
System Functions and Subroutines

This chapter describes extensions to Fortran 77 that are related to the IRIX compiler and operating system.

- “Library Functions” summarizes the Fortran run-time library functions.
- “Extended Intrinsic Subroutines” describes the extensions to the Fortran intrinsic subroutines.
- “Extended Intrinsic Functions” describes the extensions to the Fortran functions.

Library Functions

The Fortran library functions provide an interface from Fortran programs to the IRIX system functions. System functions are facilities that are provided by the IRIX system kernel directly, as opposed to functions that are supplied by library code linked with your program. System functions are documented in volume 2 of the reference pages, with an overview in the intro(2) reference page.

Table 4-1 summarizes the functions in the Fortran run-time library. In general, the name of the interface routine is the same as the name of the system function that would be called from a C program. For details on a system interface routine use the command:

```
man 2 name_of_function
```

Note: You must declare the **time** function as EXTERNAL; if you do not, the compiler will assume you mean the VMS-compatible intrinsic **time** function rather than the IRIX system function. (In general it is a good idea to declare any library function in an EXTERNAL statement as documentation.)

Table 4-1 Summary of System Interface Library Routines

Function	Purpose
abort(3F)	abnormal termination
access	determine accessibility of a file
acct	enable/disable process accounting
alarm(3F)	execute a subroutine after a specified time
barrier(3P)	perform barrier operations
blockproc(2)	block processes
brk(2)	change data segment space allocation
chdir	change default directory
chmod(3F)	change mode of a file
chown(2)	change owner
chroot	change root directory for a command
close	close a file descriptor
creat	create or rewrite a file
ctime(3F)	return system time
dtime(3F)	return elapsed execution time
dup	duplicate an open file descriptor
etime(3F)	return elapsed execution time
exit(2)	terminate process with status
fcntl(2)	file control
fdate(3F)	return date and time in an ASCII string
fgetc(3F)	get a character from a logical unit
fork(2)	create a copy of this process
fputc(3F)	write a character to a Fortran logical unit

Table 4-1 (continued) Summary of System Interface Library Routines

Function	Purpose
free_barrier(3P)	free barrier
fseek(3F)	reposition a file on a logical unit
fseek64(3F)	reposition a file on a logical unit for 64-bit architecture
fstat(2)	get file status
ftell(3F)	reposition a file on a logical unit
ftell64(3F)	reposition a file on a logical unit for 64-bit architecture
gerror(3F)	get system error messages
getarg(3F)	return command line arguments
getc(3F)	get a character from a logical unit
getcwd	get pathname of current working directory
getdents(2)	read directory entries
getegid(2)	get effective group ID
gethostid(2)	get unique identifier of current host
getenv(3F)	get value of environment variables
geteuid(2)	get effective user ID
getgid(2)	get user or group ID of the caller
gethostname(2)	get current host ID
getlog(3F)	get user's login name
getpgrp	get process group ID
getpid	get process ID
getppid	get parent process ID
getsockopt(2)	get options on sockets
getuid(2)	get user or group ID of caller
gmtime(3F)	return system time

Table 4-1 (continued) Summary of System Interface Library Routines

Function	Purpose
iargc(3F)	return command line arguments
idate(3F)	return date or time in numerical form
ierrno(3F)	get system error messages
ioctl(2)	control device
isatty(3F)	determine if unit is associated with tty
itime(3F)	return date or time in numerical form
kill(2)	send a signal to a process
link(2)	make a link to an existing file
loc(3F)	return the address of an object
lseek(2)	move read/write file pointer
lseek64(2)	move read/write file pointer for 64-bit architecture
lstat(2)	get file status
ltime(3F)	return system time
m_fork(3P)	create parallel processes
m_get_myid(3P)	get task ID
m_get_numprocs(3P)	get number of subtasks
m_kill_procs(3P)	kill process
m_lock(3P)	set global lock
m_next(3P)	return value of counter
m_park_procs(3P)	suspend child processes
m_rele_procs(3P)	resume child processes
m_set_procs(3P)	set number of subtasks
m_sync(3P)	synchronize all threads
m_unlock(3P)	unset a global lock

Table 4-1 (continued) Summary of System Interface Library Routines

Function	Purpose
mkdir(2)	make a directory
mknod(2)	make a directory/file
mount(2)	mount a filesystem
new_barrier(3P)	initialize a barrier structure
nice	lower priority of a process
open(2)	open a file
oserror(3F)	get/set system error
pause(2)	suspend process until signal
perror(3F)	get system error messages
pipe(2)	create an interprocess channel
plock(2)	lock process, test, or data in memory
prctl(2)	control processes
profil(2)	execution-time profile
ptrace	process trace
putc(3F)	write a character to a Fortran logical unit
putenv(3F)	set environment variable
qsort(3F)	quick sort
read	read from a file descriptor
readlink	read value of symbolic link
rename(3F)	change the name of a file
rmdir(2)	remove a directory
sbrk(2)	change data segment space allocation
schedctl(2)	call to scheduler control
send(2)	send a message to a socket

Table 4-1 (continued) Summary of System Interface Library Routines

Function	Purpose
setblockproccnt(2)	set semaphore count
setgid	set group ID
sethostid(2)	set current host ID
setoserror(3F)	set system error
setpgrp(2)	set process group ID
setsockopt(2)	set options on sockets
setuid	set user ID
sginap(2)	put process to sleep
sginap64(2)	put process to sleep in 64-bit environment
shmat(2)	attach shared memory
shmdt(2)	detach shared memory
sighold(2)	raise priority and hold signal
sigignore(2)	ignore signal
signal(2)	change the action for a signal
sigpause(2)	suspend until receive signal
sigrelse(2)	release signal and lower priority
sigset(2)	specify system signal handling
sleep(3F)	suspend execution for an interval
socket(2)	create an endpoint for communication TCP
sproc(2)	create a new share group process
stat(2)	get file status
stime(2)	set time
symlink(2)	make symbolic link
sync	update superblock

Table 4-1 (continued) Summary of System Interface Library Routines

Function	Purpose
sysmp(2)	control multiprocessing
sysmp64(2)	control multiprocessing in 64-bit environment
system(3F)	issue a shell command
taskblock(3P)	block tasks
taskcreate(3P)	create a new task
taskctl(3P)	control task
taskdestroy(3P)	kill task
tasksetblockcnt(3P)	set task semaphore count
taskunblock(3P)	unblock task
time(3F)	return system time (must be declared EXTERNAL)
ttynam(3F)	find name of terminal port
uadmin	administrative control
ulimit(2)	get and set user limits
ulimit64(2)	get and set user limits in 64-bit architecture
umask	get and set file creation mask
umount(2)	dismount a file system
unblockproc(2)	unblock processes
unlink(2)	remove a directory entry
uscalloc(3P)	shared memory allocator
uscalloc64(3P)	shared memory allocator in 64-bit environment
uscas(3P)	compare and swap operator
usclosetpollsema(3P)	detach file descriptor from a pollable semaphore
usconfi g(3P)	semaphore and lock configuration operations
uscpssema(3P)	acquire a semaphore

Table 4-1 (continued) Summary of System Interface Library Routines

Function	Purpose
uscsetlock(3P)	unconditionally set lock
usctlsema(3P)	semaphore control operations
usdumplock(3P)	dump lock information
usdumpsema(3P)	dump semaphore information
usfree(3P)	user shared memory allocation
usfreelock(3P)	free a lock
usfreepollsema(3P)	free a pollable semaphore
usfreeseма(3P)	free a semaphore
usgetinfo(3P)	exchange information through an arena
usinit(3P)	semaphore and lock initialize routine
usinitlock(3P)	initialize a lock
usinitsema(3P)	initialize a semaphore
usmalloc(3P)	allocate shared memory
usmalloc64(3P)	allocate shared memory in 64-bit environment
usmallopt(3P)	control allocation algorithm
usnewlock(3P)	allocate and initialize a lock
usnewpollsema(3P)	allocate and initialize a pollable semaphore
usnewsema(3P)	allocate and initialize a semaphore
usopenpollsema(3P)	attach a file descriptor to a pollable semaphore
uspsema(3P)	acquire a semaphore
usputinfo(3P)	exchange information through an arena
usrealloc(3P)	user share memory allocation
usrealloc64(3P)	user share memory allocation in 64-bit environment
ussetlock(3P)	set lock

Table 4-1 (continued) Summary of System Interface Library Routines

Function	Purpose
ustestlock(3P)	test lock
ustestsema(3P)	return value of semaphore
usunsetlock(3P)	unset lock
usvsema(3P)	free a resource to a semaphore
uswsetlock(3P)	set lock
wait(2)	wait for a process to terminate
write	write to a file

Extended Intrinsic Subroutines

This section describes the intrinsic subroutines that are extensions to Fortran 77 (the intrinsic *functions* that are standard to Fortran 77 are documented in Appendix A of the *MIPSpro Fortran 77 Language Reference Manual*). The rules for using the names of intrinsic subroutines are also discussed in that appendix.

Table 4-2 gives an overview of the intrinsic subroutines and their function; they are described in detail in the sections following the topics.

Table 4-2 Overview of System Subroutines

Subroutine	Information Returned
DATE	Current date as nine-byte string in ASCII representation
IDATE	Current month, day, and year, each represented by a separate integer
ERRSNS	Description of the most recent error
EXIT	Terminates program execution
TIME	Current time in hours, minutes, and seconds as an eight-byte string in ASCII representation
MVBITS	Moves a bit field to a different storage location

DATE

The **DATE** routine returns the current date as set by the system; the format is as follows:

```
CALL DATE (buf)
```

where *buf* is a variable, array, array element, or character substring nine bytes long. After the call, *buf* contains an ASCII variable in the format *dd-*mmm*-*yy**, where *dd* is the date in digits, *mmm* is the month in alphabetic characters, and *yy* is the year in digits.

IDATE

The **IDATE** routine returns the current date as three integer values representing the month, date, and year; the format is as follows:

```
CALL IDATE (m, d, y)
```

where *m*, *d*, and *y* are either **INTEGER*4** or **INTEGER*2** values representing the current month, day and year. For example, the values of *m*, *d*, and *y* on August 10, 1989, are

```
m = 8  
d = 10  
y = 89
```

ERRSNS

The **ERRSNS** routine returns information about the most recent program error; the format is as follows:

```
CALL ERRSNS (arg1, arg2, arg3, arg4, arg5)
```

The arguments (*arg1*, *arg2*, and so on) can be either **INTEGER*4** or **INTEGER*2** variables. On return from **ERRSNS**, the arguments contain the information shown in Table 4-3.

Table 4-3 Information Returned by ERRSNS

Argument	Contents
<i>arg1</i>	IRIX global variable <i>errno</i> , which is then reset to zero after the call
<i>arg2</i>	Zero
<i>arg3</i>	Zero
<i>arg4</i>	Logical unit number of the file that was being processed when the error occurred
<i>arg5</i>	Zero

Although only *arg1* and *arg4* return relevant information, *arg2*, *arg3*, and *arg5* are always required.

EXIT

The **EXIT** routine causes normal program termination and optionally returns an exit-status code; the format is as follows:

```
CALL EXIT (status)
```

where *status* is an **INTEGER*4** or **INTEGER*2** argument containing a status code.

TIME

The **TIME** routine returns the current time in hours, minutes, and seconds; the format is as follows:

```
CALL TIME (clock)
```

where *clock* is a variable, array, array element, or character substring; it must be eight bytes long. After execution, *clock* contains the time in the format *hh:mm:ss*, where *hh*, *mm*, and *ss* are numerical values representing the hour, the minute, and the second.

MVBITS

The **MVBITS** routine transfers a bit field from one storage location to another; the format is as follows:

CALL MVBITS (*source*, *sbit*, *length*, *destination*, *dbit*)

Table 4-4 defines the arguments. Arguments can be declared as **INTEGER*2**, **INTEGER*4**, or **INTEGER*8**.

Table 4-4 Arguments to MVBITS

Argument	Type	Contents
<i>source</i>	Integer variable or array element	Source location of bit field to be transferred.
<i>sbit</i>	Integer expression	First bit position in the field to be transferred from <i>source</i> .
<i>length</i>	Integer expression	Length of the field to be transferred from <i>source</i> .
<i>destination</i>	Integer variable or array element	Destination location of the bit field
<i>dbit</i>	Integer expression	First bit in <i>destination</i> to which the field is transferred.

Extended Intrinsic Functions

Table 4-5 gives an overview of the intrinsic functions added as extensions of Fortran 77.

Table 4-5 Function Extensions

Function	Information Returned
SECNDS	Elapsed time as a floating point value in seconds. This is an intrinsic routine.
RAN	The next number from a sequence of pseudo-random numbers. This is not an intrinsic routine.

These functions are described in detail in the following sections.

SECNDS

SECNDS is an intrinsic routine that returns the number of seconds since midnight, minus the value of the passed argument; the format is as follows:

```
s = SECNDS(n)
```

After execution, *s* contains the number of seconds past midnight less the value specified by *n*. Both *s* and *n* are single-precision, floating point values.

RAN

RAN generates a pseudo-random number. The format is as follows:

```
v = RAN(s)
```

The argument *s* is an **INTEGER*4** variable or array element. This variable serves as a seed in determining the next random number. It should initially be set to a large, odd integer value. You can compute multiple random number series by supplying different variables or array elements as the seed argument to different calls of **RAN**.

Note: Because **RAN** modifies the argument *s*, calling the function with a constant can cause a core dump.

The algorithm used in **RAN** is the linear congruential method. The code is similar to the following fragment:

```
S = S * 1103515245L + 12345  
RAN = FLOAT(IAND(RSHIFT(S,16),32767))/32768.0
```

RAN is supplied for compatibility with VMS. For demanding applications, consider using the functions described in the random(3b) reference page. These can all be called using techniques described under "Using %AL" on page 42.

