I$Attach — Attach a New Device to the System

ASM CALL: **OS9 I$Attach**

INPUT: **d0.b = Access mode (Read_, Write_, Updat_)**
(a0) = Device name pointer

OUTPUT: **(a2) = Address of the device table entry**

ERROR OUTPUT: **cc = Carry bit set**
d1.w = Appropriate error code

FUNCTION: I$Attach causes an I/O device to become known to the system. You use it to attach a new device to the system, or to verify that it is already attached.

The device's name string is used to search the system module directory to see if a device descriptor module with the same name is in memory (this is the name by which the device is known). The descriptor module contains the name of the device's file manager, device driver, and other related information.

If the descriptor is found and the device is not already attached, OS-9 links to its file manager and device driver. It then places their addresses in a new device table entry. Any permanent storage needed by the device driver is allocated, and the driver's initialization routine is called to initialize the hardware. If the device has already been attached, it is not re-initialized.

The access mode parameter may be used to verify that subsequent read and/or write operations are permitted. An Attach system call is not required to perform routine I/O. It does not reserve the device in question; I$Attach simply prepares it for subsequent use by any process.

The kernel attaches all devices at open, and detaches them at close.

NOTE: Attach and Detach for devices are similar to Link and Unlink for modules; they are usually used together. However, system performance can improve slightly if all devices are attached at startup. This increments each device's use count and prevents the device from being re-initialized every time it is opened. This also has the advantage of allocating the static storage for devices all at once, which minimizes memory fragmentation. If this is done, the device driver termination routine is never executed.

SEE ALSO: I$Detach

POSSIBLE ERRORS: E$DevOvf, E$BMode, E$DevBsy, and E$MemFul.

I$ChgDir — Change Working Directory

ASM CALL: **OS9 I$ChgDir**

INPUT: **d0.b = Access mode (read/write/exec)**
(a0) = Address of the pathlist

OUTPUT: **(a0) = Updated past pathname**

ERROR OUTPUT: **cc = Carry bit set**
d1.w = Appropriate error code

FUNCTION: ChgDir changes a process's working directory to another directory file specified by the pathlist. Depending on the access mode given, either the execution or the data directory (or both) may change. The file specified must be a directory file, and the caller must have access permission for the specified mode.

ACCESS MODES:
1 = Read
2 = Write
3 = Update (read and write)
4 = Execute

If the access mode is read, write, or update, the current data directory changes. If the access mode is execute, the current execution directory changes. Both can change simultaneously.

NOTE: The shell CHD directive uses UPDATE mode, which means you must have both read and write permission to change directories from the shell. This is a recommended practice.

POSSIBLE ERRORS: E$BPNam and E$BMode.

I$Close — Close a Path to a File/Device

ASM CALL: **OS9 I$Close**

INPUT: **d0.w = Path number**

OUTPUT: **None**

ERROR OUTPUT: **cc = Carry bit set**
d1.w = Appropriate error code

FUNCTION: I$Close terminates the I/O path specified by the path. The path number is no longer valid for any OS-9 calls unless it becomes active again through an Open, Create, or Dup system call. When pathlists to non-sharable devices are closed, the devices become available to other requesting processes. If this is the last use of the path (that is, it has not been inherited or duplicated by I$Dup), all OS-9 internally managed buffers and descriptors are deallocated.

NOTE: The OS-9 F$Exit service request automatically closes any open paths. By convention, standard I/O paths are not closed unless it is necessary to change the files/devices they correspond to.

SEE ALSO: I$Detach

CAVEATS: I$Close does an implied I$Detach call. If this causes the device use count to become zero, the device termination routine is executed.

POSSIBLE ERRORS: E$BPNum

I$Create — Create a Path to New File

ASM CALL: **OS9 I$Create**

INPUT: **d0.b = Access mode (S, I, E, W, R)**
d1.w = File attributes (access permission)
d2.l = Initial allocation size (optional)
(a0) = Pathname pointer

OUTPUT: **d0.w = Path number**
(a0) = Updated past the pathlist

ERROR OUTPUT: **cc = Carry bit set**
d1.w = Appropriate error code

FUNCTION: I$Create creates a new file. On multi-file devices, the new file name is entered in the directory structure, and Create is synonymous with Open.

The access mode parameter passed in register d0.b must have the write bit set if any data is to be written to the file. The file is given the attributes passed in the register d1.w. The individual bits are defined as follows:

Mode Bits (d0.w)	Attribute Bits (d1.w)
0 = read	**0 = owner read permit**
1 = write	**1 = owner write permit**
2 = execute	**2 = owner execute permit**
5 = initial file size	**3 = public read permit**
6 = single user	**4 = public write permit**
	5 = public execute permit
	6 = non-sharable file

If the execute bit (bit 2) of the access mode byte is set, directory searching begins with the working execution directory instead of the working data directory.

The path number returned by OS-9 identifies the file in subsequent I/O service requests until the file is closed.

WRITE automatically allocates file space for the file. The SETSTAT call (SS_Size) explicitly allocates file space. If the size bit (bit 5) is set, an initial file size estimate may be passed in d2.l.

An error occurs if the pathlist specifies a file name that already exists. You cannot use I$Create to make directory files (see I$MakDir).

Create causes an implicit I$Attach call. If the device has not previously been attached, the device's initialization routine is executed.

SEE ALSO: I$Attach, I$Open, I$Close, and I$MakDir.

CAVEATS: The caller is made the owner of the file. To maintain compatibility with OS-9/6809 disk formats, there is only space for two bytes of owner ID. The LS byte of the user's group and the LS byte of the user's ID are used as the owner ID. All user's with the same group ID may access the file as the owner.

If an initial file size is specified with I$Create, the exact amount specified may not be allocated. You must execute a SS_Size SetStat after creating the file to ensure that sufficient space was allocated.

POSSIBLE ERRORS: E$PthFul and E$BPNam.

I$Delete — Delete a File

ASM CALL: **OS9 I$Delete**

INPUT: **d0.b = Access mode (read/write/exec)**
(a0) = Pathname pointer

OUTPUT: **(a0) = Updated past pathlist**

ERROR OUTPUT: **cc = Carry bit set**
d1.w = Appropriate error code

FUNCTION: I$Delete deletes the file specified by the pathlist. The caller must have non-sharable write access to the file (the file may not already be open) or an error results. An attempt to delete a non-multifile device results in an error.

The access mode is used to specify the data or execution directory (but not both) in the absence of a full pathlist. If the access mode is read, write, or update, the current data directory is assumed. If the execute bit is set, the current execution directory is assumed. Note that if a full pathlist is specified, that is, a pathlist beginning with a slash (/), the access mode is ignored.

SEE ALSO: I$Detach, I$Attach, I$Create, and I$Open.

POSSIBLE ERRORS: E$BPNam

I$Detach — Remove a Device from the System

ASM CALL: **OS9 I$Detach**

INPUT: **(a2) = Address of the device table entry**

OUTPUT: **None**

ERROR OUTPUT: **cc = Carry bit set**
d1.w = Appropriate error code

FUNCTION: I$Detach removes a device from the system device table, if not in use by any other process. If this is the last use of the device, the device driver's termination routine is called, and any permanent storage assigned to the driver is de-allocated. The device driver and file manager modules associated with the device are unlinked and may be lost if not in use by another process. It is crucial for the termination routine to remove the device from the IRQ system.

You must use the I$Detach service request to un-attach devices that were attached with the I$Attach service request. Both of these are used mainly by the kernel and are of limited use to the typical user. SCF also uses Attach/Detach to set up its second (echo) device.

Most devices are attached at startup and remain attached. Seldom used devices can be attached to the system and used for a while, then detached to free system resources when no longer needed.

SEE ALSO: I$Attach and I$Close.

CAVEATS: If an invalid address is passed in (a2), the system may crash or undergo severe damage.

I$Dup — Duplicate a Path

ASM CALL: **OS9 I$Dup**

INPUT: **d0.w = Path number of path to duplicate**

OUTPUT: **d0.w = New number for the same path**

ERROR OUTPUT: **cc = Carry bit set**
d1.w = Appropriate error code

FUNCTION: Given the number of an existing path, I$Dup returns a synonymous path number for the same file or device. I$Dup always uses the lowest available path number. For example, if you do I$Close on path #0, then do I$Dup on path #4, path #0 is returned as the new path number. In this way, the standard I/O paths may be manipulated to contain any desired paths.

The shell uses this service request when it redirects I/O. Service requests using either the old or new path numbers operate on the same file or device.

CAVEATS: This only increments the use count of a path descriptor and returns a synonymous path number. The path descriptor is NOT copied. It is usually not a good idea for more than one process to be doing I/O on the same path concurrently. On RBF files, unpredictable results may occur.

POSSIBLE ERRORS: E$PthFul and E$BPNum.

I$GetStt — Get File/Device Status

ASM CALL: **OS9 I$GetStt**

INPUT: **d0.w = Path number**
d1.w = Function code
Others = dependent on function code

OUTPUT: **Dependent on function code**

ERROR OUTPUT: **cc = Carry bit set**
d1.w = Appropriate error code

FUNCTION: This is a wild card call used to handle individual device parameters that are not uniform on all devices, or are highly hardware dependent. The exact operation of this call depends on the device driver and file manager associated with the path.

A typical use is to determine a terminal's parameters (echo on/off, delete character, etc.). It is commonly used in conjunction with the SetStt call, which sets the device operating parameters.

The mnemonics for the status codes are found in the relocatable library sys.l or usr.l. Codes 0-127 are reserved for Microware use. The remaining codes and their parameter passing conventions are user definable (see the OS-9 Technical Overview section on device drivers in Chapter 3). Presently defined function codes are listed below.

POSSIBLE ERRORS: E$BPNum

I$GetStt FUNCTION CODES:

SS_DevNum — Return Device Name (ALL)

INPUT: **d0.w = Path number**
d1.w = #SS_DevNm function code
(a0) = Address of 32 byte area for device name

OUTPUT: **Device name in 32 byte storage area, null terminated**

SS_EOF — Test for End of File (RBF, SCF, PIPE)

INPUT: **d0.w = Path number**
d1.w = #SS_EOF function code

OUTPUT: **d1.l = 0 If not EOF, (SCF never returns EOF)**

ERROR **cc = Carry bit set**
OUTPUT: **d1.w = Appropriate error code (E$EOF, if end of file)**

SS_CDFD — Return File Descriptor (CDFM)

INPUT: **d0.w = Path number**
d1.w = #SS_CDFD function code
d2.w = Number of bytes to copy
(a0) = Pointer to buffer area for file descriptor

OUTPUT: **None**

ERROR **cc = Carry bit set**
OUTPUT: **d1.w = Appropriate error code**

FUNCTION: SS_CDFD reads the file descriptor describing the path number. The file descriptor may be read for information purposes only, as there are no user changeable parameters.

SS_FD — Read File Descriptor Sector (RBF, PIPE)

INPUT: **d0.w = Path number**
d1.w = #SS_FD function code
d2.w = Number of bytes to copy(<=logical sector size of media)
(a0) = Address of buffer area for FD

OUTPUT: **File descriptor copied into buffer**

FUNCTION: Use SS_FD to inspect the file descriptor information (for example, FD_OWN and FD_DAT) and the file segment list.

SS_FDInf — Get Specified File Descriptor Sector (RBF)

INPUT: **d0.w = Path number**
d1.w = #SS_FDInf function code
d2.w = Number of bytes to copy (<=256)
d3.l = FD sector address
(a0) = Address of buffer area for FD

OUTPUT: **File descriptor copied into buffer**

NOTE: If SS_FDInf is called in user state, the caller must be a super-group user. If it is called in system state, the caller does *not* have to be a super-group user.

SS_Free — Return Amount of Free Space on Device (NRF, NVRAM file mgr.)

INPUT: **d0.l = Path number**
d1.w = #SS_Free function code

OUTPUT: **d0.l = Size of free space on device, in bytes**

SS_Opt — Read PD_OPT: The Path Descriptor Option Section. (All)

INPUT: **d0.w = Path number**
d1.w = #SS_Opt function code
(a0) = Address to put a 128 byte status packet

OUTPUT: **Status packet copied to buffer**

ERROR OUTPUT: **cc = Carry bit set**
d1.w = Appropriate error code

FUNCTION: SS_Opt reads the option section of the path descriptor and copies it into the 128 byte area pointed to by (a0). It is typically used to determine the current settings for echo, auto line feed, etc. For a complete description of the status packet, refer to Chapter 3 of the ***OS-9 Technical Overview***, the section on file manager path descriptors.

SS_Pos — Get Current File Position (RBF, PIPE)

INPUT: **d0.w = Path number**
d1.w = #SS_Pos function code

OUTPUT: **d2.l = Current file position**

ERROR **cc = Carry bit set**
OUTPUT: **d1.w = Appropriate error code**

SS_Ready — Test for Data Ready (RBF, SCF, PIPE)

INPUT: **d0.w = Path number**
d1.w = #SS_Ready function code

OUTPUT: **d1.l = Number of input characters available on SCF or pipe devices.**
RBF devices always return carry clear, d1.l=1

ERROR **cc = Carry bit set**
OUTPUT: **d1.w = Appropriate error code (E$NotRdy if no data is available)**

SS_Size — Return Current File Size (RBF, PIPE)

INPUT: **d0.w = Path number**
d1.w = #SS_Size function code

OUTPUT: **d2.l = Current file size**

ERROR **cc = Carry bit set**
OUTPUT: **d1.w = Appropriate error code**

SS_VarSect Query Support for Variable Logical Sector Sizes (RBF)

INPUT: **d0.w = path number**
d1.w - #SS_VarSect function code

***OUTPUT*:** **none**

FUNCTION: SS_VarSect is an internal call between RBF and a driver. If the driver does not return an error, the logical sector size of the media is specified in PD_SSize. If the driver returns an error, and the error is E$UnkSvc, RBF sets the path's logical sector size to 256 bytes and ignores PD_SSize. If any other error is returned, the path open is aborted and the error is returned to the caller.

I$MakDir — Make a New Directory

ASM CALL: **OS9 I$MakDir**

INPUT: **d0.b = Access mode (see below)**
d1.w = Access permissions
d2.l = Initial Allocation Size (Optional)
(a0) = Pathname pointer

OUTPUT: **(a0) = Updated past pathname**

ERROR OUTPUT: **cc = Carry bit set**
d1.w = Appropriate error code

FUNCTION: I$MakDir is the only way to create a new directory file. It creates and initializes a new directory as specified by the pathlist. The new directory file contains no entries, except for an entry for itself (specified by a dot (.)) and its parent directory (specified by double dot (..)). Makdir fails on non-multi-file devices. If the execution bit is set, OS-9 begins searching for the file in the working execution directory (unless the pathlist begins with a slash).

The caller is made the owner of the directory. MakDir does not return a path number because directory files are not opened by this request (use I$Open to do so). The new directory automatically has its directory bit set in the access permission attributes. The remaining attributes are specified by the bytes passed in register d1.w which have individual bits defined as listed below (if the bit is set, access is permitted):

Mode Bits (d0.b)
0 = read
1 = write
2 = execute
5 = initial directory size
7 = directory

Attribute Bits (d1.w)
0 = owner read permit
1 = owner write permit
2 = owner execute permit
3 = public read permit
4 = public write permit
5 = public execute permit
6 = non-sharable file
7 = directory

POSSIBLE ERRORS: E$BPNam and E$CEF.

I$Open — Open a Path to a File or Device

ASM CALL: **OS9 I$Open**

INPUT: **d0.b = Access mode (D S E W R)**
(a0) = Pathname pointer

OUTPUT: **d0.w = Path number**
(a0) = Updated past pathname

ERROR OUTPUT: **cc = Carry bit set**
d1.w = Appropriate error code

FUNCTION: I$Open opens a path to an existing file or device as specified by the pathlist. A path number is returned which is used in subsequent service requests to identify the path. If the file does not exist, an error is returned.

The access mode parameter specifies which subsequent read and/or write operations are permitted as follows (if the bit is set, access is permitted):

Mode Bits
0 = read
1 = write
2 = execute
6 = open file for non sharable use
7 = open directory file

NOTE: A non-directory file may be opened with no bits set. This allows you to examine the attributes, size, etc. with the GetStt system call, but does not permit any actual I/O on the path.

For RBF devices, use read mode instead of update if the file is not going to be modified. This inhibits record locking, and can dramatically improve system performance if more than one user is accessing the file. The access mode must conform to the access permissions associated with the file or device (see I$Create).

If the execution bit mode is set, OS-9 begins searching for the file in the working execution directory (unless the pathlist begins with a slash).

If the single user bit is set, the file is opened for non-sharable access even if the file is sharable.

Files can be opened by several processes (users) simultaneously. Devices have an attribute that specifies whether or not they are sharable on an individual basis.

Open always uses the lowest path number available for the process.

SEE ALSO: I$Attach, I$Create and I$Close.

CAVEATS: Directory files may be opened only if the Directory bit (bit 7) is set in the access mode.

POSSIBLE ERRORS: E$PthFul, E$BPNam, E$Bmode, E$FNA, E$PNNF, and E$Share.

I$Read — Read Data from a File or Device

ASM CALL: **OS9 I$Read**

INPUT: **d0.w = Path number**
d1.l = Maximum number of bytes to read
(a0) = Address of input buffer

OUTPUT: **d1.l = Number of bytes actually read**

ERROR OUTPUT: **cc = Carry bit set**
d1.w = Appropriate error code

FUNCTION: I$Read reads a specified number of bytes from the specified path number. The path must previously have been opened in read or update mode. The data is returned exactly as read from the file/device, without additional processing or editing such as backspace, line delete, etc. If there is not enough data in the file to satisfy the read request, fewer bytes are read than requested, but an end of file error is not returned.

After all data in a file has been read, the next I$Read service request returns an end of file error.

SEE ALSO: I$ReadLn

CAVEATS: The keyboard X-ON/X-OFF characters may be filtered out of the input data on SCF-type devices unless the corresponding entries in the path descriptor are set to zero. You may wish to modify the device descriptor so that these values in the path descriptor are initialized to zero when the path is opened. SCF devices usually terminate the read when a carriage return is reached.

For RBF devices, if the file is open for update, the record read is locked out. See the Record Locking section in the RBF chapter of the ***OS-9 Technical I/O Manual***.

The number of bytes requested is read unless:

- The end-of-file is reached
- An end-of-record occurs (SCF only)
- An error condition occurs

POSSIBLE ERRORS: E$BPNum, E$Read, E$BMode, and E$EOF.

I$ReadLn — Read a Text Line with Editing

ASM CALL: **OS9 I$ReadLn**

INPUT: **d0.w = Path number**
d1.l = Maximum number of bytes to read
(a0) = Address of input buffer

OUTPUT: **d1.l = Actual number of bytes read**

ERROR OUTPUT: **cc = Carry bit set**
d1.w = Appropriate error code

FUNCTION: ReadLn is similar to Read except it reads data from the input file or device until an end-of-line character is encountered. ReadLn also causes line editing to occur on SCF-type devices. Line editing refers to backspace, line delete, echo, automatic line feed, etc. Some devices (SCF) may limit the number of bytes that may be read with one call.

SCF requires that the last byte entered be an end-of-record character (normally carriage return). If more data is entered than the maximum specified, it is not accepted and a PD_OVF character (normally bell) is echoed. For example, a ReadLn of exactly one byte accepts only a carriage return to return without error and beeps when other keys are pressed.

After all data in a file has been read, the next I$ReadLn service request returns an end of file error.

SEE ALSO: I$Read

POSSIBLE ERRORS: E$BPNum, E$Read, and E$BMode.

I$Seek — Reposition the Logical File Pointer

ASM CALL: **OS9 I$Seek**

INPUT: **d0.w = Path number**
d1.l = New position

OUTPUT: **None**

ERROR OUTPUT: **cc = Carry bit set**
d1.w = Appropriate error code

FUNCTION: I$Seek repositions the path's file pointer which is the 32-bit address of the next byte in the file to be read or written. I$Seek usually does not initiate any physical positioning of the media.

You can perform a Seek to any value even if the file is not large enough. Subsequent writes automatically expand the file to the required size (if possible), but reads return an end-of-file condition. **NOTE:** A Seek to address zero is the same as a rewind operation.

Seeks to non-random access devices are usually ignored and return without error.

CAVEATS: On RBF devices, seeking to a new disk sector causes the internal sector buffer to be rewritten to disk if it has been modified. Seek does not change the state of record locks. Beware of seeking to a negative position. RBF takes negatives as large positive numbers.

POSSIBLE ERRORS: E$BPNum

I$SetStt Set File/Device Status

ASM CALL: **OS9 I$SetStt**

INPUT: **d0.w = Path number**
d1.w = Function code
Others = Function code dependent

OUTPUT: **Function code dependent**

ERROR OUTPUT: **cc = Carry bit set**
d1.w = Appropriate error code

FUNCTION: This is a "wild card" system call used to handle individual device parameters that are not uniform on all devices or are highly hardware dependent. The exact operation of this call depends on the device driver and file manager associated with the path.

A typical use is to set a terminal's parameters for backspace character, delete character, echo on/off, null padding, paging, etc. It is commonly used in conjunction with the GetStt service request which reads the device's operating parameters.

The mnemonics for the status codes are found in the relocatable library sys.l or usr.l. Codes 0-127 are reserved for Microware use. The remaining codes and their parameter passing conventions are user definable (see the OS-9 Technical Overview section on device drivers in Chapter 3). Presently defined function codes are listed below.

POSSIBLE ERRORS: E$BPNum

I$SETSTT FUNCTION CODES:

SS_Attr — Set the File Attributes (RBF, PIPE)

INPUT: **d0.w = Path number**
d1.w = #SS_Attr function code
d2.w = New attributes

OUTPUT: **none**

FUNCTION: SSAttr changes a file's attributes to the new value, if possible. It is not permitted to set the dir bit of a non-directory file, or to clear the dir bit of a non-empty directory.

SS_Close — Notifies Driver that a Path has been Closed (SCF, RBF, SBF)

INPUT: **d0.w = path number**
d1.w = SS_Close function code

OUTPUT: **none**

FUNCTION: SS_Close is an internal call for drivers.

SS_DCOff — Sends Signal when Data Carrier Detect Line Goes False (SCF)

INPUT: **d0.w = path number**
d1.w = SS_DCOff function code
d2.w = Signal code to be sent

OUTPUT: **none**

FUNCTION: When a modem has finished receiving data from a carrier, the Data Carrier Detect line goes false. SS_DCOff sends a signal code when this happens. SS_DCOn sends a signal when the line goes true.

SS_DCon — Sends Signal when Data Carrier Detect Line Goes True (SCF)

INPUT: **d0.w = path number**
d1.w = SS_DCOn function code
d2.w = Signal code to be sent

OUTPUT: **none**

FUNCTION: When a modem receives a carrier, the Data Carrier Detect line goes true. SS_DCOn sends a signal code when this happens. SS_DCOff sends a signal when the line goes false.

SS_DsRTS — Disables RTS Line (SCF)

INPUT: **d0.w = path number**
d1.w = SS_DsRTS function code

OUTPUT: **none**

FUNCTION: SS_DsRTS tells the driver to negate the RTS hardware handshake line.

SS_EnRTS — Enables RTS Line (SCF)

INPUT: **d0.w = path number**
d1.w = SS_EnRTS function code

OUTPUT: **none**

FUNCTION: SS_EnRTS tells the driver to negate the RTS hardware handshake line.

SS_Feed — Erase Tape (SBF)

INPUT: **d0.w = path number**
d1.w = SS_Feed function code
d2.l = # of blocks to erase

OUTPUT: **none**

FUNCTION: SS_Feed erases a portion of the tape. The amount of tape erased depends on the capabilities of the hardware used. SBF attempts to use the following: If -1 is passed in d2, SBF erases until the end-of-tape is reached. If d2 receives a positive parameter, SBF erases the amount of tape equivalent to that number of blocks. This depends on both the hardware used and the driver.

SS_FD — Write File Description Sector (RBF)

INPUT: **d0.w = Path Number**
d1.w = #SS_FD function code
(a0) = Address of FD sector image

OUTPUT: **none**

FUNCTION: SS_FD changes FD sector data. The path must be open for write.

NOTE: You can only change FD_OWN, FD_DAT, and FD_Creat. These are the only fields written back to disk. Only the super user can change the file's owner ID.

SS_FD should normally be used with GetStat (SS_FD) to read the FD before attempting to change FD sector data.

SS_Lock — Lock out a Record (RBF)

INPUT: **d0.w = Path Number**
d1.w = #SS_Lock function code
d2.l = Lockout size

OUTPUT: **none**

FUNCTION: SS_Lock locks out a section of the file from the current file pointer position up to the specified number of bytes.

If 0 bytes are requested, all locks are removed (Record Lock, EOF Lock, and File Lock).

If $FFFFFFFF bytes are requested, then the entire file is locked out regardless of where the file pointer is. This is a special type of file lock that remains in effect until released by SS_Lock(0), a read or write of zero bytes, or the file is closed.

There is no way to gain file lock using only read or write system calls.

SS_Open — Notifies Driver that a Path has been Opened

INPUT: **d0.w = path number**
d1.w = SS_Open function code

OUTPUT: **none**

FUNCTION: SS_Open is an internal call for drivers.

SS_Opt — Write Option Selection of Path Descriptor (ALL)

INPUT: **d0.w = Path number**
d1.w = #SS_Opt function code
(a0) = Address of a 128 byte status packet

OUTPUT: **none**

FUNCTION: SS_Opt writes the option section of the path descriptor from the 128 byte status packet pointed to by (a0). It is typically used to set the device operating parameters (echo, auto line feed, etc.). This call is handled by the file managers, and only copies values that are appropriate to be changed by user programs.

SS_Relea — Release Device (SCF, PIPE)

INPUT: **d0.w = path number**
d1.w = SS_Relea function code

OUTPUT: **none**

FUNCTION: SS_Relea releases the device from any SS_SSig, SS_DCOn, or SS_DCOff requests made by the calling process on this path.

SS_Reset — Restore Head to Track Zero (RBF, SBF)

INPUT: **d0.w = Path number**
d1.w = #SS_Reset function code

OUTPUT: **none**

FUNCTION: For RBF, this directs the disk head to track zero. It is used for formatting and for error recovery. For SBF, this rewinds the tape.

SS_RFM — Skip Tape Marks (SBF)

INPUT: **d0.w = path number**
d1.w = SS_RFM function code
d2.l = # of tape marks

OUTPUT: **none**

FUNCTION: SS_RFM skips the number of tape marks specified in d2. If d2 is negative, the tape is rewound the specified number of marks.

SS_Size — Set File Size (RBF, PIPE)

INPUT: **d0.w = Path number**
d1.w = #SS_Size function code
d2.l = Desired file size

OUTPUT: **none**

FUNCTION: SS_Size sets the file's size.

For pipe files, you can use SS_Size to reset the pipe path (d2.1=0), provided the pipe has no active readers or writers. Any other value in d2.1 is ignored.

SS_Skip — Skip Blocks (SBF)

INPUT: **d0.w = path number**
d1.w = SS_Skip function code
d2.l = # of blocks to skip

OUTPUT: **none**

FUNCTION: SS_Skip skips the number of blocks specified in d2. If the number is negative, the tape is rewound the specified number of blocks.

SS_SSig — Send Signal on Data Ready (SCF, PIPE)

INPUT: **d0.w = Path number**
d1.w = SS_SSig function code
d2.w = User defined signal code

OUTPUT: **none**

FUNCTION: SS_SSig sets up a signal to send to a process when an interactive device or pipe has data ready. SS_SSig must be reset each time the signal is sent. The device or pipe is considered busy and returns an error if any read request arrives before the signal is sent. Write requests to the device are allowed in this state.

SS_Ticks — Wait Specified Number of Ticks for Record Release (RBF)

INPUT: **d0.w = path number**
d1.w = #SS_Ticks function code
d2.l = Delay interval

OUTPUT: **none**

FUNCTION: Normally, if a read or write request is issued for a part of a file that is locked out by another user, RBF sleeps indefinitely until the conflict is removed.

You can use SS_Ticks to return an error (E$Lock) to the user program if the conflict still exists after the specified number of ticks have elapsed.

The delay interval is used directly as a parameter to RBF's conflict sleep request. The value zero (RBF's default) causes a sleep forever until the record is released. A value of one means 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 256th of a second into ticks before sleeping. This allows programmed delays to be independent of the system clock rate.

SS_WFM — Write Tape Marks (SBF)

INPUT: **d0.w = path number**
d1.w = SS_WFM function code
d2.l = # of tape marks

OUTPUT: **none**

FUNCTION: SS_WFM writes the number of tape marks specified in d2.

SS_WTrk — Write (format) track (RBF)

INPUT: **d0.w = Path number**
d1.w = #SS_WTrk function code
(a0) = Address of track buffer
For hard disks and "autosize" media, this table contains 1 logical sector of data (pattern $E5). For floppy disks, this table contains the track's physical data.
(a1) = Address of interleave table
This table contains byte entries of LSN's ordered to match the requested interleave offset. NOTE: This is a "logical" table and does not reflect the PD_SOffs base sector number.
d2 = Track number
d3.w = Side/density
The low order byte has 3 bits which can be set:
Bit 0 = SIDE (0=side zero;1=side one)
Bit 1 = DENSITY (0=single;1=double)
Bit 2 = TRACK DENSITY (0=single;1=double)
The high order byte contains the side number.
d4 = Interleave value

OUTPUT: **none**

FUNCTION: SS_Wtrk 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 side 0 of the first accessable track is specified.

I$Write — Write Data to a File or Device

ASM CALL: **OS9 I$Write**

INPUT: **d0.w = Path number**
d1.l = Number of bytes to write
(a0) = Address of buffer

OUTPUT: **d1.l = Number of bytes actually written**

ERROR OUTPUT: **cc = Carry bit set**
d1.w = Appropriate error code

FUNCTION: I$Write outputs bytes to a file or device associated with the path number specified. The path must have been opened or created in the write or update access modes.

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.

SEE ALSO: I$Open, I$Create, and I$WritLn.

CAVEATS: On RBF devices, any record that was locked is released.

POSSIBLE ERRORS: E$BPNum, E$BMode, and E$Write.

I$WritLn — Write a Line of Text with Editing

ASM CALL: **OS9 I$WritLn**

INPUT: **d0.w = Path number**
d1.l = Maximum number of bytes to write
(a0) = Address of buffer

OUTPUT: **d1.l = Actual number of bytes written**

ERROR OUTPUT: **cc = Carry bit set**
d1.w = Appropriate error code

FUNCTION: I$WriteLn is similar to Write except it writes data until a carriage return character or (d1) bytes are encountered. Line editing is also activated for character-oriented devices such as terminals, printers, etc. The line editing refers to auto line feed, null padding at end-of-line, etc.

The number of bytes actually written (returned in d1.l) does not reflect any additional bytes that may have been added by file managers or device drivers for device control. For example, if SCF appends a line feed and nulls after carriage return characters, these extra bytes are not counted.

SEE ALSO: I$Open, I$Create, and I$Write; ***OS-9 Technical I/O Manual*** chapter on SCF Drivers (line editing).

CAVEATS: On RBF devices, any record that was locked is released.

POSSIBLE ERRORS: E$BPNum, E$BMode, and E$Write.

End of Chapter 2

NOTES