ADosDanger

COLLABORATORS			
	TITLE : ADosDanger		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		August 13, 2022	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

ADosDanger

1.1 AmigaTalk to AmigaDOS Help:

```
                WARNING:  Improper usage of these Methods will (at the bare
          minimum), result in the Operating System hanging
          up, which could result in loss of data.  This is
          the least of what could happen!  Turn back now!

DANGEROUS AmigaDOS Functions/AmigaTalk Methods:


             writeFile

             setVBuf

             setFileSize

             setArgStr

             selectOutput

             selectInput

             seekFile

             runCommand

             replyPkt

             inhibit

             fWrite

             freeDosObject

             freeDosEntry

             freeDeviceProc

             freeArgs
```

```
exitProgram
 -- Avoid like the plague!

deviceProc

deleteVar

createNewProc

cliInitRun

cliInitNewcli

attemptLockDosList

allocDosObject

addDosEntry
   See Also,
 VERY DANGEROUS METHODS
```

1.2 VERY DANGEROUS AmigaDOS Methods:

```
      WARNING:  Improper usage of these Methods will (at the bare
minimum), result in the Operating System hanging
up, which could result in loss of data.  This is
the least of what could happen!  Turn back now!

VERY DANGEROUS AmigaDOS Functions/AmigaTalk Methods:


waitPkt

unLoadSeg

systemTagList

setFileSysTask

setConsoleTask

sendPkt

remSegment

remDosEntry

remAssignList

newLoadSeg

loadSeg

internalUnLoadSeg
```

```
internalLoadSeg

format
 -- Are you out of your tree??

doPacket

deleteFile

addSegment
```

1.3 writeFile (DANGEROUS):

```
                NAME
   Write -- Write bytes of data to a file

SYNOPSIS
   LONG returnedLength =  Write( BPTR file, void *buffer, LONG length );

FUNCTION
   Write writes bytes of data to the opened file 'file'. 'length'
   indicates the length of data to be transferred; 'buffer' is a
   pointer to the buffer. The value returned is the length of
   information actually written. So, when 'length' is greater than
   zero, the value of 'length' is the number of characters written.
   Errors are indicated by a value of -1.

   Note:  This is an unbuffered routine (the request is passed directly
   to the filesystem.)  Buffered I/O is more efficient for small
   reads and writes; see FPutC.

INPUTS
   file   - BCPL pointer to a file handle
   buffer - pointer to the buffer
   length - integer

RESULT
   returnedLength - integer

SEE ALSO
    Read  ,
              Seek
             ,
    Open  ,  Close ,
    FPutC

AMIGATALK INTERFACE (DangerousDOS Class):

writeFile: bptrFileHandle with: aBuffer ofSize: length

WARNING:  Make sure that aBuffer is a String of length bytes!
```

1.4 waitPkt (VERY DANGEROUS):

```
                NAME
    WaitPkt -- Waits for a packet to arrive at your pr_MsgPort

SYNOPSIS
    struct DosPacket *packet = WaitPkt( void );


FUNCTION
    Waits for a packet to arrive at your pr_MsgPort.  If anyone has
    installed a packet wait function in pr_PktWait, it will be called.
    The message will be automatically GetMsg()ed so that it is no longer
    on the port.  It assumes the message is a dos packet.  It is NOT
    guaranteed to clear the signal for the port.

RESULT
    packet - the packet that arrived at the port (from ln_Name of message).

SEE ALSO

             SendPkt
            ,
             DoPkt
            ,  AbortPkt

AMIGATALK INTERFACE (VeryDangerousDOS Class):

waitForPacket
```

1.5 unLoadSeg (VERY DANGEROUS):

```
                NAME
    UnLoadSeg -- Unload a seglist previously loaded by LoadSeg

SYNOPSIS
    void UnLoadSeg( BPTR seglist );

FUNCTION
    Unload a seglist loaded by LoadSeg.  'seglist' may be zero.
    Overlaid segments will have all needed cleanup done, including
    closing files.

INPUTS
    seglist - BCPL pointer to a segment identifier

SEE ALSO

             LoadSeg
            ,
             InternalLoadSeg
            ,
```

```
               InternalUnLoadSeg

AMIGATALK INTERFACE (VeryDangerousDOS Class):

unLoadSegment: bptrSegList
```

1.6 systemTagList (VERY DANGEROUS):

```
                NAME
    SystemTagList -- Have a shell execute a command line

SYNOPSIS
    LONG error = SystemTagList( char *command, struct TagItem *tags );


FUNCTION
    Similar to Execute(), but does not read commands from the input
    filehandle.  Spawns a Shell process to execute the command, and
    returns the returncode the command produced, or -1 if the command
    could not be run for any reason.  The input and output filehandles
    will not be closed by System, you must close them (if needed) after
    System returns, if you specified them via SYS_Input or SYS_Output.

 By default the new process will use your current Input() and Output()
 filehandles.  Normal Shell command-line parsing will be done
 including redirection on 'command'.  The current directory and path
 will be inherited from your process.  Your path will be used to find
 the command (if no path is specified).

 Note that you may NOT pass the same filehandle for both SYS_Input
 and SYS_Output.  If you want input and output to both be to the same
 CON: window, pass a SYS_Input of a filehandle on the CON: window,
 and pass a SYS_Output of NULL.  The shell will automatically set
 the default Output() stream to the window you passed via SYS_Input,
 by opening "*" on that handler.

 If used with the SYS_Asynch flag, it WILL close both it's input and
 output filehandles after running the command (even if these were
 your Input() and Output()!)

 Normally uses the boot (ROM) shell, but other shells can be specified
 via SYS_UserShell and SYS_CustomShell.  Normally, you should send
 things written by the user to the UserShell.  The UserShell defaults
 to the same shell as the boot shell.

 The tags are passed through to CreateNewProc() (tags that conflict
 with SystemTagList() will be filtered out).  This allows setting
 things like priority, etc for the new process.  The tags that are
 currently filtered out are:

    NP_Seglist,     NP_FreeSeglist, NP_Entry
    NP_Input,       NP_Output,      NP_CloseInput
    NP_CloseOutput, NP_HomeDir,     NP_Cli
```

```
 INPUTS
    command - Program and arguments
    tags    - see <dos/dostags.h>.  Note that both SystemTagList()-
              specific tags and tags from CreateNewProc() may be passed.

 RESULT
 error - 0 for success, result from command, or -1.  Note that on
         error, the caller is responsible for any filehandles or other
 things passed in via tags.  -1 will only be returned if
 dos could not create the new shell.  If the command is not
 found the shell will return an error value, normally RETURN_ERROR.


SEE ALSO
    Execute ,
              CreateNewProc
             ,
    Input   ,  Output ,
   <dos/dostags.h>

AMIGATALK INTERFACE (VeryDangerousDOS Class):

systemCommandTagList: commandString: tags: tagArray
```

1.7 setVBuf (DANGEROUS):

```
                NAME
    SetVBuf -- set buffering modes and size

SYNOPSIS
    LONG error = SetVBuf( BPTR fh, char *buff, LONG type, LONG size );

FUNCTION
    Changes the buffering modes and buffer size for a filehandle.
    With buff == NULL, the current buffer will be deallocated and a
    new one of (approximately) size will be allocated.  If buffer is
    non-NULL, it will be used for buffering and must be at least
    max( size, 208 ) bytes long, and MUST be longword aligned.  If size
    is -1, then only the buffering mode will be changed.

Note that a user-supplied buffer will not be freed if it is later
replaced by another SetVBuf() call, nor will it be freed if the
filehandle is closed.

Has no effect on the buffersize of filehandles that were not created
by
              AllocDosObject()
             .

INPUTS
    fh   - Filehandle
    buff - buffer pointer for buffered I/O or NULL.  MUST be LONG-aligned!
    type - buffering mode (see <dos/stdio.h>)
    size - size of buffer for buffered I/O (sizes less than 208 bytes
```

```
           will be rounded up to 208), or -1.

RESULT
    error - 0 if successful.  NOTE: opposite of most dos functions!

    NOTE:  fails if someone has replaced the buffer without using SetVBuf()
           - RunCommand() does this.  Remember to check error before
           freeing user-supplied buffers!

BUGS
    Not implemented until after V39.  From V36 up to V39, always
    returned 0.

SEE ALSO
     FputC  ,  FGetC ,
     UnGetC ,  Flush ,
     FRead  ,
              FWrite
             ,
     FGets  ,  FPuts ,

              AllocDosObject

AMIGATALK INTERFACE (DangerousDOS Class):

setVBuf: bptrFileHandle to: aBuffer type: t bufferSize: size
```

1.8 setFileSysTask (VERY DANGEROUS):

```
NAME
    SetFileSysTask -- Sets the default filesystem for the process

SYNOPSIS
    struct MsgPort *oldport = SetFileSysTask( struct MsgPort *port );

FUNCTION
    Sets the default filesystem task's port (pr_FileSystemTask) for the
    current process.

INPUTS
    port - The pr_MsgPort of the default filesystem for the process

RESULT
    oldport - The previous FileSysTask value

SEE ALSO
     GetFileSysTask ,  Open

AMIGATALK INTERFACE (VeryDangerousDOS Class):

setFileSystemTask: msgPort
```

1.9 setFileSize (DANGEROUS):

```
                NAME
    SetFileSize -- Sets the size of a file

SYNOPSIS
    LONG newsize = SetFileSize( BPTR fh, LONG offset, LONG mode );

FUNCTION
    Changes the file size, truncating or extending as needed.  Not all
    handlers may support this; be careful and check the return code.  If
 the file is extended, no values should be assumed for the new bytes.
 If the new position would be before the filehandle's current position
 in the file, the filehandle will end with a position at the
 end-of-file.  If there are other filehandles open onto the file, the
 new size will not leave any filehandle pointing past the end-of-file.
 You can check for this by looking at the new size (which would be
 different than what you requested).

 The seek position should not be changed unless the file is made
 smaller than the current seek position.  However, see BUGS.

 Do NOT count on any specific values to be in any extended area.

 INPUTS
    fh     - File to be truncated/extended.
    offset - Offset from position determined by mode.
    mode   - One of OFFSET_BEGINNING, OFFSET_CURRENT, or OFFSET_END.

 RESULT
    newsize - position of new end-of-file or -1 for error.

 BUGS
    The RAM: filesystem and the normal Amiga filesystem act differently
    in where the file position is left after SetFileSize().  RAM: leaves
 you at the new end of the file (incorrectly), while the Amiga ROM
 filesystem leaves the seek position alone, unless the new position
 is less than the current position, in which case you're left at the
 new EOF.

 The best workaround is to not make any assumptions about the seek
 position after SetFileSize().

 SEE ALSO

              Seek

AMIGATALK INTERFACE (DangerousDOS Class):

setFileSize: bptrFileHandle at: offset mode: mode
```

1.10 setConsoleTask (VERY DANGEROUS):

```
NAME
    SetConsoleTask -- Sets the default console for the process

SYNOPSIS
    struct MsgPort *oldport = SetConsoleTask( struct MsgPort *port );

FUNCTION
    Sets the default console task's port (pr_ConsoleTask) for the
    current process.

INPUTS
    port - The pr_MsgPort of the default console handler for the process

RESULT
    oldport - The previous ConsoleTask value.

SEE ALSO
     GetConsoleTask ,  Open

AMIGATALK INTERFACE (VeryDangerousDOS Class):

setConsoleTask: msgPort
```

1.11 setArgStr (DANGEROUS):

```
                NAME
    SetArgStr -- Sets the arguments for the current process

SYNOPSIS
    BOOL success = SetArgStr( char *ptr );

FUNCTION
    Sets the arguments for the current program.  The ptr MUST be reset
    to it's original value before process exit.  So save the original
    ptr BEFORE calling this funcion!

INPUTS
    ptr - pointer to new argument string.

RESULT
    success (DOSTRUE) or failure (FALSE).

SEE ALSO
     GetArgStr ,
              RunCommand

AMIGATALK INTERFACE (DangerousDOS Class):

setArgumentString: argString
```

1.12 sendPkt (VERY DANGEROUS):

```
                NAME
    SendPkt -- Sends a packet to a handler

SYNOPSIS
    void SendPkt( struct DosPacket *packet,
                  struct MsgPort   *port,
                  struct MsgPort   *replyport );

FUNCTION
    Sends a packet to a handler and does not wait.  All fields in the
    packet must be initialized before calling this routine.  The packet
    will be returned to replyport.  If you wish to use this with

              WaitPkt()
             , use the address of your pr_MsgPort for replyport.

INPUTS
    packet    - packet to send, must be initialized and have a message.
    port      - pr_MsgPort of handler process to send to.
    replyport - MsgPort for the packet to come back to.

NOTES
    Callable from a task.

SEE ALSO

              DoPkt
             ,
              WaitPkt
             ,

              AllocDosObject
             ,
              FreeDosObject
             ,
     AbortPkt

AMIGATALK INTERFACE (VeryDangerousDOS Class):

sendPacket: dosPacket to: msgPort replyTo: replyPort
```

1.13 selectOutput (DANGEROUS):

```
                NAME
    SelectOutput -- Select a filehandle as the default output channel

SYNOPSIS
    BPTR old_fh = SelectOutput( BPTR fh );

FUNCTION
    Set the current output as the default output for the process.
```

```
    This changes the value returned by Output().  old_fh should
    be closed or saved as needed.

INPUTS
    fh - Newly desired output handle

RESULT
    old_fh - Previous current output

SEE ALSO
     Output ,
              SelectInput
             ,
     Input

AMIGATALK INTERFACE (DangerousDOS Class):

selectOutput: bptrFileHandle
```

1.14 selectInput (DANGEROUS):

```
                NAME
    SelectInput -- Select a filehandle as the default input channel

SYNOPSIS
    BPTR old_fh = SelectInput( BPTR fh );

FUNCTION
    Set the current input as the default input for the process.
    This changes the value returned by Input().  old_fh should
    be closed or saved as needed.

INPUTS
    fh     - Newly default input handle

RESULT
    old_fh - Previous default input filehandle

SEE ALSO
     Input  ,
              SelectOutput
             ,
     Output

AMIGATALK INTERFACE (DangerousDOS Class):

selectInput: bptrFileHandle
```

1.15 seekFile (DANGEROUS):

```
                NAME
    Seek -- Set the current position for reading and writing

SYNOPSIS
    LONG oldPosition = Seek( BPTR file, LONG position, LONG mode );

FUNCTION
    Seek sets the read/write cursor for the file 'file' to the
    position 'position'. This position is used by both Read() and
 Write() as a place to start reading or writing. The result is the
 current absolute position in the file, or -1 if an error occurs, in
 which case IoErr() can be used to find more information. 'mode' can
 be OFFSET_BEGINNING, OFFSET_CURRENT or OFFSET_END. It is used to
 specify the relative start position. For example, 20 from current
 is a position 20 bytes forward from current, -20 is 20 bytes back
 from current.

 So that to find out where you are, seek zero from current. The end
 of the file is a Seek() positioned by zero from end. You cannot
 Seek() beyond the end of a file.

INPUTS
    file     - BCPL pointer to a file handle
    position - integer
    mode     - integer

RESULT
    oldPosition - integer

BUGS
    The V36 and V37 ROM filesystem (and V36/V37 l:fastfilesystem)
    returns the current position instead of -1 on an error.  It sets
IoErr() to 0 on success, and an error code on an error.  This bug
was fixed in the V39 filesystem.

SEE ALSO
     Read ,
              Write
             ,

              SetFileSize

AMIGATALK INTERFACE (DangerousDOS Class):

seek: bptrFileHandle to: position mode: mode
```

1.16 runCommand (DANGEROUS):

```
                NAME
    RunCommand -- Runs a program using the current process

SYNOPSIS
    LONG rc = RunCommand( BPTR seglist, ULONG stacksize,
```

```
                          char *argptr, ULONG argsize );

FUNCTION
    Runs a command on your process/cli.  Seglist may be any language,
    including BCPL programs.  Stacksize is in bytes.  argptr is a null-
terminated string, argsize is its length.  Returns the returncode the
program exited with in d0. Returns -1 if the stack couldn't be
allocated.

NOTE:  The argument string MUST be terminated with a newline to work
properly with ReadArgs() and other argument parsers.

RunCommand also takes care of setting up the current input filehandle
in such a way that ReadArgs() can be used in the program, and restores
the state of the buffering before returning.  It also sets the value
returned by GetArgStr(), and restores it before returning.  NOTE:
the setting of the argument string in the filehandle was added in V37.

It's usually appropriate to set the command name (via
 SetProgramName() ) before calling RunCommand().  RunCommand() sets
the value returned by  GetArgStr()  while the command is running.

INPUTS
    seglist   - Seglist of command to run.
    stacksize - Number of bytes to allocate for stack space
    argptr    - Pointer to argument command string.
    argsize   - Number of bytes in argument command.

RESULT
    rc        - Return code from executed command. -1 indicates failure

SEE ALSO

              CreateNewProc
             ,
              SystemTagList
             ,
     Execute ,  GetArgStr ,
     SetProgramName ,  ReadArgs

AMIGATALK INTERFACE (DangerousDOS Class):

runCommand: bptrSegmentList args: argString count: argSize stack: stackSize
```

1.17 replyPkt (DANGEROUS):

```
                NAME
    ReplyPkt -- replies a packet to the person who sent it to you

SYNOPSIS
    void ReplyPkt( struct DosPacket *packet, LONG result1, LONG result2 );

FUNCTION
    This returns a packet to the process which sent it to you.  In
```

```
    addition, puts your pr_MsgPort address in dp_Port, so using ReplyPkt()
again will send the message to you.  (This is used in "ping-ponging"
packets between two processes).  It uses result 1 & 2 to set the
dp_Res1 and dp_Res2 fields of the packet.

INPUTS
    packet  - packet to reply, assumed to set up correctly.
    result1 - first result
    result2 - secondary result

SEE ALSO

              DoPkt
             ,
              SendPkt
             ,

              WaitPkt
             ,  IoErr

AMIGATALK INTERFACE (DangerousDOS Class):

replyPacket: dosPacketObject primaryResult: result1 secondaryResult: result2
```

1.18 remSegment (VERY DANGEROUS):

```
                NAME
    RemSegment - Removes a resident segment from the resident list

SYNOPSIS
    BOOL success = RemSegment( struct Segment *segment );

FUNCTION
    Removes a resident segment from the Dos resident segment list,
    unloads it, and does any other cleanup required.  Will only succeed
    if the seg_UC (usecount) is 0.

INPUTS
    segment - the segment to be removed

SEE ALSO
     FindSegment ,
              AddSegment

AMIGATALK INTERFACE (VeryDangerousDOS Class):

removeSegment: segmentObject
```

1.19 remDosEntry (VERY DANGEROUS):

```
                NAME
    RemDosEntry -- Removes a Dos List entry from it's list

SYNOPSIS
    BOOL success = RemDosEntry( struct DosList *dlist );

FUNCTION
    This removes an entry from the Dos Device list.  The memory associated
    with the entry is NOT freed.  NOTE: you must have locked the Dos List
with the appropriate flags before calling this routine.  Handler
writers should see the AddDosEntry() caveats about locking and use
a similar workaround to avoid deadlocks.

INPUTS
    dlist   - Device list entry to be removed.

SEE ALSO

              AddDosEntry
             ,  FindDosEntry ,
     NextDosEntry ,  LockDosList ,
     MakeDosEntry ,
              FreeDosEntry

AMIGATALK INTERFACE (VeryDangerousDOS Class):

removeDosEntry: dosList
```

1.20 remAssignList (VERY DANGEROUS):

```
NAME
    RemAssignList -- Remove an entry from a multi-dir assign

SYNOPSIS
    BOOL success = RemAssignList( char *name, BPTR lock );

FUNCTION
    Removes an entry from a multi-directory assign.  The entry removed is
    the first one for which SameLock with 'lock' returns that they are on
    the same object.  The lock for the entry in the list is unlocked (not
    the entry passed in).

INPUTS
    name - Name of device to remove lock from (without trailing ':')
    lock - Lock associated with the object to remove from the list

BUGS
    In V36 through V39.23 dos, it would fail to remove the first lock
    in the assign.  Fixed in V39.24 dos (after the V39.106 kickstart).

SEE ALSO
     Lock ,  AssignLock ,
     AssignPath ,  AssignLate ,
```

```
     DupLock ,  AssignAdd ,
     UnLock

AMIGATALK INTERFACE (VeryDangerousDOS Class):

removeAssignList: assignmentName from: bptrLock
```

1.21 newLoadSeg (VERY DANGEROUS):

```
                NAME
    NewLoadSeg -- Improved version of LoadSeg for stacksizes

SYNOPSIS
    BPTR seglist = NewLoadSeg( char *file, struct TagItem *tags );

FUNCTION
    Does a LoadSeg on a file, and takes additional actions based on the
    tags supplied.

Clears unused portions of Code and Data hunks (as well as BSS hunks).
(This also applies to InternalLoadSeg() and LoadSeg()).

NOTE to overlay users:  NewLoadSeg() does NOT return seglist in
both D0 and D1, as
             LoadSeg
             does.  The current ovs.asm uses LoadSeg(),
and assumes returns are in D1.  We will support this for LoadSeg() ONLY.

INPUTS
    file - Filename of file to load
    tags - pointer to tagitem array

RESULT
    seglist - Seglist loaded, or NULL

BUGS
    No tags are currently defined.

SEE ALSO

             LoadSeg
            ,
             UnLoadSeg
            ,

             InternalLoadSeg
            ,
             InternalUnLoadSeg

AMIGATALK INTERFACE (VeryDangerousDOS Class):

newLoadSegment: fileName tags: tagArray
```

1.22 loadSeg (VERY DANGEROUS):

```
                NAME
    LoadSeg -- Scatterload a loadable file into memory

SYNOPSIS
    BPTR seglist = LoadSeg( char *name )

FUNCTION
    The file 'name' should be a load module produced by the linker.
    LoadSeg() scatterloads the CODE, DATA and BSS segments into memory,
chaining together the segments with BPTR's on their first words.
The end of the chain is indicated by a zero.  There can be any number
of segments in a file.  All necessary re-location is handled by
LoadSeg().

In the event of an error any blocks loaded will be unloaded and a
NULL result returned.

If the module is correctly loaded then the output will be a pointer
at the beginning of the list of blocks. Loaded code is unloaded via
a call to UnLoadSeg().

INPUTS
    name - pointer to a null-terminated string

RESULT
    seglist - BCPL pointer to a seglist

SEE ALSO

              UnLoadSeg
             ,
              InternalLoadSeg
             ,

              InternalUnLoadSeg
             ,
              CreateProc
             ,

              CreateNewProc
             ,
              NewLoadSeg
             .

AMIGATALK INTERFACE (VeryDangerousDOS Class):

loadSegment: segmentName
```

1.23 internalUnLoadSeg (VERY DANGEROUS):

```
                NAME
```

```
    InternalUnLoadSeg -- Unloads a seglist loaded with InternalLoadSeg()

SYNOPSIS
    BOOL success = InternalUnLoadSeg( BPTR seglist,
                                      void (*FreeFunc)( char *, ULONG )
                                    );

FUNCTION
    Unloads a seglist using freefunc to free segments.  Freefunc is called
    as for InternalLoadSeg.  NOTE:  Will call Close() for overlaid
    seglists.

INPUTS
    seglist  - Seglist to be unloaded
    FreeFunc - Function called to free memory

RESULT
    success - returns whether everything went OK (since this may close
              files).  Also returns FALSE if seglist was NULL.

BUGS
    Really should use tags

SEE ALSO

              LoadSeg
             ,
              UnLoadSeg
             ,

              InternalLoadSeg
             ,
              NewLoadSeg
             ,
     Close

AMIGATALK INTERFACE (VeryDangerousDOS Class):

internalUnLoadSegment: bptrSegList freeFuncPtr: freeFunc
```

1.24 internalLoadSeg (VERY DANGEROUS):

```
                NAME
    InternalLoadSeg -- Low-level load routine

SYNOPSIS
    BPTR seglist = InternalLoadSeg( BPTR  fh,
                                    BPTR  table,
                                    LONG *functionarray,
                                    LONG *stack
                                  );

FUNCTION
    Loads from fh.  Table is used when loading an overlay, otherwise
```

```
    should be NULL.  Functionarray is a pointer to an array of functions.
Note that the current Seek position after loading may be at any point
after the last hunk loaded.  The filehandle will not be closed.  If a
stacksize is encoded in the file, the size will be stuffed in the
LONG pointed to by stack.  This LONG should be initialized to your
default value: InternalLoadSeg() will not change it if no stacksize
is found. Clears unused portions of Code and Data hunks (as well as
BSS hunks).  (This also applies to LoadSeg() and NewLoadSeg()).

If the file being loaded is an overlaid file, this will return
-(seglist).  All other results will be positive.

NOTE to overlay users:  InternalLoadSeg() does NOT return seglist in
both D0 and D1, as LoadSeg does.  The current ovs.asm uses LoadSeg(),
and assumes returns are in D1.  We will support this for LoadSeg()
ONLY.

INPUTS
    fh         - Filehandle to load from.
    table          - When loading an overlay, otherwise ignored.
    functionarray - Array of function to be used for read, alloc, and free.

      FuncTable[0]->Actual = ReadFunc( readhandle, buffer, length ), DOSBase
                     D0                 D1          D2      D3        A6

      FuncTable[1]->Memory = AllocFunc( size, flags ), Execbase
                     D0                  D0    D1       A6

      FuncTable[2]->FreeFunc( memory, size ), Execbase
                              A1      D0      A6

    stack - Pointer to storage (ULONG) for stacksize.

RESULT
    seglist - Seglist loaded or NULL.  NOT returned in D1!

BUGS
    Really should use tags.

SEE ALSO

              LoadSeg
             ,
              UnLoadSeg
             ,

              NewLoadSeg
             ,
              InternalUnLoadSeg

AMIGATALK INTERFACE (VeryDangerousDOS Class):

internalLoadSegment: bptrFileHandle ovlyTable: bptrTable
          funcArray: fArray          stackPtr: stack
```

1.25 inhibit (DANGEROUS):

```
NAME
    Inhibit -- Inhibits access to a filesystem

SYNOPSIS
    BOOL success = Inhibit( char *filesystem, LONG flag );

FUNCTION
    Sends an ACTION_INHIBIT packet to the indicated handler.  This stops
    all activity by the handler until uninhibited.  When uninhibited,
    anything may have happened to the disk in the drive, or there may no
    longer be one.

INPUTS
    filesystem - Name of device to inhibit (with ':')
    flag       - New status.  DOSTRUE = inhibited,
                                FALSE = uninhibited

AMIGATALK INTERFACE (DangerousDOS Class):

inhibit: fileSystem flags: flag
```

1.26 fWrite (DANGEROUS):

```
NAME
    FWrite -- Writes a number of blocks to an output (buffered)

SYNOPSIS
    LONG count = FWrite( BPTR fh, char *buf, ULONG blocklen, ULONG blocks )

FUNCTION
    Attempts to write a number of blocks, each blocklen long, from the
    specified buffer to the output stream.  May return less than the
    number of blocks requested, if there is some error such as a full
    disk or r/w error.  This call is buffered.

INPUTS
    fh       - filehandle to use for buffered I/O
    buf      - Area to write bytes from.
    blocklen - number of bytes per block.  Must be > 0.
    blocks   - number of blocks to write.  Must be > 0.

RESULT
    count - Number of _blocks_ written.  On an error, the number of
            blocks actually written is returned.

BUGS
    Doesn't clear IoErr() before starting.  If you want to find out
    about errors, use SetIoErr( 0 ) before calling.

SEE ALSO
     FPutC ,  FRead ,
     FPuts
```

```
AMIGATALK INTERFACE (DangerousDOS Class):

fileWrite: bptrFileHandle to: aBuffer blkSize: blockLength
    count: blockCount
```

1.27 freeDosObject (DANGEROUS):

```
                NAME
    FreeDosObject -- Frees an object allocated by
              AllocDosObject()
                SYNOPSIS
    void FreeDosObject( ULONG type, void *ptr );

FUNCTION
    Frees an object allocated by AllocDosObject().  Do NOT call for
    objects allocated in any other way.

INPUTS
    type - type passed to AllocDosObject()
    ptr  - ptr returned by AllocDosObject()

BUGS
    Before V39, DOS_CLI objects will only have the struct
    CommandLineInterface freed, not the strings it points to.  This
    is fixed in V39 dos.  Before V39, you can workaround this bug by
    using FreeVec() on cli_SetName, cli_CommandFile, cli_CommandName,
    and cli_Prompt, and then setting them all to NULL.  In V39 or
    above, do NOT use the workaround.

SEE ALSO

                AllocDosObject
               , FreeVec, <dos/dos.h>

AMIGATALK INTERFACE (DangerousDOS Class):

freeDosObject: dosObject type: t  " Tested "
```

1.28 freeDosEntry (DANGEROUS):

```
                NAME
    FreeDosEntry -- Frees an entry created by  MakeDosEntry

SYNOPSIS
    void FreeDosEntry( struct DosList *dlist );

FUNCTION
    Frees an entry created by MakeDosEntry().  This routine should be
    eliminated and replaced by a value passed to FreeDosObject()!
```

```
INPUTS
    dlist - DosList to free.

SEE ALSO

              AddDosEntry
             ,
              RemDosEntry
             ,
     FindDosEntry ,  LockDosList ,
     NextDosEntry ,  MakeDosEntry

AMIGATALK INTERFACE (DangerousDOS Class):

freeDosEntry: dosListObject
```

1.29 freeDeviceProc (DANGEROUS):

```
                NAME
    FreeDeviceProc -- Releases port returned by  GetDeviceProc()

SYNOPSIS
    void FreeDeviceProc( struct DevProc *devproc );

FUNCTION
    Frees up the structure created by GetDeviceProc(), and any associated
    temporary locks.

Decrements the counter incremented by GetDeviceProc().  The counter
is in an extension to the 1.3 process structure.  After calling
FreeDeviceProc(), do not use the port or lock again!  It is safe to
call FreeDeviceProc(NULL).

INPUTS
    devproc - A value returned by GetDeviceProc()

BUGS
    Counter not currently active in 2.0.

SEE ALSO
     GetDeviceProc ,
              DeviceProc
             ,
     AssignLock    ,  AssignLate ,
     AssignPath

AMIGATALK INTERFACE (DangerousDOS Class):

freeDeviceProcess: devProcessObject
```

1.30 freeArgs (DANGEROUS):

```
NAME
    FreeArgs - Free allocated memory after  ReadArgs()

SYNOPSIS
    void FreeArgs( struct RDArgs *rdargs );

FUNCTION
    Frees memory allocated to return arguments in from ReadArgs().  If
    ReadArgs allocated the RDArgs structure it will be freed.  If NULL
    is passed in this function does nothing.

INPUTS
    rdargs - structure returned from ReadArgs() or NULL.

SEE ALSO
     ReadArgs ,  ReadItem ,
     FindArg

AMIGATALK INTERFACE (DangerousDOS Class):

freeArgs: rdArgsObject
```

1.31 format (VERY DANGEROUS):

```
NAME
    Format -- Causes a filesystem to initialize itself

SYNOPSIS
    BOOL success = Format( char *filesystem, char *volumename,
                           ULONG dostype );

FUNCTION
    Interface for initializing new media on a device.  This causes the
    filesystem to write out an empty disk structure to the media, which
    should then be ready for use.  This assumes the media has been low-
    level formatted and verified already.

The filesystem should be inhibited before calling Format() to make
sure you don't get an ERROR_OBJECT_IN_USE.

INPUTS
    filesystem - Name of device to be formatted.  ':' must be supplied.
    volumename - Name for volume (if supported).  No ':'.
    dostype    - Type of format, if filesystem supports multiple types.

BUGS
    Existed, but was non-functional in V36 dos.  (The volumename wasn't
converted to a BSTR.)  Workaround:  Require V37, or under V36
convert volumename to a BPTR to a BSTR before calling Format().
Note:  A number of printed packet docs for ACTION_FORMAT are wrong
as to the arguments.
```

```
AMIGATALK INTERFACE (VeryDangerousDOS Class):

formatDisk: diskName on: volumeName type: dosType
```

1.32 exitProgram (DANGEROUS):

```
                NAME
    Exit -- Exit from a program

SYNOPSIS
    void Exit( LONG returnCode );

FUNCTION
    Exit() is currently for use with programs written as if they
    were BCPL programs.  This function is not normally useful for
    other purposes.

In general, therefore, please DO NOT CALL THIS FUNCTION!

In order to exit, C programs should use the C language exit()
function (note the lower case letter "e").  Assembly programs should
place a return code in D0, and execute an RTS instruction with
their original stack ptr.

IMPLEMENTATION

    The action of Exit() depends on whether the program which called it
    is running as a command under a CLI or not. If the program is
running under the CLI the command finishes and control reverts to
the CLI. In this case, returnCode is interpreted as the return code
from the program.

If the program is running as a distinct process, Exit() deletes the
process and release the space associated with the stack, segment
list and process structure.

INPUTS
    returnCode - integer

SEE ALSO

              CreateProc
             ,
              CreateNewProc

AMIGATALK INTERFACE (DangerousDOS Class):

exitProgram: returnCode
```

1.33 doPacket (VERY DANGEROUS):

```
                NAME
    DoPkt -- Send a dos packet and wait for reply

SYNOPSIS
    LONG result1 = DoPkt( struct MsgPort *port, LONG action,
                          LONG arg1, LONG arg2, LONG arg3,
                          LONG arg4, LONG arg5 );

FUNCTION
    Sends a packet to a handler and waits for it to return.  Any secondary
    return will be available in D1 AND from  IoErr() .  DoPkt() will work
even if the caller is an exec task and not a process; however it will
be slower, and may fail for some additional reasons, such as being
unable to allocate a signal.  DoPkt() uses your pr_MsgPort for the
reply, and will call pr_PktWait.  (See BUGS regarding tasks, though).

Only allows 5 arguments to be specified.  For more arguments (packets
support a maximum of 7) create a packet and use
             SendPkt()
            /
             WaitPkt()
            .

INPUTS
    port    - pr_MsgPort of the handler process to send to.
    action  - the action requested of the filesystem/handler
    arg1, arg2, arg3, arg4,arg5 - arguments, depend on the action & may not
                                  all be required.

RESULT
    result1 - the value returned in dp_Res1, or FALSE if there was some
              problem in sending the packet or recieving it.
    result2 - Available from IoErr() AND in register D1.

BUGS
    Using DoPkt() from tasks doesn't work in V36.
Use
             AllocDosObject()
            , PutMsg(), and WaitPort()/GetMsg()
for a workaround, or you can call
             CreateNewProc()
             to start a process to
do Dos I/O for you.  In V37, DoPkt() will allocate, use, and free the
MsgPort required.

NOTES
    Callable from a task (under V37 and above).

SEE ALSO

             AllocDosObject
            ,
             FreeDosObject
            ,

             SendPkt
```

```
              ,
               WaitPkt
              ,

               CreateNewProc
              ,  AbortPkt

AMIGATALK INTERFACE (VeryDangerousDOS Class):

doPacket: action onPort: msgPort arguments: argArray
```

1.34 deviceProc (DANGEROUS):

```
                NAME
    DeviceProc -- Return the process MsgPort of specific I/O handler

SYNOPSIS
    struct MsgPort *process = DeviceProc( char *name );

FUNCTION
    DeviceProc() returns the process identifier of the process which
    handles the device associated with the specified name. If no
process handler can be found then the result is zero. If the name
refers to an assign then a directory lock is returned in  IoErr() .
This lock should not be  UnLock() ed or  Examine() ed (if you wish to do
so,  DupLock()  it first).

BUGS
    In V36, if you try to DeviceProc() something relative to an assign
made with  AssignPath() , it will fail.  This is because there's no
way to know when to unlock the lock.  If you're writing code for
V36 or later, it is highly advised you use  GetDeviceProc()  instead,
or make your code conditional on V36 to use GetDeviceProc()/

               FreeDeviceProc()
              .

SEE ALSO
     GetDeviceProc ,
               FreeDeviceProc
              ,
     DupLock ,  UnLock ,
     Examine

AMIGATALK INTERFACE (DangerousDOS Class):

makeDeviceProcess: deviceName
```

1.35 deleteVar (DANGEROUS):

```
                NAME
    DeleteVar -- Deletes a local or environment variable

SYNOPSIS
    BOOL success = DeleteVar( char *name, ULONG flags );

FUNCTION
    Deletes a local or environment variable.

INPUTS
    name   - pointer to an variable name.  Note variable names follow
             filesystem syntax and semantics.
    flags  - combination of type of var to delete (low 8 bits), and
             flags to control the behavior of this routine.  Currently
             defined flags include:

     GVF_LOCAL_ONLY  - delete a local (to your process) variable.
     GVF_GLOBAL_ONLY - delete a global environment variable.

     The default is to delete a local variable if found, otherwise
     a global environment variable if found (only for LV_VAR).

RESULT
    success - If non-zero, the variable was sucessfully deleted, FALSE
              indicates failure.

BUGS
    LV_VAR is the only type that can be global

SEE ALSO
     GetVar ,  SetVar ,
     FindVar ,
              DeleteFile
             ,
    <dos/var.h>

AMIGATALK INTERFACE (DangerousDOS Class):

deleteVar: varName flags: f
```

1.36 deleteFile (VERY DANGEROUS):

```
NAME
    DeleteFile -- Delete a file or directory

SYNOPSIS
    BOOL success = DeleteFile( char *name );

FUNCTION
    This attempts to delete the file or directory specified by 'name'.
    An error is returned if the deletion fails.  Note that all the files
    within a directory must be deleted before the directory itself can
    be deleted.
```

```
INPUTS
    name - pointer to a null-terminated string

AMIGATALK INTERFACE (VeryDangerousDOS Class):

deleteFile: fileOrDirName
```

1.37 CreateProc (DANGEROUS):

```
                NAME
    CreateProc -- Create a new process

SYNOPSIS
    struct MsgPort *process = CreateProc( char *name,
                                          LONG  pri,
                                          BPTR  seglist,
                                          LONG  stackSize )

FUNCTION
    CreateProc() creates a new AmigaDOS process of name 'name'.  AmigaDOS
    processes are a superset of exec tasks.

A seglist, as returned by
              LoadSeg()
             , is passed as 'seglist'.
This represents a section of code which is to be run as a new
process. The code is entered at the first hunk in the segment list,
which should contain suitable initialization code or a jump to
such.  A process control structure is allocated from memory and
initialized.  If you wish to fake a seglist (that will never
have DOS UnLoadSeg() called on it), use this code:

        DS.L    0   ;Align to longword
        DC.L    16   ;Segment "length" (faked)
        DC.L    0   ;Pointer to next segment
        ...start of code...

The size of the root stack upon activation is passed as
'stackSize'.  'pri' specifies the required priority of the new
process.  The result will be the process msgport address of the new
process, or zero if the routine failed.  The argument 'name'
specifies the new process name.  A zero return code indicates error.

The seglist passed to CreateProc() is not freed when it exits; it
is up to the parent process to free it, or for the code to unload
itself.

Under V36 and later, you probably should use
              CreateNewProc()
              instead.

INPUTS
    name      - pointer to a null-terminated string
    pri       - signed long (range -128 to +127)
```

```
    seglist   - BCPL pointer to a seglist
    stackSize - integer (must be a multiple of 4 bytes)

RESULT
    process   - pointer to new process msgport

SEE ALSO

              CreateNewProc
             ,
              LoadSeg
             ,

              UnLoadSeg

AMIGATALK INTERFACE (DangerousDOS Class):

createProcess: processName  priority: pri
     segments: bptrSegmentList stack: stackSize
```

1.38 createNewProc (DANGEROUS):

```
                NAME
    CreateNewProc -- Create a new process

SYNOPSIS
    struct Process *process = CreateNewProc( struct TagItem *tags );

FUNCTION
    This creates a new process according to the tags passed in.  See
    dos/dostags.h for the tags.

You must specify one of NP_Seglist or NP_Entry.  NP_Seglist takes a
seglist (as returned by LoadSeg()).  NP_Entry takes a function
pointer for the routine to call.

There are many options, as you can see by examining dos/dostags.h.
The defaults are for a non-CLI process, with copies of your
CurrentDir, HomeDir (used for PROGDIR:), priority, consoletask,
windowptr, and variables.  The input and output filehandles default
to opens of NIL:, stack to 4000, and others as shown in dostags.h.
This is a fairly reasonable default setting for creating threads,
though you may wish to modify it (for example, to give a descriptive
name to the process.)

CreateNewProc() is callable from a task, though any actions that
require doing Dos I/O (DupLock() of currentdir, for example) will not
occur.

NOTE:  If you call CreateNewProc() with both NP_Arguments, you must
not specify an NP_Input of NULL.  When NP_Arguments is specified, it
needs to modify the input filehandle to make ReadArgs() work properly.

INPUTS
```

```
    tags - a pointer to a TagItem array.


RESULT
    process - The created process, or NULL.  Note that if it returns
              NULL, you must free any items that were passed in via
              tags, such as if you passed in a new current directory
              with NP_CurrentDir.

BUGS
    In V36, NP_Arguments was broken in a number of ways, and probably
    should be avoided (instead you should start a small piece of your
    own code, which calls
              RunCommand()
              to run the actual code you wish
    to run).  In V37, NP_Arguments works, though see the note above.

SEE ALSO

              LoadSeg
             ,
              CreateProc
             ,
     ReadArgs ,
              RunCommand
             ,
    <dos/dostags.h>

AMIGATALK INTERFACE (DangerousDOS Class):

createNewProcess: tagArray
```

1.39 cliInitRun (DANGEROUS):

```
                NAME
    CliInitRun -- Set up a process to be a shell from initial packet

SYNOPSIS
    LONG flags = CliInitRun( struct DosPacket *packet );

FUNCTION
    This function initializes a process and CLI structure for a new
    shell, from parameters in an initial packet passed by the system
(Run, System(), Execute()).  The format of the data in the packet
is purposely not defined.  The setup includes all the normal fields
in the structures that are required for proper operation (current
directory, paths, input streams, etc).

It returns a set of flags containing information about what type
of shell invocation this is.

Definitions for the values of fn:
    Bit 31     Set to indicate flags are valid
    Bit  3     Set to indicate asynch system call
```

```
    Bit  2     Set if this is a System() call
    Bit  1     Set if user provided input stream
    Bit  0     Set if RUN provided output stream

If Bit 31 is 0, then you must check  IoErr()  to determine if an error
occurred.  If IoErr() returns a pointer to your process, there has
been an error, and you should clean up and exit.  The packet will
have already been returned by
             CliInitNewcli()
            .  If it isn't a pointer
to your process and Bit 31 is 0, you should wait before replying
the packet until after you've loaded the first command (or when you
exit).  This helps avoid disk "gronking" with the Run command.
(Note:  This is different from what you do for CliInitNewcli().)

If Bit 31 is 1, then if Bit 3 is one, ReplyPkt() the packet
immediately (Asynch System()), otherwise wait until your shell exits
(Sync System(), Execute()).
(Note:  This is different from what you do for CliInitNewcli().)

This function is very similar to CliInitNewcli().

INPUTS
    packet - the initial packet sent to your process MsgPort

RESULT
    fn - flags or a pointer

SEE ALSO

             CliInitNewcli()
            ,
             ReplyPkt
            ,

             WaitPkt
            ,  Execute ,
     IoErr , System()

AMIGATALK INTERFACE (DangerousDOS Class):

cliInitRun: dosPacketObject
```

1.40 cliInitNewcli (DANGEROUS):

```
                NAME
    CliInitNewcli -- Set up a process to be a shell from initial packet

SYNOPSIS
    LONG flags = CliInitNewcli( struct DosPacket *packet );

FUNCTION
    This function initializes a process and CLI structure for a new
    shell, from parameters in an initial packet passed by the system
```

```
(NewShell or NewCLI, etc).  The format of the data in the packet
is purposely not defined.  The setup includes all the normal fields
in the structures that are required for proper operation (current
directory, paths, input streams, etc).

It returns a set of flags containing information about what type
of shell invocation this is.

Definitions for the values of fn:
    Bit 31     Set to indicate flags are valid
    Bit  3     Set to indicate asynch system call
    Bit  2     Set if this is a System() call
    Bit  1     Set if user provided input stream
    Bit  0     Set if RUN provided output stream

If Bit 31 is 0, then you must check  IoErr()  to determine if an error
occurred.  If IoErr() returns a pointer to your process, there has
been an error, and you should clean up and exit.  The packet will
have already been returned by CliInitNewcli().  If it isn't a pointer
to your process and Bit 31 is 0, reply the packet immediately.
(Note:  This is different from what you do for
             CliInitRun()
            .)

This function is very similar to CliInitRun().

INPUTS
    packet - the initial packet sent to your process MsgPort

RESULT
    fn - flags or a pointer

SEE ALSO

             CliInitRun()
            ,
             ReplyPkt
            ,

             WaitPkt
            ,  IoErr

AMIGATALK INTERFACE (DangerousDOS Class):

cliInitNewCLI: dosPacketObject
```

1.41 attemptLockDosList (DANGEROUS):

```
NAME
    AttemptLockDosList -- Attempt to lock the Dos Lists for use

SYNOPSIS
    struct DosList *dlist = AttemptLockDosList( ULONG flags );
```

```
FUNCTION
    Locks the dos device list in preparation to walk the list.  If the
    list is 'busy' then this routine will return NULL.  See  LockDosList()
    for more information.

INPUTS
    flags - Flags stating which types of nodes you want to lock.

RESULT
    dlist - Pointer to the beginning of the list or NULL.  Not a valid
            node!

BUGS
    In V36 through V39.23 dos, this would return NULL or 0x00000001 for
    failure.  Fixed in V39.24 dos (after kickstart 39.106).

SEE ALSO
     LockDosList ,  UnLockDosList ,
     NextDosEntry , Forbid()

AMIGATALK INTERFACE (DangerousDOS Class):

attemptLockDosList: flags
```

1.42 allocDosObject (DANGEROUS):

```
                NAME
    AllocDosObject -- Creates a dos object

SYNOPSIS
    void *ptr = AllocDosObject( ULONG type, struct TagItem *tags );

FUNCTION
    Create one of several dos objects, initializes it, and returns it
    to you.  Note the DOS_STDPKT returns a pointer to the sp_Pkt of the
    structure.

This function may be called by a task for all types and tags defined
in the V37 includes (DOS_FILEHANDLE through DOS_RDARGS and ADO_FH_Mode
through ADO_PromptLen, respectively).  Any future types or tags
will be documented as to whether a task may use them.

INPUTS
    type - type of object requested
    tags - pointer to taglist with additional information

RESULT
    packet - pointer to the object or NULL

BUGS
    Before V39, DOS_CLI should be used with care since
              FreeDosObject()
                    can't free it.

SEE ALSO
```

```
              FreeDosObject
             ,
    <dos/dostags.h>, <dos/dos.h>

AMIGATALK INTERFACE (DangerousDOS Class):

allocDosObject: type tags: tagArray   " Tested "
```

1.43 addSegment (VERY DANGEROUS):

```
                NAME
    AddSegment - Adds a resident segment to the resident list

SYNOPSIS
    BOOL success = AddSegment( char *name, BPTR seglist, LONG type )

FUNCTION
    Adds a segment to the Dos resident list, with the specified Seglist
    and type (stored in seg_UC - normally 0).  NOTE:  Currently unused
    types may cause it to interpret other registers (D4-?) as additional
    parameters in the future.

Do NOT build Segment structures yourself!

INPUTS
    name    - name for the segment
    seglist - Dos seglist of code for segment
    type    - initial usecount, normally 0

RESULT
    success - success or failure

SEE ALSO
     FindSegment ,
              RemSegment
             ,

              LoadSeg

AMIGATALK INTERFACE (VeryDangerousDOS Class):

addSegment: bptrSegList named: segmentName useCount: count
```

1.44 addDosEntry (DANGEROUS):

```
                NAME
    AddDosEntry -- Add a Dos List entry to the lists

SYNOPSIS
```

```
    LONG success = AddDosEntry( struct DosList *dlist );

FUNCTION
    Adds a device, volume or assign to the dos devicelist.  Can fail if it
    conflicts with an existing entry (such as another assign to the same
name or another device of the same name).  Volume nodes with different
dates and the same name CAN be added, or with names that conflict with
devices or assigns.  Note:  The dos list does NOT have to be locked to
call this.  Do not access dlist after adding unless you have locked the
Dos Device list.

An additional note concerning calling this from within a handler:
in order to avoid deadlocks, your handler must either be multi-
threaded, or it must attempt to lock the list before calling this
function.  The code would look something like this:

if (AttemptLockDosList( LDF_xxx | LDF_WRITE ))
   {
   rc = AddDosEntry( ... );
   UnLockDosList( LDF_xxx | LDF_WRITE );
   }

If
              AttemptLockDosList()
              fails (i.e. it's locked already), check for
messages at your filesystem port (don't wait!) and try the
AttemptLockDosList() again.

INPUTS
    dlist   - Device list entry to be added.

RESULT
    success - Success/Failure indicator

SEE ALSO

              RemDosEntry
             ,  FindDosEntry ,
     NextDosEntry ,  LockDosList ,
     MakeDosEntry ,
              FreeDosEntry
             ,

              AttemptLockDosList

AMIGATALK INTERFACE (DangerousDOS Class):

addDosEntry: dosListObject
```