

Function 1 (81h) - Output the Character in AL

INT 14 Entry Registers:

AH = 1 (81h)
AL = Character to send.
DX = port number

Return Values:

AH = Port Status (defined under Function 3). (bits 0-6)

Errors Returned:

If bit 7 of the AH register is on, the driver was unable to send the character. This is a time out on the send. Bits 0-6 of AH are the same as AH in a status request call.

Description:

Function 1 attempts to send the character in AL. If the port is busy, it queues the character for output. Otherwise, the character goes right to the port. If the output queue is full, the function enters a timed wait during which it calls MOS to switch tasks. If the driver is for a smart card, the character will likely be queued right away since the driver may not communicate directly with the port.

This function uses an internal time out count. See function 21 for an output character function where the time out value may be specified.

CHAPTER 12

Function 2 (82h) - Receive a Character into AL

INT 14 Entry Registers:

AH = 2 (82h)
DX = port number.

Return Values:

AH = Port Status (defined under Function 3). (bits 0-6).
AL = Character from port or input buffer.

Errors Returned:

If bit 7 of the AH register is on, a receive character time out occurred, i.e. the input buffer is empty. Bits 0 - 6 of AH are the same as AH in the status function call.

Description:

Function 2 attempts to dequeue a character from the input buffer associated with the specified port. If the buffer is empty it enters a timed loop awaiting a character. During this loop, the driver calls MOS to allow for a task switch to increase throughput. If the loop terminates with no character received, the function indicates a time out via the AH register along with the port status in bits 0 - 6 and returns.

This function uses an internal time out count. See function 22 for an input character function where the time out value may be specified.

Function 3 (83h) - Return Port Status

INT 14 Entry Registers:

AH = 3 (83h)

DX = port number

Returned Values:

AH = Port Status

AL = Modem Status Register

Errors Returned: None

Description: This function returns the status of the port. The definition of the bits are:

Port Status (AH)

- 0 - data ready
- 1 - data overrun error
- 2 - parity error
- 3 - framing error
- 4 - break interrupt
- 5 - xmitter holding reg. empty
- 6 - xmitter shift reg. empty
- 7 - not used

Modem Status (AL)

- 0 - Change in CTS
- 1 - Change in DSR
- 2 - Change in Ring Indicator
- 3 - Change in CD
- 4 - CTS
- 5 - DSR
- 6 - RI
- 7 - CD

CHAPTER 12

Function 4 (84h) - Alternate Port Initialization

INT 14 Entry Registers:

AH = 4 (84h)
AL = bits 0 - 4 same as function 0
 bits 5 - 7 ignored
CX:BX = baud rate. BX is low order word
DX = port number

Returned Values:

AH = 0 - Success.

Errors Returned:

AH = 0FFh - Invalid transmission speed.

Description:

Function 4 provides for transmission speeds the ROM BIOS doesn't support. It enables a port disabled by function 11.

Function 5 (85h) - Change Port Protocol

INT 14 Entry Registers:

AH = 5 (85h)
AL = 0xxx0000 - no protocol
AL = 0xxx0001 - enable (disable) rcv XON/XOFF
AL = 0xxx0010 - enable (disable) xmit XON/XOFF
AL = 0xxx0100 - enable (disable) dtr/dsr
AL = 0xxx1000 - enable (disable) rts/cts

New AL values in release 4.00:

AL = 1xxxxxxx - allow change of CD monitoring
AL = 1xx1xxxx - enable (disable) CD monitoring
AL = 1x1xxxxx - enable (disable) auto reboot

BH = New XOFF characters
BL = New XON characters
DX = port number

Returned Values:

None

Errors returned:

AH = 0FFh - Invalid protocol

Description:

This function allows the user to change protocols. If XON/XOFF is desired and BX is non-zero, BH is the new XOFF character and BL is the new XON character. This allows the driver to support the XPC protocol which is functionally the same as XON/XOFF except for the characters representing XON and XOFF. This function allows for using the hardware flow-control sequences.

CHAPTER 12

With release 4.00 of MOS, this INT 14 function includes some new features. When the most significant bit is set, bit 7, then bits 5 and 4 are regarded as being significant. Note that the lower four bits are still significant in this case.

When bits 7 and 4 are set, the new carrier detection feature is enabled. This is intended for use when a modem is attached to a serial port to support a remote workstation. When \$SERIAL.SYS detects that the modem's carrier has dropped, all successive incoming data will be ignored.

This state will persist until the carrier signal is found to be high again. This is very useful for noise reduction. This same feature can be activated with the command line parameter CN=R.

When bits 5 and 7 are set, the new automatic reboot-on-carrier-drop feature is enabled. When the carrier drops, the next call the MOS kernel makes to function 8 will return with the CY flag set. The kernel will then restart the task associated with the serial port. This CY return indication will only occur for the first call to function 8 after the carrier drop transition.

Function 6 (86h) - Driver "ID" Function

INT 14 Entry Registers:

AH = 6 (86h)
DX = port number

Returned Values:

AH = returns with the high-order bit set.
AL = The number of the highest function the driver supports.

Errors Returned:

None

Description:

Set DX=0 to determine if a serial port driver has been installed on the system. If no serial driver is installed, the high bit of AH will be clear. The initial release of the PC-EmuLink™ software used this function to determine if it could use the driver or the ROM BIOS. The driver allows Emulink to do graphics due to the ability to have interrupt driven, buffered I/O. Returning the highest function supported in AL will allow terminal device drivers to determine if the driver supports the string handling function.

The driver must examine the DX register to see if the specified port is under its control. If so, it executes the function and returns. If not, it must pass the request to the next driver in the chain. The AX register has the information in it upon return, which must be passed to the program that invoked the INT 14H call.

CHAPTER 12

Function 7 (87h) - Send RS232C "Break"

INT 14 Entry Registers:

AH = 7 (87h)

BX = The number of timer ticks to send the break.

DX = port number

Returned Values:

None

Errors Returned:

None

Description:

Function 7 allows the sending of a "break" signal for whatever reason is desired. BX is the timer for holding the break. The "break" causes a framing error to the receiver.

Function 8 (88h) - Input Status Check

INT 14 Entry Registers:

AH = 8 (88h)

DX = port number

Return Values:

If the carry flag is set (1) then the carrier monitoring logic has detected a loss of carrier and is signalling that the task associated with this port should be restarted. This feature was introduced with release 4.00. See the discussion of function 5 for more details.

If the zero flag is clear (0), the next character in the input buffer to be dequeued is in the AL register. If the zero flag is set (1), the input buffer is empty.

Errors Returned:

None

Description:

This function allows for the examination of the next character in the input buffer without removing it from the queue. The terminal device drivers supplied with MOS use this function. Use function 2 or function 15 to remove the character from the queue.

CHAPTER 12

Function 9 (89h) - Reset I/O Buffer Pointers

INT 14 Entry Registers:

AH = 9 (89h)

DX = port number

Returned Values:

None

Errors returned:

None

Description:

This function resets the I/O buffer pointers to their original values, effectively clearing the buffers. This function flushes both the input and output buffers. If a selective flush is required, see function 19.

Function 10 (8Ah) - Input Queue Check

INT 14 Entry Registers:

AH = 10 (8Ah)

DX = port number

Returned Values:

AX = the number of characters in the input buffer.

Errors Returned:

None

Description:

This function reports the number of characters in the input queue. It is provided for applications that may require this to determine whether to send an XOFF, etc.

CHAPTER 12

Function 11 (8Bh) - Disable the Port

INT 14 Entry Registers:

AH = 11 (8Bh)

DX = port number. If the high bit is set, only disable the port if the IRQ associated with it is unique.

Returned values:

AL = 0 if no errors.

Errors Returned:

AL = 1 if the port was not disabled because the IRQ associated with it was not unique.

AL = 2 if the IRQ is already reserved.

Description:

There may be occasions when it is necessary for the driver to relinquish control of one of its ports. This function provides that ability. The method by which the driver "knows" it no longer controls the port is up to the writer of the driver.

This function was originally provided to allow for the sharing of a port between PC-EmuLink and LANLink™ software. Although neither could control the port simultaneously, this function allowed some form of control. The port is "re-enabled" or returned to the driver's control via calls 0, 4, or 13.

See Extended Services function 16H for more information on disabling serial ports.

Calling this function with the high bit set in the entry DX register is to accommodate the case where a serial port and modem will be shared between remote workstation use and telecommunications use. This makes it possible to define a port such as 03F8 to \$SERIAL.SYS for workstation use and also use the same port for a telecommunication package which must itself have absolute control over the port. The MOS DSPORT command uses this function.

CHAPTER 12

Function 12 (8Ch) - Return Current Port Parameters

INT 14 Entry Registers:

AH = 12 (8Ch)
DX = port number

Return Values:

AL = Line parameters as defined by functions 0, and 4.
AH = flow control configuration information. *
 (See the entry data for function 5)
CX:BX = baud rate. CX = high order word.
DL = xoff character (0 if not using xon/xoff) *
DH = xon character (0 if not using xon/xoff) *

* denotes items added in release 4.00

Errors Returned:

AH = 0ffh if the port number is invalid.

Description:

Function 12 returns the current port parameters. It is intended for use by applications that will take over the port using function 11. This function is used by PC-EmuLink. EmuLink issues this call prior to issuing a function 11 call to disable the port to allow LANSAT to control the port. When EmuLink regains control of the port, it re-init's it using the parameters from this call.

Function 13 (8Dh) - Register a Port with a Terminal

INT 14 Entry Registers:

AH = 13 (8Dh)
DX = port number

Returned Values:

ES:BX = pointer to a byte "flag". The flag is 0ffh if there is a character in the input buffer and zero if the buffer is empty.

Errors Returned:

AH = 0ffh if the port number is invalid. Otherwise AH = 0.

Description:

The "flag" is the "key ready" flag mentioned earlier. The driver's interrupt service routine is responsible for setting this flag to indicate a character ready. The dequeue functions (2,15) are responsible for clearing it when the input queue is empty. If the driver is for a smart card, the card's operating system should set this flag in the dual-port RAM, if applicable.

CHAPTER 12

Function 14 (8Eh) - String Output

INT 14 Entry Registers:

AH = 14 (8Eh)
CX = number of characters to transfer
DX = port number
ES:BX = pointer to string
SI = time out

Returned Values:

AX = number of characters transferred.

Errors Returned:

If the zero flag is clear (0), the transfer was successful. If the zero flag is set (1), a time out occurred during the transfer, i.e. the output buffer is full.

Description:

This function allows for the transfer for multiple characters via one "INT" call rather than a single character in one call, thereby reducing software interrupt overhead.

If SI is zero, the function uses an internal time out. If SI is non-zero, SI is the number of timer ticks allowed to send one character. If any one character in the string takes more than SI ticks to send, the function returns an error.

Function 15 (8Fh) - String Input

INT 14 Entry Registers:

AH = 15 (8Fh)
CX = MAXIMUM number of characters to read
DX = port number
ES:BX = pointer to receiving buffer
SI = time out

Returned Values:

AX = Number of characters transferred.

Errors Returned:

If the zero flag is set (1), the driver's input buffer is empty and a time out occurred waiting for a character from the port.

Description:

Function 15 allows for more than one character to be read during a single INT call, and therefore decrease the INT overhead significantly. This function tells the driver the maximum number of characters to transfer. The driver will always return in AX the number of characters transferred.

If SI is zero, the function uses an internal time out. If SI is non-zero, it is the number of timer ticks allowed to receive each character. If more than SI timer ticks go by before receiving a character, the function returns an error.

CHAPTER 12

Function 16 (90h) - Link to another Serial Driver

INT 14 Entry Registers:

AH = 16 (90h)
ES:BX = CS:IP of calling driver's INT 14 entry point.

Returned Values:

None

Errors Returned:

None

Description:

This function provides a mechanism for loading and linking more than one serial driver. Each driver linked into the INT 14 chain should store the calling driver's INT 14h CS:IP in its code segment.

Upon INT 14 entry, the first serial driver in the chain should check the DX register against the maximum port number currently under the driver's control. If DX is greater than its highest port number, the highest port number should be subtracted from DX and control passed to the next driver in the chain.

Function 17 (91h) - Write Modem Control Register

INT 14 Entry Registers:

AH = 17 (91h)

AL = Value to write to the 8250 Modem Control Register
(base+4)

DX = port number

Returned Values:

None

Errors Returned:

None

Description:

This call was introduced with release 4.00 of MOS. It allows modem control signals such as DTR and RTS to be manipulated. Remember that the DX register value is still a logical port number rather than an absolute port address.

CHAPTER 12

Function 18 (92h) - Return Driver Description

INT 14 Entry Registers:

AH = 18 (92h)
DX = port number

Returned Values:

DS:BX = pointer to a 40 byte buffer which contains a string which identifies the serial driver. This string is read-only.

Errors Returned:

None

Description:

This function was introduced in release 4.00 of MOS. It provides access to an identifying string within the serial driver. The MOS INFO command displays this string.

Function 19 (93h) - Selective Buffer Flush

INT 14 Entry Registers:

AH = 19 (93h)

AL = 1 to flush input buffer

AL = 2 to flush output buffer

AL = 3 to flush both

DX = port number

Returned Values:

None

Errors Returned:

None

Description:

This function allows just the input buffer or just the output buffer to be flushed. This function was introduced in release 4.00 of MOS. It is useful when supporting file transfer protocols.

CHAPTER 12

Function 20 (94h) - Output Queue Check

INT 14 Entry Registers:

AH = 20 (94h)

DX = port number

Returned Values:

AX = the number of characters in the output buffer.

Errors Returned:

None

Description:

This function reports the number of characters in the output queue. It is provided for applications that may require this to determine whether to send an XOFF, etc.

Function 21 (95h) - Output the Character in AL

INT 14 Entry Registers:

AH = 21 (95h)
AL = Character to send.
DX = port number
SI = time out

Return Values:

AH = Port Status (defined under Function 3). (bits 0-6)

Errors Returned:

If bit 7 of the AH register is on, the driver was unable to send the character. This is a time out on the send. Bits 0-6 of AH are the same as AH in a status request call.

Description:

Function 21 attempts to send the character in AL. If the port is busy, it queues the character for output. Otherwise, the character goes right to the port. If the output queue is full, the function enters a timed wait during which it calls MOS to switch tasks. If the driver is for a smart card, the character will likely be queued right away since the driver may not communicate directly with the port.

SI is the number of timer ticks the function will continue attempting to send the character before giving up and returning an error. If SI is zero, the function uses an internal time out.

If SI is non-zero, the function will use SI as the send time out.

CHAPTER 12

Function 22 (96h) - Receive a Character into AL

INT 14 Entry Registers:

AH = 22 (96h)
DX = port number.
SI = time out

Return Values:

AH = Port Status (defined under Function 3). (bits 0-6).
AL = Character from port or input buffer.

Errors Returned:

If bit 7 of the AH register is on, a receive character time out occurred, i.e. the input buffer is empty. Bits 0 - 6 of AH are the same as AH in the status function call.

Description:

Function 22 attempts to dequeue a character from the input buffer associated with the specified port. If the buffer is empty it enters a timed loop awaiting a character. During this loop, the driver calls MOS to allow for a task switch to increase throughput. If the loop terminates with no character received, the function indicates a time out via the AH register along with the port status in bits 0 - 6 and returns.

If SI is zero, the function uses an internal time out. If SI is non-zero, the function waits SI timer ticks to receive a character before giving up.

Function 23 (97h) - Declare port ownership

INT 14 Entry Registers:

AH = 23 (97h)

DX = port number.

BX = TCB segment/selector address of owner task

Return Values:

None

Errors Returned:

None

Description:

This function is provided for the sake of serial device drivers which require a task to port association. A task to port association would be required when dynamically allocated I/O buffers are used to support each task. Serial drivers which don't require this function should establish a null handler for AH = 23H.

A serial workstation based terminal device driver (e.g. pterm.sys) will issue this call when its register function is being processed. Should an error be detected by the terminal device driver's register function after calling this function, it will issue a call to INT 14 function 11 to signal that resources be deallocated.

To insure that resources are deallocated when a task is removed, a driver should establish a hook into the TCBUNREG call vector. See Chapter 6 for more details on this.

CHAPTER 12

NETBIOS Interface

Purpose

The NETBIOS emulator provides MOS with a software emulator which makes tasks appear to be nodes on a network. Tasks can pass information among themselves using the NETBIOS calls.

Implementation

The emulator provides a limited number of emulated adapters. You should configure the driver for one adapter for each task that will be using NETBIOS calls.

NCBs (Network Control Blocks) should always be within the task's normal RAM space. This space is between the application's PSP and the top of memory indicated in the PSP. This does not include interrupt vectors and video RAM.

Post routines are called with task switching turned off, so they must be quick.

Functions

The driver is loaded during CONFIG.SYS processing. When the driver is loaded, it parses the command line and initializes itself for the number of tasks specified. If no parameters are specified, the driver initializes for 9 tasks. The necessary structures for adapters, sessions and such, are allocated for each "user". The driver sets these software interrupt vectors: 21, 2A, 2F and 5C. Then, it returns to MOS.

The driver is called for NETBIOS functions via INT 5Ch with ES:BX pointing to a Network Control Block. The block is initialized by the application. This means allocating memory, filling in the corresponding fields with the necessary information, and allocating any additional buffer space.

A better method for calling the driver would be to for the application to store the INT 5C entry point values in a double-word location in its code segment, and use the following assembler routine:

```
pushf
cli
call dword ptr [xxxx]
```

where xxxx is the address where the INT 5C entry point is stored as mentioned above. On 80386 systems, this method simulates the software interrupt instruction without its excessive overhead. Make sure the entry point is stored offset:segment in memory.

The following chart shows the NCB:

CHAPTER 12

<u>Field</u>	<u>Size</u>	<u>Description</u>
Command code	db	desired operation
Return code	db	return status from NETBIOS
local session #	db	logical session number
NCB number	db	offset into name table
msg buffer ptr	dd	pointer to msg buff
buffer length	dw	byte count of msg buffer
call name	db	16 bytes. Local/Remote adapter you wish to call. If a CHAIN SEND call, 1st 2 bytes are the length of the second message buffer. The next 4 bytes are a pointer to a msg buffer. Last 10 bytes reserved.
name	db	16 bytes. Local adapter name.
Receive time out	db	Receive datagram time out
Send time out	db	Send datagram time out
Post routine ptr	dd	Pointer to post routine
Adapter number	db	0=1st adapter. 1=second
CMD Status field	db	command status
Reserved	db	14 reserved bytes.

NOTE: The size column in this chart is an assembler mnemonic which identifies the field. DB is a byte, DW is a word (2 bytes), DD is a double-word (4 bytes).

The following is a more detailed description of the fields of the NCB.

THE COMMAND FIELD

This 1-byte field is the function the driver must perform. Each function code has two modes of operation depending upon how you wish the driver to treat the call. To distinguish between the two, set the high order bit of the command code byte to 1. If bit 7 is 1, the driver interprets the call as a "no-wait" request. If bit 7 is 0, the call is a "wait" call.

The difference between wait and no-wait is that the no-wait functions allow the requester to continue processing while the driver attempts to perform the request. A wait call does not return to the requester until the call is completed.

When the no-wait mode is used, the driver queues the request and determines whether or not it can execute it. It then returns with 2 return codes. The first or immediate return code is a general indicator of whether or not the driver could perform the required function. If it can't, the first command return code gives a general idea of where the problem lies. The second, or final, return code is posted when the command completes.

CHAPTER 12

The caller may be notified of a function's completion in two ways. If the post routine pointer in the NCB is zero, the driver will not interrupt the requesting task and the final return code will be set in the command completed field of the NCB. If the post routine PTR is non-zero, the driver will call that address when finished and the final return code can be retrieved from the AL register or the command completed field in the NCB.

Immediate return code values are:

- 00H - Request OK.
- 03H - Command code is invalid
- 21H - Network interface busy
- 22H - Command queue overflow
- 24H - Adapter number in NCB field invalid
- 26H - Cannot cancel the command
- 4XH - Request cannot proceed (X=any #)
- 50H-FEH - The adapter malfunctioned

THE RETURN CODE FIELD

This one-byte field will contain the final return code when the driver completes the task's request. The return code is FFH while the command is pending. If the post address field is zero and the request is using the no-wait option, the driver will set this field to FFH and proceed. Upon completion of the command, the driver will set a final return status in the command completed field and an error code in this field. Your routine should loop upon the command completed field instead of the return code field. If the post field is non-zero, the driver will call that routine upon completion and the error code may be obtained from the AL register instead of this field. See the section on error recovery for more information.

THE LOCAL SESSION FIELD

This is your local session number. The number is obtained from the CALL or LISTEN functions when they have successfully completed. Use this field for the SEND and RECEIVE commands. Datagrams do not use this field. The range of valid local session numbers is 01H-0FEH. The RESET command sets the local session field to the maximum number of sessions supported. It is a 1 byte field.

THE NUMBER FIELD

The driver maintains a table of names for the ADD NAME and ADD GROUP NAME commands. The byte field in the NCB is used to store the number these commands return when the above commands are called. This byte is used as an offset into the name table. Use of this number rather than a name is required with all datagram support commands and with the RECEIVE ANY commands. The number can range from 01H to 0FEH. The RESET command sets this field to the maximum number of command blocks to be supported.

THE MESSAGE BUFFER POINTER FIELD

Since the primary use of the NETBIOS driver is to pass messages, this double-word field is used for that purpose. It is stored as a normal double-word pointer.

THE MESSAGE BUFFER LENGTH FIELD

This is a word value which tells the driver the number of bytes to transfer. The receive oriented commands use this field to indicate the number of bytes actually received. The send oriented commands use this field to tell how many bytes were actually sent.

CHAPTER 12

THE CALL NAME FIELD

All 16 bytes of this field are used to tell the driver the name with which you wish to communicate. The CHAIN SEND command only uses 6 bytes of this field. The first two are the length of the transmission buffer, the last four are the buffer address stored offset:segment. The remaining 10 bytes are reserved.

THE NAME FIELD

These 16 bytes are the name making the call. Use all of them. Spaces are acceptable.

THE RECEIVE TIME OUT FIELD

The driver uses this time out field to setup error control monitoring.

THE SEND TIME OUT FIELD

The driver uses this time out field to setup error control monitoring.

THE POST ADDRESS FIELD

These four bytes are an address that is called when the driver completes a command. It is used only with no-wait calls to the driver. It gives the driver an entry point into the requesting routine.

The post routine must preserve all registers and is responsible for setting any registers which it may need to use besides those listed next. The CS, ES, AL, and BX registers are set for the NCB being serviced. The post routine must end with an IRET.

If the post field is 0, the driver will set the command completed field with the final return code. Also, the driver is unable to interrupt the requesting task. Use of this field allows the driver to do its thing while the requesting task can continue processing.

THE ADAPTER NUMBER FIELD

This field is used to distinguish between different adapters. The MOS driver requires that this field be zero.

THE COMMAND COMPLETED FIELD

The driver uses this one-byte field to post a final return code for the requesting task. A zero value in this field indicates the command completed successfully. A non-zero values means the command completed with an error and an FFH means the command is in progress. Check this byte only if you do not set a post address and are using the no-wait option.

THE RESERVED FIELD

These 14 bytes are reserved.

CHAPTER 12

The NETBIOS Functions

The RESET Function

This call clears the name table, the session table and cancels all sessions.

The CANCEL Function

This command tells the driver to cancel the command currently in progress. Commands you can't cancel are CANCEL, ADD NAME, ADD GROUP NAME, DELETE NAME, SEND DATAGRAM, SEND BROADCAST DATAGRAM, SESSION STATUS, and RESET.

The ADAPTER STATUS Function

Since the driver is a software emulation, this call tells you version number, traffic and error statistics, adapter resource statistics and the contents of the local name table. The other fields in the buffer are not really significant. See the IBM[®] PC Network Technical Reference Manual for the format of the buffer.

The UNLINK Function

This function is used internally by the driver and should NOT be used by an application.

The ADD NAME Function

This call adds the name in the NCB to the driver's internal name table.

The ADD GROUP NAME Function

This function adds a group name to the driver's internal name table.



The DELETE NAME Function

This call removes a name from the internal tables.


The CALL Function

Call requests a session with another name. The time out interval is irrelevant because communication is essentially instantaneous.

The LISTEN Function

In order to establish a session, any relevant names must issue the listen function first. Another name may then call the listening function and establish a session. The time out for listen is ignored.

The SESSION STATUS Function



This gets the status of a session established by the CALL and LISTEN functions.

The HANG UP Function

This call breaks an established session. There is no time out. It either works instantly or not at all.


The SEND Function

This sends data to the receiver in a session.

The CHAIN SEND Function

This sends multiple buffers to a receiver.

The RECEIVE Function



This allows a name to receive data from a particular session.

CHAPTER 12

The RECEIVE ANY Function

This call allows a name to receive data from any session.

The SEND DATAGRAM Function

Sends a datagram to a name.

The SEND BROADCAST DATAGRAM Function

Sends a broadcast datagram to every name that is listening for a broadcast datagram.

The RECEIVE DATAGRAM Function

Allows a name to receive a datagram from another name.

The RECEIVE BROADCAST DATAGRAM Function

Allows a name to receive a broadcast datagram from any other name.

Error Recovery

See the IBM PC Network Technical Reference Manual for error information for all the functions. Normally, we recommend that if you get an error, you retry a certain number of times before terminating.

Most errors are either programmer errors or network errors. A programmer error might be failure to allocate enough message buffer space. If it's a network error, retry it a couple of times. If it still fails, please contact your technical support representative with detailed error information so the problem can be addressed.

The following is a list of INT 2AH and INT 2FH calls supported by the NETBIOS driver.

PC Network Interrupt 2AH

Data Passed:

AH = 0 Check for interface present

Data Returned:

AH = 0 not installed

AH = 1 installed

Comments: MOS \$NETBIOS returns AH = 1.

Data Passed:

AX = 0300h Check for absolute disk access allowed
DS:SI = pointer to ASCIIZ disk device name to check

Data Returned:

CY = set. no direct access allowed

CY = clear. direct access allowed

Comments: MOS \$NETBIOS always returns CY clear.

Data Passed:

AX = 0400h execute a NETBIOS function with error retry
ES:BX = pointer to NCB

CHAPTER 12

Data Returned:

AX = 0 no error occurred

AH = 1

AL = error. an error occurred, return code is in AL

Comments: MOS \$NETBIOS will execute the NCB once and never retry the error.

Data Passed:

AX = 0401h execute a NETBIOS function with no error retry

ES:BX = pointer to NCB

Data Returned:

AX = 0 no error occurred

AH = 1

AL = error. an error occurred, return code is in AL

Comments: Same as ax = 0400h.

Data Passed:

AX = 0500h Get network resource information

Data Returned:

AX = reserved

BX = number of network names available

CX = number of network commands available

DX = number of network sessions available

Comments: MOS \$NETBIOS supports this completely; the numbers are given on a per-task basis.

PC Network Interrupt 2FH

Data Passed:

AX = b700h See if APPEND command installed

Data Returned:

AL = 0, not installed

AL = non-zero, installed

Comments: MOS \$NETBIOS always returns al = 0.

Data Passed:

AX = b800h See if PC Network Program installed

Data Returned:

AL = 0, not installed

AL = non-zero, installed

BX = installed component flags

xxxxxxxx xxxx1xxx: Redirector installed

xxxxxxxx 1xxxxxxx: Receiver installed

xxxxxxxx xxxxx1xx: Messenger installed

xxxxxxxx x1xxxxxx: Server installed

Comments: MOS \$NETBIOS always returns al = 0ffh, bx = 00cch.

Data Passed:

AX = b803h Get current post address

Data Returned:

ES:BX = current post address

CHAPTER 12

Comments: MOS \$NETBIOS initially returns a pointer to an IRET instruction inside the \$NETBIOS driver. The post address is not used by the MOS \$NETBIOS driver for any purpose whatsoever.

Data Passed:

AX = b804h Set new post address
ES:BX = new post address

Data Returned:

no change

Comments: MOS \$NETBIOS will store this address and return it to the next get current post address call, but it does not use the address in any way.

Memory Management Interface

Purpose

To provide MOS with extended memory management to allow for an increased number and size of tasks.

WARNING: If your application must access the system's memory driver, be prepared for future changes! Restrict your dependence on this interface to a single, easily replaced procedure!

Functions

The memory management device driver gives MOS access to extended memory. It is environment specific. Different drivers are provided for 80386-based computers and also for 80286-based and lesser computers using an AT-GIZMO or similar product. 80286 and 8086/8088-based computers with no memory management are limited to 640k or less.

This section will provide the software developer with the necessary information to be able to call the device driver for services. This section will not provide the information necessary to write your own memory management driver. Developers desiring to write their own driver should contact The Software Link's Independent Vendors Department for more information.

CHAPTER 12

The following is a function summary. (Functions 6 through 9 are 80386 specific functions).

<u>Function</u>	<u>Purpose</u>
0	Reserved
1	Allocate Extended Memory
2	Release Extended Memory
3	Remap memory range
4	Unmap memory range
5	Physical memory remap
6	Protect Memory
7	Unprotect Memory
8	Set I/O Permission
9	Clear I/O Permission

Initialization

To access the driver, open the file `$$MEMDEV`, set the handle to raw mode, and read four bytes. Don't forget to close the device when finished. The read will return a double-word pointer. This pointer points to another double-word pointer, which is the driver's entry point. In other word, to get the driver's entry point, get the double-word in memory pointed to by the 4 bytes returned from the read call. See Chapter 13 for details on read a vector from a driver in raw mode.

Once this is obtained, load the AH register with the desired function number and any other registers necessary, and make a far call to the driver.

Special Considerations

Memory is allocated in banks of 4k. Function 0 is Reserved.

Function 1 - Allocate Extended Memory

Registers upon entry:

AH = 1

CX = Number of banks desired

Returned Values:

AX = 0 - Success

AX = 1 - Allocated less than requested

BX = Starting bank number

CX = Actual number of banks allocated

Errors Returned:

None

Description:

This gives applications access to extended memory. The application must be prepared to handle the case where not enough memory is available. The starting bank number returned in BX is a "handle" the driver uses to keep track of segments of allocated memory. All other registers are preserved.

CHAPTER 12

Function 2 - Release Extended Memory

Registers upon entry:

AH = 2
BX = Starting bank number
CX = Number of banks to free

Return Values:

AX = 0 - Success

Errors Returned:

AX = 1 - Attempt to release unallocated memory. The same interpretation if CX = 0.

Description:

Function 2 allows the freeing of extended memory for other applications or tasks. BX and CX are destroyed.

Function 3 - Logical Memory Remap

Registers upon entry:

AH = 3
BX = Starting bank number
CX = Number of banks
ES = Segment address to remap to

Return Values:

AH = 0

Errors Returned:

None

Description:

This function allows the driver to remap physical memory from one logical address to another. It assumes that function 1 has already been called. The driver keeps a table of beginning segment addresses when memory is allocated. The "handle" passed in BX is used by the driver to get the starting segment address for remapping. ES is the starting segment address of the destination address range and must be on segment boundaries. ES must be on a 4k boundary. All remapping is done in banks.

CHAPTER 12

Function 4 - Unmap Logical Memory

Registers upon entry:

AH = 4

CX = Number of banks

ES = Segment address from previous remap call

Return Values:

AX = 0 - Success

Errors Returned:

None

Description:

This call to the device driver makes it turn off remapping from a function 3 call. ES must be on a 4k boundary.

Function 5 - Physical Memory Remap

Registers upon entry:

AH = 5
BX = Starting physical bank number
CX = number of banks
ES = Segment address to map into

Return Values:

AX = 0

Errors Returned:

None

Description:

This call tells the driver to remap memory using an actual physical bank address in BX rather than a "handle" as the logical remap functions do. A physical bank number might be 10000h, which is the beginning address of the second megabyte of RAM. This function does not require the memory to be allocated as the logical call does.

CHAPTER 12

Function 6 - Protect Memory (80386 ONLY)

Registers upon entry:

AH = 6

BX = Handle returned from a previous logical allocate call +
100h or a physical bank number.

CX = Number of banks to protect

Return Values:

AX = 0 - Success

Errors Returned:^o

AX = Non-zero if CX is an invalid number of banks.

Description:

Function 6 sets up a memory protection table for the tasks. If an attempt to write to memory outside of the allowed range occurs, an INT 6 is generated with CS:IP of the offending instruction on the stack. A brief description of the INT 6 handler will follow the discussion of the memory management functions. BX and CX are destroyed.

Function 7 - Unprotect Memory (80386 ONLY)

Registers upon entry:

AH = 7

BX = Handle returned from a previous logical allocate call + 100h or a physical bank number.

CX = Number of banks to protect

Return Values:

AX = 0

Errors Returned:

AX = Non-zero if CX is an invalid number of banks.

Description:

This function is the converse of function 6 and frees the previously protected memory making it unprotected. BX and CX are destroyed.

CHAPTER 12

Function 8 - Set I/O Protection (80386 ONLY)

Registers upon entry:

AH = 8

ES:BX = pointer to a 128-byte bit map

Return Values:

AX = 0 - Success

Errors Returned:

None

Description:

This function sets the I/O protection map for 80386-based systems. Byte 0, bit 0 of the 128-byte map corresponds to I/O address 000. Bit 7 of byte 127 is I/O address 3ffh. A zero bit in a position allows I/O for that address. A one bit prohibits I/O for its corresponding address. The BX and CX registers are destroyed by this call.

Function 9 - Clear I/O Protection (80386 ONLY)

Registers upon entry:

AH = 9

Return Values:

AX = 0

Errors Returned:

None

Description:

This is a quick method of clearing the entire I/O permission map. It does not clear ports 4 and 81h. BX and CX are destroyed.

CHAPTER 12

The INT 6 Handler

The memory management device driver generates an INT 6 upon certain exceptions, such as attempts by a program to write outside of its memory space or access a protected I/O port. There are other internal exceptions that result in the driver issuing an INT 6. It's necessary for MOS to control this vector to maintain system integrity and stability. This vector lets the driver terminate gracefully when an application causes an exception.

Since the INT 6 is a software interrupt, the stack contains the IP, CS and FLAGS. It is NOT recommended that this vector be intercepted by an application. MOS will take it back and NOT transfer control to the offending routine.

Using \$PIPE.SYS

When you install \$PIPE.SYS, the user specifies a device name by which the driver will be referenced. To access the driver, the application or user opens the device by using the name that the driver was given when it was loaded. Thus, more than one copy of MOS's pipe driver can be installed as long as each has a unique name.

Information is written to and read from the driver in first-in first-out (FIFO) order. Should you attempt to read from an empty pipe, your task will be suspended until someone writes to the pipe. If you attempt to write to a full pipe, your task is suspended until the pipe is read from by another task.

Tasks may determine the status of the pipe using IOCTL read control string calls. The application should read a 5-byte control string from the driver. This will return a structure in the following format.

Word	Size of pipe buffer in characters
Word	Number of characters currently queued
Byte	End of file flag. Set when pipe is empty

Applications may determine if the pipe is empty by checking the second field in the structure for a zero value. You can determine if the pipe is full by comparing the first two fields for equality.

Applications set the state of the EOF flag by writing a 5-byte control string using the IOCTL write control string call. The first 4 bytes of the string are ignored. The fifth byte is placed in the driver's EOF flag, where other tasks may read it. Setting this flag will cause a read call to the driver to not suspend even though the driver's buffer is empty.

This page intentionally left blank.