

5.12 Set Extended Attributes	28
5.13 Lock Byte Range	28
5.14 Unlock Byte Range	28
5.15 Query Volume Information	28
5.16 Set Volume Information	28
5.17 File Control Operations	28
5.17.1 External File Control Operations	28
5.17.1.1 Assign Event	29
5.17.1.2 Disconnect	29
5.17.1.3 Listen	30
5.17.1.4 Peek	31
5.17.1.5 Query Event Information	32
5.17.1.6 Transceive	33
5.17.1.7 Wait For Named Pipe	34
5.17.1.8 Impersonate	35
5.17.2 Internal File Control Operations	36
5.17.2.1 Internal Read	36
5.17.2.2 Internal Write	36
5.17.2.3 Internal Transceive	36
5.18 Flush Buffers	36
5.19 Set New File Size	36
5.20 Cancel I/O Operation	37
5.21 Device Control Operations	37
5.22 Close Handle	37
6. OS/2 API Emulation	37
6.1 DosCallNmPipe	37
6.2 DosConnectNmPipe	37
6.3 DosDisconnectNmPipe	38
6.4 DosMakeNmPipe	38
6.5 DosPeekNmPipe	38
6.6 DosQNmPHandState	39
6.7 DosQNmPipeInfo	39
6.8 DosQNmPipeSemState	39
6.9 DosRawReadNmPipe	39
6.10 DosRawWriteNmPipe	39
6.11 DosSetNmPHandState	40
6.12 DosSetNmPipeSem	40
6.13 DosTransactNmPipe	40
6.14 DosWaitNmPipe	40

1. Introduction

This specification discusses the named pipe facilities of **NT OS/2**. Named pipes provide a full duplex interprocess communication (IPC) mechanism that can be used locally or across a network to access application servers. Named pipes provide the transport medium that is used for the Microsoft remote procedure call (RPC) capabilities.

Named pipes are used extensively by the OS/2 and LAN Manager components of the **NT OS/2** system, and therefore, must be implemented as efficiently as possible.

There are two manifestations of named pipes, those that are local to a system and those that are remote. This specification addresses both types of named pipes.

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

2. Goals

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

1. Provide the basic primitives necessary to compatibly emulate the OS/2 named pipe capabilities.
2. Provide protection and security attributes for named pipes 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 named pipes without having to enter the OS/2 subsystem.
4. Provide a fully qualified name space for named pipes that fits into the **NT OS/2** name structure in a straightforward manner.
5. Provide a high performance design and implementation of named pipes.

Although it is a major temptation, it is not a goal to "fix" the semantics of OS/2 named pipes. Minor discrepancies, however, will exist between OS/2 and **NT OS/2** named pipes where OS/2 capabilities or semantics are incompatible with those of **NT OS/2**, e.g., the named pipe naming and the asynchronous I/O model.

3. Overview of OS/2 Named Pipes

A named pipe provides a full duplex channel that can be used to implement an interprocess communication (IPC) mechanism between two processes. OS/2 uses named pipes to implement location-independent remote procedure call (RPC) capabilities and for communicating with servers on a remote system.

Named pipes have two ends: 1) a **client** end, and 2) a **server** end. Both ends are full duplex—data written from one end can be read from the other end and vice versa.

The server end of a named pipe is created when a new instance of a named pipe is created, or when a previously created instance is reused. A new instance of a named pipe is created with the **DosMakeNmPipe** API in OS/2.

Before either the **client** or the **server** ends of a named pipe can be used, the **server** end must be connected. In OS/2 this is accomplished with the **DosConnectNmPipe** API.

Once an instance of a named pipe is created and the **server** end is connected, then the **client** end of the named pipe can be created using the OS/2 **DosOpen** API.

When both the **server** end of a named pipe is connected and the **client** end is opened, information can flow over the pipe using the OS/2 **DosRead** and **DosWrite** APIs.

Named pipes are created with five attributes:

1. A pipe type which is either message or byte stream.
2. A count that limits the **maximum** number of simultaneous instances of the named pipe that can be created.
3. An input buffer size that specifies the size of the buffer that is used for inbound data on the **server** side of the named pipe.
4. An output buffer size that specifies the size of the buffer that is used for outbound data from the **server** side of the named pipe.
5. A default timeout value that is to be used if a timeout value is not specified when the **DosWaitNmPipe** API is executed.

The type of a named pipe determines how information is written into the named pipe. If the named pipe is a message pipe, then information is written into the pipe in the form of messages which include the byte count and the data of the message. If the named pipe is a byte stream pipe, then only the data is written into the named pipe.

The **maximum** instance count is established when the first instance of a specific named pipe is created (i.e., one of a given name) and cannot later be modified. Thereafter, up to the **maximum** instance count of simultaneous instances of the named pipe can be created to provide an IPC mechanism between any pair of processes.

The input and output buffer sizes are considered hints to the system for the sizes of the buffers that are needed to buffer inbound and outbound data. The actual buffer sizes may be either the system default or the specified buffer sizes rounded up to the next allocation boundary.

The default timeout value specifies a default for the amount of time that a client can wait for an available instance of a named pipe.

Once the first instance of a named pipe is created subsequent instances of an identically named pipe are subject to the `maximum instances` parameter. In addition, the type of pipe and the default timeout value are ignored and cannot be set when subsequent instances of the named pipe are created.

In addition to the five attribute parameters, two mode parameters can be specified when an instance of a named pipe is created or opened:

1. The read mode, which can be either message mode or byte stream mode, but which must be compatible with the type of the named pipe.
2. The blocking mode, which can be either blocking or nonblocking.

The read mode of a named pipe determines how data will be read from the pipe. If the named pipe is a message pipe, then data can be read in either message mode or byte stream mode. However, if the named pipe is a byte stream pipe, then data can only be read in byte stream mode.

The blocking mode determines what happens when a request cannot be satisfied immediately. If the mode is blocking, then an implied wait occurs until an operation is completed. Otherwise, the operation returns immediately with an error status.

Standard open parameters can also be specified when an instance of a named pipe is created or opened which define the access that is desired to the named pipe (e.g., read only, write only, or read/write access), whether the named pipe handle is inherited when a child process is created, and whether write behind is allowed on writes to the named pipe.

The open access parameters also specify the configuration of the named pipe when the first instance of a named pipe is created. A named pipe can have a full duplex or a simplex configuration. A full duplex named pipe allows data to flow in both directions, whereas a simplex named pipe only allows data to flow in one direction. The direction of data flow and configuration are determined by the read only (outbound), write only (inbound), and read/write (full duplex) open access parameters specified by the server when the first instance of a named pipe is created.

The server end of a named pipe can be reused by disconnecting the client end. In OS/2, this is accomplished using the **DosDisconnectNmPipe** API. The server end of a named pipe can also be disconnected by closing the respective file handle, but this deletes the instance of the named pipe and it cannot be reused.

The client end of a named pipe is disconnected by simply closing the respective file handle.

OS/2 supplies 14 APIs that are specific to named pipes. These APIs are intended mainly for use by a server. In addition, eleven standard OS/2 I/O system APIs can be executed using a file handle to a named pipe.

4. Overview of NT OS/2 Named Pipes

4.1 Implementation Alternatives

Named pipes must be integrated into the **NT OS/2** I/O system such that standard read and write requests can be used to read data from and write data to a named pipe. It also must be possible to accomplish LAN Manager server and client redirection of named pipes without having to call the OS/2 subsystem.

There are several ways of integrating named pipes into **NT OS/2** that meet these requirements:

1. Implement the named pipe capabilities as an installable file system and extend **NtCreateFile** so that the named pipe attributes required by OS/2 can be specified directly in the **NT OS/2** system service call.
2. Implement the named pipe capabilities as an installable file system and use extended attributes as the means of defining the named pipe attributes required by OS/2.
3. Implement named pipes as a separate object that is created with its own API, but which can be opened via a pipe driver.
4. Implement the named pipe capabilities as an installable file system and add an **NT OS/2** I/O system API that specifically creates an instance of a named pipe.

The first alternative requires an already complicated API to be further extended to accommodate yet another special case.

The second alternative overloads the use of extended attributes to have a special meaning for named pipes. Extended attributes are not the most efficient or convenient way of specifying the attribute values and would require special rules about when they could be read and written.

The third alternative would create a nonstandard object whose API was partly buried in the I/O system and partly in object-specific APIs.

The fourth alternative adds an additional API to the **NT OS/2** I/O system that has special meaning and is only applicable to named pipes.

The fourth alternative has been chosen as the means of implementing the named pipe capabilities in **NT OS/2**. Although this provides an additional I/O system API that is specific to named pipes, it is the most straightforward and efficient implementation.

4.2 Named Pipe Directories

In OS/2, named pipes have a rigid name syntax with the following form:

\PIPE\pipe-name

This syntax is recognized by the OS/2 **DosOpen** API and is routed to the appropriate system component. The LAN Manager redirector is also capable of recognizing names of the following form:

`\\server-name\PIPE\pipe-name`

The redirector transforms the request into a tree connection to a server and then performs the appropriate SMB generation.

The **NT OS/2** named pipe driver will also implement a flat name space. The syntax for an **NT OS/2** named pipe is of the following form:

`\Device\NamedPipe\pipe-name1`

\The object name space in NT OS/2 is more general and hierarchical, and we would like named pipes to follow that scheme; however, because of issues involving persistent named pipes, and guaranteeing proper behavior given reparse the first named pipe driver will use a flat name space. Once the issues are resolved named pipes can be extended to existing file systems as a special file using reparse or by maintaining a named pipe database in a system file.

The syntax for a remote **NT OS/2** named pipe is of the following form:

`\Device\LanmanRedirector\server-name\Pipe\pipe-name`

4.3 Read/Write Buffering Strategy

4.3.1 OS/2 Read/Write Buffering Strategy

The OS/2 named pipe capabilities use a two circular buffers for buffering inbound and outbound writes to a named pipe. This design is dictated by the synchronous I/O model of OS/2 and it controls the amount of system buffering space that is consumed. The data is copied twice for each write and read of a named pipe. One copy occurs when the data is written from a user buffer into a named pipe and another copy occurs when the data is read out of the named pipe into a user buffer.

An OS/2 named pipe can be either a message or byte stream pipe, which determines how write data is stored in the pipe buffers. Message pipes can be read in either message mode or byte stream mode. Byte stream pipes can only be read in byte stream mode. In addition, a blocking mode can be specified for each open of an instance of a named pipe. The blocking mode determines whether reads from, and writes to the named pipe block if sufficient data or space is not available in the named pipe.

¹The string "`\Device\NamedPipe`" refers to the named pipe driver, while the string "`\Device\NamedPipe\`" represents the root directory of the named pipe file system.

A message named pipe stores the size of a message and the data for the message. A byte stream named pipe simply stores the data and no additional information. Reads from a message named pipe attempt to read a complete message from the pipe in either message mode or byte stream mode. If the complete message does not fit in the supplied read buffer, then a full buffer is returned along with an error status that signifies that there is more data in the message. Reads from a byte stream named pipe can only be made in byte stream mode and return the data that is currently in the pipe up to the size of the supplied buffer.

Each inbound and outbound buffer for a named pipe has a read lock, a write lock, a read semaphore, and a write semaphore. These are used to synchronize the reading and writing of data to and from the buffer.

When a write to a named pipe buffer begins, the write lock is acquired to prevent any other writer from writing into the buffer until the current write is finished. If the write must block because of a lack of available space in the buffer, then the reader semaphore is signaled, the writer lock is not released, and the writer waits for a reader to signal the write semaphore. The write lock is released at the completion of the write operation.

The following describes the OS/2 named pipe write logic.

```
if (message pipe) then
  if (blocking mode) then
    write message to pipe, synchronize with reader
    return size of message written
  else
    if (space for message header plus one byte) and
      (data buffer size greater than pipe buffer size) then
      write data to pipe, synchronize with reader
      return size of message written
    else
      if (space for data buffer and message header) then
        write data to pipe
        return size of message written
      else
        return buffer overflow error
      endif
    endif
  endif
else
  if (blocking mode) then
    write data to pipe, synchronize with reader
    return count of bytes written
  else
    if (space available in pipe buffer) then
      write data to pipe (minimum data buffer/pipe space)
      return count of bytes written
    else
      return buffer overflow error
    endif
  endif
endif
```

When a read from a named pipe buffer begins, the read lock is acquired to prevent any other reader from reading from the buffer until the current read is finished. If the read must block because of a lack of available data in the buffer, then the writer semaphore is signaled, the reader lock is not released, and the reader waits for the read semaphore to be signaled. The read lock is released at the completion of the read operation.

The following describes the OS/2 named pipe read logic.

```

while (data not available in pipe) do
    if (blocking mode) then
        wait for available data in pipe
    else
        return no data available error
    endif
endwhile

if (message pipe) then
    if (data buffer size greater or equal message size) then
        if (message mode) then
            read message from pipe, synchronize with writer
            return size of message read
        else
            read available data or message from pipe
            if (complete message read) then
                return size of message read
            else
                reduce size of message by available data bytes
                return count of data bytes read
            endif
        endif
    else
        if (message mode) then
            read data from pipe, synchronize with writer
            reduce size of message by data buffer size
            return more data error
        else
            read available data from pipe
            reduce size of message by available data bytes
            return count of data bytes read
        endif
    endif
else
    read available data from pipe
    return count of data bytes read
endif

```

4.3.2 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. In addition, **NT OS/2** supports I/O transfers that are buffered by the system rather than requiring buffers to be locked down and nonswappable. Therefore, the buffering scheme used for the **NT OS/2** implementation of named pipes differs markedly from that of OS/2.

The blocking mode of OS/2 is emulated in **NT OS/2** with a completion mode. The completion mode can be specified such that read and write operations are completed immediately or they are queued and subject to completion when space is available or data is present.

The inbound and outbound buffers for a named pipe are not actually allocated to real memory in **NT OS/2**. Instead, the creator of an instance of a named pipe is simply charged memory quota for these buffers. Writers and readers 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 and a write request is queued rather than completed immediately, then the writer is charged for any additional quota that is required. Likewise, a reader is charged quota if no data is available, the quota charged to the creator is exhausted, and the read request is queued rather than completed immediately.

The named pipe capabilities of **NT OS/2** requires that the data be copied twice. However, reads can "pull" data from write buffers and writes can "push" data into read buffers.

The following is a somewhat simplified discussion of the buffering scheme used for named pipes in **NT OS/2**. Boundary conditions and differing pipe types and modes are not considered. The pipe type is assumed to be message, the read mode is assumed to be message, and the completion mode is assumed to be queued operation.

The exact behavior of the **NT OS/2** named pipe buffering depends on whether a read occurs before a write or vice versa.

If a write operation occurs before a read operation, then 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 named pipe 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 first-in-first-out list of writers. If quota was not charged to the writer, then the writer's I/O request can be completed immediately. The system buffer will be deallocated and the creator's quota returned when a matching read arrives. Otherwise, the write request type is converted to a buffered request so that upon completion, the I/O system will deallocate the system buffer and return memory quota as appropriate.

At this point, the I/O operation is either pending or has been completed and control is returned to the caller. If another write request is received before the first set of write data is read, 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 read request arrives at the read end of the pipe and it is determined that write data is available at the write end of the pipe. The 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. If the original write I/O request was completed at the time of the write, then the system buffer is deallocated and memory quota is returned for named pipe write buffering. However, if the write I/O was not completed at the time of the write, then completion of the write requires writing the I/O status block, setting the completion event, deallocating the system buffer, and returning quota to the writer.

If an access violation occurs during a copy from the output buffer to a system buffer, then the write operation is immediately terminated. Previously completed read I/O requests, if any, are not backed out. 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.

The following describes the **NT OS/2** named pipe write logic.

```

probe output buffer for read access
while (read pending) and (output buffer size not zero) do
    copy data from output buffer to read buffer
    if (read buffer greater or equal output buffer) then
        reduce output buffer size to zero
        set read I/O status to successful completion
    else
        reduce output buffer by read buffer size
        if (message mode read) then
            set read I/O status to buffer overflow
        else
            set read I/O status to successful completion
        endif
    endif
    remove read request from read pending list
    complete read I/O operation, return quota
endwhile
if (output buffer size not zero) then
    if (pipe quota available) then
        allocate write buffer, charge quota to pipe
        copy data from output buffer to write buffer
        insert write request in write pending list
        set write I/O status to successful completion
        complete write I/O operation
        return successful completion
    else
        if (queued operation) or ((message pipe) and
            (output buffer not original size)) then
            allocate write buffer, charge quota to writer
            copy data from output buffer to write buffer
            insert write request in write pending list
            return operation pending
        else
            if (output buffer not original size) then
                set write I/O status to successful completion
                complete write I/O request
                return successful completion
            else
                abort write I/O operation
                return no space available
            endif
        endif
    endif
    set write I/O status to successful completion
    complete write I/O request
    return successful completion
endif

```

以上内容仅为本文档的试下载部分，为可阅读页数的一半内容。如要下载或阅读全文，请访问：<https://d.book118.com/388042142070006032>