 55, 113, 1118

Q

qsort() 86, 1120
quick sort 1120
quotient
 ldiv() 447
 return 210, 447

R

raise() 63, 1122
rand() 66, 1123
random number
 rand() 1123
 return 1123
 set seed for generator 1205
 srand() 1205
RBF caching
 _os_ss_cache() 885
 disable 885
 enable 885
rbf.h 29
read 736
 bytes from path 1124, 1127
 data from
 device 802
 file 291, 802
 device path options 727
 event value 642
 event without waiting 229
 file descriptor sector 731
 fread() 291
 logical unit options 736
 path descriptor option section 738
 text line 804
 word from file 365

read input
scatter data
 readv() 1129
read() 55, 113, 1124
readdir() 113, 1126
readln() 55, 113, 1127
readv() 1129
realloc() 69, 107, 1131
record
 _os_ss_lock() 898
 _os_ss_ticks() 911
 _ss_lock() 1221
 _ss_ticks() 1232
 lock out 898, 1221
 wait for release 911, 1232
regcmp() 1133
regerror() 1134
regex() 1135
regexec() 1135
regexp.h 29
registers
 get value of
 get<name>() 315
 set value of 1151
regs.h 30
regular expression handler
 comparison 1135
 compiler 1133
 error handler 1134
release
 device 902, 1227
 ownership of resource lock 806
Release Thread Waiting on Signal 1020
Release Threads Waiting for Condition Variable 1014
remainder
 ldiv() 447
 return 210, 447
remove
 device 208, 609
 from IRQ table 756, 758
 file 1136

pending alarm request 142, 546, 820, 821
Remove Routine at Top of Stack 1010
remove() 55, 106, 1136
rename file 903, 1137
rename() 55, 106, 1137
re-open file 298
report file pointer position 312
reposition file pointer 307, 835
request
 internal memory 384
 system memory 880
Request, Clear Interrupt 1052
reset
 alarm clock 822
 existing alarm request 547
 position of directory stream 1139
 signal intercept context stack 867
resize
 current process' signal queue block 869
 data memory area 781
 memory block 462, 1131
resource lock
 _os_acqlk() 85, 536
 _os_caqlk() 85, 567
 _os_crlk() 85, 596
 _os_ddlk() 85, 599
 _os_dellk() 85, 605
 _os_rellk() 86, 806
 _os_waitlk() 86, 954
 acquire 85, 536
 activate 86, 954
 conditionally acquire ownership of 85, 567
 create 85, 596
 deadlock situation 85, 599
 delete descriptor 85, 605
 get ownership 536
 list of functions 85
 release ownership 86, 806
restore
 device 1228
 head 904

interrupt level 400
retension tape drive 905
return
 amount of free space on device 735
 character 266
 current
 file size 744
 interrupt level 401
 location 1288
 static storage pointer 359
data for memory map 742
device
 name 723, 861
 type 725
event information 636
exception table base address 333
file
 descriptor 720
 pointer to zero 1138
fixed block of memory 807
free() 293
from interrupt exception 808
memory 293, 448
parent process ID 342
parts of real number 497
pointer 1126
portions of floating point number 300
process ID 341
quotient and remainder 210
random integer 1123
size of unused stack area 294
system
 identification 924
 memory 882
user ID 364
 word from file 365
Return Specified Key Value 1043
re-validate module CRC 1162
rewind() 43, 1138
rewinddir() 114, 1139
rindex() 87, 114, 1140

Routine at Top of Stack, Remove 1010
Routine Once per Process, Execute 1091
RTS line

_os_ss_dsrts() 890
_os_ss_enrts() 891
_ss_dsrts() 1219
_ss_enrts() 1220
disable 890, 1219
enable 891, 1220

S

S_IREAD 562
S_IWRITE 562
sbf.h 30
sbrk() 69, 114, 1142
scanf() 55, 1144
scanlist 106, 304
scanset 304
scf.h 31
SCSI device
 _os_gs_dsize() 728
 get size of 728
search
 array 163
 bit map for free area 833
 for character in string 387, 1140
 memory 483
 string for pattern 272, 274
SEEK_END 307
SEEK_SET 307
seekdir() 114, 1147
select() 1149
semaphore
 initialize data structure for use 836
 release 841
 reserve 838
semaphore.h 31
semaphores 62
send
 data carrier

lost signal [1217](#)
present signal [1218](#)
signal
 after specified seconds [134](#)
 at Gregorian date/time [136, 538](#)
 at Julian date/time [138, 542](#)
 at specific time [140, 143, 540, 544, 549, 812](#)
 on data ready [907, 1230](#)
 on DCD false [887](#)
 on DCD true [888](#)
 to another process [842](#)
 to process [441](#)
 to program [1122](#)
 system-state subroutine [818](#)
serial connection
 [_os_ss_break\(\) 884](#)
 break [884](#)
service request table initialization [853, 855](#)
set
 alarm clock [824](#)
 alternate working module directory [559](#)
 bits in bit map [553](#)
 buffer for I/O stream [1183](#)
 current file
 position [309](#)
 size [1229](#)
 cyclic alarm clock [816](#)
 device
 path options [889](#)
 status [851](#)
 error trap handler [915, 918](#)
 event variable [231, 643, 645, 647, 651](#)
 exception table base address [1164](#)
 file
 attributes [883, 1213](#)
 size [908](#)
 status [851, 1177](#)
 global descriptor pointer [1165](#)
 I/O for new process [754](#)
 IRQ level [396](#)
 memory reclamation bound [296](#)

minimum allocation size 476
path options 1223
position of next readdir 1147
process
 priority 850, 1175
 signal handler 390
relative event variable 233, 649
seed for random number generator 1205
system
 date 846, 848
 global variable 857, 1180
 time 846, 848, 1167
user ID 859, 1182
value of registers
 set<name>() 1151
Set Cancel State 1096
Set Cancel Type 1098
Set Forked Threads Detached 995
Set Non-Blocking Lock 1072
Set of Signal Values, Specify 1102
Set OS-9 Priority for Calling Thread 1100
Set Priority Member 1000
Set Process-Shared Attribute 1034, 1070, 1085, 1087, 1089
Set Stack Size 1005
set up signal intercept trap 749
set user identification 1166
set_excpt_base() 95, 1164
set_gdtr() 95, 1165
setbuf() 55, 1161
setgid() 1166
setime() 94, 114, 1167
setjmp() 73, 1169
setjmp.h 31
setlocale() 73, 1171–1174
setpr() 84, 114, 1175
setpwent() 1176
SetStat
 _os_setstat() 45, 851
 _os_setsys() 46, 857
 _os_ss_attr() 46, 883
 _os_ss_break() 46, 884

_os_ss_cache() 46, 885
_os_ss_close() 46, 886
_os_ss_dcoff() 46, 887
_os_ss_dcon() 46, 888
_os_ss_dopt() 46, 889
_os_ss_dsrts() 46, 890
_os_ss_enrts() 46, 891
_os_ss_erase() 46, 892
_os_ss_fd() 47, 893
_os_ss_fillbuf 894
_os_ss_fillbuff() 47
_os_ss_flushmap() 47, 895
_os_ss_hdlink() 47, 896
_os_ss_lock() 47, 898
_os_ss_luopt() 47, 899
_os_ss_popt() 47, 901
_os_ss_relea() 47, 902
_os_ss_rename() 47, 903
_os_ss_reset() 47, 904
_os_ss_reten() 905
_os_ss_rfm() 47, 906
_os_ss_sendsig() 48, 907
_os_ss_size() 48, 908
_os_ss_skip() 48, 909
_os_ss_skipend() 48, 910
_os_ss_ticks() 48, 911
_os_ss_wfm() 48, 912
_os_ss_wtrack() 48, 913
_os9_ss_open() 47, 900
_ss_attr() 48, 1213
_ss_dcoff() 48, 1217
_ss_dcon() 48, 1218
_ss_dsrts() 48, 1219
_ss_enrts() 48, 1220
_ss_lock() 48, 1221
_ss_opt() 48, 1223
_ss_pfd() 49, 1225
_ss_rel() 49, 1227
_ss_rest() 49, 1228
_ss_size() 49, 1229
_ss_ssig() 49, 1230

_ss_tiks() 49, 1232
_ss_wtrk() 49, 1234
break serial connection 46, 884
disable
 RBF caching 46, 885
 RTS line 46, 48, 890, 1219
enable
 RBF caching 46, 885
 RTS line 46, 48, 891, 1220
erase tape 46, 892
fill path buffer with data 47, 894
flush cached bit map information for RBF device 47, 895
list of functions 43
lock out record 47, 48, 898, 1221
make hard link to existing file 47, 896
notify driver that path has been
 closed 46, 886
 opened 47, 900
put file descriptor sector 49, 1225
release device 47, 49, 902, 1227
rename file 47, 903
restore
 device 49, 1228
 head to track zero 47, 904
retension pass on tape drive 905
send
 data carrier
 lost signal to process 48, 1217
 present signal to process 48, 1218
 signal
 on data ready 48, 49, 907, 1230
 when DCD line goes false 46, 887
 when DCD line goes true 46, 888
set
 current file size 49, 1229
 device path options 46, 889
file
 attributes 46, 48, 883, 1213
 size 48, 908
 status 48, 1177
file/device status 45, 851

path options 48, 1223
system global variable 46, 857
setstat() 48, 1177
skip
 blocks 48, 909
 tape marks 47, 906
 to end of tape 48, 910
wait
 for record release 49, 1232
 specified number of ticks for record release 48, 911
write
 file descriptor sector 47, 893
 logical unit options 47, 899
 option section of path descriptor 47, 901
 tape marks 48, 912
 track 48, 49, 913, 1234
setstat() 48, 114, 1177
setsys.h 31
settrap.h 32
setuid() 73, 115, 1182
setvbuf() 55, 1183
sg_codes.h 32
sgstat.h 32
shell
 execute command 1283
 system() 1283
SIG1010 101, 1193
SIG1111 101, 1193
SIGABRT 99, 130, 148, 1191
SIGADDR 101, 1193
SIGALIGN 103, 1195
SIGALRM 100, 1192
SIGBNDCHK 102, 1194
SIGBRKPT 102, 1194
SIGCHECK 102, 103, 1195
SIGCHK 101, 1193
SIGCOPRCV 1193
SIGDBG 102, 1194
SIGDBLFLT 102, 1194
SIGFMTERR 1193
SIGFPE 99, 1191

SIGGPROT 102, 1194
SIGHUP 100, 1192
SIGILL 99, 104, 1191
SIGINST 102, 1195
SIGINT 99, 1191
SIGINVTSS 102, 1194
SIGKILL 100, 1192
sigmask() 63, 115, 1185–1187
signal
 _ev_pulse() 227
 _os_intercept() 749
 _os_send() 842
 _os_sigmask() 865
 _ss_ssig() 1230
cancel signal to send on data ready 1227
define mask 1185
event occurrence 227, 231, 233, 235, 640, 643, 645,
 647, 649, 651, 653
kill 441
mask 865
pause() 965
raise() 1122
reset intercept context stack 867
send 842
 on data ready 1230
 to program 441, 1122
set up intercept trap 749
unmask 865
wait for 965
signal handler
 control for process 1185
 defined 1188
 intercept() 390
 set up 390
 sigmask() 1185
 signal() 1188
 specify 1188
Signal Values, Specify Set of 1102
signal() 63, 98, 104, 1188–1196
Signal, Release Thread Waiting on 1020
signal.h 33

signals

 asynchronous 1195
 available 1190
 default handling 1190
 function handling 1195
 ignore handling 1190
 SIGNMI 102, 1194
 SIGPIPE 100, 1192
 SIGPRIV 101, 103, 1193, 1195
 SIGQUIT 100, 1192
 SIGSEGNP 102, 1194
 SIGSEGV 99, 1191
 SIGSTACK 102, 1194
 SIGTERM 99, 1191
 SIGTRACE 101, 1193
 SIGTRAPV 101, 1193
 SIGUNIRQ 1193
 SIGUSR1 100, 1192
 SIGUSR2 100, 1192
 SIGWAKE 100, 1192
 sin() 66, 1197
 sine function 1197
 arc 147
 hyperbolic 1198
 sinh() 66, 1198
 Size, Get Stack 991
 Size, Set Stack 1005
 skip
 blocks 909
 tape marks 906
 to end of tape 910
 sleep
 _os_sleep() 870
 _os9_sleep() 872
 current process 870
 for specified interval 1310
 tsleep() 1310
 sleep() 85, 115
 Specified Function at Exit, Execute 1012
 Specified Interval, Wait for 1022
 Specified Key Value, Return 1043

specify
 signal handling 1188
Specify Set of Signal Values 1102
sprintf() 55, 1202
sqrt() 66, 1204
square root function 1204
srand() 66, 1205
srqcmem() 69, 115, 1206
sscanf() 55, 1215
stack
 _freemem() 294
 _stacksiz() 1236
 determine size of area 294
 freemem() 294
 get size of 1236
 stacksiz() 1236
Stack Address, Get 989
stack frame
 allocate space
 alloca() 135
Stack Size, Get 991
Stack Size, Set 1005
Stack, Remove Routine at Top of 1010
stacksiz() 1236
standard white space characters 432
start
 next process 788
 system debugger 1281
stat() 1237
State Attribute, Get Detach 983
State, Set Cancel 1096
static storage
 _os_initdata() 748
 change current pointer 175
 get_static() 359
 initialize 748
 return current pointer 359
status functions
 _gs_devn() 370
 _gs_eof() 372
 _gs_gfd() 374

_gs_opt() 376
_gs_pos() 378
_gs_rdy() 380
_gs_size() 382
_os_getstat() 712
_os_getsys() 714
_os_gs_cpypd() 721
_os_gs_cstats() 722
_os_gs_devnm() 723
_os_gs_devtyp() 725
_os_gs_dopt() 727
_os_gs_dsize() 728
_os_gs_edt() 729
_os_gs_eof() 730
_os_gs_fd() 731
_os_gs_faddr() 732
_os_gs_fdinf() 733
_os_gs_luopt() 736
_os_gs_parity() 737
_os_gs_popt() 738
_os_gs_pos() 740
_os_gs_ready() 743
_os_gs_size() 744
_os_setstat() 851
_os_setsys() 857
_os_sgetstat() 860
_os_sgs_devnm() 861
_os_ss_attr() 883
_os_ss_break() 884
_os_ss_cache() 885
_os_ss_close() 886
_os_ss_dcöff() 887
_os_ss_dcon() 888
_os_ss_dopt() 889
_os_ss_dsrts() 890
_os_ss_enrts() 891
_os_ss_erase() 892
_os_ss_fd() 893
_os_ss_fillbuff() 894
_os_ss_flushmap() 895
_os_ss_hdlink() 896

_os_ss_lock() 898
_os_ss_luopt() 899
_os_ss_popt() 901
_os_ss_relea() 902
_os_ss_rename() 903
_os_ss_reset() 904
_os_ss_reten() 905
_os_ss_rfm() 906
_os_ss_sendsig() 907
_os_ss_size() 908
_os_ss_skip() 909
_os_ss_skipend() 910
_os_ss_ticks() 911
_os_ss_wfm() 912
_os_ss_wtrack() 913
_os9_gs_cdfd() 720
_os9_gs_free() 735
_os9_ss_open() 900
_ss_attr() 1213
_ss_dcoff() 1217
_ss_dcon() 1218
_ss_dsrts() 1219
_ss_enrts() 1220
_ss_lock() 1221
_ss_opt() 1223
_ss_pfd() 1225
_ss_rel() 1227
_ss_rest() 1228
_ss_size() 1229
_ss_ssig() 1230
_ss_tiks() 1232
_ss_wtrk() 1234
break serial connection 884
calculate parity of file descriptor 737
check for end of file 372
disable
 RBF caching 885
 RTS line 890, 1219
enable
 RBF caching 885
 RTS line 891, 1220

erase tape 892
examine system global variable 714
fill path buffer with data 894
flush cached bit map information for RBF device 895
get
 cache status information 722
 current file
 position 378, 740
 size 382
 device name 370
 file
 descriptor block address 732
 descriptor sector 374
 status 356
 file/device status 712
 I/O interface edition number 729
 path options 376
 size of SCSI device 728
 specified file descriptor sector 733
GetStat call using system path number 860
getstat() 356
lock out record 898, 1221
make hard link to existing file 896
notify driver that path has been
 closed 886
 opened 900
put file descriptor sector 1225
read
 device path options 727
 file descriptor sector 731
 logical unit options 736
 path descriptor option section 738
release device 902, 1227
rename file 903
restore
 device 1228
 head to track zero 904
retension pass on tape drive 905
return
 amount of free space on device 735
 current file size 744

device name 723, 861
device type 725
file descriptor 720
send
 data carrier
 lost signal to process 1217
 present signal to process 1218
signal
 on data ready 907, 1230
 when DCD line goes
 false 887
 true 888
set
 current file size 1229
 device path options 889
 file
 attributes 883, 1213
 size 908
 status 1177
 file/device status 851
 path options 1223
 system global variables 857
setstat() 1177
skip
 blocks 909
 tape marks 906
 to end of tape 910
test for
 data available 380
 data ready 743
 end of file 730
wait
 for record release 1232
 specified number of ticks for record release 911
write
 file descriptor sector 893
 logical unit options 899
 option section of path descriptor 901
 tape marks 912
 track 913, 1234

Status, Store Thread Exit 1054
stdarg.h 33
stdde.h 33
stdio.h 34
stdlib.h 35
store current value of file position indicator 267
Store Thread Exit Status 1054
strcat() 87, 1240
strchr() 87, 1241
strcmp() 87, 1242
strcoll() 87, 1243
strcpy() 87, 1244
strcspn() 87, 1245
strerror() 87
strftime() 87, 94, 1248–1250
strncpy() 87, 114, 1251
string
 break into tokens 1264
 catenate 1240, 1254
 compare 191, 1242, 1243, 1255
 compute length 1259
 convert to
 double 1261
 long 1266, 1269
 copy 1244, 1256
 old OS-9 strings 1251
 determine length 1253
 findnstr() 272
 findstr() 274
 get
 from file 354
 length 1245
 index() 387
 input conversion 301
 locate 1241
 first occurrence of 1257, 1260
 last occurrence of 1258
 output to file 290, 1117
 return next option letter
 getopt() 338
 search for character in 387

- search for first instance of pattern 274
- search for first occurrence of pattern 272
 - strcat() 1240
 - strchr() 1241
 - strcmp() 1242
 - strcoll() 1243
 - strcpy() 1244
 - strcspn() 1245
 - strncpy() 1251
 - strlen() 1253
 - strncat() 1254
 - strncmp() 1255
 - strncpy() 1256
 - strupr() 1257
 - strrchr() 1258
 - strspn() 1259
 - strstr() 1260
 - strtod() 1261
 - strtok() 1264
 - strtol() 1266
 - strtoul() 1269
 - strxfrm() 1272
- transform 1272
- string handling functions
 - break string into tokens 88, 1264
 - compute string length 88, 1259
 - copy old OS-9 strings 87, 1251
 - determine string length 88, 1253
 - findnstr() 87, 272
 - findstr() 87, 274
 - get string length 87, 1245
 - index() 87, 387
 - list of 87
 - locate
 - first occurrence of string 88, 1257, 1260
 - last occurrence of string 88, 1258
 - string 87, 1241
 - map error message string 87
 - place formatted time in buffer 87, 94, 1248, 1249
 - rindex() 87, 1140
 - search

for character in string 87, 387, 1140
string for pattern 87, 272, 274
strcat() 87, 1240
strchr() 87, 1241
strcmp() 87, 1242
strcoll() 87, 1243
strcpy() 87, 1244
strcspn() 87, 1245
strerror() 87
strftime() 87, 94, 1248, 1249
strlcpy() 87, 1251
string
 catenation 87, 88, 1240, 1254
 comparison 87, 88, 1242, 1243, 1255
 copy 87, 88, 1244, 1256
 to double conversion 88, 1261
 to long conversion 88, 1266, 1269
strlen() 88, 1253
strncat() 88, 1254
strncmp() 88, 1255
strncpy() 88, 1256
strpbrk() 88, 1257
strrchr() 88, 1258
strspn() 88, 1259
strstr() 88, 1260
strtod() 88, 1261
strtok() 88, 1264
strtol() 88, 1266
strtoul() 88, 1269
strxfrm() 88, 1272
 transform string 88, 1272
string.h 36
strings.h 36
strlen() 88, 1253
strncat() 88, 1254
strncmp() 88, 1255
strncpy() 88, 1256
strpbrk() 88, 1257
strrchr() 88, 1258
strspn() 88, 1259
strstr() 88, 1260

strtod() 88, 1261–1263
strtok() 88, 1264
strtol() 88, 1266–1268
strtoul() 88, 1269–1271
structure assignment 1239
strxfrm() 88, 1272
suspend
 process 921
 task 965
Suspendability Counter, Decrement 1106
Suspendability Counter, Increment 1108
Suspension Counter, Decrement 1093
Suspension Counter, Increment 1110
svctbl.h 36
synchronous I/O multiplexing 1149
sys_clib.l 11, 108
sys_mktime() 94, 114, 1276
sysglob.h 37
SYSRAM 473, 496, 784, 1206
system
 _os_sysid() 924
 _os9_setime() 848
 catastrophic occurrence 791
 memory
 request 880, 1209
 return 1211
 return identification 924
 set date/time 848
system call processing 666, 672, 762
system debugger
 _os_sysdbg() 922
 _sysdbg() 1281
 call 922, 1281
system global variables
 _getsys() 360
 _os_getsys() 714
 _os_setsys() 857
 _setsys() 1180
 change 857, 1180
 examine 714, 1180
 get 360

set 857, 1180
system() 40, 1283
system_addr() 1285

T

tan() 66, 1286
tangent function 1286
 arc 150, 151
 hyperbolic 1287
tanh() 66, 1287
tape
 _os_ss_erase() 892
 _os_ss_reset() 904
 _os_ss_reten() 905
 _os_ss_rfm() 906
 _os_ss_skip() 909
 _os_ss_skipend() 910
 _os_ss_wfm() 912
 erase 892
 restore head to track zero 904
 retension 905
 skip
 blocks 909
 marks 906
 to end 910
 write marks 912
Target Thread, Cancel 1008
task
 _os_alltsk() 557
 allocate 557
 pause() 965
 suspend 965
 termination 253, 254
telldir() 115, 1288
tempnam() 1289
temporary files
 create names for 1289
TERM 1291
TERMCAP 1291
Termcap

get entry 1291
tgetent() 1291
termcap.h 37
terminal
 check capability reference 1293
 get capability ID 1294, 1295
 tgetflag() 1293
 tgetnum() 1294
 tgetstr() 1295
terminal manipulation functions
 check terminal capability presence 89, 1293
 get
 cursor movement capability 89, 1297
 termcap entries 89, 1291
 terminal capability 89, 1295
 terminal capability ID 89, 1294
 list of 89
 output capability string 89, 1308
 tgetent() 89, 1291
 tgetflag() 89, 1293
 tgetnum() 89, 1294
 tgetstr() 89, 1295
 tgoto() 89, 1297
 tputs() 89, 1308
terminal name
 find
 isatty() 408
terminate
 _exit() 253
 calling process 677
 I/O for exiting process 753
 program 254
 task 253, 254
Terminate Calling Thread 1041
termio.h 37
termlib.l 11
test
 _isascii() 406
 _isjis_kana() 414, 416, 420
 _issjis1() 428
 _issjis2() 430

error indicator 263
for
 alphabetic character 404
 alphanumeric 402
 ASCII 406
 control code 410
 data available 380
 data ready 743
 digit 412
 end of file 262, 730
 Hankaku-kana 414, 416, 420
 hexadecimal 437
 Kanji 428, 430
 lowercase 422
 printable character 424
 printing character 418
 punctuation 426
 uppercase 435
 white space 432
isalnum() 402
isalpha() 404
isascii() 406
iscntrl() 410
isdigit() 412
isgraph() 418
islower() 422
isprint() 424
ispunct() 426
isspace() 432
isupper() 435
isxdigit() 437
Test for Pending Cancel 1112
tgetent() 89, 1291
tgetflag() 89, 1293
tgetnum() 89, 1294
tgetstr() 89, 1295
tgoto() 89, 1297
Thread Exit Status, Store 1054
Thread IDs, Compare 1040
Thread Process, Interrupt 1050
Thread Value with a Key, Associate a 1104

Thread Waiting on Signal, Release 1020
Thread, Cancel Target 1008
Thread, Create New 1036
Thread, Orphan Designated 1038
Thread, Set OS-9 Priority for Calling 1100
Thread, Terminate Calling 1041
Threads Detached, Set Forked 995
Threads Waiting for Condition Variable, Release 1014
Thread-Specific Data Key, Create 1058
Thread-Specific Data Key, Delete 1060
time
 _os9_alarm_atdate() 538
 _os9_alarm_atjul() 542
 broken-down structure 1276
 get current 362
 place formatted time in buffer 1248, 1249
 send signal at Julian date/time 542
time functions
 _gregorian() 93, 368
 _julian() 93, 439
 _os_gettime() 94, 696
 _os_gregorian() 94, 718
 _os_julian() 94, 760
 _os_setime() 94, 846
 _os9_gettime() 94, 698
 _os9_setime() 94, 848
 _sysdate() 94
 asctime() 93, 145
 broken-down
 convert to calendar 493
 clock() 93, 187
 convert
 broken-down to
 calendar 94, 493
 string format 93, 145
 calendar to
 Greenwich Mean Time 93, 367
 local 94, 457
 string format 93, 206
 date/time to Gregorian value 93, 368
 date/time to Julian value 93

julian date and time [439](#)
local to Greenwich Mean Time [94, 1276](#)
ctime() [93, 206](#)
difftime() [93, 209](#)
find temporal difference [93, 209](#)
get [698, 1299](#)
 calendar time [94](#)
 current [94](#)
 Gregorian date [94, 718](#)
 Julian date [94, 760](#)
 processor time [93, 187](#)
 system date/time [94, 696](#)
 system time [93, 335](#)
getime() [93, 335](#)
gmtime() [93, 367](#)
list of [93](#)
localtime() [94, 457](#)
mktime() [94, 493](#)
set [94, 846, 848, 1167](#)
setime() [94, 1167](#)
sys_mktime() [94, 1276](#)
time() [94, 1299](#)
time zones [90](#)
 defaults [108](#)
time() [94, 1299](#)
time.h [38](#)
times() [1300](#)
tmpfile() [55, 1301](#)
tmpnam() [73, 1302](#)
toascii() [1303](#)
tolower() [42, 1305](#)
Top of Stack, Remove Routine at [1010](#)
toupper() [42, 1307](#)
tputs() [89, 1308](#)
track
 _os_ss_wtrack() [913](#)
 _ss_wtrk() [1234](#)
 format [913](#)
 write [913, 1234](#)
transform string [1272](#)
translate

character 1303
memory address 939
trap handler
 _os_strap() 915
 _os_tlink() 932
 _os_tlinkm() 934
 _os9_strap() 918
 install system state module 932
 install user module 934
 set for error 915, 918
tsleep() 85, 115, 1310
Type, Set Cancel 1098
TZ 90, 108

U

un.h 38
unget character 1312
ungetc() 56, 1312
unlink
 event 237, 657
 file 1314, 1316
 from module 528
 module 943, 944
unlink() 56, 115, 1314
unlinkx() 56, 115, 1316
unload module 530
unmask
 interrupts 398
 signals 865
user accounting 941
user ID
 _os_id() 745
 _os_setuid() 859
 _os9_id() 747
 determine 364
 get 745, 747
 set 859, 1182
 setuid() 1182
user_addr() 1318
UTC 90, 367

V

va_arg() 73, 1319
va_end() 74, 1320
va_start() 74, 1321
valid file access modes 260
validate module 947
Value of Attribute, Get 1064, 1077, 1079, 1081
Value with a Key, Associate a Thread 1104
Value, Return Specified Key 1043
Values, Specify Set of Signal 1102
variable parameter list
 end references to 1320
 get parameter 1319
 initialize 1321
 va_arg() 1319
 va_end() 1320
 va_start() 1321
Variable, Block on a Condition 1026
Variable, Release Threads Waiting for Condition 1014
verify
 attached device 159
 module 945, 947
vfprintf() 56, 1323
VIDEO1 473, 496, 784, 1206
VIDEO2 473, 496, 784, 1207
vprintf() 56, 1325
vsprintf() 56, 1327

W

wait
 for child process to terminate 949
 for event to occur 239, 655, 658, 660
 for process termination 1329
 for record release 1232
 for relative event to occur 241, 662, 664
 for signal 965
 specified number of ticks 911
Wait for Specified Interval 1022
wait() 85, 115, 1329

Waiting for Condition Variable, Release Threads 1014
Waiting on Signal, Release Thread 1020
wcstombs() 78, 1333
wctomb() 78, 1335
WET 90
write 961
 bytes to path 1337, 1339
 data 313, 956
 file descriptor sector 893
 fwrite() 313
 integer to I/O address 962
 line of text 958
 logical unit options 899
 option section of path descriptor 901
 string to file 290
 tape marks 912
 track 913, 1234
 word to I/O address 963
write iovcnt buffer data 1341
write() 56, 115, 1337
writeln() 56, 115, 1339
writev() 1341

Y

YST 90

Product Discrepancy Report

To: Microware Customer Support

FAX: 515-224-1352

From: _____

Company: _____

Phone: _____

Fax: _____ Email: _____

Product Name:

Description of Problem:

Host Platform_____

Target Platform_____



MICROWARE SOFTWARE