Portable Systems Group

OS/2 Emulation Subsystem Specification

Author: *Steven R. Wood*

Revision 1.0, January 19, 1990
Original Draft August 15, 1989

1. Overview

This specification describes the design and implementation of the OS/2 Emulation Subsystem for NT OS/2. The subsystem consists of a dynamic link library (DLL) that resides in the OS/2 application's address space, a server process that maintains global state across all OS/2 applications and an NT OS/2 Kernel Extension that runs in Kernel Mode and implements the OS/2 Semaphore primitives.

The DLL exports as entry points all of the 32 bit Dos32 API's defined by OS/2 Version 2.0 (Cruiser). Some of the entry points, that only manipulate process private state are implemented entirely in the DLL. Others call the OS/2 Subsystem server to access and/or modify the global state it maintains. Finally, the 32 bit Dos Semaphore API's call the OS/2 Subsystem Kernel Extension.

For the remainder of this document, these three components will be referred to as: the OS/2 Server, the OS/2 DLL and the OS/2 Kernel Extension.

1.1 OS/2 DLL State

The OS/2 DLL maintains the following information for each OS/2 process:

- Current drive
- Current directory for each drive
- Environment variables
- Command line
- OS/2 File handle table
- Hard Error and Verify flags
- Thread Information Block (TIB) for each thread

1.2 OS/2 Server State

The OS/2 Server maintains the following state:

- Hierarchy of OS/2 processes
- List of threads for each process

- Exit list procedures for each process
- Shadow of file handle table for each process
- Queues, Pipes and Shared memory objects
- Keyboard buffer for Dos32Read calls to Standard Input
- Listen Thread that listens for connection requests from OS/2 applications.
- Keyboard Thread that is waiting on a Presentation Manager message queue for keyboard events. This message queue is associated with any character mode window. In the current OS/2 1.1 implementation, this thread runs in the task manager process.
- Request Threads. The number of request threads will vary dynamically based on the number of outstanding connections to OS/2 applications. The exact ratio will be determined during performance analysis.
- Exception Port Thread that is waiting for exceptions for OS/2 application threads that were not handled.
- Session Manager Thread that is dedicated to servicing requests from the NT OS/2 Session Manager

1.3 OS/2 Kernel Extension State

The OS/2 Kernel Extension maintains the following state:

- OS/2 Event Semaphore objects
- OS/2 Mutex Semaphore objects
- OS/2 MuxWait Semaphore objects

1.4 Process Structure

The OS/2 Server is responsible for creating all OS/2 processes and maintain a process tree structure that describes the relationship between OS/2 processes. For each process the following information is maintained:

- OS/2 PID value

- NT OS/2 Process Handle
- Parent process
- Sibling process list
- Child process list
- Thread Table
- File Handle Table

Process creation is the result of one of several external events:

- an OS/2 application calls Dos32ExecPgm
- an OS/2 application calls Dos32StartSession
- the NT OS/2 Session Manager calls the OS/2 Server to start an OS/2 application.
- opens the image file
- creates a process with that image file mapped
- extracts the entry address and program type from the image header
- allocates a stack and fills in the TEB with the stack bounds
- creates a suspended thread with an initial context that points to the correct entry address and stack
- Client Id
- Process and Thread handles
- Type of image file

If the type of the image is not OS/2, then the OS/2 Server will pass the information returned by SmCreateImageFileProcess back to the Session Manager and allow it to communicate the information to the appropriate subsystem (e.g. Posix). When this happens, a node is still created in the OS/2 process structure so that the foreign process has a valid process Id in the OS/2 world.

Finally, the OS/2 Server can be called by the Session Manager with an OS/2 process that was created by another subsystem calling the SmCreateImageFileProcess routine. In this case the OS/2 Server will add the process as a top level OS/2 process whose parent process is the dummy process at the root of the OS/2 process tree.

Threads within an OS/2 process are also created and managed by the OS/2 Server. The server will maintain a doubly linked list of all the threads created by the client calling the Dos32CreateThread API within a given OS/2 process. For each thread, the following information will be maintained:

- Thread list pointers
- Client Id
- OS/2 TID value
- NT OS/2 Thread Handle
- Address of OS/2 TIB in client's address space
- Address of NT OS/2 TEB in client's address space

1.5 Name Processing

All file name parsing occurs in the OS/2 DLL. It maintains the following information in the address space of each OS/2 process:

- Current Drive
- Current Directory for each drive

```
\OS2\Drives\A: => \Device\Floppy1
\OS2\Drives\B: => \Device\Floppy2
\OS2\Drives\C: => \Device\SCSI0
\"LogonDirectory"\OS2\Drives\A: => \OS2\Drives\A:
\"LogonDirectory"\OS2\Drives\B: => \OS2\Drives\B:
\"LogonDirectory"\OS2\Drives\C: => \OS2\Drives\C:
\"LogonDirectory"\OS2\Drives\D: => \"LogonDirectory"\Net\Portasys
```

The double level of indirection is to allow separation of network connections between logon sessions. In order to map an OS/2 file name, into an NT OS/2 file name, the following logic will be performed by the OS/2 DLL:

- If no drive letter, supply current drive from process state.
- If first character after drive letter, colon is not a path separator, then supply current directory for the drive letter from process state.
- Scan the remainder of the file name, removing any relative path specifiers (. and ..) by shifting file name characters left and removing path separators.
- At the same time convert any forward slash (/) path separators to back slashes (\).
- Finally, insert the \"LogonDirectory"\OS2\Drives\ string at the front of the file name.

When querying a name from NT OS/2, a reverse of some of the logic above needs to be performed. Since the only API calls that return path names are the FindFirst and FindNext, the FindFirst code can cache the user specified path name so that it and FindNext can use it to format the return buffer. This prevents the OS/2 DLL from having to decode the reverse symbolic link path that leads from \Device\SCSI0 to C:

1.6 File Handle Processing

The OS/2 Server will maintain the OS/2 File Handle table in its process state. The file handle table will be indexed by OS/2 File Handles, which are small integers, starting from 0 and going to some maximum amount. The OS/2 DLL will impose no limit on the number of file handles, other than available memory for the file handle table. The OS/2 DLL will allocate chunks of memory that hold 64 file handles. If more than 64 file handles are created, then two chunks will be allocated, one to hold the second group of file handles and another to act as a layer of indirection that leads to either the first or second chunks of file handles.

For each file handle, the following information is maintained:

- NT OS/2 File Handle
- Flags
- Handler

The handler associated with each file handle will enable the API stubs to dispatch to the appropriate code based on the type of the file handle (NT OS/2 File Handle, OS/2 Pipe Handle, etc.).

1.7 32 Bit OS/2 API Summary

Below is a complete list of all the 32-Bit OS/2 API calls supported by OS/2 2.0 (aka Cruiser). For each call, it is identified whether the call is implemented in the OS/2 Server, the OS/2 DLL, the OS/2 Kernel Extension or not implemented. In the case of calls implemented in the OS/2 Server, there is also work done in the OS/2 DLL to prepare the parameters for the server and to process the results from the call to the server.

Dos32QuerySysInfo	DLL
Dos32Error	DLL
Dos32CreateThread	Server
Dos32WaitChild	Server
Dos32WaitThread	Server
Dos32EnterCritSec	Server
Dos32ExitCritSec	Server
Dos32ExecPgm	Server
Dos32Exit	Server
Dos32ExitList	Server
Dos32GetThreadInfo	DLL
Dos32SetPriority	DLL
Dos32KillProcess	Server
Dos32ResumeThread	Server
Dos32SuspendThread	Server
Dos32CreatePipe	Server
Dos32CallNPipe	Server
Dos32ConnectNPipe	Server
Dos32DisConnectNPipe	Server
Dos32CreateNPipe	Server
Dos32PeekNPipe	Server
Dos32QueryNPHState	Server
Dos32QueryNPipeInfo	Server
Dos32QueryNPipeSemState	Server
Dos32RawReadNPipe	Server
Dos32RawWriteNPipe	Server
Dos32SetNPHState	Server
Dos32SetNPipeSem	Server
Dos32TransactNPipe	Server
Dos32WaitNPipe	Server
Dos32CreateQueue	Server
Dos32OpenQueue	Server
Dos32CloseQueue	Server
Dos32PeekQueue	Server
Dos32PurgeQueue	Server
Dos32QueryQueue	Server
Dos32ReadQueue	Server
Dos32WriteQueue	Server
Dos32CreateEventSem	Kernel Extension
Dos32OpenEventSem	Kernel Extension
Dos32CloseEventSem	Kernel Extension
Dos32ResetEventSem	Kernel Extension
Dos32PostEventSem	Kernel Extension
Dos32WaitEventSem	Kernel Extension
Dos32QueryEventSem	Kernel Extension
Dos32CreateMutexSem	Kernel Extension
Dos32OpenMutexSem	Kernel Extension
Dos32CloseMutexSem	Kernel Extension
Dos32RequestMutexSem	Kernel Extension
Dos32ReleaseMutexSem	Kernel Extension
Dos32QueryMutexSem	Kernel Extension
Dos32CreateMuxWaitSem	Kernel Extension
Dos32OpenMuxWaitSem	Kernel Extension

Dos32CloseMuxWaitSem	Kernel Extension
Dos32WaitMuxWaitSem	Kernel Extension
Dos32AddMuxWaitSem	Kernel Extension
Dos32DeleteMuxWaitSem	Kernel Extension
Dos32QueryMuxWaitSem	Kernel Extension
Dos32GetDateTime	DLL
Dos32SetDateTime	DLL
Dos32Sleep	DLL
Dos32AsyncTimer	DLL
Dos32StartTimer	DLL
Dos32StopTimer	DLL
Dos32AliasMem	not implemented
Dos32AllocMem	DLL
Dos32AllocSharedMem	Server
Dos32GetNamedSharedMem	Server
Dos32GetSharedMem	Server
Dos32GiveSharedMem	Server
Dos32FreeMem	DLL
Dos32SetMem	DLL
Dos32QueryMemState	not implemented
Dos32QueryMem	DLL
Dos32SubAlloc	DLL
Dos32SubFree	DLL
Dos32SubSet	DLL
Dos32LoadModule	Server
Dos32FreeModule	Server

Dos32QueryProcAddr	Server
Dos32QueryModuleHandle	Server
Dos32QueryModuleName	Server
Dos32GetResource	Server
Dos32QueryAppType	Server
Dos32Beep	DLL
Dos32DevConfig	DLL
Dos32PhysicalDisk	not implemented
Dos32ScanEnv	DLL
Dos32SearchPath	DLL
Dos32QueryVerify	DLL
Dos32SetVerify	DLL
Dos32SetMaxFH	DLL
Dos32Open	Server
Dos32SetFHState	DLL
Dos32QueryFHState	DLL
Dos32QueryHType	DLL
Dos32QueryFileMode	DLL
Dos32SetFileMode	DLL
Dos32SetFileInfo	DLL
Dos32QueryFileInfo	DLL

Dos32ResetBuffer	DLL
Dos32SetFilePtr	DLL
Dos32Read	DLL
Dos32Write	DLL
Dos32Close	Server
Dos32DevIOCtl	not implemented
Dos32DupHandle	Server
Dos32FileIO	DLL
Dos32SetFileLocks	DLL
Dos32SetFileSize	DLL
Dos32FindFirst	DLL
Dos32FindNext	DLL
Dos32FindClose	DLL
Dos32FindNotifyFirst	DLL
Dos32FindNotifyNext	DLL
Dos32FindNotifyClose	DLL
Dos32SetDefaultDisk	DLL
Dos32QueryCurrentDisk	DLL
Dos32SetCurrentDir	DLL
Dos32QueryCurrentDir	DLL
Dos32Delete	DLL

Dos32EditName	DLL
Dos32QueryPathInfo	DLL
Dos32SetPathInfo	DLL
Dos32SetCurrentDir	DLL
Dos32CreateDir	DLL
Dos32DeleteDir	DLL
Dos32Move	DLL
Dos32Copy	DLL
Dos32FSAttach	not implemented
Dos32FSCtl	not implemented
Dos32QueryFSAttach	not implemented
Dos32SetFSInfo	not implemented
Dos32QueryFSInfo	not implemented
Dos32GetMessage	DLL
Dos32InsertMessage	DLL
Dos32PutMessage	DLL
Dos32SetProcessCp	DLL
Dos32QueryCp	DLL
Dos32QueryCtryInfo	DLL
Dos32QueryDBCSEnv	DLL
Dos32QueryCollate	DLL
Dos32MapCase	DLL
Dos32StartSession	Server
Dos32SetSession	Server
Dos32SelectSession	Server
Dos32StopSession	Server
Dos32SetExceptionHandler	DLL
Dos32UnsetExceptionHandler	DLL
Dos32RaiseException	DLL
Dos32UnwindException	DLL
Dos32SendException	eliminated(D658)
Dos32FlagProcess	eliminated(D658)
Dos32ErrClass	DLL

1.8 Rationale for Not Implemented OS/2 API Calls

Dos32QueryMemState is an internal API added for Component Test and performance testing. It is not part of the OS/2 2.0 API, even though it appears in BSEDOS.H.

Dos32AliasMem is an internal API added to support the 32 to 16 bit thunk code. It is not part of the OS/2 2.0 API, even though it appears in BSEDOS.H.

The five Installable File System calls: Dos32FSAttach, Dos32FSCtl, Dos32QueryFSAttach, Dos32SetFSInfo and Dos32QueryFSInfo are not implemented because Portable OS/2 is not compatible with existing IFS implementations.

Dos32DevIOCtl is not implemented because Portable OS/2 is not compatible with existing OS/2 device drivers. In addition, the Dos32DevIOCtl API in OS/2 V2.0 is only specified to work with 16 bit device drivers.

Dos32PhysicalDisk is not implemented because it provides a means for accessing the physical media via Dos32DevIOCtl calls, which is not implemented for Portable OS/2. We made need to support the ability of the Dos32PhysicalDisk API to return partition information for a drive, but for now there is no plan to do so.