INTRODUCTION TO C SYSTEM CALLS

This section of the C compiler manual is a guide to the system calls available from C programs.

It is NOT intended as a definitive description of OS-9 service requests as these are described in the OS-9 SYSTEM PROGRAMMER'S MANUAL. However, for most calls, enough information is available here to enable the programmer to write system calls into programs without looking further.

The names used for the system calls are chosen so that programs transported from other machines or operating systems should compile and run with as little modification as possible. However, care should be taken as the parameters and returned values of some calls may not be compatible with those on other systems. Programmers that are already familiar with OS-9 names and values should take particular care. Some calls do not share the same names as the OS-9 assembly language equivalents. The assembly language equivalent call is shown, where there is one, on the relevant page of the C call description, and a cross-reference list is provided for those already familiar with OS-9 calls.

The normal error indication on return from a system call is a returned value of -1. The relevant error will be found in the predefined int "errno". Errno always contains the error from the last erroneous system call. Definitions for the errors for inclusion in the program are in "<errno.h>".

In the "SEE ALSO" sections on the following pages, unless otherwise stated, the references are to other system calls.

Where "#include" files are shown, it is not mandatory to include them, but it might be convenient to use the manifest constants defined in them rather than integers; it certainly makes for more readable programs.

Abort - stop the program and produce a core dump

USAGE

```
abort()
```

DESCRIPTION

This call causes a memory image to be written out to the file "core" in the current directory, and then the program exits with a status of 1.

Abs - Absolute value

USAGE

```
int abs(i)
int i;
```

DESCRIPTION

ABS returns absolute value of its integer operand.

CAVEATS

You get what the hardware gives on the largest negative number.

Access - give file accessibility

USAGE

```
access(fname,perm)
char *name;
int perm;
```

DESCRIPTION

Access returns 0 if the access modes specified in "perm" are correct for the user to access "fname". -1 is returned if the file cannot be accessed.

The value for "perm" may be any legal OS-9 mode as used for "open()" or "creat()", it may be zero, which tests whether the file exists, or the path to it may be searched.

CAVEATS

NOTE that the "perm" value is NOT compatible with other systems.

DIAGNOSTICS

The appropriate error indication, if a value of -1 is returned, may be found in "errno".

Chain - load and execute a new program

USAGE

```
chain(modname,paramsize,paramptr,type,lang,datasize)
char *modname,*paramptr;
```

ASSEMBLER EQUIVALENT

```
os9  F$CHAIN
```

DESCRIPTION

The action of F$CHAIN is described fully in the OS-9 documentation. Chain implements the service request as described with one important exception: chain will NEVER return to the caller. If there is an error, the process will abort and return to its parent process. It might be wise, therefore, for the program to check the existence and access permissions of the module before calling chain. Permissions may be checked by using "modlink()" or "modload()" followed by an "munlink()".

"Modname" should point to the name of the desired module. "Paramsize" is the length of the parameter string (which should normally be terminated with a "\n"), and "paramptr" points to the parameter string. "Type" is the module type as found in the module header (normally 1: program), and "lang" should match the language nibble in the module header (C programs have 1 for 6809 machine code here). "Datasize" may be zero, or it may contain the number of 256 byte pages to give to the new process as initial allocation of data memory.

Chdir,Chxdir - change directory

USAGE

```
chdir(dirname)
char *dirname;

chxdir(dirname)
char *dirname;
```

ASSEMBLER EQUIVALENT

```
os9 I$CHGDIR
```

DESCRIPTION

These calls change the current data directory and the current execution directory, respectively, for the running task. "Dirname" is a pointer to a string that gives a pathname for a directory.

DIAGNOSTICS

Each call returns 0 after a successful call, or -1 if "dirname" is not a directory path name, or it is not searchable.

SEE ALSO

OS-9 shell commands "chd" and "chx".

Chmod - change access permissions of a file

USAGE

```
#include <modes.h>

chmod(fname,perm)
char *fname;
```

DESCRIPTION

Chmod changes the permission bits associated with a file. "Fname" must be a pointer to a file name, and "perm" should contain the desired bit pattern.

The allowable bit patterns are defined in the include file as follows:

```
          /* permissions */
               #define  S_IREAD   0x01     /* owner read */
               #define  S_IWRITE  0x02     /* owner write */
               #define  S_EXEC    0x04     /* owner execute */
               #define  S_IOREAD  0x08     /* public read */
               #define  S_IOWRITE 0x10     /* public write */
               #define  S_IOEXEC  0x20     /* public execute */
               #define  S_ISHARE  0x40     /* sharable */
               #define  S_IFDIR   0x80     /* directory */
```

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

DIAGNOSTICS

A successful call returns NULL(0). A -1 is returned if the caller is not entitled to change permissions or "fname" cannot be found.

SEE ALSO

OS-9 command "attr"

Chown - change the ownership of a file

USAGE

```
chown(fname,ownerid)
char *fname;
```

DESCRIPTION

This call is available only to the super user. "Fname" is a pointer to a file name, and "ownerid" is the new user-id.

DIAGNOSTICS

Zero is returned from a successful call. -1 is returned on error.

Close - close a file

USAGE

```
close(pn)
```

ASSEMBLER EQUIVALENT

```
os9  I$CLOSE
```

DESCRIPTION

Close takes a path number, "pn", as returned from system calls "open()", "creat()", or "dup()", and closes the associated file.

Termination of a task always closes all open files automatically, but it is necessary to close files where multiple files are opened by the task, and it is desired to re-use path numbers to avoid going over the system or process path number limit.

SEE ALSO

creat(),open(),dup().

Crc - compute a cyclic redundancy count

USAGE

```
crc(start,count,accum)
char *start,accum[3];
```

ASSEMBLER EQUIVALENT

```
os9  F$CRC
```

DESCRIPTION

This call accumulates a crc into a three byte array at "accum" for "count" bytes starting at "start". All three bytes of "accum" should be initialized to 0xff before the first call to "crc()". However, repeated calls can be subsequently made to cover an entire module. If the result is to be used as an OS-9 module crc, it should have its bytes complemented before insertion at the end of the module.

Creat - create a new file

USAGE

```
#include <modes.h>

creat(fname,perm)
char *fname;
```

ASSEMBLER EQUIVALENT

```
os9  I$CREATE
```

DESCRIPTION

Creat returns a path number to a new file available for writing, giving it the permissions specified in "perm" and making the task user the owner. If, however, "fname" is the name of an existing file, the file is truncated to zero length, and the ownership and permissions remain unchanged. NOTE, that unlike the OS-9 assembler service request, creat does not return an error if the file already exists. "Access()" should be used to establish the existence of a file if it is important that a file should not be overwritten.

It is unnecessary to specify writing permissions in "perm" in order to write to the file in the current task.

The permissions allowed are defined in the include file as follows:

```
              /* permissions */
#define  S_IPRM     0xff     /* mask for permission bits */
#define  S_IREAD    0x01     /* owner read */
#define  S_IWRITE   0x02     /* owner write */
#define  S_IEXEC    0x04     /* owner execute */
#define  S_IOREAD   0x08     /* public read */
#define  S_IOWRITE  0x10     /* public write */
#define  S_IOEXEC   0x20     /* public execute */
#define  S_ISHARE   0x40     /* sharable */
```

Directories may not be created with this call; use "mknod()" instead.

DIAGNOSTICS

This call returns -1 if there are too many files open. If the pathname cannot be searched, if permission to write is denied, or if the file exists and is a directory.

SEE ALSO

write(),close(),chmod()

Defdrive - get default system drive

USAGE

```
char *defdrive()
```

DESCRIPTION

A call to defdrive returns a pointer to a string containing the name of the default system drive. The method used is to consult the "Init" module for the default directory name. The name is copied to a static data area and a pointer to it is returned.

DIAGNOSTICS

-1 is returned if the "Init" module cannot be linked to.

Dup - duplicate an open path number

USAGE

```
dup(pn)
```

ASSEMBLER EQUIVALENT

```
os9  I$DUP
```

DESCRIPTION

Dup takes the path number, "pn", as returned from "open()" or "creat()" and returns another path number associated with the same file.

DIAGNOSTICS

A -1 is returned if the call fails because there are too many files open or the path number is invalid.

SEE ALSO

open(),creat(),close()

Exit,_Exit - task termination

USAGE

```
exit(status)

_exit(status)
```

ASSEMBLER EQUIVALENT

```
os9  F$EXIT
```

DESCRIPTION

Exit is the normal means of terminating a task. Exit does any cleaning up operations required before terminating, such as flushing out any file buffers (see Standard I/o), but _exit does not.

A task finishing normally, that is returning from "main()", is equivalent to a call - "exit(0)".

The status passed to exit is available to the parent task if it is executing a "wait".

SEE ALSO

wait()

Getpid - get the task id

USAGE

```
getpid()
```

ASSEMBLER EQUIVALENT

```
os9  F$ID
```

DESCRIPTION

A number unique to the current running task is often useful in creating names for temporary files. This call returns the task's system id (as returned to its parent by "os9fork").

SEE ALSO

,si 5
os9fork() Standard Library function mktemp.

Getstat - get file status

USAGE

```
#include <sgstat.h>
getstat(code,filenum,buffer)       /* code 0 */
char *buffer;

getstat(code,filenum)              /* codes 1 and 6 */

getstat(code,filenum,size)         /* code 2 */
long *size;

getstat(code,filenum,pos)          /* code 5 */
long *pos;
```

ASSEMBLER EQUIVALENT

```
os9 I$GETSTT
```

DESCRIPTION

A full description of getstat can be found in the OS-9 System Programmer's Manual.

"Code" must be the value of one of the standard codes for the getstat service request. "filenum" must be the path number of an open file.

The form of the call depends on the value of "code".

Code 0: "Buffer" must be the address of a 32 byte buffer into which the relevant status packet is copied. The header file has the definitions of the various file and device structures for use by the program.

Code 1: Code 1 only applies to SCF devices and to test for data available. The return value is zero if there is data available. -1 is returned if there is no data.

Code 2 "Size" should be the address of a long integer into which the current file size is placed. The return value of the function is -1 on error and 0 on success.

Code 5 "Pos" should be the address of a long integer into which the current file size is placed. The return value of the function is -1 on error and 0 on success.

Code 6 Returns -1 on EOF and error and 0 on success.

NOTE that when one of the previous calls returns -1, the actual error is returned in errno.

Getuid - return user id

USAGE

```
getuid()
```

ASSEMBLER EQUIVALENT

```
os9  F$ID
```

DESCRIPTION

Getuid returns the real user id of the current task (as maintained in the password file).

Intercept - set function for interrupt processing

USAGE

```
intercept(func)
int (*func)();       /* i.e. "func" is a pointer to a function
                        returning an int */
```

ASSEMBLER EQUIVALENT

F$ICPT

DESCRIPTION

Intercept instructs OS-9 to pass control to the function "func" when an interrupt(signal) is received by the current process.

If the interrupt processing function has an argument, it will contain the value of the signal received. On return from "func", the process resumes at the point in the program where it was interrupted by the signal. "Interrupt()" is an alternative to the use of "signal()" to process interrupts.

As an example, suppose we wish to ensure that a partially completed output file is deleted if an interrupt is received. The body of the program might include:

```
char *temp_file = "temp"; /* name of temporary file */
int pn = 0;               /* path number */
int intrupt();            /* predeclaration */

...

intercept(intrupt);       /* route interrupt processing */
pn = creat(temp_file,3);  /* make a new file */

...

write(pn,string,count)    /* write string to temp file */

...

close(pn);
pn=0;

...
```

The interrupt routine might be coded:

```
intrupt(sig);
{
     if (pn){ /* only done if pn refers to an open file */
          close(pn);
          unlink(temp_file); /* delete */
     }
exit(sig);
}
```

CAVEATS

"Intercept()" and "signal()" are mutually incompatible so that calls to both must not appear in the same program. The linker guards against this by giving an "entry name clash - _sigint" error if it is attempted.

SEE ALSO

signal()

Kill - send an interrupt to a task

USAGE

```
#include <signal.h>
kill(tid,interrupt)
```

DESCRIPTION

Kill sends the interrupt type "interrupt" to the task with id "tid".

Both tasks, sender and receiver, must have the same user id unless the user is the super user.

The include file contains definitions of the defined signals as follows:

```
       /* OS-9 signals */
#define        SIGKILL 0   /* system abort (cannot be caught or
                                       ignored)*/
#define        SIGWAKE 1    /* wake up */
#define        SIGQUIT 2    /* keyboard abort */
#define        SIGINT  3    /* keyboard interrupt */
```

Other user-defined signals may, of course, be sent.

DIAGNOSTICS

Kill returns 0 from a successful call and -1 if the task does not exist, the effective user ids do not match, or the user is not the system manager.

SEE ALSO

signal() OS-9 shell command "kill"

Lseek - position in file

USAGE

```
long lseek(pn,position,type)
long position;
```

ASSEMBLER EQUIVALENT

```
os9  I$SEEK
```

DESCRIPTION

The read or write pointer for the open file with the path number, "pn", is positioned by lseek to the specified place in the file. The "type" indicates from where "position" is to be measured: if 0, from the beginning of the file, if 1, from the current location, or if 2, from the end of the file.

Seeking to a location beyond the end of a file open for writing and then writing to it, creates a "hole" in the file which appears to be filled with zeros from the previous end to the position sought.

The returned value is the resulting position in the file unless there is an error, so to find out the current position use

```
lseek(pn,0l,1);
```

CAVEATS

The argument "position" MUST be a long integer. Constants should be explicitly made long by appending an "l", as above, and other types should be converted using a cast;

```
e.g.  lseek(pn,(long)pos,1);
```

Notice also, that the return value from lseek is itself a long integer.

DIAGNOSTICS

-1 is returned if "pn" is a bad path number, or attempting to seek to a position before the beginning of a file.

SEE ALSO

open(),creat() Standard Library function "fseek"

Mknod - create a directory

USAGE

```
#include <modes.h>

mknod(fname,desc)
char *fname;
```

ASSEMBLER EQUIVALENT

```
os9  I$MAKDIR
```

DESCRIPTION

This call may be used to create a new directory. "Fname" should point to a string containing the desired name of the directory. "Desc" is a descriptor specifying the desired mode (file type) and permissions of the new file.

The include file defines the possible values for "desc" as follows:

```
#define  S_IREAD   0x01      /* owner read */
#define  S_IWRITE  0x02      /* owner write */
#define  S_IEXEC   0x04      /* owner execute */
#define  S_IOREAD  0x08      /* public read */
#define  S_IOWRITE 0x10      /* public write */
#define  S_IOEXEC  0x20      /* public execute */
#define  S_ISHARE  0x40      /* sharable */
```

DIAGNOSTICS

Zero is returned if the directory has been successfully made; -1 if the file already exists.

SEE ALSO

OS-9 command "makdir"

Modload, Modlink - return a pointer to a module structure

USAGE

```
#include <module.h>
mod_exec *modlink(modname,type,language)
char *modname;

mod_exec *modload(filename,type,language)
char *filename;
```

ASSEMBLER EQUIVALENT

```
os9  F$LINK

os9  F$LOAD
```

DESCRIPTION

Each of these calls return a pointer to an OS-9 memory module.

Modlink will search the module directory for a module with the same name as "modname" and, if found, increment its link count.

Modload will open the file which has the path list specified by "filename" and loads modules from the file adding them to the module directory. The returned value is a pointer to the first module loaded.

Above, each is shown as returning a pointer to an executable module, but it will return a pointer to whatever type of module is found.

DIAGNOSTICS

-1 is returned on error.

SEE ALSO

munlink()

Munlink - unlink a module

USAGE

```
#include <module.h>
munlink(mod)
mod_exec *mod;
```

ASSEMBLER EQUIVALENT

```
os9  F$UNLINK
```

DESCRIPTION

This call informs the system that the module pointed to by "mod" is no longer required by the current process. Its link count is decremented, and the module is removed from the module directory if the link count reaches zero.

SEE ALSO

modlink(),modload()

_os9 - system call interface from C programs

USAGE

```
#include <os9.h>

_os9(code,reg)
char code;
struct registers *reg;
```

DESCRIPTION

_os9 enables a programmer to access virtually any OS-9 system call directly from a C program without having to resort to assembly language routines.

Code is one of the codes that are defined in os9.h. os9.h contains codes for the F$ and I$ function/service requests, and it also contains getstt, setstt, and error codes.

The input registers(reg) for the system calls are accessed by the following structure that is defined in os9.h:

```
struct registers {
     char rg_cc,rg_a,rg_b,rg_dup;
     unsigned rg_x,rg_y,rg_u;
};
```

An example program that uses _os9 is presented on the following page.

DIAGNOSTICS

-1 is returned if the OS-9 call failed. 0 is returned on success.

Program example:

```
#include <os9.h>
#include <modes.h>

/* this program does an I$GETSTT call to get file size */
main(argc,argv)
int argc;
char **argv;
{
     struct registers reg;
     int path;

/* tell linker we need longs */
     pflinit();

/* low level open(file name is first command line param */
     path=open(*++argv,S_IREAD);

/* set up regs for call to OS-9 */
     reg.rg_a=path;
     reg.rg_b=SS_SIZE;

     if(_os9(I_GETSTT,&reg) == 0)
          printf("filesize = %1x\n", /* success */
          (long) (reg.rg_x << 16)+reg.rg_u);
     else printf("OS9 error #%d\n",reg.rg_b & 0xff); /*failed*/

     dumpregs(&reg);  /* take a look at the registers */
}

dumpregs(r)
register struct registers *r;
{
     printf("cc=%02x\n",r->rg_cc & 0xff);
     printf(" a=%02x\n",r->rg_a & 0xff);
     printf(" b=%02x\n",r->rg_b & 0xff);
     printf("dp=%02x\n",r->rg_dp & 0xff);
     printf(" x=%04x\n",r->rg_x);
     printf(" y=%04x\n",r->rg_y);
     printf(" u=%04x\n",r->rg_u);
}
```

Open - open a file for read/write access

USAGE

```
open(fname,mode)
char *fname;
```

ASSEMBLER EQUIVALENT

```
os9 I$OPEN
```

DESCRIPTION

This call opens an existing file for reading if "mode" is 1, writing if "mode" is 2, or reading and writing if "mode" is 3. NOTE that these values are OS-9 specific and not compatible with other systems. "Fname" should point to a string representing the pathname of the file.

Open returns an integer as "path number" which should be used by i/o system calls referring to the file.

The position where reads or writes start is at the beginning of the file.

DIAGNOSTICS

-1 is returned if the file does not exist, if the pathname cannot be searched, if too many files are already open, or if the file permissions deny the requested mode.

SEE ALSO

Creat(),read(),write(),dup(),close()

Os9fork - create a process

USAGE

```
os9fork(modname,paramsize,paramptr,type,lang,datasize)
char *modname,*paramptr;
```

ASSEMBLER EQUIVALENT

```
os9 F$FORK
```

DESCRIPTION

The action of F$FORK is described fully in the OS-9 System Programmer's Manual. Os9fork will create a process that will run concurrently with the calling process. When the forked process terminates, it will return to the calling process.

"Modname" should point to the name of the desired module. "Paramsize" is the length of the parameter string which should normally be terminated with a '\n', and "paramptr" points to the parameter string. "Type" is the module type as found in the header(normally 1: program), and "lang" should match the language nibble in the module header(C programs have 1 for 6809 machine code here). "Datasize" may be zero, or it may contain the number of 256 byte pages to give to the new process as initial allocation of memory.

DIAGNOSTICS

-1 will be returned on error, or the ID number of the child process will be returned on success.

Pause - halt and wait for interrupt

USAGE

```
pause()
```

ASSEMBLER EQUIVALENT

```
os9  I$SLEEP         with a value of 0
```

DESCRIPTION

Pause may be used to halt a task until an interrupt is received from "kill".

Pause always returns -1.

SEE ALSO

kill(),signal() OS-9 shell command "kill"

Prerr - print error message

USAGE

```
prerr(filnum,errcode)
```

ASSEMBLER EQUIVALENT

```
os9 F$PERR
```

DESCRIPTION

PRERR prints an error message on the output path as specified by "filnum" which must be the path number of an open file. The message depends on "errcode" which will normally be a standard OS-9 error code.

Read,Readln - read from a file

USAGE

```
read(pn,buffer,count)
char *buffer;

readln(pn,buffer,count)
char *buffer;
```

ASSEMBLER EQUIVALENT

```
os9  I$READ

os9  I$READLN
```

DESCRIPTION

The path number, "pn", is an integer which is one of the standard path numbers 0, 1, or 2, or the path number should have been returned by a successful call to "open", "creat", or "dup". "Buffer" is a pointer to space with at least "count" bytes of memory into which read will put the data from the file.

It is guaranteed that at most "count" bytes will be read, but often less will be, either because, for readln, the file represents a terminal and input stops at the end of a line, or for both, end-of-file has been reached.

Readln causes "line-editing" such as echoing to take place and returns once the first "\n" is encountered in the input or the number of bytes requested has been read. Readln is the preferred call for reading from the user's terminal.

Read does not cause any such editing. See the OS-9 manual for a fuller description of the actions of these calls.

DIAGNOSTICS

Read and readln return the number of bytes actually read (0 at end-of-file) or -1 for physical i/o errors, a bad path number, or a ridiculous "count".

NOTE that end-of-file is not considered an error, and no error indication is returned. Zero is returned on EOF.

SEE ALSO

open(),creat(),dup()

Sbrk,Ibrk - request additional working memory

USAGE

```
char *sbrk(increase)
char *ibrk(increase)
```

DESCRIPTION

Sbrk requests an allocation from free memory and returns a pointer to its base.

"Sbrk()" requests the system to allocate "new" memory from outside the initial allocation.

Users should read the Memory Management section of this manual for a fuller explanation of the arrangement.

Ibrk requests memory from inside the initial memory allocation.

DIAGNOSTICS

Sbrk and ibrk return -1 if the requested amount of contiguous memory is unavailable.

Setpr - set process priority

USAGE

```
setpr(pid,priority)
```

ASSEMBLER EQUIVALENT

```
os9 F$SPRIOR
```

DESCRIPTION

SETPR sets the process identified by "pid"(process id) to have a priority of "priority". The lowest priority is 0 and the highest is 255.

DIAGNOSTICS

The call will return -1 if the process does not have the same user id as the caller.

Setime,Getime - Set and get system time

USAGE

```
#include <time.h>
setime(byffer)
getime(buffer)
struct sgtbuf *buffer   /* defined in time.h */
```

ASSEMBLER EQUIVALENT

```
os9 F$STIME
os9 G$GTIME
```

DESCRIPTION

GETIME returns system time in buffer.
SETIME sets system time from buffer.

Setuid - set user id

USAGE

```
setuid(uid)
```

ASSEMBLER EQUIVALENT

```
os9 F$SUSER
```

DESCRIPTION

This call may be used to set the user id for the current task. Setuid only works if the caller is the super user(user id 0).

DIAGNOSTICS

Zero is returned from a successful call, and -1 is returned on error.

SEE ALSO

getuid()

Setstat - set file status

USAGE

```
#include <sgstat.h>
setstat(code,filenum,buffer)     /* code 0 */
char *buffer;

setstat(code,filenum,size)       /* code 2 */
long size;
```

ASSEMBLER EQUIVALENT

```
os9 F$SETSTT
```

DESCRIPTION

For a detailed explanation of this call, see the OS-9 System Programmer's Manual.

"Filenum" must be the path number of a currently open file. The only values for code at this time are 0 and 2. When "code" is 0, "buffer" should be the address of a 32 byte structure which is written to the option section of the path descriptor of the file. The header file contains definitions of various structures maintained by OS-9 for use by the programmer. When code is 2, "size" should be a long integer specifying the new file size.

Signal - catch or ignore interrupts

USAGE

```
#include <signal.h>

(*signal(interrupt,address))()
(*address)();
```

(Which means:"signal" returns a pointer to a function, "address" is a pointer to a function.)

DESCRIPTION

This call is a comprehensive method of catching or ignoring signals sent to the current process. Notice that "kill()" does the sending of signals, and "signal()" does the catching.

Normally, a signal sent to a process causes it to terminate with the status of the signal. If, in advance of the anticipated signal, this system call is used, the program has the choice of ignoring the signal or designating a function to be executed when it is received. Different functions may be designated for different signals.

The values for "address" have the following meanings:

0 = reset to the default i.e. abort when received
1 = ignore; this will apply until reset to another value
Otherwise: taken to be the address of a C function which is to be executed on receipt of the signal.

If the latter case is chosen, when the signal is received by the process the "address" is reset to 0, the default, before the function is executed. This means that if the next signal received should be caught then another call to "signal()" should be made immediately. This is normally the first action taken by the "interrupt" function. The function may access the signal number which caused its execution by looking at its argument. On completion of this function the program resumes at the point at which it was "interrupted" by the signal.

An example should help to clarify all this. Suppose a program needs to create a temporary file which should be deleted before exiting. The body of the program might contain fragments like this:

```
pn  =  creat("temp",3);        /*  create  a temporary file */
signal(2,intrupt);             /*   ensure   tidying   up   */
signal(3,intrupt);
write(pn,string,count);        /*  write  to temporary file */
```

```
close(pn);                        /* finished writing */
unlink("temp");                   /* delete it */
exit(0);                           /* normal exit */
```

The call to "signal()" will ensure that if a keyboard or quit signal is received then the function "intrupt()" will be executed and this might be written:

```
intrupt(sig)
{
close(pn);                    /* close it if open */
unlink("temp");               /* delete it */
exit(sig);                    /* received signal as exit
                                 status*/
}
```

In this case, as the function will be exiting before another signal is received, it is unnecessary to call "signal()" again to reset its pointer. Note that either the function "intrupt()" should appear in the source code before the call to "signal()", or it should be pre-declared.

The signals used by OS-9 are defined in the header file as follows:

```
/* OS-9 signals */
#define        SIGKILL 0   /* system abort (cannot be caught or
                                  ignored)*/
#define        SIGWAKE 1     /* wake up */
#define        SIGQUIT 2     /* keyboard abort */
#define        SIGINT  3     /* keyboard interrupt */

/* special addresses */
#define        SIG_DFL 0     /* reset to default */
#define        SIG_IGN 1     /* ignore */
```

Please note that there is another method of trapping signals, namely "intercept()" (q.v.). However, since "signal()" and "intercept()" are mutually incompatible, calls to both of them must not appear in the same program. The link-loader will prevent the creation of an executable program in which both are called by aborting with an "entry name clash" error for "_sigint".

SEE ALSO

intercept() OS-9 shell command "kill" kill()

Stacksize, Freemem - obtain stack reservation size

USAGE

```
stacksize()
freemem()
```

DESCRIPTION

For a description of the meaning and use of this call, the user is referred to the Memory Management section of this manual.

If the stack check code is in effect, a call to stacksize will return the maximum number of bytes of stack used at the time of the call. This call can be used to determine the stack size required by a program.

Freemem() will return the number of bytes of the stack that has not been used.

SEE ALSO

ibrk(),sbrk(),freemem() Global variable "memend" and value "end".

Strass - byte by byte copy

USAGE

```
_strass(s1,s2,count)
char *s1,*s2;
```

DESCRIPTION

Until such time as the compiler can deal with structure assignment, this function is useful for copying one structure to another.

"Count" bytes are copied from memory location at "s2" to memory at "s1" regardless of the contents.

Tsleep - put process to sleep

USAGE

```
tsleep(ticks)
```

ASSEMBLER EQUIVALENT

```
os9  F$SLEEP
```

DESCRIPTION

Tsleep deactivates the calling process for a specified number of system "ticks" or indefinitely if "ticks" is zero. A tick is system dependent but is usually 100ms.

For a fuller description of this call, see the OS-9 System Programmer's Manual.

Unlink - remove directory entry

USAGE

```
unlink(fname)
```

ASSEMBLER EQUIVALENT

```
os9  I$DELETE
```

DESCRIPTION

Unlink deletes the directory entry whose name is pointed to by "fname". If the entry was the last link to the file, the file itself is deleted and the disc space occupied made available for re-use. If, however, the file is open, in any active task, the deletion of the actual file is delayed until the file is closed.

ERRORS

Zero is returned from a successful call, -1 if the file does not exist, if its directory is write-protected, or cannot be searched, if the file is a non-empty directory or a device.

SEE ALSO

OS-9 command "kill" link()

Wait - wait for task termination

USAGE

```
wait(status) int *status;

wait(0)
```

ASSEMBLER EQUIVALENT

```
os9  F$WAIT
```

DESCRIPTION

Wait is used to halt the current task until a child task has terminated.

The call returns the task id of the terminating task and places the status of that task in the integer pointed to by "status" unless "status" is 0. A wait must be executed for each child task spawned.

The status will contain the argument of the "exit" or "_exit" call in the child task or the signal number if it was interrupted. A normally terminating C program with no call to "exit" or "_exit" has an implied call of "exit(0)".

CAVEATS

NOTE that the status is the OS-9 status code and is not compatible with codes on other systems.

DIAGNOSTICS

-1 is returned if there is no child to be waited for.

SEE ALSO

fork(),signal(),exit(),_exit()

Write,Writeln - write to a file or device

USAGE

```
write(pn,buffer,count) char *buffer;

writeln(pn,buffer,count) char *buffer;
```

ASSEMBLER EQUIVALENT

```
os9  I$WRITE

os9  I$WRITLN
```

DESCRIPTION

"Pn" must be a value returned by "open", "creat" or "dup" or should be a 0(stdin), 1(stdout), or 2(stderr).

"Buffer" should point to an area of memory from which "count" bytes are to be written. Write returns the actual number of bytes written, and if this is different from "count", an error has occurred.

Writes in multiples of 256 bytes to file offset boundaries of 256 bytes are the most efficient.

Write causes no "line-editing" to occur on output. Writeln causes line-editing and only writes up to the first "\n" in the buffer if this is found before "count" is exhausted. For a full description of the actions of these calls the, reader is referred to the OS-9 documentation.

DIAGNOSTICS

-1 is returned if "pn" is a bad path number, if "count" is ridiculous or on physical i/o error.

SEE ALSO

creat(),open()