Portable Systems Group

NT OS/2 Mailslot Specification

Author: *Manny Weiser*

Original Draft December 28, 1990
Revision 1.1, January 10, 1991
Revision 1.2, March 11, 1991

1. Introduction

This specification discusses the mailslot facilities of **NT OS/2**. Mailslots are a form of interprocess communication (IPC). They provide a facility for unidirectional message passing. The creator of a mailslot is the only process that can read from the mailslot. Other processes can only write messages to the mailslot.

The real value of mailslots is their usefulness in a network context. Mailslots can be used to send messages to either a single machine, or to all machines in a LAN Manager domain. More importantly, the network server needn't be running to support remote mailslots. The **NT OS/2** LAN Manager redirector can receive second class mailslot messages. The LAN Manager server must be running in order to receive first class mailslot messages. The second class mailslot allows simple peer-to-peer communication without the memory burden of running the server. Remote mailslots are described in greater detail in the **NT OS/2** LAN Manager Redirector Specification. Mailslot send classes are described later in this document.

In addition to describing the **NT OS/2** mailslot facilities, this specification also discusses the way in which the Win32 mailslot APIs are emulated.

2. Goals

The major goals for the mailslot capabilities of **NT OS/2** are the following:

1. Provide the basic primitives necessary to compatibly emulate the OS/2 LAN Manager mailslot capabilities.

2. Provide protection and security attributes for mailslots that are comparable to the capabilities provided for files and other **NT OS/2** objects.

3. Provide for LAN Manager server and client redirection of mailslots without having to enter the OS/2 subsystem.

4. Provide a fully qualified name space for mailslots that fits into the **NT OS/2** name structure in a straightforward manner.

3. Overview of OS/2 Mailslots

OS/2 mailslots provide a unidirectional IPC facility. Used locally (unnetworked) mailslots are analogous to a unidirectional-message mode-blocking named pipe.

Mailslot messages consist of a message buffer and a priority. The priority is an integer number in the range 0 to 9. Zero is the lowest priority. The OS/2 implementation treats mailslot messages as having only 2 priorities: zero and non-zero. Priority zero messages are written onto the end of the message buffer. Higher priority messages are copied to the front of the circular message buffer if their priority is higher than the message at the head of the buffer.

A mailslot is created by calling the **DosMakeMailslot** API. The creator of the mailslot receives a handle to the server side of the mailslot. Only the owner of a server side handle may read messages from the mailslot.

Under OS/2, mailslots are not part of the file system in any sense. A process cannot obtain a handle to a mailslot using **DosOpen**, nor can it use the handle obtained from **DosMakeMailslot** and pass it to a file system API such as **DosRead** or **DosWrite**.

The only way to read a mailslot is to use the **DosReadMailslot** or the **DosPeekMailslot** API. The **DosReadMailslot** API supplies a buffer for the data, but does not supply a buffer size. The buffer supplied must be at least as large as the buffer size defined by **DosMakeMailslot**. **DosReadMailslot** also returns size and priority information about the next message in the mailslot, but there is no guarantee that this message will be the next message that is read. The API takes a timeout parameter, which is the maximum time to wait for a message to become available to read if there are no messages waiting to be read when the call is issued.

DosPeekMailslot reads the next message from the mailslot but does not remove it from the mailslot buffer. It does not wait for a message to become available.

DosMailslotInfo returns configuration and status information about the mailslot and the current first message in the mailslot.

DosDeleteMailslot closes and deletes the mailslot and discards all unread messages.

The only action that can be performed on a mailslot by a process that does not have a server side handle is to write to the mailslot using the **DosWriteMailslot** API. This API, takes the name of the mailslot, rather than a handle, as a parameter. The mailslot name has one of the following forms:

- o \\*Mailslot*\*Name*. The target is a local mailslot.
- o \\Server\Mailslot\Name. The target is a remote mailslot.
- o \\Domain\Mailslot\Name. The target is the set of remote mailslots with this name in the domain *Domain*.
- o \\*\\*Mailslot*\*Name*. The target is the set of remote mailslots with this name in the workstation's primary domain.

In addition to multiple mailslot name formats **DosWriteMailslot** also supports two classes of mailslot: first class and second class. A first class mailslot write guarantees delivery, while a second class mailslot write does not.

A second class write to a local mailslot is identical to a first class write. A first class write can be used to write a message to a remote machine running as a server. However, in order to write to a remote workstation only machine, or to do a broadcast write to a domain of machines, only a second class write can be used.

DosWriteMailslot allows specification of a timeout. This is the maximum time to wait for enough space to become available in the mailslot buffer to complete the write.

4. Overview of NT OS/2 Mailslots

4.1 Implementation Alternatives

Mailslots, as defined by OS/2 LAN Manager, are not a perfect fit for an NT file system. Special semantics exist for creating, reading and writing to mailslots.

There are two ways to implement **NT OS/2** mailslots:

1. Implement the mailslot capabilities as a separate object with its own complete set of APIs.

2. Implement the mailslot capabilities as a file system.

The first alternative is attractive because mailslot APIs can be designed to directly support OS/2 LAN Manager semantics. However, using this approach would complicate the security and networking implementations.

The second alternative has the advantage that it allows mailslots to use the built-in file system features of the NT I/O system. However, some allowance must be made to adapt to mailslot semantics.

There are several ways to implement mailslots as a file system.

1. Extend or bend the currently existing I/O system APIs, by adding new parameters, or by redefining old parameters that are not needed by the mailslot file system.

2. Use extended attributes as the means of defining the mailslot attributes required by OS/2.

3. Create a new API for creating a mailslot, and use a combination of existing APIs and the NtFsControlFile sub-APIs to implement the desired features.

The first alternative does not add any new APIs to the system but requires either adding special case code to the I/O system or adding parameters to **NtReadFile** and **NtWriteFile**. This is clearly unacceptable.

The second alternative also requires special case code for dealing with extended attributes on a mailslot.

The third alternative requires the addition of a new mailslot only API.

Mailslots will be implemented as a file system with a new API to create the mailslot and existing APIs for all other mailslot functions. This will be the easiest to implement and will yield an efficient implementation.

NT OS/2 will not support priorities or mailslot send classes. Priorities as implemented in OS/2 are essentially useless. The absence of a first class mailslot write may be noticed, but this is unlikely, as there are no known applications that use first class mailslots.

4.2 Read/Write Buffering Strategy

4.2.1 OS/2 Read/Write Buffering Strategy

OS/2 implements mailslots using a synchronous I/O model with a single circular buffer. The buffer must fit in a single 64KB segment. For each write/read operation there are always 2 data copies. Write data is copied from the user buffer to the mailslot buffer. Read data is copied from the mailslot buffer to the user buffer.

The read and write operations are controlled by four semaphores. Two signaling semaphores are used to signal waiting readers or writers that data is available to be read or that space is now available to write into the mailslot buffer. The other two semaphores are used to restrict reading or writing to a single thread at one time.

When a write begins the writer obtains the write lock. If there is no space available in the mailslot buffer the writer waits on the write semaphore for the timeout period specified by the caller. If there is space in the mailslot buffer, or space becomes available, the writer copies its data onto the end of the mailslot buffer.

If the write message is a high priority message (priority greater than zero) and its priority is greater than the priority of the message at the head of the mailslot message queue, then the writer must also obtain the read lock, copy the message ahead of the first message in the buffer, update the next-message-to-read pointer and release the read lock.

When the write has completed, it signals the read semaphore, indicating that write data is available and releases the write lock.

When a read operation begins the reader obtains the read lock. If there is no message available to read, the reader waits on the read semaphore for the time period specified by the caller. If a message is available to be read, the reader copies the message and updates the next message available pointer.

When the read is complete, the reader signals the write semaphore, indicating that there is space in the mailslot buffer, and releases the read lock.

4.2.2 NT OS/2 Read/Write Buffering Strategy

NT OS/2 supports an asynchronous I/O model and uses the concept of quotas to control the allocation of system buffers.

The mailslot buffer is not actually allocated to real memory in **NT OS/2**. Instead, the creator of a mailslot is simply charged memory quota for the buffer. Writers can use up to the quota charged to the creator without having any quota charged against themselves. If the quota charged to the creator is exhausted then the writer is charged for any additional memory that is required. Likewise, a reader is charged quota, equal to the size of the user's read buffer, if no data is available, the quota charged to the creator is exhausted, and the read request is queued rather than completed immediately.

The following is a somewhat simplified discussion of the buffering scheme used for mailslots in **NT OS/2**. The exact behavior of the **NT OS/2** mailslot buffering depends on whether a read occurs before a write or vice versa.

When a write operation occurs the writer's output buffer is probed for read accessibility in the requesting mode. A system buffer is allocated that is the required size to hold the write data and memory quota is charged to the writer if and only if the quota charged to the creator of the mailslot instance has been exhausted (e.g., because of a previous read or write request). A buffer header is initialized at the front of the system buffer, the write data is copied into the system buffer, and the buffer header is inserted into the prioritized list of writers. The writer's I/O request is always completed immediately. The system buffer will be deallocated and the creator's quota returned when a matching read arrives.

At this point the write has been completed and control is returned to the caller. If another write request is received before the first write message is read, then the same operations are performed and the new request is placed at the appropriate place in the pending queue.

At some subsequent point in time, a read request arrives and it is determined that write data is available. The caller's input buffer is probed for write accessibility in the requesting mode. The read then proceeds to "pull" (copy) data directly from the system buffer that was previously allocated for the write data into the user's input buffer. At the completion of the copy, the read I/O request is completed.

Completion of the read request involves writing the I/O status block and setting the completion event. The system buffer for the original write request is deallocated and memory quota is returned for mailslot write buffering.

If an access violation occurs during a copy from the output buffer to a system buffer, then the write operation is immediately terminated. This has no effect on the integrity of the system. A malicious writer could easily accomplish the same effect by simply writing a shortened message. The write I/O status is set to access violation, the write I/O request is completed, and successful completion is returned as the service status.

If a read operation occurs before a write, then the reader's input buffer is probed for write accessibility in the requesting mode. A system buffer is allocated that is the required size to hold the input data and memory quota is charged to the reader if and only if the quota charged to the creator of the mailslot instance has been exhausted (e.g., because of a previous read request). A buffer header is initialized at the front of the system buffer and the header is inserted in a first-in-first-out list of readers. The read request type is converted to a buffered request so that upon completion, the I/O system will copy the received data from the system buffer into the reader's input buffer, deallocate the system buffer, and return memory quota as appropriate.

At this point, the I/O operation is pending and control is returned to the caller. If another read request is received before the first read is completed, then the same operations are performed and the new request is placed at the end of the pending queue.

At some subsequent point in time, a write request arrives and it is determined that a read is pending. The caller's output buffer is probed for read accessibility in the requesting mode. The write then proceeds to "push" (copy) data directly from the output buffer into the system buffer that was previously allocated for the read operation. At the completion of the copy, the read and write I/O requests are both completed.

When a read operation is queued, a timer is started and set for the timeout specified by the read operation. A pointer to the timer object is saved in the header of the read buffer. If the read operation is dequeued for completion before the timer expires then the timer is cancelled, and the operation completes normally.

If the timeout DPC is called, the queue of pending reads is searched for the read corresponding to the expired timer. If it is not found the read is assumed to have completed and there is no need to take any action. If the read is found it is dequeued from the pending queue and completed with an error status.

If the buffer supplied by the reader is not large enough to read the entire mailslot message the read operation fails.

Completion of the write request involves writing the I/O status block and setting the completion event, whereas completion of the read request requires copying the read data from the system buffer to the reader's input buffer, deallocating the system buffer and returning the memory quota as appropriate, writing the I/O status block, and setting the completion event.

If an access violation occurs during a copy from a system buffer to the input buffer, then the read operation is immediately terminated. Previously completed write I/O requests are not backed out. This has no effect on the integrity of the system. A malicious reader could easily accomplish the same effect by simply reading and discarding information. The read I/O status is set to access violation and the read I/O request is completed with an error status.

5. NT OS/2 Mailslot I/O Operations

The following subsections describe the **NT OS/2** I/O operations with respect to mailslots. Additional information can be found in the **NT OS/2** I/O System Specification.

5.1 Create Mailslot

A server end handle to a mailslot is obtained by calling the **NtCreateMailslotFile** function:

NTSTATUS
NtCreateMailslotFile (
    OUT PHANDLE *FileHandle,*
    IN ULONG *DesiredAccess,*
    IN POBJECT_ATTRIBUTES *ObjectAttributes,*
    OUT PIO_STATUS_BLOCK *IoStatusBlock,*
    IN ULONG *CreateOptions,*
    IN ULONG *MailslotQuota,*
    IN ULONG *MaximumMessageSize,*
    IN PTIME *ReadTimeout*
    );

Parameters:

FileHandle - A pointer to a variable that receives the file handle value.

DesiredAccess - Specifies the type of access that the caller requires to the mailslot.

DesiredAccess Flags:

SYNCHRONIZE – The file handle may be waited on to synchronize with the completion of I/O operations.

READ_CONTROL – The ACL and ownership information associated with the mailslot may be read.

WRITE_DAC – The discretionary ACL associated with the mailslot may be written.

WRITE_OWNER – Ownership information associated with the mailslot may be written.

FILE_READ_DATA – Data may be read from the mailslot.

FILE_WRITE_DATA – Data may be written to the mailslot.

FILE_READ_ATTRIBUTES - Mailslot attributes flags may be read.

FILE_WRITE_ATTRIBUTES - Mailslot attribute flags may be written.

The three following values are the generic access types that the caller may request along with their mapping to specific access rights:

GENERIC_READ - Maps to *FILE_READ_DATA* and *FILE_READ_ATTRIBUTES*.

GENERIC_WRITE - Maps to *FILE_WRITE_DATA* and *FILE_WRITE_ATTRIBUTES*.

GENERIC_EXECUTE - Maps to *SYNCHRONIZE*.

ObjectAttributes – A pointer to a structure that specifies the object attributes; refer to the I/O System Specification for details.

IoStatusBlock - A pointer to a structure that receives the final completion status. The actual action taken by the system is written into the *Information* field of this structure.

CreateOptions – Specifies the options that should be used when creating the mailslot.

<u>CreateOptions Flags</u>:

FILE_SYNCHRONOUS_IO_ALERT - Indicates that all operations on the mailslot are to be performed synchronously. Any wait that is performed on behalf of the caller is subject to premature termination by alerts.

FILE_SYNCHRONOUS_IO_NONALERT - Indicates that all operations on the mailslot are to be performed synchronously. Any wait that is performed on behalf of the caller is not subject to premature termination by alerts.

MailslotQuota - Specifies the pool quota that is reserved for the mailslot. If set to *MAILSLOT_SIZE_AUTO* to file system will set the pool quota to zero. This means that all mailslot quota will come from readers or writers.

MaximumMessageSize - Specifies the maximum size message that can be written to the mailslot.

ReadTimeout - If specified, specifies the maximum amount of time that a read operation can block waiting for a mailslot message to become available. The default setting is *MAILSLOT_WAIT_FOREVER*.

This service creates a mailslot. The Access Control List (ACL) from the object attributes parameter defines the discretionary access control for the mailslot.

The create options, and share access are set to their specified values.

The actual pool quota that is reserved for the mailslot is either the system default, the system minimum, the system maximum, or the specified quota rounded up to the next allocation boundary.

The name of the mailslot is taken from the object attributes parameter, which must be specified.

The mailslot is deleted, along with any unread message, when the last reference to the creation handle is closed.

If *STATUS_SUCCESS* is returned as the service status, then the mailslot was successfully created.

If *STATUS_INVALID_PARAMETER* is returned as the service status, then an invalid value was specified for one or more of the input parameters.

5.2 Create File

The **NtCreateFile** function can be used to open a client end handle to an instance of a specified mailslot.

In order to use this function to open a mailslot, the mailslot must already exist and the *CreateDisposition* value must be specified as either *FILE_OPEN* or *FILE_OPEN_IF*.

ShareAccess should be set to *FILE_SHARE_WRITE* | *FILE_SHARE_READ*.

If a mailslot of the specified name cannot be found, then *STATUS_OBJECT_PATH_NOT_FOUND* is returned as the service status.

5.3 Open File

The **NtOpenFile** function can be used to open a client end handle to an instance of a specified mailslot.

ShareAccess should be set to *FILE_SHARE_WRITE* | *FILE_SHARE_READ*.

If a mailslot of specified name cannot be found, then *STATUS_OBJECT_PATH_NOT_FOUND* is returned as the service status.

5.4 Read File

The **NtReadFile** function can be used to read data from the server end of a mailslot. Priority information for the message is discarded. If the mailslot is empty, the operation will be queued. The operation will complete when either of the following is true:

1. A write operation occurs and data becomes available to be read.
2. The mailslot file handle is closed.

The byte offset and key parameters of the **NtReadFile** function are ignored by the mailslot file system.

If *STATUS_PENDING* is returned as the service status, then the read I/O operation is pending and its completion must be synchronized using the standard **NT OS/2** mechanisms. Any other service status indicates that the read I/O operation has already been completed or will never complete. If a success status is returned, then the I/O status block contains the I/O completion information. Otherwise, the service status determines any error that may have occurred.

If the I/O status *STATUS_ACCESS_VIOLATION* is returned, then part of the read buffer became inaccessible after it was probed for write access.

If the I/O status *STATUS_BUFFER_OVERFLOW* is returned, then the read I/O operation was completed successfully, but the size of the input buffer was not large enough to hold the entire input message. A full buffer of data is returned; additional data can be read from the message using the **NtReadFile** function. The I/O status block contains the number of bytes that were read.

If the I/O status *STATUS_SUCCESS* is returned, then the read I/O operation was completed successfully and the I/O status block contains the number of bytes that were read.

5.5 Write File

The **NtWriteFile** function can be used to write data to a mailslot.

The byte offset and key parameters of the **NtWriteFile** function are ignored by the mailslot file system.

If a success status is returned, then the I/O status block contains the I/O completion information. Otherwise, the service status determines any error that may have occurred.

If the size of the message buffer is larger then the maximum message size specified by the caller of **NtCreateMailslotFile**, then the operation will complete with the I/O status *STATUS_BUFFER_TOO_SMALL* and no data is written to the mailslot.

If the I/O status *STATUS_ACCESS_VIOLATION* is returned, then part of the write buffer became inaccessible after it was probed for read access.

If the I/O status *STATUS_SUCCESS* is returned, then the write I/O operation was completed successfully and the I/O status block contains the number of bytes that were written.

5.6 Read Terminal File

This function is not supported by the mailslot file system.

5.7 Query Directory Information

The **NtQueryDirectoryFile** function can be used to enumerate files within the root mailslot file system directory (i.e., "\Device\Mailslot\"). All the standard **NT OS/2** information classes are supported. **NtOpenFile** is used to open the root mailslot directory.

5.8 Notify Change Directory

The **NtNotifyChangeDirectoryFile** function can be used to monitor modifications to the root mailslot file system directory. The standard **NT OS/2** capabilities are supported.

5.9 Query File Information

Information about a mailslot can be obtained with the **NtQueryInformationFile** function. Most information classes, not including extended attribute information, are supported for mailslots with special interpretation of the returned data as appropriate. An additional information class is also provided to return information that is specific to mailslots.

Information is returned by the mailslot file system for mailslots and for the mailslot root directory. The following subsections describe the information that is returned for mailslot entries. The information returned for the root directory is identical to the information that is returned by other file systems and is described in the **NT OS/2** I/O System Specification.

5.9.1 Basic Information

Basic information about a mailslot includes the creation time, the time of the last access, the time of the last write, the time of the last change, and the attributes of the mailslot. The file attribute value for a mailslot is **FILE_ATTRIBUTE_NORMAL**.

5.9.2 Standard Information

Standard information about a mailslot includes the allocation size, the end of file offset, the device type, the number of hard links, whether a delete is pending, and the directory indicator.

The allocation size is the amount of pool quota charged to the creator. The end of file offset is the number of bytes that are available in the buffer. The device type is **FILE_DEVICE_MAILSLOT**, the number of hard links is one, delete pending is TRUE, and the directory indicator is FALSE.

5.9.3 Internal Information

Internal information about a mailslot includes a mailslot file-system-specific identifier.

5.9.4 Extended Attribute Information

The extended attribute information size is always returned as zero by the mailslot file system.

5.9.5 Access Information

Access information about a mailslot includes the granted access flags.

5.9.6 Name Information

Name information about a mailslot includes the name of the mailslot.

5.9.7 Position Information

Position information about a mailslot includes the current byte offset. The current byte offset is the number of bytes that are available to be read in the mailslot buffer.

5.9.8 Mode Information

Mode information about a mailslot includes the I/O mode of the mailslot.

5.9.9 Alignment Information

The alignment information class is not supported by the mailslot file system.

5.9.10 All Information

The all information class includes information that can be returned by all file systems and is described above under each of the individual subsections.

5.9.11 Mailslot Information

Mailslot information on a mailslot includes: The quota charged for the mailslot buffer, the maximum message size. An access of FILE_READ_ATTRIBUTE is required to query the mailslot information of a mailslot.

FileMailslotQueryInformation - Data type is *FILE_MAILSLOT_QUERY_INFORMATION*.

```
typedef struct _FILE_MAILSLOT_QUERY_INFORMATION {
    ULONG MaximumMessageSize;
    ULONG MailslotQuota;
    ULONG NextMessageSize;
    ULONG MessagesAvailable;
    TIME ReadTimeout;
} FILE_MAILSLOT_QUERY_INFORMATION;
```

FILE_MAILSLOT_QUERY_INFORMATION:

MaximumMessageSize - The size, in bytes, of the largest message than can be written to the mailslot.

MailslotQuota - The amount of pool quota that is reserved for the mailslot buffer.

NextMessageSize - The size of the next message avaible in the mailslot. If no message is available a value of *MAILSLOT_NO_MESSAGE* is returned.

MessagesAvailable - The number of messages currently available at the mailslot.

ReadTimeout - The current read timeout for the mailslot. See **NtCreateMailslotFile** for a full description.

5.10 Set File Information

Information about a mailslot can be changed with the **NtSetInformationFile** function. Most information classes are supported for mailslots with the exception of link and position information.

Information can be set for mailslots. The following subsections describe the information that can be set for mailslots.

5.10.1 Basic Information

Basic information about a mailslot that can be set includes the creation time, the time of the last access, the time of the last write, the time of the last change, and the attributes of the mailslot.

The associated times included in this class can be set to any appropriate value. The file attribute field can only be set to **FILE_ATTRIBUTE_NORMAL**.

5.10.2 Disposition Information

The disposition information class is not supported by the mailslot file system.

Mailslots are always considered temporary and are deleted when the creator of the mailslot closes all of its handles.

5.10.3 Link Information

This information class is not supported by the mailslot file system.

5.10.4 Position Information

This information class is not supported by the mailslot file system.

5.10.5 Mode Information

Mode information about a mailslot that can be set includes the I/O mode of the mailslot.

5.10.6 Mailslot Information

Maislot information on a maillot that can be set includes: The read timeout.

FileMailslotSetInformation - Data type is *FILE_MAILSLOT_SET_INFORMATION*.

```
typedef struct _FILE_MAILSLOT_SET_INFORMATION {
    PTIME ReadTimeout;
} FILE_MAILSLOT_SET_INFORMATION;
```

FILE_MAILSLOT_SET_INFORMATION:

ReadTimeout - The read timeout for the mailslot. See **NtCreateMailslotFile** for more information.

5.11 Query Extended Attributes

This function is not supported by the mailslot file system.

5.12 Set Extended Attributes

This function is not supported by the mailslot file system.

5.13 Lock Byte Range

This function is not supported by the mailslot file system.

5.14 Unlock Byte Range

This function is not supported by the mailslot file system.

5.15 Query Volume Information

This function is not supported by the mailslot file system.

5.16 Set Volume Information

This function is not supported by the mailslot file system.

5.17 File Control Operations

The following subsections describe file control operations that can be performed using a handle that is open to mailslot. The peek function can only be executed using a handle that is open to the server end of a mailslot.

5.17.1 Peek

The peek file control operation reads data from a mailslot but does not actually remove the data. This operation may performed only using a server side handle to the mailslot.

The control code for this operation is *FSCTL_MAILSLOT_PEEK*.

This operation returns two buffers. The "input" buffer contains the parameter buffer for the peek operation. This buffer must be large enought to contain the strcuture specified below. The "output" buffer specifies the the data buffer. The parameter buffer has the following format:

```
typedef struct _FILE_MAILSLOT_PEEK_BUFFER {
    ULONG ReadDataAvailable;
    ULONG NumberOfMessages;
    ULONG MessageLength;
} FILE_MAILSLOT_PEEK_BUFFER;
```

FILE_MAILSLOT_PEEK_BUFFER:

ReadDataAvailable - The number of bytes of read data that are available in the mailslot buffer.

NumberOfMessages - The number of messages that are currently in the mailslot.

MessageLength - The number of bytes that are contained in the first message in the mailslot.

This function is similar to the **NtReadFile** function for a mailslot; however, no data is actually removed from the mailslot and the operation is always completed immediately, i.e., it never causes an I/O operation to be queued.

If the I/O status *STATUS_ACCESS_VIOLATION* is returned, then part of the output buffer became inaccessible after it was probed for write access.

If the I/O status *STATUS_BUFFER_OVERFLOW* is returned, then the peek I/O operation was completed successfully, but the size of the output buffer was not large enough to hold the entire input message. A full buffer of data is returned; the actual message size can be determined from information placed in the output buffer. The I/O status block contains the number of bytes that were read including the mailslot information.

If the I/O status *STATUS_SUCCESS* is returned, then the peek I/O operation was completed successfully and the I/O status block contains the number of bytes that were read including the mailslot information.

5.18 Flush Buffers

This function is not supported by the mailslot file system.

5.19 Set New File Size

This function is not supported by the mailslot file system.

5.20 Cancel I/O Operation

The **NtCancelIoFile** function can be used to cancel all I/O operations that were issued by the subject thread for the specified mailslot. Both read and write operations initiated by the subject thread are canceled.

5.21 Device Control Operations

No device control operations are supported by the mailslot file system.

5.22 Close Handle

The **NtClose** function can be used to close a handle to the specified mailslot.

If the specified handle is the last handle that is open to the server side of the specified mailslot, then the state of the mailslot is set to closing. Read and write operations that are pending are completed with an I/O status of *STATUS_PIPE_CLOSED*.

6. Win32 API Emulation

The following subsections discuss the emulation of the Win32 mailslot facilities using the capabilities provided by **NT OS/2**. Only those Win32 functions which require special handling with respect to mailslots are included.

6.1 CreateMailslot

This Win32 API creates a mailslot and opens a server side handle to the newly created object.

This API can be emulated with the **NtCreateMailslotFile** service.

The Win32 inheritance bit of the security attributes is the same as the **NT OS/2** handle attributes field of the object attributes parameter.

The Win32 access bits of the open mode are the same as the **NT OS/2** desired access parameter.

The Win32 message size is the same as the **NT OS/2** maximum message size.

The Win32 mailslot size is the same as the **NT OS/2** mailslot quota parameter.

6.2 GetMailslotInfo

This Win32 API obtains configuration and status information about the mailslot.

This API can be emulated with the **NtQueryInformationFile** service, with the information class *FileMailslotQueryInformation.*

6.3 SetMailslotInfo

This Win32 API set configuration information about the mailslot.

This API can be emulated with the **NtSetInformationFile** service, with the information class *FileMailslotSetInformation*.

Revision History:

Original Draft, December 28, 1990

Revision 1.1, January 5, 1991

1. Removed default read timeout and write send class.
2. Changed NtFsControlFile function FSCTL_MAILSLOT_WRITE to use separate input buffers for parameters and data.
3. Changed discussion of OS/2 API implementation to a discussion of Win32 API implementation.
4. Several editorial changes.

Revision 1.2, March 11, 1990

1. Added read timeout to NtCreateMailslotFile and made it queryable and settable.
2. Added new information class, FileMailslotSetInformation.
3. Removed message priorities, and the file system control function FSCTL_MAILSLOT_WRITE and FSCTL_MAILSLOT_READ.
4. Changed file system control function FSCTL_MAILSLOT_PEEK to use separate buffers to return parameters and data.
5. Update Win32 API discussion to conform with updated Win32 mailslot APIs.