Chapter 4

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.
- "Intrinsic Subroutine Extensions" describes the extensions to the Fortran intrinsic subroutines.
- "Function Extensions" describes the extensions to the Fortran functions.

Library Functions

The Fortran library functions provide an interface from Fortran programs to the system in the same way that the C library provides for C programs. The compiler automatically loads an interface routine when it processes the associated call.

Table 4−1summarizes the functions in the Fortran run−time library.

Table 4−1 Summary of System Interface Library Routines

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

fseek	reposition a file on a logical unit
fstat	get file status
ftell	reposition a file on a logical unit
gerror	get system error messages
getarg	return command line arguments
getc	get a character from a logical unit
getcwd	get pathname of current working directory
getdents	read directory entries
getegid	get effective group ID
gethostid	get unique identifier of current host
getenv	get value of environment variables
geteuid	get effective user ID
getgid	get user or group ID of the caller
gethostname	get current host ID
getlog	get user's login name
getpgrp	get process group ID
getpid	get process ID
getppid	get parent process ID
getsockopt	get options on sockets
getuid	get user or group ID of caller
gmtime	return system time
iargc	return command line arguments
idate	return date or time in numerical form
ierrno	get system error messages
ioctl	control device
isatty	determine if unit is associated with tty
itime	return date or time in numerical form
kill	send a signal to a process
link	make a link to an existing file
loc	return the address of an object
lseek	move read/write file pointer
lstat	get file status
ltime	return system time
m_fork	create parallel processes
m_get_myid	get task ID
m_get_numprocs	get number of subtasks
m_kill_procs	kill process
m_lock	set global lock
m_next	return value of counter
m_park_procs	suspend child processes
m_rcle_procs	resume child processes
m_set_procs	set number of subtasks
m_sync	synchronize all threads

m_unlock	unset a global lock
mkdir	make a directory
mknod	make a directory/file
mount	mount a filesystem
new_barrier	initialize a barrier structure
nice	lower priority of a process
open	open a file
oserror	get/set system error
pause	suspend process until signal
perror	get system error messages
pipe	create an interprocess channel
plock	lock process, test, or data in memory
prctl	control processes
profil	execution−time profile
ptrace	process trace
putc	write a character to a Fortran logical unit
putenv	set environment variable
qsort	quick sort
read	read from a file descriptor
readlink	read value of symbolic link
rename	change the name of a file
rmdir	remove a directory
sbrk	change data segment space allocation
schedctl	call to scheduler control
send	send a message to a socket
setblockproccnt	set semaphore count
setgid	set group ID
sethostid	set current host ID
setoserror	set system error
setpgrp	set process group ID
setsockopt	set options on sockets
setuid	set user ID
sginap	put process to sleep
shmat	attach shared memory
shmdt	detach shared memory
sighold	raise priority and hold signal
sigignore	ignore signal
signal	change the action for a signal
sigpause	suspend until receive signal
sigrelse	release signal and lower priority
sigset	specify system signal handling
sleep	suspend execution for an interval
socket	create an endpoint for communication TCP

sproc	create a new share group process
stat	get file status
stime	set time
symlink	make symbolic link
sync	update superblock
sysmp	control multiprocessing
system	issue a shell command
taskblock	block tasks
taskcreate	create a new task
taskctl	control task
taskdestroy	kill task
tasksetblockcnt	set task semaphore count
taskunblock	unblock task
time[a]	return system time
ttynam	find name of terminal port
uadmin	administrative control
ulimit	get and set user limits
umask	get and set file creation mask
umount	dismount a file system
unblockproc	unblock processes
unlink	remove a directory entry
uscalloc	shared memory allocator
uscas	compare and swap operator
usclosepollsema	detach file descriptor from a pollable semaphore
usconfig	semaphore and lock configuration operations
uscpsema	acquire a semaphore
uscsetlock	unconditionally set lock
usctlsema	semaphore control operations
usdumplock	dump lock information
usdumpsema	dump semaphore information
usfree	user shared memory allocation
usfreelock	free a lock
usfreepollsema	free a pollable semaphore
usfreesema	free a semaphore
usgetinfo	exchange information through an arena
usinit	semaphore and lock initialize routine
usinitlock	initialize a lock
usinitsema	initialize a semaphore
usmalloc	allocate shared memory
usmallopt	control allocation algorithm
usnewlock	allocate and initialize a lock
usnewpollsema	allocate and initialize a pollable semaphore
usnewsema	allocate and initialize a semaphore

usopenpollsem	attach a file descriptor to a pollable semaphore
uspsema	acquire a semaphore
usputinfo	exchange information through an arena
usrealloc	user share memory allocation
ussetlock	set lock
ustest lock	test lock
ustestsema	return value of semaphore
ustrace	trace
usunsetlock	unset lock
usvsema	free a resource to a semaphore
uswsetlock	set lock
wait	wait for a process to terminate
write	write to a file

[a] The library function *time* can be invoked only if it is declared in an external statement.
Otherwise, it will be misinterpreted as the VMS−compatible intrinsic subroutine *time*.

You can display information on a *function* with the *man* command:

% man *function*

Intrinsic Subroutine Extensions

This section describes the intrinsic subroutines that are extensions to Fortran 77. The rules for using the intrinsic subroutines are

- The subroutine names are specially recognized by the compiler. A user−written subroutine with the same name as a system subroutine must be declared in an **EXTERNAL** statement in the calling subprogram.
- Using a user−written subroutine with the same name as a system subroutine in one subprogram does not preclude using the actual system subroutine in a different subprogram.
- To pass the name of a system subroutine as an argument to another subprogram, the name of the system subroutine must be declared in an **INTRINSIC** statement in the calling subprogram.
- When a system subroutine name is passed as an argument to another subprogram, the call to the system subroutine via the formal parameter name in the receiving subprogram must use the primary calling sequence for the subprogram (when there is more than one possible calling sequence).

Table 4−2gives an overview of the system subroutines and their function; they are described in detail in the sections following the table.

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

Although only *arg1* and *agr4* 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 4−4defines the arguments. Arguments can be declared as **INTEGER*2** or **INTEGER*4**.

Table 4−4 Arguments to MVBITS

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

Function Extensions

Table 4−5gives an overview of the 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 arguments; 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

The **RAN** routine generates a random number; the format is as follows:

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

The argument *s* is an **INTEGER*4** variable or array element; *s* serves as a seed in determining the next random number and should initially be set to a large, odd integer value. This permits the computation of multiple random number series by supplying different variable names as the seed argument to **RAN**.

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