OS/2 2.0 Technical Library

Control Program Programming Reference

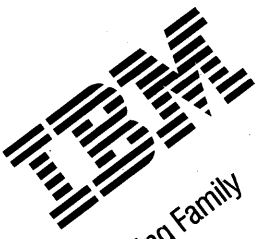
Version 2.00

IBM Programming Family

OS/2 2.0 Technical Library

Control Program Programming Reference

Version 2.00



Programming Family

Note

Before using this information and the product it supports, be sure to read the general information under "Notices" on page iii.

First Edition (March 1992)

The following paragraph does not apply to the United Kingdom or any country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This publication could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time.

It is possible that this publication may contain reference to, or information about, IBM products (machines and programs), programming, or services that are not announced in your country. Such references or information must not be construed to mean that IBM intends to announce such IBM products, programming, or services in your country.

Requests for technical information about IBM products should be made to your IBM Authorized Dealer or your IBM Marketing Representative.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Commercial Relations, IBM Corporation, Purchase, NY 10577.

COPYRIGHT LICENSE: This publication contains printed sample application programs in source language, which illustrate OS/2 programming techniques. You may copy and distribute these sample programs in any form without payment tc IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the OS/2 application programming interface.

Each copy of any portion of these sample programs or any derivative work, which is distributed to others, must include a copyright notice as follows: "© (your company name) (year) All Rights Reserved."

Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any of IBM's intellectual property rights or other legally protectible rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, programs, or services, except those expressly designated by IBM, are the user's responsibility.

Trademarks

The following terms, denoted by an asterisk (*) in this publication, are trademarks of the IBM Corporation in the United States and/or other countries:

AT
Common User Access
CUA
IBM
IBM PC AT
Operating System/2
OS/2
Personal System/2
Presentation Manager
PS/2
SAA
Systems Application Architecture

The following terms, denoted by a double asterisk (**) in this publication, are trademarks of another company as follows:

Intel	Intel Corporation
80386	Intel Corporation
80386SX	Intel Corporation
80387	Intel Corporation
80387 NPX	Intel Corporation
80486	Intel Corporation

Double-Byte Character Set (DBCS)

Throughout this publication, you will see references to specific values for character strings. The values are for the single-byte character set (SBCS). If you use the double-byte character set (DBCS), note that one DBCS character equals two SBCS characters.

Preface

About this Book

The OS/2 2.0 *Control Program Programming Reference* is a detailed technical reference for application programmers creating programs using OS/2 system functions. These functions are also called the control program functions. The control program functions carry out such tasks as allocating memory and performing file operations.

The reference does not give guidance on how to use the functions, nor does it contain information about how the functions are related to each other. It is intended to be used in conjunction with the OS/2 2.0 *Programming Guide*.

Prerequisite Knowledge

The OS/2 2.0 Technical Library is intended for professional application developers knowledgeable in at least one programming language in which OS/2 programs can be written. The information in the Technical Library assumes that you are new to programming with OS/2 and the Presentation Manager. You should understand the OS/2 services available to users.

Related Publications

The OS/2 2.0 *Getting Started* manual introduces the programming concepts that you should understand before you begin developing applications to run on an OS/2 system, and describes the set of books, tools, programming aids, and sample programs that make up the OS/2 2.0 Developer's Toolkit.

Organization of this Book

Contents

Chapter 1. Introduction

The purpose of this reference is to provide information about control program functions, parameters, return codes, and constants of the OS/2* 2.0 operating system. This chapter provides information about the notation conventions and function descriptions used in this reference.

The functions can be used in full-screen and Presentation Manager* sessions to perform basic operating-system functions, such as file input and output, memory allocation, and thread and process creation, control, and communication.

An example in C language is shown at the end of each function.

Conventions Used in Function Descriptions

The documentation of each function contains these sections:

Function name
The function name is listed in alphabetical order at the top of each page, followed by a brief description of the function.

Parameters
Each parameter is listed with its C language data type, parameter type, and a brief description.

- All data types are written in uppercase. A data type of 'Pxxxxxxx' implicitly defines a pointer to the data type 'xxxxxxx'.

  The term NULL applied to a parameter indicates the presence of the parameter, but with no value.

- There are three parameter types:

  Input
  Specified by the programmer.

  Output
  Returned by the operating system.

  Input/Output
  Specified by the programmer and modified by the operating system.

- A brief description is provided with each parameter. Where appropriate, restrictions are also included. In some cases, the parameter points to a structure.

Returns
A list of possible return codes or errors (when appropriate) is included in this section. Some functions do not have return codes. Refer to Appendix B, "Errors" on page B-1 for a complete list of all return codes and their descriptions.

Remarks
This section contains additional information about the function, when required.

Related Functions
This list shows the functions (if any) that are related to the function being described.

Example Code
An example is shown in C language for each function.

Programming Note: The functions in this book are named in mixed-case for readability, but are known to the system as uppercase character strings. If you are using a compiler that generates a mixed-case external name, you should code the functions in uppercase.

---

* Trademark of the IBM Corporation.

Programming Considerations

The Presentation Manager component of the Operating System/2* system is based on the IBM* Systems Application Architecture* (SAA*) Common Programming Interface – a software interface for the design and development of applications.

The Presentation Manager component implements the Common User Access* (CUA*) interface, which you can use to attain consistency in the appearance and behavior of you applications on IBM Personal System/2* (PS/2*) systems.

The operating system supports the addressing capabilities of the Intel** 80386**, 80386SX**, 80387**, 80387 NPX**, and 80486** processors, with page-level memory protection. One page is 4KB (KB equals 1024 bytes) of contiguous physical memory.

** Trademark of the Intel Corporation.

Chapter 2. Control Program Functions

DosAcknowledgeSignalException – Acknowledge Signal Exception

#define INCL_DOSEXCEPTIONS

APIRET DosAcknowledgeSignalException (ULONG ulSignalNumber)

DosAcknowledgeSignalException indicates that a process wants to receive further signals.

Parameters

ulSignalNumber (ULONG) – input

The number of the signal to be acknowledged. Valid signals are:

Number	Signal
1	XCPT_SIGNAL_INTR
3	XCPT_SIGNAL_KILLPROC
4	XCPT_SIGNAL_BREAK

Returns

Return Code.

DosAcknowledgeSignalException returns the following values:

0	NO_ERROR
209	ERROR_INVALID_SIGNAL_NUMBER

Remarks

DosAcknowledgeSignalException is used to tell the system that the process wishes to receive further signal exceptions.

This function may be used by any thread in the process, but will only affect thread 1.

See Appendix C, "System Exceptions" on page C-1 for a detailed list of the system exceptions.

Related Functions

- DosEnterMustComplete
- DosExitMustComplete
- DosRaiseException
- DosSendSignalException
- DosSetExceptionHandler
- DosSetSignalExceptionFocus
- DosUnsetExceptionHandler
- DosUnwindException

Example Code

This example shows how a thread can indicate that it wants to receive new signals. Typically, this function would be issued from within a signal handling routine.

```
#define INCL_DOSEXCEPTIONS   /* Exception values */
#include <os2.h>
#include <stdio.h>

ULONG   ulSignalNum;  /* Number of signal to be acknowledged */
APIRET  rc;           /* Return code */

   ulSignalNum = XCPT_SIGNAL_INTR;
                      /* Register for new Ctrl-C signals */

   rc = DosAcknowledgeSignalException(ulSignalNum);

   if (rc != 0)
     {
       printf("DosAcknowledgeSignalException error: return code = %ld",
                     rc);
     }
```

DosAddMuxWaitSem – Add MuxWait Semaphore

#define INCL_DOSSEMAPHORES

APIRET DosAddMuxWaitSem (HMUX hmux, PSEMRECORD ppSemRec)

DosAddMuxWaitSem adds a mutex semaphore or an event semaphore to a muxwait-semaphore list.

Parameters

hmux (HMUX) – input

The handle of the muxwait semaphore that is to receive the additional semaphore.

ppSemRec (PSEMRECORD) – input

A pointer to the semaphore record that is to be added to the muxwait list.

Returns

Return Code.

DosAddMuxWaitSem returns the following values:

0	NO_ERROR
6	ERROR_INVALID_HANDLE
8	ERROR_NOT_ENOUGH_MEMORY
87	ERROR_INVALID_PARAMETER
100	ERROR_TOO_MANY_SEMAPHORES
105	ERROR_SEM_OWNER_DIED
284	ERROR_DUPLICATE_HANDLE
292	ERROR_WRONG_TYPE

Remarks

The Semaph.C sample program demonstrates the use of mutex, event, and muxwait semaphores. In the application, several threads are sharing access to the same resource:

- A mutex (mutual exclusion) semaphore is used to ensure that only one thread at a time has access to the resource.
- An event semaphore is used to signal a thread to give up the resource. The event can be posted by the user, or posted by the application at fixed time intervals.
- A muxwait (multiple wait) semaphore is used to check for a stop event or a user signal to give up the resource.

The Semaph.C sample program is in the C:\TOOLKT20\C\SAMPLES\SEMAPH directory.

Related Functions

- DosCloseMutexSem
- DosCreateMuxWaitSem
- DosDeleteMuxWaitSem
- DosOpenMuxWaitSem
- DosQueryMuxWaitSem
- DosWaitMuxWaitSem

Example Code

This example adds a new event semaphore to an existing muxwait semaphore. Assume that the handle of the muxwait semaphore has been placed into *hmux* already. Assume that the handle of the event semaphore has been placed into *hev* already, and that the corresponding user identifier value has been placed into *ulUser*.

```
#define INCL_DOSSEMAPHORES   /* Semaphore values */
#include <os2.h>
#include <stdio.h>

HMUX        hmux;     /* Muxwait semaphore handle */
SEMRECORD   SemRec;   /* Muxwait semaphore list entry to add
                         to semaphore list */
HEV         hev;      /* Event semaphore handle */
ULONG       ulUser;   /* User sempahore ID value */
APIRET      rc;       /* Return code */

   SemRec.hsemCur = (PVOID) hev;  /* Initialize the semaphore */
   SemRec.ulUser = ulUser;        /*   list entry to pass to  */
                                  /*   DosAddMuxWait          */

   rc = DosAddMuxWaitSem(hmux, &SemRec);

   if (rc != 0)
     {
       printf("DosAddMuxWaitSem error: return code = %ld", rc);
       return;
     }
```

DosAllocMem –
Allocate a Private Memory Object Memory

```
#define INCL_DOSMEMMGR
```

APIRET DosAllocMem (PPVOID ppBaseAddress, ULONG ulObjectSize,
ULONG ulAllocationFlags)

DosAllocMem allocates a private memory object within the virtual-address space.

Parameters

ppBaseAddress (PPVOID) – output

A pointer to a variable that will receive the base address of the allocated private memory object.

ulObjectSize (ULONG) – input

A value specifying the size (in bytes) of the private memory object to allocate. The size is rounded up to the next page-size boundary.

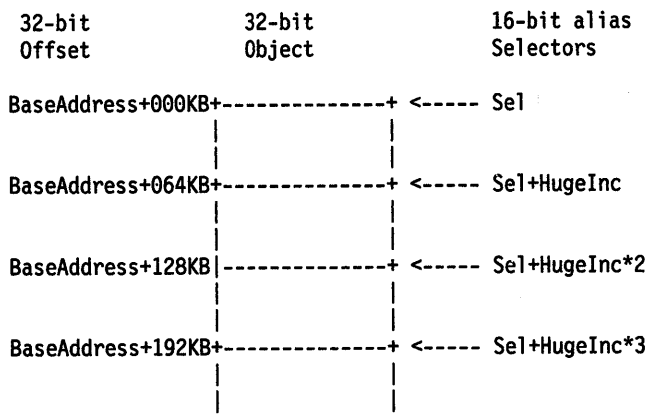
ulAllocationFlags (ULONG) – input

A set of flags describing the allocation attributes and desired access protection for the private memory object.

Allocation Attributes

- If the PAG_COMMIT bit (0x00000010) is set, all pages in the private memory object are initially committed.
- If the OBJ_TILE bit (0x00000040) is set, the private memory object must be allocated in the first 512MB of virtual-address space, with 16-bit selectors mapping the memory object.

The 16-bit selectors are allocated to map the 32-bit object at 64KB boundaries. The figure below shows how the 16-bit alias selectors map the 32-bit object.



HugeInc is the huge increment used for DosAllocHuge.

Desired Access Protection

- If the PAG_EXECUTE bit (0x00000004) is set, execute access to the committed pages in the private memory object is desired.
- If the PAG_READ bit (0x00000001) is set, read access is desired.
- If the PAG_WRITE bit (0x00000002) is set, write access is desired.

- If the PAG_GUARD bit (0x00000008) is set, access to the committed pages in the private memory object causes a "guard page entered" condition to be raised in the subject process.
- At least one of the bits PAG_READ, PAG_WRITE, or PAG_EXECUTE must be set. All other bits must be clear.

Returns

Return Code.

DosAllocMem returns the following values:

0	NO_ERROR
8	ERROR_NOT_ENOUGH_MEMORY
87	ERROR_INVALID_PARAMETER
95	ERROR_INTERRUPT

Remarks

DosAllocMem can be used to reserve, or reserve and commit, linear address space for a private memory object.

The operating system allocates a range of private pages large enough to fulfill the specified allocation request from the private virtual-address space of the subject process. The base address of the object is returned in the BaseAddress parameter.

The allocated memory object is rounded up to a multiple of 4KB in size.

The committed memory allocated by DosAllocMem can be swapped.

Any access protection can be applied to committed pages within a private memory object. Committed pages are initially allocated and backed by demand pages. The first attempt to read or write the page causes a page of zeros to be created.

If a failure occurs during the allocation, no pages are allocated, and an appropriate error code is returned.

With the Intel 80386 processor, execute and read access are equivalent. Also, write access implies both read and execute access.

The guard-page attribute is intended to provide automatic stack-growth and stack-limit checking. An application may also use it in other data structures.

Reserved pages that are not committed are given an access protection of "no access".

Related Functions

- DosAllocSharedMem
- DosFreeMem

DosAllocMem –
Allocate a Private Memory Object Memory

Example Code

This example allocates a private memory object within the virtual address space.

```
#define  INCL_DOSMEMMGR   /* Memory Manager values */
#include <os2.h>
#include <stdio.h>
#include <bsememf.h>   /* Get flags for memory management */

PVOID    BaseAddress;     /* Pointer to the base address of the
                             allocated memory object */
ULONG    Objectsize;      /* Size in bytes of the
                             memory object */
ULONG    AllocationFlags; /* Flags describing characteristics
                             of the memory object */
APIRET   rc;              /* Return code */

   Objectsize = 6000;     /* Ask for a memory object of 6000 */
                          /*   bytes, this size will be      */
                          /*   rounded up to 8KB             */

   AllocationFlags = PAG_WRITE | PAG_READ;
                          /* Permit read and write access to */
                          /*   the memory block, but do not  */
                          /*   have it immediately committed */
                          /*   within memory                 */

   rc = DosAllocMem(&BaseAddress, Objectsize, AllocationFlags);

   if (rc != 0)
     {
       printf("DosAllocMem error: return code = %ld", rc);
       return;
     }
```

DosAllocSharedMem – Allocate a Shared Memory Object

#define INCL_DOSMEMMGR

APIRET DosAllocSharedMem (PPVOID ppBaseAddress, PSZ pszName, ULONG ulObjectSize, ULONG ulFlags)

DosAllocSharedMem allocates a shared memory object within the virtual-address space.

Parameters

ppBaseAddress (PPVOID) – output

A pointer to a variable that will receive the base address of the allocated range of pages. The operating system will determine where to allocate the virtual address for the shared memory object.

pszName (PSZ) – input

An optional address of the name string associated with the shared memory object to be allocated. The name is an ASCIIZ string in the format of an OS/2 file name, and is in the subdirectory, \SHAREMEM\; for example, \SHAREMEM\PUBLIC.DAT.

ulObjectSize (ULONG) – input

A value specifying the size, in bytes, of the shared memory object to allocate. The size is rounded up to the next page-size boundary.

ulFlags (ULONG) – input

A set of flags describing the allocation attributes and desired access protection for the shared memory object.

Allocation Attributes

- If the PAG_COMMIT bit (0x00000010) is set, all pages in the shared memory object are initially committed.
- If the OBJ_GIVEABLE bit (0x00000200) is set, the access to the memory object can be given to another process using the DosGiveSharedMem function.
- If the OBJ_GETTABLE bit (0x00000100) is set, the memory object can be accessed by another process that knows the address of the memory and calls the DosGetSharedMem function.
- If the OBJ_TILE bit (0x00000040) is set, the shared memory object must be allocated in the first 512MB of virtual-address space, with 16-bit selectors mapping the memory object.

  The 16-bit selectors are allocated to map the 32-bit object at 64KB boundaries. The figure in the description of the **Parameters** for DosAllocMem shows how the 16-bit alias selectors map the 32-bit object.

Desired Access Protection

- If the PAG_EXECUTE bit (0x00000004) is set, execute access to the committed pages in the private memory object is desired.
- If the PAG_READ bit (0x00000001) is set, read access is desired.
- If the PAG_WRITE bit (0x00000002) is set, write access is desired.
- If the PAG_GUARD bit (0x00000008) is set, access to the committed pages in the private memory object causes a "guard page entered" condition to be raised in the subject process.
- At least one of the bits of PAG_READ, PAG_WRITE, or PAG_EXECUTE must be set. All other bits must be clear.

DosAllocSharedMem – Allocate a Shared Memory Object

Returns

Return Code.

DosAllocSharedMem returns the following values:

0	NO_ERROR
8	ERROR_NOT_ENOUGH_MEMORY
87	ERROR_INVALID_PARAMETER
95	ERROR_INTERRUPT
123	ERROR_INVALID_NAME
183	ERROR_ALREADY_EXISTS

Remarks

DosAllocSharedMem allocates a shared memory object within the virtual-address space.

Allocating a shared memory object causes the creation of an object that describes a region of memory that can be shared. The virtual-address space in the calling process is allocated and mapped to the shared memory object.

The virtual-address space for a shared memory object is reserved at the same location in the virtual address space of every process. This allows any process to gain access to the shared object at the same virtual address where it was originally allocated.

When the shared memory object is given a name, the shared memory object can be shared by other processes that gain access through the shared memory name (see DosGetNamedSharedMem).

To specify the name for the shared memory object, the name string provided must include the prefix "\SHAREMEM\".

It is an error to request giveable or gettable named shared memory.

If the shared memory object is unnamed, it may be specified as giveable or gettable. Unnamed shared memory may be shared by all processes that get access to the shared memory object (see DosGetSharedMem), or are given access to the shared memory object (see DosGiveSharedMem).

It is an error to request non-giveable or non-gettable unnamed shared memory.

The allocated memory object is rounded up to a multiple of 4KB in size.

The committed memory allocated by DosAllocSharedMem is movable and can be swapped.

With the Intel 80386 processor, execute and read access are equivalent. Also, write access implies both read and execute access.

The tiled allocation attribute is provided for compatibility with the existing 16-bit implementation of the operating system. If the shared memory object is tiled, the virtual address for the shared memory object will be within the first 512MB of the virtual address space, with 16-bit selectors mapping the memory object.

Related Functions

- DosAllocMem
- DosFreeMem
- DosGetNamedSharedMem
- DosGetSharedMem
- DosGiveSharedMem

Example Code

This example allocates a shared named memory object within the virtual address space.

```
#define  INCL_DOSMEMMGR   /* Memory Manager values */
#include <os2.h>
#include <stdio.h>
#include <bsememf.h>   /* Get flags for memory management */

PVOID    BaseAddress;     /* Pointer to the base address of the
                             allocated range of pages */
UCHAR    Name[40];        /* Pointer to the name of the shared
                             memory object to be allocated */
ULONG    ObjectSize;      /* Size in bytes of the
                             the memory object */
ULONG    Flags;           /* Flags describing characteristics
                             of the memory object */
APIRET   rc;              /* Return code */

   strcpy(Name,"\\SHAREMEM\\BLOCK1.DAT");
                          /* Name of shared memory object to be */
                          /*   created                          */

   ObjectSize = 6000;     /* Ask for a memory object of 6000    */
                          /*   bytes. This size will be rounded */
                          /*   to 8KB.                          */

   Flags = PAG_WRITE | PAG_READ;
                          /* Permit read and write access to the */
                          /*   named shared memory block, but do */
                          /*   not have its pages immediately    */
                          /*   committed within virtual memory   */

   rc = DosAllocSharedMem(&BaseAddress, Name,
                          ObjectSize, Flags);

   if (rc != 0)
     {
       printf("DosAllocSharedMem error: return code = %ld", rc);
       return;
     }
```

DosAsyncTimer –
Start an Asynchronous Timer

#define INCL_DOSASYNCTIMER

APIRET DosAsyncTimer (ULONG ulTimeInterval, HSEM hsemSemHandle, PHTIMER ppHandle)

DosAsyncTimer starts an asynchronous, single-interval timer.

Parameters

ulTimeInterval (ULONG) – input

The time, in milliseconds, before the event semaphore specified by *SemHandle* is posted. (The system rounds this value up to the next clock tick.)

hsemSemHandle (HSEM) – input

The handle of an event semaphore that will be posted when the time specified by *TimeInterval* has elapsed. This semaphore must be a shared event semaphore, and should be reset before issuing DosAsyncTimer.

ppHandle (PHTIMER) – output

A pointer to the timer handle. This handle can be passed to DosStopTimer to stop the timer before its time interval expires.

Returns

Return Code.

DosAsyncTimer returns the following values:

0	NO_ERROR
323	ERROR_TS_SEMHANDLE
324	ERROR_TS_NOTIMER

Remarks

DosAsyncTimer starts a single-interval timer. The timer runs asynchronously to the calling thread, and posts an event semaphore when the specified time interval expires.

Time intervals for DosAsyncTimer, DosStartTimer, and DosSleep are specified in milliseconds; however, it is important to recognize that the actual duration of the specified time interval will be affected by two factors:

- First, the system clock keeps track of time in less precise units known as *clock ticks*. The duration of a clock tick depends on the frequency of the system-clock interrupt that is used by your computer. (To determine the duration of the clock tick on your computer, issue DosQuerySysInfo and examine the timer-interval field.)

  Because clock ticks are less precise than millisecond values, any time interval that is specified in milliseconds will be rounded up to the next clock tick.

- Second, because the system is a priority-based, multitasking operating system, there is no guarantee that a thread will resume immediately after the timer interval expires. If a higher-priority process or thread is running, the timed thread blocks. (To minimize the inaccuracy caused by preemptive scheduling, an application can dedicate a thread to managing time-critical tasks, and then raise that thread to a higher priority.)

These factors usually cause the timer interval to be longer than requested; however, it will generally be within a few clock ticks.

Related Functions

- DosGetDateTime
- DosSetDateTime
- DosSleep
- DosStartTimer
- DosStopTimer
- DosCreateEventSem
- DosOpenEventSem
- DosResetEventSem
- DosWaitEventSem

Example Code

The following example shows the use of DosAsyncTimer.

```
#define INCL_BASE
#define OS2_API32
#define INCL_DOSDATETIME   /* Date and time values */
#include <os2.h>
#include <stdio.h>
#include <string.h>

main()
{
    APIRET rc;                      /* Return code */
    ULONG  TimeInterval = 10000L;   /* Time delay */
    UCHAR  szName[20];              /* Event-semaphore name */
    HSEM   SemHandle = 0;           /* Event-semaphore handle repository */
    ULONG  flAttr = 0;              /* Ignored semaphore creation
                                       attributes */
    BOOL32 fState = FALSE;          /* Initial state of semaphore
                                       (FALSE = "reset") */
    HTIMER Handle = 0;              /* Timer handle */
    PHTIMER pHandle = &Handle;       /* Pointer to timer handle */
    ULONG ulTimeout = 20000L;       /* Time-out value */

    /* Create an event semaphore to be posted by the timer. */

    strcpy(szName,"\\SEM32\\TIMASYNC");   /* Set up semaphore name */

    rc = DosCreateEventSem(szName, (PULONG) &SemHandle, flAttr, fState);

    if (rc != 0)                          /* Error received */
      {
        printf("DosCreateEventSem failed -- rc = %ld", rc);
        DosExit(0,1);
      }

    /* Start the timer. */

    rc = DosAsyncTimer(TimeInterval, SemHandle, &Handle);

    if (rc != 0)                          /* Error received */
      {
        printf("DosAsyncTimer failed -- rc = %ld", rc);
        DosExit(0,1);
      }

    /* Wait for the semaphore to be posted.  */

    rc = DosWaitEventSem((ULONG) SemHandle, ulTimeout);

    if (rc != 0)                          /* Error received */
```

```
        {
          printf("DosWaitEventSem failed -- rc = %ld", rc);
          DosExit(0,1);
        }

      /* Indicate that the timer has posted the event semaphore. */

      printf("Timer has posted semaphore.");
      DosExit(0,0);
}
```

DosBeep – Generate Sound from the Speaker

#define INCL_DOSPROCESS

APIRET DosBeep (ULONG ulFrequency, ULONG ulDuration)

DosBeep generates sound from the speaker.

Parameters

ulFrequency (ULONG) – input

Cycles per second (Hertz) in the range of hex 25 to hex 7FFF.

ulDuration (ULONG) – input

The length of the sound in milliseconds.

Returns

Return Code.

DosBeep returns the following values:

0	NO_ERROR
395	ERROR_INVALID_FREQUENCY

Related Functions

- DosDevConfig
- DosDevIOCtl
- DosPhysicalDisk

Example Code

This example generates a beep for 1 second (1000 milliseconds) at a frequency of 1380.

```
#define INCL_DOSPROCESS   /* Process and thread values */
#include <os2.h>
#include <stdio.h>

#define BEEP_FREQUENCY 1380
#define BEEP_DURATION 1000

APIRET  rc;      /* Return code */

   rc = DosBeep(BEEP_FREQUENCY,
                BEEP_DURATION);
```

DosCallNPipe –
Perform Procedure Call Transaction

```
#define INCL_DOSNMPIPES
```

```
APIRET DosCallNPipe (PSZ pszFileName, PVOID pInBuffer, ULONG ulInBufferLen,
                     PVOID pOutBuffer, ULONG ulOutBufferLen, PULONG pBytesOut,
                     ULONG ulTimeOut)
```

DosCallNPipe makes a procedure call to a duplex message pipe.

Parameters

pszFileName (PSZ) – input

The ASCIIZ name of the pipe to be opened. Pipe names must include the prefix \PIPE\ and must conform to file-system naming conventions. When communicating with a remote process, the computer name must also be included, using the format \\ComputerName\PIPE\FileName.

pInBuffer (PVOID) – input

A pointer to the buffer that is to be written to the pipe.

ulInBufferLen (ULONG) – input

The number of bytes to be written.

pOutBuffer (PVOID) – output

A pointer to the buffer for returned data.

ulOutBufferLen (ULONG) – input

The maximum size, in bytes, of returned data.

pBytesOut (PULONG) – output

The address of a doubleword where the system returns the number of bytes actually read.

ulTimeOut (ULONG) – input

The maximum time, in milliseconds, to wait for a pipe instance to become available.

Returns

Return Code.

DosCallNPipe returns the following values:

0	NO_ERROR
2	ERROR_FILE_NOT_FOUND
11	ERROR_BAD_FORMAT
95	ERROR_INTERRUPT
230	ERROR_BAD_PIPE
231	ERROR_PIPE_BUSY
233	ERROR_PIPE_NOT_CONNECTED
234	ERROR_MORE_DATA

Remarks

DosCallNPipe combines the functions of DosOpen, DosTransactNPipe, and DosClose for a duplex message pipe. If no instances of a pipe are available, DosCallNPipe waits for a specified time interval, and returns ERROR_INTERRUPT if the time interval elapses.

If this function is issued for a pipe that is not a duplex message pipe, ERROR_BAD_FORMAT is returned.

If an invalid pipe name is specified, DosCallNPipe returns ERROR_FILE_NOT_FOUND.

If *OutBuffer* is too small to contain the response message, ERROR_MORE_DATA is returned.

If the server process has not issued DosConnectNPipe to put the pipe into a listening state, DosCallNPipe returns ERROR_PIPE_BUSY.

Related Functions

- DosConnectNPipe
- DosCreateNPipe
- DosDisConnectNPipe
- DosPeekNPipe
- DosQueryNPHState
- DosQueryNPipeInfo
- DosQueryNPipeSemState
- DosSetNPHState
- DosSetNPipeSem
- DosTransactNPipe
- DosWaitNPipe
- DosClose
- DosDupHandle
- DosOpen
- DosRead
- DosResetBuffer
- DosWrite

DosCallNPipe –
Perform Procedure Call Transaction

Example Code

This example is a "procedure call" through a named pipe. It returns with the results of the procedure call, assuming that the call does not time out past the user-supplied time-out period.

```
#define INCL_DOSNMPIPES   /* Named-pipe values */
#include <os2.h>
#include <stdio.h>

UCHAR    FileName[40];   /* Pipe name */
UCHAR    InBuffer[800];  /* Write-buffer address  */
ULONG    InBufferLen;    /* Write-buffer length   */
UCHAR    OutBuffer[800]; /* Read-buffer address   */
ULONG    OutBufferLen;   /* Read-buffer length    */
ULONG    BytesOut;       /* Bytes read (returned) */
ULONG    TimeOut;        /* Maximum wait time     */
APIRET   rc;             /* Return code */

   strcpy(FileName,"\PIPE\PIPE1");

   strcpy(InBuffer,"Command 1: Start Proc 1");
                        /* Set input buffer to contain the */
                        /*   desired procedure call data   */

   InBufferLen = strlen(InBuffer);
                        /* Set length indicator for input  */
                        /*   buffer                        */

   OutBufferLen = 800;  /* Max data length for output */
                        /*   (return) buffer          */

   TimeOut = 30000;     /* Timeout of 30 seconds (units */
                        /*   are milliseconds)          */

   rc = DosCallNPipe(FileName, InBuffer, InBufferLen, OutBuffer,
                           OutBufferLen, &BytesOut, TimeOut);
                        /* On successful return, variable  */
                        /*   BytesOut will contain the     */
                        /*   number of bytes written to    */
                        /*   the output buffer (OutBuffer) */

   if (rc != 0)
     {
       printf("DosCallNPipe error: return code = %ld", rc);
       return;
     }
```

DosCancelLockRequest — Cancel an Outstanding DosSetFileLocks Request

#define INCL_DOSFILEMGR

APIRET DosCancelLockRequest (HFILE FileHandle, PFILELOCK ppLockRange)

DosCancelLockRequest cancels an outstanding DosSetFileLocks request.

Parameters

FileHandle (HFILE) — input

File handle used in the DosSetFileLocks function that is to be cancelled.

ppLockRange (PFILELOCK) — input

Address of the structure describing the region to be locked by DosSetFileLocks. The structure is as follows:

FileOffset (LONG) — **Input**
The offset to the beginning of the range to be locked.

RangeLength (LONG) — **Input**
The length, in bytes, of the range to be locked.

Returns

Return Code.

DosCancelLockRequest returns the following values:

0	NO_ERROR
6	ERROR_INVALID_HANDLE
87	ERROR_INVALID_PARAMETER
173	ERROR_CANCEL_VIOLATION

Remarks

DosCancelLockRequest allows a process to cancel the lock range request of an outstanding DosSetFileLocks function.

If two threads in a process are waiting on a lock file range, and another thread issues DosCancelLockRequest for that lock file range, then both waiting threads are released.

Not all file-system drivers (FSDs) can cancel an outstanding lock request.

Local Area Network (LAN) servers cannot cancel an outstanding lock request if they use a version of the operating system prior to OS/2 Version 2.00.

Related Functions

- DosSetFileLocks

Example Code

This example opens a file, writes some data to it, locks a block of the data, and then cancels the lock request.

```
#define INCL_DOSFILEMGR   /* File Manager values */
#include <os2.h>
#include <stdio.h>
```

DosCancelLockRequest – Cancel an Outstanding DosSetFileLocks Request

```
#define OPEN_FILE 0x01
#define CREATE_FILE 0x10
#define FILE_ARCHIVE 0x20
#define FILE_EXISTS OPEN_FILE
#define FILE_NOEXISTS CREATE_FILE
#define DASD_FLAG 0
#define INHERIT 0x80
#define WRITE_THRU 0
#define FAIL_FLAG 0
#define SHARE_FLAG 0x10
#define ACCESS_FLAG 0x02

#define FILE_NAME "test.dat"
#define FILE_SIZE 800L
#define FILE_ATTRIBUTE FILE_ARCHIVE
#define EABUF 0L
#define NULL_RANGE 0L

#define LOCK_FLAGS 0

HFILE   FileHandle;
ULONG   Wrote;
ULONG   Action;
PSZ     FileData[100];
ULONG   LockTimeout = 2000;
APIRET  rc;                 /* Return code */

struct LockStrc
   {
   long Offset;
   long Range;
   } Area;

int i;

   Action = 2;
   strcpy(FileData, "Data...");
   Area.Offset = 4;
   Area.Range = 100;

  rc = DosOpen(FILE_NAME,               /* File path name */
               &FileHandle,             /* File handle */
               &Action,                 /* Action taken */
               FILE_SIZE,               /* File primary allocation */
               FILE_ATTRIBUTE,          /* File attribute */
               FILE_EXISTS | FILE_NOEXISTS,   /* Open function type */
               DASD_FLAG | INHERIT |        /* Open mode of file */
               WRITE_THRU | FAIL_FLAG |
               SHARE_FLAG | ACCESS_FLAG,
               EABUF);              /* No extended attributes */

   if (rc != 0)
     {
       printf("DosOpen error: return code = %ld",rc);
       return;
     }

      {
      for(i=0; i<200; ++i)
         DosWrite(FileHandle,           /* File handle */
                  FileData,             /* User buffer */
                  sizeof(FileData),     /* Buffer length */
                  &Wrote);              /* Bytes written */
```

```
rc = DosSetFileLocks(FileHandle,          /* File handle */
                     NULL_RANGE,          /* Unlock range */
                     (PFILELOCK) &Area,   /* Lock range */
                     &LockTimeout,        /* Lock time-out */
                     LOCK_FLAGS);         /* Request flags */

if (rc != 0)
  {
    printf("DosSetFileLocks error: return code = %ld",rc);
    return;
  }

rc = DosCancelLockRequest(FileHandle,       /* File handle */
                     (PFILELOCK) &Area);    /* Lock range */

if (rc != 0)
  {
    printf("DosCancelLockRequest error: return code = %ld",rc);
    return;
  }

}
```

DosClose –
Close a Handle to a File, Pipe, or Device

#define INCL_DOSFILEMGR

APIRET DosClose (HFILE FileHandle)

DosClose closes a handle to a file, pipe, or device.

Parameters

FileHandle (HFILE) – input

The handle returned by a previous call to DosCreateNPipe, DosCreatePipe, DosDupHandle, or DosOpen.

Returns

Return Code.

DosClose returns the following values:

0	NO_ERROR
2	ERROR_FILE_NOT_FOUND
5	ERROR_ACCESS_DENIED
6	ERROR_INVALID_HANDLE

Remarks

Issuing DosClose with the handle to a file closes a handle to a file, pipe, or device.

If additional handles to a file were created with DosDupHandle, DosClose must be issued for the duplicate handles before the directory is updated, and information in internal buffers is written to the medium.

Closing a device handle causes the device to be notified of the close, if appropriate.

Named-Pipe Considerations

DosClose closes a named pipe by handle. When all handles that refer to one end of a pipe are closed, the pipe is considered broken.

If the client end closes, no other process can reopen the pipe until the serving end issues DosDisConnectNPipe, followed by DosConnectNPipe.

If the server end closes when the pipe is already broken, the pipe is immediately deallocated; otherwise, it is not deallocated until the last client handle is closed.

Related Functions

- DosConnectNPipe
- DosCreateNPipe
- DosDisConnectNPipe
- DosDupHandle
- DosOpen
- DosResetBuffer

Example Code

This example opens a file, then closes it.

```
#define INCL_DOSFILEMGR   /* File Manager values */
#include <os2.h>
#include <stdio.h>

#define OPEN_FILE 0x01
#define CREATE_FILE 0x10
#define FILE_ARCHIVE 0x20
#define FILE_EXISTS OPEN_FILE
#define FILE_NOEXISTS CREATE_FILE
#define DASD_FLAG 0
#define INHERIT 0x80
#define WRITE_THRU 0
#define FAIL_FLAG 0
#define SHARE_FLAG 0x10
#define ACCESS_FLAG 0x02

#define FILE_NAME "test.dat"
#define FILE_SIZE 800L
#define FILE_ATTRIBUTE FILE_ARCHIVE
#define EABUF 0L

HFILE   FileHandle;
ULONG   Wrote;
ULONG   Action;
PSZ     FileData[100];
APIRET  rc;             /* Return code */

   Action = 2;
   strcpy(FileData, "Data...");
   rc = DosOpen(FILE_NAME,              /* File path name */
                &FileHandle,            /* File handle */
                &Action,                /* Action taken */
                FILE_SIZE,              /* File primary allocation */
                FILE_ATTRIBUTE,         /* File attribute */
                FILE_EXISTS | FILE_NOEXISTS,   /* Open function type */
                DASD_FLAG | INHERIT |          /* Open mode of the file */
                WRITE_THRU | FAIL_FLAG |
                SHARE_FLAG | ACCESS_FLAG,
                EABUF);                 /* No extended attributes */

   if (rc != 0)
     {
       printf("DosOpen error: return code = %ld",rc);
       return;
     }

   rc = DosClose(FileHandle);         /* File Handle */

   if (rc != 0)
     {
       printf("DosClose error: return code = %ld",rc);
       return;
     }
```

DosCloseEventSem – Close Event Semaphore

#define INCL_DOSSEMAPHORES

APIRET DosCloseEventSem (HEV hev)

DosCloseEventSem closes an event semaphore.

Parameters

hev (HEV) – input

The handle of the event semaphore to close.

Returns

Return Code.

DosCloseEventSem returns the following values:

0	NO_ERROR
6	ERROR_INVALID_HANDLE
301	ERROR_SEM_BUSY

Related Functions

- DosCreateEventSem
- DosOpenEventSem
- DosPostEventSem
- DosQueryEventSem
- DosResetEventSem
- DosWaitEventSem

Example Code

This example closes an event semaphore. Assume that the handle of the semaphore has been placed into *hev* already.

```
#define INCL_DOSSEMAPHORES   /* Semaphore values */
#include <os2.h>
#include <stdio.h>

HEV     hev;     /* Event semaphore handle */
APIRET  rc;      /* Return code */

   rc = DosCloseEventSem(hev);

   if (rc != 0)
     {
       printf("DosCloseEventSem error: return code = %ld", rc);
       return;
     }
```

DosCloseMutexSem — Close Mutex Semaphore

#define INCL_DOSSEMAPHORES

APIRET DosCloseMutexSem (HMTX hmtx)

DosCloseMutexSem closes a mutex semaphore.

Parameters

hmtx (HMTX) — input

The handle of the mutex semaphore to close.

Returns

Return Code.

DosCloseMutexSem returns the following values:

0	NO_ERROR
6	ERROR_INVALID_HANDLE
301	ERROR_SEM_BUSY

Remarks

DosCloseMutexSem closes (ends access to) a mutex semaphore for all of the threads in the calling process.

When all of the processes that had opened the semaphore have either closed the semaphore or have ended, the semaphore is deleted by the system.

Related Functions

- DosCreateMutexSem
- DosOpenMutexSem
- DosQueryMutexSem
- DosReleaseMutexSem
- DosRequestMutexSem

Example Code

This example closes a mutex semaphore. Assume that the handle of the semaphore has been placed into *hmtx* already.

```
#define INCL_DOSSEMAPHORES   /* Semaphore values */
#include <os2.h>
#include <stdio.h>

HMTX    hmtx;    /* Mutex semaphore handle */
APIRET  rc;      /* Return code */

   rc = DosCloseMutexSem(hmtx);

   if (rc != 0)
     {
       printf("DosCloseMutexSem error: return code = %ld", rc);
       return;
     }
```

DosCloseMuxWaitSem – Close MuxWait Semaphore

#define INCL_DOSSEMAPHORES

```
APIRET DosCloseMuxWaitSem (HMUX hmux)
```

DosCloseMuxWaitSem closes a muxwait semaphore.

Parameters

hmux (HMUX) – input

The handle of the muxwait semaphore to close.

Returns

Return Code.

DosCloseMuxWaitSem returns the following values:

0	NO_ERROR
6	ERROR_INVALID_HANDLE
301	ERROR_SEM_BUSY

Remarks

DosCloseMuxWaitSem closes (ends access to) a muxwait semaphore for all of the threads in the calling process.

When all of the processes that opened the semaphore have either closed the semaphore or have ended, the system deletes the semaphore.

Related Functions

- DosAddMuxWaitSem
- DosCreateMuxWaitSem
- DosDeleteMuxWaitSem
- DosOpenMuxWaitSem
- DosQueryMuxWaitSem
- DosWaitMuxWaitSem

Example Code

This example closes a muxwait semaphore. Assume that the handle of the semaphore has been placed into *hmux* already.

```
#define INCL_DOSSEMAPHORES   /* Semaphore values */
#include <os2.h>
#include <stdio.h>

HMUX    hmux;    /* Muxwait semaphore handle */
APIRET  rc;      /* Return code */

   rc = DosCloseMuxWaitSem(hmux);

   if (rc != 0)
     {
       printf("DosCloseMuxWaitSem error: return code = %ld", rc);
       return;
     }
```

DosCloseQueue – Close Queue

#define INCL_DOSQUEUES

APIRET DosCloseQueue (HQUEUE QueueHandle)

DosCloseQueue ends access to a queue, or deletes a queue.

Parameters

QueueHandle (HQUEUE) – input

The handle of the queue to be closed. This is the handle received from a previous call to DosCreateQueue or DosOpenQueue.

Returns

Return Code.

DosCloseQueue returns the following values:

0 NO_ERROR
337 ERROR_QUE_INVALID_HANDLE

Remarks

DosCloseQueue ends further processing of a queue by the requesting process. The action taken depends on whether the requester is the owner of the queue or a writer of the queue.

For the owner, any outstanding elements are deleted. Other processes that have the queue open will receive the ERROR_QUE_INVALID_HANDLE return code on their next request.

For a writer of the queue, access to the queue is ended, but the queue elements are not deleted.

Related Functions

- DosCreateQueue
- DosOpenQueue
- DosPeekQueue
- DosPurgeQueue
- DosQueryQueue
- DosReadQueue
- DosWriteQueue

Example Code

The following code fragment creates and opens a queue named SPECIAL.QUE for a server process, then closes the queue.

```
#define INCL_DOSQUEUES   /* Queue values */
#include <os2.h>
#include <stdio.h>

#define QUE_NAME "\\QUEUES\\SPECIAL.QUE"

HQUEUE QueueHandle;     /* Queue handle */
APIRET rc;              /* Return code  */

  rc = DosCreateQueue(&QueueHandle,       /* Queue handle */
       QUE_FIFO |                         /* Ordering of elements */
       QUE_CONVERT_ADDRESS,       /* 16-bit address conversion */
       QUE_NAME);                        /* Queue name string */

     rc = DosCloseQueue(QueueHandle);      /* Queue handle */
```

DosCloseVDD –
Close a Virtual Device Driver Handle

#define INCL_DOSMVDM

APIRET DosCloseVDD (HVDD VDDHandle)

DosCloseVDD closes the specified virtual device driver (VDD) handle.

Parameters

VDDHandle (HVDD) – input

The handle of the virtual device driver to be closed. Specify the handle that was returned by a previous call to DosOpenVDD.

Returns

Return Code.

DosCloseVDD returns the following values:

0	NO_ERROR
6	ERROR_INVALID_HANDLE
644	ERROR_INVALID_CALLER

Remarks

DosCloseVDD closes the specified virtual device driver (VDD) handle.

Related Functions

- DosOpenVDD
- DosRequestVDD

Example Code

This example closes a virtual device driver (VDD). Assume that the handle of the VDD has been placed into *VDDHandle* already.

```
#define INCL_DOSMVDM   /* Multiple DOS sessions values */
#include <os2.h>
#include <stdio.h>
#include <vdmm.h>

HVDD    VDDHandle;   /* Handle of VDD */
APIRET  rc;          /* Return code */

   rc = DosCloseVDD(VDDHandle);

   if (rc != 0)
     {
       printf("DosCloseVDD error: return code = %ld", rc);
       return;
     }
```

DosConnectNPipe – Connect Named Pipe

#define INCL_DOSNMPIPES

APIRET DosConnectNPipe (HPIPE hpipeHandle)

DosConnectNPipe prepares a named pipe for a client process.

Parameters

hpipeHandle (HPIPE) – input

The named-pipe handle to connect (returned to the server process by DosCreateNPipe).

Returns

Return Code.

DosConnectNPipe returns the following values:

0	NO_ERROR
95	ERROR_INTERRUPT
109	ERROR_BROKEN_PIPE
230	ERROR_BAD_PIPE
233	ERROR_PIPE_NOT_CONNECTED

Remarks

DosConnectNPipe is issued by a server process to put a named pipe into the listening state. This enables a client process to gain access to the pipe by calling DosOpen.

If the client end of the pipe is already open when DosConnectNPipe is issued, DosConnectNPipe returns immediately and has no effect. If the client end is closed, the result depends on whether the pipe is in blocking mode or nonblocking mode. (Blocking/nonblocking mode is specified when the pipe is created; it can also be changed by DosSetNPHState).

- If the pipe is in blocking mode, DosConnectNPipe waits for a client to open the pipe before returning.
- If the pipe is in nonblocking mode, DosConnectNPipe returns immediately with ERROR_PIPE_NOT_CONNECTED. Nevertheless, the pipe is placed into the listening state, permitting a client to subsequently issue DosOpen successfully.

Multiple DosConnectNPipe calls can be issued for a pipe that is in nonblocking mode. If the pipe is not already either open or closing, the first call to DosConnectNPipe puts the pipe into the listening state; subsequent calls merely test the pipe state.

If the pipe was previously opened and then closed by a client, but has not yet been disconnected by the server, DosConnectNPipe returns ERROR_BROKEN_PIPE.

If the function is interrupted while it is waiting for a client to open the pipe, ERROR_INTERRUPT is returned.

If DosConnectNPipe is called by a client process, ERROR_BAD_PIPE is returned.

DosConnectNPipe – Connect Named Pipe

Related Functions

- DosCallNPipe
- DosCreateNPipe
- DosDisConnectNPipe
- DosPeekNPipe
- DosQueryNPHState
- DosQueryNPipeInfo
- DosQueryNPipeSemState
- DosSetNPHState
- DosSetNPipeSem
- DosTransactNPipe
- DosWaitNPipe
- DosClose
- DosDupHandle
- DosOpen
- DosRead
- DosResetBuffer
- DosWrite

Example Code

This example shows how to wait for a new client to open a named pipe (through a DosOpen call). Assume that a previous call to DosCreateNPipe provided the named pipe handle that is contained in *Handle*.

```
#define INCL_DOSNMPIPES   /* Named-pipe values */
#include <os2.h>
#include <stdio.h>

HPIPE   Handle;     /* Pipe handle */
APIRET  rc;         /* Return code */

   rc = DosConnectNPipe(Handle);

   if (rc != 0)
     {
       printf("DosConnectNPipe error: return code = %ld", rc);
       return;
     }
```

DosCopy – Copy a File or Subdirectory

#define INCL_DOSFILEMGR

APIRET DosCopy (PSZ pszSourceName, PSZ pszTargetName, ULONG ulOpMode)

DosCopy copies the source file or subdirectory to the destination file or subdirectory.

Parameters

pszSourceName (PSZ) – input

Address of the ASCIIZ path name of the source file, subdirectory, or character device. Global file-name characters are not allowed.

pszTargetName (PSZ) – input

Address of the ASCIIZ path name of the target file, subdirectory, or character device. Global file-name characters are not allowed.

ulOpMode (ULONG) – input

Doubleword bit map that defines how the DosCopy function is done.

Bit	Description
31 – 3	Reserved; must be set to zero.
2	(DCPY_FAILEAS 0x00000004) Discard the EAs if the source file contains EAs and the destination file system does not support EAs. 0: Discard the EAs (extended attributes) if the destination file system does not support EAs. 1: Fail the copy if the destination file system does not support EAs.
1	(DCPY_APPEND 0x00000002) Append the source file to the target file's end of data. 0: Replace the target file with the source file. 1: Append the source file to the target file's end of data. This is ignored when copying a directory, or if the target file does not exist.
0	(DCPY_EXISTING 0x00000001) Existing Target File Disposition. 0: Do not copy the source file to the target if the file name already exists within the target directory. If a single file is being copied and the target already exists, an error is returned. 1: Copy the source file to the target even if the file name already exists within the target directory.

Returns

Return Code.

DosCopy returns the following values:

0	NO_ERROR
2	ERROR_FILE_NOT_FOUND
3	ERROR_PATH_NOT_FOUND
5	ERROR_ACCESS_DENIED
26	ERROR_NOT_DOS_DISK
32	ERROR_SHARING_VIOLATION
36	ERROR_SHARING_BUFFER_EXCEEDED
87	ERROR_INVALID_PARAMETER
108	ERROR_DRIVE_LOCKED

DosCopy –
Copy a File or Subdirectory

112	ERROR_DISK_FULL
206	ERROR_FILENAME_EXCED_RANGE
267	ERROR_DIRECTORY
282	ERROR_EAS_NOT_SUPPORTED
283	ERROR_NEED_EAS_FOUND

Remarks

DosCopy copies all files and subdirectories in the source path to the target path. Global file-name characters are not allowed in source or target names. The source and the target can be on different drives.

If an I/O error occurs, DosCopy takes the following actions:

- If the source name is that of a subdirectory, deletes the file being copied from the target path.
- If the source name is that of a file to be replaced, deletes the file from the target path.
- If the source name is that of a file to be appended, resizes the target file to its original size.

Read-only files in the target path cannot be replaced by a DosCopy request. If such files exist in the target, and *OpMode* bit flag0 is set to 1, any attempt to replace these files with files from the source will result in an error.

When copying is specified for a single file that has *OpMode* bit 1 set to 1, the operation proceeds even if the file already exists and its *OpMode* bit 0 is set to 0. That is, *OpMode* bit 0 is significant only when replacing a file, not when appending a file.

If a device name is specified as the target, the source name must be a file, not a directory. When the request is issued, *OpMode* bit flags 0 and 1 are ignored.

File-object attributes, such as date of creation, and time of creation, are always copied from the source to the target; however, extended attributes (EAs) are not copied in every case. DosCopy copies EAs from the source to the target when creating a file or directory, or when replacing an existing file on the target. However, it does not copy them when appending an existing file or when copying files to an existing directory on the target. If the file system of the target does not support EAs, DosCopy ends and returns an error.

If the source file object contains a need EA, and the destination file system does not support EAs, DosCopy fails regardless of the value of *OpMode* bit 2.

DosQuerySysInfo is called by an application during initialization to determine the maximum path length allowed by the operating system.

Related Functions

- DosMove
- DosQueryCurrentDisk
- DosQuerySysInfo
- DosSetDefaultDisk

Example Code

This example copies a source file into a different directory.

```
#define INCL_DOSFILEMGR   /* File Manager values */
#include <os2.h>
#include <stdio.h>

UCHAR   SourceName[40];  /* Source path name */
UCHAR   TargetName[40];  /* Target path name */
ULONG   OpMode;          /* Operation mode   */
APIRET  rc;              /* Return code      */

   strcpy(SourceName,"C:\\DIR1\\MYFILE1");
                         /* Source file path name */

   strcpy(TargetName,"\\DIR7\\SUBDIR\\MYFILE3");
                         /* Target file path name relative to */
                         /*   current drive letter (since the */
                         /*   path name contains no drive     */
                         /*   letter)                         */

   OpMode = DCPY_EXISTING | DCPY_FAILEAS;
                         /* Control bits: 1) Copy the source    */
                         /*   file to the target directory even */
                         /*   if a file with that name already  */
                         /*   exists in the directory,          */
                         /*   2) Replace rather than append the */
                         /*   target file, 3) make the copy fail*/
                         /*   if target file system does not    */
                         /*   support the copy of Extended      */
                         /*   Attributes (if any) in the source */
                         /*   file                              */

   rc = DosCopy(SourceName, TargetName, OpMode);

   if (rc != 0)
     {
       printf("DosCopy error: return code = %ld", rc);
       return;
     }
```

DosCreateDir – Create a Directory

#define INCL_DOSFILEMGR

APIRET DosCreateDir (PSZ pszDirName, PEAOP2 ppEABuf)

DosCreateDir creates a new directory.

Parameters

pszDirName (PSZ) – input

Address of the ASCIIZ directory path name, which may contain a drive specification. If no drive is specified, the current drive is assumed.

DosQuerySysInfo is called by an application during initialization to determine the maximum path length allowed by the operating system.

ppEABuf (PEAOP2) – input/output

Address of the extended attribute buffer, which contains an EAOP2 data structure.

On input, the fpGEA2List field and oError fields are ignored. The EA setting operation is performed on the information contained in FEA2List. If extended attributes are not to be defined or modified, then EABuf must be set to null.

On output, fpGEA2List and fpFEA2List are unchanged. The area that fpFEA2List points to is unchanged. If an error occurred during the set, oError is the offset of the FEA2 where the error occurred. The return code is the error code corresponding to the condition generating the error. If no error occurred, oError is undefined.

If *EABuf* is zero, then no extended attributes are defined for the directory.

Returns

Return Code.

DosCreateDir returns the following values:

0	NO_ERROR
3	ERROR_PATH_NOT_FOUND
5	ERROR_ACCESS_DENIED
26	ERROR_NOT_DOS_DISK
87	ERROR_INVALID_PARAMETER
108	ERROR_DRIVE_LOCKED
206	ERROR_FILENAME_EXCED_RANGE
254	ERROR_INVALID_EA_NAME
255	ERROR_EA_LIST_INCONSISTENT
	ERROR_EA_VALUE_UNSUPPORTABLE

Remarks

DosCreateDir enables an application to define extended attributes for a subdirectory at the time of its creation.

If any subdirectory names specified in the path name do not exist, the subdirectory is not created. Upon successful return, a subdirectory is created at the end of the specified path.

An application must issue DosQuerySysInfo to determine the maximum path length that the operating system supports. The returned value should be used to dynamically allocate buffers that are to be used to store paths.

If a program with its NEWFILES bit set tries to create a directory on an FAT file system drive and specifies blanks immediately preceding the dot in a file name, the system rejects the name. For example, if c: is an FAT file system drive, the name "file .txt" is rejected, but "file.txt" is accepted.

Related Functions

- DosQuerySysInfo

Example Code

This example creates a new directory. Assume that no extended attributes need to be defined for the new directory.

```
#define INCL_DOSFILEMGR   /* File Manager values */
#include <os2.h>
#include <stdio.h>

UCHAR   DirName[40]; /* New directory name string */
PEAOP2  EABuf;       /* Extended attribute buffer */
APIRET  rc;          /* Return code */

   strcpy(DirName,"\\PROG\\SRC\\UTIL");
                     /* New directory name */

   EABuf = 0;        /* Indicate that no EAs are defined */

   rc = DosCreateDir(DirName, EABuf);

   if (rc != 0)
     {
       printf("DosCreateDir error: return code = %ld", rc);
       return;
     }
```

DosCreateEventSem – Create Event Semaphore

#define INCL_DOSSEMAPHORES

APIRET DosCreateEventSem (PSZ pszName, PHEV ppphev, ULONG ulflattr, BOOL32 f32fState)

DosCreateEventSem creates an event semaphore.

Parameters

pszName (PSZ) – input

A pointer to the ASCIIZ name of the semaphore.

Semaphore names are validated by the file system, and must include the prefix \SEM32\. A maximum of 255 characters is allowed. If these requirements are not met, ERROR_INVALID_NAME is returned. If the semaphore already exists, ERROR_DUPLICATE_NAME is returned.

If this field is null, the semaphore is unnamed. Unnamed event semaphores can be either private or shared, depending on *flAttr*. They are identified by the semaphore handle that *phev* points to.

By default, named semaphores are shared.

ppphev (PHEV) – output

A pointer to the handle of the event semaphore.

ulflattr (ULONG) – input

A set of flags that specify the attributes of the event semaphore. If the DC_SEM_SHARED bit is set, the semaphore is shared. This bit is checked only if the semaphore is unnamed (that is, if *Name* is null), because all named semaphores are shared.

f32fState (BOOL32) – input

Describes the initial state of the semaphore:

Value	Definition
0	(FALSE) The initial state of the semaphore is "set."
1	(TRUE) The initial state of the semaphore is "posted."

Returns

Return Code.

DosCreateEventSem returns the following values:

0	NO_ERROR
8	ERROR_NOT_ENOUGH_MEMORY
87	ERROR_INVALID_PARAMETER
123	ERROR_INVALID_NAME
285	ERROR_DUPLICATE_NAME
290	ERROR_TOO_MANY_HANDLES

Related Functions

- DosCloseEventSem
- DosOpenEventSem
- DosPostEventSem
- DosQueryEventSem
- DosResetEventSem
- DosWaitEventSem

Example Code

This example creates a system event semaphore.

```
#define INCL_DOSSEMAPHORES   /* Semaphore values */
#include <os2.h>
#include <stdio.h>

UCHAR   Name[40]; /* Semaphore name */
HEV     hev;      /* Event semaphore handle */
ULONG   flAttr;   /* Creation attributes */
BOOL32  fState;   /* Initial state of semaphore */
APIRET  rc;       /* Return code  */

   strcpy(Name,"\\SEM32\\EVENT1");
                  /* Name of the new system event semaphore */

   flAttr = 0;    /* Unused, since this is a named semaphore */

   fState = 0;    /* The initial state of the semaphore is */
                  /*   "set"                               */

   rc = DosCreateEventSem(Name, &hev, flAttr, fState);
                  /* On successful return, the hev variable */
                  /*   contains the handle of the new       */
                  /*   system event semaphore               */

   if (rc != 0)
     {
       printf("DosCreateEventSem error: return code = %ld", rc);
       return;
     }
```

DosCreateMutexSem – Create Mutex Semaphore

#define INCL_DOSSEMAPHORES

```
APIRET DosCreateMutexSem (PSZ pszName, PHMTX ppphmtx, ULONG ulflAttr,
                          BOOL32 f32fState)
```

DosCreateMutexSem creates a mutex semaphore.

Parameters

pszName (PSZ) – input

A pointer to the ASCIIZ name of the semaphore.

Semaphore names are validated by the file system, and must include the prefix \SEM32\. A maximum of 255 characters is allowed. If these requirements are not met, ERROR_INVALID_NAME is returned. If the semaphore already exists, ERROR_DUPLICATE_NAME is returned.

If this field is null, the semaphore is unnamed. Unnamed mutex semaphores can be either private or shared, depending on *flAttr*. They are identified by the semaphore handle that *phmtx* points to.

By default, named semaphores are shared.

ppphmtx (PHMTX) – output

A pointer to the handle of the mutex semaphore.

ulflAttr (ULONG) – input

A set of flags that specify the attributes of the semaphore. If the DC_SEM_SHARED bit is set, the semaphore is shared. This bit is checked only if the semaphore is unnamed (that is, if *Name* is null), because all named semaphores are shared.

f32fState (BOOL32) – input

Describes the initial state of the semaphore:

Value	Definition
0	(FALSE) The initial state of the semaphore is "unowned."
1	(TRUE) The initial state of the semaphore is "owned."

Returns

Return Code.

DosCreateMutexSem returns the following values:

0	NO_ERROR
8	ERROR_NOT_ENOUGH_MEMORY
87	ERROR_INVALID_PARAMETER
123	ERROR_INVALID_NAME
285	ERROR_DUPLICATE_NAME
290	ERROR_TOO_MANY_HANDLES

Remarks

DosCreateMutexSem creates a mutual exclusion (mutex) semaphore, and opens it for all of the threads in the current process.

Related Functions

- DosCloseMutexSem
- DosOpenMutexSem
- DosQueryMutexSem
- DosReleaseMutexSem
- DosRequestMutexSem

Example Code

This example creates a system mutex semaphore.

```
#define INCL_DOSSEMAPHORES   /* Semaphore values */
#include <os2.h>
#include <stdio.h>

UCHAR   Name[40]; /* Semaphore name */
HMTX    hmtx;      /* Mutex semaphore handle */
ULONG   flAttr;    /* Creation attributes */
BOOL32  fState;    /* Initial state of the
                      mutex semaphore */
APIRET  rc;        /* Return code  */

   strcpy(Name,"\\SEM32\\MUTEX1");
                /* Name of the new system mutex semaphore */

   flAttr = 0;    /* Unused, since this is a named semaphore */

   fState = 0;    /* The initial state of the semaphore is */
                  /*   "unowned"                           */

   rc = DosCreateMutexSem(Name, &hmtx, flAttr, fState);
                /* On successful return, the hmtx variable */
                /*   contains the handle of the new        */
                /*   system mutex semaphore                */

   if (rc != 0)
     {
       printf("DosCreateMutexSem error: return code = %ld", rc);
       return;
     }
```

DosCreateMuxWaitSem – Create MuxWait Semaphore

```
#define INCL_DOSSEMAPHORES
```

```
APIRET DosCreateMuxWaitSem (PSZ pszName, PHMUX ppphmux, ULONG ulcSemRec,
                            PSEMRECORD pppSemRec, ULONG ulflAttr)
```

DosCreateMuxWaitSem creates a muxwait semaphore.

Parameters

pszName (PSZ) – input

A pointer to the ASCIIZ name of the semaphore.

Semaphore names are validated by the file system, and must include the prefix \SEM32\. A maximum of 255 characters is allowed. If these requirements are not met, ERROR_INVALID_NAME is returned. If the semaphore already exists, ERROR_DUPLICATE_NAME is returned.

If this field is null, the muxwait semaphore is unnamed. Unnamed semaphores can be either private or shared, depending on *flAttr*. They are identified by the semaphore handle that *phmux* points to.

By default, named semaphores are shared.

ppphmux (PHMUX) – output

A pointer to the handle of the muxwait semaphore.

ulcSemRec (ULONG) – input

The count of semaphore record entries in *pSemRec*.

pppSemRec (PSEMRECORD) – input

A pointer to the array of semaphore record entries to put into the muxwait semaphore. This is the list of event or mutex semaphores that must be posted or released in order for the muxwait semaphore to clear.

ulflAttr (ULONG) – input

A set of flags that specify the attributes of the semaphore:

- If the DC_SEM_SHARED bit is set, the semaphore is shared; otherwise, it is a private semaphore. This bit is checked only if the semaphore is unnamed (that is, if *Name* is null), because all named semaphores are shared.
- If the DCMW_WAIT_ANY bit is set, the semaphore clears when any event semaphore in its *pSemRec* list is posted, or when any mutex semaphore in its *pSemRec* list is released. When any one of the semaphores is cleared, the thread waiting for the muxwait semaphore can continue execution.
- If the DCMW_WAIT_ALL bit is set, the semaphore clears when all of the event semaphores in its *pSemRec* list have been posted, or when all of the mutex semaphores in its *pSemRec* list have been released. When all of the semaphores are cleared, the thread waiting for the muxwait semaphore can continue execution.

Returns

Return Code.

DosCreateMuxWaitSem returns the following values:

0	NO_ERROR
6	ERROR_INVALID_HANDLE
8	ERROR_NOT_ENOUGH_MEMORY
87	ERROR_INVALID_PARAMETER
100	ERROR_TOO_MANY_SEMAPHORES
105	ERROR_SEM_OWNER_DIED
123	ERROR_INVALID_NAME
284	ERROR_DUPLICATE_HANDLE
285	ERROR_DUPLICATE_NAME
290	ERROR_TOO_MANY_HANDLES
292	ERROR_WRONG_TYPE

Remarks

DosCreateMuxWaitSem creates a multiple wait (muxwait) semaphore, and opens it for all of the threads in the current process.

Related Functions

- DosAddMuxWaitSem
- DosCloseMuxWaitSem
- DosDeleteMuxWaitSem
- DosOpenMuxWaitSem
- DosQueryMuxWaitSem
- DosWaitMuxWaitSem

Example Code

This example creates a system muxwait semaphore with two initial event semaphore components. Assume that the two event semaphore handles have been placed into *hev0* and *hev1* already. Assume that the two corresponding user semaphore identifier values have been placed into *ulUser0* and *ulUser1* already.

DosCreateMuxWaitSem – Create MuxWait Semaphore

```
#define INCL_DOSSEMAPHORES   /* Semaphore values */
#include <os2.h>
#include <stdio.h>

UCHAR       Name[40]; /* Semaphore name */
HMUX        hmux;     /* Muxwait semaphore handle */
ULONG       cSemRec;  /* Number of entries in pSemRec */
SEMRECORD   SemRec[2];   /* List of mutex or event semaphores */
ULONG       flAttr;   /* Creation attributes */
HEV         hev0;     /* Event semaphore handle */
HEV         hev1;     /* Event semaphore handle */
ULONG       ulUser0;  /* User semaphore ID value */
ULONG       ulUser1;  /* User semaphore ID value */
APIRET      rc;       /* Return code  */

   strcpy(Name,"\\SEM32\\MUXWAIT1");
                /* Name of the new system muxwait semaphore */

   cSemRec = 2;   /* Number of initial entries in muxwait */
                  /*   semaphore                          */

   SemRec[0].hsemCur = (PULONG) hev0;  /* Initialize the list   */
   SemRec[0].ulUser = ulUser0;         /*   of event semaphores */
   SemRec[1].hsemCur = (PULONG) hev1;  /*   that defines the    */
   SemRec[1].ulUser = ulUser1;         /*   muxwait semaphore   */

   flAttr = DCMW_WAIT_ANY;
                /* Indicate that the user of this muxwait */
                /*   semaphore will wait for any one of   */
                /*   its event semaphores to clear        */

   rc = DosCreateMuxWaitSem(Name, &hmux, cSemRec, SemRec,
                                  flAttr);
                /* On successful return, the hmux variable */
                /*   contains the handle of the new        */
                /*   system muxwait semaphore              */

   if (rc != 0)
     {
       printf("DosCreateMuxWaitSem error: return code = %ld", rc);
       return;
     }
```

DosCreateNPipe — Create Named Pipe

#define INCL_DOSNMPIPES

```
APIRET DosCreateNPipe (PSZ pszFileName, PHPIPE ppPipeHandle, ULONG ulOpenMode,
                       ULONG ulPipeMode, ULONG ulOutBufSize, ULONG ulInBufSize,
                       ULONG ulTimeOut)
```

DosCreateNPipe creates a named pipe.

Parameters

pszFileName (PSZ) — input

The ASCIIZ name of the pipe to be opened. Pipe names must include the prefix \PIPE\ and must conform to file-system naming conventions.

ppPipeHandle (PHPIPE) — output

A pointer to the variable in which the system returns the handle of the pipe that is created.

ulOpenMode (ULONG) — input

OpenMode contains the following bit fields:

Bit	Description
31 — 16	Reserved.
15	Reserved and must be 0.
14	Write-through bit. Possible values are: 0 = (NP_WRITEBEHIND) (0x0000) Write-behind to remote pipes is allowed. 1 = (NP_NOWRITEBEHIND) (0x4000) Write-behind to remote pipes is not allowed. This bit is meaningful only for a remote pipe. Occasionally, data written to a remote pipe is buffered locally and then sent across the network to the pipe at a later time. Setting the write-through bit ensures that data is sent to the remote pipe as soon as it is written.
13 — 8	Reserved.
7	Inheritance flag. Possible values are: 0 = (NP_INHERIT) (0x0000) The pipe handle is inherited by a child process. 1 = (NP_NOINHERIT) (0x0080) The pipe handle is private to the current process and cannot be inherited. This bit is not inherited by child processes.
6 — 3	Reserved and must be 0.
2 — 0	Access mode: The pipe access is defined as follows: 000 = (NP_ACCESS_INBOUND) (0x0000) Inbound pipe (client to server) 001 = (NP_ACCESS_OUTBOUND) (0x0001) Outbound pipe (server to client) 010 = (NP_ACCESS_DUPLEX) (0x0002) Duplex pipe (server to and from client) Any other value is invalid.

ulPipeMode (ULONG) — input

PipeMode contains the following bit fields:

Bit	Description

31 – 16 Reserved.

15 Blocking mode. Blocking mode is defined as either "blocking"or "nonblocking," as follows:

0 = (NP_WAIT) (0x0000) Blocking mode: DosRead and DosWrite block if no data is available.

1 = (NP_NOWAIT) (0x8000) Nonblocking mode: DosRead and DosWrite return immediately if no data is available.

DosRead normally blocks until at least partial data can be returned. DosWrite blocks by default until all of the requested bytes have been written. Nonblocking mode changes this behavior as follows:

DosRead returns immediately with ERROR_NO_DATA if no data is available.

DosWrite returns immediately with a value of 0 for *BytesWritten* if there is not enough buffer space available in the pipe; otherwise, the entire data area is transferred.

14 – 12 Reserved.

11 – 10 Type of named pipe. The pipe type is defined as follows:

00 = (NP_TYPE_BYTE) (0x0000) The pipe is a byte pipe; that is, data is written to the pipe as an undifferentiated stream of bytes.

01 = (NP_TYPE_MESSAGE) (0x0400) The pipe is a message pipe; that is, data is written to the pipe as messages. The system records the length of each message in the first two bytes of the message, which are called the *message header*. A header of all zeroes is reserved, and zero-length messages are not allowed.

9 – 8 Read mode. The read mode is defined as follows:

00 = (NP_READMODE_BYTE) (0x0000) Byte-read mode: Read the pipe as a byte stream.

01 = (NP_READMODE_MESSAGE) (0x0100) Message-read mode: Read the pipe as a message stream.

Message pipes can be read as either byte streams or message streams, depending on the value of this bit. Byte pipes can be read only as byte streams.

7 – 0 lCount (Instance count). When the first instance of a named pipe is created, lCount specifies how many instances (including the first instance) may be created. Possible values are:

Value	Definition
1	This is the only instance permitted (the pipe is unique).
1 < value < 255	The number of instances is limited to the value specified.
–1	(NP_UNLIMITED_INSTANCES) (0x00FF) The number of instances is unlimited.
0	Reserved value.
	The *lCount* parameter is ignored when specifying any instance of a pipe other than the first one. Subsequent attempts to create a pipe instance fail if the maximum number of allowed instances already exists. When multiple instances are allowed, multiple clients can simultaneously open the same pipe name; they will receive handles to distinct pipe instances.

ulOutBufSize (ULONG) – input

Tells the system how many bytes to allocate for the outbound (server to client) buffer.

ulInBufSize (ULONG) — input

Tells the system how many bytes to allocate for the inbound (client to server) buffer.

ulTimeOut (ULONG) — input

The default value for the *TimeOut* parameter of DosWaitNPipe. This value may be set only when the first instance of the pipe name is being created. If the value is 0, a system-wide default value (50 ms) is chosen.

Returns

Return Code.

DosCreateNPipe returns the following values:

0	NO_ERROR
3	ERROR_PATH_NOT_FOUND
8	ERROR_NOT_ENOUGH_MEMORY
84	ERROR_OUT_OF_STRUCTURES
87	ERROR_INVALID_PARAMETER
231	ERROR_PIPE_BUSY

Related Functions

- DosCallNPipe
- DosConnectNPipe
- DosDisConnectNPipe
- DosPeekNPipe
- DosQueryNPHState
- DosQueryNPipeInfo
- DosQueryNPipeSemState
- DosSetNPHState
- DosSetNPipeSem
- DosTransactNPipe
- DosWaitNPipe
- DosClose
- DosDupHandle
- DosOpen
- DosRead
- DosResetBuffer
- DosWrite

Example Code

This example creates a named pipe.

```
#define INCL_DOSNMPIPES   /* Named-pipe values */
#include <os2.h>
#include <stdio.h>

UCHAR   FileName[40]; /* Pipe name */
HPIPE   PipeHandle;  /* Pipe handle (returned) */
ULONG   OpenMode;    /* Open-mode parameters   */
ULONG   PipeMode;    /* Pipe-mode parameters   */
ULONG   OutBufSize;  /* Size of the out-buffer */
ULONG   InBufSize;   /* Size of the in-buffer  */
ULONG   TimeOut;     /* Default value for
                        DosWaitNPipe time-out
                        parameter   */
APIRET  rc;          /* Return code */

  strcpy(FileName,"\\PIPE\\PIPE1");
```

DosCreateNPipe — Create Named Pipe

```
OpenMode = NP_ACCESS_DUPLEX;
                  /* Specify full duplex access to named */
                  /*   pipe, no inheritance to child     */
                  /*   process, and no write-through     */
                  /*   (write-through only affects       */
                  /*   remote pipes)                     */

PipeMode = NP_WMESG | NP_RMESG | 0x01;
                  /* Specify block on Read/Write if no    */
                  /*   data available, message stream     */
                  /*   pipe for both reading and writing, */
                  /*   and an instance count of 1         */
                  /*   (only one instance of the named    */
                  /*   pipe can be created at a time)     */

OutBufSize = 4096; /* The outgoing buffer must be 4KB     */
                   /*   in size                           */

InBufSize = 2048; /* The incoming buffer must be 2KB      */
                  /*   in size                            */

TimeOut = 10000;  /* Time-out is 10 seconds (units are    */
                  /*   in milliseconds)                   */

rc = DosCreateNPipe(FileName, &PipeHandle, OpenMode,
                      PipeMode, OutBufSize, InBufSize,
                      TimeOut);

if (rc != 0)
  {
    printf("DosCreateNPipe error: return code = %ld", rc);
    return;
  }
```

DosCreatePipe – Create Unnamed Pipe

#define INCL_DOSQUEUES

APIRET DosCreatePipe (PHFILE ppReadHandle, PHFILE ppWriteHandle, ULONG ulPipeSize)

DosCreatePipe creates an unnamed pipe.

Parameters

ppReadHandle (PHFILE) – output

A pointer to a doubleword where the read handle for the pipe is returned.

ppWriteHandle (PHFILE) – output

A pointer to a doubleword where the write handle for the pipe is returned.

ulPipeSize (ULONG) – input

The amount of storage to reserve for the pipe.

Returns

Return Code.

DosCreatePipe returns the following values:

0	NO_ERROR
8	ERROR_NOT_ENOUGH_MEMORY

Related Functions

- DosClose
- DosDupHandle
- DosRead
- DosWrite

DosCreatePipe –
Create Unnamed Pipe

Example Code

This example creates an unnamed pipe. The current process may use the unnamed pipe for communication between itself and a child process.

```
#define INCL_DOSQUEUES   /* Queue values */
#include <os2.h>
#include <stdio.h>

HFILE    ReadHandle;    /* Pointer to the read handle */
HFILE    WriteHandle;   /* Pointer to the write handle */
ULONG    PipeSize;      /* Pipe size */
APIRET   rc;            /* Return code */

   PipeSize = 4096;     /* Ask for 4KB of internal storage */
                        /*   for the pipe                  */

   rc = DosCreatePipe(&ReadHandle, &WriteHandle, PipeSize);
                      /* On successful return, the ReadHandle  */
                      /*   variable contains the read handle   */
                      /*   for the pipe, and the WriteHandle   */
                      /*   variable contains the write handle  */
                      /*   for the pipe                        */

   if (rc != 0)
     {
       printf("DosCreatePipe error: return code = %ld", rc);
       return;
     }
```

DosCreateQueue – Create Queue

#define INCL_DOSQUEUES

```
APIRET DosCreateQueue (PHQUEUE ppRWHandle, ULONG ulQueueFlags,
                       PSZ pszQueueName)
```

DosCreateQueue creates a queue.

Parameters

ppRWHandle (PHQUEUE) – output

A pointer to the read/write handle of the queue that is being created. After DosCreateQueue returns, this handle may be used immediately by the requesting process; it is not necessary to issue DosOpenQueue.

ulQueueFlags (ULONG) – input

A set of flags that indicate which priority-ordering algorithm to use when placing elements in the queue, and whether or not to convert to 32-bit addresses the addresses of elements that are placed in the queue by 16-bit processes.

Priority-algorithm flag

Value	Definition
0	(QUE_FIFO) FIFO queue
1	(QUE_LIFO) LIFO queue
2	(QUE_PRIORITY) Priority queue: the requesting process specifies priority 0 to 15, with 15 being the highest priority.

Address-conversion flag

Value	Definition
0	(QUE_NOCONVERT_ADDRESS) The data addresses of elements placed in the queue by 16-bit processes are not converted.
4	(QUE_CONVERT_ADDRESS) The data addresses of elements placed in the queue by 16-bit processes are converted to 32-bit data addresses.

pszQueueName (PSZ) – input

A pointer to the ASCIIZ name of the queue. The name string must include \QUEUES\ as the first element of the path. For example, \QUEUES\RETRIEVE\CONTROL.QUE is a valid queue name. This name must be specified by a client process in a DosOpenQueue request before the client process can add an element to the queue.

Returns

Return Code.

DosCreateQueue returns the following values:

0	NO_ERROR
87	ERROR_INVALID_PARAMETER
332	ERROR_QUE_DUPLICATE
334	ERROR_QUE_NO_MEMORY
335	ERROR_QUE_INVALID_NAME

DosCreateQueue – Create Queue

Related Functions

- DosCloseQueue
- DosOpenQueue
- DosPeekQueue
- DosPurgeQueue
- DosQueryQueue
- DosReadQueue
- DosWriteQueue

Example Code

This example creates and opens a queue named SPECIAL.QUE for a server process.

```
#define INCL_DOSQUEUES   /* Queue values */
#include <os2.h>
#include <stdio.h>

#define QUE_NAME "\\QUEUES\\SPECIAL.QUE"

HQUEUE QueueHandle;
APIRET rc;                                  /* Return code */

   rc = DosCreateQueue(&QueueHandle,     /* Queue handle */
        QUE_FIFO |                        /* Ordering for elements */
        QUE_CONVERT_ADDRESS,        /* 16-bit address conversion */
        QUE_NAME);                        /* Queue name string  */
```

DosCreateThread – Create an Asynchronous Thread

#define INCL_DOSPROCESS

```
APIRET DosCreateThread (PTID ppThreadID, PFNTHREAD ppThreadAddr,
                        ULONG ulThreadArg, ULONG ulThreadFlags, ULONG ulStackSize)
```

DosCreateThread creates an asynchronous thread of execution under the current process.

Parameters

ppThreadID (PTID) – output

Address of a doubleword where the thread identifier of the created thread is returned.

ppThreadAddr (PFNTHREAD) – input

Address of the code to be executed when the thread begins execution. This function is called near, accepts a single parameter *ThreadArg*, and returns a doubleword exit status (see DosExit). Returning from the function without executing DosExit causes the thread to end. In this case, the exit status is the value in EAX when the thread ends.

ulThreadArg (ULONG) – input

An argument that is passed to the target thread routine as a parameter. It is usually a pointer to a parameter block.

ulThreadFlags (ULONG) – input

If bit 0 is set to 0, the new thread starts immediately. If bit 0 is set to 1, the thread is created in the suspended state, and the creator of the thread must issue DosResumeThread to start the new thread's execution. If bit 1 is set to 0, the system uses the default method for initializing the thread's stack. If bit 1 is set to 1, the system precommits all the pages in the stack. One page is 4KB.

ulStackSize (ULONG) – input

The size, in bytes, of the new thread's stack. The size is rounded up to the nearest page-size boundary or two pages, whichever is larger. The system allocates the stack upon creation of the thread, and deallocates it upon completion of the thread. The system provides dynamic stack storage commitment up to the limit specified in *StackSize* by using the guard-page technique. See **Remarks** for more details.

Returns

Return Code.

DosCreateThread returns the following values:

0	NO_ERROR
8	ERROR_NOT_ENOUGH_MEMORY
95	ERROR_INTERRUPT
115	ERROR_PROTECTION_VIOLATION
164	ERROR_MAX_THRDS_REACHED

Remarks

DosCreateThread creates an asynchronous thread of execution under the current process.

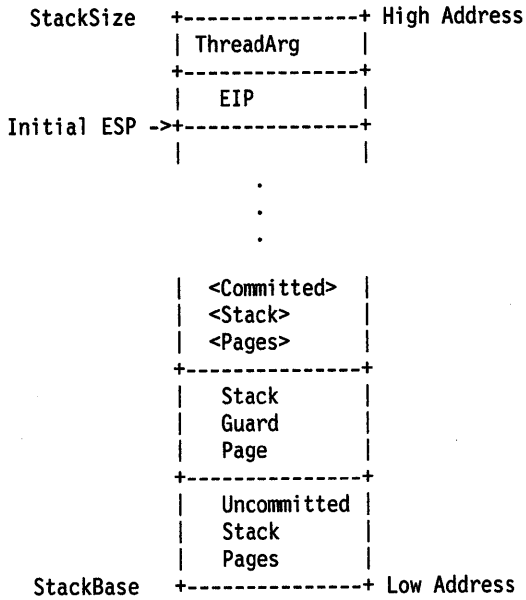
The operating system creates the first thread of a process when it starts the executable file. This thread is dispatched with a regular class priority. To start another thread of execution under the current process, the current thread issues DosCreateThread. The thread's initial dispatch point is the address specified for *ThreadAddr*. The started thread has a unique stack and register context and the same priority as the requesting thread.

DosCreateThread – Create an Asynchronous Thread

The created thread can access all files and resources owned by the parent process. The thread shares resources with other threads of the process. Any thread in the process can open a file or device, and any other thread can issue a read or write to that handle. This is also true for pipes, queues, and system-managed semaphores

When a thread is created, the system creates a Thread Information Block (TIB) to maintain per-thread information (TID, priority, and so on) in the user address space. See DosGetInfoBlocks for details on the TIB layout.

When a thread is created, its initial dispatch point is provided by *ThreadAddr*. This routine is invoked by Near Call, and when that routine returns or issues DosExit, the thread ends. The format of the thread's stack when the thread begins executing at *ThreadAddr* is:



When the system allocates the stack for the thread, a guard page is set up to facilitate dynamic stack growth. When a thread attempts to use stack in or "below" the guard page, a guard-page exception is generated. The default system action for this exception is to attempt to grow the stack by committing another page and moving the guard page. Since only a single guard page is committed at a time, and the page size of the 80386 processor is 4KB, a local stack allocation that is greater than 4KB must be handled by a stack probe that is performed by a compiler-generated routine.

The default stack commitment has one committed page, and a guard page is set up below the committed page. The pages beyond the guard page are uncommitted. If the system cannot allocate another guard page when the guard-page exception is not handled, a guard-page-allocation failure exception is generated. It is essential that applications and language runtime routines handle the guard-page-allocation exception. For more details on guard-page exception management, see DosSetExceptionHandler.

A thread started with DosCreateThread ends upon return of this call or when DosExit is issued. Any thread can temporarily stop the execution of other threads in its process with DosSuspendThread, DosResumeThread, DosEnterCritSec, and DosExitCritSec.

Any thread can also examine and change the priority at which it and other threads execute with DosGetInfoBlocks and DosSetPriority.

Related Functions

- DosExit
- DosKillThread
- DosResumeThread
- DosSuspendThread
- DosWaitThread

Example Code

This example creates a new thread within a process. Assume that the desired initial program address for the new thread has been stored in *ThreadAddr* already. Assume that *ThreadArg* has been set to contain the long parameter that is sent to the new thread.

```
#define INCL_DOSPROCESS     /* Process and thread values */
#include <os2.h>
#include <stdio.h>

TID         ThreadID;       /* New thread ID (returned) */
PFNTHREAD   ThreadAddr;     /* Program address */
ULONG       ThreadArg;      /* Parameter to thread routine */
ULONG       ThreadFlags;    /* When to start thread,
                               how to allocate stack */
ULONG       StackSize;      /* Size in bytes of new thread's stack */
APIRET      rc;             /* Return code */

   ThreadFlags = 0;         /* Indicate that the thread is to */
                            /*   be started immediately       */

   StackSize = 4096;        /* Set the size for the new */
                            /*   thread's stack         */

   rc = DosCreateThread(&ThreadID, ThreadAddr, ThreadArg,
                        ThreadFlags, StackSize);
                            /* On successful return, the */
                            /*   ThreadID variable will  */
                            /*   contain the TID of the  */
                            /*   newly-created thread    */

   if (rc != 0)
     {
       printf("DosCreateThread error: return code = %ld", rc);
       return;
     }
```

DosDebug –
Enable the Calling Program to Control Another Program for Debugging

---

```
#define INCL_DOSPROCESS
```

```
APIRET DosDebug (PVOID pDbgBuf)
```

DosDebug enables the calling application to control another application for debugging purposes.

Parameters

pDbgBuf (PVOID) – input

Address of a DosDebug Buffer Structure.

Returns

Return Code.

DosDebug returns the following values:

0	NO_ERROR
87	ERROR_INVALID_PARAMETER
95	ERROR_INTERRUPT
115	ERROR_PROTECTION_VIOLATION

Remarks

DosDebug allows one process (the debugger) to control the execution of another process that is being debugged (the debuggee).

A process must be selected for debugging when it is started. See DosExecPgm or DosStartSession for how this is done. Once a process has been selected for debugging, you must use DosDebug to control and examine its execution.

If no error is returned, a notification resides in the Debug Buffer structure. The **Command** field (**Cmd**) of the Debug Buffer structure determines which notification is set. The data returned with the notification varies, depending on the command passed in the **Command** field of the Debug Buffer structure.

If the return code is set to ERROR_INTERRUPT, a debug notification might have been lost, depending on the command that was interrupted.

DosDebug can also return with a return value set by another function.

For details about the commands that are available with DosDebug, see Appendix D, "DosDebug Commands" on page D-1.

For details about the notifications that are available with DosDebug, see Appendix E, "DosDebug Notifications" on page E-1.

Example Code

DosDebug provides a set of functions that permit one process to control another process. In this example, the calling process uses DosDebug to modify a word in a controlled process. Assume that all the necessary steps have been taken already so that the calling process controls the second process. Also assume that process identifier of the controlled process has been placed into *PID* already, the address of the word to be modified in the controlled process has been placed into *Addr*

already, and the value to be substituted in the controlled process has been placed into *Value* already.

```
#define INCL_DOSPROCESS   /* Process and thread values */
#include <os2.h>
#include <stdio.h>

struct debug_buffer
{
  ULONG   Pid;        /* Debuggee Process ID */
  ULONG   Tid;        /* Debuggee Thread ID */
  LONG    Cmd;        /* Command or Notification */
  LONG    Value;      /* Generic Data Value */
  ULONG   Addr;       /* Debuggee Address */
  ULONG   Buffer;     /* Debugger Buffer Address */
  ULONG   Len;        /* Length of Range */
  ULONG   Index;      /* Generic Identifier Index */
  ULONG   MTE;        /* Module Handle */
  ULONG   EAX;        /* Register Set */
  ULONG   ECX;
  ULONG   EDX;
  ULONG   EBX;
  ULONG   ESP;
  ULONG   EBP;
  ULONG   ESI;
  ULONG   EDI;
  ULONG   EFlags;
  ULONG   EIP;
  ULONG   CSLim;      /* Byte Granular Limits */
  ULONG   CSBase;     /* Byte Granular Base */
  UCHAR   CSAcc;      /* Access Bytes */
  UCHAR   CSAtr;      /* Attribute Bytes */
  USHORT  CS;
  ULONG   DSLim;
  ULONG   DSBase;
  UCHAR   DSAcc;
  UCHAR   DSAtr;
  USHORT  DS;
  ULONG   ESLim;
  ULONG   ESBase;
  UCHAR   ESAcc;
  UCHAR   ESAtr;
  USHORT  ES;
  ULONG   FSLim;
  ULONG   FSBase;
  UCHAR   FSAcc;
  UCHAR   FSAtr;
  USHORT  FS;
  ULONG   GSLim;
  ULONG   GSBase;
  UCHAR   GSAcc;
  UCHAR   GSAtr;
  USHORT  GS;
  ULONG   SSLim;
  ULONG   SSBase;
  UCHAR   SSAcc;
  UCHAR   SSAtr;
  USHORT  SS;
};

struct debug_buffer DbgBuf;
```

DosDebug —
Enable the Calling Program to Control Another Program for Debugging

```
                    /* Debug buffer */
ULONG   PID;        /* Process ID of controlled process */
ULONG   Addr;       /* Address within the controlled process */
LONG    Value;      /* Value to be substituted within the
                       controlled process */
APIRET  rc;         /* Return code */

   DbgBuf.Cmd = DBG_C_WriteMem;
                    /* Indicate that a Write Word command  */
                    /*   is requested                      */

   DbgBuf.Pid = PID;
                    /* Place PID of controlled process */
                    /*   into the debug buffer         */

   DbgBuf.Addr = Addr;
                /* Place the word address (within the          */
                /*   controlled process) into the debug buffer */

   DbgBuf.Value = Value;
                  /* Place the value to be updated into the     */
                  /*   specified word of the controlled process */

   rc = DosDebug(&DbgBuf);

   if (rc != 0)
     {
       printf("DosDebug error: return code = %ld", rc);
       return;
     }
```

DosDelete – Remove a File Name from a Directory

#define INCL_DOSFILEMGR

APIRET DosDelete (PSZ pszFileName)

DosDelete removes a file name from a directory. The deleted file may be recoverable.

Parameters

pszFileName (PSZ) – input

Address of the name of the file to be deleted.

Returns

Return Code.

DosDelete returns the following values:

0	NO_ERROR
2	ERROR_FILE_NOT_FOUND
3	ERROR_PATH_NOT_FOUND
5	ERROR_ACCESS_DENIED
26	ERROR_NOT_DOS_DISK
32	ERROR_SHARING_VIOLATION
36	ERROR_SHARING_BUFFER_EXCEEDED
87	ERROR_INVALID_PARAMETER
206	ERROR_FILENAME_EXCED_RANGE

Remarks

Global file-name characters are not permitted in the name of the file to be deleted.

Read-only files cannot be deleted by DosDelete. To delete a read-only file, you must first issue DosSetFileInfo to change the file's read-only attribute to zero, then delete the file.

If a storage directory for the drive has been defined with the SET DELDIR command, the UNDELETE command may recover the deleted file.

DosDelete cannot be used to delete directories. Issue DosDeleteDir to delete a directory.

Related Functions

- DosDeleteDir
- DosForceDelete
- DosSetFileInfo

DosDelete –
Remove a File Name from a Directory

Example Code

This example deletes a file named test.dat from the current directory.

```
#define INCL_DOSFILEMGR   /* File Manager values */
#include <os2.h>
#include <stdio.h>

#define FILE_DELETE "test.dat"

APIRET  rc;             /* Return code  */

   rc = DosDelete(FILE_DELETE);   /* File path name */

   if (rc != 0)
     {
       printf("DosDelete error: return code = %ld",rc);
       return;
     }
```

DosDeleteDir – Delete a Directory

```
#define INCL_DOSFILEMGR

APIRET DosDeleteDir (PSZ pszDirName)
```

DosDeleteDir removes a subdirectory from the specified disk.

Parameters

pszDirName (PSZ) – input

Address of the fully qualified path name of the subdirectory to be removed.

Returns

Return Code.

DosDeleteDir returns the following values:

0	NO_ERROR
2	ERROR_FILE_NOT_FOUND
3	ERROR_PATH_NOT_FOUND
5	ERROR_ACCESS_DENIED
16	ERROR_CURRENT_DIRECTORY
26	ERROR_NOT_DOS_DISK
87	ERROR_INVALID_PARAMETER
108	ERROR_DRIVE_LOCKED
206	ERROR_FILENAME_EXCED_RANGE

Remarks

The subdirectory must be empty; that is, it cannot have hidden files or directory entries other than the "." and ".." entries. To delete files, use DosDelete.

The root directory and current directory cannot be removed.

Related Functions

- DosDelete
- DosForceDelete

DosDeleteDir – Delete a Directory

Example Code

This example deletes a subdirectory. Assume that the subdirectory was empty before the attempt to remove it.

```
#define INCL_DOSFILEMGR   /* File Manager values */
#include <os2.h>
#include <stdio.h>

UCHAR   DirName[40]; /* New directory name string */
APIRET  rc;          /* Return code */

   strcpy(DirName,"\\PROG\\SRC\\UTIL");
                     /* Directory to be deleted */

   rc = DosDeleteDir(DirName);

   if (rc != 0)
     {
       printf("DosDeleteDir error: return code = %ld", rc);
       return;
     }
```

DosDeleteMuxWaitSem – Delete MuxWait Semaphore

#define INCL_DOSSEMAPHORES

APIRET DosDeleteMuxWaitSem (HMUX hmux, HSEM hsem)

DosDeleteMuxWaitSem deletes an event semaphore or a mutex semaphore from a muxwait-semaphore list.

Parameters

hmux (HMUX) – input

The handle of the muxwait semaphore that is to have a semaphore deleted from its semaphore record list.

hsem (HSEM) – input

The handle of the semaphore that is to be deleted from the semaphore record list of the muxwait semaphore.

Returns

Return Code.

DosDeleteMuxWaitSem returns the following values:

0	NO_ERROR
6	ERROR_INVALID_HANDLE
286	ERROR_EMPTY_MUXWAIT

Remarks

DosDeleteMuxWaitSem deletes an event semaphore or a mutex semaphore from the existing list of semaphores in a muxwait semaphore.

This function can be called by any thread in the process that created the muxwait semaphore. Other processes can also call this function, but they must first gain access to the muxwait semaphore by calling DosOpenMuxWaitSem.

Related Functions

- DosAddMuxWaitSem
- DosCloseMuxWaitSem
- DosCreateMuxWaitSem
- DosOpenMuxWaitSem
- DosQueryMuxWaitSem
- DosWaitMuxWaitSem

DosDeleteMuxWaitSem – Delete MuxWait Semaphore

Example Code

This example deletes an event or mutex semaphore from an existing muxwait semaphore. Assume that the handle of the muxwait semaphore has been placed into *hmux* already. Assume that the handle of the event semaphore has been placed into *hsem* already.

```
#define INCL_DOSSEMAPHORES   /* Semaphore values */
#include <os2.h>
#include <stdio.h>

HMUX    hmux;    /* Muxwait semaphore handle */
HSEM    hSem;    /* Handle of semaphore to be
                    deleted from muxwait list */
APIRET  rc;      /* Return code */

   rc = DosDeleteMuxWaitSem(hmux, hSem);

   if (rc != 0)
     {
       printf("DosDeleteMuxWaitSem error: return code = %ld", rc);
       return;
     }
```

DosDevConfig – Get Information about Attached Devices

#define INCL_DOSPROCESS

APIRET DosDevConfig (PVOID pDeviceInfo, ULONG ulDeviceType)

DosDevConfig gets information about attached devices.

Parameters

pDeviceInfo (PVOID) – output

Address of the area where the information is returned.

ulDeviceType (ULONG) – input

Indicates what device information to return, as follows:

Item	Defined Name	Size	Returned Device Information
0	DEVINFO_PRINTER	BYTE	Number of attached printers.
1	DEVINFO_RS232	BYTE	Number of RS232 ports.
2	DEVINFO_FLOPPY	BYTE	Number of diskette drives.
3	DEVINFO_COPROCESSOR	BYTE	Presence of math coprocessor hardware: 0 = No coprocessor hardware. 1 = Coprocessor hardware installed.
4	DEVINFO_SUBMODEL	BYTE	PC Submodel Type. The returned information is the system submodel byte.
5	DEVINFO_MODEL	BYTE	PC Model Type. The returned information is the system model byte.
6	DEVINFO_ADAPTER	BYTE	Type of primary display adapter: 0 = Monochrome or printer adapter. 1 = Other.

Returns

Return Code.

DosDevConfig returns the following values:

0	NO_ERROR
87	ERROR_INVALID_PARAMETER

Related Functions

- DosBeep
- DosDevIOCtl
- DosPhysicalDisk

DosDevConfig –
Get Information about Attached Devices

Example Code

This example gets information about the model type, monitor, and coprocessor, and displays it.

```
#define INCL_DOSDEVICES   /* Device values */
#include <os2.h>
#include <stdio.h>

#define MACHINE_MODEL 5
#define DISPLAY_TYPE 6
#define FIND_COPROCESSOR 3

BYTE    DeviceInfo;  /* Device information */
APIRET  rc;          /* Return code */

   if(!DosDevConfig(&DeviceInfo,            /* Returned information */
                    MACHINE_MODEL))          /* Device type item number */
      printf("Model Type %d ",DeviceInfo);

   if(!DosDevConfig(&DeviceInfo,            /* Returned information */
                    DISPLAY_TYPE))           /* Device type item number */
      if (DeviceInfo)
         printf("Color display ");
      else
         printf("Mono display ");

   if(!DosDevConfig(&DeviceInfo,            /* Returned information */
                    FIND_COPROCESSOR))       /* Device type item number */
      if (DeviceInfo)
         printf("Coprocessor");
      else
         printf("No Coprocessor");
```

DosDevIOCtl – Perform Control Function on a Device Specified by an Opened Device Handle

#define INCL_DOSPROCESS

```
APIRET DosDevIOCtl (HFILE DevHandle, ULONG ulCategory, ULONG ulFunction,
                    PVOID pParmList, ULONG ulParmLengthMax,
                    PULONG pParmLengthInOut, PVOID pDataArea,
                    ULONG ulDataLengthMax, PULONG pDataLengthInOut)
```

DosDevIOCtl performs control functions on a device specified by an opened device handle.

Parameters

DevHandle (HFILE) – input (hDevice) 5

Device handle returned by DosOpen, or a standard (open) device handle.

ulCategory (ULONG) – input (usCategory) 4

Device category. The valid range is 0 to 255.

ulFunction (ULONG) – input (usFunction) 3

Device-specific function code. The valid range is 0 to 255.

pParmList (PVOID) – input (pvParms) 2

Address of the command-specific argument list.

* **ulParmLengthMax** (ULONG) – input sizeof (pvParms)

Length, in bytes, of *ParmList*. This is the maximum length of the data to be returned in *ParmList*. *ParmLengthInOut* may be larger than this on input, but not on output.

* **pParmLengthInOut** (PULONG) – input/output (NULL ?)

On input, a pointer to the length, in bytes, of the parameters passed by the application in *Parmlist*. On output, a pointer to the length, in bytes, of the parameters returned.

If this function returns ERROR_BUFFER_OVERFLOW, then *ParmLengthInOut* points to the size of the buffer required to hold the parameters returned. No other data is returned in this case.

pDataArea (PVOID) – input (pvData) 1

Address of the data area.

* **ulDataLengthMax** (ULONG) – input sizeof pvData

Length, in bytes, of *DataArea*. This is the maximum length of the data to be returned in *DataArea*. *DataLengthInOut* may be larger than this on input, but not on output.

* **pDataLengthInOut** (PULONG) – input/output (NULL)?

On input, a pointer to the length, in bytes, of the data passed by the application in *DataArea*. On output, a pointer to the length, in bytes, of the data returned.

If this function returns ERROR_BUFFER_OVERFLOW, then *DataLengthInOut* points to the size of the buffer required to hold the data returned.

```
                USHORT DosDevIOCtl      16 bit function
1               (PVOID  pvData,
2                PUOID  pvParms,
3                USHORT usFunction,
4                USHORT usCategory,
5                HFILE  hDevice )
```

DosDevIOCtl – Perform Control Function on a Device Specified by an Opened Device Handle

Returns

Return Code.

DosDevIOCtl returns the following values:

0	NO_ERROR
1	ERROR_INVALID_FUNCTION
6	ERROR_INVALID_HANDLE
15	ERROR_INVALID_DRIVE
31	ERROR_GEN_FAILURE
87	ERROR_INVALID_PARAMETER
111	ERROR_BUFFER_OVERFLOW
115	ERROR_PROTECTION_VIOLATION
117	ERROR_INVALID_CATEGORY
119	ERROR_BAD_DRIVER_LEVEL
163	ERROR_UNCERTAIN_MEDIA
165	ERROR_MONITORS_NOT_SUPPORTED

Remarks

Values returned in the range hex FF00 through FFFF are user-dependent error codes. Values returned in the range hex FE00 through FEFF are device-driver-dependent error codes.

This function provides a generic, expandable IOCtl facility.

A null (zero) value for *DataArea* specifies that this parameter is not defined for the generic IOCtl function being specified. A null value for *DataArea* causes the values passed in *DataLengthMax* and *DataLengthInOut* to be ignored.

A null (zero) value for *ParmList* specifies that this parameter is not defined for the generic IOCtl function being specified. A null value for *ParmList* causes the values passed in *ParmLengthMax* and *ParmLengthInOut* to be ignored.

The kernel formats a generic IOCtl packet and calls the device driver. Because OS/2 Version 1.0 and Version 1.1 device drivers do not understand generic IOCtl packets with *DataLengthMax, DataLengthInOut, ParmLengthMax*, and *ParmLengthInOut*, the kernel does not pass these fields to the device driver. Device drivers that are marked as level 2 or higher must support receipt of the generic IOCtl packets with associated length fields.

Do not pass a non-null pointer with a zero length.

Refer to the *Physical Device Driver Reference* for a complete listing of the generic IOCtl control functions (the IOCtl interface).

Related Functions

- DosBeep
- DosDevConfig
- DosPhysicalDisk

Example Code

This example demonstrates how a process can issue control functions to a device. Assume that the calling process has placed an appropriate device handle into *DevHandle*. Assume that the specified device recognizes a category code of hex 83 and a function code of hex 63. Assume that the specified device control function that is utilized in this example requires no input parameter string or input data, and returns no parameters. Assume that it returns a set of data to the user in a user-supplied data buffer.

DosDevIOCtl – Perform Control Function on a Device Specified by an Opened Device Handle

```
#define INCL_DOSDEVICES   /* Device values */
#include <os2.h>
#include <stdio.h>

HFILE   DevHandle;       /* Device handle specifies the device */
ULONG   Category;        /* Device category */
ULONG   Function;        /* Device function */
PVOID   ParmList;        /* Command-specific argument list */
ULONG   ParmLengthMax;   /* Command arguments list max length */
ULONG   ParmLengthInOut; /* Command arguments length (returned) */
UCHAR   DataArea[200];   /* Data area */
ULONG   DataLengthMax;   /* Data area maximum length */
ULONG   DataLengthInOut; /* Data area length (returned) */
APIRET  rc;              /* Return code */

   Category = 0x83;     /* Specify device driver category hex 83 */

   Function = 0x63;     /* Specify device driver function hex 63 */

   ParmList = 0;         /* Indicate that no input parameters */
   ParmLengthInOut = 0;  /*   are being passed to the device, */
   ParmLengthMax = 0;    /*   and that there is no buffer to  */
                         /*   receive parameters back from    */
                         /*   the device                      */

   DataLengthInOut = 0;  /* Indicate that no input data is */
                         /*   being passed to the device   */

   DataLengthMax = 200;  /* Indicate the maximum amount of data */
                         /*   (in bytes) that can be returned   */
                         /*   to the caller by the device       */

   rc = DosDevIOCtl(DevHandle, Category, Function, ParmList,
                    ParmLengthMax, &ParmLengthInOut, DataArea,
                    DataLengthMax, &DataLengthInOut);
                      /* On successful return, the DataArea  */
                      /*   buffer contains the data passed   */
                      /*   back by the device, and the       */
                      /*   DataLengthInOut variable contains */
                      /*   the size of that data.  In this   */
                      /*   example, the device is assumed to */
                      /*   pass back no parameters, so the   */
                      /*   ParmLengthInOut variable will     */
                      /*   remain unchanged.                 */

   if (rc != 0)
     {
       printf("DosDevIOCtl error: return code = %ld", rc);
       return;
     }
```

DosDisConnectNPipe – Disconnect Named Pipe

#define INCL_DOSNMPIPES

APIRET DosDisConnectNPipe (HPIPE hpipeHandle)

DosDisConnectNPipe acknowledges that a client process has closed a named pipe.

Parameters

hpipeHandle (HPIPE) – input

The named-pipe handle to disconnect (returned to the server process by DosCreateNPipe).

Returns

Return Code.

DosDisConnectNPipe returns the following values:

0	NO_ERROR
109	ERROR_BROKEN_PIPE
230	ERROR_BAD_PIPE

Remarks

DosDisConnectNPipe is issued by a server process to acknowledge that a client process has closed a named pipe. (If a client process tries to issue DosDisConnectNPipe, ERROR_BAD_PIPE is returned.) The pipe cannot be opened by another client process until the server process issues this function, followed by DosConnectNPipe.

Until the client's close has been acknowledged, the server process will receive a value of zero for *BytesRead* (indicating end-of-file) if it tries to read from the pipe, and ERROR_BROKEN_PIPE if it tries to write to it. Clients that attempt to open the pipe receive ERROR_PIPE_BUSY.

Any threads that are blocked on the pipe are awakened by DosDisConnectNPipe. A thread that is blocked on a DosWrite request returns ERROR_BROKEN_PIPE. A thread that is blocked on a DosRead request returns a value of zero for *BytesRead*.

If the client end of the pipe is open when DosDisConnectNPipe is issued, it is forced to close, and the client receives an error code on its next operation. Note that when a client is forced to close in this manner, data may be discarded before it has been read by the client.

DosDisConnectNPipe makes the client's handle invalid, but it does not free the handle. Therefore, a client that is forced off a pipe by DosDisConnectNPipe must still issue DosClose to free the handle resource.

Related Functions

- DosCallNPipe
- DosConnectNPipe
- DosCreateNPipe
- DosPeekNPipe
- DosQueryNPHState
- DosQueryNPipeInfo
- DosQueryNPipeSemState
- DosSetNPHState
- DosSetNPipeSem
- DosTransactNPipe
- DosWaitNPipe

- DosClose
- DosDupHandle
- DosOpen
- DosRead
- DosResetBuffer
- DosWrite

Example Code

This example forces a named pipe to close. Assume that a previous call to DosCreateNPipe provided the named pipe handle that is contained in *Handle*.

```
#define INCL_DOSNMPIPES   /* Named-pipe values */
#include <os2.h>
#include <stdio.h>

HPIPE   Handle;      /* Pipe handle */
APIRET  rc;          /* Return code */

   rc = DosDisConnectNPipe(Handle);

   if (rc != 0)
     {
       printf("DosDisconnectNPipe error: return code = %ld", rc);
       return;
     }
```

DosDupHandle –
Get a New Handle for an Open File

#define INCL_DOSFILEMGR

```
APIRET DosDupHandle (HFILE OldFileHandle, PHFILE ppNewFileHAndle)
```

DosDupHandle gets a new handle for an open file.

Parameters

OldFileHandle (HFILE) – input

File handle to duplicate, or alias.

ppNewFileHAndle (PHFILE) – input/output

Address of a doubleword that, on input, describes how the handle is to be duplicated and, on output contains the duplicate file handle.

On input, the value and meaning of this doubleword are as follows:

Value	Definition
hex FFFFFFFF	Allocate a new file handle and return it here.
not hex FFFFFFFF	Assign this value as the new file handle. A valid value is any of the handles assigned to standard I/O, or the handle of a file currently opened by the process.

On output, a value of hex FFFFFFFF returns a value for *NewFileHandle*, allocated by the operating system.

Returns

Return Code.

DosDupHandle returns the following values:

0	NO_ERROR
4	ERROR_TOO_MANY_OPEN_FILES
6	ERROR_INVALID_HANDLE
114	ERROR_INVALID_TARGET_HANDLE

Remarks

Duplicating the handle duplicates and ties all handle-specific information between *OldFileHandle* and *NewFileHandle*. For example, if you move the read/write pointer of either handle with DosRead, DosSetFilePtr, or DosWrite, the pointer for the other handle also is changed.

The valid values for *NewFileHandle* include the following handles for standard I/O, which are always available to the process:

hex 00000000	Standard input
hex 00000001	Standard output
hex 00000002	Standard error.

If a file-handle value of a currently open file is specified in *NewFileHandle*, the file handle is closed before it is redefined as the duplicate of *OldFileHandle*. Avoid using arbitrary values for *NewFileHandle*.

Issuing DosClose for a file handle does not affect the duplicate handle.

DosDupHandle – Get a New Handle for an Open File

Related Functions

- DosClose
- DosCreatePipe
- DosOpen
- DosRead
- DosSetFHState
- DosSetFilePtr
- DosSetMaxFH
- DosSetRelMaxFH
- DosWrite

Example Code

This example opens a file, creates a second file handle, then closes the file with the second handle.

```
#define INCL_DOSFILEMGR   /* File Manager values */
#include <os2.h>
#include <stdio.h>

#define OPEN_FILE 0x01
#define CREATE_FILE 0x10
#define FILE_ARCHIVE 0x20
#define FILE_EXISTS OPEN_FILE
#define FILE_NOEXISTS CREATE_FILE
#define DASD_FLAG 0
#define INHERIT 0x80
#define WRITE_THRU 0
#define FAIL_FLAG 0
#define SHARE_FLAG 0x10
#define ACCESS_FLAG 0x02

#define FILE_NAME "test.dat"
#define FILE_SIZE 800L
#define FILE_ATTRIBUTE FILE_ARCHIVE
#define EABUF 0L

HFILE   FileHandle;
HFILE   NewHandle;
ULONG   Wrote;
ULONG   Action;
PSZ     FileData[100];
APIRET  rc;             /* Return code */

   Action = 2;
   strcpy(FileData, "Data...");

   rc = DosOpen(FILE_NAME,                 /* File path name */
                &FileHandle,               /* File handle */
                &Action,                   /* Action taken */
                FILE_SIZE,                 /* File primary allocation */
                FILE_ATTRIBUTE,            /* File attribute */
                FILE_EXISTS | FILE_NOEXISTS,    /* Open function type */
                DASD_FLAG | INHERIT |           /* Open mode of the file */
                WRITE_THRU | FAIL_FLAG |
                SHARE_FLAG | ACCESS_FLAG,
                EABUF);                    /* No extended attributes */

   if (rc != 0)
     {
       printf("DosOpen error: return code = %ld",rc);
       return;
     }
```

DosDupHandle —
Get a New Handle for an Open File

```
rc = DosDupHandle(FileHandle,      /* Existing file handle */
                  &NewHandle);     /* New file handle */

if (rc != 0)
  {
    printf("DosDupHandle error: return code = %ld",rc);
    return;
  }

rc = DosClose(NewHandle);          /* Close with new file handle */

if (rc != 0)
  {
    printf("DosClose error: return code = %ld",rc);
    return;
  }
```

DosEditName – Edit File and Directory Name

#define INCL_DOSFILEMGR

```
APIRET DosEditName (ULONG ulEditLevel, PSZ pszSourceString, PSZ pszEditString,
                    PBYTE pbTargetBuf, ULONG ulTargetBufLen)
```

DosEditName edits file and directory names indirectly by transforming one ASCII string into another, using global file-name characters for editing or search operations on the string.

Parameters

ulEditLevel (ULONG) – input

The level of editing semantics to use in transforming the source string. If the value of *EditLevel* is 1, the system uses editing semantics for OS/2 Version 1.2.

pszSourceString (PSZ) – input

Address of the ASCIIZ string to transform. Global file-name characters are specified only in the subdirectory or file-name component of the path name, and are interpreted as search characters. *SourceString* should contain only the component of the path name to edit, not the entire path.

pszEditString (PSZ) – input

Address of the ASCIIZ string to use for editing. Global file-name characters specified in the edit string are interpreted as editing characters. Because only the name component of a path name is transformed, this string does not include the path component.

pbTargetBuf (PBYTE) – output

Address of the buffer for the resulting ASCIIZ string.

ulTargetBufLen (ULONG) – input

The length of the buffer, in bytes, for the resulting string.

Returns

Return Code.

DosEditName returns the following values:

0	NO_ERROR
87	ERROR_INVALID_PARAMETER
123	ERROR_INVALID_NAME

Remarks

DosEditName is used to search for and edit names of files and subdirectories. Typically, it is used in conjunction with such functions as DosMove and DosCopy, which do not permit the use of global file-name characters, to perform repetitive operations on files.

An example of an editing operation is: *SourceString* = "foo.bar"; *EditString* = "*.baz"; result = "FOO.BAZ." In the editing process, the string is changed to uppercase.

Global file-name characters have two uses: searching and editing. If they are specified in *SourceString*, they are interpreted as search characters; in *EditString*, they are interpreted as editing characters. This difference can be illustrated with an example using the COPY utility. The user types the following:

```
copy *.old *.new
```

DosEditName –
Edit File and Directory Name

In the source, "*" acts as a search character and determines which files to return to the user. In the target, "*" functions as an editing character by constructing new names for the matched files.

When used as search characters in *SourceString*, global file-name characters simply match files and behave like any other search characters. They have the following meanings:

. The period (.) has no special meaning itself, but "?" gives it one.

* The asterisk will form a match with any character, including a blank, or with the absence of a character. The matching operation does not cross the null character or the backslash (\), which means that only the file name is matched, not an entire path.

? The question mark matches 1 character, unless what it would match is a "." or the terminating null characters, in which case it matches 0 characters. It also does not cross "\".

Any character other than * and ? matches itself, including ".".

Searching is not case-sensitive.

If a file name does not have a period (.), an implicit one is automatically appended to the end during searching operations. For example, searching for "foo." would return "foo".

When used in *EditString*, global file-name characters have the following meanings:

. The period (.) in the target synchronizes pointers. It causes the source pointer to match a corresponding pointer to the period in the target. Counting starts from the left of the pointers.

? The question mark copies one character, unless what it would copy is a period (.), in which case it copies no characters. It also copies no characters when the end of the source string is reached.

* The asterisk copies characters from the source to the target until it finds a source character that matches the character following it in the target.

Editing is case-insensitive and case-preserving. If conflicts arise between the case of the source and that of the editing string, the case of the editing string is used, for example:

```
source string:      "file.txt"
editing string:     "*E.TMP"
destination string: "filE.TMP"

copy file.txt  *E.tmp  ->  filE.tmp
```

Related Functions

- DosCopy
- DosMove
- DosQuerySysInfo

Example Code

This example transforms a source string into a destination string through the use of an editing string. Both the source and editing strings can contain global file name characters. These global file name characters control the form of the string transformation.

```
#define INCL_DOSFILEMGR   /* File Manager values */
#include <os2.h>
#include <stdio.h>

ULONG   EditLevel;         /* Level of meta editing semantics  */
UCHAR   SourceString[80];  /* String to transform */
UCHAR   EditString[80];    /* Editing string */
UCHAR   TargetBuf[200];    /* Destination string buffer */
ULONG   TargetBufLen;      /* Destination string buffer length */
APIRET  rc;                /* Return code */

   EditLevel = 1;          /* Use OS/2 1.2 editing semantics */

   strcpy(SourceString,"xyz.src");

   strcpy(EditString,"*.bak");

   TargetBufLen = 200;     /* Length of target buffer (bytes) */

   rc = DosEditName(EditLevel, SourceString, EditString, TargetBuf,
                    TargetBufLen);
                      /* On successful return, the   */
                      /*   transformed source string */
                      /*   is found within the       */
                      /*   target buffer.  In this   */
                      /*   example, TargetBuf should */
                      /*   hold the string "XYZ.BAK" */

   if (rc != 0)
     {
       printf("DosEditName error: return code = %ld", rc);
       return;
     }
```

DosEnterCritSec – Disable Thread Switching

#define INCL_DOSPROCESS

APIRET DosEnterCritSec ()

DosEnterCritSec disables thread switching for the current process.

Parameters

Returns

Return Code.

DosEnterCritSec returns the following values:

0	NO_ERROR
309	ERROR_INVALID_THREADID
484	ERROR_CRITSEC_OVERFLOW

Remarks

DosEnterCritSec causes other threads in the process to block themselves and give up their time slice. After a DosEnterCritSec request is made, no dynamic link calls should be made until the corresponding DosExitCritSec is completed.

If a signal occurs, thread 1 begins execution to process the signal even though another thread in the process has a DosEnterCritSec active. Thread 1 of a process is its initial thread of execution, not a thread created with DosCreateThread. Any processing done by thread 1 to satisfy the signal must not include accessing the critical resource intended to be protected by DosEnterCritSec.

A count is maintained of the number of times DosEnterCritSec is issued without a corresponding DosExitCritSec. The count is incremented by DosEnterCritSec and decremented by DosExitCritSec. Normal thread dispatching is not restored until the count is zero. The outstanding DosEnterCritSec count is maintained in a word. If an overflow occurs, the count is set to the maximum value, no operation is performed, and the request returns with ERROR_CRITSEC_OVERFLOW.

A thread can also execute code without having to give up time slices to other threads in its process if it requests a priority class that is higher than those of the other threads. A thread's priority is examined with DosGetInfoBlocks, and changed with DosSetPriority.

ERROR_INVALID_THREADID is returned when an invalid attempt is made to enter a critical section of code in a signal handler or exception handler.

ERROR_INVALID_THREADID is also returned when a dynamic link library (DLL) routine incorrectly issues DosEnterCritSec.

Related Functions

- DosCreateThread
- DosExitCritSec

Example Code

This example enters a section that will not be pre-empted, performs a simple task, and then exits quickly.

```
#define INCL_DOSPROCESS
#include <os2.h>
#include <stdio.h>

BOOL   flag;

   DosEnterCritSec();        /* Enter critical code section */
   flag = TRUE;              /* Perform some work */
   DosExitCritSec();         /* Exit critical code section */
```

DosEnterMustComplete – Enter Must Complete

#define INCL_DOSEXCEPTIONS

APIRET DosEnterMustComplete (PULONG ppulNesting)

DosEnterMustComplete provides entry into a section of code in which asynchronous exceptions are held.

Parameters

ppulNesting (PULONG) – output

A pointer to a value that is equal to the number of DosEnterMustComplete requests minus the number of DosExitMustComplete requests for the current thread.

Returns

Return Code.

DosEnterMustComplete returns the following values:

0	NO_ERROR
650	ERROR_NESTING_TOO_DEEP

Remarks

DosEnterMustComplete notifies the system that the thread is entering a section of code in which asynchronous exceptions (signals and asynchronous process terminations) are to be held, rather than being immediately delivered to the thread.

For a detailed list of the system exceptions, see Appendix C, "System Exceptions" on page C-1.

Related Functions

- DosAcknowledgeSignalException
- DosExitMustComplete
- DosRaiseException
- DosSendSignalException
- DosSetExceptionHandler
- DosSetSignalExceptionFocus
- DosUnsetExceptionHandler
- DosUnwindException

Example Code

This example shows how a thread can notify the system that the thread is entering a section of code in which asynchronous exceptions (signals and asynchronous process terminations) are to be held, rather than being delivered to the thread immediately.

Assume that the unsigned long variable *NestingLevel* is a global program variable that is used to maintain the number of nested calls to DosEnterMustComplete that are currently in effect for the section of code. Assume that *NestingLevel* was originally initialized to zero, and that its value is subsequently incremented by calls to DosEnterMustComplete, and decremented by calls to DosExitMustComplete.

```
#define INCL_DOSEXCEPTIONS   /* Exception values */
#include <os2.h>
#include <stdio.h>

extern ULONG   NestingLevel;  /* Number of signal to be acknowledged */
APIRET  rc;                   /* Return code */

   rc = DosEnterMustComplete(&NestingLevel);

   if (rc != 0)
     {
       printf("DosEnterMustComplete error: return code = %ld",
                       rc);
       return;
     }
```

DosEnumAttribute –
Identify Names and Lengths of Extended Attributes

#define INCL_DOSFILEMGR

```
APIRET DosEnumAttribute (ULONG ulRefType, PVOID pFileRef, ULONG ulEntryNum,
                         PVOID pEnumBuf, ULONG ulEnumBufSize, PULONG pEnumCnt,
                         ULONG ulInfoLevel)
```

DosEnumAttribute identifies names and lengths of extended attributes for a specific file or subdirectory.

Parameters

ulRefType (ULONG) – input

A value that indicates whether *FileRef* points to a handle or to an ASCIIZ name:

Value	Definition
0	(ENUMEA_REFTYPE_FHANDLE): Handle of a file.
1	(ENUMEA_REFTYPE_PATH): ASCIIZ name of a file or subdirectory.

pFileRef (PVOID) – input

Address of the handle of a file returned by DosOpen; or the ASCIIZ name of a file or subdirectory.

ulEntryNum (ULONG) – input

Ordinal of an entry in the file object's EA list, which indicates where in the list to begin the return of EA information. The value 0 is reserved. A value of 1 indicates the file object's first EA; a value of 2, the second; and so on.

pEnumBuf (PVOID) – output

Address of the buffer where EA information is returned. Level 1 information is returned in a data structure of type DENA2.

ulEnumBufSize (ULONG) – input

The length, in bytes, of the buffer pointed to by *EnumBuf*.

pEnumCnt (PULONG) – input/output

On input, the address of a doubleword containing the number of EAs for which information is requested. A value of –1 requests that information be returned for as many EAs whose information fits in EnumBuf.

On output, this is the address of a doubleword containing the actual number of EAs for which information is returned. When this value is greater than 1, enumerated information is returned in a series of DENA2 data structures. Each data structure is aligned on a doubleword boundary. The first field of the data structure (oNextEntryOffset) contains the number of bytes to the beginning of the next data structure. A value of zero for oNextEntryOffset indicates that this is the last data structure.

ulInfoLevel (ULONG) – input

Level of information required. Only the value 1 (ENUMEA_LEVEL_NO_VALUE) can be specified, indicating return of level 1 information.

DosEnumAttribute — Identify Names and Lengths of Extended Attributes

Returns

Return Code.

DosEnumAttribute returns the following values:

0	NO_ERROR
3	ERROR_PATH_NOT_FOUND
5	ERROR_ACCESS_DENIED
6	ERROR_INVALID_HANDLE
8	ERROR_NOT_ENOUGH_MEMORY
87	ERROR_INVALID_PARAMETER
111	ERROR_BUFFER_OVERFLOW
124	ERROR_INVALID_LEVEL
206	ERROR_FILENAME_EXCED_RANGE

Remarks

The structure that DosEnumAttribute returns is used to calculate the size of the buffer needed to hold the full extended attribute (FEA2) information for a DosQueryPathInfo or DosQueryFileInfo call that actually gets the FEA2. The buffer size is calculated as follows:

Four bytes (for oNextEntryOffset) +
One byte (for fEA) +
One byte (for cbName) +
Two bytes (for cbValue) +
Value of cbName (for the name of the EA) +
One byte (for terminating NULL in cbName) +
Value of cbValue (for the value of the EA)

Each entry must start on a doubleword boundary.

A process can continue through a file's EA list by reissuing DosEnumAttribute with *EntryNum* set to the value specified in the previous call, plus the value returned in *EnumCnt*.

DosEnumAttribute does not control the specific ordering of EAs; it merely identifies them. Extended attributes can have multiple readers and writers, just as the files they are associated with can. If a file is open in a sharing mode that allows other processes to modify the file's EA list, repetitively calling DosEnumAttribute to back up to an EA's position may return inconsistent results. For example with DosSetFileInfo or DosSetPathInfo, another process can edit the EA list between calls by your process to DosEnumAttribute. Therefore, the EA returned when *EntryNum* is 11 for the first call might not be the same EA returned when *EntryNum* is 11 for the next call.

To prevent EAs from being modified between calls to DosEnumAttribute for a specified file handle or file name, the calling function must open the file in deny-write sharing mode before it calls DosEnumAttribute. If a subdirectory name is specified, modification by other processes is not a concern, because no sharing is possible.

When a value of 1 is specified for *RefType*, the EAs returned are current only when the call was made, and may have been changed by another thread or process since then.

Related Functions

- DosCreateDir
- DosOpen
- DosQueryFileInfo
- DosQueryPathInfo
- DosSetFileInfo
- DosSetPathInfo

DosEnumAttribute –
Identify Names and Lengths of Extended Attributes

Example Code

This example identifies the names and lengths of extended attributes that are associated with a specified file. Assume that the file has been opened already, and that the handle of the file has been loaded into *FileRef*. Assume that the file has at least 6 different extended attributes associated with it. In the example, extended attributes 3 through 6 will be read into the caller's buffer.

```
#define INCL_DOSFILEMGR   /* File Manager values */
#include <os2.h>
#include <stdio.h>

ULONG    RefType;       /* Type of reference */
HFILE    FileRef;       /* Handle (in this example) or Name */
ULONG    EntryNum;      /* Starting entry in EA list */
UCHAR    EnumBuf[200];  /* Data buffer */
ULONG    EnumBufSize;   /* Data buffer size */
ULONG    EnumCnt;       /* Count of entries to return */
ULONG    InfoLevel;     /* Level of information requested  */
APIRET   rc;            /* Return code  */

   RefType = ENUMEA_REFTYPE_FHANDLE;
                       /* Indicate that the FileRef variable   */
                       /*   contains a file handle rather than */
                       /*   an ASCII name                      */

   EntryNum = 3;       /* Start with the current extended */
                       /*   attribute number 3            */

   EnumBufSize = 200;  /* Size of data buffer that will    */
                       /*   receive the extended attribute */
                       /*   current valuess (in bytes)     */

   EnumCnt = 4;        /* Ask to see the current values of */
                       /*   four extended attributes       */
                       /*   (extended attributes 3 to 6).  */

   InfoLevel = ENUMEA_LEVEL_NO_VALUE;
                       /* Ask for Level 1 information      */
                       /*   characterizing the specified   */
                       /*   extended attributes            */

   rc = DosEnumAttribute(RefType, &FileRef, EntryNum, EnumBuf,
                         EnumBufSize, &EnumCnt, InfoLevel);

                       /* On successful return, the EnumBuf     */
                       /*   data buffer contains a chain of     */
                       /*   Level 1 data structures that        */
                       /*   each describe a single extended     */
                       /*   attribute.  Each Level 1 data       */
                       /*   structure can have a different      */
                       /*   size because both the name and the  */
                       /*   value of an extended attribute are  */
                       /*   of variable length.  Also, the      */
                       /*   EnumCnt variable will have been     */
                       /*   updated to contain the total number */
                       /*   extended attribute data structures  */
                       /*   that were placed in the data buffer.*/
   if (rc != 0)
     {
       printf("DosEnumAttribute error: return code = %ld", rc);
       return;
     }
```

DosErrClass – Provide More Information about Return Values

#define INCL_DOSMISC

APIRET DosErrClass (ULONG ulCode, PULONG pClass, PULONG pAction, PULONG pLocus)

DosErrClass provides more information about return values that have been received from other control-program functions.

Parameters

ulCode (ULONG) – input

A non-zero return value returned by a control-program function. (A non-zero return value indicates that an error has occurred.)

pClass (PULONG) – output

A pointer to a doubleword whose value indicates the classification for the error. The following values are possible:

Value	Name	Description
1	ERRCLASS_OUTRES	Out of resources
2	ERRCLASS_TEMPSIT	Temporary situation
3	ERRCLASS_AUTH	Authorization failed
4	ERRCLASS_INTRN	Internal error
5	ERRCLASS_HRDFAIL	Device hardware failure
6	ERRCLASS_SYSFAIL	System failure
7	ERRCLASS_APPERR	Probable application error
8	ERRCLASS_NOTFND	Item not located
9	ERRCLASS_BADFMT	Bad format for function or data
10	ERRCLASS_LOCKED	Resource or data locked
11	ERRCLASS_MEDIA	Incorrect media, cyclic redundancy check (CRC) error
12	ERRCLASS_ALREADY	Action already taken or done, or resource already exists
13	ERRCLASS_UNK	Unclassified
14	ERRCLASS_CANT	Cannot perform requested action
15	ERRCLASS_TIME	Timeout

pAction (PULONG) – output

A pointer to a doubleword whose value indicates the recommended corrective action for the error. The following values are possible:

Value	Name	Description
1	ERRACT_RETRY	Retry immediately
2	ERRACT_DLYRET	Delay and retry
3	ERRACT_USER	Bad user input - get new values
4	ERRACT_ABORT	Terminate in an orderly manner
5	ERRACT_PANIC	Terminate immediately
6	ERRACT_IGNORE	Ignore error
7	ERRACT_INTRET	Retry after user intervention

pLocus (PULONG) – output

A pointer to a doubleword whose value indicates where the error originated. The following values are possible:

DosErrClass –
Provide More Information about Return Values

Value	Name	Description
1	ERRLOC_UNK	Unknown
2	ERRLOC_DISK	Random-access device such as a disk
3	ERRLOC_NET	Network
4	ERRLOC_SERDEV	Serial device
5	ERRLOC_MEM	Memory

Returns

Return Code.

DosErrClass returns the following values:

0 NO_ERROR

Remarks

DosErrClass receives a non-zero return value from another control-program function as input. It then classifies the return value, tells where in the system the error occurred, and recommends a corrective action.

When called by a family-mode application, DosErrClass can return a valid error classification only for errors that have actually occurred. Also, the classifications of a given return value might not be the same for family-mode and OS/2-mode applications.

Related Functions

- DosError

Example Code

In the following example, an attempt is made to delete a nonexistent file. The return value is then passed to DosErrClass so that more information about the error can be obtained, including any corrective actions that may be taken.

```
#define INCL_DOSQUEUES
#include <os2.h>
#include <stdio.h>

#define FILE_DELETE "adlkjf.dkf"

ULONG  Error;
ULONG  Class;
ULONG  Action;
ULONG  Locus;
APIRET rc;          /* Return code */

   Error = DosDelete(FILE_DELETE);     /* File name path */

   rc = DosErrClass(Error,             /* Return value to be analyzed */
                    &Class,            /* Error classification */
                    &Action,           /* Recommended action */
                    &Locus);           /* Error locus */

   if (rc != 0)
     {
       printf("DosErrClass error: return code = %ld",rc);
       return;
     }
```

DosError – Disables or Enables Error Notification to End User

#define INCL_DOSMISC

APIRET DosError (ULONG ulFlags)

DosError disables or enables error notification to end users.

Parameters

ulFlags (ULONG) – input

A doubleword bit field, defined as shown below. The unused high-order bits are reserved, and must be zero. The following values can be specified for this parameter. You can combine them with the "logical or" (|) operator:

FERR_DISABLEHARDERR (0x00000000)
Disable hard error pop-ups.

FERR_ENABLEHARDERR (0x00000001)
Enable hard error pop-ups.

FERR_ENABLEEXCEPTION (0x00000000)
Enable program exception and untrapped numeric-processor exception pop-ups.

FERR_DISABLEEXCEPTION (0x00000002)
Disable program exception and untrapped numeric-processor exception pop-ups.

Returns

Return Code.

DosError returns the following values:

0	NO_ERROR
87	ERROR_INVALID_PARAMETER

Remarks

DosError disables or enables end-user notification of hard errors, program exceptions, or untrapped, numeric-processor exceptions.

If DosError is not issued, user notification for hard errors and exceptions is enabled.

Related Functions

- DosErrClass

DosError –
Disables or Enables Error Notification to End User

Example Code

In the following example, pop-up windows for hard errors and exceptions are disabled, then enabled again.

```
#define INCL_DOSMISC   /* Error and exception values */
#include <os2.h>
#include <stdio.h>

#define ENABLE_EXCEPTION 0
#define DISABLE_EXCEPTION 2
#define ENABLE_HARDERROR 1
#define DISABLE_HARDERROR 0
#define DISABLE_ERRORPOPUPS DISABLE_EXCEPTION | DISABLE_HARDERROR
#define ENABLE_ERRORPOPUPS ENABLE_EXCEPTION | ENABLE_HARDERROR

APIRET rc;        /* Return code */

   rc = DosError(DISABLE_ERRORPOPUPS); /* Action flag for disable */

   if (rc != 0)
     {
       printf("DosError error: return code = %ld",rc);
       return;
     }

   rc = DosError(ENABLE_ERRORPOPUPS);  /* Action flag for enable */

   if (rc != 0)
     {
       printf("DosError error: return code = %ld",rc);
       return;
     }
```

DosExecPgm – Execute Another Program as a Child Process

#define INCL_DOSPROCESS

```
APIRET DosExecPgm (PCHAR ppObjNameBuf, LONG lObjNameBufL, ULONG ulExecFlags,
                   PSZ pszArgPointer, PSZ pszEnvPointer,
                   PRESULTCODES ppReturnCodes, PSZ pszPgmPointer)
```

DosExecPgm allows a program to request that another program execute as a child process.

Parameters

ppObjNameBuf (PCHAR) – output

Address of the buffer in which the name of the object that contributed to the failure of DosExecPgm is returned.

lObjNameBufL (LONG) – input

Length, in bytes, of the buffer described by *ObjNameBuf*.

ulExecFlags (ULONG) – input

Indicates how the program runs in relation to the requester, and whether execution is under conditions for debugging. The values of this field are as follows:

Value	Definition
0	(EXEC_SYNC): Execution is synchronous to the parent process. The termination code and result code are stored in the two-doubleword structure pointed to by *ReturnCodes*.
1	(EXEC_ASYNC): Execution is asynchronous to the parent process. When the child process ends, its result code is discarded. The process ID is stored in the first doubleword of the two-doubleword structure pointed to by *ReturnCodes*
2	(EXEC_ASYNCRESULT): Execution is asynchronous to the parent process. When the child process ends, its result code is saved for examination by a DosWaitChild request. The process ID is stored in the first doubleword of the two-doubleword structure pointed to by ReturnCodes.
3	(EXEC_TRACE): Execution is the same as if a value of 2 were specified for *ExecFlags*. Debugging conditions are present for the child process.
4	(EXEC_BACKGROUND): Execution is asynchronous to and detached from the parent-process session. When the detached process starts, it is not affected by the ending of the parent process. The detached process is treated as an orphan of the parent process. A program executed with this option runs in the background, and should not require any input from the keyboard or output to the screen other than VioPopups. It should not issue any console I/O calls (VIO, KBD, or MOU functions).
5	(EXEC_LOAD): The program is loaded into storage and made ready to execute, but is not executed until the session manager dispatches the threads belonging to the process.
6	(EXEC_ASYNCRESULTDB): Execution is the same as if a value of 2 were specified for *ExecFlags*, with the addition of debugging conditions being present for the child process and any of its descendants. In this way, it is possible to debug even detached and synchronous processes.

Some memory is consumed for uncollected result codes. Issue DosWaitChild to release this memory. If result codes are not collected, then a value of 0 or 1 should be used for *ExecFlags*.

DosExecPgm –
Execute Another Program as a Child Process

pszArgPointer (PSZ) – input

Address of the ASCIIZ argument strings passed to the program. These strings represent command parameters, which are copied to the environment segment of the new process.

The convention used by CMD.EXE is that the first of these strings is the program name (as entered from the command prompt or found in a batch file), and the second string consists of the parameters for the program. The second ASCIIZ string is followed by an additional byte of zeros. A value of zero for the address of *ArgPointer* means that no arguments are to be passed to the program.

pszEnvPointer (PSZ) – input

Address of the ASCIIZ environment strings passed to the program. These strings represent environment variables and their current values. An environment string has the following form:

```
variable=value
```

The last ASCIIZ environment string must be followed by an additional byte of zeros.

A value of 0 for the address of *EnvPointer* results in the new process' inheriting the environment of its parent process.

When the new process is given control, it receives:

- A pointer to its environment segment
- The fully qualified file specification of the executable file
- A copy of the argument strings.

A coded example of this follows:

```
eo:   ASCIIZ string 1  ; environment string 1
      ASCIIZ string 2  ; environment string 2
.
.
.
      ASCIIZ string n  ; environment string n
      Byte of 0
.
.
.
po:   ASCIIZ           ; string of file name
                       ; of program to run.
.
.
.
ao:   ASCIIZ           ; argument string 1
                       ; (name of program being started
                       ; for the case of CMD.EXE)
      ASCIIZ           ; argument string 2
                       ; (program parameters following
                       ; program name for the case of
                       ; CMD.EXE)
      Byte of 0
```

The beginning of the environment segment is "eo", and "ao" is the offset of the first argument string in that segment. The offset to the command line, "ao", is passed to the program on the stack at SS:[ESP+16].

The environment strings typically have the form: parameter = value

A value of zero for *EnvPointer* causes the newly created process to inherit the parent's environment unchanged.

ppReturnCodes (PRESULTCODES) – output

Address of the two-doubleword structure where the process ID, or the termination code and the result code indicating the reason for ending the child process are returned. This structure also

is used by a DosWaitChild request, which waits for an asynchronous child process to end. This structure contains two doublewords, as follows:

termcodepid (ULONG)
For an asynchronous request, the process identifier of the child process. For a synchronous request, the termination code furnished by the system describes why the child process ended. The values of the termination codes are as follows:

Value	Definition
0	(TC_EXIT): Normal exit
1	(TC_HARDERROR): Hard-error halt
2	(TC_TRAP): Trap operation for a 16-bit child process
3	(TC_KILLPROCESS): Unintercepted DosKillProcess
4	(TC_EXCEPTION): Exception operation for a 32-bit child process

resultcode (ULONG)
Result code specified by the terminating synchronous process on its last DosExit function.

pszPgmPointer (PSZ) – input

Address of the name of the file that contains the program to be executed. When the environment is passed to the target program, this name is copied into "po" in the environment description shown above.

If the string appears to be a fully qualified file specification (that is, it contains a " : " in the second position, or it contains a " \ ", or both), then the file name must include the extension, and the program is loaded from the indicated drive:directory.

If the string is not a fully qualified path, the current directory is searched. If the file name is not found in the current directory, each drive:directory specification in the PATH defined in the current-process environment is searched for this file. Note that any extension (.XXX) is acceptable for the executable file being loaded.

Returns

Return Code.

DosExecPgm returns the following values:

0	NO_ERROR
1	ERROR_INVALID_FUNCTION
2	ERROR_FILE_NOT_FOUND
3	ERROR_PATH_NOT_FOUND
4	ERROR_TOO_MANY_OPEN_FILES
5	ERROR_ACCESS_DENIED
8	ERROR_NOT_ENOUGH_MEMORY
10	ERROR_BAD_ENVIRONMENT
11	ERROR_BAD_FORMAT
13	ERROR_INVALID_DATA
26	ERROR_NOT_DOS_DISK
32	ERROR_SHARING_VIOLATION
33	ERROR_LOCK_VIOLATION
36	ERROR_SHARING_BUFFER_EXCEEDED
89	ERROR_NO_PROC_SLOTS
95	ERROR_INTERRUPT
108	ERROR_DRIVE_LOCKED
127	ERROR_PROC_NOT_FOUND
182	ERROR_INVALID_ORDINAL
190	ERROR_INVALID_MODULETYPE
191	ERROR_INVALID_EXE_SIGNATURE
192	ERROR_EXE_MARKED_INVALID
195	ERROR_INVALID_MINALLOCSIZE

DosExecPgm –
Execute Another Program as a Child Process

196 ERROR_DYNLINK_FROM_INVALID_RING

Remarks

DosExecPgm allows a program to request that another program execute as a child process.

The target program is located and loaded into storage (if necessary), a process is created for it and placed into execution. The execution of a child process can be synchronous or asynchronous to the execution of its parent process. If synchronous execution is indicated, the requesting thread waits for completion of the child process. Other threads in the requesting process may continue to run.

If asynchronous execution is indicated, DosExecPgm places the process ID of the started child process into the first doubleword of the *ReturnCodes* structure. If a value of 2 is specified for *ExecFlags*, the parent process can issue DosWaitChild (after DosExecPgm) to examine the result code returned when the child process ends. If the value of *ExecFlags* is 1, the result code of the asynchronous child process is not returned to the parent process.

If synchronous execution is indicated, DosExecPgm places the termination code and result code into the *ReturnCodes* structure.

The new process is created with an address space separate and distinct from its parent; that is, a new linear address space is built for the process.

The new process inherits all file handles and pipes of its parent, although not necessarily with the same access rights:

- Files are inherited except for those opened with no inheritance indicated.
- Pipes are inherited.

A child process inherits file handles obtained by its parent process with DosOpen calls that indicated inheritance. The child process also inherits handles to pipes created by the parent process with DosCreatePipe. This means that the parent process has control over the meanings of standard input, output, and error. For example, the parent could write a series of records to a file, open the file as standard input, open a listing file as standard output, and then execute a sort program that takes its input from standard input and writes to standard output.

Because a child process can inherit handles, and a parent process controls the meanings of handles for standard I/O, the parent can duplicate inherited handles as handles for standard I/O. This permits the parent process and the child process to coordinate I/O to a pipe or file. For example, a parent process can create two pipes with DosCreatePipe requests. It can issue DosDupHandle to redefine the read handle of one pipe as standard input (hex 0000), and the write handle of the other pipe as standard output (hex 0001). The child process uses the standard I/O handles, and the parent process uses the remaining read and write pipe handles. Thus, the child process reads what the parent process writes to one pipe, and the parent process reads what the child process writes to the other pipe.

When an inherited file handle is duplicated, the position of the file pointer is always the same for both handles, regardless of which handle repositions the file pointer.

An asynchronous process that was started because the value of *ExecFlags* was 3 or 6 is provided a trace flag facility. This facility and the trace buffers provided by DosDebug enable a debugger to perform breakpoint debugging. DosStartSession provides additional debugging capabilities that allow a debugger to trace all processes associated with an application running in a child session, regardless of whether the process is started with DosExecPgm or DosStartSession.

A detached process is treated as an orphan of the parent process and runs in the background. Thus, it cannot make any VIO, KBD, or MOU calls, except from within a video pop-up requested by VioPopUp. To test whether a program is running detached, use the following method. Issue a video call, (for example, VioGetAnsi). If the call is not issued within a video pop-up and the process is detached, the video call returns error code ERROR_VIO_DETACHED.

DosExecPgm – Execute Another Program as a Child Process

You may use DosExecPgm to start a process that is of the same type as the starting process. Process types include Presentation Manager, text-windowed, and full-screen. You may not use DosExecPgm to start a process that is of a different type than the starting process.

You must use DosStartSession to start a new process from a process that is of a different type. For example, use DosStartSession to start a Presentation Manager process from a non-Presentation Manager process.

The following are the register conventions for 32-bit programs:

Register	Definition
EIP	Starting program entry address
ESP	Top of stack address
CS	Code selector for the base of the linear address space
DS, ES, SS	Data selector for the base of the linear address space
FS	Data selector for the thread information block
GS	0
EAX, EBX	0
ECX, EDX	0
ESI, EDI	0
EBP	0
[ESP+0]	Return address to the routine that calls DosExit
[ESP+4]	Module handle for the program module
[ESP+8]	0
[ESP+12]	Address of the environment data object
[ESP+16]	Offset to the command line in the environment data object.

Related Functions

- DosCreatePipe
- DosCreateThread
- DosExit
- DosKillProcess
- DosKillThread
- DosOpen
- DosWaitChild

Example Code

This example starts the program simple.exe and then waits for it to finish. It then prints the termination code and return code.

```
#define INCL_DOSPROCESS       /* Process and thread values */

#include <os2.h>
#include <stdio.h>

#define START_PROGRAM "simple.exe"

#ifndef _RESULTCODES
 typedef struct _RESULTCODES {  /* Result codes */

   ULONG codeTerminate;        /* Termination code or process ID */
```

DosExecPgm –
Execute Another Program as a Child Process

```
    ULONG codeResult;           /* Exit code */

 } RESULTCODES;
#endif

CHAR        LoadError[100];
PSZ         Args;
PSZ         Envs;
RESULTCODES ReturnCodes;
APIRET      rc;                         /* Return code */

  rc = DosExecPgm(LoadError,            /* Object name buffer */
                  sizeof(LoadError),    /* Length of object name buffer */
                  EXEC_SYNC,            /* Asynchronous/Trace flags */
                  Args,                 /* Argument string */
                  Envs,                 /* Environment string */
                  &ReturnCodes,         /* Termination codes */
                  START_PROGRAM);       /* Program file name */

  if (rc == 0)
    {
      printf("Termination Code %d  Return Code %d \n",
             ReturnCodes.codeTerminate,
             ReturnCodes.codeResult);
    }

----------------simple.exe-----------------

#define INCL_DOSPROCESS             /* Process and thread values */

#include <os2.h>

#define RETURN_CODE 0

main()
   {
     printf("Hello!\n");
     DosExit(EXIT_PROCESS,          /* End thread/process */
             RETURN_CODE);          /* Result code */
   }
```

DosExit – Issued When a Thread Finishes Executing

#define INCL_DOSPROCESS

VOID DosExit (ULONG ulActionCode, ULONG ulResultCode)

DosExit is issued when a thread finishes executing. The current thread or process ends.

Parameters

ulActionCode (ULONG) – input

Ends the process and all of its threads. The values of this field are as follows:

Value	Definition
0	(EXIT_THREAD): The current thread ends.
1	(EXIT_PROCESS): All threads in the process end.

ulResultCode (ULONG) – input

Program's completion code. It is passed to any thread that issues DosWaitChild for this process.

Returns

The return value is VOID.

Remarks

DosExit is issued when a thread completes executing. The current thread or process ends.

DosExit allows a thread to terminate itself or be terminated by another thread in its process. If *ActionCode* is 0 and the specified thread is the last thread executing in the process, the process ends. If *ActionCode* is 1, the process ends.

The system can start threads on behalf of an application. Thus, if the intent of DosExit is to terminate the process, a value of 1 should be specified for *ActionCode* to end all the threads belonging to the process.

Do not end thread 1 without ending the process. Thread 1 is the initial thread of execution, not a thread started by a DosCreateThread request. When thread 1 ends, any monitors or signal processing routines set for this process also end. To avoid unpredictable results, DosExit should be issued with a value of 1 for *ActionCode* to ensure that the process ends.

When a process is ending, all but one thread is ended, and that thread executes routines whose addresses have been specified with DosExitList. After resources have been released by the exit list routines, this thread and all other resources owned by the process are released.

Related Functions

- DosExecPgm
- DosExitList
- DosKillThread
- DosWaitChild

DosExit –
Issued When a Thread Finishes Executing

Example Code

In this example, the main routine starts up another program, simple.exe, and then expects a return code of 3 to be returned. Simple.exe sets the return code with DosExit.

```
#define INCL_DOSPROCESS             /* Process and thread values */
#include <os2.h>
#include <stdio.h>

#define START_PROGRAM "simple.exe"
#define RETURN_OK 3

CHAR        LoadError[100];
PSZ         Args;
PSZ         Envs;
RESULTCODES ReturnCodes;
APIRET      rc;             /* Return code */

  rc = DosExecPgm(LoadError,                    /* Object name buffer */
                  sizeof(LoadError),            /* Length of object name
                                                    buffer */
                  EXEC_SYNC,                    /* Asynchronous/Trace
                                                       flags */
                  Args,                         /* Argument string */
                  Envs,                         /* Environment string */
                  &ReturnCodes,                 /* Termination codes */
                  START_PROGRAM);               /* Program file name */

      if (ReturnCodes.codeResult == RETURN_OK)  /* Check result code */
         printf("things are ok...");
      else
         printf("something is wrong...");

----------------simple.exe-----------------

#define INCL_DOSPROCESS          /* Process and thread values */
#include <os2.h>

#define RETURN_CODE 3

main()
   {
     printf("Hello!\n");
     DosExit(EXIT_THREAD,        /* End thread/process */
             RETURN_CODE);       /* Result code */
   }
```

DosExitCritSec – Restore Normal Thread Dispatching for Current Process

#define INCL_DOSPROCESS

APIRET DosExitCritSec ()

DosExitCritSec restores normal thread dispatching for the current process.

Parameters

Returns

Return Code.

DosExitCritSec returns the following values:

0	NO_ERROR
309	ERROR_INVALID_THREADID
485	ERROR_CRITSEC_UNDERFLOW

Remarks

DosExitCritSec is used following DosEnterCritSec to restore normal thread switching to the threads of a process.

A count is maintained of the number of times DosEnterCritSec is issued without a corresponding DosExitCritSec. The count is incremented by DosEnterCritSec, and decremented by DosExitCritSec. Normal thread dispatching is not restored until the count is zero.

The outstanding count is maintained in a word. If an underflow occurs (the count is decremented below zero), the count is set to zero, no operation is performed, and the request returns with ERROR_CRITSEC_UNDERFLOW.

ERROR_INVALID_THREADID is returned when an invalid attempt is made to exit a critical section of code in a signal handler or exception handler.

ERROR_INVALID_THREADID is also returned when a dynamic link library (DLL) routine incorrectly issues DosExitCritSec.

Related Functions

- DosCreateThread
- DosEnterCritSec

Example Code

This example enters a section that will not be pre-empted, performs a simple task, and then exits quickly.

```
#define INCL_DOSPROCESS     /* Process and thread values */
#include <os2.h>
#include <stdio.h>

BOOL flag;

   DosEnterCritSec();       /* Enter critical code
                               code section */
   flag = TRUE;             /* Perform some work */
   DosExitCritSec();        /* Exit critical code section */
```

DosExitList –
Maintain a List of Routines that Execute when the Current Process Ends

---

#define INCL_DOSPROCESS

APIRET DosExitList (ULONG ulFunctionOrder, PFNEXITLIST ppRtnAddress)

DosExitList maintains a list of routines that execute when the current process ends.

Parameters

ulFunctionOrder (ULONG) – input

Contains two one-byte fields in the low-order word. The high-order word is zero.

The low-order byte of the low-order word indicates which function DosExitList performs. This function can update the list of routines, or transfer to the next address on the termination list at the completion of a routine. The values of the byte and their meanings are as follows:

Value	Definition
1	(EXLST_ADD): Add an address to the termination list.
2	(EXLST_REMOVE): Remove an address from the termination list.
3	(EXLST_EXIT): When termination processing completes, transfer to the next address on the termination list.

The high-order byte of the low-order word indicates the invocation order. This value is valid only when the low-order byte is 1 (add an address). For the other low-order byte values, the high-order byte of the low-order word must be set to zero.

The invocation order indicates where the routine address is to be placed in an ordered list. This list determines the order in which the exit list routines are invoked. Routines with a value of 0 are invoked first, and routines with a value of 255 are invoked last. If more than one routine is added with the same invocation order value, the last routine to be added is invoked first. The following values are used by OS/2 components:

Value	Definition
X'80' – X'88'	OS/2 Extended Edition Database Manager
X'90' – X'98'	OS/2 Extended Edition Communications Manager
X'A0' – X'A8'	OS/2 Presentation Manager
X'B0'	OS/2 Keyboard (KBD) component
X'C0'	OS/2 Video (VIO) component
X'D0'	OS/2 Interprocess Communication (IPC) Queues component

ppRtnAddress (PFNEXITLIST) – input

The address of a routine to be executed.

Returns

Return Code.

DosExitList returns the following values:

0	NO_ERROR
1	ERROR_INVALID_FUNCTION
8	ERROR_NOT_ENOUGH_MEMORY
13	ERROR_INVALID_DATA

DosExitList – Maintain a List of Routines that Execute when the Current Process Ends

Remarks

DosExitList is issued to define a routine that is to be given control when a process completes its execution. Multiple routines may be defined to receive control when a process is ending. For each process, the operating system maintains a list of addresses of defined exit list routines.

When the process is ending, the operating system transfers control to each address in this list. If there are multiple addresses in the list, they will each get control in numerical order by function invocation order, that is, low (0) will be first, and high (hex FF) will be last. In case of duplicate entries for the same value, the routines will be executed in LIFO (last in, first out) order.

Library modules can issue DosExitList to free resources or to clear flags and semaphores in case the client process ends without notifying them.

Before transferring control to the routines in the termination list, the operating system resets the stack to its initial value. The routine must be in the address space of the ending process. The termination routine should perform its processing and then issue DosExitList with a value of 3 (EXLST_EXIT) for *FunctionOrder*. The termination routine should be as short as possible.

Most system functions are allowed in an exit list routine. However, DosCreateThread and DosExecPgm are not.

An exit list routine must not call functions that have a better function order priority (that is, a lower value for *FunctionOrder*) than itself. For example, an exit list routine with a function order value of hex 9A can use Presentation Services functions but not Communications Manager functions.

When the exit list routine receives control, the first parameter on the stack (located at ESP+4) contains a termination code that describes why the process ended. The values of the termination codes are as follows:

Value	Definition
0	(TC_EXIT) Normal exit
1	(TC_HARDERROR) Hard-error halt
2	(TC_TRAP) Trap operation for a 16-bit child process
3	(TC_KILLPROCESS) Unintercepted DosKillProcess
4	(TC_EXCEPTION) Exception operation for a 32-bit child process

When the exit list routine receives control, all system semaphores owned by the process have their ownership transferred to the thread that performs exit list processing. This allows the thread to request serialization semaphores without danger of blocking in case the semaphore was held by another thread in the process that has already ended.

Note: All exit list routines *must* be declared as VOID APIENTRY. This ensures the integrity of the stack.

Related Functions

- DosCreateThread
- DosExecPgm
- DosExit
- DosKillProcess
- DosKillThread

DosExitList –
Maintain a List of Routines that Execute when the Current Process Ends

Example Code

In this example, TestRoutine is added to the exit-list sequence. Routines in the exit-list sequence must use DosExitList instead of DosExit to end.

```
#define INCL_DOSPROCESS             /* Process and thread values */
#define INCL_VIO
#include <os2.h>
#include <stdio.h>
#define ROUTINE_ORDER 0x0000EE00
#define VIO_HANDLE 0

APIRET  rc;                             /* Return code */

/* All exit list routines must be declared as VOID APIENTRY. */
/* This ensures the integrity of the stack. */

VOID APIENTRY TestRoutine2()
   {
   APIRET  r;                                   /* Return code */

   VioWrtTTY("This runs last...\n",             /* String to be written */
             18,                                /* Length of string     */
             VIO_HANDLE);                       /* Video handle         */
   r = DosExitList(EXLST_EXIT,                      /* Function request
                                                       code/order     */
                   (PFNEXITLIST) TestRoutine2);  /* Address of routine */
   }

main()
   {
   rc = DosExitList(EXLST_ADD | ROUTINE_ORDER,   /* Function request
                                                          code/order */
                    (PFNEXITLIST) TestRoutine2); /* Address of routine */
   }
```

DosExitMustComplete – Exit Must Complete

#define INCL_DOSEXCEPTIONS

APIRET DosExitMustComplete (PULONG ppulNesting)

DosExitMustComplete provides exit from a section of code in which asynchronous exceptions are held.

Parameters

ppulNesting (PULONG) – output

A pointer to a value that is equal to the number of DosEnterMustComplete requests minus the number of DosExitMustComplete requests for the current thread.

Returns

Return Code.

DosExitMustComplete returns the following values:

0	NO_ERROR
300	ERROR_ALREADY_RESET

Remarks

DosExitMustComplete notifies the system that the calling thread is leaving a section of code in which any asynchronous exceptions (signals and asynchronous process terminations) that may have occurred were held, rather than being immediately delivered to the thread.

For a detailed list of the system exceptions, see Appendix C, "System Exceptions" on page C-1.

Related Functions

- DosAcknowledgeSignalException
- DosEnterMustComplete
- DosRaiseException
- DosSendSignalException
- DosSetExceptionHandler
- DosSetSignalExceptionFocus
- DosUnsetExceptionHandler
- DosUnwindException

DosExitMustComplete – Exit Must Complete

Example Code

This example shows how a thread can notify the system that the thread is leaving a section of code in which asynchronous exceptions (signals and asynchronous process terminations) are to be held, rather than being delivered to the thread immediately.

Assume that the unsigned long variable *NestingLevel* is a global program variable that is used to maintain the number of nested calls to DosEnterMustComplete that are currently in effect for the section of code. Assume that *NestingLevel* was originally initialized to zero, and that its value is subsequently incremented by calls to DosEnterMustComplete, and decremented by calls to DosExitMustComplete.

```
#define INCL_DOSEXCEPTIONS   /* Exception values */
#include <os2.h>
#include <stdio.h>

extern  ULONG  NestingLevel;  /* Number of signal to be acknowledged */
APIRET  rc;                   /* Return code */

   rc = DosExitMustComplete(&NestingLevel);

   if (rc != 0)
     {
       printf("DosExitMustComplete error: return code = %ld",
                      rc);
       return;
     }
```

DosFindClose – Close a Handle to a Find Request

#define INCL_DOSFILEMGR

APIRET DosFindClose (HDIR hdirDirHandle)

DosFindClose closes the handle to a find request; that is, ends a search.

Parameters

hdirDirHandle (HDIR) – input

The handle previously associated with a DosFindFirst function by the system, or used with a DosFindNext directory search function.

Returns

Return Code.

DosFindClose returns the following values:

0	NO_ERROR
6	ERROR_INVALID_HANDLE

Remarks

When DosFindClose is issued, any subsequent issuance of DosFindNext for the closed handle (*DirHandle*) fails unless an intervening DosFindFirst specifying the handle is issued.

Related Functions

- DosFindFirst
- DosFindNext

DosFindClose –
Close a Handle to a Find Request

Example Code

This example searches for a file, then ends the search.

```
#define INCL_DOSFILEMGR       /* File Manager values */
#include <os2.h>
#include <stdio.h>

#define SEARCH_PATTERN "*.*"
#define FILE_ATTRIBUTE 0

HDIR          FindHandle;
FILEFINDBUF3  FindBuffer;
ULONG         FindCount;
APIRET        rc;         /* Return code */

   FindHandle = 0x0001;
   FindCount = 1;

   rc = DosFindFirst(SEARCH_PATTERN,        /* File pattern */
                     &FindHandle,           /* Directory search handle */
                     FILE_ATTRIBUTE,        /* Search attribute */
                     (PVOID) &FindBuffer,   /* Result buffer */
                     sizeof(FindBuffer),    /* Result buffer length */
                     &FindCount,            /* # of entries to find */
                     FIL_STANDARD);          /* Level 1 file information */

   if (rc != 0)
     {
       printf("DosFindFirst error: return code = %ld",rc);
       return;
     }

   rc = DosFindClose(FindHandle);           /* Directory search handle */

   if (rc != 0)
     {
       printf("DosFindClose error: return code = %ld",rc);
       return;
     }
```

DosFindFirst – Find the First File Object

#define INCL_DOSFILEMGR

```
APIRET DosFindFirst (PSZ pszFileName, PHDIR ppDirHandle, ULONG ulAttribute,
                     PVOID pResultBuf, ULONG ulResultBufLen, PULONG pSearchCount,
                     ULONG ulFileInfoLevel)
```

DosFindFirst finds the first file object or group of file objects whose names match the specification. The specification can include extended attributes associated with a file or directory.

Parameters

pszFileName (PSZ) – input

Address of the ASCIIZ path name of the file or subdirectory to be found. The name component may contain global file-name characters.

ppDirHandle (PHDIR) – input/output

Address of the handle associated with this DosFindFirst request. The values that can be specified for the handle are:

Value	Description
hex 00000001	(HDIR_SYSTEM) The system assigns the handle for standard output, which is always available to a process.
hex FFFFFFFF	(HDIR_CREATE) The system allocates and returns a handle. Upon return to the caller, *DirHandle* contains the handle allocated by the system.

The DosFindFirst handle is used with subsequent DosFindNext requests. Reuse of this handle in another DosFindFirst request closes the association with the previous DosFindFirst request, and opens a new association with the current DosFindFirst request.

ulAttribute (ULONG) – input

Attribute value that determines the file objects to be searched for. The bit values are as follows:

Bit	Description
31 – 14	Reserved; must be zero.
13	(MUST_HAVE_ARCHIVED 0x00002000) Must have Archive bit; excludes files without the archive bit set if bit 13 is set to 1. Files may have the Archive bit set if bit 13 is set to 0.
12	(MUST_HAVE_DIRECTORY 0x00001000) Must have Subdirectory bit; excludes files that are not subdirectories if bit 12 is set to 1. Files may have the Subdirectory bit set if bit 12 is set to 0.
11	Reserved; must be zero.
10	(MUST_HAVE_SYSTEM 0x00000400) Must have System File bit; excludes non-system files if bit 10 is set to 1. Files may be system files if bit 10 is set to 0.
9	(MUST_HAVE_HIDDEN 0x00000200) Must have Hidden File bit; excludes non-hidden files if bit 9 is set to 1. Files may be non-hidden if bit 9 is set to 0.
8	(MUST_HAVE_READONLY 0x00000100) Must have Read-Only File bit; excludes writeable files if bit 8 is set to 1. Files may be read-only if bit 8 is set to 0.
7 – 6	Reserved; must be zero.
5	(FILE_ARCHIVED 0x00000020) May have Archive bit; includes files with the Archive bit set if bit 5 is set to 1. Excludes files with the Archive bit set if bit 5 is set to 0.

4 (FILE_DIRECTORY 0x00000010) May have Subdirectory bit; includes files that are subdirectories if bit 4 is set to 1. Excludes files that are subdirectories if bit 4 is set to 0.

3 Reserved; must be zero.

2 (FILE_SYSTEM 0x00000004) May have System File bit; includes system files if bit 2 is set to 1. Excludes system files if bit 2 is set to 0.

1 (FILE_HIDDEN 0x00000002) May have Hidden File bit; includes hidden files if bit 1 is set to 1. Excludes hidden files if bit 1 is set to 0.

0 (FILE_READONLY 0x00000001) May have Read-Only File bit; includes read-only files if bit 0 is set to 1. Excludes read-only files if bit 0 is set to 0.

These bits may be set individually or in combination. For example, an attribute value of hex 00000021 (bits 5 and 0 set to 1) indicates searching for read-only files that have been archived.

Bits 8 through 13 are "Must-have" flags. These allow you to obtain files that definitely have the given attributes. For example, if the Must have Subdirectory bit is set to 1, then all returned items are subdirectories.

If a Must-have bit is set to 1 and the corresponding May-have bit is set to zero, no items are returned for that attribute.

Attribute cannot specify the volume label. Volume labels are queried using DosQueryFSInfo.

pResultBuf (PVOID) – input/output

Address of the directory search structures for file object information levels 1 through 3. The structure required for *ResultBuf* is dependent on the value specified for *FileInfoLevel*. The information returned reflects the most recent call to DosClose or DosResetBuffer.

For Level 1 File Information:
On output, *ResultBuf* contains the *FILEFINDBUF3* data structure. This is used without EAs (extended attributes).

The *oNextEntryOffset* field indicates the number of bytes from the beginning of the current structure to the beginning of the next structure. When this field is zero, the last structure has been reached.

For Level 2 File Information:
On output, *ResultBuf* contains the *FILEFINDBUF4* data structure. This is used with EAs.

The *cbList* field contains the size, in bytes, of the file's entire extended attribute (EA) set on disk. You can use this field to calculate the maximum size of the buffer needed for level 3 file information. The size of the buffer required to hold the entire EA set is less than or equal to twice the size of the EA set on disk.

For Level 3 File Information:
On input, *ResultBuf* contains an EAOP2 data structure. *fpGEA2List* contains a pointer to a GEA2 list, which defines the attribute names whose values are to be returned. Entries in the *GEA2* list must be aligned on a doubleword boundary. Each *oNextEntryOffset* field must contain the number of bytes from the beginning of the current entry to the beginning of the next entry.

On output, *ResultBuf* contains a structure with a set of records, each aligned on a doubleword boundary. These records represent the directory entry and associated extended attributes (EAs) for the matched file object. *fpFEA2List* in the *EAOP2* data structure contains a pointer to the first FEA2 list.

ResultBuf has the following format:

- The EAOP2 data structure
- Level 1 file information FILEFINDBUF3
- Length of the entire EA set on disk (*cbList*)
- A FEA2LIST data structure

- Length of the name string of the file object (*cbName*)
- Name of the file object matched by the input pattern (*achName*)

The records following the *EAOP2* data structure are repeated for the remainder of the file objects found.

Even if there is not enough room to hold all of the requested information, as for return code ERROR_BUFFER_OVERFLOW, the *cbList* field of the *FEA2LIST* data structure is valid if there is at least enough space to hold it.

When buffer overflow occurs, *cbList* contains the size on disk of the entire EA set for the file, even if only a subset of its attributes was requested. The size of the buffer required to hold the EA set is less than or equal to twice the size of the EA set on disk. If no error occurs, *cbList* includes the pad bytes (for doubleword alignment) between *FEA2* structures in the list.

If a particular attribute is not attached to the object, *ResultBuf* has an *FEA2* structure containing the name of the attribute, and the length value is zero.

ulResultBufLen (ULONG) – input

The length, in bytes, of *ResultBuf*.

pSearchCount (PULONG) – input/output

On input, the address of the number of matching entries requested in *ResultBuf*. On output, the number of entries placed into *ResultBuf*.

ulFileInfoLevel (ULONG) – input

The level of file information required.

Value	Description
1	(FIL_STANDARD) Level 1 file information
2	(FIL_QUERYEASIZE) Level 2 file information
3	(FIL_QUERYEASFROMLIST) Level 3 file information

The structures described in *ResultBuf* indicate the information returned for each of these levels.

Regardless of the level specified, a DosFindFirst request (and an associated DosFindNext) request on a handle returned by DosFindFirst) always includes level 1 information as part of the information that is returned.

However, when level 1 information is specifically requested, and *Attribute* specifies hidden files, system files, or subdirectory files, an inclusive search is made. That is, all normal file entries plus all entries matching any specified attributes are returned. Normal files are files without any mode bits set. They may be read from or written to.

Returns

Return Code.

DosFindFirst returns the following values:

0	NO_ERROR
2	ERROR_FILE_NOT_FOUND
3	ERROR_PATH_NOT_FOUND
6	ERROR_INVALID_HANDLE
18	ERROR_NO_MORE_FILES
26	ERROR_NOT_DOS_DISK
87	ERROR_INVALID_PARAMETER
108	ERROR_DRIVE_LOCKED
111	ERROR_BUFFER_OVERFLOW
113	ERROR_NO_MORE_SEARCH_HANDLES
206	ERROR_FILENAME_EXCED_RANGE
208	ERROR_META_EXPANSION_TOO_LONG

DosFindFirst –
Find the First File Object

254	ERROR_INVALID_EA_NAME
255	ERROR_EA_LIST_INCONSISTENT
275	ERROR_EAS_DIDNT_FIT

Remarks

DosFindFirst returns directory entries (up to the number requested in *SearchCount*) and extended-attribute information for as many files or subdirectories whose names, attributes, and extended attributes match the specification, and whose information fits in *ResultBuf*. On output, *SearchCount* contains the actual number of directory entries returned.

The file name pointed to by *FileName* can contain global file-name characters.

DosFindNext uses the directory handle associated with DosFindFirst to continue the search started by the DosFindFirst request.

Any non-zero return code, except ERROR_EAS_DIDNT_FIT, indicates that no handle has been allocated. This includes such non-error indicators as ERROR_NO_MORE_FILES.

For ERROR_EAS_DIDNT_FIT, a search handle is returned, and a subsequent call to DosFindNext gets the next matching entry in the directory. You can use DosQueryPathInfo to retrieve the extended attributes (EAs) for the matching entry by using the same EA arguments used for the DosFindFirst call, and the name that was returned by DosFindFirst.

For ERROR_EAS_DIDNT_FIT, only information for the first matching entry is returned. This entry is the one whose extended attributes did not fit in the buffer. The information returned is in the format of that returned for information level 2. No further entries are returned in the buffer, even if they could fit in the remaining space.

Related Functions

- DosClose
- DosFindClose
- DosFindNext
- DosQueryFileInfo
- DosQueryPathInfo
- DosQuerySysInfo
- DosResetBuffer
- DosSearchPath
- DosSetFileInfo
- DosSetPathInfo

Example Code

This example gets the first file in the current directory.

```
#define INCL_DOSFILEMGR   /* File Manager values */
#include <os2.h>
#include <stdio.h>

#define NORMAL_FILES 0
#define SEARCH_PATTERN "*.*"
#define FILE_ATTRIBUTE NORMAL_FILES

HDIR          FindHandle;
FILEFINDBUF3  FindBuffer;
ULONG         FindCount;
APIRET        rc;           /* Return code */

   FindHandle = 0x0001;
   FindCount = 1;

  rc = DosFindFirst(SEARCH_PATTERN,        /* File pattern */
                    &FindHandle,           /* Directory search handle */
                    FILE_ATTRIBUTE,        /* Search attribute */
                    (PVOID) &FindBuffer,   /* Result buffer */
                    sizeof(FindBuffer),    /* Result buffer length */
                    &FindCount,            /* # of entries to find */
                    FIL_STANDARD);        /* Return level 1 file info */

  if (rc != 0)
    {
     printf("DosFindFirst error: return code = %ld", rc);
    }
```

DosFindNext –
Find the Next Set of File Objects

#define INCL_DOSFILEMGR

```
APIRET DosFindNext (HDIR hdirDirHandle, PVOID pResultBuf, ULONG ulResultBufLen,
                    PULONG pSearchCount)
```

DosFindNext finds the next set of file objects whose names match the specification in a previous call to DosFindFirst or DosFindNext.

Parameters

hdirDirHandle (HDIR) – input

The handle of the directory.

pResultBuf (PVOID) – input/output

The address of the directory search information structure. The information returned reflects the most recent call to DosClose or DosResetBuffer.

For the continuation of a Level 3 File Information search, this buffer should contain input in the same format as a Level 3 File Information search by DosFindFirst.

See the description of the *ResultBuf* parameter in DosFindFirst for information about the output data that the file system driver places into this buffer.

ulResultBufLen (ULONG) – input

The length, in bytes, of *ResultBuf*.

pSearchCount (PULONG) – input/output

On input, the address of the number of matching entries requested in *ResultBuf*. On output, the number of entries placed into *ResultBuf*.

Returns

Return Code.

DosFindNext returns the following values:

0	NO_ERROR
6	ERROR_INVALID_HANDLE
18	ERROR_NO_MORE_FILES
26	ERROR_NOT_DOS_DISK
87	ERROR_INVALID_PARAMETER
111	ERROR_BUFFER_OVERFLOW
275	ERROR_EAS_DIDNT_FIT

Remarks

If ERROR_BUFFER_OVERFLOW is returned, further calls to DosFindNext start the search from the same entry.

If ERROR_EAS_DIDNT_FIT is returned, the buffer is too small to hold the extended attributes (EAs) for the first matching entry being returned. A subsequent call to DosFindNext gets the next matching entry. This enables the search to continue if the extended attributes being returned are too large for the buffer. You can use DosQueryPathInfo to retrieve the extended attributes for the matching entry by using the same EA arguments used for the call to DosFindFirst, and the name that was returned by DosFindFirst.

In the case of ERROR_EAS_DIDNT_FIT, only information for the first matching entry is returned. This is the entry whose extended attributes did not fit in the buffer. The information returned is in the

format of Level 2 File Information (*FILEFINDBUF4*). No further entries are returned in the buffer, even if they could fit in the remaining space.

Related Functions

- DosClose
- DosFindClose
- DosFindFirst
- DosFindNext
- DosQueryFileInfo
- DosQueryPathInfo
- DosQuerySysInfo
- DosResetBuffer
- DosSearchPath
- DosSetFileInfo
- DosSetPathInfo

DosFindNext –
Find the Next Set of File Objects

Example Code

This example gets the first file in the current directory, and then gets the next file.

```
#define INCL_DOSFILEMGR   /* File Manager values */
#include <os2.h>
#include <stdio.h>

#define NORMAL_FILES 0
#define SEARCH_PATTERN "*.*"
#define FILE_ATTRIBUTE NORMAL_FILES

HDIR          FindHandle;
FILEFINDBUF3  FindBuffer;
ULONG         FindCount;
APIRET        rc;           /* Return code */

   FindHandle = 0x0001;
   FindCount = 1;

   rc = DosFindFirst(SEARCH_PATTERN,        /* File pattern */
                     &FindHandle,           /* Directory search handle */
                     FILE_ATTRIBUTE,        /* Search attribute */
                     (PVOID) &FindBuffer,   /* Result buffer */
                     sizeof(FindBuffer),    /* Result buffer length */
                     &FindCount,            /* Number of entries to find */
                     FIL_STANDARD);        /* Return level 1 file info */

   if (rc != 0)
     {
       printf("DosFindFirst error: return code = %ld",rc);
       return;
     }

   rc = DosFindNext(FindHandle,         /* Directory handle */
                    (PVOID) &FindBuffer,  /* Result buffer */
                    sizeof(FindBuffer), /* Result buffer length */
                    &FindCount);        /* Number of entries to find */

   if (rc != 0)
     {
       printf("DosFindNext error: return code = %ld",rc);
       return;
     }
```

DosForceDelete — Remove a File Name from a Directory

```
#define INCL_DOSFILEMGR

APIRET DosForceDelete (PSZ pszFileName)
```

DosForceDelete removes a file name from a directory. The deleted file is not recoverable.

Parameters

pszFileName (PSZ) — input

Address of the name of the file to be deleted.

Returns

Return Code.

DosForceDelete returns the following values:

0	NO_ERROR
2	ERROR_FILE_NOT_FOUND
3	ERROR_PATH_NOT_FOUND
5	ERROR_ACCESS_DENIED
26	ERROR_NOT_DOS_DISK
32	ERROR_SHARING_VIOLATION
36	ERROR_SHARING_BUFFER_EXCEEDED
87	ERROR_INVALID_PARAMETER
206	ERROR_FILENAME_EXCED_RANGE

Remarks

Global file-name characters are not permitted in the name of the file to be deleted.

Read-only files cannot be deleted by DosForceDelete. To delete a read-only file, you must first issue DosSetFileInfo to change the file's read-only attribute to zero, then delete the file.

The deleted file cannot be recovered with the UNDELETE command. You may want to issue DosForceDelete to delete a temporary file that you would not want to recover.

DosForceDelete cannot be used to delete directories. Issue DosDeleteDir to delete a directory.

Related Functions

- DosDelete
- DosDeleteDir
- DosSetFileInfo

DosForceDelete –
Remove a File Name from a Directory

Example Code

This example deletes a file named test.dat from the current directory. The deleted file cannot be recovered.

```
#define INCL_DOSFILEMGR   /* File Manager values */
#include <os2.h>
#include <stdio.h>

#define FILE_DELETE "test.dat"

APIRET  rc;             /* Return code */

   rc = DosForceDelete(FILE_DELETE);   /* File path name */

   if (rc != 0)
     {
       printf("DosForceDelete error: return code = %ld",rc);
       return;
     }
```

DosFreeMem – Free a Private or Shared Memory Object

```
#define INCL_DOSMEMMGR

APIRET DosFreeMem (PVOID pBaseAddress)
```

DosFreeMem frees a private or shared memory object from the virtual-address space of the process.

Parameters

pBaseAddress (PVOID) – input

The base virtual address of the private or shared memory object whose reference is to be freed.

Returns

Return Code.

DosFreeMem returns the following values:

0	NO_ERROR
5	ERROR_ACCESS_DENIED
95	ERROR_INTERRUPT
487	ERROR_INVALID_ADDRESS

Remarks

DosFreeMem releases a previously allocated private or shared memory object from the virtual-address space of the subject process. The released pages are given an access protection of no access.

Freeing a shared memory object decrements the reference count for the associated object. If the resulting count is zero (that is, no other references to the shared memory object exist throughout the system), then the object is deleted. The deletion of the shared memory object releases the backing storage for the committed pages within the object.

Related Functions

- DosAllocMem
- DosAllocSharedMem

DosFreeMem –
Free a Private or Shared Memory Object

Example Code

This example allocates and then frees a private memory object from the virtual address space.

```
#define  INCL_DOSMEMMGR   /* Memory Manager values */
#include <os2.h>
#include <stdio.h>
#include <bsememf.h>   /* Get flags for memory management */

PVOID    BaseAddress;     /* Pointer to the base address of the
                             allocated memory object */
ULONG    Objectsize;      /* Size in bytes of the
                             memory object */
ULONG    AllocationFlags; /* Flags describing characteristics
                             of the memory object */
APIRET   rc;              /* Return code */

   Objectsize = 6000;     /* Ask for a memory object of 6000    */
                          /*   bytes. This size will be rounded */
                          /*   to 8KB.                          */

   AllocationFlags = PAG_WRITE | PAG_READ;
                          /* Permit read and write access to     */
                          /*   the memory block, but do not have */
                          /*   it immediately committed within   */
                          /*   memory                            */

   rc = DosAllocMem(&BaseAddress, Objectsize, AllocationFlags);

   if (rc != 0)
     {
       printf("DosAllocMem error: return code = %ld", rc);
       return;
     }

   rc = DosFreeMem(BaseAddress);   /* Free the memory object */

   if (rc != 0)
     {
       printf("DosFreeMem error: return code = %ld", rc);
       return;
     }
```

DosFreeModule – Frees the Reference to the Dynamic Link Module

---

```
#define INCL_DOSMODULEMGR
```

APIRET DosFreeModule (HMODULE hmodModHandle)

DosFreeModule frees the reference to the dynamic link module for this process.

Parameters

hmodModHandle (HMODULE) – input

The handle of the dynamic link module that is to be freed.

Returns

Return Code.

DosFreeModule returns the following values:

0	NO_ERROR
6	ERROR_INVALID_HANDLE
12	ERROR_INVALID_ACCESS
95	ERROR_INTERRUPT

Remarks

DosFreeModule frees the reference to the dynamic link module for this process.

If the dynamic link module is no longer used by any process, the module is freed from system memory.

The module identified by the handle must have been loaded using DosLoadModule. If the handle is invalid, an error is returned.

After this function has completed, the module handle is no longer valid, and may not be used to refer to the dynamic link module. Procedure entry addresses returned for this module are also no longer valid, and will cause a protection fault if they are invoked.

Related Functions

- DosLoadModule
- DosQueryModuleName

DosFreeModule –
Frees the Reference to the Dynamic Link Module

Example Code

This example tries to load module ABCD. The system searches LIBPATH. If unsuccessful, the system tries to load the module from the program's directory (in case the user forgot to update LIBPATH).

```
#define INCL_DOSMODULEMGR       /* Module Manager values */
#include <os2.h>
#include <stdio.h>

#define MODULE_NAME "abcd"
#define FULL_MODULE_NAME "\\nifty\\abcd.dll"

CHAR    LoadError[100];
HMODULE ModuleHandle;
APIRET  rc;                /* Return code */

   if (DosLoadModule(LoadError,              /* Object name buffer */
                     sizeof(LoadError),      /* Length of object name
                                                buffer */
                     MODULE_NAME,            /* Module name string */
                     &ModuleHandle) == 2)    /* Module handle */

      rc = DosLoadModule(LoadError,          /* Object name buffer */
                         sizeof(LoadError),  /* Length of object name
                                                buffer */
                         FULL_MODULE_NAME,  /* Module name string */
                         &ModuleHandle);     /* Module handle */

   rc = DosFreeModule(ModuleHandle);         /* Module handle */
```

DosFreeResource – Free a Resource

#define INCL_DOSMODULEMGR

APIRET DosFreeResource (PVOID pResAddr)

DosFreeResource frees a resource that was loaded by DosGetResource.

Parameters

pResAddr (PVOID) – input

The address of the resource to be freed.

Returns

Return Code.

DosFreeResource returns the following values:

0	NO_ERROR
5	ERROR_ACCESS_DENIED

Remarks

DosFreeResource frees a resource that was loaded by DosGetResource.

After the last reference to a resource is freed, the memory becomes available for reuse by the system. However, the memory is not reused until the system determines that it cannot satisfy a memory allocation request. This allows the resource to remain in memory in case the process issues DosGetResource again. The system thus avoids having to read the contents of the resource from the disk again.

Related Functions

- DosGetResource

Example Code

This example frees a resource that was previously loaded by DosGetResource. Assume that the address of the resource has been placed into *ResAddr* already.

```
#define INCL_DOSFREERESOURCE
#include <os2.h>
#include <stdio.h>

PVOID    ResAddr;      /* Resource address */
APIRET   rc;           /* Return code */

  rc = DosFreeResource(ResAddr);

  if (rc != 0)
    {
      printf("DosFreeResource error: return code = %ld", rc);
      return;
    }
```

DosFSAttach – Attach a Device

```
#define INCL_DOSFILEMGR
```

```
APIRET DosFSAttach (PSZ pszDevicename, PSZ pszFSDName, PVOID pDataBuffer,
                    ULONG ulDataBufferLen, ULONG ulOpFlag)
```

DosFSAttach attaches or detaches a drive to or from a remote FSD (file system driver), or a pseudocharacter device name to or from a local or remote FSD.

Parameters

pszDevicename (PSZ) – input

A drive designation or a pseudocharacter device name when *OpFlag* is 0 or 1. A drive designation is an ASCIIZ string consisting of the drive name followed by a colon. If an attachment is successful, all requests to that drive are routed to the specified file-system driver. If a detachment is successful, the drive is removed from the system's name space.

DeviceName points to the name of a spooled device when *OpFlag* is 2 or 3. The *DeviceName* format is the same as above. Requests to that name are not seen by the file-system driver.

A pseudocharacter device name (single file device) is an ASCIIZ string consisting of the file-name subdirectory \DEV\. All requests to that name are routed to the specified file-system driver after a successful attachment. A successful detachment removes the name from the system's name space.

pszFSDName (PSZ) – input

Address of the ASCIIZ name of the remote file-system driver that is to be attached to or detached from the device specified by *DeviceName*. For spooled objects, this pointer is set to 0. The pointer to *FSDName* must be set to 0 when *OpFlag* is 2 or 3.

pDataBuffer (PVOID) – input

Address of the user-supplied file-system driver argument data area when *OpFlag* is 0 or 1. The meaning of the data is specific to the file-system driver. *DataBuffer* contains contiguous ASCIIZ strings; the first word of the buffer contains the number of ASCIIZ strings. When *OpFlag* is 2, *DataBuffer* points to a SpoolAttach structure as follows:

```
WORD   hNmPipe;   Handle of named pipe opened by spooler
BYTE   cbSpoolObj; Length of name of spooler object (excluding NULL)
ASCIIZ szSpoolObj; Name of spooler object
```

When *OpFlag* is 3, *DataBuffer* is set to zero.

ulDataBufferLen (ULONG) – input

The length, in bytes, of *Data Buffer*.

ulOpFlag (ULONG) – input

The type of operation to be performed.

- Attach = 0
- Detach = 1
- SpoolAttach = 2
- SpoolDetach = 3

Returns

Return Code.

DosFSAttach returns the following values:

0	NO_ERROR
8	ERROR_NOT_ENOUGH_MEMORY
15	ERROR_INVALID_DRIVE
124	ERROR_INVALID_LEVEL
252	ERROR_INVALID_FSD_NAME
253	ERROR_INVALID_PATH

Remarks

The redirection of drive letters that represent local drives is not supported.

File-system drivers cannot use DosFSAttach to establish open connections that are not attached to a name in the system's name space. They must issue DosFSCtl for such purposes as optimizing UNC connections or establishing access rights. DosFSAttach creates attachments only to drives or devices in the system's name space.

Related Functions

- DosFSCtl

DosFSAttach – Attach a Device

Example Code

This example attaches a drive to a remote FSD (file system driver). Assume that the FSD does not require any user-supplied data arguments.

```
#define INCL_DOSFILEMGR   /* File Manager values */
#include <os2.h>
#include <stdio.h>

UCHAR   DeviceName[8]; /* Device name or drive letter string */
UCHAR   FSDName[40];   /* FSD name */
PVOID   DataBuffer;    /* Attach argument data */
ULONG   DataBufferLen; /* Buffer length */
ULONG   OpFlag;        /* Attach or detach */
APIRET  rc;            /* Return code */

   strcpy(DeviceName,"Y:");
                       /* Drive letter with which to attach the */
                       /*   file system driver                  */

   strcpy(FSDNAME,"\\lan03\\src");

   DataBuffer = 0;     /* Assume that no user-supplied data */
                       /*   arguments are required          */

   DataBufferLen = 0;  /* No data buffer supplied */

   OpFlag = 0;         /* Indicate Attach request */

   rc = DosFSAttach(DeviceName, FSDName, DataBuffer,
                                DataBufferLen, OpFlag);

   if (rc != 0)
     {
       printf("DosFSAttach error: return code = %ld", rc);
       return;
     }
```

DosFSCtl – Communicate with a File System

#define INCL_DOSFILEMGR

```
APIRET DosFSCtl  (PVOID pDataArea, ULONG ulDataLengthMax, PULONG pDataLengthInOut,
                  PVOID pParmList, ULONG ulParmLengthMax, PULONG pParmLengthInOut,
                  ULONG ulFunctionCode, PSZ pszRouteName, HFILE FileHandle,
                  ULONG ulRouteMethod)
```

DosFSCtl provides an extended standard interface between an application and a file-system driver (FSD).

Parameters

pDataArea (PVOID) – input

Address of the data area.

ulDataLengthMax (ULONG) – input

The length, in bytes, of *DataArea*. This is the maximum length of the data to be returned by the file-system driver in *DataArea*. *DataLengthInOut* may be larger than this on input, but not on output.

pDataLengthInOut (PULONG) – input/output

On input, a pointer to the length, in bytes, of the data passed to the file-system driver in *DataArea*. On output, a pointer to the length, in bytes, of the data returned by the file-system driver in *DataArea*. If this function returns ERROR_BUFFER_OVERFLOW, *DataLengthInOut* points to the size of the buffer required to hold the data returned by the file-system driver.

pParmList (PVOID) – input

Address of the command-specific parameter list.

ulParmLengthMax (ULONG) – input

The length, in bytes, of *ParmList*. This is the maximum length of the data to be returned by the file-system driver in *ParmList*. *ParmLengthInOut* may be larger than this on input, but not on output.

pParmLengthInOut (PULONG) – input/output

On input, a pointer to the length, in bytes, of the parameters passed to the file-system driver in *ParmList*. On output, a pointer to the length, in bytes, of the parameters returned by the file-system driver in *ParmList*. If this function returns ERROR_BUFFER_OVERFLOW, *ParmLengthInOut* points to the size of the buffer required to hold the parameters returned by the file-system driver. No other data is returned in this case.

ulFunctionCode (ULONG) – input

The function code that is specific to the file-system driver. For remote file-system drivers, two kinds of DosFSCtl functions are possible: functions that are handled locally, and functions that are exported across the network. If bit hex 4000 is set in *FunctionCode*, this indicates to the remote file-system driver (FSD) that the function should be exported.

Function codes from hex 0000 to hex 7FFF are reserved for use by the operating system. Function codes from hex 8000 to hex BFFF are FSD-defined DosFSCtl functions handled by the local file-system driver. Function codes from hex C000 to hex FFFF are FSD-defined DosFSCtl functions exported to the server.

DosFSCtl –
Communicate with a File System

FunctionCode may have one of the following values:

Value	Definition
1	(FSCTL_ERROR_INFO) Returns error-code information from the file-system driver. On input, the error code is passed to the file-system driver in the first word of *ParmList*. On output, the ASCIIZ string returned in *DataArea* is an explanation of the error code.
2	(FSCTL_MAX_EASIZE) Queries the file-system driver for the maximum size of individual EAs (extended attributes), and the maximum size of the full EA list that it supports. The information is returned in *DataArea* in the following format:

```
EASizeBuf
    USHORT  cb_MaxEASize;      /* Max. size of one EA */
    ULONG   cb_MaxEAListSize;  /* Max. size of full EA list */
```

pszRouteName (PSZ) – input

Address of the ASCIIZ name of the file-system driver, or the path name of a file or directory that the operation applies to.

FileHandle (HFILE) – input

File-specific or device-specific handle.

ulRouteMethod (ULONG) – input

Selects how the request is routed, as follows:

Value	Definition
1	(FSCTL_HANDLE) *FileHandle* directs routing. *RouteName* must be a null pointer (0L). The file-system driver associated with the handle receives the request.
2	(FSCTL_PATHNAME) *RouteName* refers to a path name that directs routing. *FileHandle* must be –1. The file-system driver associated with the drive that the path name refers to at the time of the request receives the request. The path name need not refer to a file or directory that actually exists, only to a drive. A relative path name may be used; it is processed like any other path name.
3	(FSCTL_FSDNAME) *RouteName* refers to a file-system driver name that directs routing. *FileHandle* must be –1. The named file-system driver receives the request.

Returns

Return Code.

DosFSCtl returns the following values:

0	NO_ERROR
1	ERROR_INVALID_FUNCTION
6	ERROR_INVALID_HANDLE
87	ERROR_INVALID_PARAMETER
95	ERROR_INTERRUPT
111	ERROR_BUFFER_OVERFLOW
117	ERROR_INVALID_CATEGORY
124	ERROR_INVALID_LEVEL
252	ERROR_INVALID_FSD_NAME

Related Functions

- DosFSAttach

Example Code

This example demonstrates how a process can communicate with a file system driver (FSD). Assume that the calling process has placed an appropriate file handle into *FileHandle*. Assume that the specified file system recognizes a function code of hex 8100, and that the function accepts an ASCII string as input, requires no specific command parameter list, and returns a string of ASCII characters to the caller.

```
#define INCL_DOSFILEMGR   /* File Manager values */
#include <os2.h>
#include <stdio.h>

UCHAR   DataArea[100];    /* Data area */
ULONG   DataLengthMax;    /* Max. length of Data area */
ULONG   DataLengthInOut;  /* Data area length, in and out */
PVOID   ParmList;         /* Parameter list */
ULONG   ParmLengthMax;    /* Max. length of Parameter list */
ULONG   ParmLengthInOut;  /* Parameter list length, in and out */
ULONG   FunctionCode;     /* Function code */
PSZ     RouteName;        /* Path or FSD name */
HFILE   FileHandle;       /* File handle */
ULONG   RouteMethod;      /* Method for routing */
APIRET  rc;               /* Return code */

   FunctionCode = 0x8100; /* Indicate the function to request of */
                          /*   the file system                   */

   strcpy(DataArea,"PARM1: 98");
                          /* ASCII string to pass to file system */

   DataLengthMax = 100;   /* Tell the file system the maximum */
                          /*   amount of data it can return   */

   DataLengthInOut = strlen(DataArea);
                          /* On input, this is the number of */
                          /*   bytes sent to the file system */

   ParmList = 0;          /* In this example, assume that no   */
   ParmLengthMax = 0;     /*   specific command parameter list */
   ParmLengthInOut = 0;   /*   is required by the file system  */
                          /*   for this function code          */

   RouteMethod = 1;       /* Indicate that the file handle    */
   RouteName = 0;         /*   directs routing (this implies  */
                          /*   that the RouteName variable is */
                          /*   unused in this example)        */

   rc = DosFSCtl(DataArea, DataLengthMax, &DataLengthInOut,
                 ParmList, ParmLengthMax, &ParmLengthInOut,
                 FunctionCode, RouteName, FileHandle,
                 RouteMethod);
                          /* On successful return, the DataArea */
                          /*   buffer will contain the ASCII    */
                          /*   string sent by the file system   */
                          /*   in response to the function      */
                          /*   request, and the DataLengthInOut */
                          /*   variable will contain the number */
                          /*   of bytes placed in the buffer by */
                          /*   the file system                  */
```

```
if (rc != 0)
  {
    printf("DosFSCtl error: return code = %ld", rc);
    return;
  }
```

DosGetDateTime – Get Current Date and Time

```
#define INCL_DOSDATETIME

APIRET DosGetDateTime  (PDATETIME  ppPDateTime)
```

DosGetDateTime gets the current date and time.

Parameters

ppPDateTime (PDATETIME) – output
Pointer to the DateTime data structure.

Hours (UCHAR) Current hour, using values 0 through 23.

Minutes (UCHAR) Current minute, using values 0 through 59.

Seconds (UCHAR) Current second, using values 0 through 59.

Hundredths (UCHAR) Current hundredths of a second, using values 0 through 99.

Day (UCHAR) Current day of the month, using values 1 through 31.

Month (UCHAR) Current month of the year, using values 1 through 12.

Year (USHORT) Current year.

TimeZone (SHORT) The difference in minutes between the current time zone and Greenwich Mean Time (GMT). This value is positive for time zones west of Greenwich, England, and negative for time zones east of Greenwich. A value of -1 indicates that the time zone is undefined.

DayofWeek (UCHAR) Current day of the week, using values 0 through 6. (Sunday is equal to 0.)

Returns

Return Code.

DosGetDateTime returns no values.

Remarks

DosGetDateTime gets the date and time that are maintained by the operating system.

To set the date and time, issue DosSetDateTime.

Related Functions

- DosAsyncTimer
- DosSetDateTime
- DosSleep
- DosStartTimer
- DosStopTimer

DosGetDateTime – Get Current Date and Time

Example Code

The following example shows the use of DosGetDateTime.

```
#define INCL_DOSDATETIME   /* Date and time values */
#include <os2.h>

DATETIME   DateTime;
APIRET     rc;        /* Return code */

rc = DosGetDateTime(&DateTime);          /* Date/Time structure */
```

DosGetInfoBlocks — Get the Addresses of Information Blocks

#define INCL_DOSPROCESS

APIRET DosGetInfoBlocks (PTIB ppppptib, PPIB pppppib)

DosGetInfoBlocks returns the address of the Thread Information Block (TIB) of the current thread. This function also returns the address of the Process Information Block (PIB) of the current process.

Parameters

ppppptib (PTIB) — output

Address of a doubleword in which the address of the Thread Information Block (TIB) of the current thread is returned.

Refer to the **Remarks** section for a description of the Thread Information Block.

pppppib (PPIB) — output

Address of a doubleword in which the address of the Process Information Block (PIB) of the current process is returned.

Refer to the **Remarks** section for a description of the Process Information Block.

Returns

Return Code.

DosGetInfoBlocks returns no values.

Remarks

DosGetInfoBlocks returns the address of the Thread Information Block (TIB) of the current thread. This function also returns the address of the Process Information Block (PIB) of the current process.

Several items of per-thread information are kept in a read/write area of the process address space called the Thread Information Block, or TIB. You can access this information directly after calling DosGetInfoBlocks. Each data item is a doubleword field that describes the current thread as follows:

Field	Description
Exception List Chain	Current thread exception-handler chain
Base ESP	Address of the base of the thread stack
Thread Stack Limit	Address of the end of the thread stack
System-specific Thread Information	Address of a thread information block that is specific to an operating system
Version	Version number of the Thread Information Block
Thread Ordinal Number	Ordinal number of the thread. There is a unique thread ordinal number for every thread in the system.

The System-specific Thread Information block contains the following doubleword fields:

Field	Description
TID	Current thread identifier
Priority	Current thread priority
Version	Version number of the System-specific Thread Information block

DosGetInfoBlocks –
Get the Addresses of Information Blocks

Must Complete The low-order word maintains a count for DosEnterMustComplete and DosExitMustComplete. The high-order word is reserved.

Several items of per-process information are kept in a read/write area of the process address space called the Process Information Block, or PIB. You can access this information directly after calling DosGetInfoBlocks. Each data item is a doubleword field that describes the current process as follows:

Field	Description
PID	Process identifier
PPID	Parent process identifier
Module Handle	Module handle of the current process
Command Line	Address of the command line
Environment	Address of the environment block
Status	Status of the current process
Type	Type of the current process.

Related Functions

- DosCreateThread

Example Code

This example returns the address of the Thread Information Block (TIB) of the current thread. It also returns the address of the Process Information Block (PIB) of the current process. The calling thread can subsequently browse either control block.

```
#define INCL_DOSPROCESS     /* Process and thread values */
#include <os2.h>
#include <stdio.h>

PTIB   pptib;       /* Address of a pointer to the
                       Thread Information Block */
PPIB   pppib;       /* Address of a pointer to the
                       Process Information Block */
APIRET rc;          /* Return code */

   rc = DosGetInfoBlocks(&pptib, &pppib);
                  /* On successful return, the variable pptib */
                  /*   contains the address of the TIB, and   */
                  /*   the variable pppib contains the        */
                  /*   address of the PIB                     */
```

DosGetMessage – Retrieve a Message

#define INCL_DOSMISC

APIRET DosGetMessage (PCHAR pplvTable, ULONG ullvCount, PCHAR ppDataArea, ULONG ulDataLength, ULONG ulMsgNumber, PSZ pszFileName, PULONG pMsgLength)

DosGetMessage retrieves a message from the specified system message file, and inserts variable text-string information into the message.

Parameters

pplvTable (PCHAR) – input

A pointer table. Each doubleword pointer points to an ASCIIZ string or a double-byte character-set (DBCS) string ending in nulls. A maximum of nine strings can be present.

ullvCount (ULONG) – input

The number of variable insertion text strings (0 to 9). If *lvCount* is 0, *lvTable* is ignored.

ppDataArea (PCHAR) – output

The address of the caller's buffer area where the system returns the requested message. If the message is too long to fit in the caller's buffer, then as much of the message text as possible is returned, with the appropriate error return code.

ulDataLength (ULONG) – input

The length, in bytes, of the caller's buffer area.

ulMsgNumber (ULONG) – input

The message number requested.

pszFileName (PSZ) – input

The drive designation, path, and name of the file where the message can be found. The drive designation and path are optional. This specifies a file that was previously prepared by the MKMSGF utility program.

pMsgLength (PULONG) – output

The actual length, in bytes, of the message returned.

Returns

Return Code.

DosGetMessage returns the following values:

0	NO_ERROR
2	ERROR_FILE_NOT_FOUND
206	ERROR_FILENAME_EXCED_RANGE
316	ERROR_MR_MSG_TOO_LONG
317	ERROR_MR_MID_NOT_FOUND
318	ERROR_MR_UN_ACC_MSGF
319	ERROR_MR_INV_MSGF_FORMAT
320	ERROR_MR_INV_IVCOUNT
321	ERROR_MR_UN_PERFORM

DosGetMessage – Retrieve a Message

Remarks

DosGetMessage retrieves a message from the specified system message file, and inserts variable text-string information into the message.

If *lvCount* is greater than 9, DosGetMessage returns an error indicating that *lvCount* is out of range.

If the numeric value of x in the %x sequence for %1 to %9 is less than or equal to *lvCount*, then text insertion, through substitution for %x, is performed for all occurrences of %x in the message. Otherwise, text insertion is ignored, and the %x sequence is returned in the message unchanged. Text insertion is performed for all text strings defined by *lvCount* and *lvTable*.

Variable data insertion does not depend on blank character delimiters, nor are blanks automatically inserted.

For warning and error messages, the 7-character message ID (3-character component ID concatenated with a 4-digit message number) followed by a colon and a blank character is returned as part of the message text. DosGetMessage determines the type of message based on the message classification generated in the output file of MKMSGF.

The following is an example of a sample error message returned with the message ID:

SYS0002: The system cannot find the file specified

DosGetMessage retrieves messages previously prepared by MKMSGF to create a message file, or by MSGBIND to bind a message segment to an .EXE file. First, DosGetMessage tries to retrieve the message from memory in the message segment bound to the .EXE program. If the message cannot be found, DosGetMessage retrieves the message from the message file on DASD (direct access storage device, such as a diskette or fixed-disk).

If the file name is not fully qualified, DosGetMessage searches the following directories for the default drive and path:

1. The system root directory
2. The current working directory
3. Directories listed in the DPATH (protect-mode) statement
4. Directories listed in the APPEND (DOS session) statement.

If a message cannot be retrieved because of a DASD error or a file-not-found condition, the system places an error message into the user's buffer area.

The following error conditions cause the system to place an error message into the user's buffer area:

- Unable to format the system message

  An error message is returned as a result of an invalid parameter (for example, invalid message number or invalid *lvCount*).

- Unable to read the system message file

  An error message is returned when the system message file cannot be read (for example, because of a DASD error or an invalid message-file format).

- Unable to find the system message file

  An error message is returned when the system message file cannot be found.

The presence of the message in memory (EXE bound) or on DASD is not apparent to the caller, and is handled by DosGetMessage. In both cases, you refer to the message by message number and file name.

For DosGetMessage to be called from an input/output privilege level (IOPL) code segment, the following statement must be in the program's definition (.DEF) file:

```
SEGMENT '_MSGSEG' CLASS 'MSGSEGCODE' IOPL CONFORMING
```

In OS/2 Version 2.00, the message segment or object is packed with other application code. If the size of the code segment or object and the bound messages exceeds 64KB, then the message segment or object may be isolated from the application program code by placing the following statement into the program's definition (.DEF) file:

```
SEGMENT  '_MSGSEG' CLASS 'CODE' LOADONCALL       (16-bit application)


SEGMENT  '_MSGSEG32' CLASS 'CODE' LOADONCALL     (32-bit application)
```

Related Functions

- DosInsertMessage
- DosPutMessage
- DosQueryMessageCp

Example Code

This example retrieves a message from a system message file, and inserts variable text-string information into the message. Assume that the third message within the message file contains the string "%1 Error at Station %2". Assume that the caller of DosGetMessage wants to convert this message into the string "Automation Failure Error at Station 69B". Assume that the path name of the relevant message file is "D:\MESSAGE\AUTOMSG.MSG".

```
#define INCL_DOSMISC   /* Miscellaneous values */
#include <os2.h>
#include <stdio.h>

UCHAR   *IvTable[2];    /* Table of variables to insert */
ULONG    IvCount;       /* Number of variables */
UCHAR    DataArea[80];   /* Message buffer (returned) */
ULONG    DataLength;    /* Length of buffer */
ULONG    MsgNumber;     /* Number of the message */
UCHAR    FileName[40];  /* Message file path-name string */
ULONG    MsgLength;     /* Length of message (returned) */
UCHAR    Field1[20];    /* String to substitute into variable
                           field %1 of the message */
UCHAR    Field2[20];    /* String to substitute into variable
                           field %2 of the message */
APIRET   rc;            /* Return code */

   strcpy(Field1,"Automation Failure");
                       /* Define the field with which to   */
                       /*   perform the first substitution */

   strcpy(Field2,"69B");
                       /* Define the field with which to    */
                       /*   perform the second substitution */

   IvTable[0] = Field1; /* Set up the array of pointers to */
   IvTable[1] = Field2; /*   substitute strings            */

   IvCount = 2;         /* Two variable message fields in */
                        /*   message                      */

   DataLength = 80;     /* Data buffer that will receive the */
```

DosGetMessage — Retrieve a Message

```
                    /* complete message is 80 bytes long */

MsgNumber = 3;      /* Specify the third message in the */
                    /*   message file                   */

strcpy(FileName,"D:\\MESSAGE\\AUTOMSG.MSG");
                    /* Path name of the message file */

rc = DosGetMessage(IvTable, IvCount, DataArea, DataLength,
                     MsgNumber, FileName, &MsgLength);

                    /* On successful return, the DataArea */
                    /*   buffer contains the complete     */
                    /*   message (with its two variable   */
                    /*   fields appropriately updated),   */
                    /*   and the MsgLength variable       */
                    /*   contains the length of the       */
                    /*   message that was placed into the */
                    /*   DataArea buffer                  */
if (rc != 0)
  {
    printf("DosGetMessage error: return code = %ld", rc);
    return;
  }
```

DosGetNamedSharedMem – Obtain Access to a Named Shared Memory Object

#define INCL_DOSMEMMGR

```
APIRET DosGetNamedSharedMem (PPVOID ppBaseAddress, PSZ pszSharedMemName,
                             ULONG ulAttributeFlags)
```

DosGetNamedSharedMem obtains access to a named shared memory object.

Parameters

ppBaseAddress (PPVOID) – output

A pointer to a variable that will receive the base address of the shared memory object.

pszSharedMemName (PSZ) – input

The address of the name string associated with the shared memory object. The name is an ASCIIZ string in the format of an OS/2 file name, and is in the subdirectory \SHARMEM\, for example, \SHAREMEM\PUBLIC.DAT.

ulAttributeFlags (ULONG) – input

A set of attribute flags that specify the desired access protection for the shared memory object.

Desired Access Protection

- If the PAG_EXECUTE bit (0x00000004) is set, execute access to the committed pages in the shared memory object is desired.
- If the PAG_READ bit (0x00000001) is set, read access is desired.
- If the PAG_WRITE bit (0x00000002) is set, write access is desired.
- If the PAG_GUARD bit (0x00000008) is set, access to the committed pages in the shared memory object causes a "guard page entered" condition to be raised in the subject process.
- At least one of the bits PAG_READ, PAG_WRITE, or PAG_EXECUTE must be specified.

Returns

Return Code.

DosGetNamedSharedMem returns the following values:

0	NO_ERROR
2	ERROR_FILE_NOT_FOUND
8	ERROR_NOT_ENOUGH_MEMORY
87	ERROR_INVALID_PARAMETER
95	ERROR_INTERRUPT
123	ERROR_INVALID_NAME
212	ERROR_LOCKED

Remarks

DosGetNamedSharedMem obtains access to a named shared memory object.

Getting a named shared memory object allocates the virtual address (of the shared memory object) in the virtual-address space of the process.

When the name of the shared memory object is specified, the name string provided must include the prefix "\SHAREMEM\".

With the Intel 80386 processor, execute and read access are equivalent. Also, write access implies both read and execute access.

DosGetNamedSharedMem — Obtain Access to a Named Shared Memory Object

The value *BaseAddress* returned to the process issuing this function will be the same as that returned to the process that created the shared memory object.

Related Functions

- DosAllocSharedMem
- DosGetSharedMem
- DosGiveSharedMem

Example Code

This example obtains access to a named shared memory object.

```
#define  INCL_DOSMEMMGR   /* Memory Manager values */
#include <os2.h>
#include <stdio.h>
#include <bsememf.h>   /* Get flags for memory management */

PVOID    BaseAddress;     /* Pointer to the base address of
                             the shared memory object */
UCHAR    Name[40];        /* Pointer to the name of the shared
                             memory object to be allocated */
ULONG    AttributeFlags;  /* Flags describing characteristics
                             of the shared memory object */
APIRET   rc;              /* Return code */

   strcpy(Name,"\\SHAREMEM\\BLOCK1.DAT");
                          /* Name of shared memory object */

   AttributeFlags = PAG_WRITE | PAG_READ;
                          /* Request read and write access to */
                          /*   the shared memory object       */

   rc = DosGetNamedSharedMem(&BaseAddress, Name,
                             AttributeFlags);

   if (rc != 0)
     {
       printf("DosGetNamedSharedMem error: return code = %ld", rc);
       return;
     }
```

DosGetResource – Return the Address of the Resource Object

#define INCL_DOSMODULEMGR

```
APIRET DosGetResource (HMODULE hmodModHandle, ULONG ulTypeID, ULONG ulNameID,
                       PPVOID ppOffset)
```

DosGetResource returns the address of the specified resource object.

Parameters

hmodModHandle (HMODULE) – input

The handle of the module that has the required resource. A value of zero means to get the address from the current process. A value other than zero is a module handle that was returned by DosLoadModule.

ulTypeID (ULONG) – input

The type identifier of the 32-bit resource.

ulNameID (ULONG) – input

The name identifier of the 32-bit resource.

ppOffset (PPVOID) – output

The address of a doubleword in which the offset of the resource is returned.

Returns

Return Code.

DosGetResource returns the following values:

0	NO_ERROR
6	ERROR_INVALID_HANDLE
87	ERROR_INVALID_PARAMETER

Remarks

DosGetResource returns the address of the specified resource object.

Resource objects are read-only data objects that can be accessed dynamically at run time. The access key is two 32-bit numbers. The first number is a type ID; the second, a name ID. These are similar to the file extension and file-name portions of a file name.

Resource objects are placed into an executable file by the Resource Compiler (RC.EXE).

Related Functions

- DosFreeResource
- DosLoadModule

DosGetResource –
Return the Address of the Resource Object

Example Code

This example loads a resource object and returns the address of the object. Assume that the handle of the module that contains the desired resource has been placed into *ModHandle* already. Assume that the appropriate resource type identifier has been placed into *TypeID* already, and that the appropriate resource name identifier has been placed into *NameID* already. The two identifiers are derived from the development process that created the module (and its contained resources).

```
#define  INCL_DOSRESOURCES    /* Resource types */
#include <os2.h>
#include <stdio.h>

HMODULE   ModHandle;   /* Handle of the module that has the
                          required resource */
ULONG     TypeID;      /* Resource type ID */
ULONG     NameID;      /* Resource name ID */
PVOID     Offset;      /* Offset of the resource (returned) */
APIRET    rc;          /* Return code */

   rc = DosGetResource(ModHandle, TypeID, NameID, &Offset);
                       /* On successful return, the Offset   */
                       /*   variable contains the linear     */
                       /*   offset of the specified resource */
                       /*   within the module                */

   if (rc != 0)
     {
       printf("DosGetResource error: return code = %ld", rc);
       return;
     }
```

DosGetSharedMem – Obtain Access to a Shared Memory Object

#define INCL_DOSMEMMGR

APIRET DosGetSharedMem (PVOID pBaseAddress, ULONG ulAttributeFlags)

DosGetSharedMem obtains access to a shared memory object.

Parameters

pBaseAddress (PVOID) – input

The base virtual address of the gettable shared memory object as assigned by DosAllocSharedMem.

ulAttributeFlags (ULONG) – input

A set of attribute flags that specify the desired access protection for the shared memory object.

Desired Access Protection

- If the PAG_EXECUTE bit (0x00000004) is set, execute access to the committed pages in the shared memory object is desired.
- If the PAG_READ bit (0x00000001) is set, read access is desired.
- If the PAG_WRITE bit (0x00000002) is set, write access is desired.
- If the PAG_GUARD bit (0x00000008) is set, access to the committed pages in the shared memory object causes a "guard page entered" condition to be raised in the subject process.
- At least one of the bits, PAG_READ, PAG_WRITE, or PAG_EXECUTE must be specified.

Returns

Return Code.

DosGetSharedMem returns the following values:

0	NO_ERROR
5	ERROR_ACCESS_DENIED
8	ERROR_NOT_ENOUGH_MEMORY
87	ERROR_INVALID_PARAMETER
95	ERROR_INTERRUPT
212	ERROR_LOCKED

Remarks

DosGetSharedMem obtains access to a shared memory object.

Getting access to a shared memory object allocates the virtual address (of the shared memory object) in the virtual-address space of the process.

The virtual address of the gettable shared memory object is the base address assigned when the gettable shared memory object was created. The creating and receiving processes must use some form of InterProcess Communication (IPC) to exchange this value.

The shared memory object specified by the virtual address must be gettable (that is, it must have been created using DosAllocSharedMem with the OBJ_GETTABLE attribute set).

Gettable shared memory objects are mapped at the same virtual address in all processes that obtain access to the shared memory object.

DosGetSharedMem –
Obtain Access to a Shared Memory Object

The desired access protection applied to committed pages must be compatible with the access protection granted to the shared memory object when it was created.

With the Intel 80386 processor, execute and read access are equivalent. Also, write access implies both read and execute access.

Related Functions

- DosAllocSharedMem
- DosGetNamedSharedMem
- DosGiveSharedMem

Example Code

This example obtains access to a shared memory object that was created as an unnamed gettable shared memory object.

```
#define  INCL_DOSMEMMGR   /* Memory Manager values */
#include <os2.h>
#include <stdio.h>
#include <bsememf.h>  /* Get flags for memory management */

PVOID   BaseAddress;     /* Pointer to the shared
                            memory object */
ULONG   AttributeFlags;  /* Flags describing characteristics
                            of the shared memory object */
APIRET  rc;              /* Return code */

   AttributeFlags = PAG_WRITE | PAG_READ;
                         /* Request read and write access to */
                         /*   the shared memory object       */

   rc = DosGetSharedMem(&BaseAddress, AttributeFlags);

   if (rc != 0)
     {
       printf("DosGetSharedMem error: return code = %ld", rc);
       return;
     }
```

DosGiveSharedMem – Give Another Process Access to a Shared Memory Object

#define INCL_DOSMEMMGR

APIRET DosGiveSharedMem (PVOID pBaseAddress, PID idProcessId, ULONG ulAttributeFlags)

DosGiveSharedMem gives another process access to a shared memory object.

Parameters

pBaseAddress (PVOID) – input

The base virtual address of the giveable shared memory object as assigned by DosAllocSharedMem.

idProcessId (PID) – input

The identifier of the target process that is to receive access to the shared memory object.

ulAttributeFlags (ULONG) – input

A set of attribute flags that specify the desired access protection for the shared memory object.

Desired Access Protection

- If the PAG_EXECUTE bit (0x00000004) is set, execute access to the committed pages in the shared memory object is desired.
- If the PAG_READ bit (0x00000001) is set, read access is desired.
- If the PAG_WRITE bit (0x00000002) is set, write access is desired.
- If the PAG_GUARD bit (0x00000008) is set, access to the committed pages in the shared memory object causes a a "guard page entered" condition to be raised in the subject process.
- At least one of the bits PAG_READ, PAG_WRITE, or PAG_EXECUTE must be specified.

Returns

Return Code.

DosGiveSharedMem returns the following values:

0	NO_ERROR
5	ERROR_ACCESS_DENIED
8	ERROR_NOT_ENOUGH_MEMORY
87	ERROR_INVALID_PARAMETER
95	ERROR_INTERRUPT
212	ERROR_LOCKED
303	ERROR_INVALID_PROCID
487	ERROR_INVALID_ADDRESS

Remarks

DosGiveSharedMem gives another process access to a shared memory object.

Giving access to a shared memory object allocates the virtual address (of the shared memory object) in the virtual-address space of the target process. This is similar to the target process' performing a DosGetSharedMem operation on the specified shared memory object.

The virtual address of the giveable shared memory object is the base address assigned when the giveable shared memory object was created. The creating and receiving processes must use some form of InterProcess Communication (IPC) to exchange this value.

DosGiveSharedMem –
Give Another Process Access to a Shared Memory Object

Giveable shared memory objects are mapped at the same virtual address in all processes that obtain access to the shared memory object.

The shared memory object specified by the virtual address must be giveable (that is, it must have been created with the OBJ_GIVEABLE attribute set on a call to DosAllocSharedMem).

The desired access protection applied to committed pages must be compatible with the access protection granted to the shared memory object when it was created.

With the Intel 80386 processor, execute and read access are equivalent. Also, write access implies both read and execute access.

Related Functions

- DosAllocSharedMem
- DosGetNamedSharedMem
- DosGetSharedMem

Example Code

This example shows how a process that has created a giveable unnamed shared memory object can give access to that object to another process.

```
#define  INCL_DOSMEMMGR   /* Memory Manager values */
#include <os2.h>
#include <stdio.h>
#include <bsememf.h>   /* Get flags for memory management */

PVOID    BaseAddress;     /* Pointer to the shared
                             memory object */
UCHAR    Name[40];        /* Pointer to the name of the shared
                             memory object to be allocated */
ULONG    ObjectSize;      /* Size in bytes of the
                             the memory object */
ULONG    AttributeFlags;  /* Flags describing characteristics
                             of the shared memory object */
PID      ProcessID;       /* ID of the process that is to receive
                             access to the shared memory object  */
APIRET   rc;              /* Return code */

   strcpy(Name,"");       /* Create an unnamed shared memory */
                          /*   block                         */

   ObjectSize = 6000;     /* Ask for a memory object of 6000    */
                          /*   bytes. This size will be rounded */
                          /*   to 8KB.                          */

   AttributeFlags = OBJ_GIVEABLE | PAG_WRITE | PAG_READ;
                         /* Create a giveable unnamed shared    */
                         /*   memory object.  Permit read and   */
                         /*   write access to the named shared  */
                         /*   memory block, but do not have its */
                         /*   pages immediately committed       */
                         /*   within virtual memory.            */

   rc = DosAllocSharedMem(&BaseAddress, Name,
                         ObjectSize, AttributeFlags);

   if (rc != 0)
     {
       printf("DosAllocSharedMem error: return code = %ld", rc);
       return;
```

```
  }

/* Assume that the variable ProcessID has already been loaded */
/*   with a valid Process ID for another process.  Also       */
/*   assume that this process wishes to give the same         */
/*   Read/Write access to the other process.                  */

rc = DosGiveSharedMem(BaseAddress, ProcessID, AttributeFlags);

if (rc != 0)
  {
    printf("DosGiveSharedMem error: return code = %ld", rc);
    return;
  }
```

DosInsertMessage –
Insert Variable Text-string Information into a Message

```
#define INCL_DOSMISC
```

```
APIRET DosInsertMessage (PCHAR pplvTable, ULONG ullvCount, PSZ pszMsgInput,
                         ULONG ulsgInLength, PCHAR ppDataArea, ULONG ulDataLength,
                         PULONG pMsgLength)
```

DosInsertMessage inserts variable text-string information into a message.

Parameters

pplvTable (PCHAR) – input

A pointer table. Each doubleword pointer points to an ASCIIZ string or a double-byte character-set (DBCS) string ending in nulls. A maximum of nine strings can be present.

ullvCount (ULONG) – input

The number of variable insertion text strings (0 to 9). If *lvCount* is 0, *lvTable* is ignored.

pszMsgInput (PSZ) – input

The address of the input message.

ulsgInLength (ULONG) – input

The length, in bytes, of the input message.

ppDataArea (PCHAR) – output

The address of the caller's buffer area where the system returns the requested message. If the message is too long to fit in the caller's buffer, then as much of the message text as possible is returned, with the appropriate error return code.

ulDataLength (ULONG) – input

The length, in bytes, of the caller's buffer area.

pMsgLength (PULONG) – output

The length, in bytes, of the updated message returned.

Returns

Return Code.

DosInsertMessage returns the following values:

0	NO_ERROR
316	ERROR_MR_MSG_TOO_LONG
320	ERROR_MR_INV_IVCOUNT

Remarks

DosInsertMessage inserts variable text-string information into a message.

DosInsertMessage differs from DosGetMessage in that it does not retrieve a message. It is particularly useful when messages are loaded early, before actual insertion text strings are known.

If *lvCount* is greater than 9, DosInsertMessage returns an error indicating that *lvCount* is out of range. A default message also is placed into the caller's buffer. Refer to DosGetMessage for details about default messages.

If the numeric value of x in the %x sequence for %1-%9 is less than or equal to *lvCount*, then text insertion, by substitution for %x, is performed for all occurrences of %x in the message. Otherwise,

text insertion is ignored, and the %x sequence is returned in the message unchanged. Text insertion is performed for all text-strings defined by *IvCount* and *IvTable*.

Variable data insertion does not depend on blank character delimiters, nor are blanks automatically inserted.

Related Functions

- DosGetMessage
- DosPutMessage
- DosQueryMessageCp

Example Code

This example inserts variable text-string information into a message that resides within program memory. Assume that the message that resides within the program character string variable named *Message*, and contains the string "%1 Error at Station %2". Assume that the caller of DosInsertMessage wants to convert this message into the string "Automation Failure Error at Station 69B".

```
#define INCL_DOSMISC   /* Miscellaneous values */
#include <os2.h>
#include <stdio.h>

UCHAR   *IvTable[2];   /* Table of variables to insert */
ULONG    IvCount;     /* Number of variables */
UCHAR    MsgInput[40] = "%1 Error at Station %2";
                      /* Input message */
ULONG    MsgInLength; /* Length of input message */
UCHAR    DataArea[80];   /* Message buffer (returned) */
ULONG    DataLength;  /* Length of updated message buffer */
ULONG    MsgLength;   /* Length of updated message (returned) */
UCHAR    Field1[20];   /* String to substitute into variable
                          field %1 of the message */
UCHAR    Field2[20];   /* String to substitute into variable
                          field %2 of the message */
APIRET   rc;          /* Return code */

   strcpy(Field1,"Automation Failure");
                       /* Define the field with which to   */
                       /*   perform the first substitution */

   strcpy(Field1,"69B");
                       /* Define the field with which to    */
                       /*   perform the second substitution */

   IvTable[0] = Field1; /* Set up the array of pointers to */
   IvTable[1] = Field2; /*   substitute strings            */

   IvCount = 2;         /* Two variable message fields in */
                        /*   message                      */

   MsgInLength = strlen(MsgInput);
                       /* Length of input message */

   DataLength = 80;     /* Data buffer that will receive the */
                        /*   complete message is 80 bytes in */
                        /*   size                            */

   rc = DosInsertMessage(IvTable, IvCount, MsgInput, MsgInLength,
                         DataArea, DataLength, &MsgLength);
                       /* On successful return, the DataArea */
                       /*   buffer contains the complete     */
```

```
                    /*   message (with its two variable   */
                    /*   fields appropriately updated),   */
                    /*   and the MsgLength variable       */
                    /*   contains the length of the       */
                    /*   message that was placed into the */
                    /*   DataArea buffer                  */

if (rc != 0)
  {
    printf("DosInsertMessage error: return code = %ld", rc);
    return;
  }
```

DosKillProcess – Flag a Process to Terminate

#define INCL_DOSPROCESS

APIRET DosKillProcess (ULONG ulActionCode, PID idProcessID)

DosKillProcess flags a process to terminate, and returns the termination code to its parent (if any).

Parameters

ulActionCode (ULONG) – input

The processes to be flagged for termination. The values of this field are as follows:

Value	Definition
0	(DKP_PROCESSTREE): A process and all its descendant processes. The process must be either the current process, or it must have been directly created by the current process using DosExecPgm with a value of 2 (EXEC_ASYNCRESULT) for *ExecFlags*. After the indicated process terminates, its descendants are flagged for termination. The indicated process need not still be executing. If it has terminated, its descendants are still flagged for termination.
1	(DKP_PROCESS): Any process. Only the indicated process is flagged for termination.

idProcessID (PID) – input

Process ID of the process, or root process of the process tree to be flagged for termination.

Returns

Return Code.

DosKillProcess returns the following values:

0	NO_ERROR
13	ERROR_INVALID_DATA
217	ERROR_ZOMBIE_PROCESS
303	ERROR_INVALID_PROCID
305	ERROR_NOT_DESCENDANT

Remarks

DosKillProcess allows a process to send the KILLPROCESS exception to another process or group of processes. The default action of the system is to terminate each of the processes. A process may intercept this action by installing an exception handler for the KILLPROCESS exception (see DosSetExceptionHandler). In such a case, the program will ensure the integrity of its files, and then issue DosExit.

If there is no exception handler, or if no handler handles the exception , then DosKillProcess affects the process as if one of its threads has issued DosKillProcess for the entire process. All file buffers are written, and the handles opened by the process are closed. Any internal buffers managed by the program externally of the system are not written. An example of such a buffer is a C-language library internal character buffer.

The parent of the process gets the "Unintercepted DosKillProcess" termination code when it issues DosWaitChild.

The "ERROR_ZOMBIE_PROCESS" error code indicates that the specified process has ended, but its parent has not yet issued DosWaitChild to get its return code.

DosKillProcess – Flag a Process to Terminate

Related Functions

- DosExecPgm
- DosExit
- DosExitList
- DosKillThread
- DosWaitChild

Example Code

This example ends a process, and returns a termination code to its parent process (if any). Assume that the target PID value has been placed into *ProcessID* already.

```
#define INCL_DOSPROCESS     /* Process and thread values */
#include <os2.h>
#include <stdio.h>

ULONG   ActionCode;   /* Processes identified
                         for termination */
PID     ProcessID;    /* ID of process or root
                         of process tree */
APIRET  rc;           /* Return code */

   ActionCode = 0;    /* Indicate that the specified process  */
                      /*   and all of its child processes     */
                      /*   have been targeted for termination */

   rc = DosKillProcess(ActionCode, ProcessID);

   if (rc != 0)
     {
       printf("DosKillProcess error: return code = %ld", rc);
       return;
     }
```

DosKillThread – Allow a Thread to End another Thread

```
#define INCL_DOSPROCESS

APIRET DosKillThread (TID idThreadID)
```

DosKillThread allows a thread to end another thread in the current process.

Parameters

idThreadID (TID) – input

Identifier of the thread within the current process to be ended.

Returns

Return Code.

DosKillThread returns the following values:

0	NO_ERROR
170	ERROR_BUSY
309	ERROR_INVALID_THREADID

Remarks

DosKillThread allows a thread to end another thread in the current process.

DosKillThread returns to the caller without waiting for the ending thread to complete its termination processing.

You cannot use this function to end the current thread. If you use DosKillThread to end thread 1, the entire process ends. This is similar to issuing DosExit for thread 1.

If the thread to be ended is executing 16-bit code, or has been created by a 16-bit request, ERROR_BUSY is returned.

Related Functions

- DosCreateThread
- DosExitList
- DosExecPgm
- DosExit
- DosKillProcess
- DosResumeThread
- DosSuspendThread
- DosWaitChild
- DosWaitThread

DosKillThread – Allow a Thread to End another Thread

Example Code

This example shows how a thread within a process can end another thread within that process. Assume that the target thread ID (of the thread that the caller wants to end) has been placed into *ThreadID* already.

```
#define INCL_DOSPROCESS     /* Process and thread values */
#include <os2.h>
#include <stdio.h>

TID     ThreadID;   /* ID of thread to be ended */
APIRET  rc;         /* Return code */

   rc = DosKillThread(ThreadID);

   if (rc != 0)
     {
       printf("DosKillThread error: return code = %ld", rc);
       return;
     }
```

DosLoadModule — Load a Dynamic Link Module

#define INCL_DOSMODULEMGR

APIRET DosLoadModule (PSZ pszObjNameBuf, ULONG ulObjNameBufL, PSZ pszModName, PHMODULE ppModHandle)

DosLoadModule loads a dynamic link module, and returns a handle for the module.

Parameters

pszObjNameBuf (PSZ) — input

The address of a buffer into which the name of an object that contributed to the failure of DosLoadModule is to be placed.

ulObjNameBufL (ULONG) — input

The length, in bytes, of the buffer described by *ObjNameBuf*.

pszModName (PSZ) — input

The address of an ASCIIZ name string that contains the dynamic link module name. The file-name extension used for dynamic link libraries is .DLL.

ppModHandle (PHMODULE) — output

The address of a doubleword in which the handle for the dynamic link module is returned.

Returns

Return Code.

DosLoadModule returns the following values:

0	NO_ERROR
2	ERROR_FILE_NOT_FOUND
3	ERROR_PATH_NOT_FOUND
4	ERROR_TOO_MANY_OPEN_FILES
5	ERROR_ACCESS_DENIED
8	ERROR_NOT_ENOUGH_MEMORY
11	ERROR_BAD_FORMAT
26	ERROR_NOT_DOS_DISK
32	ERROR_SHARING_VIOLATION
33	ERROR_LOCK_VIOLATION
36	ERROR_SHARING_BUFFER_EXCEEDED
95	ERROR_INTERRUPT
108	ERROR_DRIVE_LOCKED
123	ERROR_INVALID_NAME
127	ERROR_PROC_NOT_FOUND
180	ERROR_INVALID_SEGMENT_NUMBER
182	ERROR_INVALID_ORDINAL
190	ERROR_INVALID_MODULETYPE
191	ERROR_INVALID_EXE_SIGNATURE
192	ERROR_EXE_MARKED_INVALID
194	ERROR_ITERATED_DATA_EXCEEDS_64K
195	ERROR_INVALID_MINALLOCSIZE
196	ERROR_DYNLINK_FROM_INVALID_RING
198	ERROR_INVALID_SEGDPL
199	ERROR_AUTODATASEG_EXCEEDS_64K
201	ERROR_RELOCSRC_CHAIN_EXCEEDS_SEGLIMIT
206	ERROR_FILENAME_EXCED_RANGE

DosLoadModule – Load a Dynamic Link Module

295	ERROR_INIT_ROUTINE_FAILED

Remarks

DosLoadModule loads a dynamic link module, and returns a handle for the module.

If the file is an OS/2 dynamic link module, then the module is loaded, and a handle is returned. The returned handle is used for freeing the dynamic link module, getting procedure addresses, and getting the fully qualified file name.

DosLoadModule cannot be issued from ring 2 if the dynamic library routine has an initialization routine, or the process will be terminated.

If the module has an initialization routine that is in an object that has IOPL indicated, any process attempting to use the module will cause a general protection fault, and will be terminated.

Related Functions

- DosExecPgm
- DosFreeModule
- DosQueryModuleName
- DosQueryProcAddr

Example Code

This example loads a module.

```
#define INCL_DOSMODULEMGR     /* Module Manager values */
#include <os2.h>
#include <stdio.h>

#define MODULE_NAME "abcd"
#define FULL_MODULE_NAME "\\nifty\\abcd.dll"

CHAR    LoadError[100];
HMODULE ModuleHandle;
APIRET  rc;           /* Return code */

  rc = DosLoadModule(LoadError,              /* Object name buffer */
                     sizeof(LoadError),      /* Length of object name
                                                buffer */
                     MODULE_NAME,            /* Module name string */
                     &ModuleHandle);         /* Module handle */

  if (rc != 0)                               /* Error occurred */
    {
     printf("DosLoadModule error: return code = %ld", rc);
     return;
    }
```

DosMapCase – Perform Case Mapping

#define INCL_DOSNLS

```
APIRET DosMapCase (ULONG ulLength, PCOUNTRYCODE ppStructure,
                   PCHAR ppBinaryString)
```

DosMapCase performs case mapping on a string of binary values that represent ASCII characters.

Parameters

ulLength (ULONG) – input

The length, in bytes, of the string of binary values to be case-mapped.

ppStructure (PCOUNTRYCODE) – input

A two-doubleword input data structure as follows:

- Doubleword 0: Country Code
- Doubleword 1: Code Page Identifier

Doubleword zero is the binary value of the selected country code, in which 0 means use the case map table for the default system country code. Doubleword one is the binary value of the selected code page identifier, in which 0 means use the case map table for the current process code page of the caller.

The following table shows the country, country code, primary code page, and secondary code page identifier values:

Table 2-1 (Page 1 of 2). Code page identifiers

Country	Country Code	Primary	Secondary
Asian English	099	437	850
Australia	061	437	850
Belgium	032	437	850
Canadian French	002	863	850
Czechoslovakia	042	852	850
Denmark	045	865	850
Finland	358	437	850
France	033	437	850
Germany	049	437	850
Hungary	036	852	850
Iceland	354	850	861
Italy	039	437	850
Japan	081	932	437, 850
Japan SAA	081	942	437, 850
Korea	082	934	437, 850
Korea SAA	082	944	437, 850
Latin America	003	437	850
Netherlands	031	437	850

DosMapCase – Perform Case Mapping

Table 2-1 (Page 2 of 2). Code page identifiers			
Country	**Country Code**	**Primary**	**Secondary**
Norway	047	865	850
People's Republic of China	086	936	437, 850
People's Republic of China SAA	086	946	437, 850
Poland	048	852	850
Portugal	351	860	850
Spain	034	437	850
Sweden	046	437	850
Switzerland	041	437	850
Taiwan	088	938	437, 850
Taiwan SAA	088	948	437, 850
Turkey	090	857	850
United Kingdom	044	437	850
United States	001	437	850
Yugoslavia	038	852	850

Note: Code pages 932, 934, 936, 938, 942, 944, 946, and 948 are supported only with the Asian version of the operating system on Asian hardware.

ppBinaryString (PCHAR) – input/output

The string of binary characters that are to be case-mapped. They are case-mapped in place, and they replace the input, so the results appear in *BinaryString*.

Returns

Return Code.

DosMapCase returns the following values:

0	NO_ERROR
397	ERROR_NLS_OPEN_FAILED
398	ERROR_NLS_NO_CTRY_CODE
401	ERROR_NLS_TYPE_NOT_FOUND
476	ERROR_CODE_PAGE_NOT_FOUND

Remarks

DosMapCase performs case mapping on a string of binary values that represent ASCII characters.

The case map in the country file (the default name is COUNTRY.SYS) that corresponds to the system country code or selected country code, and to the process code page or selected code page, is used to perform the case mapping.

Related Functions

- DosQueryCollate
- DosQueryCp
- DosQueryCtryInfo
- DosQueryDBCSEnv
- DosSetProcessCp

Example Code

This example case maps a string for the default country, and code page 850.

```
#define INCL_DOSNLS   /* National Language Support values */
#include <os2.h>
#include <stdio.h>

#define CURRENT_COUNTRY 0
#define NLS_CODEPAGE 850

COUNTRYCODE Country;
CHAR        BinString[30];
APIRET      rc;                          /* Return code */


   Country.country = CURRENT_COUNTRY;    /* Country code */

   Country.codepage = NLS_CODEPAGE;      /* Code page */

   strcpy(BinString,"Howdy");            /* String to map */

   rc = DosMapCase(sizeof(BinString),    /* Length of string */
                   &Country,             /* Input data structure */
                   BinString);           /* String */
```

DosMove –
Move a File Object

#define INCL_DOSFILEMGR

APIRET DosMove (PSZ pszOldPathName, PSZ pszNewPathName)

DosMove moves a file object to another location, and changes its name.

Parameters

pszOldPathName (PSZ) – input

Address of the old path name of the file or subdirectory to be moved.

pszNewPathName (PSZ) – input

Address of the new path name of the file or subdirectory.

Returns

Return Code.

DosMove returns the following values:

0	NO_ERROR
2	ERROR_FILE_NOT_FOUND
3	ERROR_PATH_NOT_FOUND
5	ERROR_ACCESS_DENIED
17	ERROR_NOT_SAME_DEVICE
26	ERROR_NOT_DOS_DISK
32	ERROR_SHARING_VIOLATION
36	ERROR_SHARING_BUFFER_EXCEEDED
87	ERROR_INVALID_PARAMETER
108	ERROR_DRIVE_LOCKED
206	ERROR_FILENAME_EXCED_RANGE
250	ERROR_CIRCULARITY_REQUESTED
251	ERROR_DIRECTORY_IN_CDS

Remarks

DosMove can be used to change only the name of a file or subdirectory, allowing the file object to remain in the same subdirectory. Global file-name characters are not allowed in the source or target name.

If the specified paths are different, the subdirectory location of the file object is changed also. If a drive is specified for the target, it must be the same as the one specified or implied by the source.

Any attempts to move a parent subdirectory to one of its descendant subdirectories will be rejected, because a subdirectory cannot be both an ancestor and a descendant of the the same subdirectory.

Any attempts to move the current subdirectory or any of its ancestors for the current process, or any other process, will be rejected.

Attributes (times and dates) of the source file object are moved to the target. If read-only files exist in the target path, they are not replaced.

During initialization by an application, DosQuerySysInfo is called to determine the maximum path length allowed by the operating system.

DosMove can be used to change the case of a file on a drive that is controlled by a file system driver (FSD). The following example would change the name of the file to "File.Txt.".

```
DosMove("file.txt","File.Txt")
```

Related Functions

- DosClose
- DosCopy
- DosDelete
- DosQuerySysInfo
- DosQueryCurrentDisk
- DosSetDefaultDisk

Example Code

This example moves a file to another directory, and stores it there under a different name.

```
#define INCL_DOSFILEMGR   /* File Manager values */
#include <os2.h>
#include <stdio.h>

UCHAR   OldPathName[40];  /* Old path name string */
UCHAR   NewPathName[40];  /* New path name string */
APIRET  rc;               /* Return code */

   strcpy(OldPathName,"D:\\PROG\\SRC\\FILE1.DLL");

   strcpy(NewPathName,"C:\\OS2\\DLL\\XYZ.DLL");

   rc = DosMove(OldPathName, NewPathName);

   if (rc != 0)
     {
       printf("DosMove error: return code = %ld", rc);
       return;
     }
```

DosOpen –
Open a File

```
#define INCL_DOSFILEMGR
```

```
APIRET DosOpen  (PSZ pszFileName, PHFILE ppFileHandle, PULONG pActionTaken,
                 ULONG ulFileSize, ULONG ulFileAttribute, ULONG ulOpenFlag,
                 ULONG ulOpenMode, PEAOP2 ppEABuf)
```

DosOpen opens a new file, an existing file, or a replacement for an existing file. An open file can have extended attributes.

Parameters

pszFileName (PSZ) – input

Address of the ASCIIZ path name of the file or device to be opened.

ppFileHandle (PHFILE) – output

Address of the handle for the file.

pActionTaken (PULONG) – output

Address of the variable that receives the value that specifies the action taken by the DosOpen function. If DosOpen fails, this value has no meaning. Otherwise, it is one of the following values:

Value	Definition
1	(FILE_EXISTED) File already existed.
2	(FILE_CREATED) File was created.
3	(FILE_TRUNCATED) File existed and was changed to a given size (file was replaced).

ulFileSize (ULONG) – input

New logical size of the file (end of data, EOD), in bytes. This parameter is significant only when creating a new file or replacing an existing one. Otherwise, it is ignored. It is an error to create or replace a file with a nonzero length if the *OpenMode* Access-Mode flag is set to read-only.

ulFileAttribute (ULONG) – input

Doubleword field containing file attribute bits:

Bit	Description
31 – 6	Reserved, must be 0.
5	(FILE_ARCHIVED 0x00000020) File has been archived.
4	(FILE_DIRECTORY 0x00000010) File is a subdirectory.
3	Reserved, must be 0.
2	(FILE_SYSTEM 0x00000004) File is a system file.
1	(FILE_HIDDEN 0x00000002) File is hidden and does not appear in a directory listing.
0	(FILE_READONLY 0x00000001) File can be read from, but not written to.
0	(FILE_NORMAL 0x00000000) File can be read from or written to.

File attributes apply only if the file is created.

These bits may be set individually or in combination. For example, an attribute value of hex 00000021 (bits 5 and 0 set to 1) indicates a read-only file that has been archived.

ulOpenFlag (ULONG) – input

Doubleword field that indicates the action to be taken depending on whether the file exists or does not exist.

Bits	Description
31 – 8	Reserved, must be 0.
7 – 4	0000: (OPEN_ACTION_FAIL_IF_NEW) Open an existing file; fail if the file does not exist. 0001: (OPEN_ACTION_CREATE_IF_NEW) Create the file if the file does not exist.
3 – 0	0000: (OPEN_ACTION_FAIL_IF_EXISTS) Open the file; fail if the file already exists. 0001: (OPEN_ACTION_OPEN_IF_EXISTS) Open the file if it already exists. 0010: (OPEN_ACTION_REPLACE_IF_EXISTS) Replace the file if it already exists.

ulOpenMode (ULONG) – input

Doubleword field that describes the mode of the open function.

Bit	Description
31 – 16	Reserved, must be zero.
15	(OPEN_FLAGS_DASD 0x00008000) Direct Open flag: 0: *FileName* represents a file to be opened normally. 1: *FileName* is "drive:" (such as C: or A:), and represents a mounted disk or diskette volume to be opened for direct access.
14	(OPEN_FLAGS_WRITE_THROUGH 0x00004000) Write-Through flag: 0: Writes to the file may go through the file-system driver's cache. The file-system driver writes the sectors when the cache is full or the file is closed. 1: Writes to the file may go through the file-system driver's cache, but the sectors are written (the actual file I/O operation is completed) before a synchronous write call returns. This state of the file defines it as a synchronous file. For synchronous files, this bit must be set, because the data must be written to the medium for synchronous write operations. This bit flag is not inherited by child processes.
13	(OPEN_FLAGS_FAIL_ON_ERROR 0x00002000) Fail-Errors flag. Media I/O errors are handled as follows: 0: Reported through the system critical-error handler. 1: Reported directly to the caller by way of a return code. Media I/O errors generated through an IOCtl Category 8 function always get reported directly to the caller by way of return code. The Fail-Errors function applies only to non-IOCtl handle-based file I/O calls. This flag bit is not inherited by child processes.
12	(OPEN_FLAGS_NO_CACHE 0x00001000) No-Cache/Cache flag: 0: The file-system driver should place data from I/O operations into its cache. 1: I/O operations to the file need not be done through the file-system driver's cache. The setting of this bit determines whether file-system drivers should place data into the cache. Like the write-through bit, this is a per-handle bit, and is not inherited by child processes.
11	Reserved; must be 0.

10–8 The locality of reference flags contain information about how the application is to get access to the file. The values are as follows:

Value	Definition
000	(OPEN_FLAGS_NO_LOCALITY 0x00000000) No locality known.
001	(OPEN_FLAGS_SEQUENTIAL 0x00000100) Mainly sequential access.
010	(OPEN_FLAGS_RANDOM 0x00000200) Mainly random access.
011	(OPEN_FLAGS_RANDOMSEQUENTIAL 0x00000300) Random with some locality.

7 (OPEN_FLAGS_NOINHERIT 0x00000080) Inheritance flag:

0: File handle is inherited by a process created from a call to DosExecPgm.

1: File handle is private to the current process.

This bit is not inherited by child processes.

6–4 Sharing Mode flags. This field defines any restrictions to file access placed by the caller on other processes. The values are as follows:

Value	Definition
001	(OPEN_SHARE_DENYREADWRITE 0x00000010) Deny read/write access.
010	(OPEN_SHARE_DENYWRITE 0x00000020) Deny write access.
011	(OPEN_SHARE_DENYREAD 0x00000030) Deny read access.
100	(OPEN_SHARE_DENYNONE 0x00000040) Deny neither read nor write access (deny none).

Any other value is invalid.

3 Reserved; must be 0.

2–0 Access-Mode flags. This field defines the file access required by the caller. The values are as follows:

Value	Definition
000	(OPEN_ACCESS_READONLY 0x00000000) Read-only access
001	(OPEN_ACCESS_WRITEONLY 0x00000001) Write-only access
010	(OPEN_ACCESS_READWRITE 0x00000002) Read/write access.

Any other value is invalid, as are any other combinations.

File sharing requires the cooperation of sharing processes. This cooperation is communicated through sharing and access modes. Any sharing restrictions placed on a file opened by a process are removed when the process closes the file with a DosClose request.

Sharing Mode

Specifies the type of file access that other processes may have. For example, if other processes can continue to read the file while your process is operating on it, specify Deny Write. The sharing mode prevents other processes from writing to the file but still allows them to read it.

Access Mode

Specifies the type of file access (access mode) needed by your process. For example, if your process requires read/write access, and another process has already opened the file with a sharing mode of Deny None, your DosOpen request succeeds. However, if the file is open with a sharing mode of Deny Write, the process is denied access.

If the file is inherited by a child process, all sharing and access restrictions also are inherited.

If an open file handle is duplicated by a call to DosDupHandle, all sharing and access restrictions also are duplicated.

ppEABuf (PEAOP2) – input/output

On input, the address of the extended-attribute buffer, which contains an EAOP2 structure. *fpFEA2List* points to a data area where the relevant *FEA2* list is to be found. *fpGEA2List* and *oError* are ignored.

On output, *fpGEA2List* and *fpFEA2List* are unchanged. The area that *fpFEA2List* points to is unchanged. If an error occurred during the set, *oError* is the offset of the *FEA2* entry where the error occurred. The return code from DosOpen is the error code for that error condition. If no error occurred, *oError* is undefined.

If *EABuf* is zero, then no extended attributes are defined for the file.

If extended attributes are not to be defined or modified, the pointer *EABuf* must be set to zero.

Returns

Return Code.

DosOpen returns the following values:

0	NO_ERROR
2	ERROR_FILE_NOT_FOUND
3	ERROR_PATH_NOT_FOUND
4	ERROR_TOO_MANY_OPEN_FILES
5	ERROR_ACCESS_DENIED
12	ERROR_INVALID_ACCESS
26	ERROR_NOT_DOS_DISK
32	ERROR_SHARING_VIOLATION
36	ERROR_SHARING_BUFFER_EXCEEDED
82	ERROR_CANNOT_MAKE
87	ERROR_INVALID_PARAMETER
99	ERROR_DEVICE_IN_USE
108	ERROR_DRIVE_LOCKED
110	ERROR_OPEN_FAILED
112	ERROR_DISK_FULL
206	ERROR_FILENAME_EXCED_RANGE
231	ERROR_PIPE_BUSY

Remarks

A successful DosOpen request returns a handle for accessing the file. The read/write pointer is set at the first byte of the file. The position of the pointer can be changed with DosSetFilePtr or by read and write operations on the file.

The file's date and time can be queried with DosQueryFileInfo. They are set with DosSetFileInfo.

The read-only attribute of a file can be set with the ATTRIB command.

FileAttribute cannot be set to Volume Label. To set volume-label information, issue DosSetFSInfo with a logical drive number. Volume labels cannot be opened.

FileSize affects the size of the file only when the file is new or a is replacement. If an existing file is opened, *FileSize* is ignored. To change the size of the existing file, issue DosSetFileSize.

The value in *FileSize* is a recommended size. If the full size cannot be allocated, the open request may still succeed. The file system makes a reasonable attempt to allocate the new size in an area that is as nearly contiguous as possible on the medium. When the file size is extended, the values of the new bytes are undefined.

DosOpen — Open a File

The Direct Open bit provides direct access to an entire disk or diskette volume, independent of the file system. This mode of opening the volume that is currently on the drive returns a handle to the calling function; the handle represents the logical volume as a single file. The calling function specifies this handle with a DosDevIOCtl Category 8, Function 0 request to prevent other processes from accessing the logical volume. When you are finished using the logical volume, issue a DosDevIOCtl Category 8, Function 1 request to allow other processes to access the logical volume.

The file-handle state bits can be set by DosOpen and DosSetFHState. An application can query the file-handle state bits, as well as the rest of the Open Mode field, by issuing DosQueryFHState.

You can use an *EAOP2* structure to set extended attributes in *EABuf* when creating a file, replacing an existing file, or truncating an existing file. No extended attributes are set when an existing file is just opened.

A replacement operation is logically equivalent to atomically deleting and re-creating the file. This means that any extended attributes associated with the file also are deleted before the file is re-created.

Related Functions

- DosClose
- DosDevIOCtl
- DosDupHandle
- DosQueryHType
- DosSetFileInfo
- DosSetFilePtr
- DosSetFileSize
- DosSetMaxFH
- DosSetRelMaxFH

Example Code

This example opens a file.

```
#define INCL_DOSFILEMGR   /* File Manager values */
#include <os2.h>
#include <stdio.h>

#define OPEN_FILE 0x01
#define CREATE_FILE 0x10
#define FILE_ARCHIVE 0x20
#define FILE_EXISTS OPEN_FILE
#define FILE_NOEXISTS CREATE_FILE
#define DASD_FLAG 0
#define INHERIT 0x80
#define WRITE_THRU 0
#define FAIL_FLAG 0
#define SHARE_FLAG 0x10
#define ACCESS_FLAG 0x02

#define FILE_NAME "test.dat"
#define FILE_SIZE 800L
#define FILE_ATTRIBUTE FILE_ARCHIVE
#define EABUF 0L

HFILE   FileHandle;
ULONG   Wrote;
ULONG   Action;
PSZ     FileData[100];
APIRET  rc;             /* Return code */

   Action = 2;
```

```
strcpy(FileData,"Data...");

rc = DosOpen(FILE_NAME,                 /* File path name */
             &FileHandle,              /* File handle */
             &Action,                  /* Action taken */
             FILE_SIZE,                /* File primary allocation */
             FILE_ATTRIBUTE,           /* File attribute */
             FILE_EXISTS | FILE_NOEXISTS,      /* Open function type */
             DASD_FLAG | INHERIT |            /* Open mode of the file */
             WRITE_THRU | FAIL_FLAG |
             SHARE_FLAG | ACCESS_FLAG,
             EABUF);                   /* No extended attributes */

if (rc != 0)
  {
    printf("DosOpen error: return code = %ld", rc);
    return;
  }
```

DosOpenEventSem – Open Event Semaphore

```
#define INCL_DOSSEMAPHORES
```

APIRET DosOpenEventSem (PSZ pszName, PHEV ppphev)

DosOpenEventSem opens an event semaphore.

Parameters

pszName (PSZ) – input

A pointer to the ASCIIZ name of the semaphore to open.

This field is null if the semaphore is either an unnamed, shared event semaphore or a private event semaphore (private semaphores are always unnamed). An unnamed event semaphore is identified by the pointer to the event semaphore handle (*phev*). If this field is not null, then the semaphore is a named shared semaphore, and *phev* must be set to zero.

ppphev (PHEV) – input/output

On input, a pointer to the event-semaphore handle to open if *Name* is null. If *Name* is not null, set *phev* to zero.

On output, a pointer to the event-semaphore handle that was opened.

Returns

Return Code.

DosOpenEventSem returns the following values:

0	NO_ERROR
6	ERROR_INVALID_HANDLE
8	ERROR_NOT_ENOUGH_MEMORY
87	ERROR_INVALID_PARAMETER
123	INVALID_NAME
187	ERROR_SEM_NOT_FOUND
291	ERROR_TOO_MANY_OPENS

Remarks

DosOpenEventSem opens (obtains access to) an event semaphore for all of the threads in the calling process.

Note: The process that created the semaphore has immediate access to the semaphore, and does not need to call DosOpenEventSem.

Related Functions

- DosCloseEventSem
- DosCreateEventSem
- DosPostEventSem
- DosQueryEventSem
- DosResetEventSem
- DosWaitEventSem

Example Code

This example opens a system event sempahore.

```
#define INCL_DOSSEMAPHORES   /* Semaphore values */
#include <os2.h>
#include <stdio.h>

UCHAR   Name[40]; /* Semaphore name */
HEV     hev;      /* Event semaphore handle */
APIRET  rc;       /* Return code */

   strcpy(Name,"\\SEM32\\EVENT1");
                  /* Name of the system event semaphore */

   rc = DosOpenEventSem(Name, &hev);
                  /* On successful return, the hev variable */
                  /*   contains the handle of the system    */
                  /*   event semaphore                      */

   if (rc != 0)
     {
       printf("DosOpenEventSem error: return code = %ld", rc);
       return;
     }
```

DosOpenMutexSem – Open Mutex Semaphore

```
#define INCL_DOSSEMAPHORES
```

APIRET DosOpenMutexSem (PSZ pszName, PHMTX ppphmtx)

DosOpenMutexSem opens a mutex semaphore.

Parameters

pszName (PSZ) – input

A pointer to the ASCIIZ name of the semaphore to open.

This field is null if the semaphore is either an unnamed, shared mutex semaphore or a private mutex semaphore (private semaphores are always unnamed). An unnamed mutex semaphore is identified by the pointer to the mutex-semaphore handle (*phmtx*). If this field is not null, the semaphore is a named shared semaphore, and *phmtx* must be set to zero.

ppphmtx (PHMTX) – input/output

On input, a pointer to the mutex-semaphore handle to open if *Name* is null; otherwise, this field is set to zero. On output, a pointer to the mutex-semaphore handle that was opened.

Returns

Return Code.

DosOpenMutexSem returns the following values:

0	NO_ERROR
6	ERROR_INVALID_HANDLE
8	ERROR_NOT_ENOUGH_MEMORY
87	ERROR_INVALID_PARAMETER
105	ERROR_SEM_OWNER_DIED
123	ERROR_INVALID_NAME
187	ERROR_SEM_NOT_FOUND
291	ERROR_TOO_MANY_OPENS

Remarks

DosOpenMutexSem opens (obtains access to) a mutual exclusion (mutex) semaphore for all of the threads in the calling process.

Note: The process that created the semaphore has immediate access to the semaphore, and does not need to call DosOpenMutexSem.

Related Functions

- DosCloseMutexSem
- DosCreateMutexSem
- DosQueryMutexSem
- DosReleaseMutexSem
- DosRequestMutexSem

Example Code

This example opens a system mutex sempahore.

```
#define INCL_DOSSEMAPHORES   /* Semaphore values */
#include <os2.h>
#include <stdio.h>

UCHAR   Name[40]; /* Semaphore name */
HMTX    hmtx;     /* Mutex semaphore handle */
APIRET  rc;       /* Return code */

   strcpy(Name,"\\SEM32\\MUTEX1");
                  /* Name of the system mutex semaphore */

   rc = DosOpenMutexSem(Name, &hmtx);
                  /* On successful return, the hmtx variable */
                  /*   contains the handle of the system     */
                  /*   mutex semaphore                       */

   if (rc != 0)
     {
       printf("DosOpenMutexSem error: return code = %ld", rc);
       return;
     }
```

DosOpenMuxWaitSem – Open MuxWait Semaphore

#define INCL_DOSSEMAPHORES

APIRET DosOpenMuxWaitSem (PSZ pszName, PHMUX ppphmux)

DosOpenMuxWaitSem opens a muxwait semaphore.

Parameters

pszName (PSZ) – input

A pointer to the ASCIIZ name of the semaphore to open.

This field is null if the semaphore is either an unnamed, shared muxwait semaphore or a private muxwait semaphore (private semaphores are always unnamed). An unnamed muxwait semaphore is identified by the pointer to the muxwait-semaphore handle (*phmux*).

If this field is not null, the semaphore is a named, shared semaphore, and *phmux* must be set to 0.

ppphmux (PHMUX) – input/output

On input, a pointer to the muxwait-semaphore handle to open if *Name* is null; otherwise, this field is set to zero. On output, a pointer to the muxwait-semaphore handle that was opened.

Returns

Return Code.

DosOpenMuxWaitSem returns the following values:

0	NO_ERROR
6	ERROR_INVALID_HANDLE
8	ERROR_NOT_ENOUGH_MEMORY
87	ERROR_INVALID_PARAMETER
105	ERROR_SEM_OWNER_DIED
123	ERROR_INVALID_NAME
187	ERROR_SEM_NOT_FOUND
291	ERROR_TOO_MANY_OPENS

Remarks

DosOpenMuxWaitSem opens (obtains access to) a multiple wait (muxwait) semaphore for all of the threads in the calling process.

Note: The process that created the semaphore has immediate access to the semaphore, and does not need to call DosOpenMuxWaitSem.

Related Functions

- DosAddMuxWaitSem
- DosCloseMuxWaitSem
- DosCreateMuxWaitSem
- DosDeleteMuxWaitSem
- DosQueryMuxWaitSem
- DosWaitMuxWaitSem

Example Code

This example opens a system muxwait semaphore.

```
#define INCL_DOSSEMAPHORES   /* Semaphore values */
#include <os2.h>
#include <stdio.h>

UCHAR   Name[40]; /* Semaphore name */
HMUX    hmux;      /* Muxwait semaphore handle */
APIRET  rc;        /* Return code */

   strcpy(Name,"\\SEM32\\MUXWAIT1");
                 /* Name of the system muxwait semaphore */

   rc = DosOpenMuxWaitSem(Name, &hmux);
                 /* On successful return, the hmux variable */
                 /*   contains the handle of the system     */
                 /*   muxwait semaphore                     */

   if (rc != 0)
     {
       printf("DosOpenMuxWaitSem error: return code = %ld", rc);
       return;
     }
```

DosOpenQueue — Open Queue

#define INCL_DOSQUEUES

APIRET DosOpenQueue (PPID ppOwnerPID, PHQUEUE ppQueueHandle, PSZ pszQueueName)

DosOpenQueue gives a client process access to a queue.

Parameters

ppOwnerPID (PPID) — output

A pointer to the process identification of the queue's server process.

ppQueueHandle (PHQUEUE) — output

A pointer to the write handle of the queue to be opened.

pszQueueName (PSZ) — input

A pointer to the ASCIIZ name of the queue to be opened. This is the name that was specified by the server process when it created the queue with DosCreateQueue. The name string must include \QUEUES\ as the first element of the path.

Returns

Return Code.

DosOpenQueue returns the following values:

0	NO_ERROR
334	ERROR_QUE_NO_MEMORY
341	ERROR_QUE_PROC_NO_ACCESS
343	ERROR_QUE_NAME_NOT_EXIST

Remarks

DosOpenQueue opens a queue for a client process.

If the queue was created by a call to the 16-bit DosCreateQueue function, the queue is not accessible to 32-bit DosOpenQueue requests, and ERROR_QUE_PROC_NO_ACCESS is returned.

Related Functions

- DosCloseQueue
- DosCreateQueue
- DosPeekQueue
- DosPurgeQueue
- DosQueryQueue
- DosReadQueue
- DosWriteQueue

Example Code

The following example opens a queue named SPECIAL.QUE for a client process, then closes the queue.

```
#define INCL_DOSQUEUES   /* Queue values */
#include <os2.h>
#include <stdio.h>

#define QUE_NAME "\\QUEUES\\SPECIAL.QUE"

PID     OwnerPID;
HQUEUE  QueueHandle;
APIRET  rc;          /* Return code */

  rc = DosOpenQueue(&OwnerPID,         /* Server process ID */
                    &QueueHandle,      /* Queue handle */
                    QUE_NAME);          /* Queue name string */

  if (rc != 0)
    {
      printf("DosOpenQueue error: return code = %ld",rc);
      return;
    }

  rc = DosCloseQueue(QueueHandle);          /* Queue handle */

  if (rc != 0)
    {
      printf("DosCloseQueue error: return code = %ld",rc);
      return;
    }
```

DosOpenVDD –
Open a Virtual Device Driver

#define INCL_DOSMVDM

APIRET DosOpenVDD (PSZ pszVDDName, PHVDD ppVDDHandle)

DosOpenVDD opens a virtual device driver (VDD), and returns a handle for it.

Parameters

pszVDDName (PSZ) – input

The ASCIIZ name of the virtual device driver to be opened.

ppVDDHandle (PHVDD) – output

The address of a doubleword variable where the handle of the virtual device driver is returned.

Returns

Return Code.

DosOpenVDD returns the following values:

0	NO_ERROR
643	ERROR_VDD_NOT_FOUND
644	ERROR_INVALID_CALLER

Remarks

DosOpenVDD opens a virtual device driver, and returns a handle for it.

If *VDDName* specifies the name of an OS/2 virtual device driver, the returned handle allows an OS/2 protected-mode application to communicate with a virtual device driver by issuing DosRequestVDD.

Issue DosCloseVDD to close the handle of the virtual device driver.

Related Functions

- DosCloseVDD
- DosRequestVDD

Example Code

This example opens a sample virtual device driver (VDD). Assume that the sample virtual device driver has registered a name of "VSAMPLE" with the operating system.

```
#define INCL_DOSMVDM     /* Multiple DOS sessions values */
#include <os2.h>
#include <stdio.h>
#include <vdmm.h>

UCHAR   VDDName[10];  /* Name of VDD */
HVDD    VDDHandle;    /* Handle of VDD */
APIRET  rc;           /* Return code */

   strcpy(VDDName,"VSAMPLE");
                      /* Name that the virtual device driver */
                      /*   chose to register                 */

   rc = DosOpenVDD(VDDName, &VDDHandle);
                      /* On successful return, the VDDHandle */
                      /*   variable contains the handle of   */
                      /*   the virtual device driver         */

   if (rc != 0)
     {
       printf("DosOpenVDD error: return code = %ld", rc);
       return;
     }
```

DosPeekNPipe – Peek Named Pipe

#define INCL_DOSNMPIPES

```
APIRET DosPeekNPipe (HPIPE hpipeHandle, PVOID pBuffer, ULONG ulBufferLen,
                     PULONG pBytesRead, PAVAILDATA pBytesAvail, PULONG pPipeState)
```

DosPeekNPipe examines the data in a named pipe without removing it.

Parameters

hpipeHandle (HPIPE) – input

The named-pipe handle to examine. DosCreateNPipe returns the server handle; DosOpen returns the client handle.

pBuffer (PVOID) – output

A pointer to the output buffer.

ulBufferLen (ULONG) – input

The number of bytes to be read.

pBytesRead (PULONG) – output

A pointer to the number of bytes that were read.

pBytesAvail (PAVAILDATA) – output

A pointer to the 4-byte buffer in which the system returns the number of bytes that were available. The buffer structure is:

Bit	Description
32 – 16	The number of bytes that were buffered in the pipe (including message-header bytes and bytes that have been examined).
15 – 0	The number of bytes that were in the current message (0 for a byte-stream pipe).

pPipeState (PULONG) – output

A pointer to a value that represents the state of the named pipe.

Value	Definition
1	(NP_STATE_DISCONNECTED) Disconnected
2	(NP_STATE_LISTENING) Listening
3	(NP_STATE_CONNECTED) Connected
4	(NP_STATE_CLOSING) Closing

The pipe is in a *disconnected state* immediately after a call to DosCreateNPipe, or DosDisConnectNPipe. A disconnected pipe cannot accept a call to DosOpen. The server must issue DosDisConnectNPipe before the pipe can be opened by a client.

The pipe is in a *listening state* after the server issues DosConnectNPipe. A listening pipe is ready to accept a DosOpen request. If the pipe is not in a listening state, DosOpen returns ERROR_PIPE_BUSY.

The pipe is in a *connected state* after a client has successfully issued DosOpen. The connected pipe allows the server and the client to read and write to the pipe, provided both have valid handles.

The pipe is in a *closing state* after the last DosClose request has been made to the pipe by either the client or the server. When DosClose has been issued for the client handle and all of its duplicates, the client end of the pipe is closed. The serving end must acknowledge the closing of

the client end by issuing either DosDisConnectNPipe or DosClose. Issuing DosClose deallocates the pipe.

Returns

Return Code.

DosPeekNPipe returns the following values:

0	NO_ERROR
230	ERROR_BAD_PIPE
231	ERROR_PIPE_BUSY
233	ERROR_PIPE_NOT_CONNECTED

Remarks

DosPeekNPipe examines the current contents of a named pipe without removing it. It also returns information about the state of the pipe.

DosPeekNPipe never blocks, even if the pipe is in blocking mode; if the pipe cannot be accessed immediately, ERROR_PIPE_BUSY is returned. Because this function does not block, it returns only what is currently in the pipe. Thus, if a message pipe is being examined, only a portion of a message may be returned, even though the specified buffer length could accommodate the entire message.

The value returned in *PipeState* can be used by the client or the server to determine the current state of the pipe and to take appropriate action.

Related Functions

- DosCallNPipe
- DosConnectNPipe
- DosCreateNPipe
- DosDisConnectNPipe
- DosQueryNPHState
- DosQueryNPipeInfo
- DosQueryNPipeSemState
- DosSetNPHState
- DosSetNPipeSem
- DosTransactNPipe
- DosWaitNPipe
- DosClose
- DosDupHandle
- DosOpen
- DosRead
- DosResetBuffer
- DosWrite

DosPeekNPipe — Peek Named Pipe

Example Code

This example peeks into a named pipe. It reads the data in the pipe without removing the data from the pipe. Assume that a previous call to DosOpen provided the named-pipe handle that is contained in *Handle*.

```
#define INCL_DOSNMPIPES   /* Named-pipe values */
#include <os2.h>
#include <stdio.h>

HPIPE    Handle;      /* Pipe handle */
UCHAR    Buffer[200]; /* Address of user buffer */
ULONG    BufferLen;   /* Buffer length */
ULONG    BytesRead;   /* Bytes read (returned) */
struct _AVAILDATA  BytesAvail;  /* Bytes available (returned) */
ULONG    PipeState;   /* Pipe state (returned) */
APIRET   rc;          /* Return code */

   BufferLen = 200;  /* Length of the read buffer */

   rc = DosPeekNPipe(Handle, Buffer, BufferLen,
                     &BytesRead, &BytesAvail, &PipeState);
                     /* On successful return, the input buffer */
                     /*   Buffer will contain up to the first  */
                     /*   200 bytes from the named pipe, the   */
                     /*   variable BytesRead will contain the  */
                     /*   number of bytes read into Buffer,    */
                     /*   the variable BytesAvail will contain */
                     /*   the total number of bytes that were  */
                     /*   available in the pipe, and the       */
                     /*   variable PipeState will contain a    */
                     /*   value indicating the state of the    */
                     /*   named pipe                           */

   if (rc != 0)
     {
       printf("DosPeekNPipe error: return code = %ld", rc);
       return;
     }
```

DosPeekQueue — Peek Queue

#define INCL_DOSQUEUES

```
APIRET DosPeekQueue (HQUEUE QueueHandle, PREQUESTDATA ppRequest,
                     PULONG pDataLength, PPVOID ppDataAddress,
                     PULONG pElementCode, BOOL32 f32NoWait, PBYTE pbElemPriority,
                     HEV SemHandle)
```

DosPeekQueue examines a queue element without removing it from the queue.

Parameters

QueueHandle (HQUEUE) — input

The handle of the queue from which an element is to be examined.

ppRequest (PREQUESTDATA) — output

A pointer to a two-doubleword data area that returns the following information:

Doubleword	Description
1	The identification of the process (PID) that added the element to the queue.
2	An event code that is specified by the application. The data in this word is the same as the data in the *Request* parameter of the DosWriteQueue request for the corresponding queue element. The value of this data is understood by both the client thread and the server thread. There is no special meaning to this data, and the operating system does not alter it.

pDataLength (PULONG) — output

A pointer to the length of the examined data. This field is the same as the *DataLength* that was furnished by DosWriteQueue when the element was added to the queue.

ppDataAddress (PPVOID) — output

A pointer to the address of the element that is to be examined. (This field may or may not be the same as the *DataAddress* that was returned by DosWriteQueue when the element was added to the queue. If QUE_CONVERT_ADDRESS was specified when the queue was created, then the addresses of any elements that are written to the queue by the 16-bit DosWriteQueue function are converted to 32-bit addresses.)

pElementCode (PULONG) — input/output

A pointer to an indicator that specifies whether to start at the beginning of the queue or at a particular element.

Value	Definition
0	The field is set to 0 by the application to indicate "examine the first element in the queue," according to the order that was specified when the queue was created (FIFO, LIFO, or priority).
non-0	The field is set to non-0 by the DosPeekQueue function to identify the element that has been examined (output), or by the owner to indicate "examine the next element" (input).

Note: By contrast, when a DosReadQueue request follows DosPeekQueue, DosReadQueue removes the same element that is identified by *ElementCode*, not the next element in the queue.

f32NoWait (BOOL32) — input

Specifies the action to be performed when there are no elements in the queue:

DosPeekQueue — Peek Queue

Value	Definition
0	(DCWW_WAIT) The requesting thread waits until an element is placed in the queue.
1	(DCWW_NOWAIT) The requesting thread does not wait, and DosPeekQueue returns with ERROR_QUE_EMPTY.

pbElemPriority (PBYTE) — output

A pointer to the element's priority value. This is the value that was specified for *ElemPriority* when DosWriteQueue added the element to the queue. *ElemPriority* is a numeric value in the range of 0 to 15, with 15 being the highest priority.

SemHandle (HEV) — input

The handle of an event semaphore that is to be posted when data is added to the queue and *NoWait* is set to 1. *SemHandle* is ignored if *NoWait* is set to 0. The semaphore may be either private or shared, depending on whether the queue is shared across processes.

Note: The first time an event-semaphore handle is supplied in a DosPeekQueue or DosReadQueue request for which *NoWait* is set to 1, the handle is saved by the system. The same handle must be supplied in all subsequent DosPeekQueue and DosReadQueue requests that are issued for that queue.

Returns

Return Code.

DosPeekQueue returns the following values:

0	NO_ERROR
87	ERROR_INVALID_PARAMETER
330	ERROR_QUE_PROC_NOT_OWNED
333	ERROR_QUE_ELEMENT_NOT_EXIST
337	ERROR_QUE_INVALID_HANDLE
340	ERROR_QUE_PREV_AT_END
342	ERROR_QUE_EMPTY
433	ERROR_QUE_INVALID_WAIT

Remarks

DosPeekQueue examines a queue element without removing it from the queue. This function can be used only by the queue's server process and its threads.

If the *NoWait* parameter is set to 1, an event semaphore must be provided so that the calling thread can determine when an element has been placed into the queue. The semaphore is created by calling DosCreateEventSem, and its handle is supplied in the *SemHandle* parameter of DosPeekQueue.

The first time an event-semaphore handle is supplied in a DosPeekQueue or DosReadQueue request for which *NoWait* has been set to 1, the handle is saved by the system. The same handle must be supplied in all subsequent DosPeekQueue and DosReadQueue requests that are issued for the same queue; if a different handle is supplied, ERROR_INVALID_PARAMETER is returned.

When a client process adds an element to the queue, the system automatically opens and posts the semaphore. The server can either issue DosQueryEventSem periodically to determine whether the semaphore has been posted, or it can issue DosWaitEventSem. DosWaitEventSem causes the calling thread to block until the semaphore is posted.

After the event semaphore has been posted, the calling thread must call DosPeekQueue again to examine the newly added queue element.

Related Functions

- DosCloseQueue
- DosCreateQueue
- DosOpenQueue
- DosPurgeQueue
- DosQueryQueue
- DosReadQueue
- DosWriteQueue

Example Code

This example peeks into a queue. It reads the data within the queue without removing the data from the queue. Assume that a previous call to DosOpenQueue provided the queue handle that is contained in *QueueHandle* Assume that the identifier of the process that owns the queue has been placed into *OwningPID* already.

```
#define INCL_DOSQUEUES   /* Queue values */
#include <os2.h>
#include <stdio.h>

HQUEUE        QueueHandle;   /* Queue handle */
REQUESTDATA   Request;       /* Request-identification data */
ULONG         DataLength;    /* Length of examined element */
PVOID         DataAddress;   /* Address of examined element */
ULONG         ElementCode;   /* Indicator of examined element */
BOOL32        NoWait;        /* No wait if queue is empty */
BYTE          ElemPriority;  /* Priority of examined element */
HEV           SemHandle;     /* Semaphore handle */
PID           OwningPID;     /* PID of queue owner */
APIRET        rc;            /* Return code */

   Request.pid = OwningPID;  /* Set request data block to */
                             /*   indicate queue owner    */

   ElementCode = 0;          /* Indicate that the peek should */
                             /*   start at the front of the   */
                             /*   queue                       */

   NoWait = 0;               /* Indicate that the peek call   */
                             /*   should wait if the queue is */
                             /*   currently empty             */

   SemHandle = 0;            /* Unused since this is a call   */
                             /*   that synchronously waits    */

   rc = DosPeekQueue(QueueHandle, &Request, &DataLength,
                        &DataAddress, &ElementCode, NoWait,
                        &ElemPriority, SemHandle);
                           /* On successful return, the      */
                           /*   DataLength variable contains */
                           /*   the size of the element on   */
                           /*   the queue that is pointed to */
                           /*   by the pointer within the    */
                           /*   DataAddress variable, the    */
                           /*   ElementCode variable has     */
                           /*   been updated to indicate the */
                           /*   next queue element, the      */
                           /*   ElemPriority variable has    */
                           /*   been updated to contain the  */
                           /*   priority of the queue        */
                           /*   element pointed to by the    */
                           /*   DataAddress variable, and    */
```

```
                                /*   the Request.ulData variable  */
                                /*   contains any special data    */
                                /*   that the DosWriteQueue       */
                                /*   caller placed into the queue */

    if (rc != 0)
      {
        printf("DosPeekQueue error: return code = %ld", rc);
        return;
      }
```

DosPhysicalDisk – Obtain Information about Partitionable Disks

#define INCL_DOSPROCESS

```
APIRET DosPhysicalDisk (ULONG ulFunction, PVOID pDataPtr, ULONG ulDataLen,
                        PVOID pParmPtr, ULONG ulParmLen)
```

DosPhysicalDisk obtains information about partitionable disks.

Parameters

ulFunction (ULONG) – input

The type of information to obtain about the partitionable disks, as follows:

Value	Definition
1	(INFO_COUNT_PARTITIONABLE_DISKS) Obtain the total number of partitionable disks.
2	(INFO_GETIOCTLHANDLE) Obtain a handle to use with Category 9 IOCtls.
3	(INFO_FREEIOCTLHANDLE) Release a handle for a partitionable disk.

pDataPtr (PVOID) – input

The address of the buffer where the returned information is placed.

The output data for each function is as follows (all lengths are in bytes):

Function	DataLen	Returned Information
1	2	Total number of partitionable disks in the system (1-based).
2	2	Handle for the specified partitionable disk for the Category 9 IOCtls.
3	0	None - pointer must be zero.

ulDataLen (ULONG) – input

The length, in bytes, of the data buffer.

pParmPtr (PVOID) – input

The address of the buffer used for input parameters.

The input parameters required for each function are as follows (all lengths are in bytes):

Function	ParmLen	Input Parameters
1	0	None - must be set to zero.
2	string length	ASCIIZ string that specifies the partitionable disk.
3	2	Handle obtained from Function 2.

The ASCIIZ string used to specify the partitionable disk must be of the following format:

number : <null byte>

Where:

number specifies the partitionable disk number (1-based) in ASCII.

colon (:) must be present.

<null byte> is the byte of zero for the ASCIIZ string.

DosPhysicalDisk –
Obtain Information about Partitionable Disks

ulParmLen (ULONG) – input

The length, in bytes, of the parameter buffer.

Returns

Return Code.

DosPhysicalDisk returns the following values:

0	NO_ERROR
1	ERROR_INVALID_FUNCTION
87	ERROR_INVALID_PARAMETER

Remarks

DosPhysicalDisk obtains information about partitionable disks. The handle returned for the specified partitionable disk can *only* be used with the DosDevIOCtl function for the Category 9 Generic IOCtl. Use of the handle for a physical partitionable disk is not permitted for handle-based file system functions, such as DosRead or DosClose.

Related Functions

- DosBeep
- DosDevConfig
- DosDevIOCtl

Example Code

This example obtains the total number of partitionable disks in the system. A partitonable disk is a physical disk drive that the calling process can format into partitions.

```
#define INCL_DOSDEVICES   /* Device values */
#include <os2.h>
#include <stdio.h>

ULONG   Function;     /* Type of information */
ULONG   DataBuf;      /* Data return buffer */
ULONG   DataLen;      /* Data return buffer length */
PVOID   ParmPtr;      /* Pointer to user-supplied information */
ULONG   ParmLen;      /* Length of user-supplied information */
APIRET  rc;           /* Return code */

   Function = INFO_COUNT_PARTITIONABLE_DISKS;
                      /* Indicate that a count of the number of */
                      /*   partitionable disks within the       */
                      /*   system is requested                  */

   ParmPtr = 0;       /* No input parameters are relevant for */
   ParmLen = 0;       /*   the requested DosPhysicalDisk      */
                      /*   function                           */

   DataLen = 2;       /* Number of bytes in data return buffer */

   rc = DosPhysicalDisk(Function, &DataBuf, DataLen,
                                  ParmPtr, ParmLen);
                      /* On successful return, the DataBuf   */
                      /*   variable contains the number of   */
                      /*   partitionable disks in the system */

   if (rc != 0)
     {
       printf("DosPhysicalDisk error: return code = %ld", rc);
       return;
     }
```

DosPostEventSem – Post Event Semaphore

```
#define INCL_DOSSEMAPHORES
```

```
APIRET DosPostEventSem (HEV hev)
```

DosPostEventSem posts an event semaphore.

Parameters

hev (HEV) – input

The handle of the event semaphore to post.

Returns

Return Code.

DosPostEventSem returns the following values:

0	NO_ERROR
6	ERROR_INVALID_HANDLE
298	ERROR_TOO_MANY_POSTS
299	ERROR_ALREADY_POSTED

Remarks

DosPostEventSem posts an event semaphore, causing all of the threads that were blocked on DosWaitEventSem requests for that semaphore to execute.

This function can be called by any thread in the process that created the semaphore. Other processes can also call this function, but they must first gain access to the semaphore by calling DosOpenEventSem.

Related Functions

- DosCloseEventSem
- DosCreateEventSem
- DosOpenEventSem
- DosQueryEventSem
- DosResetEventSem
- DosWaitEventSem

Example Code

This example posts a system event semaphore. Assume that the handle of the semaphore has been placed into *hev* already.

```
#define INCL_DOSSEMAPHORES   /* Semaphore values */
#include <os2.h>
#include <stdio.h>

HEV     hev;     /* Event semaphore handle */
APIRET  rc;      /* Return code */

   rc = DosPostEventSem(hev);

   if (rc != 0)
     {
       printf("DosPostEventSem error: return code = %ld", rc);
       return;
     }
```

DosPurgeQueue – Purge Queue

```
#define INCL_DOSQUEUES
```

```
APIRET DosPurgeQueue (HQUEUE QueueHandle)
```

DosPurgeQueue purges a queue of all its elements.

Parameters

QueueHandle (HQUEUE) – input

The handle of the queue to be purged.

Returns

Return Code.

DosPurgeQueue returns the following values:

0	NO_ERROR
330	ERROR_QUE_PROC_NOT_OWNED
337	ERROR_QUE_INVALID_HANDLE

Remarks

The server process issues DosPurgeQueue to empty a queue of all its elements. This function is not available to client processes.

Warning: This is an unconditional purge of all elements in the queue.

Related Functions

- DosCloseQueue
- DosCreateQueue
- DosOpenQueue
- DosPeekQueue
- DosPurgeQueue
- DosQueryQueue
- DosReadQueue
- DosWriteQueue

Example Code

This example shows how the owner of a queue can empty the queue of all data elements. Assume that the owner of the queue has saved the queue's handle (obtained in a previous call to DosCreateQueue) in *QueueHandle*.

```
#define INCL_DOSQUEUES   /* Queue values */
#include <os2.h>
#include <stdio.h>

HQUEUE   QueueHandle;     /* Queue handle */
APIRET   rc;              /* Return code  */

   rc = DosPurgeQueue(QueueHandle);

   if (rc != 0)
     {
       printf("DosPurgeQueue error: return code = %ld", rc);
       return;
     }
```

DosPutMessage –
Sends a Message to an Output File or Device

```
#define INCL_DOSMISC
```

```
APIRET DosPutMessage (HFILE FileHandle, ULONG ulMessageLength,
                      PCHAR ppMessageBuffer)
```

DosPutMessage sends a message to an output file or device.

Parameters

FileHandle (HFILE) – input

The handle of the output file or device.

ulMessageLength (ULONG) – input

The length, in bytes, of the message to be sent.

ppMessageBuffer (PCHAR) – input

The buffer that contains the message to be sent.

Returns

Return Code.

DosPutMessage returns the following values:

0	NO_ERROR
6	ERROR_INVALID_HANDLE
19	ERROR_WRITE_PROTECT
321	ERROR_MR_UN_PERFORM

Remarks

DosPutMessage sends a message that is currently in a buffer to an output file or device.

Screen width is assumed to be 80 characters. If a word would go past column 78, it is moved to the beginning (column 1) of a new line.

DosPutMessage assumes that the starting cursor position is column 1 when handling a word wrap. If the last character to be positioned on a line is a double-byte character, the character is not bisected.

Related Functions

- DosGetMessage
- DosInsertMessage
- DosQueryMessageCp

Example Code

This example sends an edited message to a file. Assume that the message string contained in *MessageBuffer* has been constructed through the use of DosGetMessage or DosInsertMessage already. Assume that *MessageLength* was set (by the call) to the length of the message string that is contained in the buffer. Assume that *FileHandle* has been set to the file handle of the desired output file already.

```
#define INCL_DOSMISC   /* Miscellaneous values */
#include <os2.h>
#include <stdio.h>

HFILE   FileHandle;        /* Handle of output file or device */
ULONG   MessageLength;     /* Length of message buffer */
UCHAR   MessageBuffer[80]; /* Message buffer */
APIRET  rc;                /* Return code */

   rc = DosPutMessage(FileHandle, MessageLength, MessageBuffer);

   if (rc != 0)
     {
       printf("DosPutMessage error: return code = %ld", rc);
       return;
     }
```

DosQueryAppType – Return the Application Type

```
#define INCL_DOSMODULEMGR
```

APIRET DosQueryAppType (PSZ pszExeFileName, PULONG pAppType)

DosQueryAppType returns the application type of an executable file.

Parameters

pszExeFileName (PSZ) – input

An ASCIIZ string that contains the file name of the executable file for which the flags are to be returned.

If the string appears to be a fully qualified path (that is, it contains a " : " in the second position, or it contains a " \ ", or both), then the file is located in the indicated drive:directory. If neither of these is true, and this file name is not found in the current directory, each drive:directory specification in the path defined in the current program's environment is searched for this file. Note that any extension (.xxx) is acceptable for the executable file name. If no extension is specified, a default extension of ".exe" is used.

pAppType (PULONG) – output

A doubleword that will contain flags denoting the application type, as determined by reading the executable file header specified by *ExeFileName*. Note that the call sequence passes a pointer to a location in application memory to return the application type flags.

AppType is defined as follows:

Bits 2, 1 and 0 indicate the application type as specified in the header:

000 - (FAPPTYP_NOTSPEC, 0x00000000): Application type is not specified in the executable header.

001 - (FAPPTYP_NOTWINDOWCOMPAT, 0x00000001): Application type is not-window-compatible.

010 - (FAPPTYP_WINDOWCOMPAT, 0x00000002): Application type is window-compatible.

011 - (FAPPTYP_WINDOWAPI, 0x00000003): Application type is window-API.

Bit 3 (FAPPTYP_BOUND, 0x00000008): Set to 1 if the executable file has been "bound" (by the BIND command) as a Family API application. Bits 0, 1, and 2 still apply.

Bit 4 (FAPPTYP_DLL, 0x00000010): Set to 1 if the executable file is a dynamic link library (DLL) module. Bits 0, 1, 2, 3, and 5 will be set to 0.

Bit 5 (FAPPTYP_DOS, 0x00000020): Set to 1 if the executable file is in PC/DOS format. Bits 0, 1, 2, 3, and 4 will be set to 0.

Bit 6 (FAPPTYP_PHYSDRV, 0x00000040): Set to 1 if the executable file is a physical device driver.

Bit 7 (FAPPTYP_VIRTDRV, 0x00000080): Set to 1 if the executable file is a virtual device driver.

Bit 8 (FAPPTYP_PROTDLL, 0x00000100): Set to 1 if the executable file is a protected-memory dynamic link library module.

Bits 9-13 Reserved.

Bit 14 (FAPPTYP_32BIT, 0x00004000): Set to 1 for 32-bit executable files.

Bit 15 Reserved.

Returns

Return Code.

DosQueryAppType returns the following values:

0	NO_ERROR
2	ERROR_FILE_NOT_FOUND
3	ERROR_PATH_NOT_FOUND
4	ERROR_TOO_MANY_OPEN_FILES
11	ERROR_BAD_FORMAT
15	ERROR_INVALID_DRIVE
32	ERROR_SHARING_VIOLATION
108	ERROR_DRIVE_LOCKED
110	ERROR_OPEN_FAILED
191	ERROR_INVALID_EXE_SIGNATURE
192	ERROR_EXE_MARKED_INVALID

Remarks

DosQueryAppType returns the application type of an executable file.

The Presentation Manager shell uses this function to determine the application type that is being executed.

The application type is specified at link time in the module definition file.

Related Functions

- DosLoadModule
- DosQueryProcType

Example Code

This example obtains the application type of an executable file.

```
#define INCL_DOSSESMGR   /* Session Manager values */
#include <os2.h>
#include <stdio.h>

UCHAR    ExecutableFileName[200];
                  /* Executable file path name string */
ULONG    AppType;  /* Application type flags (returned) */
APIRET   rc;       /* Return code */

   strcpy("C:\\OS2\\SYSLOG.EXE");
                  /* Get the application type of the OS/2 */
                  /*   system error log formatter        */

   rc = DosQueryAppType(ExecutableFileName, &AppType);
                  /* On successful return, the AppType      */
                  /*   variable contains a set of bit flags */
                  /*   that describe the application type   */
                  /*   of the specified executable file     */

   if (rc != 0)
     {
       printf("DosQueryAppType error: return code = %ld", rc);
       return;
     }
```

DosQueryCollate –
Obtain a Collating Sequence Table from the Country File

```
#define INCL_DOSNLS
```

```
APIRET DosQueryCollate (ULONG ulLength, PCOUNTRYCODE ppStructure,
                        PCHAR ppMemBuff, PULONG pDataLength)
```

DosQueryCollate obtains a collating sequence table that resides in the country file.

Parameters

ulLength (ULONG) – input

The length, in bytes, of the data area (*MemBuff*) provided by the caller. A length value of 256 bytes is sufficient.

ppStructure (PCOUNTRYCODE) – input

A two-doubleword input data structure as follows:

- Doubleword 0: Country Code
- Doubleword 1: Code Page Identifier

Doubleword zero is the binary value of the selected country code, in which 0 means return the collate table for the default system country code. Doubleword one is the binary value of the selected code page identifier, in which 0 means return the collate table for the current process code page of the caller.

Refer to the **Parameters** for DosMapCase for a table of values for country code and code page identifier.

ppMemBuff (PCHAR) – output

The data area where the collating sequence table is returned. The caller provides this data area. The input parameter *Length* specifies the length of this area.

If this area is too small to hold all of the available information, then as much information as possible is provided in the available space (in the order in which the data would appear). If the amount of data returned is not enough to fill the memory area provided by the caller, then the memory that is unaltered by the available data is zeroed out. The format of the information returned in this buffer is as follows:

1 Byte	Sort weight of ASCII (0)
1 Byte	Sort weight of ASCII (1)
...	(additional values in collating order)
1 Byte	Sort weight of ASCII (255)

pDataLength (PULONG) – output

The length, in bytes, of the collating sequence table returned.

Returns

Return Code.

DosQueryCollate returns the following values:

0	NO_ERROR
397	ERROR_NLS_OPEN_FAILED
398	ERROR_NLS_NO_CTRY_CODE
399	ERROR_NLS_TABLE_TRUNCATED
401	ERROR_NLS_TYPE_NOT_FOUND
476	ERROR_CODE_PAGE_NOT_FOUND

DosQueryCollate – Obtain a Collating Sequence Table from the Country File

Remarks

DosQueryCollate obtains a collating sequence table (for characters hex 00 through hex FF) that resides in the country file (the default name is COUNTRY.SYS). It is used by the SORT utility to sort text according to the collating sequence.

The collating table returned corresponds to the system country code or selected country code, and to the process code page or selected code page.

Related Functions

- DosMapCase
- DosQueryCp
- DosQueryCtryInfo
- DosQueryDBCSEnv
- DosSetProcessCp

Example Code

This example gets a collating sequence table for the current country, and code page 850.

```
#define INCL_DOSNLS   /* National Language Support values */
#include <os2.h>
#include <stdio.h>

#define CURRENT_COUNTRY 0
#define NLS_CODEPAGE 850

COUNTRYCODE Country;
CHAR        CollBuffer[256];
ULONG       Length;
APIRET      rc;                             /* Return code */

   Country.country = CURRENT_COUNTRY;
   Country.codepage = NLS_CODEPAGE;

   rc = DosQueryCollate(sizeof(CollBuffer),   /* Length of data area
                                                 provided */
                        &Country,          /* Input data structure */
                        CollBuffer,        /* Data area to contain collate
                                              table */
                        &Length);          /* Length of table */

   if (rc != 0)
     {
       printf("DosQueryCollate error: return code = %ld",rc);
       return;
     }
```

DosQueryCp –
Query Current Process Code Page

#define INCL_DOSNLS

APIRET DosQueryCp (ULONG ulLength, PULONG pCodePageList, PULONG pDataLength)

DosQueryCp allows a process to query its current process code page and the prepared system code pages.

Parameters

ulLength (ULONG) – input

The length, in bytes, of *CodePageList*.

pCodePageList (PULONG) – output

The returned data list, in which the first doubleword is the current code page identifier of the calling process.

If one or two code pages have been prepared for the system, then the second doubleword is the first prepared code page, and the third doubleword is the second prepared code page.

If the data length is less than the number of bytes needed to return all of the prepared system code pages, then the returned list is truncated.

The code page identifiers have the following values:

Value	Description
437	United States
850	Multilingual
852	Latin 2 (Czechoslovakia, Hungary, Poland, Yugoslavia)
857	Turkish
860	Portuguese
861	Iceland
863	Canadian French
865	Nordic
932	Japan
934	Korea
936	People's Republic of China
938	Taiwan
942	Japan SAA
944	Korea SAA
946	People's Republic of China SAA
948	Taiwan SAA

Note: Code pages 932, 934, 936, 938, 942, 944, 946, and 948 are supported only with the Asian version of the operating system on Asian hardware.

pDataLength (PULONG) – output

The length, in bytes, of the returned data.

DosQueryCp – Query Current Process Code Page

Returns

Return Code.

DosQueryCp returns the following values:

0	NO_ERROR
473	ERROR_CPLIST_TOO_SMALL
474	ERROR_CP_NOT_MOVED

Remarks

DosQueryCp allows a process to query its current process code page and the prepared system code pages.

The process code page identifier previously set by DosSetProcessCp or inherited by the process is returned to the caller. An input list size of two bytes returns only the current process code page identifier. If no code pages have been prepared with the CODEPAGE command, a length of two and a current code page identifier value of zero are returned.

The system code page identifiers are returned to the caller in the same order as they appear in the CODEPAGE command. The code page identifiers are returned in the following order:

1. The current code page of the process (one of the system code pages).
2. The primary (default) system code page.
3. The secondary system code page, if specified.

Related Functions

- DosMapCase
- DosQueryCollate
- DosQueryCtryInfo
- DosQueryDBCSEnv
- DosSetProcessCp

Example Code

This example gets the current code page, and then up to three other prepared code pages.

```
#define INCL_DOSNLS   /* National Language Support values */
#include <os2.h>
#include <stdio.h>

ULONG  CpList[8];
ULONG  CpSize;
APIRET rc;                              /* Return code */

   rc = DosQueryCp(sizeof(CpList),      /* Length of list */
                   CpList,              /* List */
                   &CpSize);            /* Length of returned list */

   if (rc != 0)
     {
       printf("DosQueryCp error: return code = %ld",rc);
       return;
     }
```

DosQueryCtryInfo – Obtain Country Dependent Formatting Information

---

```
#define INCL_DOSNLS
```

```
APIRET DosQueryCtryInfo (ULONG ulLength, PCOUNTRYCODE ppStructure,
                         PCOUNTRYINFO ppMemBuff, PULONG pDataLength)
```

DosQueryCtryInfo obtains country-dependent formatting information that resides in the country file.

Parameters

ulLength (ULONG) – input

The length, in bytes, of the data area (*MemBuff*) provided by the caller. A length value of 40 bytes is sufficient.

ppStructure (PCOUNTRYCODE) – input

A two-doubleword input data structure as follows:

- Doubleword 0: Country Code
- Doubleword 1: Code Page Identifier

Doubleword zero is the binary value of the selected country code, in which 0 means return the country information for the default system country code. Doubleword one is the binary value of the selected code page identifier, in which 0 means return the country information for the current process code page of the caller.

Refer to the **Parameters** for DosMapCase for a table of values for country code and code page identifier.

ppMemBuff (PCOUNTRYINFO) – output

The data area where the country-dependent information is placed. The caller provides this data area. The input parameter *Length* specifies the size of this area.

If this area is too small to hold all of the available information, then as much information as possible is provided in the available space (in the order in which the data would appear). If the amount of data returned is not enough to fill the memory area provided by the caller, then the memory that is unaltered by the available data is zeroed out. The format of the information returned in this buffer is as follows:

1 DWord Country Code.

1 DWord Reserved (set to zero).

1 DWord Date format: 0 = mm/dd/yy, 1 = dd/mm/yy, 2 = yy/mm/dd.

5 Byte Currency indicator, null terminated.

2 Byte Thousands separator, null terminated.

2 Byte Decimal separator, null terminated.

2 Byte Date separator, null terminated.

2 Byte Time separator, null terminated.

1 Byte Bit field for currency format:

Bit 0: 1 = currency indicator follows money value. 0 = currency indicator precedes money value.

Bit 1: Number of spaces (0 or 1) between currency indicator and money value.

Bit 2: When this bit is set, ignore the first two bits; the currency indicator replaces the decimal indicator.

1 Byte Binary number of decimal places used in the currency indication.

1 Byte Time format for file directory presentation:

Bit 0: 1 = 24 hour 0 = 12 hour with "a" or "p"

2 Word Reserved (set to zero).

2 Byte Data list separator, null terminated.

5 Word Reserved (set to zero).

pDataLength (PULONG) – output

The length, in bytes, of the country-dependent data returned.

Returns

Return Code.

DosQueryCtryInfo returns the following values:

0	NO_ERROR
397	ERROR_NLS_OPEN_FAILED
398	ERROR_NLS_NO_CTRY_CODE
399	ERROR_NLS_TABLE_TRUNCATED
401	ERROR_NLS_TYPE_NOT_FOUND
476	ERROR_CODE_PAGE_NOT_FOUND

Remarks

DosQueryCtryInfo obtains country-dependent formatting information that resides in the country file (the default name is COUNTRY.SYS).

The country-dependent information returned corresponds to the system country code or selected country code, and to the process code page or selected code page.

Related Functions

- DosMapCase
- DosQueryCollate
- DosQueryCp
- DosQueryDBCSEnv
- DosSetProcessCp

DosQueryCtryInfo –
Obtain Country Dependent Formatting Information

Example Code

This example gets country-dependent information.

```
#define INCL_DOSNLS   /* National Language Support values */
#include <os2.h>
#include <stdio.h>

#define CURRENT_COUNTRY 0
#define NLS_CODEPAGE 850

COUNTRYCODE  Country;
COUNTRYINFO  CtryBuffer;
ULONG        Length;
APIRET       rc;                          /* Return code */

   Country.country = CURRENT_COUNTRY;
   Country.codepage = NLS_CODEPAGE;

   rc = DosQueryCtryInfo(sizeof(CtryBuffer),  /* Length of data area
                                                 provided */
                         &Country,          /* Input data structure */
                         &CtryBuffer,       /* Data area to be filled
                                               by function */
                         &Length);          /* Length of data
                                               returned */
   if (rc != 0)
     {
       printf("DosQueryCtryInfo error: return code = %ld",rc);
       return;
     }
```

DosQueryCurrentDir – Get the Full Path Name of the Current Directory

#define INCL_DOSFILEMGR

```
APIRET DosQueryCurrentDir (ULONG ulDriveNumber, PBYTE pbDirPath,
                           PULONG pDirPathLen)
```

DosQueryCurrentDir gets the full path name of the current directory for the requesting process on the specified drive.

Parameters

ulDriveNumber (ULONG) – input

Drive number. The value 0 means the current drive, 1 means drive A, 2 means drive B, 3 means drive C, and so on.

pbDirPath (PBYTE) – output

Address of the fully qualified path name of the current directory.

pDirPathLen (PULONG) – input/output

Address of the length, in bytes, of the *DirPath* buffer. On input, this field contains the length, in bytes, of the directory path buffer. On output, if an error is returned because the buffer is too small, this field contains the required length, in bytes, of the buffer.

Returns

Return Code.

DosQueryCurrentDir returns the following values:

0	NO_ERROR
15	ERROR_INVALID_DRIVE
26	ERROR_NOT_DOS_DISK
108	ERROR_DRIVE_LOCKED
111	ERROR_BUFFER_OVERFLOW

Remarks

The drive letter is not part of the returned string. The string does not begin with a backslash, and it ends with a byte containing hex 00.

The system provides the length of the returned path-name string in *DirPathLen*, which does not include the ending null byte. If the *DirPath* buffer is not large enough to hold the current-directory path string, the system returns the required length, in bytes, for the *DirPath* buffer in *DirPathLen*.

For file-system drivers, the case of the current directory is set at the time of creation. For example, if the directory "c:\bin" is created, and is called with a *DirName* of "c:\bin", the current directory returned by DosQueryCurrentDir will be "c:\bin."

Programs running without the NEWFILES bit set are allowed to issue DosSetCurrentDir for a directory that is not in the 8.3 file-name format.

An application must issue DosQuerySysInfo to determine the maximum path length supported by the operating system. The returned value should be used to dynamically allocate buffers that are to be used to store paths.

DosQueryCurrentDir –
Get the Full Path Name of the Current Directory

Related Functions

- DosQueryCurrentDisk
- DosQuerySysInfo
- DosSetCurrentDir
- DosSetDefaultDisk

Example Code

This example gets the full path name of the current directory on the specified drive.

```
#define INCL_DOSFILEMGR   /* File Manager values */
#include <os2.h>
#include <stdio.h>

ULONG    DriveNumber;   /* Drive number */
UCHAR    DirPath[256];  /* Directory path buffer (returned) */
ULONG    DirPathLen;    /* Directory path buffer length
                           (number of bytes) */
APIRET   rc;            /* Return code */

   DriveNumber = 3;     /* Specify drive C */

   DirPathLen = 256;    /* Length of the DirPath buffer */

   rc = DosQueryCurrentDir(DriveNumber, DirPath, &DirPathLen);
                        /* On successful return, the directory  */
                        /*   path name string is returned in    */
                        /*   the DirPath buffer, and the length */
                        /*   of the path name is returned in    */
                        /*   the DirPathLen variable            */

   if (rc != 0)
     {
       printf("DosQueryCurrentDir error: return code = %ld", rc);
       return;
     }
```

DosQueryCurrentDisk – Get the Current Default Drive

#define INCL_DOSFILEMGR

APIRET DosQueryCurrentDisk (PULONG pDriveNumber, PULONG pLogicalDriveMap)

DosQueryCurrentDisk gets the current default drive for the requesting process.

Parameters

pDriveNumber (PULONG) – output

Address of the number of the default drive. The value 1 means drive A, 2 means drive B, 3 means drive C, and so on.

pLogicalDriveMap (PULONG) – output

Address of the bit map (stored in the low-order portion of the 32-bit, doubleword area) where the system returns the mapping of the logical drives. Logical drives A to Z have one-to-one mapping with bit positions 0 to 25 of the map; for example, bit 0 represents drive A, bit 1 represents drive B, and so on. The settings of these bits indicate which drives exist, as follows:

Value	Definition
0	The logical drive does not exist.
1	The logical drive exists.

Returns

Return Code.

DosQueryCurrentDisk returns the following values:

0	NO_ERROR

Related Functions

- DosSetDefaultDisk

DosQueryCurrentDisk — Get the Current Default Drive

Example Code

This example gets the current default drive and a bit map that specifies which logical drives are currently valid for the calling process.

```
#define INCL_DOSFILEMGR   /* File Manager values */
#include <os2.h>
#include <stdio.h>

ULONG   DriveNumber;      /* Default drive number (returned) */
ULONG   LogicalDriveMap;  /* Drive map area (returned) */
APIRET  rc;               /* Return code */

   rc = DosQueryCurrentDisk(&DriveNumber, &LogicalDriveMap);
                          /* On return, the DriveNumber variable */
                          /*   contains a number (1-26) that     */
                          /*   indicates the current logical     */
                          /*   drive, and the LogicalDriveMap    */
                          /*   variable contains a bit map       */
                          /*   (in bits 0-25 of the long word)   */
                          /*   that specifies which logical      */
                          /*   drives are currently valid        */

   if (rc != 0)
     {
       printf("DosQueryCurrentDisk error: return code = %ld", rc);
       return;
     }
```

DosQueryDBCSEnv — Obtain a DBCS Environmental Vector

#define INCL_DOSNLS

```
APIRET DosQueryDBCSEnv (ULONG ulLength, PCOUNTRYCODE ppStructure,
                        PCHAR ppMemoryBuffer)
```

DosQueryDBCSEnv obtains a DBCS (double-byte character set) environmental vector that resides in the country file.

Parameters

ulLength (ULONG) — input

The length, in bytes, of the data area (*MemoryBuffer*) provided by the caller. A length value of 12 bytes is sufficient. The caller can always determine if all of the information has been obtained, because it terminates with four bytes of zeros. A length of 4 is sufficient for information returned from non-DBCS-related countries.

ppStructure (PCOUNTRYCODE) — input

A two-doubleword input data structure as follows:

- Doubleword 0: Country Code
- Doubleword 1: Code Page Identifier

Doubleword zero is the binary value of the selected country code, in which 0 means return the DBCS information for the default system country code. Doubleword one is the binary value of the selected code page identifier, in which 0 means return the DBCS information for the current process code page of the caller.

Refer to the **Parameters** for DosMapCase for a table of values for country code and code page identifier.

ppMemoryBuffer (PCHAR) — output

The data area where the country-dependent information for the DBCS environmental vector is returned. The caller provides this memory area. The size of the area is specified by the input parameter *Length*.

If this area is too small to hold all of the available information, then as much information as possible is provided in the available space (in the order in which the data would appear). Assuming that the data area is large enough, the valid information is terminated by two bytes of zero. The format of the information returned in this buffer is as follows:

2 Bytes First range definition for DBCS lead byte values:

Byte 1: binary start value (inclusive) for range one
Byte 2: binary stop value (inclusive) for range one

2 Bytes Second range definition:

Byte 1 binary start value for range two
Byte 2 binary stop value for range two

2 Bytes Nth range definition:

Byte 1: binary start value for Nth range
Byte 2: binary stop value for Nth range

2 Bytes Two bytes of binary zero terminate the list.

```
For example: DB   81H,9FH
             DB   E0H,FCH
             DB   0,0
```

DosQueryDBCSEnv –
Obtain a DBCS Environmental Vector

Returns

Return Code.

DosQueryDBCSEnv returns the following values:

0	NO_ERROR
397	ERROR_NLS_OPEN_FAILED
398	ERROR_NLS_NO_CTRY_CODE
399	ERROR_NLS_TABLE_TRUNCATED
401	ERROR_NLS_TYPE_NOT_FOUND
476	ERROR_CODE_PAGE_NOT_FOUND

Remarks

DosQueryDBCSEnv obtains a double-byte character set environmental vector that resides in the country file (the default name is COUNTRY.SYS).

The vector returned corresponds to the system country code or selected country code, and to the process code page or selected code page.

A double-byte character set is for a code page that contains more than 256 characters. A DBCS data string may contain both SBCS (single-byte character set) and DBCS (double-byte character set) characters.

A DBCS character is two bytes in length. It contains a lead byte and a trail byte. A lead byte is in the range returned by DosQueryDBCSEnv. A trail byte is not restricted to any range. The trail byte always follows the lead byte in a DBCS character.

Related Functions

- DosMapCase
- DosQueryCollate
- DosQueryCp
- DosQueryCtryInfo
- DosSetProcessCp

Example Code

This example obtains a DBCS (double-byte character set) environmental vector that resides in the country file (default name COUNTRY.SYS). The vector returned corresponds to the system country code or selected country code and the process code page or selected code page.

```
#define INCL_DOSNLS   /* National Language Support values */
#include <os2.h>
#include <stdio.h>

ULONG         Length;       /* Length of data
                               area provided */
COUNTRYCODE   Structure;    /* Input data structure */
UCHAR         MemoryBuffer[12]; /* DBCS environmental
                               vector (returned) */
APIRET        rc;           /* Return code */

   Length = 12;             /* A length of 12 bytes is     */
                            /*   sufficient to contain the */
                            /*   DBCS data returned        */

   Structure.country = 0;   /* Use the default system */
                            /*   country code         */

   Structure.codepage = 0;  /* Return DBSC information for the */
                            /*   caller's current process code */
                            /*   page                          */

   rc = DosQueryDBCSEnv(Length, &Structure, MemoryBuffer);

                            /* On successful return, the buffer */
                            /*   MemoryBuffer contains the      */
                            /*   country-dependent information  */
                            /*   for the DBCS environmental     */
                            /*   vector                         */

   if (rc != 0)
     {
       printf("DosQueryDBCSEnv error: return code = %ld", rc);
       return;
     }
```

DosQueryEventSem – Query Event Semaphore

#define INCL_DOSSEMAPHORES

APIRET DosQueryEventSem (HEV hev, PULONG ppulPostCt)

DosQueryEventSem retrieves the post count for an event semaphore.

Parameters

hev (HEV) – input

The handle of the event semaphore to query.

ppulPostCt (PULONG) – output

A pointer to the semaphore's post count. The post count is the number of times DosPostEventSem has been called since the last time the event semaphore was in the reset state.

Returns

Return Code.

DosQueryEventSem returns the following values:

0	NO_ERROR
6	ERROR_INVALID_HANDLE
87	ERROR_INVALID_PARAMETER

Remarks

DosQueryEventSem returns the post count for an event semaphore. The post count is the number of times that DosPostEventSem has been called since the last time the semaphore was in the reset state.

This function can be called by any thread in the process that created the semaphore. Other processes can also call this function, but they must first gain access to the semaphore by calling DosOpenEventSem.

Related Functions

- DosCloseEventSem
- DosCreateEventSem
- DosOpenEventSem
- DosPostEventSem
- DosResetEventSem
- DosWaitEventSem

Example Code

This example retrieves the post count for an event semaphore. Assume that the handle of the semaphore has been placed into *hev* already.

```
#define INCL_DOSSEMAPHORES   /* Semaphore values */
#include <os2.h>
#include <stdio.h>

HEV     hev;       /* Event semaphore handle  */
ULONG   ulPostCt;  /* Current post count for the semaphore
                      (returned) */
APIRET  rc;        /* Return code */

   rc = DosQueryEventSem(hev, &ulPostCt);
                  /* On successful return, the ulPostCt  */
                  /*   variable contains the number of   */
                  /*   posts that have been performed on */
                  /*   the event semaphore since the     */
                  /*   last time it was reset            */

   if (rc != 0)
     {
       printf("DosQueryEventSem error: return code = %ld", rc);
       return;
     }
```

DosQueryFHState – Query File Handle State

#define INCL_DOSFILEMGR

APIRET DosQueryFHState (HFILE FileHandle, PULONG pFileHandleState)

DosQueryFHState queries the state of the specified file handle.

Parameters

FileHandle (HFILE) – input

Handle of the file to be queried.

pFileHandleState (PULONG) – output

Address of the contents of the open-mode word defined in a previous DosOpen function.

Bit	Description
15	(OPEN_FLAGS_DASD 0x00008000) Direct Open flag: 0 : *PathName* represents a file to be opened normally. 1 : *PathName* is "drive:" (such as C: or A:). It represents a mounted disk or diskette volume to be opened for direct access.
14	(OPEN_FLAGS_WRITE_THROUGH 0x00004000) Write-Through flag: 0 : Write operations to the file go through the file system buffer cache. 1 : Write operations to the file may go through the file system buffer cache, but the sectors are written (the actual file I/O operation is completed) before a synchronous write call returns. This state of the file defines it as a synchronous file. For synchronous files, this bit is set to 1 because the data must be written to the medium for synchronous write operations. The Write-Through flag bit is not inherited by child processes.
13	(OPEN_FLAGS_FAIL_ON_ERROR 0x00002000) Fail-Errors flag. Media I/O errors are handled as follows: 0 : Reported through the system critical-error handler. 1 : Reported directly to the caller by a return code. Media I/O errors generated through an IOCtl Category 8 function always are reported directly to the caller by a return code. The Fail-Errors function applies only to non-IOCtl handle-based file I/O functions. The Fail-Errors flag bit is not inherited by child processes.
12	(OPEN_FLAGS_NO_CACHE 0x00001000) Cache or No-Cache: 0 : The disk driver should place data from I/O operations into the cache on this file. 1 : I/O operations to the file need not be done through the disk-driver cache. The setting of this bit determines whether it is worth caching the data for file-systems drivers and device drivers. This bit, like the Write-Through bit, is a per-handle bit. This bit is not inherited by child processes.
11 – 8	Reserved bits.
7	(OPEN_FLAGS_NOINHERIT 0x00000080) Inheritance flag: 0 : The file handle is inherited by a process that is created by issuing DosExecPgm.

1 : The file handle is private to the current process.

This bit is not inherited by child processes.

6–4 Sharing-Mode flags: Define the operations other processes can perform on the file:

Value	Definition
001	(OPEN_SHARE_DENYREADWRITE) Deny read and write access.
010	(OPEN_SHARE_DENYWRITE) Deny write access.
011	(OPEN_SHARE_DENYREAD) Deny read access.
100	(OPEN_SHARE_DENYNONE) Deny neither read nor write access (deny none).

Any other value is invalid.

3 Reserved.

2–0 Access-Mode flags. File access is assigned as follows:

Value	Definition
000	(OPEN_ACCESS_READONLY) Read-only access.
001	(OPEN_ACCESS_WRITEONLY) Write-only access.
010	(OPEN_ACCESS_READWRITE) Read and write access.

Any other value is invalid.

Returns

Return Code.

DosQueryFHState returns the following values:

0	NO_ERROR
6	ERROR_INVALID_HANDLE

Remarks

When the application cannot handle a critical error that occurs, critical-error handling can be reset to the system. This is done by having DosSetFHState turn off the fail/errors bit and then reissuing the I/O function. The expected critical error recurs, and control is passed to the system critical-error handler. The precise time that the effect of this function is visible at the application level is unpredictable when asynchronous I/O operations are pending.

The Direct Open bit parameter is the "Direct I/O flag." It provides an access mechanism to a disk or diskette volume independent of the file system. This mode should be used only by system programs and not by application programs.

Named-Pipe Considerations

As defined by the operating system, D = 0. Other bits are as defined by DosCreateNPipe (serving end), DosOpen (client end), or the last DosSetFHState.

Related Functions

- DosDevIOCtl
- DosOpen
- DosSetFHState

DosQueryFHState – Query File Handle State

Example Code

This example queries the state of a file, given its file handle. Assume that the appropriate file handle has been placed into *FileHandle* already.

```
#define INCL_DOSFILEMGR   /* File Manager values */
#include <os2.h>
#include <stdio.h>

HFILE    FileHandle;      /* File handle */
ULONG    FileHandleState; /* File handle state (returned) */
APIRET   rc;              /* Return code */

   rc = DosQueryFHState(FileHandle, &FileHandleState);
                         /* On successful return, the    */
                         /*   FileHandleState variable   */
                         /*   contains a set of          */
                         /*   information characterizing */
                         /*   the current state of the   */
                         /*   file (as it has been       */
                         /*   opened by the calling      */
                         /*   process)                   */

   if (rc != 0)
     {
       printf("DosQueryFHState error: return code = %ld", rc);
       return;
     }
```

DosQueryFileInfo – Query File Information

#define INCL_DOSFILEMGR

APIRET DosQueryFileInfo (HFILE FileHandle, ULONG ulFileInfoLevel, PVOID pFileInfoBuf, ULONG ulFileInfoBufSize)

DosQueryFileInfo gets file information.

Parameters

FileHandle (HFILE) – input

File handle.

ulFileInfoLevel (ULONG) – input

Level of file information required. A value of 1, 2, or 3 can be specified, as follows:

Value	Description
1	(FIL_STANDARD) Level 1 file information
2	(FIL_QUERYEASIZE) Level 2 file information
3	(FIL_QUERYEASFROMLIST) Level 3 file information

Level 4 is reserved.

The structures described in *FileInfoBuf* indicate the information returned for each of these levels.

pFileInfoBuf (PVOID) – output

Address of the storage area where the system returns the requested level of file information. File information, where applicable, is at least as accurate as the most recent DosClose, DosResetBuffer, DosSetFileInfo, or DosSetPathInfo.

Level 1 File Information
FileInfoBuf contains the FILESTATUS3 data structure, to which file information is returned.

Level 2 File Information
FileInfoBuf contains the FILESTATUS4 data structure. This is similar to the Level 1 structure, with the addition of the cbList field after the attrFile field.

The cbList field is an unsigned doubleword. On output, this field contains the size, in bytes, of the file's entire extended attribute (EA) set on disk. You can use this value to calculate the size of the buffer required to hold the EA information returned when a value of 3 is specified for *FileInfoLevel*. The buffer size is less than or equal to twice the size of the file's entire EA set on disk.

Level 3 File Information
On input, *FileInfoBuf* contains an EAOP2 data structure. fpGEA2List points to a GEA2 list defining the attribute names whose values are returned. The GEA2 data structures must be aligned on a doubleword boundary. Each oNextEntryOffset field must contain the number of bytes from the beginning of the current entry to the beginning of the next entry in the GEA2 list. The oNextEntryOffset field in the last entry of the GEA2 list must be zero. fpFEA2List points to a data area where the relevant FEA2 list is returned. The length field of this FEA2 list is valid, giving the size of the FEA2 list buffer. oError is ignored.

On output, *FileInfoBuf* is unchanged. The buffer pointed to by fpFEA2List is filled in with the returned information. If the buffer that fpFEA2List points to is not large enough to hold the returned information (the return code is ERROR_BUFFER_OVERFLOW), cbList is still valid, assuming there is at least enough space for it. Its value is the size of the entire EA set on disk for the file, even though only a subset of attributes was requested.

DosQueryFileInfo — Query File Information

ulFileInfoBufSize (ULONG) — input

The length, in bytes, of *FileInfoBuf*.

Returns

Return Code.

DosQueryFileInfo returns the following values:

0	NO_ERROR
5	ERROR_ACCESS_DENIED
6	ERROR_INVALID_HANDLE
111	ERROR_BUFFER_OVERFLOW
124	ERROR_INVALID_LEVEL
130	ERROR_DIRECT_ACCESS_HANDLE
254	ERROR_INVALID_EA_NAME
255	ERROR_EA_LIST_INCONSISTENT

Remarks

In the FAT file system, only date and time information contained in level-1 file information can be modified. Zero is returned for the creation and access dates and times.

To return information contained in any of the file information levels, DosQueryFileInfo must be able to read the open file. DosQueryFileInfo works only when the file is opened for read access, with a deny-write sharing mode specified for access by other processes. If another process that has specified conflicting sharing and access modes has already opened the file, any call to DosOpen will fail.

DosEnumAttribute returns the lengths of extended attributes. This information can be used to calculate what size *FileInfoBuf* needs to be to hold full-extended-attribute (FEA) information returned by DosQueryFileInfo when Level 3 is specified. The size of the buffer is calculated as follows:

Four bytes (for fea_oNextEntryOffset) +
One byte (for fea_usFlags) +
One byte (for fea_cbName) +
Two bytes (for fea_cbValue) +
Value of cbName (for the name of the EA) +
One byte (for terminating NULL in fea_cbName) +
Value of cbValue (for the value of the EA)

Related Functions

- DosClose
- DosEnumAttribute
- DosOpen
- DosQueryPathInfo
- DosResetBuffer
- DosSetFileInfo
- DosSetFileSize
- DosSetPathInfo

Example Code

This example obtains file information for a specified file. The example will obtain the Level 1 information set for the file. The Level 1 information set for a file includes the dates and times of creation, last access and last writing. It also includes information about the size of the file and the file's standard attributes. Assume that the handle of the desired file has been placed into *FileHandle* already.

```
#define INCL_DOSFILEMGR   /* File Manager values */
#include <os2.h>
#include <stdio.h>

HFILE        FileHandle;      /* File handle */
ULONG        FileInfoLevel;   /* Level of file info required */
FILESTATUS3  FileInfoBuf;     /* File info buffer */
ULONG        FileInfoBufSize; /* File data buffer size */
APIRET       rc;              /* Return code */

   FileInfoLevel = 1;    /* Indicate that Level 1 information */
                         /*   is desired                      */

   FileInfoBufSize = sizeof(FILESTATUS3);
                        /* Size of the buffer that will      */
                        /*   receive the Level 1 information */

   rc = DosQueryFileInfo(FileHandle, FileInfoLevel,
                           &FileInfoBuf, FileInfoBufSize);
                        /* On successful return, the Level 1 */
                        /*   file information is in the      */
                        /*   FileInfoBuf buffer              */

   if (rc != 0)
     {
       printf("DosQueryFileInfo error: return code = %ld", rc);
       return;
     }
```

DosQueryFSAttach – Query Attached File System

#define INCL_DOSFILEMGR

APIRET DosQueryFSAttach (PSZ pszDeviceName, ULONG ulOrdinal, ULONG ulFSAInfoLevel, PFSQBUFFER2 ppDataBuffer, PULONG pDataBufferLen)

DosQueryFSAttach obtains information about an attached file system (local or remote), or about a character device or pseudocharacter device attached to the file system.

Parameters

pszDeviceName (PSZ) – input

A drive designation or the name of a character or pseudocharacter device. If it is a drive designation, *DeviceName* is an ASCIIZ string consisting of a drive name followed by a colon. If it is a character or pseudocharacter device name, *DeviceName* is an ASCIIZ string consisting of a file name and the subdirectory, DEV. *DeviceName* is ignored if level 2 or 3 is specified for *FSAInfoLevel*.

ulOrdinal (ULONG) – input

An index into the list of character or pseudocharacter devices, or the set of drives. *Ordinal* always starts at 1. The ordinal position of an item in a list has no significance. *Ordinal* is used only to step through the list. The mapping from *Ordinal* to the item is volatile, and may change from one call to DosQueryFSAttach to the next. *Ordinal* is ignored if level 1 is specified for *FSAInfoLevel*.

ulFSAInfoLevel (ULONG) – input

Level of information returned in *DataBuffer*:

- Level 1 (FSAIL_QUERYNAME) returns data for the drive or device name specified in *DeviceName*. *Ordinal* is ignored.
- Level 2 (FSAIL_DEVNUMBER) returns data for the entry in the list of character or pseudocharacter devices selected by *Ordinal*. *DeviceName* is ignored.
- Level 3 (FSAIL_DRVNUMBER) returns data for the entry in the list of drives selected by *Ordinal*. *DeviceName* is ignored.

ppDataBuffer (PFSQBUFFER2) – output

Address of the buffer for returned information. The buffer has the following format:

iType (**USHORT**)
Type of item.

Value	Definition
1	(FSAT_CHARDEV) Resident character device
2	(FSAT_PSEUDODEV) Pseudocharacter device
3	(FSAT_LOCALDRV) Local drive
4	(FSAT_REMOTEDRV) Remote drive attached to the file-system driver.

cbName (**USHORT**)
Length, in bytes, of the item name, not counting null.

cbFSDName (**USHORT**)
Length, in bytes, of the file-system driver name, not counting null.

cbFSAData (**USHORT**)
Length, in bytes, of the file-system driver Attach data returned by the file-system driver.

szName (UCHAR)
Item name. The name is an ASCIIZ string.

szFSDName (UCHAR)
Name of the file-system driver that the item is attached to. The name is an ASCIIZ string.

rgFSAData (UCHAR)
File-system driver Attach data returned by the file-system driver.

Note:
The szFSDName is the file-system driver name exported by the file-system driver. This name is not necessarily the same as the file-system driver name in the boot sector.

For local character devices (iType = 1), cbFSDName = 0, and szFSDName contains only a terminating null byte; cbFSAData = 0.

For local drives (iType = 3), szFSDName contains the name of the file-system driver attached to the drive at the time of the call. This information changes dynamically. If the drive is attached to the kernel's resident file system, szFSDName contains FAT or an unknown name. Since the resident file system gets attached to any disk that other file-system drivers refuse to mount, it is possible to have a disk that does not contain a recognizable file system, but yet gets attached to the resident file system. In this case, it is possible to detect the difference, and this information would help programs to preserve data on a disk that was not properly recognized.

pDataBufferLen (PULONG) – input/output

On input, the address of the length, in bytes, of the return buffer (*DataBuffer*). On output, it is the length, in bytes, of the data returned in *DataBuffer* by the file-system driver.

Returns

Return Code.

DosQueryFSAttach returns the following values:

0	NO_ERROR
15	ERROR_INVALID_DRIVE
111	ERROR_BUFFER_OVERFLOW
124	ERROR_INVALID_LEVEL
259	ERROR_NO_MORE_ITEMS

Remarks

DosQueryFSAttach returns information about all block devices, and all character and pseudocharacter devices. The subject of the information returned by this call changes frequently. Therefore, the information that this function returns may no longer be valid when you receive it.

The information returned for disks attached to the resident file system of the kernel can be used to determine:

- If the kernel recognized the disk as one attached to its file system, or
- If the kernel attached its file system to the disk because no other file-system drivers were attached to the disk.

This information can be important for a program that needs to know what file system is attached to the drive. A situation could arise where the file-system driver that recognizes a certain disk has not been loaded into the system. There is then a potential for the data on the disk to be destroyed because the wrong file system gets attached to the disk by default.

DosQueryFSAttach – Query Attached File System

Related Functions

- DosFSAttach
- DosQuerySysInfo

Example Code

This example returns information about an attached file system.

```
#define INCL_DOSFILEMGR   /* File Manager values */
#include <os2.h>
#include <stdio.h>

UCHAR       DeviceName[8]; /* Device name or drive letter string */
ULONG       Ordinal;       /* Ordinal of entry in name list */
ULONG       FSAInfoLevel;  /* Type of attached FSD data required */
FSQBUFFER2  DataBuffer;    /* Returned data buffer */
ULONG       DataBufferLen; /* Buffer length */
APIRET      rc;            /* Return code */

   strcpy(DeviceName,"Y:");  /* Logical drive of attached */
                             /*   file system             */

   FSAInfoLevel = 1;         /* Indicate that the request is to  */
                             /*   return information about the   */
                             /*   drive whose name is specified  */
                             /*   within the DeviceName variable */
                             /*   (also indicate that the        */
                             /*   Ordinal variable is to be      */
                             /*   ignored)                       */

   DataBufferLen = sizeof(FSQBUFFER2);
                             /* Length of data buffer */

   rc = DosQueryFSAttach(DeviceName, Ordinal, FSAInfoLevel,
                           &DataBuffer, &DataBufferLen);

                             /* On successful return, the          */
                             /*   DataBuffer structure contains    */
                             /*   a set of information describing  */
                             /*   the specified attached file      */
                             /*   system, and the DataBufferLen    */
                             /*   variable contains the size of    */
                             /*   information within the structure */

   if (rc != 0)
     {
       printf("DosQueryFSAttach error: return code = %ld", rc);
       return;
     }
```

DosQueryFSInfo – Query File System Information

#define INCL_DOSFILEMGR

APIRET DosQueryFSInfo (ULONG ulDriveNumber, ULONG ulFSInfoLevel, PVOID pFSInfoBuf, ULONG ulFSInfoBufSize)

DosQueryFSInfo gets information from a file-system device.

Parameters

ulDriveNumber (ULONG) – input

Logical drive number for the disk about which information is to be retrieved. This parameter can be any value from 0 through 26. If this parameter is zero, information about the disk in the current drive is retrieved. Otherwise, 1 specifies drive A, 2 specifies drive B, and so on.

When a logical drive is specified, the media in the drive is examined (for a local drive only), and the request is passed to the file system driver (FSD) responsible for managing that media, or to the FSD that is attached to the drive.

ulFSInfoLevel (ULONG) – input

Level of file information required.

Value	Description
1	(FSIL_ALLOC) Level 1 information
2	(FSIL_VOLSER) Level 2 information

pFSInfoBuf (PVOID) – output

Address of the storage area where the system returns the requested level of file information.

Level 1 Information

When a value of 1 is specified for *FSInfoLevel*, the information is returned in the following format:

filesysid (ULONG)
File system ID.

sectornum (ULONG)
Number of sectors per allocation unit.

unitnum (ULONG)
Number of allocation units.

unitavail (ULONG)
Number of allocation units available.

bytesnum (USHORT)
Number of bytes per sector.

Level 2 Information

When a value of 2 is specified for *FSInfoLevel*, the information is returned in the following format:

VolumeSerialNum (ULONG)
Volume serial number.

volumelength (BYTE)
Length of the volume label, not including the null.

volumelabel (CHAR)
Volume label. This is an ASCIIZ string.

DosQueryFSInfo — Query File System Information

ulFSInfoBufSize (ULONG) — input

The length, in bytes, of the buffer that receives the file-system information.

Returns

Return Code.

DosQueryFSInfo returns the following values:

0	NO_ERROR
15	ERROR_INVALID_DRIVE
111	ERROR_BUFFER_OVERFLOW
124	ERROR_INVALID_LEVEL
125	ERROR_NO_VOLUME_LABEL

Remarks

DosQueryFSInfo gets information from a file-system device.

Trailing blanks supplied at the time the volume label is defined are not considered part of the label, and are not returned as valid label data. The volume label is limited to a length of 11 bytes.

Volume Serial Number is a unique 32-bit number that the operating system uses to identify its disk or diskette volumes. The hard error prompts the user for an unmounted removable volume by displaying both the Volume Serial Number (an 8-digit hexadecimal number) and the Volume Label.

If the disk or diskette has no volume serial number, the volume-serial-number information is returned as binary zeros. If there is no volume label, the volume-label information is returned as blank spaces.

If there is no volume serial number or volume label for disk or diskette volumes formatted by DOS, this information is not displayed by the Hard Error handler.

Related Functions

- DosSetFSInfo

Example Code

This example obtains information about the file system that is associated with a particular logical drive.

```
#define INCL_DOSFILEMGR   /* File Manager values */
#include <os2.h>
#include <stdio.h>

ULONG   DriveNumber;   /* Drive number */
ULONG   FSInfoLevel;   /* File system data required */
UCHAR   FSInfoBuf[40]; /* File system info buffer */
ULONG   FSInfoBufSize; /* File system info buffer size */
APIRET  rc;            /* Return code */

   DriveNumber = 3;    /* Specify drive C */

   FSInfoLevel = FSIL_ALLOC;
                       /* Indicate that file system allocation */
                       /*   information is requested           */

   FSInfoBufSize = 40; /* Size of return data buffer */

   rc = DosQueryFSInfo(DriveNumber, FSInfoLevel, FSInfoBuf,
                       FSInfoBufSize);
                       /* On successful return, the data buffer */
                       /*   FSInfoBuf contains a set of         */
                       /*   information about space allocation  */
                       /*   within the specified file system    */

   if (rc != 0)
     {
       printf("DosQueryFSInfo error: return code = %ld", rc);
       return;
     }
```

DosQueryHType – Query Handle Type

#define INCL_DOSFILEMGR

APIRET DosQueryHType (HFILE FileHandle, PULONG pHandleType, PULONG pFlagWord)

DosQueryHType determines whether a handle refers to a file or a device.

Parameters

FileHandle (HFILE) – input

File handle.

pHandleType (PULONG) – output

Address of the value indicating the handle type. *HandleType* consists of two bytes:

Bit	Description
15	Network bit: 0: The handle refers to a local file, device, or pipe. 1: The handle refers to a remote file, device, or pipe.
14–8	Reserved.
7–0	*HandleClass* describes the handle class. It may take on the following values in the low byte of *HandleType*:

Value	Definition
0	Disk file
1	Character device
2	Pipe.

Values greater than 2 are reserved.

pFlagWord (PULONG) – output

Address of the device-driver attribute word if *HandleType* indicates a local character device.

Returns

Return Code.

DosQueryHType returns the following values:

0	NO_ERROR
6	ERROR_INVALID_HANDLE

Remarks

DosQueryHType enables programs that are interactive or file-oriented to determine the source of their input. For example, CMD.EXE suppresses writing prompts if the input is from a disk file.

Related Functions

- DosOpen

Example Code

This example determines whether a given file handle refers to a file or a device. Assume that the desired file handle has been placed into *FileHandle* already.

```
#define INCL_DOSFILEMGR   /* File Manager values */
#include <os2.h>
#include <stdio.h>

HFILE    FileHandle;    /* File handle */
ULONG    HandType;      /* Handle type (returned) */
ULONG    FlagWord;      /* Device driver attribute (returned) */
APIRET   rc;            /* Return code */

   rc = DosQueryHType(FileHandle, &HandType, &FlagWord);
                       /* On successful return, the variable  */
                       /*   HandType contains a value that    */
                       /*   characterizes the type of file    */
                       /*   handle, and the variable FlagWord */
                       /*   contains the associated device    */
                       /*   driver attribute word if HandType */
                       /*   indicates that the file handle is */
                       /*   associated with a local character */
                       /*   device                            */

   if (rc != 0)
     {
       printf("DosQueryHType error: return code = %ld", rc);
       return;
     }
```

DosQueryMem –
Obtain Information about a Range of Pages

#define INCL_DOSMEMMGR

```
APIRET DosQueryMem  (PVOID pBaseAddress, PULONG pulRegionSize,
                     PULONG pulAllocationFlags)
```

DosQueryMem obtains information about a range of pages within the virtual-address space of the subject process.

Parameters

pBaseAddress (PVOID) – input

The base address of the range of pages to be queried.

pulRegionSize (PULONG) – input/output

On input, this parameter points to a variable that contains the size, in bytes, of the range of pages to be queried. The initial value of the variable is rounded to include all pages addressed by the requested base address and size. Upon return, this parameter points to a variable that contains the actual size, in bytes, of the queried range of pages.

pulAllocationFlags (PULONG) – output

A pointer to a variable that receives a set of attribute flags describing the type of allocation and access protection for the specified range of pages.

Allocation Type

- If the PAG_COMMIT bit (0x00000010) is set, the pages within the specified region are committed.
- If the PAG_FREE bit (0x00004000) is set, the pages within the specified region are free.
- If the PAG_SHARED bit (0x00002000) is set, the pages within the specified region are in a shared memory object. Otherwise, the pages are in a private memory object.
- If the PAG_BASE bit (0x00010000) is set, the first page in the specified region is the first page in an allocated memory object.

Access Protection

- If the PAG_EXECUTE bit (0x00000004) is set, execute access to the committed range of pages is allowed.
- If the PAG_READ bit (0x00000001) is set, read access to the committed range of pages is allowed.
- If the PAG_WRITE bit (0x00000002) is set, write access to the committed range of pages is allowed.
- If the PAG_GUARD bit (0x00000008) is set, access to the committed range of pages causes a "guard page entered" condition to be raised in the subject process.

Returns

Return Code.

DosQueryMem returns the following values:

0	NO_ERROR
87	ERROR_INVALID_PARAMETER
95	ERROR_INTERRUPT
487	ERROR_INVALID_ADDRESS

DosQueryMem – Obtain Information about a Range of Pages

Remarks

DosQueryMem provides the capability to determine the type and access protection of a range of pages within the virtual-address space of the subject process. This is the only memory-management function that accepts an address range that is not entirely contained within a previously allocated memory object.

The state of the first page within the region is determined, then subsequent entries in the virtual-address space of the process are scanned from the base address upward until either the entire range of pages has been scanned, a page with a nonmatching set of attributes is encountered, or the first page in an adjacent allocated memory object is encountered. The region attributes, the length of the range of pages with matching attributes, and an appropriate error code are returned.

If the entire requested range of pages does not have a matching set of attributes, then the returned *RegionSize* parameter value can be used to calculate the address and length of the range of pages that were not scanned.

Page scanning stops when the first page in an adjacent allocated memory object is encountered. This allows the calling application to determine the appearance of the virtual memory map, including object boundaries.

A region of pages that is neither committed nor free is considered reserved, that is, it is contained within an allocated memory object but has an access protection of "no access".

If the allocation type returned indicates that the pages are reserved, that is, neither PAG_COMMIT nor PAG_FREE is set, then the access protection returned is the same as was specified when the object was allocated in the address space of the requesting process.

With the Intel 80386 processor, execute and read access are equivalent. Also, write access implies both read and execute access.

Related Functions

- DosSetMem

DosQueryMem –
Obtain Information about a Range of Pages

Example Code

This example obtains information about a range of pages within the virtual address space of the calling process. Assume that the base address for the DosQueryMem function was previously obtained by the process.

```
#define  INCL_DOSMEMMGR   /* Memory Manager values */
#include <os2.h>
#include <stdio.h>
#include <bsememf.h>   /* Get flags for memory management */

PVOID    BaseAddress;     /* Pointer to the range of memory
                             whose attributes are being queried */

ULONG    RegionSize;      /* Long value that on input contains the
                             size, in bytes, of the region of
                             memory to be queried, and on output
                             contains the size, in bytes, of the
                             region that was queried */

ULONG    Flags;           /* Long value that on output will receive
                             a set of attribute flags that describe
                             the type of allocation and access
                             protection within the queried
                             region */

APIRET   rc;              /* Return code */

   RegionSize = 16384;    /* Ask to query a four-page region  */
                          /*   starting at the specified base */
                          /*   address                        */

   rc = DosQueryMem(BaseAddress, &RegionSize, &Flags);
                          /* On return, the RegionSize and    */
                          /*   Flags variables will be set to */
                          /*   values that characterize the   */
                          /*   queried region                 */

   if (rc != 0)
     {
       printf("DosQueryMem error: return code = %ld", rc);
       return;
     }
```

DosQueryMessageCp – Retrieve a Message File List of Code Pages and Language Identifiers

#define INCL_DOSMISC

```
APIRET DosQueryMessageCp  (PCHAR ppBufferArea, ULONG ulBufferLength,
                           PSZ pszFileName, PULONG pDataLength)
```

DosQueryMessageCp retrieves a message file list of code pages and language identifiers.

Parameters

ppBufferArea (PCHAR) – output

Pointer to the caller's buffer area, where the system returns the requested message file list of code pages and language identifiers.

ulBufferLength (ULONG) – input

The length, in bytes, of *BufferArea*.

pszFileName (PSZ) – input

The drive designation, path, and name of the message file. The drive designation and path are optional. This specifies a file that was previously prepared by the MKMSGF utility program.

pDataLength (PULONG) – output

Pointer to the doubleword that receives the actual length, in bytes, of the returned data.

Returns

Return Code.

DosQueryMessageCp returns the following values:

0	NO_ERROR
2	ERROR_FILE_NOT_FOUND
206	ERROR_FILENAME_EXCED_RANGE
318	ERROR_MR_UN_ACC_MSGF
319	ERROR_MR_INV_MSGF_FORMAT
321	ERROR_MR_UN_PERFORM

Remarks

DosQueryMessageCp retrieves the message file list of code pages and language identifiers.

The system returns the requested message file list of code pages and language identifiers in the caller's buffer (*BufferArea*). It has the following format:

Length	Field
WORD	Code page count (N)
WORD	Code page identifier. This field occurs N times, once per code page.
DWORD	Language identifier

This data structure is repeated for each subfile within the specified message file.

DosQueryMessageCp – Retrieve a Message File List of Code Pages and Language Identifiers

The code page identifier can have the following values:

Value	Description
437	United States
850	Multilingual
852	Latin 2 (Czechoslovakia, Hungary, Poland, Yugoslavia)
857	Turkish
860	Portuguese
861	Iceland
863	Canadian French
865	Nordic
932	Japan
934	Korea
936	People's Republic of China
938	Taiwan
942	Japan SAA
944	Korea SAA
946	People's Republic of China SAA
948	Taiwan SAA

Note: Code pages 932, 934, 936, 938, 942, 944, 946, and 948 are supported only with the Asian version of the operating system on Asian hardware.

The language identifier is a doubleword. The low-order word identifies a language family, and the high-order word identifies a specific version of that language (a sublanguage).

The language and sublanguage identifier values 0 through 255 are reserved for system use. The values 256 through 511 are reserved for application use.

The MKMSGF utility program performs range checking on the language and sublanguage identifier values. The value 0 means a null or unspecified language or sublanguage. Only the values defined in the following table are valid below 256. Any values from 256 through 511 are valid. Any identifier value greater than 511 is invalid.

The following table shows the valid language and sublanguage identifier values. Column 1 is the language family identifier, and column 2 is the sublanguage identifier. Column 3 shows the language, and column 4 shows the principal country for this language.

Table 2-2 (Page 1 of 3). Language identifiers

Family	Subl.	Language	Principal country
0	0	null	null
1	1	Arabic	Arab countries
2	1	Bulgarian	Bulgaria
3	2	Spanish	Spain
3	3	Spanish Mexican	Mexico
4	1	Traditional Chinese	Republic of China

Table 2-2 (Page 2 of 3). Language identifiers			
Family	**Subl.**	**Language**	**Principal country**
4	2	Simplified Chinese	People's Republic of China
5	1	Czech	Czechoslovakia
6	1	Danish	Denmark
7	1	German	Germany
7	2	Swiss German	Switzerland
8	1	Greek	Greece
9	1	U.K. English	United Kingdom
9	2	U.S. English	United States
10	1	Finnish	Finland
11	1	French	France
11	2	Belgian French	Belgium
11	3	Canadian French	Canada
11	4	Swiss French	Switzerland
12	1	Hebrew	Israel
13	1	Hungarian	Hungary
14	1	Icelandic	Iceland
15	1	Italian	Italy
15	2	Swiss Italian	Switzerland
16	1	Japanese	Japan
17	1	Korean	Korea
18	1	Dutch	Netherlands
18	2	Belgian Dutch	Belgium
19	1	Norwegian - Bokmal	Norway
19	2	Norwegian - Nynorsk	Norway
20	1	Polish	Poland
21	1	Portuguese	Portugal
22	2	Brazilian Portuguese	Brazil
23	1	Rhaeto-Romanic	Switzerland
24	1	Serbo-Croatian (Cyrillic)	Yugoslavia
24	2	Serbo-Croatian (Latin)	Yugoslavia
25	1	Slovakian	Czechoslovakia
26	1	Albanian	Albania
27	1	Swedish	Sweden
28	1	Thai	Thailand
29	1	Turkish	Turkey

DosQueryMessageCp – Retrieve a Message File List of Code Pages and Language Identifiers

Table 2-2 (Page 3 of 3). Language identifiers			
Family	**Subl.**	**Language**	**Principal country**
30	1	Urdu	Pakistan
31	1	Russian	U.S.S.R.
32	1	Catalan	Spain

Related Functions

- DosGetMessage
- DosInsertMessage
- DosPutMessage

Example Code

This example obtains a list of code-page identifiers and language identifiers that are associated with a specified message file. Assume that the path name of the relevant message file is "D:\MESSAGE\AUTOMSG.MSG".

```
#define INCL_DOSMISC   /* Miscellaneous values */
#include <os2.h>
#include <stdio.h>

UCHAR    BufferArea[20];  /* Buffer for the returned list */
ULONG    BufferLength;    /* Length of the buffer area */
UCHAR    FileName[40];    /* Message file path-name string */
ULONG    DataLength;      /* Length of the returned data */
APIRET   rc;              /* Return code */

   strcpy(FileName,"D:\\MESSAGE\\AUTOMSG.MSG";
                         /* Path name of the message file */

   BufferLength = 20;     /* Length of the buffer area (bytes) */

   rc = DosQueryMessageCp(BufferArea, BufferLength, FileName,
                          &DataLength);

                         /* On successful return, the       */
                         /*   BufferArea buffer contains a  */
                         /*   set of information concerning */
                         /*   the code-page identifiers and */
                         /*   language identifiers that are */
                         /*   associated with the message   */
                         /*   file                          */

   if (rc != 0)
     {
       printf("DosQueryMessageCp error: return code = %ld", rc);
       return;
     }
```

DosQueryModuleHandle – Return the Handle of a Dynamic Link Module Previouly Loaded

---

#define INCL_DOSMODULEMGR

APIRET DosQueryModuleHandle (PSZ pszModName, PHMODULE ppModHandle)

DosQueryModuleHandle returns the handle of a dynamic link module that was previously loaded.

Parameters

pszModName (PSZ) – input

The address of an ASCIIZ name string containing the dynamic link module name. The file-name extension used for dynamic link libraries is .DLL.

ppModHandle (PHMODULE) – output

The address of a doubleword in which the handle for the dynamic link module is returned.

Returns

Return Code.

DosQueryModuleHandle returns the following values:

0	NO_ERROR
123	ERROR_INVALID_NAME

Remarks

DosQueryModuleHandle returns the handle of a dynamic link module that was previously loaded.

The module name must match the name of the module already loaded. Otherwise, an error code is returned. This is a way of testing whether a dynamic link module is already loaded.

Related Functions

- DosFreeModule
- DosLoadModule
- DosQueryModuleName

DosQueryModuleHandle – Return the Handle of a Dynamic Link Module Previouly Loaded

Example Code

This example attempts obtain the handle of a dynamic link module. This allows the caller to test whether a given dynamic link module is currently loaded.

```
#define INCL_DOSMODULEMGR   /* Module Manager values */
#include <os2.h>
#include <stdio.h>

UCHAR      ModuleName[200];  /* Module name string */
HMODULE    ModuleHandle;     /* Module handle (returned) */
APIRET     rc;               /* Return code */

   strcpy(ModuleName,"C:\\OS2\\DLL\\PMREXX.DLL");
                             /* See if the PMREXX module is */
                             /*   loaded                    */

   rc = DosQueryModuleHandle(ModuleName, &ModuleHandle);
                             /* On successful return, the    */
                             /*   ModuleHandle variable      */
                             /*   contains the handle of the */
                             /*   module                     */

   if (rc != 0)
     {
       printf("DosQueryModuleHandle error: return code = %ld", rc);
       return;
     }
```

DosQueryModuleName – Return Fully Qualified Name with Referenced Module Handle

#define INCL_DOSMODULEMGR

```
APIRET DosQueryModuleName (HMODULE hmodModHandle, ULONG ulBufferLength,
                           PCHAR ppNameBuffer)
```

DosQueryModuleName returns the fully qualified drive, path, file name, and extension associated with the referenced module handle.

Parameters

hmodModHandle (HMODULE) – input

The handle of the dynamic link module that is being referenced. This handle is provided in DI on entry to a module, or on the initialization entry to a dynamic link routine.

ulBufferLength (ULONG) – input

The maximum length of the buffer, in bytes, where the name will be stored.

ppNameBuffer (PCHAR) – output

The address of the buffer where the fully qualified drive, path, file name, and extension of the module are returned.

Returns

Return Code.

DosQueryModuleName returns the following values:

0	NO_ERROR
6	ERROR_INVALID_HANDLE
24	ERROR_BAD_LENGTH

Remarks

DosQueryModuleName returns the fully qualified drive, path, file name, and extension associated with the referenced module handle.

If the buffer is not large enough, an error is returned.

Related Functions

- DosFreeModule
- DosLoadModule
- DosQueryModuleHandle

DosQueryModuleName – Return Fully Qualified Name with Referenced Module Handle

Example Code

Given a dynamic link module handle, this example obtains the fully qualified drive, path, file name, and extension associated with the module. Assume that the appropriate module handle has been placed into *ModuleHandle* already.

```
#define INCL_DOSMODULEMGR   /* Module Manager values */
#include <os2.h>
#include <stdio.h>

HMODULE   ModuleHandle;  /* Module handle */
ULONG     BufferLength;  /* Buffer length */
UCHAR     Buffer[256];   /* Buffer (returned) */
APIRET    rc;            /* Return code */

   BufferLength = 256;   /* Length of return buffer */

   rc = DosQueryModuleName(ModuleHandle, BufferLength, Buffer);
                         /* On successful return, the buffer   */
                         /*   named Buffer will contain the    */
                         /*   fully qualified path name of the */
                         /*   specified module                 */

   if (rc != 0)
     {
       printf("DosQueryModuleName error: return code = %ld", rc);
       return;
     }
```

DosQueryMutexSem – Query Mutex Semaphore

#define INCL_DOSSEMAPHORES

```
APIRET DosQueryMutexSem (HMTX hmtx, PPID ppppidOwner, PTID ppptidOwner,
                         PULONG ppulCount)
```

DosQueryMutexSem retrieves information about the owner of a mutex semaphore.

Parameters

hmtx (HMTX) – input

The handle of the mutex semaphore to query.

ppppidOwner (PPID) – output

A pointer to the process ID of either the current owner of the mutex semaphore, or a process that has ended without releasing the semaphore.

ppptidOwner (PTID) – output

A pointer to the thread ID of either the current owner of the mutex semaphore, or a process that has ended without releasing the semaphore.

ppulCount (PULONG) – output

A pointer to the request count for the semaphore. The request count is the number of calls to DosRequestMutexSem, minus the number of calls to DosReleaseMutexSem, that have been made for the semaphore by the owning thread. If the semaphore is unowned, this value will be zero. If the owning thread has ended, the value will be the request count for the ended owner.

Returns

Return Code.

DosQueryMutexSem returns the following values:

0	NO_ERROR
6	ERROR_INVALID_HANDLE
87	ERROR_INVALID_PARAMETER
105	ERROR_SEM_OWNER_DIED

Remarks

DosQueryMutexSem returns the process identification (PID) and thread identification (TID) of a mutex semaphore's current owner, as well as the request count for the semaphore. The request count is the number of calls to DosRequestMutexSem, minus the number of calls to DosReleaseMutexSem, that have been made for the semaphore by the owning thread.

This function can be called by any thread in the process that created the semaphore. Threads in other processes can also call this function, but they must first gain access to the semaphore by calling DosOpenMutexSem.

Related Functions

- DosCloseMutexSem
- DosCreateMutexSem
- DosOpenMutexSem
- DosReleaseMutexSem
- DosRequestMutexSem

DosQueryMutexSem – Query Mutex Semaphore

Example Code

This example obtains information about a mutex semaphore. Assume that the handle of the semaphore has been placed into *hmtx* already.

```
#define INCL_DOSSEMAPHORES   /* Semaphore values */
#include <os2.h>
#include <stdio.h>

HMTX    hmtx;       /* Mutex semaphore handle */
PID     pidOwner;   /* PID of current mutex semaphore owner
                       (returned) */
TID     tidOwner;   /* TID of current mutex semaphore owner
                       (returned) */
ULONG   ulCount;    /* Request count for the semaphore
                       (returned) */
APIRET  rc;         /* Return code */

   rc = DosQueryMutexSem(hmtx, &pidOwner, &tidOwner,
                                &ulCount);
                  /* On successful return, the pidOwner */
                  /*   variable contains the PID of the */
                  /*   semaphore's owner, the tidOwner  */
                  /*   variable contains the TID of the */
                  /*   semaphore's owner, the ulCount   */
                  /*   variable contains the request    */
                  /*   count for the semaphore's owner  */

   if (rc != 0)
     {
       printf("DosQueryMutexSem error: return code = %ld", rc);
       return;
     }
```

DosQueryMuxWaitSem – Query MuxWait Semaphore

#define INCL_DOSSEMAPHORES

```
APIRET DosQueryMuxWaitSem (HMUX hmux, PULONG ppcSemRec,
                           PSEMRECORD pppSemRec, PULONG ppflAttr)
```

DosQueryMuxWaitSem retrieves the semaphore records from a muxwait-semaphore list.

Parameters

hmux (HMUX) – input

The handle of the muxwait semaphore to query.

ppcSemRec (PULONG) – input/output

On input, a pointer to the maximum number of semaphore record entries that can be contained in the list pointed to by *pSemRec*. On output, a pointer to the number of semaphore record entries returned in the list pointed to by *pSemRec*. If the list pointed to by *pSemRec* is not large enough to hold all of the semaphore records in the specified muxwait semaphore, the system sets the value pointed to by *pSemRec* to the number of semaphore records in the specified muxwait semaphore, and sets the return code to ERROR_PARAM_TOO_SMALL.

pppSemRec (PSEMRECORD) – output

A pointer to the semaphore record entries in the muxwait-semaphore list. This is the list of event or mutex semaphores that must be posted or released in order for the muxwait semaphore to be cleared.

ppflAttr (PULONG) – output

The *flAttr* attribute flags that were passed by DosCreateMuxWaitSem, as follows:

- If the DC_SEM_SHARED bit is set, the semaphore is shared.
- If the DCMW_WAIT_ANY bit is set, the semaphore waits for any event semaphore in the muxwait-semaphore list to be posted, or for any mutex semaphore in the list to be released. When any one of the semaphores is cleared, the thread that is waiting on the muxwait semaphore can continue executing.
- If the DCMW_WAIT_ALL bit is set, the semaphore waits for all of the event semaphores in the muxwait list to be posted, or for all of the mutex semaphores in the list to be released. When all of the semaphores are cleared, the thread that is waiting on the muxwait semaphore can continue executing.

Returns

Return Code.

DosQueryMuxWaitSem returns the following values:

0	NO_ERROR
6	ERROR_INVALID_HANDLE
8	ERROR_NOT_ENOUGH_MEMORY
87	ERROR_INVALID_PARAMETER
105	ERROR_SEM_OWNER_DIED
289	ERROR_PARAM_TOO_SMALL

DosQueryMuxWaitSem – Query MuxWait Semaphore

Remarks

DosQueryMuxWaitSem retrieves the semaphore records from a muxwait-semaphore list.

The process must have previously opened the muxwait semaphore by issuing DosCreateMuxWaitSem or DosOpenMuxWaitSem. If the muxwait semaphore does not exist, then the system returns the ERROR_INVALID_HANDLE return code to the caller.

The value that *pcSemRec* points to on input must be the maximum number of semaphore record entries that can be contained in the list pointed to by *pSemRec*. For example, if the list pointed to by *pSemRec* can contain ten semaphore record entries, then you should set the input value pointed to by *pcSemRec* to ten before issuing DosQueryMuxWaitSem.

If the list pointed to by *pSemRec* is not large enough to hold all of the semaphore records in the specified muxwait semaphore, then the system sets the value pointed to by *pcSemRec* to the number of semaphore records in the specified muxwait semaphore, and sets the return code to ERROR_PARAM_TOO_SMALL. This allows you to issue DosQueryMuxWaitSem again, with the correct amount of memory for the muxwait-semaphore list.

The system returns the ERROR_SEM_OWNER_DIED return code if any of the mutex semaphores in the muxwait semaphore have been placed into the owner-died state. This means that a thread ended while it owned at least one mutex semaphore, and at least one mutex semaphore is part of the muxwait semaphore. It also means that the mutex semaphore has not yet been removed by DosCloseMutexSem.

When the system returns the ERROR_SEM_OWNER_DIED return code, you should issue DosQueryMutexSem for each mutex semaphore in the muxwait-semaphore list to determine which ones are invalid. For each mutex semaphore that results in the ERROR_SEM_OWNER_DIED return code from DosQueryMutexSem, issue DosCloseMutexSem to close the mutex semaphore. Semaphore handles may be used again, so the mutex semaphores that are closed *must* be deleted from the muxwait semaphore.

Related Functions

- DosAddMuxWaitSem
- DosCloseMuxWaitSem
- DosCreateMuxWaitSem
- DosDeleteMuxWaitSem
- DosOpenMuxWaitSem
- DosWaitMuxWaitSem

Example Code

This example retrieves the semaphore records from a muxwait semaphore. Assume that the handle of the semaphore has been placed into *hmux* already.

```
#define INCL_DOSSEMAPHORES   /* Semaphore values */
#include <os2.h>
#include <stdio.h>

HMUX        hmux;       /* Muxwait semaphore handle */
ULONG       cSemRec;    /* Number of entries in SemRec */
SEMRECORD   SemRec[5];  /* List of mutex or event semaphores
                           that comprise the muxwait semaphore
                           (returned) */
ULONG       flAttr;     /* Muxwait semaphore creation
                           attributes (returned) */
APIRET      rc;         /* Return code */

   cSemRec = 5;         /* SemRec has room for 5 entries */

   rc = DosQueryMuxWaitSem(hmux, &cSemRec, SemRec, &flAttr);
                        /* On successful return, the SemRec     */
                        /*   buffer contains a list of          */
                        /*   structures that define each of     */
                        /*   the semaphores that constitute     */
                        /*   the MuxWait semaphore, and the     */
                        /*   flAttr variable contains the       */
                        /*   attribute flags that were passed   */
                        /*   in through the DosCreateMuxWaitSem */
                        /*   function that created the muxwait  */
                        /*   semaphore.                         */

   /* If the SemRec buffer is not large enough to              */
   /*   contain all of the constituent semaphores of the       */
   /*   muxwait semaphore, then an error is returned to the    */
   /*   caller, and cSemRec contains the number of             */
   /*   constituent semaphores that comprise the muxwait       */
   /*   semaphore.                                             */

   if (rc != 0)
     {
       printf("DosQueryMuxWaitSem error: return code = %ld", rc);
       return;
     }
```

DosQueryNPHState – Query Named Pipe Handle State

#define INCL_DOSNMPIPES

APIRET DosQueryNPHState (HPIPE hpipeHandle, PULONG pPipeHandleState)

DosQueryNPHState returns information about a named-pipe handle.

Parameters

hpipeHandle (HPIPE) – input

The named-pipe handle to query. (The server handle is returned by DosCreateNPipe; the client handle is returned by DosOpen.)

pPipeHandleState (PULONG) – output

A pointer to the named-pipe handle state. This parameter contains the following bit fields:

Bit	Description
31–16	Reserved.
15	Blocking mode. Blocking mode is defined as either "blocking" or "nonblocking," as follows: 0 = (NP_WAIT) (0x0000) Blocking mode: DosRead and DosWrite block if no data is available. 1 = (NP_NOWAIT) (0x8000) Nonblocking mode: DosRead and DosWrite return immediately if no data is available. DosRead normally blocks until at least partial data can be returned. DosWrite blocks by default until all of the requested bytes have been written. Nonblocking mode changes this behavior as follows: DosRead returns immediately with a value of zero for *BytesRead* if no data is available. DosWrite returns immediately with a value of zero for *BytesWritten* if there is not enough buffer space available in the pipe; otherwise, the entire data area is transferred.
14	Specifies whether the handle is for the server or client end of the pipe, as follows: 0 = (NP_END_CLIENT) (0x0000) The handle is for the client end of the pipe. 1 = (NP_END_SERVER) (0x4000) The handle is for the server end of the pipe.
13–12	Reserved.
11–10	Type of named pipe. The pipe type is defined as follows: 00 = (NP_TYPE_BYTE) (0x0000) The pipe is a byte pipe; that is, data is written to the pipe as an undifferentiated stream of bytes. 01 = (NP_TYPE_MESSAGE) (0x0400) The pipe is a message pipe; that is, data is written to the pipe as messages. The system records the length of each message in the first two bytes of the message, which are called the *message header*.
9–8	Read mode. The read mode is defined as follows: 00 = (NP_READMODE_BYTE) (0x0000) Byte-read mode: Read the pipe as a byte stream. 01 = (NP_READMODE_MESSAGE) (0x0100) Message-read mode: Read the pipe as a message stream.

7–0 Instance count: When the first instance of a named pipe is created, this field specifies how many instances (including the first instance) can be created. Possible values are:

Value	Definition
1	This is the only instance permitted (the pipe is unique).
1 < value < 255	The number of instances is limited to the value specified.
–1	(NP_UNLIMITED_INSTANCES) (0x00FF) The number of instances is unlimited.
0	Reserved value.

Returns

Return Code.

DosQueryNPHState returns the following values:

0	NO_ERROR
230	ERROR_BAD_PIPE
233	ERROR_PIPE_NOT_CONNECTED

Remarks

DosQueryNPHState returns the following information about a pipe handle and the attributes of the pipe:

- The end of the pipe that the handle is for (server or client end)
- The pipe type (byte pipe or message pipe)
- The instance count
- The blocking mode (blocking or nonblocking)
- The read mode (byte-read mode or message-read mode).

The values for the pipe type and instance count cannot be changed, so they are always the same as those that were specified when the pipe was created with DosCreateNPipe. The information returned for blocking mode and read mode, however, can come from different sources:

- If the handle is for the server end of the pipe, then the blocking mode and the read mode were set with DosCreateNPipe, but may have been reset with DosSetNPHState.
- If the handle is for the client end of the pipe, then the blocking mode and the read mode were set to "blocking" and "byte-read" by the system when the client issued DosOpen. However, they may have been reset with DosSetNPHState.

Related Functions

- DosCallNPipe
- DosConnectNPipe
- DosCreateNPipe
- DosDisConnectNPipe
- DosPeekNPipe
- DosQueryNPipeInfo
- DosQueryNPipeSemState
- DosSetNPHState
- DosSetNPipeSem
- DosTransactNPipe
- DosWaitNPipe
- DosClose
- DosDupHandle
- DosOpen

DosQueryNPHState – Query Named Pipe Handle State

- DosRead
- DosResetBuffer
- DosWrite

Example Code

This example returns information relating to the nature of a named pipe. Assume that a previous call to DosOpen or DosCreateNPipe provided the named pipe handle that is contained in *Handle*.

```
#define INCL_DOSNMPIPES   /* Named-pipe values */
#include <os2.h>
#include <stdio.h>

HPIPE    Handle;            /* Pipe handle */
ULONG    PipeHandleState;   /* Pipe-handle state */
APIRET   rc;                /* Return code */

   rc = DosQueryNPHState(Handle, &PipeHandleState);
                         /* On successful return, the variable */
                         /*   PipeHandleState will contain     */
                         /*   information that describes the   */
                         /*   nature of the named pipe         */

   if (rc != 0)
     {
       printf("DosQueryNPHState error: return code = %ld", rc);
       return;
     }
```

DosQueryNPipeInfo — Query Named Pipe Information

#define INCL_DOSNMPIPES

```
APIRET DosQueryNPipeInfo (HPIPE hpipeHandle, ULONG ulInfoLevel, PVOID pInfoBuf,
                          ULONG ulInfoBufSize)
```

DosQueryNPipeInfo returns information about a named pipe.

Parameters

hpipeHandle (HPIPE) — input

The named-pipe handle to query. (The server handle is returned by DosCreateNPipe; the client handle is returned by DosOpen).

ulInfoLevel (ULONG) — input

Level of the required pipe information. Only levels 1 and 2 are supported.

pInfoBuf (PVOID) — output

A pointer to the storage area in which the requested level of named-pipe information is returned.

- When *InfoLevel* is equal to 1, information about the pipe itself is returned in the following format:

  outbufsize (**USHORT**)
  The actual size of the buffer for outbound data.

  inbufsize (**USHORT**)
  The actual size of the buffer for inbound data.

  maxnuminstances (**UCHAR**)
  The maximum number of pipe instances.

  numinstances (**UCHAR**)
  The current number of pipe instances.

  namelength (**UCHAR**)
  The length of the pipe name.

  pipename (**CHAR**)
  The name of the pipe (including \\ComputerName if the pipe is on a remote system).

- When *InfoLevel* is equal to 2, the buffer will contain a unique 2-byte identifier of the client.

ulInfoBufSize (ULONG) — input

The length, in bytes, of *InfoBuf*.

Returns

Return Code.

DosQueryNPipeInfo returns the following values:

0	NO_ERROR
111	ERROR_BUFFER_OVERFLOW
124	ERROR_INVALID_LEVEL
230	ERROR_BAD_PIPE

DosQueryNPipeInfo – Query Named Pipe Information

Remarks

DosQueryNPipeInfo returns all of the level-1 or level-2 information about a named pipe that will fit in the *InfoBuf* storage area.

If the length of the pipe name is greater than 255 bytes, then a length of 0 is returned in the *namelength* field. However, the full ASCIIZ name is still returned in the *pipename* field.

If there is more information than will fit in *InfoBuf*, ERROR_BUFFER_OVERFLOW is returned.

Related Functions

- DosCallNPipe
- DosConnectNPipe
- DosCreateNPipe
- DosDisConnectNPipe
- DosPeekNPipe
- DosQueryNPHState
- DosQueryNPipeSemState
- DosSetNPHState
- DosSetNPipeSem
- DosTransactNPipe
- DosWaitNPipe
- DosClose
- DosDupHandle
- DosOpen
- DosRead
- DosResetBuffer
- DosWrite

Example Code

This example returns information relating to the nature and current state of a named pipe. Assume that a previous call to DosOpen or DosCreateNPipe provided the named pipe handle that is contained in *Handle*.

```
#define INCL_DOSNMPIPES   /* Named-pipe values */
#include <os2.h>
#include <stdio.h>

HPIPE    Handle;        /* Pipe handle */
ULONG    InfoLevel;     /* Pipe data required */
PIPEINFO InfoBuf;       /* Pipe information data structure */
ULONG    InfoBufSize;   /* Pipe data-buffer size */
APIRET   rc;            /* Return code */

   InfoLevel = 1;       /* Ask for standard level of pipe */
                        /*   information                  */

   InfoBufSize = sizeof(PIPEINFO);
                        /* Length of pipe information data */
                        /*   structure                     */

   rc = DosQueryNPipeInfo(Handle, InfoLevel, &InfoBuf, InfoBufSize);
                       /* On successful return, the pipe        */
                       /*   information data structure contains */
                       /*   a set of information describing the */
                       /*   nature and the current state of the */
                       /*   named pipe                          */

   if (rc != 0)
     {
       printf("DosQueryNPipeInfo error: return code = %ld", rc);
       return;
     }
```

DosQueryNPipeSemState – Query Named Pipe Operations

#define INCL_DOSNMPIPES

APIRET DosQueryNPipeSemState (HSEM hsemSemHandle, PPIPESEMSTATE ppInfoBuf, ULONG ulInfoBufLen)

DosQueryNPipeSemState returns information about local named pipes that are attached to a semaphore.

Parameters

hsemSemHandle (HSEM) – input

The handle of a shared event or muxwait semaphore that was previously attached to one or more named pipes with DosSetNPipeSem.

ppInfoBuf (PPIPESEMSTATE) – output

A pointer to a buffer containing a record for each named pipe that is attached to the semaphore. Each record contains the following fields:

fStatus (**BYTE**)
A coded value that indicates the status of the named pipe:

Value	Definition
0	(NPSS_EOI) End of information buffer. No more information records follow, and subsequent fields in this information record have no defined value.
1	(NPSS_RDATA) Read data is available.
2	(NPSS_WSPACE) Write space is available.
3	(NPSS_CLOSE) The pipe is closed.

fFlag (**BYTE**)
A bit field that indicates additional information about the state of the named pipe:

Bit	Description
7–1	Reserved
0	(NPSS_WAIT) If set, a thread is waiting at the other end of the pipe.

usKey (**USHORT**)
The value specified for *KeyHandle* when DosSetNPipeSem was issued.

usAvail (**USHORT**)
If **fStatus** has a value of 1, this field contains the number of bytes of data that are available to read from the pipe. If **fStatus** has a value of 2, this field contains the number of bytes of write space that are available in the pipe.

ulInfoBufLen (ULONG) – input

The size, in bytes, of *InfoBuf*.

Returns

Return Code.

DosQueryNPipeSemState returns the following values:

0	NO_ERROR
87	ERROR_INVALID_PARAMETER
111	ERROR_BUFFER_OVERFLOW

Remarks

DosQueryNPipeSemState returns information about the status of local named pipes that are attached to a shared event or multiple-wait (muxwait) semaphore. (Event semaphores are attached to local named pipes by calling DosSetNPipeSem.)

A record is returned for each local named pipe that is attached to the specified semaphore and whose state is either closed or allows blocking-mode input and output to be done. However, there is no guarantee that the records in the buffer refer only to named pipes that were opened by the process making this call. If the same semaphore has been attached to different named pipes by multiple processes, information about named pipes that are not accessible to the caller can be returned. For this reason, communicating processes should have a convention for key values to help identify the named pipes of interest. (A key value is specified when DosSetNPipeSem is called to attach the semaphore to a named pipe.)

If a process wants data in the buffer to refer only to its own named pipes, it must use a private event semaphore.

Related Functions

- DosCallNPipe
- DosConnectNPipe
- DosCreateNPipe
- DosDisConnectNPipe
- DosPeekNPipe
- DosQueryNPHState
- DosQueryNPipeInfo
- DosSetNPHState
- DosSetNPipeSem
- DosTransactNPipe
- DosWaitNPipe
- DosClose
- DosDupHandle
- DosOpen
- DosRead
- DosResetBuffer
- DosWrite

DosQueryNPipeSemState — Query Named Pipe Operations

Example Code

This example returns information about all named pipes in blocking mode on the local computer that are associated with a shared event semaphore. The semaphore handle used in the call to DosQueryNPipeSemState was previously associated with the named pipes through the use of DosSetNPipeSem.

```
#define INCL_DOSNMPIPES   /* Named-pipe values */
#include <os2.h>
#include <stdio.h>

HSEM  SemHandle;    /* Semaphore handle */
UCHAR InfoBuf[200]; /* Buffer that will contain an array */
                    /*   of Named Pipe semaphore state   */
                    /*   information block structures    */
ULONG  InfoBufLen;  /* Length of InfoBuf */
APIRET rc;          /* Return code */

   InfoBufLen = 200; /* Total buffer length */

   rc = DosQueryNPipeSemState(SemHandle, (PVOID) &InfoBuf,
                              InfoBufLen);
                    /* On successful return, the buffer will   */
                    /*   contain an array of named pipe        */
                    /*   semaphore state information block     */
                    /*   structures that contain information   */
                    /*   concerning the states of the various  */
                    /*   named pipes that have been associated */
                    /*   with the specified system semaphore   */

   if (rc != 0)
     {
       printf("DosQueryNPipeSemState error: return code = %ld",rc);
       return;
     }
```

DosQueryPathInfo – Query Path Information

#define INCL_DOSFILEMGR

APIRET DosQueryPathInfo (PSZ pszPathName, ULONG ulPathInfoLevel, PVOID pPathInfoBuf, ULONG ulPathInfoBufSize)

DosQueryPathInfo gets file information for a file or subdirectory.

Parameters

pszPathName (PSZ) – input

Address of the ASCIIZ full path name of the file or subdirectory. Global file-name characters can be used in the name only for level 5 file information.

DosQuerySysInfo is called by an application during initialization to determine the maximum path length allowed by the operating system.

ulPathInfoLevel (ULONG) – input

The level of path information required. A value of 1, 2, 3, or 5 can be specified, as follows:

Value	Description
1	(FIL_STANDARD) Level 1 file information
2	(FIL_QUERYEASIZE) Level 2 file information
3	(FIL_QUERYEASFROMLIST) Level 3 file information
5	(FIL_QUERYFULLNAME) Level 5 file information

Level 4 is reserved.

The structures described in *PathInfoBuf* indicate the information returned for each of these levels.

pPathInfoBuf (PVOID) – output

Address of the storage area containing the requested level of path information. Path information, where applicable, is based on the most recent DosClose, DosResetBuffer, DosSetFileInfo, or DosSetPathInfo.

Level 1 File Information
PathInfoBuf contains the FILESTATUS3 data structure, in which path information is returned.

Level 2 File Information
PathInfoBuf contains the FILESTATUS4 data structure. This is similar to the Level 1 structure, with the addition of the cbList field after the attrFile field.

The cbList field is an unsigned doubleword. On output, this field contains the size, in bytes, of the file's entire extended attribute (EA) set on disk. You can use this value to calculate the size of the buffer required to hold the EA information returned when a value of 3 is specified for *PathInfoLevel*. The buffer size is less than or equal to twice the size of the file's entire EA set on disk.

Level 3 File Information
This is a subset of the EA information of the file.

On input, *PathInfoBuf* contains an EAOP2 data structure. fpGEA2List points to a GEA2 that defines the attribute names whose values are returned. The GEA2 data structures must be aligned on a doubleword boundary. Each oNextEntryOffset field must contain the number of bytes from the beginning of the current entry to the beginning of the next entry in the GEA2 list. The oNextEntryOffset field in the last entry of the GEA2 list must be zero. fpFEA2List points to a data area where the relevant FEA2 list is returned. The length field of this FEA2 list is valid, giving the size of the FEA2 list buffer. oError is ignored.

DosQueryPathInfo – Query Path Information

On output, *PathInfoBuf* is unchanged. If an error occurs, oError points to the GEA2 entry that caused the error. The buffer pointed to by fpFEA2List is filled in with the returned information. If the buffer that fpFEA2List points to is not large enough to hold the returned information (the return code is ERROR_BUFFER_OVERFLOW), cbList is still valid, assuming there is at least enough space for it. Its value is the size, in bytes, of the file's entire EA set on disk, even though only a subset of attributes was requested. The size of the buffer required to hold the EA information is less than or equal to twice the size of the file's entire EA set on disk.

Level 5 File Information
Level 5 returns the fully qualified ASCIIZ name of *PathName* in *PathInfoBuf*. *PathName* may contain global file-name characters.

ulPathInfoBufSize (ULONG) – input

The length, in bytes, of *PathInfoBuf*.

Returns

Return Code.

DosQueryPathInfo returns the following values:

0	NO_ERROR
3	ERROR_PATH_NOT_FOUND
32	ERROR_SHARING_VIOLATION
111	ERROR_BUFFER_OVERFLOW
124	ERROR_INVALID_LEVEL
206	ERROR_FILENAME_EXCED_RANGE
254	ERROR_INVALID_EA_NAME
255	ERROR_EA_LIST_INCONSISTENT

Remarks

For DosQueryPathInfo to return information contained in any of the file information levels, the file object must be opened for read access, with a deny-write sharing mode specified for access by other processes. Thus, if the file object is already accessed by another process that holds conflicting sharing and access rights, a call to DosQueryPathInfo fails.

Related Functions

- DosClose
- DosCreateDir
- DosEnumAttribute
- DosOpen
- DosQueryFileInfo
- DosResetBuffer
- DosSetFileInfo
- DosSetPathInfo

Example Code

This example illustrates how DosQueryPathInfo can be used to obtain information about a directory. DosQueryPathInfo is similar to DosQueryFileInfo. DosQueryPathInfo accepts a path name as an input parameter. DosQueryFileInfo accepts a file handle of an open file as an input parameter. Both functions return the same classes of file information. An important difference between them is that DosQueryPathInfo can be used to obtain information about files and directories, while DosQueryFileInfo can only be used to obtain information about open files.

This example obtains the Level 1 information set for a specified directory. The Level 1 information set for the directory includes the dates and times of creation, last access and last writing. It also includes information about the size of the directory and the directory's standard attributes.

```
#define INCL_DOSFILEMGR   /* File Manager values */
#include <os2.h>
#include <stdio.h>

UCHAR        PathName[60];    /* File or directory path name
                                 string */
ULONG        PathInfoLevel;   /* Data required */
FILESTATUS3  PathInfoBuf;     /* File info buffer */
ULONG        PathInfoBufSize; /* Data buffer size */
APIRET       rc;              /* Return code */

  strcpy(PathName,"D:\\TOOLS\\UTIL\\DIR1");
                       /* Name of the specified directory */

  PathInfoLevel = 1;   /* Indicate that Level 1 information */
                       /*   is desired                      */

  PathInfoBufSize = sizeof(FILESTATUS3);
                       /* Size of the buffer that will      */
                       /*   receive the Level 1 information */

  rc = DosQueryPathInfo(PathName, PathInfoLevel, &PathInfoBuf,
                        PathInfoBufSize);
                       /* On successful return, the Level 1 */
                       /*   directory information is in the */
                       /*   PathInfoBuf buffer              */

  if (rc != 0)
    {
      printf("DosQueryPathInfo error: return code = %ld", rc);
      return;
    }
```

DosQueryProcAddr – Return the Address of the Specified Procedure within a Dynamic Link Module

---

#define INCL_DOSMODULEMGR

```
APIRET DosQueryProcAddr (HMODULE hmodModHandle, ULONG ulOrdinal,
                         PSZ pszProcName, PFN pProcAddr)
```

DosQueryProcAddr returns the address of the specified procedure within a dynamic link module.

Parameters

hmodModHandle (HMODULE) – input

The handle of the dynamic link module that contains the procedure.

ulOrdinal (ULONG) – input

The ordinal number of the procedure whose address is desired. If the ordinal number is non-zero, *ProcName* is ignored.

pszProcName (PSZ) – input

The address of an ASCIIZ name string that contains the procedure name that is being referenced.

Calls to DosQueryProcAddr for entries within the DOSCALLS module are supported for ordinal references only. References to the DOSCALLS module by name strings are not supported, and will return an error. Dynamic link ordinal numbers for DOSCALLS routines are resolved by linking with OS2386.LIB.

pProcAddr (PFN) – output

The address of a doubleword where the procedure address is returned.

Returns

Return Code.

DosQueryProcAddr returns the following values:

0	NO_ERROR
6	ERROR_INVALID_HANDLE
123	ERROR_INVALID_NAME
65079	ERROR_ENTRY_IS_CALLGATE

Remarks

DosQueryProcAddr returns the address of the specified procedure within a dynamic link module.

If you receive return code ERROR_INVALID_HANDLE, issue DosLoadModule and repeat this call.

If you issue DosQueryProcAddr to obtain the address of an entry point that may only be accessed via a call gate, you receive the return code ERROR_ENTRY_IS_CALLGATE.

DosQueryProcAddr – Return the Address of the Specified Procedure within a Dynamic Link Module

Related Functions

- DosFreeModule
- DosLoadModule
- DosQueryModuleName
- DosQueryProcType

Example Code

This example obtains the address of a specified procedure within a dynamic link module. Assume that the appropriate module handle has been placed into *ModuleHandle* already. The example specifies the procedure by name rather than by ordinal number.

```
#define INCL_DOSMODULEMGR   /* Module Manager values */
#include <os2.h>
#include <stdio.h>

HMODULE   ModuleHandle;  /* Module handle */
ULONG     Ordinal;       /* Ordinal number of procedure */
UCHAR     ProcName[40];  /* Procedure name string */
PFN       ProcAddress;   /* Procedure address (returned) */
APIRET    rc;            /* Return code */

   strcpy(ProcName,"SearchProc2");
                         /* Specify the name of the procedure */

   Ordinal = 0;          /* The zero value indicates that the */
                         /*   procedure name string has been  */
                         /*   provided                        */

   rc = DosQueryProcAddr(ModuleHandle, Ordinal, ProcName,
                            &ProcAddress);
                         /* On successful return, the       */
                         /*   ProcAddress variable contains */
                         /*   the address of the procedure  */
                         /*   within the module             */

   if (rc != 0)
     {
       printf("DosQueryProcAddr error: return code = %ld", rc);
       return;
     }
```

DosQueryProcType –
Return Procedure Type within a Dynamic Link Module

```
#define INCL_DOSMODULEMGR
```

```
APIRET DosQueryProcType (HMODULE hmodModHandle, ULONG ulOrdinal,
                         PSZ pszProcName, PULONG pProcType)
```

DosQueryProcType returns the type of the specified procedure within a dynamic link module. The type returned indicates whether the specified procedure is a 16-bit or 32-bit callable procedure.

Parameters

hmodModHandle (HMODULE) – input

The handle of the dynamic link module that contains the procedure.

ulOrdinal (ULONG) – input

The ordinal number of the procedure whose type is desired. If the ordinal number is non-zero, *ProcName* is ignored.

pszProcName (PSZ) – input

The address of an ASCIIZ name string that contains the procedure name that is being referenced.

Calls to DosQueryProcType for entries within the DOSCALLS module are supported for ordinal references only. References to the DOSCALLS module by name strings are not supported, and will return an error. Dynamic link ordinal numbers for DOSCALLS routines are resolved by linking with OS2386.LIB.

pProcType (PULONG) – output

The address of a doubleword where the procedure type is returned. The value returned in this field is one of the following:

Value	Definition
0	(PT_16BIT): Procedure is 16-bit.
1	(PT_32BIT): Procedure is 32-bit.

Returns

Return Code.

DosQueryProcType returns the following values:

0	NO_ERROR
6	ERROR_INVALID_HANDLE
123	ERROR_INVALID_NAME
182	ERROR_INVALID_ORDINAL

Remarks

DosQueryProcType returns the type of the specified procedure within a dynamic link module.

The type returned indicates whether the specified procedure is a 16-bit or 32-bit callable procedure.

If return code ERROR_INVALID_HANDLE is received, issue DosLoadModule and then issue DosQueryProcType again.

DosQueryProcType – Return Procedure Type within a Dynamic Link Module

Related Functions

- DosFreeModule
- DosLoadModule
- DosQueryModuleName
- DosQueryProcAddr

Example Code

This example obtains the type of a procedure that resides within a specified dynamic link module. Assume that the appropriate module handle has been placed into *ModuleHandle* already. The example specifies the procedure by name rather than by ordinal number.

```
#define INCL_DOSMODULEMGR   /* Module Manager values */
#include <os2.h>
#include <stdio.h>

HMODULE   ModuleHandle;  /* Module handle */
ULONG     Ordinal;       /* Ordinal number of procedure */
UCHAR     ProcName[40];  /* Procedure name string */
ULONG     ProcType;      /* Procedure type (returned) */
APIRET    rc;            /* Return code */

   strcpy(ProcName,"SearchProc2");
                         /* Specify the name of the procedure */

   Ordinal = 0;          /* The zero value indicates that the */
                         /*   procedure name string has been  */
                         /*   provided                        */

   rc = DosQueryProcType(ModuleHandle, Ordinal, ProcName,
                              &ProcType);
                         /* On successful return, the ProcType */
                         /*   variable will contain a value    */
                         /*   indicating whether the specified */
                         /*   procedure is a 16-bit procedure  */
                         /*   or a 32-bit procedure            */

   if (rc != 0)
     {
       printf("DosQueryProcType error: return code = %ld", rc);
       return;
     }
```

DosQueryQueue — Query Queue

#define INCL_DOSQUEUES

APIRET DosQueryQueue (HQUEUE QueueHandle, PULONG pNumberElements)

DosQueryQueue queries the number of elements in a queue.

Parameters

QueueHandle (HQUEUE) — input

The handle of the queue to be queried.

pNumberElements (PULONG) — output

A pointer to the number of elements in the queue.

Returns

Return Code.

DosQueryQueue returns the following values:

0	NO_ERROR
337	ERROR_QUE_INVALID_HANDLE

Remarks

DosQueryQueue returns the number of elements that are currently in a queue. This function can be used by the server process and its threads, as well as by any client processes that have gained access to the queue by issuing DosOpenQueue.

If the server process closes the queue before this request is made, ERROR_QUE_INVALID_HANDLE is returned.

Related Functions

- DosCloseQueue
- DosCreateQueue
- DosOpenQueue
- DosPeekQueue
- DosPurgeQueue
- DosReadQueue
- DosWriteQueue

Example Code

This example finds the number of entries in a queue. Assume that the caller has placed the handle of the queue into *QueueHandle* already.

```
#define INCL_DOSQUEUES   /* Queue values */
#include <os2.h>
#include <stdio.h>

HQUEUE   QueueHandle;     /* Queue handle */
ULONG    NumberElements;  /* Size of the queue */
APIRET   rc;              /* Return code */

   rc = DosQueryQueue(QueueHandle, &NumberElements);
                          /* On succesful return, the variable */
                          /*   NumberElements contains the     */
                          /*   number of entries currently     */
                          /*   in the queue                    */

   if (rc != 0)
     {
       printf("DosQueryQueue error: return code = %ld", rc);
       return;
     }
```

DosQueryResourceSize — Return the size of the Specified Resource Object

---

#define INCL_DOSMODULEMGR

```
APIRET DosQueryResourceSize (HMODULE hmodModHandle, ULONG ulTypeID,
                             ULONG ulNameID, PULONG pSize)
```

DosQueryResourceSize returns the size of the specified resource object.

Parameters

hmodModHandle (HMODULE) — input

The handle of the module that has the required resource. A value of zero means to get the size from the current process. A value other than zero is a module handle that was returned by DosLoadModule.

ulTypeID (ULONG) — input

The type identifier of the resource. This field can have one of the following values:

Value	Definition
1	(RT_POINTER): Mouse pointer shape
2	(RT_BITMAP): Bit map
3	(RT_MENU): Menu template
4	(RT_DIALOG): Dialog template
5	(RT_STRING): String tables
6	(RT_FONTDIR): Font directory
7	(RT_FONT): Font
8	(RT_ACCELTABLE): Accelerator tables
9	(RT_RCDATA): Binary data
10	(RT_MESSAGE): Error message tables
11	(RT_DLGINCLUDE): Dialog include file name
12	(RT_VKEYTBL): Key to vkey tables
13	(RT_KEYTBL): Key to UGL tables
14	(RT_CHARTBL): Glyph to character tables
15	(RT_DISPLAYINFO): Screen display information
16	(RT_FKASHORT): Function key area short form
17	(RT_FKALONG): Function key area long form
18	(RT_HELPTABLE): Help table for Help manager
19	(RT_HELPSUBTABLE): Help subtable for Help manager
20	(RT_FDDIR): DBCS unique/font driver directory
21	(RT_FD): DBCS unique/font driver

ulNameID (ULONG) — input

The name identifier of the resource.

pSize (PULONG) — output

The address of a doubleword in which the size, in bytes, of the resource is returned.

DosQueryResourceSize — Return the size of the Specified Resource Object

Returns

Return Code.

DosQueryResourceSize returns the following values:

0	NO_ERROR
6	ERROR_INVALID_HANDLE
87	ERROR_INVALID_PARAMETER

Remarks

DosQueryResourceSize returns the size of the specified resource object.

Resource objects are read-only data objects that can be accessed dynamically at run time. The access key is two numbers. The first number is a type ID; the second, a name ID. These are similar to the file-extension and file-name portions of a file name.

Resource objects are placed into an executable file by the Resource Compiler (RC.EXE).

This function obtains the size of resources loaded from 16-bit executable files or dynamic link libraries (DLLs), since the size is not explicitly stored in most resources.

Related Functions

- DosFreeResource
- DosGetResource
- DosLoadModule

DosQueryResourceSize — Return the size of the Specified Resource Object

Example Code

This example obtains the size of a specified resource object that resides within a dynamic link module. Assume that the handle of the module that contains the desired resource has been placed into *ModHandle* already. Assume that the appropriate resource type identifier has been placed into *TypeID* already, and that the appropriate resource name identifier has been placed into *NameID* already. The two identifiers are derived from the development process that created the module (and its contained resources).

```
#define  INCL_DOSRESOURCES   /* Resource types */
#include <os2.h>
#include <stdio.h>

HMODULE   ModHandle;  /* Handle of the module that has the
                         required resource */
ULONG     TypeID;     /* Resource type identifier */
ULONG     NameID;     /* Resource name identifier */
ULONG     Size;       /* Size of the resource (returned) */
APIRET    rc;         /* Return code */

   rc = DosQueryResourceSize(ModHandle, TypeID, NameID, &Size);
                     /* On successful return, the Size  */
                     /*   variable contains the size of */
                     /*   specified resource (in bytes) */

   if (rc != 0)
     {
       printf("DosQueryResourceSize error: return code = %ld", rc);
       return;
     }
```

DosQuerySysInfo — Return Values of Static System Variables

#define INCL_DOSFILEMGR

APIRET DosQuerySysInfo (ULONG ulStartIndex, ULONG ulLastIndex, PVOID pDataBuf, ULONG ulDataBufLen)

DosQuerySysInfo returns values of static system variables.

Parameters

ulStartIndex (ULONG) — input

Ordinal of the first system variable to return.

ulLastIndex (ULONG) — input

Ordinal of the last system variable to return.

pDataBuf (PVOID) — output

Address of the data buffer where the system returns the variable values.

ulDataBufLen (ULONG) — input

Length, in bytes, of the data buffer.

Returns

Return Code.

DosQuerySysInfo returns the following values:

0	NO_ERROR
87	ERROR_INVALID_PARAMETER
111	ERROR_BUFFER_OVERFLOW

Remarks

DosQuerySysInfo returns a single system variable or a range of system variables to a user-allocated buffer. To request a single system variable, set *StartIndex* equal to *LastIndex*. To request a range of system variables, set *StartIndex* less than *LastIndex*.

Each system variable is a doubleword value. The following list gives the ordinal index, name, and description of the system variables.

1. QSV_MAX_PATH_LENGTH: Maximum length, in bytes, of a path name.
2. QSV_MAX_TEXT_SESSIONS: Maximum number of text sessions.
3. QSV_MAX_PM_SESSIONS: Maximum number of PM sessions.
4. QSV_MAX_VDM_SESSIONS: Maximum number of DOS sessions.
5. QSV_BOOT_DRIVE: Drive from which the system was started (1 means drive A, 2 means drive B, and so on).
6. QSV_DYN_PRI_VARIATION: Dynamic priority variation flag (0 means absolute priority, 1 means dynamic priority).
7. QSV_MAX_WAIT: Maximum wait in seconds.
8. QSV_MIN_SLICE: Minimum time slice in milliseconds.
9. QSV_MAX_SLICE: Maximum time slice in milliseconds.
10. QSV_PAGE_SIZE: Memory page size in bytes. This value is 4096 for the 80386 processor.

DosQuerySysInfo –
Return Values of Static System Variables

11. QSV_VERSION_MAJOR: Major version number.
12. QSV_VERSION_MINOR: Minor version number.
13. QSV_VERSION_REVISION: Revision letter.
14. QSV_MS_COUNT: Value of a 32-bit, free-running millisecond counter. This value is zero when the system is started.
15. QSV_TIME_LOW: Low-order 32 bits of the time in seconds since January 1, 1970 at 0:00:00.
16. QSV_TIME_HIGH: High-order 32 bits of the time in seconds since January 1, 1970 at 0:00:00.
17. QSV_TOTPHYSMEM: Total number of pages of physical memory in the system. One page is 4KB.
18. QSV_TOTRESMEM: Total number of pages of resident memory in the system.
19. QSV_TOTAVAILMEM: Maximum number of pages of memory that can be allocated by all processes in the system. This number is advisory and is not guaranteed, since system conditions change constantly.
20. QSV_MAXPRMEM: Maximum number of bytes of memory that this process can allocate in its private arena. This number is advisory and is not guaranteed, since system conditions change constantly.
21. QSV_MAXSHMEM: Maximum number of bytes of memory that a process can allocate in the shared arena. This number is advisory and is not guaranteed, since system conditions change constantly.
22. QSV_TIMER_INTERVAL: Timer interval in tenths of a millisecond.
23. QSV_MAX_COMP_LENGTH: Maximum length, in bytes, of one component in a path name.

An application can specify file objects managed by an installable file system that supports long file names. Because some installable file systems support longer names than others, the application should issue DosQuerySysInfo upon initialization.

DosQuerySysInfo returns the maximum path length (QSV_MAX_PATH_LENGTH) supported by the installed file system. The path length includes the drive specifier (d:), the leading backslash (\ ), and the trailing null character. The value returned by DosQuerySysInfo can be used to allocate buffers for path names returned by other functions, for example, DosFindFirst and DosFindNext.

Related Functions

- DosCreateDir
- DosFindFirst
- DosFindNext
- DosOpen
- DosQueryCurrentDir
- DosQueryFSInfo
- DosQueryPathInfo
- DosSearchPath
- DosSetCurrentDir
- DosSetPathInfo
- DosSetFSInfo

Example Code

This example obtains the values of three static system variables.

```
#define INCL_DOSFILEMGR   /* File Manager values */
#include <os2.h>
#include <stdio.h>

ULONG   StartIndex;     /* Ordinal of 1st variable to return */
ULONG   LastIndex;      /* Ordinal of last variable to return */
UCHAR   DataBuf[50];    /* System information (returned) */
ULONG   DataBufLen;     /* Data buffer size */
APIRET  rc;             /* Return code */

   StartIndex = 2;      /* In this example we will ask for the */
   LastIndex = 4;       /*   maximum number of Text, PM and    */
                        /*   DOS sessions on the local system  */

   DataBufLen = 50;     /* Size of the supplied data buffer */

   rc = DosQuerySysInfo(StartIndex, LastIndex, DataBuf,
                                    DataBufLen);
                        /* On successful return, the three    */
                        /*   requested doubleword values will */
                        /*   be contained within the supplied */
                        /*   data buffer                      */

   if (rc != 0)
     {
       printf("DosQuerySysInfo error: return code = %ld", rc);
       return;
     }
```

DosQueryVerify –
Return the State of the Verification Flag

#define INCL_DOSFILEMGR

APIRET DosQueryVerify (PBOOL32 ppVerifySetting)

DosQueryVerify determines if write verification is enabled.

Parameters

ppVerifySetting (PBOOL32) – output

Address of the verify mode for the process.

Value	Definition
0	Verify mode is not active.
1	Verify mode is active.

Returns

Return Code.

DosQueryVerify returns the following values:

0 NO_ERROR

Remarks

When the verify mode is active, the operating system verifies that data written to the disk is recorded correctly, even though disk recording errors are rare.

Related Functions

- DosSetVerify

Example Code

This example determines if write verification is enabled.

```
#define INCL_DOSFILEMGR   /* File Manager values */
#include <os2.h>
#include <stdio.h>

BOOL32    VerifySetting; /* Pointer to current verify mode
                            (returned)  */
APIRET    rc;            /* Return code */

   rc = DosQueryVerify(&VerifySetting);
                        /* On return, the variable           */
                        /*   VerifySetting is set to a binary */
                        /*   value indicating whether or not  */
                        /*   write verification is enabled    */

   if (rc != 0)
     {
       printf("DosQueryVerify error: return code = %ld", rc);
       return;
     }
```

DosRaiseException – Raise Exception

#define INCL_DOSEXCEPTIONS

APIRET DosRaiseException (PEXCEPTIONREPORTRECORD pExceptionReportRecord)

DosRaiseException raises an exception for the current thread.

Parameters

pExceptionReportRecord (PEXCEPTIONREPORTRECORD) – input

A pointer to an exception report record that contains exception-specific information needed for the exception to be raised. The pointer to the exception report record, as well as certain handler flags in the structure, are supplied by the system.

Returns

Return Code.

DosRaiseException returns the following values:

0 NO_ERROR

Remarks

DosRaiseException enables a thread to raise a synchronous exception that has been deferred from a must-complete section. DosRaiseException can also be used to simulate an asynchronous or synchronous exception.

For a detailed list of the system exceptions, see Appendix C, "System Exceptions" on page C-1.

Related Functions

- DosAcknowledgeSignalException
- DosEnterMustComplete
- DosExitMustComplete
- DosSendSignalException
- DosSetExceptionHandler
- DosSetSignalExceptionFocus
- DosUnsetExceptionHandler
- DosUnwindException

DosRaiseException – Raise Exception

Example Code

This example shows how a thread can raise a synchronous exception that has been deferred within a must-complete section.

Assume that the variable *pexcept* has already been set to point to the exception record structure that pertains to the exception that is currently being handled.

```
#define INCL_DOSEXCEPTIONS   /* Exception values */
#include <os2.h>
#include <stdio.h>

PEXCEPTIONREPORTRECORD pexcept;
APIRET  rc;                      /* Return code */

   rc = DosRaiseException(pexcept);

   if (rc != 0)
     {
       printf("DosRaiseException error: return code = %ld", rc);
       return;
     }
```

DosRead – Read from a File, Pipe, or Device to a Buffer

#define INCL_DOSFILEMGR

```
APIRET DosRead (HFILE FileHandle, PVOID pBufferArea, ULONG ulBufferLength,
                PULONG pBytesRead)
```

DosRead reads the specified number of bytes from a file, pipe, or device to a buffer location.

Parameters

FileHandle (HFILE) – input

File handle obtained from DosOpen.

pBufferArea (PVOID) – output

Address of the buffer to receive the bytes read.

ulBufferLength (ULONG) – input

The length, in bytes, of *BufferArea*. This is the number of bytes to be read.

pBytesRead (PULONG) – output

Address of the variable to receive the number of bytes actually read.

Returns

Return Code.

DosRead returns the following values:

0	NO_ERROR
5	ERROR_ACCESS_DENIED
6	ERROR_INVALID_HANDLE
26	ERROR_NOT_DOS_DISK
33	ERROR_LOCK_VIOLATION
109	ERROR_BROKEN_PIPE
234	ERROR_MORE_DATA

Remarks

The requested number of bytes might not be read. If the value returned in *BytesRead* is zero, the process tried to read from the end of the file.

A value of zero for *BufferLength* is not considered an error. In such a case, the system treats the request as a null operation.

The file pointer is moved to the desired position by reading data, writing data, or issuing DosSetFilePtr.

If you issue DosOpen with the Direct Open flag set to 1 in the *OpenMode* parameter, you have direct access to an entire disk or diskette volume, independent of the file system. You must lock the logical volume before accessing it, and you must unlock the logical volume when you are finished accessing it. Issue DosDevIOCtl for Category 8, Function 0 to lock the logical volume, and for Category 8, Function 1 to unlock the logical volume. While the logical volume is locked, no other process can access it.

Named-Pipe Considerations

A named pipe is read as one of the following:

DosRead –
Read from a File, Pipe, or Device to a Buffer

- A byte pipe in byte-read mode
- A message pipe in message-read mode
- A message pipe in byte-read mode.

A byte pipe must be in byte-read mode to be read; an error is returned if it is in message-read mode. All currently available data, up to the size requested, is returned.

A message pipe can be read in either message-read mode or byte-read mode. When the message pipe is in message-read mode, a read operation that is larger than the next available message returns only that message. *BytesRead* is set to indicate the size of the message returned.

A read operation that is smaller than the next available message returns with the number of bytes requested and an ERROR_MORE_DATA return code. When the reading of a message is resumed after ERROR_MORE_DATA is returned, a read operation always blocks until the next piece (or the rest) of the message can be transferred. DosPeekNPipe can be issued to determine how many bytes are left in the message.

A message pipe in byte-read mode is read as if it were a byte stream, and DosRead skips over message headers. This is like reading a byte pipe in byte-read mode.

When blocking mode is set for a named pipe, a read operation blocks until data is available. In this case, the read operation never returns with *BytesRead* equal to zero, except at the end of the file. When the mode is set to message-read, messages are always read in their entirety, except when the message is bigger than the size of the read operation.

BytesRead can equal zero in nonblocking mode, but only when no data is available at the time of the read operation.

Related Functions

- DosOpen
- DosSetFilePtr
- DosWrite

Example Code

This example reads a specified number of bytes from a file into a user-supplied buffer. Assume that a file handle for the desired file has been placed into *FileHandle* already.

```
#define INCL_DOSFILEMGR   /* File Manager values */
#include <os2.h>
#include <stdio.h>

HFILE    FileHandle;      /* File Handle */
UCHAR    BufferArea[256]; /* User buffer (returned) */
ULONG    BufferLength;    /* Buffer length */
ULONG    BytesRead;       /* Bytes read (returned) */
APIRET   rc;              /* Return code */

   BufferLength = 256;    /* Size of user-supplied buffer */

   rc = DosRead(FileHandle, BufferArea, BufferLength, &BytesRead);
                          /* On successful return, the user-   */
                          /*   supplied buffer contains up to  */
                          /*   the requested number of bytes   */
                          /*   from the file, and the variable */
                          /*   BytesRead contains the number   */
                          /*   of bytes that were actually     */
                          /*   read into the buufer            */

   if (rc != 0)
     {
       printf("DosRead error: return code = %ld", rc);
       return;
     }
```

DosReadQueue — Read Queue

#define INCL_DOSQUEUES

```
APIRET DosReadQueue (HQUEUE QueueHandle, PREQUESTDATA ppRequest,
                     PULONG pDataLength, PPVOID ppDataAddress,
                     ULONG ulElementCode, BOOL32 f32NoWait, PBYTE pbElemPriority,
                     HEV SemHandle)
```

DosReadQueue reads an element from a queue.

Parameters

QueueHandle (HQUEUE) — input

The handle of the queue from which an element is to be removed.

ppRequest (PREQUESTDATA) — output

A pointer to a two-doubleword data field that returns the following information:

Value	Definition
1	The identification of the process (PID) that added the element to the queue.
2	An event code that is specified by the application. The data in this field is the same as the data that was furnished in the *Request* parameter of the DosWriteQueue request for the corresponding queue element. The value of this data is understood by both the client thread and the server thread. There is no special meaning to this data, and the operating system does not alter it.

pDataLength (PULONG) — output

A pointer to the length, in bytes, of the data that is being removed.

ppDataAddress (PPVOID) — output

A pointer to the element that is being removed from the queue. (This field may or may not be the same as the value of *DataBuffer* that was specified with DosWriteQueue when the element was added to the queue. If QUE_CONVERT_ADDRESS was specified when the queue was created, the addresses of any elements that are written to the queue by the 16-bit DosWriteQueue function are converted to 32-bit addresses.)

ulElementCode (ULONG) — input

An indicator that specifies whether to start at the beginning of the queue or to remove a queue element that was previously examined by DosPeekQueue.

Value	Definition
0	This field is set to 0 by the application to indicate "remove the first element in the queue," according to the order that was specified when the queue was created (FIFO, LIFO, or priority).
non-0	The field is set to non-0 (to the value that was returned by a previous DosPeekQueue operation) to indicate "remove the element that was examined by DosPeekQueue."

f32NoWait (BOOL32) — input

The action to be performed when no entries are found in the queue.

Value	Definition
0	(DCWW_WAIT) The requesting thread waits for an element to be added to the queue.

1 (DCWW_NOWAIT) The requesting thread does not wait, and DosReadQueue returns with ERROR_QUE_EMPTY.

pbElemPriority (PBYTE) – output

The address of the element's priority value. This is the value that was specified for *ElemPriority* by DosWriteQueue when it added the element to the queue. *ElemPriority* is a numerical value in the range of 0 to 15, with 15 being the highest priority.

SemHandle (HEV) – input

The handle of an event semaphore that is to be posted when when data is added to the queue and *NoWait* is set to 1. (This parameter is ignored if *NoWait* is set to 0.) The event semaphore may be shared or private, depending on whether the queue is shared across processes.

Note: The first time an event-semaphore handle is supplied in a DosReadQueue or DosPeekQueue request for which *NoWait* is set to 1, the handle is saved by the system. The same handle must be supplied in all subsequent DosReadQueue and DosPeekQueue requests that are issued for that queue.

Returns

Return Code.

DosReadQueue returns the following values:

0	NO_ERROR
87	ERROR_INVALID_PARAMETER
330	ERROR_QUE_PROC_NOT_OWNED
333	ERROR_QUE_ELEMENT_NOT_EXIST
337	ERROR_QUE_INVALID_HANDLE
342	ERROR_QUE_EMPTY
433	ERROR_QUE_INVALID_WAIT

Remarks

DosReadQueue reads (removes) an element from a queue. This function can be issued only by the server process and its threads.

If the *NoWait* parameter is set to 1, an event semaphore must be provided so that the calling thread can determine when an element has been placed into the queue. The semaphore is created by calling DosCreateEventSem, and its handle is supplied in the *SemHandle* parameter of DosReadQueue.

The first time an event-semaphore handle is supplied in a DosReadQueue or DosPeekQueue request for which *NoWait* has been set to 1, the handle is saved by the system. The same handle must be supplied in all subsequent DosReadQueue and DosPeekQueue requests that are issued for the same queue; if a different handle is supplied, ERROR_INVALID_PARAMETER is returned.

When a client process adds an element to the queue, the system automatically opens and posts the semaphore. The server can either issue DosQueryEventSem periodically to determine whether the semaphore has been posted, or it can issue DosWaitEventSem. DosWaitEventSem causes the calling thread to block until the semaphore is posted.

After the event semaphore has been posted, the calling thread must issue DosReadQueue again to remove the newly added queue element.

DosReadQueue – Read Queue

Related Functions

- DosCloseQueue
- DosCreateQueue
- DosOpenQueue
- DosPeekQueue
- DosPurgeQueue
- DosQueryQueue
- DosWriteQueue

Example Code

This example reads an element from the queue. Assume that the caller has placed the handle of the queue into *QueueHandle* already. Assume that the identifier of the process that owns the queue has been placed into *OwningPID* already.

```
#define INCL_DOSQUEUES   /* Queue values */
#include <os2.h>
#include <stdio.h>

HQUEUE        QueueHandle;   /* Queue handle */
REQUESTDATA   Request;       /* Request-identification data */
ULONG         DataLength;    /* Length of element received */
PVOID         DataAddress;   /* Address of element received */
ULONG         ElementCode;   /* Request a particular element */
BOOL32        NoWait;        /* No wait if queue is empty */
BYTE          ElemPriority;  /* Priority of element received */
HEV           SemHandle;     /* Semaphore handle */
PID           OwningPID;     /* PID of queue owner */
APIRET        rc;            /* Return code */

   Request.pid = OwningPID;  /* Set request data block to */
                             /*   indicate queue owner    */

   ElementCode = 0;          /* Indicate that the read should */
                             /*   start at the front of the   */
                             /*   queue                       */

   NoWait = 0;               /* Indicate that the read        */
                             /*   should wait if the queue is */
                             /*   currently empty             */

   SemHandle = 0;            /* Unused since this is a call   */
                             /*   that waits synchronously    */

   rc = DosReadQueue(QueueHandle, &Request, &DataLength,
                     &DataAddress, ElementCode, NoWait,
                     &ElemPriority, SemHandle);
                             /* On successful return, the      */
                             /*   DataLength variable contains */
                             /*   the size of the element on   */
                             /*   the queue that is pointed to */
                             /*   by the pointer within the    */
                             /*   DataAddress variable, the    */
                             /*   ElemPriority variable has    */
                             /*   been updated to contain the  */
                             /*   priority of the queue        */
                             /*   element pointed to by the    */
                             /*   DataAddress variable, and    */
                             /*   the Request.ulData variable  */
                             /*   contains any special data    */
                             /*   that the DosWriteQueue       */
                             /*   caller placed into the queue */

   if (rc != 0)
     {
       printf("DosReadQueue error: return code = %ld", rc);
       return;
     }
```

DosReleaseMutexSem – Release Mutex Semaphore

#define INCL_DOSSEMAPHORES

APIRET DosReleaseMutexSem (HMTX hmtx)

DosReleaseMutexSem relinquishes ownership of a mutex semaphore.

Parameters

hmtx (HMTX) – input

The handle of the mutex semaphore to release.

Returns

Return Code.

DosReleaseMutexSem returns the following values:

0	NO_ERROR
6	ERROR_INVALID_HANDLE
288	ERROR_NOT_OWNER

Remarks

DosReleaseMutexSem relinquishes ownership of a mutex semaphore that was requested by DosRequestMutexSem.

Only the thread that owns the mutex semaphore can issue DosReleaseMutexSem.

Related Functions

- DosCloseMutexSem
- DosCreateMutexSem
- DosOpenMutexSem
- DosQueryMutexSem
- DosRequestMutexSem

Example Code

This example relinquishes ownership of a mutex semaphore. Assume that the handle of the semaphore has been placed into *hmtx* already.

```
#define INCL_DOSSEMAPHORES   /* Semaphore values */
#include <os2.h>
#include <stdio.h>

HMTX    hmtx;    /* Mutex semaphore handle */
APIRET  rc;      /* Return code */

   rc = DosReleaseMutexSem(hmtx);

   if (rc != 0)
     {
       printf("DosReleaseMutexSem error: return code = %ld", rc);
       return;
     }
```

DosRequestMutexSem – Request Mutex Semaphore

#define INCL_DOSSEMAPHORES

APIRET DosRequestMutexSem (HMTX hmtx, ULONG ulTimeout)

DosRequestMutexSem requests ownership of a mutex semaphore.

Parameters

hmtx (HMTX) – input

The handle of the mutex semaphore to request.

ulTimeout (ULONG) – input

The time-out in milliseconds. This is the maximum amount of time the user wants to allow the thread to be blocked.

This parameter can also have the following values:

Value	Definition
0	(SEM_IMMEDIATE_RETURN) DosRequestMutexSem returns immediately without blocking the calling thread.
-1	(SEM_INDEFINITE_WAIT) DosRequestMutexSem blocks the calling thread indefinitely.

Returns

Return Code.

DosRequestMutexSem returns the following values:

0	NO_ERROR
6	ERROR_INVALID_HANDLE
95	ERROR_INTERRUPT
103	ERROR_TOO_MANY_SEM_REQUESTS
105	ERROR_SEM_OWNER_DIED
640	ERROR_TIMEOUT

Remarks

DosRequestMutexSem requests ownership of a mutex semaphore.

This function can be called by any thread in the process that created the semaphore. Threads in other processes can also call this function, but they must first gain access to the semaphore by issuing DosOpenMutexSem.

Related Functions

- DosCloseMutexSem
- DosCreateMutexSem
- DosOpenMutexSem
- DosQueryMutexSem
- DosReleaseMutexSem

DosRequestMutexSem – Request Mutex Semaphore

Example Code

This example requests ownership of a mutex semaphore. Assume that the handle of the semaphore has been placed into *hmtx* already.

ulTimeout is the number of milliseconds that the calling thread will wait for ownership of the mutex semaphore. If the specified mutex semaphore is not released during this time interval, the calling thread does not receive ownership of it.

```
#define INCL_DOSSEMAPHORES   /* Semaphore values */
#include <os2.h>
#include <stdio.h>

#ifndef ERROR_TIMEOUT
    #define  ERROR_TIMEOUT    640
    #define  ERROR_INTERRUPT   95
#endif

HMTX    hmtx;        /* Mutex semaphore handle */
ULONG   ulTimeout;   /* Number of milliseconds to wait */
APIRET  rc;          /* Return code */

   ulTimeout = 60000;  /* Wait for a maximum of 1 minute */

   rc = DosRequestMutexSem(hmtx, ulTimeout);

   if (rc == ERROR_TIMEOUT)
     {
       printf("DosRequestMutexSem call timed out");
       return;
     }

   if (rc == ERROR_INTERRUPT)
     {
       printf("DosRequestMutexSem call was interrupted");
       return;
     }

   if (rc != 0)
     {
       printf("DosRequestMutexSem error: return code = %ld", rc);
       return;
     }
```

DosRequestVDD – Request Virtual Device Driver Services

#define INCL_DOSMVDM

```
APIRET DosRequestVDD (HVDD VDDHandle, SGID sgidSessionID, ULONG ulCommand,
                      ULONG ulInputBufferLen, PVOID pInputBuffer,
                      ULONG ulOutputBufferLen, PVOID pOutputBuffer)
```

DosRequestVDD allows a protected-mode OS/2* session to communicate with a virtual device driver (VDD).

Parameters

VDDHandle (HVDD) – input

The handle of a virtual device driver (VDD) returned by a previous call to DosOpenVDD.

sgidSessionID (SGID) – input

The identifier of a specific DOS session, or null.

ulCommand (ULONG) – input

A function code that is specific to a virtual device.

ulInputBufferLen (ULONG) – input

The length, in bytes, of the application data in *InputBuffer*.

pInputBuffer (PVOID) – input

The address of the command-specific information. The system sends this data to the virtual device driver to process the specified command.

ulOutputBufferLen (ULONG) – input

The length, in bytes, of *OutputBuffer*.

pOutputBuffer (PVOID) – output

The address of the buffer where the virtual device driver returns the information for the specified command. This information is specific to the command and the virtual device driver.

Returns

Return Code.

DosRequestVDD returns the following values:

0	NO_ERROR
6	ERROR_INVALID_HANDLE
21	ERROR_NOT_READY
644	ERROR_INVALID_CALLER

Remarks

The system calls every DosRequestVDD procedure registered by VDHRegisterVDD under the VDD name associated with the specified handle. This calling continues until a virtual device driver gives a return code other than VDDREQ_PASS. There is no predefined order to the calling sequence.

DosRequestVDD –
Request Virtual Device Driver Services

Related Functions

- DosCloseVDD
- DosOpenVDD

Example Code

This example illustrates how a protected-mode OS/2 process can communicate with a virtual device driver (VDD). The example shows a protected-mode process calling a hypothetical VDD with a request to read a string of bytes from the VDD. Assume that the handle for the VDD has been placed into *VDDHandle* already. Assume that the session identifier of the specified DOS session has been placed into *SessionID* already.

```
#define INCL_DOSMVDM     /* Multiple DOS sessions values */
#include <os2.h>
#include <stdio.h>
#include <vdmm.h>

HVDD    VDDHandle;        /* Handle of VDD */
SGID    SessionID;        /* Session identifier */
ULONG   Command;          /* VDD function code */
ULONG   InputBufferLen;   /* Length of input buffer */
UCHAR   InputBuffer[10];  /* Command information */
ULONG   OutputBufferLen;  /* Length of output buffer */
UCHAR   OutputBuffer[30]; /* Output data (returned) */
APIRET  rc;               /* Return code */

   Command = 3;   /* Hypothetical command to read a string of */
                  /*  control information from the VDD        */

   strcpy(InputBuffer,"   8");
                  /* Hypothetical command parameter to the VDD */

   InputBufferLen = 4;
                  /* Length of application data being sent to */
                  /*   the VDD in the input buffer            */

   OutputBufferLen = 30;
                  /* Size of the output buffer that will      */
                  /*   accept the returned data from the VDD */

   rc = DosRequestVDD(VDDHandle, SessionID, Command,
              InputBufferLen, InputBuffer, OutputBufferLen,
              OutputBuffer);
                  /* On successful return, the output buffer */
                  /*   contains the string of bytes that was */
                  /*   read from the VDD                     */

   if (rc != 0)
     {
       printf("DosRequestVDD error: return code = %ld", rc);
       return;
     }
```

#define INCL_DOSFILEMGR

APIRET DosResetBuffer (HFILE FileHandle)

DosResetBuffer writes the buffers for the specified file to the device.

Parameters

FileHandle (HFILE) – input

The handle of the file whose buffers are to be written to the disk. If *FileHandle* has a value of hex FFFF, all of the buffers for all of the file handles of the process are written to the disk.

Returns

Return Code.

DosResetBuffer returns the following values:

0	NO_ERROR
2	ERROR_FILE_NOT_FOUND
5	ERROR_ACCESS_DENIED
6	ERROR_INVALID_HANDLE

Remarks

When DosResetBuffer is issued for a file handle, the contents of the file's buffers are written to the disk, and the file's directory entry is updated as if the file had been closed; however, the file remains open.

DosResetBuffer should be issued with caution. When files are on diskettes, issuing DosResetBuffer could have the undesirable effect of requiring the user to insert and remove a large number of diskettes.

Named-Pipe Considerations

Issuing DosResetBuffer for a named pipe results in an operation that is similar to forcing the buffer cache to the disk. The request blocks the calling process at one end of the pipe until all written data has been read at the other end.

Related Functions

- DosClose
- DosOpen
- DosWrite

Example Code

This example opens a file, writes some data to the file's buffer, then writes the file's system buffer.

```
#define INCL_DOSFILEMGR   /* File Manager values */
#include <os2.h>
#include <stdio.h>

#define OPEN_FILE 0x01
#define CREATE_FILE 0x10
#define FILE_ARCHIVE 0x20
#define FILE_EXISTS OPEN_FILE
#define FILE_NOEXISTS CREATE_FILE
```

DosResetBuffer — Reset Buffer

```
#define DASD_FLAG 0
#define INHERIT 0x80
#define WRITE_THRU 0
#define FAIL_FLAG 0
#define SHARE_FLAG 0x10
#define ACCESS_FLAG 0x02

#define FILE_NAME "test.dat"
#define FILE_SIZE 800L
#define FILE_ATTRIBUTE FILE_ARCHIVE
#define EABUF 0L

HFILE   FileHandle;
ULONG   Wrote;
ULONG   Action;
PSZ     FileData[100];
APIRET  rc;             /* Return code */

   Action = 2;
   strcpy(FileData, "Data...");

   rc = DosOpen(FILE_NAME,                 /* File path name */
                &FileHandle,              /* File handle */
                &Action,                  /* Action taken */
                FILE_SIZE,                /* File primary allocation */
                FILE_ATTRIBUTE,           /* File attribute */
                FILE_EXISTS | FILE_NOEXISTS,    /* Open function type */
                DASD_FLAG | INHERIT |          /* Open mode of the file */
                WRITE_THRU | FAIL_FLAG |
                SHARE_FLAG | ACCESS_FLAG,
                EABUF);                   /* No extended attributes */

   if (rc != 0)
     {
       printf("DosOpen error: return code = %ld",rc);
       return;
     }

   rc = DosWrite(FileHandle,           /* File handle */
                 (PVOID) FileData,     /* User buffer */
                 sizeof(FileData),     /* Buffer length */
                 &Wrote);              /* Bytes written */

   if (rc != 0)
     {
       printf("DosWrite error: return code = %ld",rc);
       return;
     }

   rc = DosResetBuffer(FileHandle);    /* File handle */

   if (rc != 0)
     {
       printf("DosResetBuffer error: return code = %ld",rc);
       return;
     }
```

DosResetEventSem – Reset Event Semaphore

#define INCL_DOSSEMAPHORES

APIRET DosResetEventSem (HEV hev, PULONG ppulPostCt)

DosResetEventSem resets an event semaphore.

Parameters

hev (HEV) – input

The handle of the event semaphore to reset.

ppulPostCt (PULONG) – output

A pointer to receive the event semaphore's post count. The post count is the number of calls to DosPostEventSem that have been made since the last time the semaphore was in the reset state.

Returns

Return Code.

DosResetEventSem returns the following values:

0	NO_ERROR
6	ERROR_INVALID_HANDLE
300	ERROR_ALREADY_RESET

Remarks

DosResetEventSem resets an event semaphore, causing all threads that subsequently call DosWaitEventSem to be blocked. It also returns the post count for the semaphore. The post count is the number of times that DosPostEventSem has been called since the last time the semaphore was in the reset state.

This function can be called by any thread in the process that created the semaphore. Threads in other processes can also call this function, but they must first gain access to the semaphore by calling DosOpenEventSem.

Related Functions

- DosCloseEventSem
- DosCreateEventSem
- DosOpenEventSem
- DosPostEventSem
- DosQueryEventSem
- DosWaitEventSem

DosResetEventSem – Reset Event Semaphore

Example Code

This example resets an event semaphore. Assume that the handle of the semaphore has been placed into *hev* already.

```
#define INCL_DOSSEMAPHORES   /* Semaphore values */
#include <os2.h>
#include <stdio.h>

HEV     hev;        /* Event semaphore handle */
ULONG   ulPostCt;   /* Post count for the event semaphore
                       (returned) */
APIRET  rc;         /* Return code */

   rc = DosResetEventSem(hev, &ulPostCt);
                    /* On successful return, the ulPostCt   */
                    /*   variable contains the number of    */
                    /*   previous posts that were performed */
                    /*   on the event semaphore before it   */
                    /*   was reset by this function         */

   if (rc != 0)
     {
       printf("DosResetEventSem error: return code = %ld", rc);
       return;
     }
```

DosResumeThread — Restart a Thread

#define INCL_DOSPROCESS

APIRET DosResumeThread (TID idThreadID)

DosResumeThread restarts a thread that was previously stopped with DosSuspendThread.

Parameters

idThreadID (TID) — input

Thread identifier of the resumed thread.

Returns

Return Code.

DosResumeThread returns the following values:

0	NO_ERROR
90	ERROR_NOT_FROZEN
309	ERROR_INVALID_THREADID

Remarks

DosResumeThread restarts a thread that was previously stopped with DosSuspendThread.

If the thread is not in a suspended state when you issue DosResumeThread for it, ERROR_NOT_FROZEN is returned.

Related Functions

- DosCreateThread
- DosSuspendThread

Example Code

This example restarts a thread that was previously suspended by DosSuspendThread. Assume that the target thread ID has been placed into *ThreadID* already.

```
#define INCL_DOSPROCESS     /* Process and thread values */
#include <os2.h>
#include <stdio.h>

TID     ThreadID;   /* Thread ID of thread to resume */
APIRET  rc;         /* Return code */

   rc = DosResumeThread(ThreadID);

   if (rc != 0)
     {
       printf("DosResumeThread error: return code = %ld", rc);
       return;
     }
```

DosScanEnv –
Search an Environment Segment for an Environment Variable

```
#define INCL_DOSFILEMGR
```

APIRET DosScanEnv (PSZ pszEnvVarName, PSZ pszResultPointer)

DosScanEnv searches an environment segment for an environment variable.

Parameters

pszEnvVarName (PSZ) – input

Address of the name of the environment variable. Do not include a trailing equal sign (" = "), since this is not part of the name.

pszResultPointer (PSZ) – output

Address of the variable where the system returns the pointer to the environment string. *ResultPointer* points to the first character of the string that is the value of the environment variable, and can be passed directly to DosSearchPath.

Returns

Return Code.

DosScanEnv returns the following values:

0	NO_ERROR
203	ERROR_ENVVAR_NOT_FOUND

Remarks

Assume that the process' environment contains this statement:

```
DPATH=c:\sysdir;c:\libdir
      |
      |
      |
      ----- ResultPointer points here after the
               following call to DosScanEnv:

DosScanEnv("DPATH", ResultPointer);
```

ResultPointer points to the first character of the value of the environment variable.

Related Functions

- DosSearchPath

DosScanEnv – Search an Environment Segment for an Environment Variable

Example Code

The following example scans the environment segment for the PATH variable, and prints its value. It then searches the path given by inserting the current directory into the value of the PATH variable for the file named 'cmd.exe', and prints the full file name.

```
#define INCL_DOS

#include <os2.h>
#include <stdio.h>

#define ENVVARNAME            "PATH"       /* Environment variable name */
#define FILENAME              "cmd.exe"    /* File for which to search */

main()
{
  PSZ       ResultPointer;          /* Environment scan result pointer
                                       (returned) */
  BYTE      ResultBuffer[128];      /* Path search result
                                               (returned) */
  APIRET    rc;                     /* return code */

  /** Scan environment segment for PATH; notice the far-string pointer **/
  /**   specification ("%Fs") used to print.                           **/

  if(!(rc=DosScanEnv(ENVVARNAME,        /* Environment variable name */
                     ResultPointer)))   /* Scan result pointer
                                           (returned) */
    printf("%s is %Fs\n", ENVVARNAME, ResultPointer);

  /** Search current directory + PATH variable for "cmd.exe"   **/
  if(!(rc=DosSearchPath(SEARCH_CUR_DIRECTORY,    /* Search control
                                                        vector */
                        ENVVARNAME,              /* Search path reference
                                                    string */
                        FILENAME,                /* File name string */
                        ResultBuffer,            /* Search result
                                                    (returned) */
                        sizeof(ResultBuffer))))  /* Length of search
                                                    result */
    printf("Found desired file -- %s\n", ResultBuffer);
}
```

DosSearchPath – Search Path

#define INCL_DOSFILEMGR

```
APIRET DosSearchPath (ULONG ulControl, PSZ pszPathRef, PSZ pszFileName,
                      PBYTE pbResultBuffer, ULONG ulResultBufferLen)
```

DosSearchPath finds files residing along paths. The path string may come from the process environment, or be supplied directly by the caller.

Parameters

ulControl (ULONG) – input

A word bit vector that controls the behavior of DosSearchPath.

Bit	Description
31 – 3	Reserved; must be zero.
2	(SEARCH_IGNORENETERRS 0x00000004) Ignore Network Errors bit. This bit controls whether the search will abort if it encounters a network error, or will continue the search with the next element. This allows you to place network paths in the PATH variable and be able to find executables in components of the PATH variable, even if the network returns an error, for example, if a server is down. If the Ignore Network Errors Bit is 0, DosSearchPath will end the search if it encounters an error from the network. If the Ignore Network Errors Bit is 1, DosSearchPath will continue the search if it encounters network errors.
1	(SEARCH_ENVIRONMENT 0x00000002) Path Source bit. This bit determines how DosSearchPath interprets *PathRef*. 0 : *PathRef* points to the actual search path. The search path string may be anywhere in the calling process's address space. Therefore, it may be in the environment, but is not required. 1 : *PathRef* points to the name of an environment variable in the process environment, and that environment variable contains the search path.
0	(SEARCH_CUR_DIRECTORY 0x00000001) Implied Current bit. This bit controls whether the current directory is implicitly on the front of the search path. 0 : DosSearchPath only searches the current directory if it appears in the search path. 1 : DosSearchPath searches the current working directory before it searches the directories in the search path. For example, Implied Current bit = 0 and path = ".\;a;b" is equivalent to Implied Current bit = 1 and path = "a;b".

pszPathRef (PSZ) – input

Address of the path. If the Path Source bit of *Control* is 0, *PathRef* is the search path that may be anywhere in the caller's address space.

If the Path Source bit of *Control* is 1, *PathRef* is the name of an environment variable that contains the search path.

A search path consists of a sequence of paths separated by a semicolon (;). It is a single ASCIIZ string. The directories are searched in the order they appear in the path. Paths that contain semicolons should be quoted. For example:

```
"c:&this  is  ;  one directory path";thisisanother
```

Environment variable names are simply strings that match name strings in the environment. The equal (=) sign is not part of the name.

pszFileName (PSZ) – input

Address of the ASCIIZ file name. It may contain global file-name characters. If *FileName* does contain global file-name characters, they remain in the result path returned in *ResultBuffer*. This allows applications like CMD.EXE to pass the output directly to DosFindFirst. If there are no global file-name characters in *FileName*, the resulting path returned in *ResultBuffer* is a fully qualified name, and may be passed directly to DosOpen, or any other system function.

pbResultBuffer (PBYTE) – output

Address of the path name of the file, if found. If *FileName* is found in one of the directories along the path, its full path name is returned in *ResultBuffer* (with global file-name characters from *FileName* left in place). The contents of *ResultBuffer* are not meaningful if DosSearchPath returns a non-zero return code.

ulResultBufferLen (ULONG) – input

The length, in bytes, of *ResultBuffer*.

Returns

Return Code.

DosSearchPath returns the following values:

0	NO_ERROR
1	ERROR_INVALID_FUNCTION
2	ERROR_FILE_NOT_FOUND
87	ERROR_INVALID_PARAMETER
111	ERROR_BUFFER_OVERFLOW
203	ERROR_ENVVAR_NOT_FOUND

Remarks

PathRef always points to an ASCIIZ string. Let DPATH be an environment variable in the environment segment of the process.

```
DPATH=c:\sysdir;c:\init   /* In the environment */
```

The following two code fragments are equivalent:

```
DosScanEnv("DPATH", &PathRef);
DosSearchPath(0, /* Path Source Bit = 0 */
    PathRef, "myprog.ini", &ResultBuffer, ResultBufLen);

DosSearchPath(2, /* Path Source Bit = 1 */
    "DPATH", "myprog.ini", &ResultBuffer, ResultBufLen);
```

They both use the search path stored as DPATH in the environment segment. In the first case, the application issues DosScanEnv to find the variable; in the second case, DosSearchPath issues DosScanEnv for the application.

DosSearchPath does not check for consistency or formatting of the names. It issues DosFindFirst on a series of names that it builds from *PathRef* and *FileName*.

To determine the size of the returned path name, *ResultBuffer* must be scanned for the ASCIIZ terminator.

An application must issue DosQuerySysInfo to determine the maximum path length that the operating system supports. The returned value should be used to dynamically allocate buffers that are to be used to store paths.

DosSearchPath – Search Path

Related Functions

- DosFindFirst
- DosFindNext
- DosQuerySysInfo
- DosScanEnv

Example Code

The following example scans the environment segment for the PATH variable and prints its value. It then searches the path given by inserting the current directory into the value of the PATH variable for the file named 'cmd.exe' and prints the full file name.

```
#define INCL_DOS

#include <os2.h>
#include <stdio.h>

#define ENVVARNAME            "PATH"       /* Environment variable name */
#define FILENAME              "cmd.exe"    /* File for which to search */

main()
{
  PSZ       ResultPointer;         /* Environment scan result pointer
                                      (returned) */
  BYTE      ResultBuffer[128];     /* Path search result
                                             (returned) */
  APIRET    rc;                    /* return code */

  /** Scan environment segment for PATH; notice the far-string pointer **/
  /**   specification ("%Fs") used to print.                           **/

  if(!(rc=DosScanEnv(ENVVARNAME,        /* Environment variable name */
                     ResultPointer)))   /* Scan result pointer
                                           (returned) */
    printf("%s is %Fs\n", ENVVARNAME, ResultPointer);

  /** Search current directory + PATH variable for "cmd.exe"   **/
  if(!(rc=DosSearchPath(SEARCH_CUR_DIRECTORY,    /* Search control
                                                    vector */
                        ENVVARNAME,              /* Search path reference
                                                    string */
                        FILENAME,                /* File name string */
                        ResultBuffer,            /* Search result
                                                    (returned) */
                        sizeof(ResultBuffer))))  /* Length of search
                                                    result */
    printf("Found desired file -- %s\n", ResultBuffer);
}
```

DosSelectSession – Select Foreground Session

#define INCL_DOSSESMGR

APIRET DosSelectSession (ULONG ulSessID)

DosSelectSession allows a parent session to switch one of its child sessions to the foreground.

Parameters

ulSessID (ULONG) – input

The identifier of the session to be switched to the foreground. The value specified must have been returned on a previous call to DosStartSession, except that a value of zero indicates switching the caller's session (that is, the parent session) to the foreground.

Returns

Return Code.

DosSelectSession returns the following values:

0	NO_ERROR
224	ERROR_SMG_NO_TARGET_WINDOW
369	ERROR_SMG_INVALID_SESSION_ID
418	ERROR_SMG_INVALID_CALL
459	ERROR_SMG_BAD_RESERVE
460	ERROR_SMG_PROCESS_NOT_PARENT
463	ERROR_SMG_RETRY_SUB_ALLOC

Remarks

DosSelectSession allows a parent session to switch one of its child sessions to the foreground. The session specified will not be brought to the foreground unless the parent session or one of its descendant sessions is currently executing in the foreground.

The foreground session for windowed applications is the session of the application that owns the window focus.

DosSelectSession may only be issued by a parent session to select itself or a child session. DosSelectSession may not be used to select a grandchild session, or any other descendant session beyond a child session. DosSelectSession may only be issued by the process that originally started the specified session (*SessID*) through DosStartSession.

DosSelectSession may only be used to select child sessions that were originally started by the caller with DosStartSession specifying a value of 1 for *Related*. That is, sessions started as independent sessions may not be selected through DosSelectSession.

When DosSelectSession is issued, the session specified will not be brought to the foreground unless the parent session or one of its descendant sessions is currently executing in the foreground.

Return code ERROR_SMG_NO_TARGET_WINDOW is a warning that the session might not be brought to the foreground. If the selected session is a Presentation Manager (PM) application, its window must be created with the FCF_TASKLIST flag bit set on. If the window is created with this bit set off, its session cannot be selected using DosSelectSession, and ERROR_SMG_NO_TARGET_WINDOW is returned.

If you issue DosSelectSession before creating the PM window of the selected session, ERROR_SMG_NO_TARGET_WINDOW is returned. However, if the PM window of the selected session

DosSelectSession – Select Foreground Session

is subsequently created with the FCF_TASKLIST flag bit set on, the window is brought to the foreground if the issuer of DosSelectSession still owns the foreground focus.

If a session still exists but its window has been destroyed, and you issue DosSelectSession for that session, ERROR_SMG_NO_TARGET_WINDOW is returned.

Related Functions

- DosSetSession
- DosStartSession
- DosStopSession

Example Code

This example illustrates how a parent session switches one of its child sessions to the foreground. Assume that the session ID of the desired child session has been placed into *SessID* already.

```
#define INCL_DOSSESMGR      /* Session Manager values */
#include <os2.h>
#include <stdio.h>

ULONG   SessID;     /* Session identifier */
APIRET  rc;         /* Return code */

   rc = DosSelectSession(SessID);

   if (rc != 0)
     {
       printf("DosSelectSession error: return code = %ld", rc);
       return;
     }
```

DosSendSignalException – Send Signal Exception

#define INCL_DOSEXCEPTIONS

APIRET DosSendSignalException (PID idpid, ULONG ulexception)

DosSendSignalException sends a Ctrl + C or a Ctrl + Break signal exception to another process.

Parameters

idpid (PID) – input

The identification of the process that is to receive the signal exception.

ulexception (ULONG) – input

The number of the signal exception to send. Only 1 (XCPT_SIGNAL_INTR) or 4 (XCPT_SIGNAL_BREAK) is allowed.

Returns

Return Code.

DosSendSignalException returns the following values:

0	NO_ERROR
1	ERROR_INVALID_FUNCTION
156	ERROR_SIGNAL_REFUSED
205	ERROR_NO_SIGNAL_SENT
209	ERROR_INVALID_SIGNAL_NUMBER
303	ERROR_INVALID_PROCID

Remarks

DosSendSignalException sends either an XCPT_SIGNAL_INTR (Ctrl+C) or an XCPT_SIGNAL_BREAK (Ctrl+Break) signal exception to another process.

For a detailed list of the system exceptions, see Appendix C, "System Exceptions" on page C-1.

Related Functions

- DosAcknowledgeSignalException
- DosEnterMustComplete
- DosExitMustComplete
- DosRaiseException
- DosSetExceptionHandler
- DosSetSignalExceptionFocus
- DosUnsetExceptionHandler
- DosUnwindException

DosSendSignalException – Send Signal Exception

Example Code

This example sends a Ctrl+C signal to another process. Assume that the process identifier of the other process has been placed into *PID* already.

```
#define INCL_DOSEXCEPTIONS   /* Exception values */
#include <os2.h>
#include <stdio.h>

PID     pid;        /* ID of the process to receive signal
                       exception */
ULONG   exception;  /* Exception number to be sent */
APIRET  rc;         /* Return code */

   exception = XCPT_SIGNAL_INTR;
                    /* Ctrl+C signal number */

   rc = DosSendSignalException(pid, exception);

   if (rc != 0)
     {
       printf("DosSendSignalException error: return code = %ld",
                       rc);
       return;
     }
```

DosSetCurrentDir — Define Current Directory

#define INCL_DOSFILEMGR

APIRET DosSetCurrentDir (PSZ pszDirName)

DosSetCurrentDir defines the current directory.

Parameters

pszDirName (PSZ) — input

Address of the directory path name. The name is an ASCIIZ string.

Returns

Return Code.

DosSetCurrentDir returns the following values:

0	NO_ERROR
2	ERROR_FILE_NOT_FOUND
3	ERROR_PATH_NOT_FOUND
5	ERROR_ACCESS_DENIED
8	ERROR_NOT_ENOUGH_MEMORY
26	ERROR_NOT_DOS_DISK
87	ERROR_INVALID_PARAMETER
108	ERROR_DRIVE_LOCKED
206	ERROR_FILENAME_EXCED_RANGE

Remarks

The directory path does not change if any member of the path does not exist. The current directory changes only for the requesting process.

For file-system drivers, the case of the current directory is set by *DirName*, and not by the case of the directories on the disk. For example, if the directory "c:\bin" is created, and a *DirName* value of "c:\bin," is specified, the current directory returned by DosQueryCurrentDir will be "c:\bin."

Programs running without the NEWFILES bit can set the current directory to a non-8.3 file-name format.

An application must issue DosQuerySysInfo to determine the maximum path length that the operating system supports. The returned value should be used to dynamically allocate buffers that are to be used to store paths.

Related Functions

- DosQueryCurrentDir
- DosQueryCurrentDisk
- DosSetDefaultDisk
- DosQuerySysInfo

DosSetCurrentDir –
Define Current Directory

Example Code

This example changes the current directory to \os2\system.

```
#define INCL_DOSFILEMGR   /* File Manager values */
#include <os2.h>
#include <stdio.h>

#define PATH "\\os2\\system"

APIRET  rc;            /* Return code  */

   rc = DosSetCurrentDir(PATH);

   if (rc != 0)
     {
       printf("DosSetCurrentDir error: return code = %ld",rc);
       return;
     }
```

DosSetDateTime – Set Current Date and Time

#define INCL_DOSDATETIME

```
APIRET DosSetDateTime (PDATETIME ppPDateTime)
```

DosSetDateTime sets the current date and time.

Parameters

ppPDateTime (PDATETIME) – input
Pointer to the DateTime data structure.

Pointer to the address of a structure that provides the following data items:

Hours (UCHAR) Current hour, using values 0 through 23.

Minutes (UCHAR) Current minute, using values 0 through 59.

Seconds (UCHAR) Current second, using values 0 through 59.

Hundredths (UCHAR) Current hundredths of a second, using values 0 through 99.

Day (UCHAR) Current day of the month, using values 1 through 31.

Month (UCHAR) Current month of the year, using values 1 through 12.

Year (USHORT) Current year.

TimeZone (SHORT) The difference in minutes between the current time zone and Greenwich Mean Time (GMT). This value is positive for time zones west of Greenwich, England, and negative for time zones east of Greenwich. A value of -1 indicates that the time zone is undefined.

DayofWeek (UCHAR) Current day of the week, using values 0 through 6. (Sunday is equal to 0.)

Returns

Return Code.

DosSetDateTime returns the following values:

0	NO_ERROR
327	ERROR_TS_DATETIME

Remarks

DosSetDateTime sets the date and time that are maintained by the operating system.

The system verifies that the day is possible for the month and year (even for leap year) and that the values specified for the parameters are within their respective limits; if either of these conditions is violated, ERROR_TS_DATETIME is returned.

To get the date and time, issue DosGetDateTime.

Related Functions

- DosAsyncTimer
- DosGetDateTime
- DosSleep
- DosStartTimer
- DosStopTimer

DosSetDateTime – Set Current Date and Time

Example Code

The following example shows the use of DosSetDateTime. In this example, the current date and time are printed. Then the system date is changed to 5/10/87, and the new information is printed.

```
#define INCL_DOSDATETIME   /* Date and time values */

#include <os2.h>
#include <stdio.h>

main()
{
  DATETIME   DateTime;       /* Structure to hold date/time info. */
  APIRET     rc;             /* Return code */

  rc = DosGetDateTime(&DateTime);     /* Address of d/t structure */

  printf("Today is %d-%d-%d; the time is %d:%d\n", DateTime.month,
         DateTime.day, DateTime.year, DateTime.hours, DateTime.minutes);

  DateTime.day = 10;

  DateTime.month = 5;

  DateTime.year = 1987;

  printf("The new date is %d-%d-%d; the time is %d:%d\n", DateTime.month,
         DateTime.day, DateTime.year, DateTime.hours, DateTime.minutes);

  rc = DosSetDateTime(&DateTime);    /* Address of d/t structure */

  printf("rc is %ld\n", rc);

}
```

#define INCL_DOSFILEMGR

APIRET DosSetDefaultDisk (ULONG ulDriveNumber)

DosSetDefaultDisk sets the specified drive as the default drive.

Parameters

ulDriveNumber (ULONG) — input

New default-drive number. The value 1 means drive A, 2 means drive B, 3 means drive C, and so on.

Returns

Return Code.

DosSetDefaultDisk returns the following values:

0	NO_ERROR
15	ERROR_INVALID_DRIVE

Related Functions

- DosQueryCurrentDir
- DosQueryCurrentDisk
- DosSetCurrentDir

Example Code

This example sets the specified drive as the default drive for the calling process.

```
#define INCL_DOSFILEMGR   /* File Manager values */
#include <os2.h>
#include <stdio.h>

ULONG   DriveNumber;   /* Default drive number */
APIRET  rc;            /* Return code */

   DriveNumber = 3;    /* Specify drive C */

   rc = DosSetDefaultDisk(DriveNumber);

   if (rc != 0)
     {
       printf("DosSetDefaultDisk error: return code = %ld", rc);
       return;
     }
```

DosSetExceptionHandler – Set Exception Handler

#define INCL_DOSEXCEPTIONS

APIRET DosSetExceptionHandler (PEXCEPTIONREGISTRATIONRECORD pppERegRec)

DosSetExceptionHandler registers an exception handler for the current thread.

Parameters

pppERegRec (PEXCEPTIONREGISTRATIONRECORD) – input

A pointer to the exception registration record that describes the exception handler to be registered. This exception registration record must be on the stack.

Returns

Return Code.

DosSetExceptionHandler returns the following values:

0 NO_ERROR

Remarks

DosSetExceptionHandler registers an exception handler for the current thread.

If you register more than one exception handler within the same procedure, each handler's exception registration record must have a lower storage address (a higher position on the stack) than the exception registration record of the previously installed handler.

For a detailed list of the system exceptions, see Appendix C, "System Exceptions" on page C-1.

Related Functions

- DosAcknowledgeSignalException
- DosEnterMustComplete
- DosExitMustComplete
- DosRaiseException
- DosSendSignalException
- DosSetSignalExceptionFocus
- DosUnsetExceptionHandler
- DosUnwindException

Example Code

This example registers an exception handler for the current thread. The example creates an exception registration record that contains a pointer to the desired exception handler. It is also possible to save the program state within the exception registration record, although the example does not show this. That program state will be passed to the exception handler when it is invoked.

Assume that the routine named *ExceptRoutine* is the exception handler that is to be registered. Assume that *ExceptRoutine* is local to the module containing this code.

```
#define INCL_DOSEXCEPTIONS   /* Exception values */
#include <os2.h>
#include <stdio.h>

ULONG _cdecl ExceptRoutine(PEXCEPTIONREPORTRECORD,
                           PEXCEPTIONREGISTRATIONRECORD,
                           PCONTEXTRECORD,
                           PVOID);

typedef struct  SysERegRec {
   PEXCEPTIONREGISTRATIONRECORD pLink;
   ULONG (_cdecl *pSysEH)(PEXCEPTIONREPORTRECORD,
                          PEXCEPTIONREGISTRATIONRECORD,
                          PCONTEXTRECORD,
                          PVOID);
} SYSEREGREC;

SYSEREGREC    RegRec;    /* Structure to pass to exception handler */
APIRET        rc;        /* Return code */

   RegRec.pLink = 0;     /* The DosSetExceptionHandler call will link */
                         /*   the exception registration record into  */
                         /*   the chain for the thread                */
   RegRec.pSysEH = ExceptRoutine;
                         /* Pointer to the exception handler */

   rc = DosSetExceptionHandler( (PEXCEPTIONREGISTRATIONRECORD)
                                    &RegRec);

   if (rc != 0)
     {
       printf("DosSetExceptionHandler error: return code = %ld", rc);
       return;
     }
```

DosSetFHState –
Set the State of a Specified File Handle

#define INCL_DOSFILEMGR

APIRET DosSetFHState (HFILE FileHandle, ULONG ulFileHandleState)

DosSetFHState sets the state of the specified file handle.

Parameters

FileHandle (HFILE) – input

File handle to be set.

ulFileHandleState (ULONG) – input

Contents of the open-mode word defined in a previous DosOpen function.

Bit	Description
15	(OPEN_FLAGS_DASD 0x00008000) This bit must be set to 0.
14	(OPEN_FLAGS_WRITE_THROUGH 0x00004000) Write-Through flag: 0 : Writes to the file may go through the system-buffer cache. 1 : Writes to the file may go through the system-buffer cache, but the data is written (the actual file I/O operation is completed) before a synchronous-write call returns. This state of the file defines it as a synchronous file. For synchronous files, this bit must be set, because the data must be written to the medium for synchronous-write operations. This flag bit is not inherited by child processes.
13	(OPEN_FAIL_ON_ERROR 0x00002000) Fail-Errors flag. Media I/O errors are handled as follows: 0 : Reported through the system critical-error handler. 1 : Reported directly to the caller by way of a return code. Media I/O errors generated through an IOCtl category 8 function are always reported directly to the caller by way of a return code. The Fail-Errors function applies only to non-IOCtl handle-based file I/O functions. This flag bit is not inherited by child processes.
12	(OPEN_FLAGS_NO_CACHE 0x00002000) Cache or No-Cache flag. The file is opened as follows: 0 : The disk driver should place data from I/O operations into cache. 1 : I/O operations to the file need not be done through the disk-driver cache. This bit is an advisory bit, and is used to advise file-system drivers and device drivers about whether the data should be cached. This bit, like the write-through bit, is a per-handle bit. This bit is not inherited by child processes.
11 – 8	These bits are reserved, and should be set to the values returned by DosQueryFHState in these positions.
7	(OPEN_FLAGS_NOINHERIT 0x00000080) Inheritance flag: 0 : File handle is inherited by a process created by DosExecPgm. 1 : File handle is private to the current process.

6–4	These bits must be set to 0. Any other values are invalid.
3	This bit is reserved, and should be set to the value returned by DosQueryFHState for this position.
2–0	These bits must be set to 0. Any other values are invalid.

Returns

Return Code.

DosSetFHState returns the following values:

0	NO_ERROR
6	ERROR_INVALID_HANDLE
87	ERROR_INVALID_PARAMETER

Remarks

The operating system does not guarantee the write order for multiple-sector write operations. If an application requires several sectors to be written in a specific order, the operator should issue the sectors as separate synchronous-write operations. Setting the Write-Through flag does not affect any previous write operation. That data can remain in the buffers.

When the application cannot handle a critical error that occurs, critical-error handling can be reset to the system. This is done by having DosSetFHState turn off the fail/errors bit, and then reissuing the I/O operation. The expected critical error recurs, and control is passed to the system critical-error handler. The precise time that the effect of this function is visible at the application level is unpredictable when asynchronous I/O operations are pending.

The file-handle-state bits set by this function can be queried by DosQueryFHState.

Named-Pipe Considerations

With DosSetFHState, the inheritance (I) bit and Write-Through (W) bit can be set or reset. Setting W to 1 prevents write-behind operations on remote pipes.

Related Functions

- DosClose
- DosDevIOCtl
- DosDupHandle
- DosExecPgm
- DosOpen
- DosQueryFHState

DosSetFHState –
Set the State of a Specified File Handle

Example Code

This example issues DosSetFHState to set the File Write-through attribute for an opened file. DosQueryFHState is issued first to obtain the file handle state bits. Assume that the appropriate file handle has been placed into *FileHandle* already.

```
#define INCL_DOSFILEMGR   /* File Manager values */
#include <os2.h>
#include <stdio.h>

HFILE   FileHandle;      /* File handle */
ULONG   FileHandleState; /* File handle state */
APIRET  rc;              /* Return code  */

   rc = DosQueryFHState(FileHandle, &FileHandleState);

   if (rc != 0)
     {
       printf("DosQueryFHState error: return code = %ld", rc);
       return;
     }

   FileHandleState |= OPEN_FLAGS_WRITE_THROUGH;
                      /* Indicate that writes to the file may */
                      /*   go through the file system buffer  */
                      /*   cache, but the sectors are to be   */
                      /*   written before any synchronous     */
                      /*   write call returns.  Only this one */
                      /*   file attribute is being changed by */
                      /*   the following DosSetFHState.       */

   rc = DosSetFHState(FileHandle,
                  FileHandleState);

   if (rc != 0)
     {
       printf("DosSetFHState error: return code = %ld", rc);
       return;
     }
```

DosSetFileInfo – Set File Information

#define INCL_DOSFILEMGR

APIRET DosSetFileInfo (HFILE FileHandle, ULONG ulFileInfoLevel, PVOID pFileInfoBuf, ULONG ulFileInfoBufSize)

DosSetFileInfo sets file information.

Parameters

FileHandle (HFILE) – input

File handle.

ulFileInfoLevel (ULONG) – input

Level of file information being set. A value of 1 or 2 can be specified, as follows:

Value	Description
1	(FIL_STANDARD) Level 1 file information
2	(FIL_QUERYEASIZE) Level 2 file information

The structures described in FileInfoBuf indicate the information being set for each of these levels.

pFileInfoBuf (PVOID) – input

Address of the storage area containing the structures for file information levels.

Level 1 File Information
FileInfoBuf contains the *FILESTATUS3* data structure where information is returned.

Level 2 File Information
FileInfoBuf contains an *EAOP2* data structure.

Level 2 sets a series of EA name/value pairs. On input, *FileInfoBuf* is an *EAOP2* data structure. *fpGEA2List* is ignored. *fpFEA2List* points to a data area where the relevant *FEA2* list is to be found. *oError* is ignored.

On output, *fpGEA2List* and *fpFEA2List* are unchanged. The area pointed to by *fpFEA2List* is unchanged. If an error occurred during the set, *oError* is the offset of the *FEA2* where the error occurred. The return code is the error code corresponding to the condition generating the error. If no error occurred, *oError* is undefined.

ulFileInfoBufSize (ULONG) – input

The length, in bytes, of *FileInfoBuf*.

Returns

Return Code.

DosSetFileInfo returns the following values:

0	NO_ERROR
1	ERROR_INVALID_FUNCTION
5	ERROR_ACCESS_DENIED
6	ERROR_INVALID_HANDLE
87	ERROR_INVALID_PARAMETER
122	ERROR_INSUFFICIENT_BUFFER
124	ERROR_INVALID_LEVEL
130	ERROR_DIRECT_ACCESS_HANDLE
254	ERROR_INVALID_EA_NAME

DosSetFileInfo — Set File Information

255	ERROR_EA_LIST_INCONSISTENT

Remarks

DosSetFileInfo is successful only when the file is opened for write access, and access by other processes is prevented by a deny-both sharing mode. If the file is already opened with conflicting sharing rights, any call to DosOpen will fail.

A value of 0 in the date and time components of a field does not change the field. For example, if both "last write date" and "last write time" are specified as 0 in the Level 1 information structure, then both attributes of the file are left unchanged. If either "last write date" or "last write time" are other than 0, both attributes of the file are set to the new values.

In the FAT file system, only the dates and times of the last write can be modified. Creation and last-access dates and times are not affected.

The last-modification date and time will be changed if the extended attributes are modified.

Related Functions

- DosClose
- DosEnumAttribute
- DosOpen
- DosQueryFileInfo
- DosQueryPathInfo
- DosResetBuffer
- DosSetFileSize
- DosSetPathInfo

Example Code

This example shows how DosSetFileInfo can change the attributes of a file. The example changes the date and time of last access to the file to the current date and time. DosQueryFileInfo is issued first to obtain the Level 1 file information block that includes the two desired access parameters. The two access parameters in the block are changed, and then DosSetFileInfo is issued to update the attributes within the file. Assume that the appropriate file handle has been placed into *FileHandle* already.

DosSetFileInfo can also change the extended attributes that are associated with a file. This example does not illustrate such a use of DosSetFileInfo.

```
#define INCL_DOSFILEMGR   /* File Manager values */
#include <os2.h>
#include <stdio.h>

HFILE       FileHandle;      /* File handle */
ULONG       FileInfoLevel;   /* File info data required */
FILESTATUS  FileInfoBuf;     /* File info buffer */
ULONG       FileInfoBufSize; /* File info buffer size */
DATETIME    DateTimeBuf;     /* Date/Time buffer */
APIRET      rc;              /* Return code */

   FileInfoLevel = 1;    /* Indicate that Level 1 information */
                         /*   is desired                      */

   FileInfoBufSize = sizeof(FILESTATUS);
                         /* Size of the buffer that will      */
                         /*   receive the Level 1 information */

   rc = DosQueryFileInfo(FileHandle, FileInfoLevel,
                         &FileInfoBuf, FileInfoBufSize);
```

```
                    /* Obtain a copy of the Level 1 */
                    /*   file information           */

if (rc != 0)
  {
    printf("DosQueryFileInfo error: return code = %ld", rc);
    return;
  }

rc = DosGetDateTime(&DateTimeBuf);
                    /* Get the current date and time */
                    /*   from the system             */

if (rc != 0)
  {
    printf("DosGetDateTime error: return code = %ld", rc);
    return;
  }

/*   Update the appropriate fields in the Level 1 */
/*     information block                          */

FileInfoBuf.fdateLastAccess.year = DateTimeBuf.year;
FileInfoBuf.fdateLastAccess.month = DateTimeBuf.month;
FileInfoBuf.fdateLastAccess.day = DateTimeBuf.day;
FileInfoBuf.ftimeLastAccess.hours = DateTimeBuf.hours;
FileInfoBuf.ftimeLastAccess.minutes = DateTimeBuf.minutes;
FileInfoBuf.ftimeLastAccess.twosecs = 0;

/*   Update the Level 1 information block associated   */
/*     with the file                                    */

rc = DosSetFileInfo(FileHandle, FileInfoLevel,
                    &FileInfoBuf, FileInfoBufSize);

if (rc != 0)
  {
    printf("DosSetFileInfo error: return code = %ld", rc);
    return;
  }
```

DosSetFileLocks –
Lock and Unlock a Range of an Open File

```
#define INCL_DOSFILEMGR

APIRET DosSetFileLocks (HFILE FileHandle, PFILELOCK ppUnLockRange,
                        PFILELOCK ppLockRange, ULONG ulTimeOut, ULONG ulFlags)
```

DosSetFileLocks locks and unlocks a range of an open file.

Parameters

FileHandle (HFILE) – input

File handle.

ppUnLockRange (PFILELOCK) – input

Address of the structure containing the offset and length of a range to be unlocked. The structure is as follows:

FileOffset (LONG) – input
The offset to the beginning of the range to be unlocked.

RangeLength (LONG) – input
The length of the range to be unlocked. A value of zero means that unlocking is not required.

ppLockRange (PFILELOCK) – input

Address of the structure containing the offset and length of a range to be locked. The structure is as follows:

FileOffset (LONG) – input
The offset to the beginning of the range to be locked.

RangeLength (LONG) – input
The length of the range to be locked. A value of zero means that locking is not required.

ulTimeOut (ULONG) – input

The maximum time, in milliseconds, that the process is to wait for the requested locks.

ulFlags (ULONG) – input

Flags that describe the action to be taken, as follows:

Bit	Description
31-2	Reserved flags
1	Atomic This bit defines a request for atomic locking. If this bit is set to 1 and the lock range is equal to the unlock range, an atomic lock occurs. If this bit is set to 1 and the lock range is not equal to the unlock range, an error is returned. If this bit is set to 0, then the lock may or may not occur atomically with the unlock.
0	Share This bit defines the type of access that other processes may have to the file range that is being locked. If this bit is set to 0 (the default), other processes have no access to the locked file range. The current process has exclusive access to the locked file range, which must not overlap any other locked file range. If this bit is set to 1, the current process and other processes have shared read-only access to the locked file range. A file range with shared access may overlap any

other file range with shared access, but must not overlap any other file range with exclusive access.

Returns

Return Code.

DosSetFileLocks returns the following values:

0	NO_ERROR
6	ERROR_INVALID_HANDLE
33	ERROR_LOCK_VIOLATION
36	ERROR_SHARING_BUFFER_EXCEEDED
87	ERROR_INVALID_PARAMETER
95	ERROR_INTERRUPT
174	ERROR_ATOMIC_LOCK_NOT_SUPPORTED
175	ERROR_READ_LOCKS_NOT_SUPPORTED

Remarks

DosSetFileLocks allows a process to lock and unlock a range in a file. The time during which a file range is locked should be short.

If the lock and unlock ranges are both zero, ERROR_LOCK_VIOLATION is returned to the caller.

If you only want to lock a file range, set the unlock file offset and the unlock range length to zero.

If you only want to unlock a file range, set the lock file offset and the lock range length to zero.

When the Atomic bit of *Flags* is set to 0, and DosSetFileLocks specifies a lock operation and an unlock operation, the unlock operation occurs first, and then the lock operation is performed. If an error occurs during the unlock operation, an error code is returned and the lock operation is not performed. If an error occurs during the lock operation, an error code is returned and the unlock remains in effect if it was successful.

The lock operation is atomic when all of these conditions are met:

- The Atomic bit is set to 1 in *Flags*
- The unlock range is the same as the lock range
- The process has shared access to the file range, and has requested exclusive access to it; or the process has exclusive access to the file range, and has requested shared access to it.

Some file system drivers (FSDs) may not support atomic lock operations. Versions of the operating system prior to OS/2 Version 2.00 do not support atomic lock operations. If the application receives the error code ERROR_ATOMIC_LOCK_NOT_SUPPORTED, the application should unlock the file range and then lock it using a non-atomic operation (with the atomic bit set to 0 in *Flags*). The application should also refresh its internal buffers before making any changes to the file.

If you issue DosClose to close a file with locks still in effect, the locks are released in no defined sequence.

If you end a process with a file open, and you have locks in effect in that file, the file is closed and the locks are released in no defined sequence.

The locked range can be anywhere in the logical file. Locking beyond the end of the file is not an error. A file range to be locked exclusively must first be cleared of any locked file subranges or overlapping locked file ranges.

If you repeat DosSetFileLocks for the same file handle and file range, then you duplicate access to the file range. Access to locked file ranges is not duplicated across DosExecPgm. The proper method of using locks is to attempt to lock the file range, and to examine the return value.

DosSetFileLocks –
Lock and Unlock a Range of an Open File

The following table shows the level of access granted when the accessed file range is locked with an exclusive lock or a shared lock. "Owner" refers to a process that owns the lock. "Non-owner" refers to a process that does not own the lock.

Action	Exclusive Lock	Shared Lock
Owner read	Success	Success
Non-owner read	Wait for unlock. Return error code after time-out.	Success
Owner write	Success	Wait for unlock. Return error code after time-out.
Non-owner write	Wait for unlock. Return error code after time-out.	Wait for unlock. Return error code after time-out.

If only locking is specified, DosSetFileLocks locks the specified file range using *LockRange*. If the lock operation cannot be accomplished, an error is returned, and the file range is not locked.

After the lock request is processed, a file range can be unlocked using the *UnLockRange* parameter of another DosSetFileLocks request. If unlocking cannot be accomplished, an error is returned.

Instead of denying read/write access to an entire file by specifying access and sharing modes with DosOpen requests, a process attempts to lock only the range needed for read/write access and examines the error code returned.

Once a specified file range is locked exclusively, read and write access by another process is denied until the file range is unlocked. If both unlocking and locking are specified by DosSetFileLocks, the unlocking operation is performed first, then locking is done.

Related Functions

- DosCancelLockRequest
- DosDupHandle
- DosExecPgm
- DosOpen

Example Code

This example opens a file, writes some data to the file, locks a block of the data, and then unlocks it.

```
#define INCL_DOSFILEMGR   /* File Manager values */
#include <os2.h>
#include <stdio.h>

#define OPEN_FILE 0x01
#define CREATE_FILE 0x10
#define FILE_ARCHIVE 0x20
#define FILE_EXISTS OPEN_FILE
#define FILE_NOEXISTS CREATE_FILE
#define DASD_FLAG 0
#define INHERIT 0x80
#define WRITE_THRU 0
#define FAIL_FLAG 0
#define SHARE_FLAG 0x10
#define ACCESS_FLAG 0x02

#define FILE_NAME "test.dat"
#define FILE_SIZE 800L
#define FILE_ATTRIBUTE FILE_ARCHIVE
#define EABUF 0L
#define NULL_RANGE 0L

#define LOCK_FLAGS 0
```

```
HFILE   FileHandle;
ULONG   Wrote;
ULONG   Action;
PSZ     FileData[100];
ULONG   LockTimeout = 2000;
APIRET  rc;                /* Return code */

struct LockStrc
   {
   long Offset;
   long Range;
   } Area;

int i;

   Action = 2;
   strcpy(FileData, "Data...");
   Area.Offset = 4;
   Area.Range = 100;

   rc = DosOpen(FILE_NAME,              /* File path name */
                &FileHandle,            /* File handle */
                &Action,                /* Action taken */
                FILE_SIZE,              /* File primary allocation */
                FILE_ATTRIBUTE,         /* File attribute */
                FILE_EXISTS | FILE_NOEXISTS,   /* Open function type */
                DASD_FLAG | INHERIT |        /* Open mode of file */
                WRITE_THRU | FAIL_FLAG |
                SHARE_FLAG | ACCESS_FLAG,
                EABUF);              /* No extended attributes */

   if (rc != 0)                 /* If open failed */
     {
       printf("DosOpen error: return code = %ld",rc);
       return;
     }

      for(i=0; i<200; ++i)
         DosWrite(FileHandle,           /* File handle */
                  FileData,             /* User buffer */
                  sizeof(FileData),     /* Buffer length */
                  &Wrote);              /* Bytes written */

      rc = DosSetFileLocks(FileHandle,         /* File handle */
                           NULL_RANGE,         /* Unlock range */
                           (PFILELOCK) &Area,  /* Lock range */
                           &LockTimeout,       /* Lock time-out */
                           LOCK_FLAGS);        /* Request flags */

      if (rc != 0)              /* If lock failed */
        {
          printf("DosSetFileLocks lock error: return code = %ld",rc);
          return;
        }

      rc = DosSetFileLocks(FileHandle,         /* File handle */
                           (PFILELOCK) &Area,  /* Unlock range */
                           NULL_RANGE,         /* Lock range */
                           &LockTimeout,       /* Lock time-out */
                           LOCK_FLAGS);        /* Request flags */

      if (rc != 0)              /* If unlock failed */
        {
```

```
    printf("DosSetFileLocks unlock error: return code = %ld",rc);
    return;
}
```

DosSetFilePtr – Move the Read/Write Pointer

#define INCL_DOSFILEMGR

```
APIRET DosSetFilePtr (HFILE FileHandle, LONG lDistance, ULONG ulMoveType,
                      PULONG pNewPointer)
```

DosSetFilePtr moves the read/write pointer according to the type of move specified.

Parameters

FileHandle (HFILE) – input

The handle returned by a previous DosOpen function.

lDistance (LONG) – input

The signed distance (offset) to move, in bytes.

ulMoveType (ULONG) – input

The method of moving. Specifies a location in the file from where the *Distance* to move the read/write pointer starts. The values and their meanings are as follows:

Value	Definition
0	(FILE_BEGIN) Move the pointer from the beginning of the file.
1	(FILE_CURRENT) Move the pointer from the current location of the read/write pointer.
2	(FILE_END) Move the pointer from the end of the file. Use this method to determine a file's size.

pNewPointer (PULONG) – output

Address of the new pointer location.

Returns

Return Code.

DosSetFilePtr returns the following values:

0	NO_ERROR
1	ERROR_INVALID_FUNCTION
6	ERROR_INVALID_HANDLE
132	ERROR_SEEK_ON_DEVICE
131	ERROR_NEGATIVE_SEEK
130	ERROR_DIRECT_ACCESS_HANDLE

Remarks

The read/write pointer in a file is a signed 32-bit number. A negative value for *Distance* moves the pointer backward in the file; a positive value moves it forward. DosSetFilePtr cannot be used to move to a negative position in the file.

DosSetFilePtr cannot be used for a character device or pipe.

DosSetFilePtr – Move the Read/Write Pointer

Related Functions

- DosOpen
- DosRead
- DosSetFileSize
- DosWrite

Example Code

This example opens the file test.dat, writes some data, and resets the file pointer to the beginning of the file.

```
#define INCL_DOSFILEMGR   /* File Manager values */
#include <os2.h>
#include <stdio.h>

#define OPEN_FILE 0x01
#define CREATE_FILE 0x10
#define FILE_ARCHIVE 0x20
#define FILE_EXISTS OPEN_FILE
#define FILE_NOEXISTS CREATE_FILE
#define DASD_FLAG 0
#define INHERIT 0x80
#define WRITE_THRU 0
#define FAIL_FLAG 0
#define SHARE_FLAG 0x10
#define ACCESS_FLAG 0x02

#define FILE_NAME "test.dat"
#define FILE_SIZE 800L
#define FILE_ATTRIBUTE FILE_ARCHIVE
#define EABUF 0L
#define MOVE_DIST 0L

HFILE   FileHandle;
ULONG   Wrote;              /* Bytes written */
ULONG   Action;             /* Action taken by DosOpen */
ULONG   Local;              /* New file pointer location */
UCHAR   FileData[100];      /* Data to write */
APIRET  rc;                 /* Return code */


   Action = 2;
   strcpy(FileData, "Data...");

   rc = DosOpen(FILE_NAME,              /* File path name */
                &FileHandle,            /* File handle */
                &Action,                /* Action taken */
                FILE_SIZE,              /* File primary allocation */
                FILE_ATTRIBUTE,         /* File attribute */
                FILE_EXISTS | FILE_NOEXISTS,      /* Open function type */
                DASD_FLAG | INHERIT |            /* Open mode of the file */
                WRITE_THRU | FAIL_FLAG |
                SHARE_FLAG | ACCESS_FLAG,
                EABUF);                 /* No extended attributes */

   if (rc != 0)
     {
       printf("DosOpen error: return code = %ld",rc);
       return;
     }

   rc = DosWrite(FileHandle,          /* File handle */
```

```
                (PVOID) FileData,     /* User buffer */
                sizeof(FileData),     /* Buffer length */
                &Wrote);              /* Bytes written */

if (rc != 0)
  {
    printf("DosWrite error: return code = %ld",rc);
    return;
  }

rc = DosSetFilePtr(FileHandle,  /* File handle */
                   MOVE_DIST,   /* Distance to move in bytes */
                   FILE_BEGIN,  /* Method of moving */
                   &Local);     /* New pointer location */

if (rc != 0)
  {
    printf("DosSetFilePtr error: return code = %ld",rc);
    return;
  }
```

DosSetFileSize –
Change the Size of a File

```
#define INCL_DOSFILEMGR

APIRET DosSetFileSize (HFILE FileHandle, ULONG ulFileSize)
```

DosSetFileSize changes the size of a file.

Parameters

FileHandle (HFILE) – input

Handle of the file whose size to be changed.

ulFileSize (ULONG) – input

New size, in bytes, of the file.

Returns

Return Code.

DosSetFileSize returns the following values:

0	NO_ERROR
5	ERROR_ACCESS_DENIED
6	ERROR_INVALID_HANDLE
26	ERROR_NOT_DOS_DISK
33	ERROR_LOCK_VIOLATION
87	ERROR_INVALID_PARAMETER
112	ERROR_DISK_FULL

Remarks

When DosSetFileSize is issued, the file must be open in a mode that allows write access.

The size of the open file can be truncated or extended. If the file size is being extended, the file system tries to allocate additional bytes in a contiguous (or nearly contiguous) space on the medium. The values of the new bytes are undefined.

Related Functions

- DosOpen
- DosQueryFileInfo
- DosQueryPathInfo

Example Code

This example changes the size of a file. Assume that the handle of the file has been placed into *FileHandle* already. Assume that the file has been opened in a writeable manner already. In this example, assume that the file is being extended. The program logic, however, would be the same if the file were being truncated. In this example, the values of the bytes that are being added to the file are undefined.

```
#define INCL_DOSFILEMGR   /* File Manager values */
#include <os2.h>
#include <stdio.h>

HFILE   FileHandle;     /* File handle */
ULONG   FileSize;       /* File's new size */
APIRET  rc;             /* Return code */

   FileSize = 20000;    /* Indicate that the new file size */
                        /*   should be 20000 bytes         */

   rc = DosSetFileSize(FileHandle, FileSize);

   if (rc != 0)
     {
       printf("DosSetFileSize error: return code = %ld", rc);
       return;
     }
```

DosSetFSInfo –
Set Information for a File System Device

```
#define INCL_DOSFILEMGR
```

APIRET DosSetFSInfo (ULONG ulDriveNumber, ULONG ulFSInfoLevel, PVOID pFSInfoBuf, ULONG ulFSInfoBufSize)

DosSetFSInfo sets information for a file system device.

Parameters

ulDriveNumber (ULONG) – input

Logical drive number. Zero means the default drive, 1 means drive A, 2 means drive B, 3 means drive C, and so on. This represents the file system driver (FSD) for the media currently in that drive. A value of hex FFFF means that *FSInfoBuf* contains the ASCIIZ path name of the FSD.

ulFSInfoLevel (ULONG) – input

Level of file information to be set. Only a value of 2 may be specified.

pFSInfoBuf (PVOID) – input

Address of the storage area where the system gets the new file system information.

Level 2 Information

Level 2 information is specified in the following format:

Byte	Description
1	Length, in bytes, of the volume label (null not included)
2 – N	Volume label. This is an ASCIIZ string.

ulFSInfoBufSize (ULONG) – input

The length, in bytes, of *FSInfoBuf*.

Returns

Return Code.

DosSetFSInfo returns the following values:

0	NO_ERROR
15	ERROR_INVALID_DRIVE
82	ERROR_CANNOT_MAKE
122	ERROR_INSUFFICIENT_BUFFER
123	ERROR_INVALID_NAME
124	ERROR_INVALID_LEVEL
154	ERROR_LABEL_TOO_LONG

Remarks

Trailing blanks supplied at the time the volume label is defined are not returned by DosQueryFSInfo.

File-system information can be set only if the volume is opened in a mode that allows write access.

Related Functions

- DosQueryCurrentDisk
- DosQueryFSInfo
- DosQuerySysInfo
- DosSetDefaultDisk

Example Code

This example shows how a thread can change the volume label for a specified logical drive.

```
#define INCL_DOSFILEMGR   /* File Manager values */
#include <os2.h>
#include <stdio.h>

ULONG       DriveNumber;    /* Drive number */
ULONG       FSInfoLevel;    /* File system data type */
VOLUMELABEL FSInfoBuf;      /* File system info buffer */
ULONG       FSInfoBufSize;  /* File system info buffer size */
APIRET      rc;             /* Return code */

   DriveNumber = 4;     /* For this example, specify drive D: */

   FSInfoLevel = FSIL_VOLSER;
                        /* Indicate that the caller wants to   */
                        /*   change the volume label for the   */
                        /*   specified drive                   */

   strcpy(FSInfoBuf.szVolLabel,"Utilities");
                        /* The new volume label for logical */
                        /*   drive D:                       */

   FSInfoBuf.cch = (BYTE)strlen(FSInfoBuf.szVolLabel);
                        /* Length of the volume label string  */
                        /*   within the VOLUMELABEL structure */

   FSInfoBufSize = sizeof(VOLUMELABEL);
                        /* Size of the entire VOLUMELABEL */
                        /*   structure                    */

   rc = DosSetFSInfo(DriveNumber, FSInfoLevel, &FSInfoBuf,
                          FSInfoBufSize);

   if (rc != 0)
     {
       printf("DosSetFSInfo error: return code = %ld", rc);
       return;
     }
```

DosSetMaxFH –
Define the Maximum Number of File Handles

#define INCL_DOSFILEMGR

APIRET DosSetMaxFH (ULONG ulNumberHandles)

DosSetMaxFH defines the maximum number of file handles for the calling process.

Parameters

ulNumberHandles (ULONG) – input

Total number of file handles to be provided.

Returns

Return Code.

DosSetMaxFH returns the following values:

0	NO_ERROR
8	ERROR_NOT_ENOUGH_MEMORY
87	ERROR_INVALID_PARAMETER

Remarks

The operating system initially allocates 20 file handles to a process. This is the recommended number for an application. However, if the system limit has not been reached, this amount can be increased with DosSetMaxFH. When DosSetMaxFH is issued, all open file handles are preserved.

Related Functions

- DosDupHandle
- DosOpen
- DosSetRelMaxFH

Example Code

This example sets the maximum number of file handles for the calling process.

```
#define INCL_DOSFILEMGR   /* File Manager values */
#include <os2.h>
#include <stdio.h>

ULONG   NumberHandles; /* Number of file handles */
APIRET  rc;            /* Return code */

   NumberHandles = 30; /* Set maximum number of file handles */
                       /*   to 30 for the calling process    */

   rc = DosSetMaxFH(NumberHandles);

   if (rc != 0)
     {
       printf("DosSetMaxFH error: return code = %ld", rc);
       return;
     }
```

DosSetMem – Set a Range of Pages within a Memory Object

#define INCL_DOSMEMMGR

APIRET DosSetMem (PVOID pBaseAddress, ULONG ulRegionSize, ULONG ulAttributeFlags)

DosSetMem commits or decommits a range of pages within a memory object, or alters their access protection.

Parameters

pBaseAddress (PVOID) – input

The base address of the range of pages whose attributes are to be changed.

ulRegionSize (ULONG) – input

A value specifying the size, in bytes, of the region whose attributes are to be changed. The size is rounded up to include all pages addressed by the requested base address and size.

ulAttributeFlags (ULONG) – input

A set of flags specifying commitment or decommitment, and desired access protection, for the specified range of pages.

Commit Type

- If the PAG_COMMIT bit (0x00000010) is set, the specified range of pages is to be committed.
- If the PAG_DECOMMIT bit (0x00000020) is set, the specified range of pages is to be decommitted.
- If neither is specified, no change in commitment is made.

Desired Access Protection

- If the PAG_EXECUTE bit (0x00000004) is set, execute access to the committed range of pages is desired.
- If the PAG_READ bit (0x00000001) is set, read access to the committed range of pages is desired.
- If the PAG_WRITE bit (0x00000002) is set, write access to the committed range of pages is desired.
- If the PAG_GUARD bit (0x00000008) is set, access to the committed range of pages causes a "guard page entered" condition to be raised in the subject process.
- If the PAG_DEFAULT bit (0x00000400) is set, the access protection assigned to the committed range of pages is the access protection specified when the object was allocated in the address space of the requesting process.
- If the PAG_DECOMMIT bit is not set, then the PAG_DEFAULT bit or at least one of the bits PAG_READ, PAG_WRITE, or PAG_EXECUTE must be specified.
- All other bits must be clear.

Returns

Return Code.

DosSetMem returns the following values:

0	NO_ERROR
5	ERROR_ACCESS_DENIED
8	ERROR_NOT_ENOUGH_MEMORY
87	ERROR_INVALID_PARAMETER
95	ERROR_INTERRUPT

DosSetMem –
Set a Range of Pages within a Memory Object

212	ERROR_LOCKED
487	ERROR_INVALID_ADDRESS
32798	ERROR_CROSSES_OBJECT_BOUNDARY

Remarks

DosSetMem can be used to commit or decommit a range of previously allocated pages in either a private or shared memory object. It also can be used to create a sparse population of committed private or shared pages within a memory object. DosSetMem can also change the access protection applied to already-committed pages within a memory object.

Each page in the virtual-address space of the process is either free, private, or shared.

The virtual address for free pages is not reserved, not committed, and not accessible. An attempt to commit or decommit a free page results in the return of an error.

The virtual address for pages in a private or shared memory object is reserved during the allocation of the memory object. Each page within a memory object can be in one of two states:

1. Committed: These pages have allocated backing storage, with access controlled by a protection code. A committed page in a private memory object may be decommitted; a committed page in a shared memory object may not be decommitted. An attempt to commit a previously committed page results in the return of an error.
2. Decommitted: These pages are not committed and are not accessible. A decommitted page may be committed if the backing storage is available. An attempt to decommit a previously decommitted page results in the return of an error.

The commitment of a reserved page in a shared object causes the page to be committed in the context of each process sharing the shared memory object.

Any access protection can be applied to committed private pages. Decommitted pages are given an access protection of "no access".

When pages are committed, they are backed by demand pages. The first attempt to read or write the page causes a page of zeros to be created.

Decommitting a private page causes the backing storage for the page to be released.

Setting the protection on a range of previously committed pages causes the old access protection to be replaced by the desired access protection. The access protection can be set only on committed pages.

Setting the access protection to PAG_GUARD causes a range of guard pages to be established. If access to this range of pages is attempted, an access violation (page fault) is generated. This fault sets the protection of the accessed page to the desired access protection, and generates a condition that signifies that a guard page has been entered. This capability is intended to provide automatic stack checking. It can also be used to separate other data structures when appropriate.

If a failure occurs, the attributes are not changed on any pages, and an appropriate error code is returned.

As each page is considered for protection, its state is determined. If the state of the page is not committed, or is not being committed, an appropriate error code is returned. Otherwise, the new protection of the page is set.

With the Intel 80386 processor, execute and read access are equivalent. Also, write access implies both read and execute access.

DosSetMem – Set a Range of Pages within a Memory Object

Related Functions

- DosAllocMem
- DosAllocSharedMem
- DosQueryMem

Example Code

This example commits a region of two pages within a previously allocated memory object, and sets read-only access rights for the region. Assume that the base address for the DosSetMem function was previously obtained by the process.

```
#define  INCL_DOSMEMMGR   /* Memory Manager values */
#include <os2.h>
#include <stdio.h>
#include <bsememf.h>   /* Get flags for memory management */

PVOID   BaseAddress;     /* Pointer to the range of pages
                            whose attributes are to be changed */
ULONG   RegionSize;      /* Size, in bytes, of the region whose
                            attributes are to be changed */
ULONG   AttributeFlags;  /* Flags describing characteristics
                            of the specified range of pages */
APIRET  rc;              /* Return code */


   RegionSize = 8192;    /* Specify a two-page region */

   AttributeFlags = PAG_COMMIT | PAG_READ;
                         /* Commit the specified region, and */
                         /*   set read-only access rights to */
                         /*   the region                     */

   rc = DosSetMem(BaseAddress, RegionSize, AttributeFlags);

   if (rc != 0)
     {
       printf("DosSetMem error: return code = %ld", rc);
       return;
     }
```

DosSetNPHState – Set Named Pipe Handle State

#define INCL_DOSNMPIPES

APIRET DosSetNPHState (HPIPE hpipeHandle, ULONG ulPipeHandleState)

DosSetNPHState resets the blocking mode and the read mode of a named pipe.

Parameters

hpipeHandle (HPIPE) – input

The named-pipe handle to reset. (The server handle is returned by DosCreateNPipe; the client handle is returned by DosOpen)

ulPipeHandleState (ULONG) – input

The named-pipe handle state. This parameter consists of the following bit fields:

Bit	Description
31 – 16	Reserved.
15	Blocking mode. The blocking mode is defined as either "blocking" or "nonblocking," as follows: 0 = (NP_WAIT) (0x0000) Blocking mode: DosRead and DosWrite wait if no data is available. 1 = (NP_NOWAIT) (0x8000) Nonblocking mode: DosRead and DosWrite return immediately if no data is available. DosRead normally blocks (waits) until at least partial data can be returned. DosWrite blocks by default until all of the requested bytes have been written. Nonblocking mode changes this behavior as follows: DosRead returns immediately with a value of zero for *BytesRead* if no data is available. DosWrite returns a value of zero for *BytesWritten* if there is not enough buffer space available in the pipe; otherwise, the entire data area is transferred.
14 – 10	Reserved.
9 – 8	Read Mode. The read mode is defined as follows: 00 = (NP_READMODE_BYTE) (0x0000) Byte-read mode: Read the pipe as a byte stream. 01 = (NP_READMODE_MESSAGE) (0x0100) Message-stream mode: Read the pipe as a message stream.
7 – 0	Reserved, must be set to 0.

Returns

Return Code.

DosSetNPHState returns the following values:

0	NO_ERROR
87	ERROR_INVALID_PARAMETER
230	ERROR_BAD_PIPE
231	ERROR_PIPE_BUSY
233	ERROR_PIPE_NOT_CONNECTED

DosSetNPHState – Set Named Pipe Handle State

Remarks

DosSetNPHState resets the blocking mode and the read mode of a named pipe. Both the blocking mode and the read mode must be specified. However, the read mode cannot be changed if the pipe is a byte pipe. (Byte pipes can be read only as byte streams.) In addition, the blocking mode cannot be changed to nonblocking if another thread is currently blocked on an I/O request to the same end of the pipe.

Related Functions

- DosCallNPipe
- DosConnectNPipe
- DosCreateNPipe
- DosDisConnectNPipe
- DosPeekNPipe
- DosQueryNPHState
- DosQueryNPipeInfo
- DosQueryNPipeSemState
- DosSetNPipeSem
- DosTransactNPipe
- DosWaitNPipe
- DosClose
- DosDupHandle
- DosOpen
- DosRead
- DosResetBuffer
- DosWrite

Example Code

This example modifies several of the control parameters that are associated with a named pipe. The caller of DosSetNPHState can change the blocking characteristics of its end of a named pipe, and whether the pipe is read as a byte stream or as a message stream. Assume that a previous call to DosOpen or DosCreateNPipe provided the named pipe handle that is contained in *Handle*.

```
#define INCL_DOSNMPIPES   /* Named-pipe values */
#include <os2.h>
#include <stdio.h>

HPIPE   Handle;           /* Pipe handle */
ULONG   PipeHandleState;  /* Pipe-handle state */
APIRET  rc;               /* Return code */

   PipeHandleState = 0;   /* Indicate that pipe Reads/Writes  */
                          /*   will block if no data is       */
                          /*   available, and that the pipe   */
                          /*   is to be read as a byte stream */


   rc = DosSetNPHState(Handle, PipeHandleState);

   if (rc != 0)
     {
       printf("DosSetNPHState error: return code = %ld",rc);
       return;
     }
```

DosSetNPipeSem –
Set Named Pipe Semaphore

#define INCL_DOSNMPIPES

APIRET DosSetNPipeSem (HPIPE hpipeHandle, HSEM hsemSemHandle, ULONG ulKeyHandle)

DosSetNPipeSem attaches a shared event semaphore to a local named pipe.

Parameters

hpipeHandle (HPIPE) – input

The named-pipe handle to which a semaphore is to be attached. (The server handle is returned by DosCreateNPipe; the client handle is returned by DosOpen.)

hsemSemHandle (HSEM) – input

The handle of an event semaphore or a multiple-wait (muxwait) semaphore that is posted when the pipe (identified by *hpipeHandle*) has either data to be read or write space available.

ulKeyHandle (ULONG) – input

A key value that distinguishes events arriving on different named pipes that are attached to the same semaphore.

Returns

Return Code.

DosSetNPipeSem returns the following values:

0	NO_ERROR
1	ERROR_INVALID_FUNCTION
6	ERROR_INVALID_HANDLE
87	ERROR_INVALID_PARAMETER
187	ERROR_SEM_NOT_FOUND
230	ERROR_BAD_PIPE
233	ERROR_PIPE_NOT_CONNECTED
292	ERROR - WRONG_TYPE

Remarks

DosSetNPipeSem works only for local pipes. If an attempt is made to attach a semaphore to a remote pipe, ERROR_INVALID_FUNCTION is returned.

If a semaphore is already attached to the specified handle, DosSetNPipeSem replaces the existing semaphore with the new one.

Related Functions

- DosCallNPipe
- DosConnectNPipe
- DosCreateNPipe
- DosDisConnectNPipe
- DosPeekNPipe
- DosQueryNPHState
- DosQueryNPipeInfo
- DosQueryNPipeSemState
- DosSetNPHState
- DosTransactNPipe
- DosWaitNPipe
- DosClose
- DosDupHandle

- DosOpen
- DosRead
- DosResetBuffer
- DosWrite
- DosCreateEventSem
- DosCloseMuxWaitSem
- DosWaitEventSem
- DosWaitMuxWaitSem

Example Code

This example associates a system semaphore with a named pipe. Associating a semaphore with a named pipe increases the flexibility with which a process can wait for events that are associated with the pipe. In this example, assume that a previous call to DosOpen or DosCreateNPipe provided the named pipe handle that is contained in *Handle*. Assume that the handle of the system semaphore also was obtained previously.

```
#define INCL_DOSNMPIPES   /* Named-pipe values */
#include <os2.h>
#include <stdio.h>

#define THIRD_KEY 3     /* A unique key that will distinguish */
                        /*   the named pipe to which the      */
                        /*   semaphore is attached            */

HPIPE   Handle;         /* Pipe handle */
HSEM    SemHandle;      /* Semaphore handle */
ULONG   KeyHandle;      /* Key value */
APIRET  rc;             /* Return code */

   KeyHandle = THIRD_KEY;

   rc = DosSetNPipeSem(Handle, SemHandle, KeyHandle);

   if (rc != 0)
     {
       printf("DosSetNPipeSem error: return code = %ld",rc);
       return;
     }
```

DosSetPathInfo –
Set Information for a File or Directory

#define INCL_DOSFILEMGR

APIRET DosSetPathInfo (PSZ pszPathName, ULONG ulFileInfoLevel, PVOID pFileInfoBuf, ULONG ulFileInfoSize, ULONG ulPathInfoFlags)

DosSetPathInfo sets information for a file or directory.

Parameters

pszPathName (PSZ) – input

Address of the ASCIIZ full path name of the file or subdirectory. Global file-name characters are not permitted.

DosQuerySysInfo is called by an application during initialization to determine the maximum path length allowed by the operating system.

ulFileInfoLevel (ULONG) – input

The level of file directory information being defined. A value of 1 or 2 can be specified, as follows:

Value	Description
1	(FIL_STANDARD) Level 1 file information
2	(FIL_QUERYEASIZE) Level 2 file information

The structures described in *FileInfoBuf* indicate the information being set for each of these levels.

pFileInfoBuf (PVOID) – input

Address of the storage area containing the file information being set.

Level 1 File Information
FileInfoBuf contains the FILESTATUS3 data structure where information is returned.

Level 2 File Information
FileInfoBuf contains an EAOP2 structure.

Level 2 sets a series of extended attribute (EA) name/value pairs. On input, *FileInfoBuf* contains an EAOP2 data structure. fpGEA2List is ignored. fpFEA2List points to a data area where the relevant FEA2 list is to be found. oError is ignored. The FEA2 data structures must be aligned on a doubleword boundary. Each oNextEntryOffset field must contain the number of bytes from the beginning of the current entry to the beginning of the next entry in the FEA2 list. The oNextEntryOffset field in the last entry of the FEA2 list must be zero.

On output, fpGEA2List and fpFEA2List are unchanged. The area that fpFEA2List points to is unchanged. If an error occurred during the set, oError is the offset of the FEA2 entry where the error occurred. The return code is the error code corresponding to the condition that caused the error. If no error occurred, oError is undefined.

ulFileInfoSize (ULONG) – input

The length, in bytes, of *FileInfoBuf*.

ulPathInfoFlags (ULONG) – input

Contains information on how the set operation is to be performed. If *PathInfoFlags* is hex 00000010 (DSPI_WRTTHRU), then all the information, including extended attributes (EAs), must be written to the disk before returning to the application. This guarantees that the EAs have been written to the disk. All other bits are reserved, and must be zero.

DosSetPathInfo – Set Information for a File or Directory

Returns

Return Code.

DosSetPathInfo returns the following values:

0	NO_ERROR
32	ERROR_SHARING_VIOLATION
87	ERROR_INVALID_PARAMETER
124	ERROR_INVALID_LEVEL
206	ERROR_FILENAME_EXCED_RANGE
122	ERROR_INSUFFICIENT_BUFFER
254	ERROR_INVALID_EA_NAME
255	ERROR_EA_LIST_INCONSISTENT

Remarks

To use DosSetPathInfo to set any level of file information for a file or subdirectory, a process must have exclusive write access to the closed file object. Thus, if the file object is already accessed by another process, any call to DosSetPathInfo will fail.

A value of 0 in the date and time components of a field causes that field to be left unchanged. For example, if both "last write date" and "last write time" are specified as 0 in the Level 1 information structure, then both attributes of the file are left unchanged. If either "last write date" or "last write time" are other than 0, then both attributes of the file are set to the new values.

For data integrity purposes, the Write-Through bit in *PathInfoFlags* should be used only to write the extended attributes to the disk immediately, instead of caching them and writing them later. Having the Write-Through bit set constantly can degrade performance.

The last-modification date and time will be changed if the extended attributes are modified.

Related Functions

- DosEnumAttribute
- DosQueryFileInfo
- DosQueryPathInfo
- DosQuerySysInfo
- DosSetFileInfo

Example Code

This example shows how DosSetPathInfo can be used to change the attributes of a file. The example changes the date and time of last access to the file to the current date and time. DosQueryPathInfo is issued first to obtain the Level 1 file information block that includes the two desired access parameters. The two access parameters within the block are changed, and then DosSetPathInfo is issued to update the attributes within the file.

DosSetPathInfo is similar to DosSetFileInfo. DosSetPathInfo accepts a path name as an input parameter. DosSetFileInfo accepts a file handle of an open file as an input parameter. Both functions can modify the same classes of file information. An important difference between them is that DosSetPathInfo can be used to modify files and directories, while DosSetFileInfo can only be used to modify open files. DosSetPathInfo can only operate on closed files.

DosSetPathInfo can also be used to change the extended attributes that are associated with a file. This example does not illustrate such a use of DosSetPathInfo.

```
#define INCL_DOSFILEMGR   /* File Manager values */
#include <os2.h>
#include <stdio.h>
```

DosSetPathInfo —
Set Information for a File or Directory

```
UCHAR        PathName[60];    /* File or directory path name string */
ULONG        FileInfoLevel;   /* File info data required */
FILESTATUS3  FileInfoBuf;     /* File info buffer */
ULONG        FileInfoSize;    /* Info buffer size */
ULONG        PathInfoFlags;   /* Control flags */
DATETIME     DateTimeBuf;     /* Date/Time buffer */
APIRET       rc;              /* Return code */

   strcpy(PathName,"D:\\TOOLS\\UTIL\\DIR1");
                        /* Name of the specified directory */

   FileInfoLevel = 1;   /* Indicate that Level 1 information */
                        /*   is desired                      */

   FileInfoSize = sizeof(FILESTATUS3);
                        /* Size of the buffer that will */
                        /*   receive the Level 1       */
                        /*   information               */

   rc = DosQueryPathInfo(PathName, FileInfoLevel, &FileInfoBuf,
                         FileInfoSize);
                        /* Obtain a copy of the Level 1 */
                        /*   file information          */
   if (rc != 0)
     {
       printf("DosQueryPathInfo error: return code = %ld", rc);
       return;
     }

   rc = DosGetDateTime(&DateTimeBuf);
                        /* Get the current date and time */
                        /*   from the system             */
   if (rc != 0)
     {
       printf("DosGetDateTime error: return code = %ld", rc);
       return;
     }

   /*   Update the appropriate fields in the Level 1 */
   /*     information block                          */

   FileInfoBuf.fdateLastAccess.year = DateTimeBuf.year;
   FileInfoBuf.fdateLastAccess.month = DateTimeBuf.month;
   FileInfoBuf.fdateLastAccess.day = DateTimeBuf.day;
   FileInfoBuf.ftimeLastAccess.hours = DateTimeBuf.hours;
   FileInfoBuf.ftimeLastAccess.minutes = DateTimeBuf.minutes;
   FileInfoBuf.ftimeLastAccess.twosecs = 0;

   /*   Update the Level 1 information block associated   */
   /*     with the file                                    */

   PathInfoFlags = 0;   /* Since extended attributes are not  */
                        /*   being updated, the function can  */
                        /*   return before the data is        */
                        /*   actually written to the disk     */

   rc = DosSetPathInfo(PathName, FileInfoLevel, &FileInfoBuf,
                       FileInfoSize, PathInfoFlags);

   if (rc != 0)
     {
       printf("DosSetPathInfo error: return code = %ld", rc);
       return;
     }
```

DosSetPriority – Change the Base Priority

#define INCL_DOSPROCESS

```
APIRET DosSetPriority (ULONG ulScope, ULONG ulPriorityClass, LONG lPriorityDelta,
                       ULONG ulID)
```

DosSetPriority changes the base priority of a child process or thread in the current process.

Parameters

ulScope (ULONG) – input

The extent of the priority change. The values of this field are as follows:

Value	Definition
0	(PRTYS_PROCESS): All the threads of any process.
1	(PRTYS_PROCESSTREE): All the threads of a process and any descendants. The indicated process must be the current process or a process created by the current process. Detached processes may not be specified. The indicated process may have terminated.
2	(PRTYS_THREAD): A single thread of the current process.

ulPriorityClass (ULONG) – input

Priority class of a process. The values of this field are as follows:

Value	Definition
0	(PRTYC_NOCHANGE): No change, leave as is
1	(PRTYC_IDLETIME): Idle-time
2	(PRTYC_REGULAR): Regular
3	(PRTYC_TIMECRITICAL): Time-critical
4	(PRTYC_FOREGROUNDSERVER): Server

lPriorityDelta (LONG) – input

Change to apply to the current base priority level of the process. This value must range from –31 (PRTYD_MINIMUM) to +31 (PRTYD_MAXIMUM).

ulID (ULONG) – input

A process identifier (*Scope* = 0 or 1) or a thread identifier (*Scope* = 2). If this operand is equal to zero, the current process or thread is assumed.

Returns

Return Code.

DosSetPriority returns the following values:

0	NO_ERROR
303	ERROR_INVALID_PROCID
304	ERROR_INVALID_PDELTA
305	ERROR_NOT_DESCENDANT
307	ERROR_INVALID_PCLASS
308	ERROR_INVALID_SCOPE
309	ERROR_INVALID_THREADID

DosSetPriority –
Change the Base Priority

Remarks

DosSetPriority allows a process to change the priority of all the threads of any process, or all the threads of the current process or a child process, as well as any descendants. It also allows a process to change the priority of a single thread within the current process.

When a process changes the priority of threads in other processes, only default priorities are changed.

Related Functions

- DosEnterCritSec
- DosGetInfoBlocks

Example Code

This example changes the base priority of another process. Assume that the target process ID has been placed into *ID* already.

```
#define INCL_DOSPROCESS     /* Process and thread values */
#include <os2.h>
#include <stdio.h>

ULONG   Scope;         /* Indicate scope of change */
ULONG   PriorityClass; /* Priority class to set */
LONG    PriorityDelta; /* Priority delta to apply */
ULONG   ID;            /* Process or thread ID */
APIRET  rc;            /* Return code */

   Scope = 0;          /* Indicate that the intent is to change */
                       /*   the priority of the process that is */
                       /*   identified by the PID contained     */
                       /*   within variable ID                  */

   PriorityClass = 0;  /* Do not change the priority class of */
                       /*   the targeted process              */

   PriorityDelta = 5;  /* Give the targeted process a priority */
                       /*   delta of +5                        */

   rc = DosSetPriority(Scope, PriorityClass, PriorityDelta, ID);

   if (rc != 0)
     {
       printf("DosSetPriority error: return code = %ld", rc);
       return;
     }
```

DosSetProcessCp — Allow a Process to Set Its Code Page

```
#define INCL_DOSNLS

APIRET DosSetProcessCp  (ULONG  ulCodePage)
```

DosSetProcessCp allows a process to set its code page.

Parameters

ulCodePage (ULONG) — input

A code page identifier that has one of the following values:

Value	Description
437	United States
850	Multilingual
852	Latin 2 (Czechoslovakia, Hungary, Poland, Yugoslavia)
857	Turkish
860	Portuguese
861	Iceland
863	Canadian French
865	Nordic
932	Japan
934	Korea
936	People's Republic of China
938	Taiwan
942	Japan SAA
944	Korea SAA
946	People's Republic of China SAA
948	Taiwan SAA

Note: Code pages 932, 934, 936, 938, 942, 944, 946, and 948 are supported only with the Asian version of the operating system on Asian hardware.

Returns

Return Code.

DosSetProcessCp returns the following values:

0	NO_ERROR
472	ERROR_INVALID_CODE_PAGE

Remarks

DosSetProcessCp sets the process code page of the calling process. The code page of a process is used in the following ways:

First, the printer code page is set to the process code page through the file system and printer spooler (the system spooler must be installed) when the process makes a request to open the printer. Calling DosSetProcessCp does not affect the code page of a printer opened prior to the call, and does not affect the code page of a printer opened by another process.

DosSetProcessCp –
Allow a Process to Set Its Code Page

Second, country-dependent information, by default, is retrieved encoded in the code page of the calling process.

Third, a newly-created process inherits its process code page from its parent process.

DosSetProcessCp does not affect the display or keyboard code page.

Related Functions

- DosMapCase
- DosQueryCollate
- DosQueryCp
- DosQueryCtryInfo
- DosQueryDBCSEnv

Example Code

This example shows how a process can set its code page.

```
#define INCL_DOSNLS   /* National Language Support values */
#include <os2.h>
#include <stdio.h>

ULONG   CodePage;   /* Code page identifier */
APIRET  rc;         /* Return code */

   CodePage = 850;  /* Choose the Multilingual code page */

   rc = DosSetProcessCp(CodePage);

   if (rc != 0)
     {
       printf("DosSetProcessCp error: return code = %ld", rc);
       return;
     }
```

DosSetRelMaxFH – Adjust the Maximum Number of File Handles

```
#define INCL_DOSFILEMGR

APIRET DosSetRelMaxFH (PLONG ppReqCount, PULONG pCurMaxFH)
```

DosSetRelMaxFH adjusts the maximum number of file handles for the calling process.

Parameters

ppReqCount (PLONG) – input

Address of the number to be added to the maximum number of file handles for the calling process. If *ReqCount* is positive, the maximum number of file handles is increased. If *ReqCount* is negative, the maximum number of file handles is decreased.

The system treats a decrease in the maximum number of file handles as an advisory request that may or may not be granted; the system may track and defer such a request.

pCurMaxFH (PULONG) – output

Address of the variable to receive the new total number of allocated file handles.

Returns

Return Code.

DosSetRelMaxFH returns the following values:

0 NO_ERROR

Remarks

All file handles that are currently open are preserved. The system may defer or disregard a request to decrease the maximum number of file handles for the current process. The return code is set to NO_ERROR even if the system defers or disregards a request for a decrease.

You should examine the value of *CurMaxFH* to determine the result of DosSetRelMaxFH.

Related Functions

- DosDupHandle
- DosOpen
- DosSetMaxFH

DosSetRelMaxFH –
Adjust the Maximum Number of File Handles

Example Code

This example increases the maximum number of file handles for the calling process.

```
#define INCL_DOSFILEMGR   /* File Manager values */
#include <os2.h>
#include <stdio.h>

LONG    ReqCount;   /* Number to add to maximum
                       handle count */
ULONG   CurMaxFH;   /* New count of handles */
APIRET  rc;         /* Return code */

   ReqCount = 2;    /* Increase the maximum number by 2 */

   rc = DosSetRelMaxFH(&ReqCount, &CurMaxFH);
                    /* On successful return, the CurMaxFH */
                    /*   variable will contain the total  */
                    /*   number of allocated file handles */
                    /*   for this process                 */

   if (rc != 0)
     {
       printf("DosSetRelMaxFH error: return code = %ld", rc);
       return;
     }
```

DosSetSession – Set Session Status

#define INCL_DOSSESMGR

APIRET DosSetSession (ULONG ulSessID, PSTATUSDATA ppStatusData)

DosSetSession sets the status of a child session.

Parameters

ulSessID (ULONG) – input

The identifier of the target session. The value specified must have been returned on a previous call to DosStartSession.

ppStatusData (PSTATUSDATA) – input

Address of the status data structure.

StatusData is a structure that contains the session status data:

SIZE	DESCRIPTION
WORD	**Length**
WORD	**SelectInd**
WORD	**BondInd**

Length is the length of the data structure in bytes, including **Length** itself. **Length** is 6 bytes.

SelectInd specifies whether the target session should be flagged as selectable or non-selectable, as follows:

Value	Definition
0	(SET_SESSION_UNCHANGED) Leaves the current setting unchanged.
1	(SET_SESSION_SELECTABLE) Makes the target session selectable.
2	(SET_SESSION_NON_SELECTABLE) Makes the target session non-selectable. A non-selectable session is not selectable from the Shell switch list, nor can the user jump to it via the system hot key. The operator may continue to select a non-selectable windowed session by pressing a mouse button within a visible part of the window.

BondInd specifies which session to bring to the foreground the next time the parent session is selected, as follows:

Value	Definition
0	(SET_SESSION_UNCHANGED) Leaves the current setting unchanged.
1	(SET_SESSION_BOND) Establishes a bond between the parent session and the child session. The child session is brought to the foreground the next time the parent session is selected. If the child session is selected, the child session is brought to the foreground.
2	(SET_SESSION_NO_BOND) Specifies bringing the parent session to the foreground the next time the parent session is selected, and bringing the child session to the foreground if the child is selected. Any bond previously established with the child session specified is broken.

DosSetSession –
Set Session Status

Returns

Return code

DosSetSession returns the following values:

0	NO_ERROR
369	ERROR_SMG_INVALID_SESSION_ID
418	ERROR_SMG_INVALID_CALL
455	ERROR_SMG_INVALID_BOND_OPTION
456	ERROR_SMG_INVALID_SELECT_OPT
460	ERROR_SMG_PROCESS_NOT_PARENT
461	ERROR_SMG_INVALID_DATA_LENGTH
463	ERROR_SMG_RETRY_SUB_ALLOC

Remarks

DosSetSession sets or resets one or both of the following parameters related to a child session:

1. Selectable/non-selectable. This parameter allows a parent session to set one of its child sessions as selectable or non-selectable from the Shell switch list.
2. Bond/no bond. This parameter allows a parent session to bond one of its child sessions to itself. This means that if the operator subsequently selects the parent session from the Shell menu (or double clutches to the parent session), then the child session will be brought to the foreground.

The parameters only affect user selections from the Shell switch list or Shell selections during system hot key processing. They do not affect selections made by the parent session. Thus, when a parent session selects its own session, its own session is brought to the foreground, even if a bond is in effect. When a parent session selects a child session, the child session is brought to the foreground, even if the parent has previously set the child non-selectable.

The above parameters may be set individually. Either can be changed without affecting the current setting of the other.

DosSetSession may only be issued by a parent session for a child session. Neither the parent session itself nor any grandchild, nor any other descendant session beyond a child session, may be the target of this function. DosSetSession may only be issued by the process that originally started the specified session (*SessID*) through DosStartSession.

DosSetSession may only be used to change the status of child sessions that were originally started by the caller with DosStartSession specifying a value of 1 for **Related**. That is, DosSetSession may not be used to change the status of sessions started as independent sessions.

A bond established between a parent session and a child session can be broken by reissuing DosSetSession and specifying either:

- **BondInd** = 2 to break the bond, or
- **BondInd** = 1 to establish a bond with a different child session. In this case, the bond with the previous child is broken.

If a bond is established between session A and its immediate child session B, and if another bond is established between session B and its immediate child session C, then if the operator selects session A, session C is brought to the foreground. However, if session A selects itself, session A is brought to the foreground. If session A selects session B, session C is brought to the foreground. In the latter case, the bond between B and C is honored.

Assume that a bond is established between session A and its immediate child session B, and assume that session B is non-selectable. The operator will not be able to select session B directly. However, if the operator selects session A, session B will be brought to the foreground.

DosSetSession – Set Session Status

A parent session may be running in either the foreground or the background when DosSetSession is issued.

Related Functions

- DosSelectSession
- DosStartSession
- DosStopSession

Example Code

This example shows a how a parent session sets the status of one of its child sessions to be non-selectable from the Shell switch list. Assume that the session ID of the desired child session has been placed into *SessID* already.

```
#define INCL_DOSSESMGR      /* Session Manager values */
#include <os2.h>
#include <stdio.h>

ULONG         SessID;        /* Session identifier  */
STATUSDATA    StatusData;    /* Session status data */
APIRET        rc;            /* Return code */

   StatusData.Length = 6;    /* Length of the StatusData */
                             /*   structure              */

   StatusData.SelectInd = 2; /* Make the specified child      */
                             /*   session non-selectable from */
                             /*   the Shell switch list       */

   StatusData.BondInd = 0;   /* Do not change the "bring to */
                             /*   foreground" attribute     */

   rc = DosSetSession(SessID, &StatusData);

   if (rc != 0)
     {
       printf("DosSetSession error: return code = %ld", rc);
       return;
     }
```

DosSetSignalExceptionFocus – Set Signal Exception Focus

```
#define INCL_DOSEXCEPTIONS
```

APIRET DosSetSignalExceptionFocus (BOOL32 f32Flag, PULONG ppulTimes)

DosSetSignalExceptionFocus causes the current process to become the focus for the Ctrl + C and Ctrl + Break signals.

Parameters

f32Flag (BOOL32) – input

This parameter may have either of two values:

Value	Definition
0	(SIG_UNSETFOCUS) Stop receiving signals.
1	(SIG_SETFOCUS) Start receiving signals.

ppulTimes (PULONG) – output

The number of times DosSetSignalExceptionFocus has been called by the current process with *Flag* set to 1, minus the number of times it has been called with *Flag* set to 0.

Returns

Return Code.

DosSetSignalExceptionFocus returns the following values:

0	NO_ERROR
1	ERROR_INVALID_FUNCTION
300	ERROR_ALREADY_RESET
303	ERROR_INVALID_PROCID
650	ERROR_NESTING_TOO_DEEP

Remarks

DosSetSignalExceptionFocus causes the calling process to become the signal focus for its screen group for the XCPT_SIGNAL_BREAK (Ctrl+Break) and XCPT_SIGNAL_INTR (Ctrl+C) signal exceptions.

You cannot issue DosSetSignalExceptionFocus from a Presentation Manager (PM) application. If you do, you get the return code ERROR_INVALID_PROCID. You can issue this function from a full-screen or windowed application.

For a detailed list of the system exceptions, see Appendix C, "System Exceptions" on page C-1.

Related Functions

- DosAcknowledgeSignalException
- DosEnterMustComplete
- DosExitMustComplete
- DosRaiseException
- DosSendSignalException
- DosSetExceptionHandler
- DosUnsetExceptionHandler
- DosUnwindException

Example Code

This example causes the current process to try to become the focus for Ctrl + C and Ctrl + Break signals. Once a process holds the focus, it can issue further DosSetSignalExceptionFocus functions that request the focus again. The process must eventually issue an equivalent number of functions that relinquish the focus. Each DosSetSignalExceptionFocus returns the net number of focus request calls in effect, so the calling process can eventually tell if more relinquish calls are required.

```
#define INCL_DOSEXCEPTIONS   /* Exception values */
#include <os2.h>
#include <stdio.h>

BOOL32  flag;     /* Set flag to start or stop receiving signals */
ULONG   ulTimes;  /* Number of times flag has been set to 1, minus
                     number of times set to 0 */
APIRET  rc;       /* Return code */

  flag = SIG_SETFOCUS;
                /* Indicate that the process should start */
                /*   receiving signal focus               */

  rc = DosSetSignalExceptionFocus(flag, &ulTimes);
                /* On successful return, the ulTimes       */
                /*   variable contains the net number of   */
                /*   times DosSetSignalExceptionFocus has  */
                /*   been called by this process to        */
                /*   request the focus (SIG_SETFOCUS)      */
                /*   minus the number of times it has been */
                /*   called to relinquish the focus        */
                /*   (SIG_UNSETFOCUS)                      */

  if (rc != 0)
    {
      printf("DosSetSignalExceptionFocus error: return code = %ld",
                  rc);
      return;
    }
```

DosSetVerify – Set Write Verification

#define INCL_DOSFILEMGR

APIRET DosSetVerify (BOOL32 f32VerifySetting)

DosSetVerify sets write verification.

Parameters

f32VerifySetting (BOOL32) – input

The state of verify mode, as follows:

Value	Definition
0	Verify mode is not active.
1	Verify mode is active.

Returns

Return Code.

DosSetVerify returns the following values:

0	NO_ERROR
118	ERROR_INVALID_VERIFY_SWITCH

Remarks

When verify mode is active, the operating system verifies that data written to the disk is recorded correctly, even though disk recording errors are rare.

Related Functions

- DosQueryVerify

Example Code

This example enables write verification for the system.

```
#define INCL_DOSFILEMGR   /* File Manager values */
#include <os2.h>
#include <stdio.h>

BOOL32   VerifySetting; /* New value of verify switch */
APIRET   rc;            /* Return code */

   VerifySetting = 1;   /* Indicate that verify mode is to be */
                        /*   activiated                       */

   rc = DosSetVerify(VerifySetting);

   if (rc != 0)
     {
       printf("DosSetVerify error: return code = %ld", rc);
       return;
     }
```

DosShutdown – Shut Down the System

#define INCL_DOSFILEMGR

APIRET DosShutdown (ULONG ulReserved)

DosShutdown locks out changes to all file systems, and writes system buffers to the disk in preparation for turning off power to the system.

Parameters

ulReserved (ULONG) – input

Doubleword, value must be zero.

Returns

Return Code.

DosShutdown returns the following values:

0	NO_ERROR
87	ERROR_INVALID_PARAMETER
274	ERROR_ALREADY_SHUTDOWN

Remarks

DosShutdown can take several minutes to complete its operation; the time depends on the amount of data in the buffers.

If other functions that change file-system data are issued while the system is shut down, either the return code ERROR_ALREADY_SHUTDOWN is set, or the other function calls are blocked permanently.

Allocated memory cannot be increased once DosShutdown has been issued. This means that in low-memory situations, some functions may fail because of a lack of memory. This is of particular importance to the process issuing DosShutdown. All memory that the calling process will ever need should be allocated before DosShutdown is issued. This includes implicit memory allocations that system functions make on behalf of DosShutdown.

When DosShutdown has completed successfully, the system can be powered-off or restarted.

Related Functions

- There are none.

DosShutdown — Shut Down the System

Example Code

This example locks out changes to all file systems, and writes system buffers to the disk in preparation for turning off power to the system.

```
#define INCL_DOSFILEMGR   /* File Manager values */
#include <os2.h>
#include <stdio.h>

ULONG   Reserved;   /* Reserved, must be zero */
APIRET  rc;         /* Return code */

   Reserved = 0;    /* Reserved, must be set to zero */

   rc = DosShutdown(Reserved);

   if (rc != 0)
     {
       printf("DosShutdown error: return code = %ld", rc);
       return;
     }
```

DosSleep – Delay Process Execution

#define INCL_DOSDATETIME

APIRET DosSleep (ULONG ulTimeInterval)

DosSleep suspends the current thread for a specified time interval.

Parameters

ulTimeInterval (ULONG) – input

The time, in milliseconds, for which the calling thread's execution will be suspended. The system rounds this value up to the next clock tick.

Returns

Return Code.

DosSleep returns the following values:

0	NO_ERROR
322	ERROR_TS_WAKEUP

Remarks

DosSleep suspends the current thread for a specified time interval. If a time interval of 0 is specified, the thread gives up the remainder of the current time slice, allowing any other ready threads of equal or higher priority to execute; the calling thread will execute again during its next scheduled time slice. If there is no other ready thread of equal or higher priority, DosSleep returns immediately; it does not give control to a thread of lower priority.

Time intervals for DosSleep, DosAsyncTimer, and DosStartTimer are specified in milliseconds; however, it is important to recognize that the actual duration of the specified time interval will be affected by two factors:

- First, the system clock keeps track of time in less precise units known as *clock ticks*. The duration of a clock tick depends on the frequency of the system-clock interrupt that is used by your computer. (To determine the duration of the clock tick on your computer, issue DosQuerySysInfo and examine the timer-interval field.)

  Because clock ticks are less precise than millisecond values, any time interval that is specified in milliseconds will be rounded up to the next clock tick.

- Second, because the system is a priority-based, multitasking operating system, there is no guarantee that a thread will resume immediately after the timer interval expires. If a higher-priority process or thread is executing, the timed thread blocks. (To minimize the inaccuracy caused by preemptive scheduling, an application can dedicate a thread to managing time-critical tasks and then raise that thread to a higher priority.)

In addition, the time interval for DosSleep refers to execution time (accumulated scheduled time slices), not to elapsed real time. Elapsed real time will be longer and will vary, depending on the hardware and on the number and priorities of other threads executing in the system. (Elapsed real time for the asynchronous timers, started by DosAsyncTimer and DosStartTimer, will be much closer to their specified time intervals because these timers run independently of the calling thread's execution.)

Because the above factors usually cause the sleep interval to be longer than requested (though generally within a few clock ticks), DosSleep should not be used as a substitute for a real-time clock.

To ensure optimal performance, do not use DosSleep in a single-thread Presentation Manager application. (See WinStartTimer.)

DosSleep –
Delay Process Execution

If the calling thread is awakened before the time interval expires (by a system exception, for example), ERROR_TS_WAKEUP is returned.

Related Functions

- DosAsyncTimer
- DosGetDateTime
- DosSetDateTime
- DosStartTimer
- DosStopTimer

Example Code

This example suspends the calling thread for one minute.

```
#define INCL_DOSPROCESS   /* Process and thread values */
#include <os2.h>
#include <stdio.h>

ULONG   TimeInterval;  /* Interval in milliseconds */
APIRET  rc;            /* Return code */

   TimeInterval = 60000;

   rc = DosSleep(TimeInterval);

   if (rc != 0)
     {
       printf("DosSleep error: return code = %ld", rc);
       return;
     }
```

#define INCL_DOSSESMGR

APIRET DosStartSession (PSTARTDATA ppStartData, PULONG pSessID, PPID ppPID)

DosStartSession allows an application to start another session, and to specify the name of the program to be started in that session.

Parameters

ppStartData (PSTARTDATA) – input

Address of the start session structure:

Size	Description
WORD	**Length**
WORD	**Related**
WORD	**FgBg**
WORD	**TraceOpt**
DWORD	**PgmTitle**
DWORD	**PgmName**
DWORD	**PgmInputs**
DWORD	**TermQ**
DWORD	**Environment**
WORD	**InheritOpt**
WORD	**SessionType**
DWORD	**IconFile**
DWORD	**PgmHandle**
WORD	**PgmControl**
WORD	**InitXPos**
WORD	**InitYPos**
WORD	**InitXSize**
WORD	**InitYSize**
WORD	**Reserved**
DWORD	**ObjectBuffer**
DWORD	**ObjectBuffLen**

Length is the length of the data structure in bytes, including **Length** itself. This value can be 24, 30, 32, 50, or 60 bytes.

A length of at least 32 bytes must be used to start a DOS session with the session type specified. A length greater than 32 is not allowed if the Session Manager detects that the Presentation Manager is not present.

When a **Length** of 24 or 30 bytes is specified, DosStartSession initializes the missing parameters to zero. This allows the Shell to provide values for the missing information, based on the installation file entry for the program being started.

Specify a **Length** of 30 bytes to use the environment and inheritance features of the system. Specify a **Length** of 50 bytes to specify the type of session to start, and to define data for windows.

A **Length** of 60 bytes allows you to use all of the functions provided by DosStartSession.

Related specifies whether the session created is related to the calling session. The values of this field are as follows:

Value	Definition
0	(SSF_RELATED_INDEPENDENT): New session is an independent session (not related)
1	(SSF_RELATED_CHILD): New session is a child session (related)

DosStartSession – Start Session

An independent session cannot be controlled by the calling program. It may not be specified as the target of DosSelectSession, DosSetSession, or DosStopSession. The **TermQ** field is ignored for independent sessions, and *SessID* and *PID* are not returned. Refer to **"Parent/Child Relationship"** in the **Remarks** section for additional information about related sessions.

FgBg specifies whether the new session should be started in the foreground or background. If a windowed session is started in the foreground, the new session will be given the window focus. The values of this field are as follows:

Value	Definition
0	(SSF_FGBG_FORE): Start session in foreground
1	(SSF_FGBG_BACK): Start session in background

TraceOpt specifies whether the program started in the new session should be executed under conditions for tracing. The values of this field are as follows:

Value	Definition
0	(SSF_TRACEOPT_NONE): No trace
1	(SSF_TRACEOPT_TRACE): Trace with no notification of descendants
2	(SSF_TRACEOPT_TRACEALL): Trace all descendant sessions

Related equals 1 and a termination queue must be supplied when a **TraceOpt** of 2 is specified. Refer to **"Debugger Considerations"** in the **Remarks** section for additional information about a **TraceOpt** of 2.

PgmTitle is the address of an ASCIIZ string that contains the program title. The string can be up to 61 bytes long, including the terminating byte of zero. If the address specified is zero, or if the ASCIIZ string is null, then the initial title is **PgmName** minus any leading drive and path information.

PgmName is either zero or the address of an ASCIIZ string that contains the fully-qualified drive, path, and file name of the program to be loaded. Refer to **"PgmName and PgmInputs Considerations"** in the **Remarks** section for additional information about a zero **PgmName** address.

PgmInputs is either zero or the address of an ASCIIZ string that contains the input arguments to be passed to the program.

TermQ is either zero or the address of an ASCIIZ string that contains the fully-qualified path and file name of a system queue (refer to DosCreateQueue). Refer to **"Parent/Child Termination Considerations"** in the **Remarks** section for additional information about the **TermQ** field.

Environment is either zero or the address of an environment string (refer to the **Parameters** for DosExecPgm) to be passed to the program started in the new session. The **Environment** may be used for independent or related DosStartSession functions. When the **Environment** field is zero, the program in the new session inherits the environment of the Shell if the **InheritOpt** field is equal to zero, or the environment of the program issuing DosStartSession if the **InheritOpt** field is equal to one.

The **Environment** field for a DOS session is reserved, and must be zero. The DOS session always processes the AUTOEXEC.BAT file on the DOS startup drive. You can define DOS environment variables in the AUTOEXEC.BAT file.

InheritOpt specifies whether the program started in the new session should inherit the calling program's environment and open file handles. The values of this field are as follows:

Value	Definition
0	(SSF_INHERTOPT_SHELL): Inherit the Shell's environment.
1	(SSF_INHERTOPT_PARENT): Inherit the environment of the program issuing the DosStartSession call.

The **InheritOpt** field may be used for independent or related DosStartSession functions. Therefore, a DosStartSession function with the **InheritOpt** field equal to one is equivalent to

DosExecPgm, except that the new program does not inherit the priority of the parent process, or the keyboard and video characteristics associated with the parent session. Also, a parent process/child process relationship is not established.

Refer to **"Parent/Child Relationship"** in the **Remarks** section for additional information about related sessions.

The **InheritOpt** field for a DOS session is different than the **InheritOpt** field for a non-DOS session. An **InheritOpt** value of 1 for a DOS session only inherits the parent's current drive and path. It does not inherit the parent's environment.

SessionType defines the type of session that should be created for this program. The values of this field are as follows:

Value	Definition
0	(SSF_TYPE_DEFAULT): Use the PgmHandle data, or allow the Shell to establish the session type.
1	(SSF_TYPE_FULLSCREEN): Start the program in a full-screen session.
2	(SSF_TYPE_WINDOWABLEVIO): Start the program in a windowed session for programs using the Base Video Subsystem.
3	(SSF_TYPE_PM): Start the program in a windowed session for programs using the Presentation Manager services (including AVIO calls).
4	(SSF_TYPE_VDM): Start the program in a full-screen DOS session.
7	(SSF_TYPE_WINDOWEDVDM): Start the program in a windowed DOS session.

IconFile is either zero or the address of an ASCIIZ string that contains the fully-qualified drive, path, and file name of an icon definition. If you do not provide an icon file name with DosStartSession, the system looks for an associated icon file (with a file extension of .ICO, or an extended attribute of .ICON). The system provides a default icon if an icon file name is not provided with DosStartSession.

PgmHandle is either zero or the program handle returned by WinAddProgram or WinQueryProgramHandle. The program handle identifies the program in the installation file to be started, the program title, the session type, and the initial window size and position. However, information may be specified with DosStartSession to override the information in the installation file for this invocation of the program.

DosStartSession does not support program groups.

PgmControl may be used to specify the initial state for a windowed application. This field is ignored for full-screen sessions.

The bits in this field have the following values:

Bit	Value	Equated Name	Initial Window State
0	0x0000	SSF_CONTROL_VISIBLE	Visible
0	0x0001	SSF_CONTROL_INVISIBLE	Invisible
1	0x0002	SSF_CONTROL_MAXIMIZE	Maximize
2	0x0004	SSF_CONTROL_MINIMIZE	Minimize
3	0x0008	SSF_CONTROL_NOAUTOCLOSE	No Auto Close
4 - 14			Reserved
15	0x8000	SSF_CONTROL_SETPOS	Use specified size and position

Note: The "No Auto Close" bit is used only for VIO Windowable applications, and is ignored for all other types of applications.

InitXPos and **InitYPos** contain the initial x and y coordinates, in pels, for the initial session window. Coordinates (0,0) indicate the bottom left corner of the display. This field is ignored for full-screen sessions.

InitXSize and **InitYSize** contain the initial x and y extent, in pels, for the initial session window. This field is ignored for full-screen sessions.

Reserved is a word of zeros, reserved for future use.

ObjectBuffer is the address of a buffer where the name of the object that contributed to the failure of DosExecPgm is returned. DosStartSession calls DosExecPgm to start all full-screen, VIO windowed, and Presentation Manager programs.

ObjectBuffLen is the length, in bytes, of the buffer pointed to by **ObjectBuffer**.

pSessID (PULONG) – output

The address of a doubleword that receives the session identifier associated with the child session created. *SessID* is returned only when the value specified for **Related** is 1. The *SessID* returned can be specified on subsequent calls to DosSelectSession, DosSetSession, and DosStopSession.

ppPID (PPID) – output

The address of a doubleword that receives the process identifier associated with the child process created. *PID* is returned only when the value specified for **Related** is 1. The *PID* returned may *not* be used on any system functions (for example, DosSetPriority) that require a parent process/child process relationship. Read **"Parent/Child Relationship"** in the **Remarks** section for more information.

Returns

Return code.

DosStartSession returns the following values:

0	NO_ERROR
369	ERROR_SMG_INVALID_SESSION_ID
418	ERROR_SMG_INVALID_CALL
460	ERROR_SMG_PROCESS_NOT_PARENT
463	ERROR_SMG_RETRY_SUB_ALLOC

Remarks

DosStartSession allows an application to start another session, and to specify the name of the program to be started in that session.

A session can be thought of as a logical console, consisting of buffers for the screen, keyboard, and mouse.

New sessions may only be started in the foreground when the caller's session (or one of the caller's descendant sessions) is currently executing in the foreground. The foreground session for windowed applications is the session of the application that owns the window focus. The new session appears in the Shell switch list.

Any protect-mode application may start any other protect-mode application in a new session, regardless of the issuing program's session type.

You may use DosExecPgm to start a process that is of the same type as the starting process. Process types include Presentation Manager, text-windowed, and full-screen. You may not use DosExecPgm to start a process that is of a different type than the starting process.

You must use DosStartSession to start a new process from a process that is of a different type. For example, use DosStartSession to start a Presentation Manager process from a non-Presentation Manager process.

Foreground/Background Considerations:

DosStartSession will only start a new session in the foreground if the program issuing DosStartSession or a descendent session is executing in the foreground session. Otherwise,

DosStartSession will override the foreground request and start the new session in the background. A unique error is returned, indicating that the new session was started in the background. The foreground session for windowed applications is the session of the application that owns the window focus. Therefore, when a windowed session is started in the foreground, the new session will be given the window focus.

Parent/Child Relationship:

When you specify a value of 1 for Related, DosStartSession establishes a parent session/child session relationship. A parent process/child process relationship is not established. The parent process is the shell process just as if the operator had started the program from the shell menu. Therefore, the *PID* returned by DosStartSession may not be used with any system functions (for example, DosSetPriority) that require a parent process/child process relationship.

Once a process has issued DosStartSession specifying a value of 1 for **Related**, no other process within that session can issue related DosStartSession functions until all the dependent sessions have ended.

Debugger Considerations:

Debuggers may want to debug all processes associated with an application, no matter how the process was started (by DosExecPgm or DosStartSession). A special trace option, **TraceOpt** value 2, has been provided for this purpose. When a value of 2 is specified for **TraceOpt**, the debugger must also supply the name of an existing queue, and a value of 1 for **Related**, on the DosStartSession function.

The Session Manager notifies the debugger whenever a new session is created through DosStartSession from the initial session started with a value of 2 for **TraceOpt**, or from any descendant session. The queue is posted regardless of how the new session is started (related, independent, with or without inheritance). Sessions started without inheritance are executed for tracing. It is the responsibility of the debugger to resume execution of the new process.

The debugger must issue DosReadQueue to receive notification when a child session is created. The word containing the request parameter, returned by DosReadQueue, will have a value of 1. The data element structure has the following format:

Size	Description
WORD	Session ID
WORD	Process ID

The debugger should issue DosReadQueue with the *NoWait* parameter set to zero. This is the only process that has addressability to the notification data element. After reading and processing the data element, the debugger must free the segment that contains the data element by issuing DosFreeMem.

The debugger may use DosSelectSession to switch itself or any descendant session into the foreground whenever the current foreground session is a descendant of the debugger.

PgmName and PgmInputs Considerations:

The program identified by **PgmName** is executed directly, with no intermediate secondary command (CMD.EXE) process. Alternatively, the program can be executed indirectly through a secondary command (CMD.EXE) process by specifying CMD.EXE for **PgmName**, and by specifying either /C or /K followed by the drive, path, and file name of the application to be loaded for **PgmInputs**. If the /C parameter is inserted at the beginning of the **PgmInputs** string, then when the application program ends, the session ends. If the /K parameter is inserted at the beginning of the **PgmInputs** string, then when the application ends, the operator sees the system command line prompt displayed. The operator can then either enter the name of another program or command to execute, or enter the EXIT command to end the session.

When the **PgmName** address is zero, or the ASCIIZ string is null, the program identified by the **PgmHandle** is started in the new session. If the **PgmHandle** is not specified, then the program specified as a parameter to the protect mode shell on the OS2_SHELL statement, or on the SHELL statement for a DOS session, in the configuration file (CONFIG.SYS) is executed and passed the specified **PgmInputs**. The default is the program name for the command processor (CMD.EXE for a non-DOS session, or COMMAND.COM for a DOS session).

DosStartSession – Start Session

The **PgmName** and **PgmInputs** strings' combined length may not exceed 1024 characters.

Program Handle Considerations:

If a process issues DosStartSession specifying only the program handle, then it must change to the working directory *before* issuing DosStartSession, and start the new process as inherited. If a process is started as non-inherited, it is up to that process to change to the correct directory.

Parent/Child Termination Considerations:

The parent must create the termination queue prior to specifying the queue name on DosStartSession. The Session Manager will continue to notify the parent session through the specified queue as long as the process issuing DosStartSession remains a parent session. When all the child sessions for a particular parent session end, the termination queue is closed by the Session Manager. An existing queue name must be specified on the next DosStartSession function if the caller wants to continue receiving termination notification messages.

The Session Manager writes a data element into the specified queue when any child session ends. The queue is posted regardless of who terminates the child session (for example, child, parent, or operator) and whether the termination is normal or abnormal.

A parent session issues DosReadQueue to receive notification when a child session has ended. The word that contains the request parameter, returned by DosReadQueue, will be zero. The data element structure has the following format:

Size	Description
WORD	Session ID of child
WORD	Result code

The process that originally issued DosStartSession should issue DosReadQueue, with the *NoWait* parameter set to zero. This is the only process that has addressability to the notification data element. After reading and processing the data element, the caller must free the segment containing the data element by issuing DosFreeMem.

An application may use the termination queue for additional interprocess communication, provided that a unique request identifier is passed via DosWriteQueue. Request identifier values 0 through 99 are reserved for the operating system. Request identifier values equal to or greater than 100 are available for application use.

When a child session ends, the result code returned in the **TermQ** data element is the result code of the program specified by **PgmName**, assuming either:

- the program is executed directly, with no intermediate secondary command (CMD.EXE) process, or
- the program is executed indirectly through a secondary command (CMD.EXE) process, and the /C parameter is specified.

Otherwise, the result code of CMD.EXE is returned.

When a child session is executing in the foreground at the time it ends, the parent session becomes the foreground session. When a parent session ends, all child sessions that it created with DosStartSession, specifying a value of 1 for **Related**, are ended. When an independent session, created specifying a value of 0 for **Related**, ends in the foreground, the Shell selects the next foreground session.

Grandchildren Considerations:

A session started through DosStartSession may issue DosStartSession. The following rules apply:

- The *SessID* specified on DosSelectSession, DosSetSession, and DosStopSession may only be the session identifier (*SessID*) of an immediate child session, not a grandchild session or any descendant other than an immediate child session.
- If a bond is established between session A and its immediate child session B, and if another bond is established between session B and its immediate child session C, then if session A

is selected, session C is brought to the foreground. Refer to DosSetSession for a description of what establishing a bond means.

- When a session ends, all of its descendants (child sessions, grandchild sessions, and so on) are ended.

Related Functions

- DosSelectSession
- DosSetSession
- DosStopSession

Example Code

This example shows how an application starts a program in another session.

```
#define INCL_DOSSESMGR      /* Session Manager values */
#include <os2.h>
#include <stdio.h>

STARTDATA   StartData;    /* Start session data structure */
ULONG       SessID;       /* Session ID (returned) */
PID         PID;          /* Process ID (returned) */
UCHAR       PgmTitle[40]; /* Program title string */
UCHAR       PgmName[80];  /* Program pathname string */
UCHAR       ObjBuf[100];  /* Object buffer */
APIRET      rc;           /* Return code */

   /*  Specify the various session start parameters  */

   StartData.Length = sizeof(STARTDATA);
                         /* Length of STARTDATA structure */

   StartData.Related = SSF_RELATED_CHILD;
                         /* Child session */

   StartData.FgBg = SSF_FGBG_BACK;
                         /* Start child session in background */

   StartData.TraceOpt = SSF_TRACEOPT_NONE;
                         /* Don't trace session */

   strcpy(PgmTitle,"Sample Program");
   StartData.PgmTitle = PgmTitle;
                         /* Session Title string */

   strcpy(PgmName,"D:\\PROG\\UTIL\\SAMPLE1.EXE");
   StartData.PgmName = PgmName;
                         /* Program path-name string */

   StartData.PgmInputs = 0;
                         /* Assume no input arguments need */
                         /*   be passed to the program     */

   StartData.TermQ = 0;   /* Assume no termination queue */

   StartData.Environment = 0;
                         /* Assume no environment string */

   StartData.InheritOpt = SSF_INHERTOPT_PARENT;
                         /* Inherit environment and open */
                         /*   file handles from parent   */
```

DosStartSession – Start Session

```
StartData.SessionType = SSF_TYPE_DEFAULT;
                        /* Allow the Shell to establish */
                        /*   the session type           */

StartData.IconFile = 0;
                        /* Assume no specific icon file */
                        /*   is provided                */

StartData.PgmHandle = 0;
                        /* Do not use the installation file */

StartData.PgmControl = SSF_CONTROL_VISIBLE
                         | SSF_CONTROL_MAXIMIZE;
                        /* Start the program as visible */
                        /*   and maximized              */

StartData.InitXPos = 30;
StartData.InitYPos = 40;
StartData.InitXSize = 200;    /* Initial window coordinates */
StartData.InitYSize = 140;    /*   and size                 */

StartData.Reserved = 0;
                        /* Reserved, must be zero */

StartData.ObjectBuffer = ObjBuf;
                        /* Object buffer to hold DosExecPgm */
                        /*   failure causes                 */

StartData.ObjectBuffLen = 100;
                        /* Size of object buffer */

rc = DosStartSession(&StartData, &SessID, &PID);
                       /* On successful return, the variable  */
                       /*   SessID contains the session ID    */
                       /*   of the new session, and the       */
                       /*   variable PID contains the process */
                       /*   ID of the new process             */

if (rc != 0)
  {
    printf("DosStartSession error: return code = %ld", rc);
    return;
  }
```

DosStartTimer – Start an Asychronous Timer

#define INCL_DOSDATETIME

APIRET DosStartTimer (ULONG ulTimeInterval, HSEM hsemSemHandle, PHTIMER ppHandle)

DosStartTimer starts an asynchronous, repeated-interval timer.

Parameters

ulTimeInterval (ULONG) – input

The time, in milliseconds, that will elapse between postings of the event semaphore specified by *SemHandle*. (The system rounds this value up to the next clock tick.)

hsemSemHandle (HSEM) – input

The handle of the event semaphore that is posted each time *TimeInterval* elapses. This semaphore must be a shared event semaphore. It should be reset between postings by calling DosResetEventSem

ppHandle (PHTIMER) – output

A pointer to the timer handle. This handle can be passed to DosStopTimer to stop the repeated-interval timer.

Returns

Return Code.

DosStartTimer returns the following values:

0	NO_ERROR
323	ERROR_TS_SEMHANDLE
324	ERROR_TS_NOTIMER

Remarks

DosStartTimer starts an asynchronous, repeated-interval timer, and posts an event semaphore each time the specified time interval expires.

Time intervals for DosStartTimer, DosAsyncTimer, and DosSleep are specified in milliseconds; however, it is important to recognize that the actual duration of the specified time interval will be affected by two factors:

- First, the system clock keeps track of time in less precise units known as *clock ticks*. The duration of a clock tick depends on the frequency of the system-clock interrupt that is used by your computer. (To determine the duration of the clock tick on your computer, issue DosQuerySysInfo and examine the *timer-interval* field.)

  Because clock ticks are less precise than millisecond values, any time interval that is specified in milliseconds will be rounded up to the next clock tick.

- Second, because the system is a priority-based, multitasking operating system, there is no guarantee that a thread will resume immediately after the timer interval expires. If a higher-priority process or thread is running, or if a hardware interrupt occurs, the timed thread blocks. (To minimize the inaccuracy caused by preemptive scheduling, an application can dedicate a thread to managing time-critical tasks and then raise that thread to a higher priority.)

These factors usually cause the timer interval to be longer than requested; however, it will generally be within a few clock ticks.

DosStartTimer –
Start an Asychronous Timer

Related Functions

- DosAsyncTimer
- DosGetDateTime
- DosSetDateTime
- DosSleep
- DosStopTimer
- DosCreateEventSem
- DosOpenEventSem
- DosResetEventSem
- DosWaitEventSem

Example Code

This example starts a periodic interval timer that runs asychronously to the calling thread. Each time the interval timer counts down to zero, it posts the specified event semaphore and then begins counting down again from the initial time value. Assume that the handle of the targeted event semaphore has been placed into *SemHandle* already.

```
#define INCL_DOSDATETIME   /* Date and time values */
#include <os2.h>
#include <stdio.h>

ULONG    TimeInterval; /* Interval (milliseconds) */
HSEM     SemHandle;    /* Event-semaphore handle */
HTIMER   Handle;       /* Timer handle (returned) */
APIRET   rc;           /* Return code */

   TimeInterval = 30000;  /* Set the periodic time interval to */
                          /*   elapse every 30 seconds         */

   rc = DosStartTimer(TimeInterval, SemHandle, &Handle);
                      /* On successful return, the variable */
                      /*   Handle will contain the handle   */
                      /*   of this periodic timer.          */
                      /*   DosStopTimer can be used later   */
                      /*   to stop the periodic timer.      */

   if (rc != 0)
     {
       printf("DosStartTimer error: return code = %ld", rc);
       return;
     }
```

DosStopSession – Stop Session

#define INCL_DOSSESMGR

APIRET DosStopSession (ULONG ulTargetOption, ULONG ulSessID)

DosStopSession ends one or all child sessions.

Parameters

ulTargetOption (ULONG) – input

Specifies whether only the session specified by *SessID*, or all sessions, should be ended, as follows:

Value	Definition
0	(STOP_SESSION_SPECIFIED) Ends only the specified session.
1	(STOP_SESSION_ALL) Ends all sessions.

ulSessID (ULONG) – input

The identifier of the session to be ended. The value specified for *SessID* must have been returned on a previous call to DosStartSession. *SessID* is ignored if *TargetOption* is equal to 1.

Returns

Return code

DosStopSession returns the following values:

0	NO_ERROR
369	ERROR_SMG_INVALID_SESSION_ID
418	ERROR_SMG_INVALID_CALL
458	ERROR_SMG_INVALID_STOP_OPTION
459	ERROR_SMG_BAD_RESERVE
460	ERROR_SMG_PROCESS_NOT_PARENT
463	ERROR_SMG_RETRY_SUB_ALLOC

Remarks

DosStopSession ends one or all child sessions.

DosStopSession may only be issued by a parent session for a child session. Neither the parent session itself nor any grandchild, nor any other descendant session beyond a child session, may be the target of this function. DosStopSession may only be issued by the process that originally started the specified session (*SessID*) with DosStartSession.

DosStopSession may only be used to end child sessions that were originally started by the caller with DosStartSession specifying a value of 1 for **Related**. That is, sessions started as independent sessions may not be stopped.

If the child session specified with DosStopSession has related sessions, these sessions will also be ended.

If a child session is executing in the foreground at the time it is ended, the parent session becomes the foreground session. DosStopSession breaks any bond that existed between the parent session and the specified child session.

A parent session may be executing in either the foreground or background when DosStopSession is issued.

DosStopSession – Stop Session

Since any process executing in the specified session may refuse to end, the only way to guarantee that the target session has ended is to wait for notification through the termination queue specified with DosStartSession.

Related Functions

- DosSelectSession
- DosSetSession
- DosStartSession

Example Code

This example shows how a parent session ends all of its child sessions.

```
#define INCL_DOSSESMGR      /* Session Manager values */
#include <os2.h>
#include <stdio.h>

ULONG   TargetOption;  /* Target option */
ULONG   SessID;        /* Session identifier */
APIRET  rc;            /* Return code */

   TargetOption = 1;   /* Indicate that all child sessions */
                       /*   are to be ended                */

   SessID = 0;         /* Unused, because target option 1 */
                       /*   is chosen                     */

   rc = DosStopSession(TargetOption, SessID);

   if (rc != 0)
     {
       printf("DosStopSession error: return code = %ld", rc);
       return;
     }
```

DosStopTimer — Stop an Asynchronous Timer

#define INCL_DOSDATETIME

```
APIRET DosStopTimer (HTIMER htimerHandle)
```

DosStopTimer stops an asynchronous timer.

Parameters

htimerHandle (HTIMER) — input

The handle of the timer to stop.

Returns

Return Code.

DosStopTimer returns the following values:

0	NO_ERROR
326	ERROR_TS_HANDLE

Remarks

DosStopTimer stops either a repeated-interval timer (started by DosStartTimer), or a single-interval timer (started by DosAsyncTimer).

When DosStopTimer is called, no assumption can be made about the state of the event semaphore that is associated with the timer. If the application is going to reuse the semaphore in conjunction with another timer, it should issue DosResetEventSem to ensure that the semaphore is in the "reset" state before starting the timer.

Related Functions

- DosAsyncTimer
- DosGetDateTime
- DosSetDateTime
- DosSleep
- DosStartTimer

DosStopTimer –
Stop an Asynchronous Timer

Example Code

This example stops a periodic timer that had been started previously with DosStartTimer. Assume that the handle of the periodic timer has been placed into *Handle* already.

```
#define INCL_DOSDATETIME   /* Date and time values */
#include <os2.h>
#include <stdio.h>

HTIMER   Handle;    /* Handle of the timer */
APIRET   rc;        /* Return code */

   rc = DosStopTimer(Handle);

   if (rc != 0)
     {
       printf("DosStopTimer error: return code = %ld", rc);
       return;
     }
```

DosSubAllocMem – Allocate a Block of Memory from a Memory Pool

#define INCL_DOSMEMMGR

APIRET DosSubAllocMem (PVOID pOffset, PPVOID ppBlockOffset, ULONG ulSize)

DosSubAllocMem allocates a block of memory from a memory pool that was previously initialized by DosSubSetMem.

Parameters

pOffset (PVOID) – input

The offset to the memory pool from which the block should be allocated.

ppBlockOffset (PPVOID) – output

The address of a doubleword in which the offset of the allocated memory block is returned.

ulSize (ULONG) – input

The size, in bytes, of the memory block requested.

Returns

Return Code.

DosSubAllocMem returns the following values:

0	NO_ERROR
87	ERROR_INVALID_PARAMETER
311	ERROR_DOSSUB_NOMEM
532	ERROR_DOSSUB_CORRUPTED

Remarks

DosSubAllocMem allocates a block of memory from a memory pool previously initialized by DosSubSetMem.

Allocation size should be a multiple of 8 bytes, otherwise it will be rounded up. The maximum value for *Size* is the size of the memory pool initialized by DosSubSetMem minus 64 bytes.

Related Functions

- DosSubFreeMem
- DosSubSetMem
- DosSubUnsetMem

DosSubAllocMem –
Allocate a Block of Memory from a Memory Pool

Example Code

This example allocates a block of memory from a memory pool that was previously initialized by DosSubSetMem. Assume that the *Offset* variable has been set to the address of the initialized memory pool already.

```
#define  INCL_DOSMEMMGR   /* Memory Manager values */
#include <os2.h>
#include <stdio.h>
#include <bsememf.h>   /* Get flags for memory management */

PVOID    Offset;        /* Offset to the memory pool from which
                           the memory block is to be allocated */
PPVOID   BlockOffset;   /* Pointer to the variable where the
                           offset of the suballocated memory
                           block is returned */
ULONG    Size;          /* Size in bytes of the memory
                           block requested */
APIRET   rc;            /* Return code */

   Size = 102;          /* Ask for 102 bytes.  This will be      */
                        /*   rounded to 104 bytes (a multiple    */
                        /*   of 8 bytes).                        */

   rc = DosSubAllocMem(Offset, BlockOffset, Size);
                        /* On return, the BlockOffset variable */
                        /*   will contain the address of the   */
                        /*   allocated block (from the memory  */
                        /*   pool).                            */

   if (rc != 0)
     {
       printf("DosSubAllocMem error: return code = %ld", rc);
       return;
     }
```

DosSubFreeMem – Free Suballocated Block of Memory

#define INCL_DOSMEMMGR

APIRET DosSubFreeMem (PVOID pOffset, PVOID pBlockOffset, ULONG ulSize)

DosSubFreeMem frees a block of memory that was previously allocated by DosSubAllocMem.

Parameters

pOffset (PVOID) – input

The offset of the memory pool to which the block is to be freed.

pBlockOffset (PVOID) – input

The offset of the memory block to be freed. The value specified must equal the *BlockOffset* value returned on a previous DosSubAllocMem function.

ulSize (ULONG) – input

The size, in bytes, of the memory block to be freed.

Returns

Return Code.

DosSubFreeMem returns the following values:

0	NO_ERROR
87	ERROR_INVALID_PARAMETER
312	ERROR_DOSSUB_OVERLAP
532	ERROR_DOSSUB_CORRUPTED

Remarks

DosSubFreeMem frees a block of memory that was previously allocated by DosSubAllocMem.

Size should be a multiple of 8 bytes, otherwise it will be rounded up. The maximum value for the *Size* parameter is the size of the memory pool initialized by DosSubSetMem minus 64 bytes.

Related Functions

- DosSubAllocMem
- DosSubSetMem
- DosSubUnsetMem

DosSubFreeMem – Free Suballocated Block of Memory

Example Code

This example frees a block of memory that was previously allocated from a memory pool. DosSubFreeMem returns the block to the memory pool. Assume that the *Offset* variable has been previously set to the address of the initialized memory pool, and that the *BlockOffset* variable has been previously set to the address of the block to be returned to the memory pool.

```
#define  INCL_DOSMEMMGR   /* Memory Manager values */
#include <os2.h>
#include <stdio.h>
#include <bsememf.h>   /* Get flags for memory management */

PVOID   Offset;        /* Offset of the memory
                          pool to which the block
                          is to be freed */
PVOID   BlockOffset;   /* Offset of memory
                          block to be freed */
ULONG   Size;          /* Size in bytes of
                          block to be freed */
APIRET  rc;            /* Return code */

   Size = 102;         /* Return 102 bytes.  This will be      */
                       /*   rounded to 104 bytes (a multiple   */
                       /*   of 8 bytes).                       */

   rc = DosSubFreeMem(Offset, BlockOffset, Size);

   if (rc != 0)
     {
       printf("DosSubFreeMem error: return code = %ld", rc);
       return;
     }
```

DosSubSetMem – Set a Memory Pool

#define INCL_DOSMEMMGR

APIRET DosSubSetMem (PVOID pOffset, ULONG ulFlags, ULONG ulSize)

DosSubSetMem initializes a memory pool for suballocation, or increases the size of a previously initialized memory pool.

Parameters

pOffset (PVOID) – input

The address of the memory pool to be used for suballocation.

ulFlags (ULONG) – input

Indicators describing the characteristics of the memory object that is being suballocated.

Bit 0 DOSSUB_INIT (0x00000001): This bit must be set to initialize a memory object for suballocation. Otherwise, the request is to attach a process to a shared memory pool that was previously initialized by another process using DosSubSetMem.

Bit 1 DOSSUB_GROW (0x00000002): If this bit is set, then the request is to increase the size of the memory pool being managed. Bit 0 then has no meaning.

Bit 2 DOSSUB_SPARSE_OBJ (0x00000004): Bit 2 is set if the requester wants a suballocation function to manage the commitment of the pages spanned by the memory pool.

All of the pages spanned by the object must be initially uncommitted. If this bit is clear, the suballocation function assumes that all of the pages spanned by the memory pool are valid and committed.

For a DosSubSetMem(Grow) request, the setting of this bit should be the same as when the memory pool was initialized. Otherwise, an error is returned.

Bit 3 DOSSUB_SERIALIZE (0x00000008): This bit is set if the requester requires access to the memory pool to be serialized.

For shared memory pools, the first DosSubSetMem(Init or Serialize) request causes the memory pool to be created and opened for the requesting process.

Subsequent DosSubSetMem(Attach or Serialize) requests cause the shared memory pool to be attached to the requesting process. The requesting process must first gain access to the memory object that the pool resides in. DosSubSetMem(Attach) is indicated when bit 0 is off.

On a DosSubSetMem(Grow) request, bit 3 should be the same as when the memory pool was initialized, or an error is returned.

ulSize (ULONG) – input

The size, in bytes, of the memory pool. If the size is not a multiple of 8 bytes, it is rounded down to a multiple of 8.

Returns

Return Code.

DosSubSetMem returns the following values:

0 NO_ERROR
87 ERROR_INVALID_PARAMETER
310 ERROR_DOSSUB_SHRINK

DosSubSetMem – Set a Memory Pool

Remarks

DosSubSetMem initializes a memory pool for suballocation, or increases the size of a previously initialized memory pool.

The requester must first allocate or gain access to the memory object in which the memory pool resides using one of the memory-management function calls.

All calls to DosSubSetMem must eventually be followed by a call to DosSubUnsetMem. This is necessary to allow the suballocation function to reset resources used to manage the memory pool.

The size of suballocation control information in the memory pool is 64 bytes. Therefore, the minimum size for DosSubSetMem is 72 bytes.

The requester should not issue DosSetMem or change the attributes of any pages spanned by a memory object that the suballocation function is managing. Otherwise, unpredictable results may occur.

All the pages spanned by the memory pool must have the same attributes. At least Read/Write access must have been requested for the pages spanned by the memory pool when the memory is allocated.

The DosSubSetMem(Grow) function is closely related to the memory and performance requirements of the requester as follows:

- If the requester requires the best performance possible on DosSubAllocMem and DosSubFreeMem functions, and a guarantee that those requests will not fail due to a lack of space on the swap device, the requester should not use the Sparse feature, because the suballocation function will dynamically commit pages and request swap file storage.

  This type of requester may wish to notify the suballocation function later that more committed memory is now available for the memory pool by using the DosSubSetMem(Grow) function.

- Most requesters do not have this kind of requirement. They should allow the suballocation function to manage the pages occupied by the memory pool, and they should initialize it with the Sparse attribute. This type of requester should not have to issue a DosSubSetMem(Grow) function later.

Related Functions

- DosSubAllocMem
- DosSubFreeMem
- DosSubUnsetMem

Example Code

This example initializes a memory pool. Assume that a memory object was previously allocated for the pool, and that the *Offset* variable was previously loaded with the virtual address of the memory object.

```
#define  INCL_DOSMEMMGR   /* Memory Manager values */
#include <os2.h>
#include <stdio.h>
#include <bsememf.h>   /* Get flags for memory management */

PVOID   Offset;   /* Address of the memory pool to be
                     used for suballocation */
ULONG   Flags;    /* Flags describing the memory object
                     that is being suballocated */
ULONG   Size;     /* Size in bytes of
                     the memory pool */
APIRET  rc;       /* Return code */

   Size = 20002;       /* Indicate a memory pool size of 20002   */
                       /*   bytes, which will be rounded down to */
                       /*   20000 bytes (a multiple of 8 bytes)  */

   Flags = DOSSUB_INIT | DOSSUB_SPARSE_OBJ;
                       /* Indicate that the memory pool is to */
                       /*   be initialized, and that memory   */
                       /*   commitment is to be managed       */
                       /*   internally within subsequent      */
                       /*   DosSubAllocMem calls              */

   rc = DosSubSetMem(Offset, Flags, Size);

   if (rc != 0)
     {
       printf("DosSubSetMem error: return code = %ld", rc);
       return;
     }
```

DosSubUnsetMem –
End the Use of a Memory Pool

```
#define INCL_DOSMEMMGR
```

APIRET DosSubUnsetMem (PVOID pOffset)

DosSubUnsetMem ends the use of a memory pool.

Parameters

pOffset (PVOID) – input

The offset of the memory pool whose use is being terminated.

Returns

Return Code.

DosSubUnsetMem returns the following values:

0	NO_ERROR
532	ERROR_DOSSUB_CORRUPTED

Remarks

DosSubUnsetMem ends the use of a memory pool.

All calls to DosSubSetMem must eventually be followed by a call to DosSubUnsetMem.

This call allows the suballocation function to release the resources that it used to manage the suballocation of the memory object. The call to DosSubUnsetMem must occur before the memory object is freed.

Related Functions

- DosSubAllocMem
- DosSubFreeMem
- DosSubSetMem

Example Code

This example shows the termination of a memory pool. Assume that the address of the memory pool was placed into *Offset* already.

```
#define  INCL_DOSMEMMGR   /* Memory Manager values */
#include <os2.h>
#include <stdio.h>
#include <bsememf.h>   /* Get flags for memory management */

PVOID   Offset;   /* Offset of the memory
                     pool whose use is
                     being terminated */
APIRET  rc;       /* Return code */

   rc = DosSubUnsetMem(Offset);

   if (rc != 0)
     {
       printf("DosSubUnsetMem error: return code = %ld", rc);
       return;
     }
```

DosSuspendThread – Suspend Execution of Another Thread

#define INCL_DOSPROCESS

APIRET DosSuspendThread (TID idThreadID)

DosSuspendThread temporarily suspends execution of another thread within the current process until DosResumeThread is issued.

Parameters

idThreadID (TID) – input

Thread identifier of the thread to be suspended.

Returns

Return Code.

DosSuspendThread returns the following values:

0	NO_ERROR
309	ERROR_INVALID_THREADID

Remarks

A thread's execution is suspended when another thread in its process issues DosSuspendThread, specifying the ID of the target thread. The thread may not be suspended immediately because it may have locked some system resources that have to be freed first. However, the thread is not allowed to execute further application program instructions until a corresponding DosResumeThread is issued.

DosSuspendThread permits the suspension of only one other thread within the current process. If a thread needs to disable all thread switching within its process so that the calling thread can execute time-critical code, it issues DosEnterCritSec and DosExitCritSec.

Related Functions

- DosCreateThread
- DosEnterCritSec
- DosResumeThread

Example Code

This example temporarily suspends the execution of another thread within the same process. A subsequent call to DosResumeThread can restart the suspended thread. Assume that the thread ID of the target thread has been placed into *ThreadID* already.

```
#define INCL_DOSPROCESS     /* Process and thread values */
#include <os2.h>
#include <stdio.h>

TID     ThreadID;     /* Thread identifier */
APIRET  rc;           /* Return code */

   rc = DosSuspendThread(ThreadID);

   if (rc != 0)
     {
       printf("DosSuspendThread error: return code = %ld", rc);
       return;
     }
```

DosTransactNPipe –
Perform Transaction on a Named Pipe

```
#define INCL_DOSNMPIPES
```

```
APIRET DosTransactNPipe (HPIPE hpipeHandle, PVOID pOutBuffer, ULONG ulOutBufferLen,
                         PVOID pInBuffer, ULONG ulInBufferLen, PULONG pBytesRead)
```

DosTransactNPipe writes to a duplex message pipe, then reads from it.

Parameters

hpipeHandle (HPIPE) – input

A named-pipe handle returned by DosCreateNPipe (for a server process) or by DosOpen (for a client process).

pOutBuffer (PVOID) – input

A pointer to the buffer that is to be written to the pipe.

ulOutBufferLen (ULONG) – input

The number of bytes to be written.

pInBuffer (PVOID) – output

A pointer to the buffer for returned data.

ulInBufferLen (ULONG) – input

The maximum size, in bytes, of returned data.

pBytesRead (PULONG) – output

A pointer to the number of bytes read.

Returns

Return Code.

DosTransactNPipe returns the following values:

0	NO_ERROR
11	ERROR_BAD_FORMAT
230	ERROR_BAD_PIPE
231	ERROR_PIPE_BUSY
233	ERROR_PIPE_NOT_CONNECTED
234	ERROR_MORE_DATA

Remarks

DosTransactNPipe is intended for use only on a duplex message pipe that is in message-read mode. If this function is issued for a pipe that is not a duplex message pipe, ERROR_BAD_FORMAT is returned.

The current setting of the pipe's blocking mode has no effect on this function; that is, even if the pipe is in nonblocking mode, DosTransactNPipe writes the entire *OutBuffer* to the pipe, and does not return until it reads a response from the pipe into *InBuffer*. If *InBuffer* is too small to contain the response message, ERROR_MORE_DATA is returned.

The function does not succeed if there is any unread data in the pipe, or if the pipe is not in message-read mode.

Related Functions

- DosCallNPipe
- DosConnectNPipe
- DosCreateNPipe
- DosDisConnectNPipe
- DosPeekNPipe
- DosQueryNPHState
- DosQueryNPipeInfo
- DosQueryNPipeSemState
- DosSetNPHState
- DosSetNPipeSem
- DosWaitNPipe
- DosClose
- DosDupHandle
- DosOpen
- DosRead
- DosResetBuffer
- DosWrite
- DosCreateEventSem
- DosCloseMuxWaitSem
- DosWaitEventSem
- DosWaitMuxWaitSem

DosTransactNPipe — Perform Transaction on a Named Pipe

Example Code

This example performs a transaction to a named pipe. In the transaction, a message is written to the pipe, and then the caller waits until a response message is read from the pipe.

```
#define INCL_DOSNMPIPES   /* Named-pipe values */
#include <os2.h>
#include <stdio.h>

HPIPE   Handle;         /* Pipe handle */
UCHAR   OutBuffer[800]; /* Write-buffer address  */
ULONG   OutBufferLen;   /* Write-buffer length   */
UCHAR   InBuffer[800];  /* Read-buffer address   */
ULONG   InBufferLen;    /* Read-buffer length    */
ULONG   BytesRead;      /* Bytes read (returned) */
APIRET  rc;             /* Return code */

   strcpy(OutBuffer,"Command 1: Start Proc 1");
                        /* Set output buffer to contain the */
                        /*   desired message to be sent     */

   OutBufferLen = strlen(OutBuffer);
                        /* Set length indicator for output   */
                        /*   buffer                         */

   InBufferLen = 800;   /* Max data length for input */
                        /*   (return) buffer         */

   rc = DosTransactNPipe(Handle, OutBuffer, OutBufferLen, InBuffer,
                         InBufferLen, &BytesRead);
                        /* On successful return, the input      */
                        /*   buffer (InBuffer) will contain a   */
                        /*   response message, and the variable */
                        /*   BytesRead will contain the size    */
                        /*   of that response message           */

   if (rc != 0)
     {
       printf("DosTransactNPipe error: return code = %ld",rc);
       return;
     }
```

DosUnsetExceptionHandler – Unset Exception Handler

#define INCL_DOSEXCEPTIONS

APIRET DosUnsetExceptionHandler (PEXCEPTIONREGISTRATIONRECORD pppERegRec)

DosUnsetExceptionHandler removes an exception handler from a thread's chain of exception handlers.

Parameters

pppERegRec (PEXCEPTIONREGISTRATIONRECORD) – input

A pointer to the exception registration record that describes the exception handler to be unregistered.

Returns

Return Code.

DosUnsetExceptionHandler returns the following values:

0	NO_ERROR
87	ERROR_INVALID_PARAMETER

Remarks

DosUnsetExceptionHandler deregisters (removes) an exception handler from a thread's chain of registered exception handlers.

For a detailed list of the system exceptions, see Appendix C, "System Exceptions" on page C-1.

Related Functions

- DosAcknowledgeSignalException
- DosEnterMustComplete
- DosExitMustComplete
- DosRaiseException
- DosSendSignalException
- DosSetExceptionHandler
- DosSetSignalExceptionFocus
- DosUnwindException

DosUnsetExceptionHandler — Unset Exception Handler

Example Code

This example removes an exception handler from the current thread's chain of registered exception handlers.

Assume that the exception registration record is intact from a previous call to DosSetExceptionHandler.

```
#define INCL_DOSEXCEPTIONS   /* Exception values */
#include <os2.h>
#include <stdio.h>

typedef struct  SysERegRec {
   PEXCEPTIONREGISTRATIONRECORD pLink;
   ULONG (_cdecl *pSysEH)(PEXCEPTIONREPORTRECORD,
                          PEXCEPTIONREGISTRATIONRECORD,
                          PCONTEXTRECORD,
                          PVOID);
} SYSEREGREC;

SYSEREGREC    RegRec;    /* Structure to pass to exception handler */
APIRET        rc;        /* Return code */

   rc = DosUnsetExceptionHandler((PEXCEPTIONREGISTRATIONRECORD)
                                      &RegRec);

   if (rc != 0)
     {
       printf("DosUnsetExceptionHandler error: return code = %ld", rc);
       return;
     }
```

DosUnwindException — Unwind Exception

#define INCL_DOSEXCEPTIONS

```
APIRET DosUnwindException (PEXCEPTIONREGISTRATIONRECORD pphandler,
                           PVOID ppTargetIP, PEXCEPTIONREPORTRECORD pppERepRec)
```

DosUnwindException calls and removes exception handlers from a thread's chain of exception handlers.

Parameters

pphandler (PEXCEPTIONREGISTRATIONRECORD) — input

This parameter can have one of the following values:

Value	Definition
Address	A pointer to the exception registration record where the unwind operation should stop.
0	(UNWIND_ALL) An exit unwind operation is performed. This removes all exception handlers from the thread, and ends the thread.
–1	(END_OF_CHAIN) All exception handlers for the thread are unwound.

ppTargetIP (PVOID) — input

A pointer to where DosUnwindException branches after calling all applicable handlers.

pppERepRec (PEXCEPTIONREPORTRECORD) — input

An optional pointer to an exception record. Set this field to zero if it is not used.

Returns

Return Code.

DosUnwindException returns the following values:

0	NO_ERROR
1	ERROR_INVALID_FUNCTION

Remarks

DosUnwindException "unwinds" (calls and removes) exception handlers from a thread's chain of registered exception handlers. It can unwind up to but not including a specified exception handler, or it can unwind all the exception handlers.

Each exception handler in the linked list from the Thread Information Block (TIB) is called with the unwind bit in the Exception Report Record structure set, indicating an unwind operation. If the call to the exception handler returns, the Exception Registration Record is removed from the linked list, and the next exception handler is processed.

For a detailed list of the system exceptions, see Appendix C, "System Exceptions" on page C-1.

DosUnwindException – Unwind Exception

Related Functions

- DosAcknowledgeSignalException
- DosEnterMustComplete
- DosExitMustComplete
- DosRaiseException
- DosSendSignalException
- DosSetExceptionHandler
- DosSetSignalExceptionFocus
- DosUnsetExceptionHandler

Example Code

This example is assumed to be called from within an exception handler. It unwinds the exception handlers up to, but not including, the specified exception handler. It causes program execution to be resumed eventually at the target address that is assumed to have been placed into *pTargetIP* already. Assume that the Exception Registration Record pointer and the Exception Report Record pointer were both obtained from the parameters that were passed to the exception handler.

```
#define INCL_DOSEXCEPTIONS   /* Exception values */
#include <os2.h>
#include <stdio.h>

PEXCEPTIONREGISTRATIONRECORD phandler;
                  /* Pointer to exception registration record
                     where unwind should stop */
PVOID  pTargetIP;  /* Pointer to where DosUnwindException branches
                     after calling all applicable handlers */
PEXCEPTIONREPORTRECORD  pERepRec;
                  /* Pointer to exception report record */

APIRET  rc;     /* Return code */

   rc = DosUnwindException(phandler, pTargetIP, pERepRec);

   if (rc != 0)
     {
       printf("DosUnwindException error: return code = %ld",rc);
       return;
     }
```

DosWaitChild – Place Current Thread in a Wait State Until Child Process Ends

#define INCL_DOSPROCESS

```
APIRET DosWaitChild (ULONG ulActionCode, ULONG ulWaitOption,
                     PRESULTCODES ppReturnCodes, PPID ppRetProcessID,
                     PID idProcessID)
```

DosWaitChild places the current thread into a wait state until an asynchronous child process ends. When the process ends, its process identifier, termination code, and result code are returned to the caller.

Parameters

ulActionCode (ULONG) – input

Indicates which process the current thread is waiting to terminate. The values of this field are as follows:

Value	Definition
0	(DCWA_PROCESS): The child process indicated by *ProcessID*.
1	(DCWA_PROCESSTREE): The child process indicated by *ProcessID* and all of its child processes.

ulWaitOption (ULONG) – input

Indicates whether to return if no child process ends. The values of this field are as follows:

Value	Definition
0	(DCWW_WAIT): Wait if no child process ends or until no child processes are outstanding.
1	(DCWW_NOWAIT): Do not wait for child processes to end.

ppReturnCodes (PRESULTCODES) – output

Address of the structure that contains the termination code and the result code indicating the reason for the child's termination. If no process furnishes a result code, the system provides the value –1.

This structure consists of two doublewords as follows:

codeTerminate (**ULONG**)
The termination code furnished by the system describing why the child terminated. The values of this field are as follows:

Value	Definition
0	(TC_EXIT): Normal exit
1	(TC_HARDERROR): Hard-error halt
2	(TC_TRAP): Trap operation for a 16-bit child process
3	(TC_KILLPROCESS): Unintercepted DosKillProcess
4	(TC_EXCEPTION): Exception operation for a 32-bit child process

codeResult (**ULONG**)
Result code specified by the terminating process on its last call to DosExit.

ppRetProcessID (PPID) – output

Address of the process identifier of the ending process.

DosWaitChild – Place Current Thread in a Wait State Until Child Process Ends

idProcessID (PID) – input

Identifier of the process whose termination is being waited for. The values of this field are as follows:

Value	Definition
0	Any child process. The current thread waits until any child process that was executed with a return code ends, or until there are no more child processes of any type to wait for.
<> 0	The indicated child process and all its descendants.

Returns

Return Code.

DosWaitChild returns the following values:

0	NO_ERROR
13	ERROR_INVALID_DATA
128	ERROR_WAIT_NO_CHILDREN
129	ERROR_CHILD_NOT_COMPLETE
184	ERROR_NO_CHILD_PROCESS
303	ERROR_INVALID_PROCID

Remarks

DosWaitChild waits for completion of a child process whose execution is asynchronous to that of its parent process. The child process is created by DosExecPgm with a value of 2 specified for *ExecFlags*. If the child process has multiple threads, the result code returned by DosWaitChild is the one passed to it by the DosExit request that ends the process.

DosWaitChild also can wait for the completion of descendant processes of a child process before it returns. However, it does not report status for descendant processes.

If there are no child processes (either active, or ended but with a return code), then DosWaitChild returns an error. If no child processes have ended, DosWaitChild can wait until one ends before returning to the parent process.

To verify that a given return code is from a specific child process, the process identifier must be checked.

To wait for all child processes and descendants to end, it is necessary to:

1. Issue DosWaitChild with a value of 0 for *ProcessID* (wait until *any* child process has ended).
2. When this DosWaitChild returns, issue a DosWaitChild request with *ProcessID* equal to the value returned for *RetProcessID* on the previous DosWaitChild request, and a value of 1 for *ActionCode* (wait for the indicated process and *all* its child processes).
3. Repeat steps 1 and 2 above until the "No child process exists" return code is received.

DosWaitChild will wait for any child processes, regardless of whether or not they were executed with a result code (by calling DosExecPgm with a value of 2 for *ExecFlags*). DosWaitChild will not return to the caller until a process *with* a return code ends, or until there are no more child processes (of any type) to wait for.

DosWaitChild – Place Current Thread in a Wait State Until Child Process Ends

Related Functions

- DosExecPgm
- DosExit
- DosKillProcess
- DosKillThread
- DosWaitThread

Example Code

This example starts a child session (the program simple.exe) and then waits for the termination of the child process.

DosWaitChild — Place Current Thread in a Wait State Until Child Process Ends

```
#define INCL_DOSPROCESS         /* Process and thread values */
#include <os2.h>
#include <stdio.h>

#ifndef _RESULTCODES
 typedef struct _RESULTCODES {  /* Result codes */

   ULONG codeTerminate;          /* Termination Code */
   ULONG codeResult;             /* Exit Code */

 } RESULTCODES;
#endif

#define START_PROGRAM "simple.exe"

CHAR        LoadError[100];
PSZ         Args;
PSZ         Envs;
RESULTCODES ReturnCodes;
ULONG       Pid;            /* Process ID (returned) */
APIRET      rc;             /* Return code */

  strcpy(Args, "-a2 -l");    /* Pass arguments '-a2' and '-l' */

  rc = DosExecPgm(LoadError,                /* Object name buffer */
                  sizeof(LoadError),        /* Length of object name
                                               buffer */
                  EXEC_ASYNCRESULT,         /* Asynchronous/Trace
                                                   flags */
                  Args,                     /* Argument string */
                  Envs,                     /* Environment string */
                  &ReturnCodes,             /* Termination codes */
                  START_PROGRAM);           /* Program file name */

  if (rc != 0)
    {
      printf("DosExecPgm error: return code = %ld",rc);
      return;
    }

  rc = DosWaitChild(DCWA_PROCESS,        /* Execution options */
                DCWW_WAIT,               /* Wait options */
                &ReturnCodes,            /* Termination codes */
                &Pid,                    /* Process ID (returned) */
                ReturnCodes.codeTerminate);  /* Process ID of process
                                                to wait for */
  if (rc != 0)
    {
      printf("DosWaitChild error: return code = %ld",rc);
      return;
    }
```

DosWaitEventSem – Wait Event Semaphore

#define INCL_DOSSEMAPHORES

APIRET DosWaitEventSem (HEV hev, ULONG ulTimeout)

DosWaitEventSem waits for an event semaphore to be posted.

Parameters

hev (HEV) – input

The handle of the event semaphore to wait for.

ulTimeout (ULONG) – input

The time-out in milliseconds. This is the maximum amount of time the user wants to allow the thread to be blocked.

This parameter can also have the following values:

Value	Definition
0	(SEM_IMMEDIATE_RETURN) DosWaitEventSem returns without blocking the calling thread.
–1	(SEM_INDEFINITE_WAIT) DosWaitEventSem blocks the calling thread indefinitely.

Returns

Return Code.

DosWaitEventSem returns the following values:

0	NO_ERROR
6	ERROR_INVALID_HANDLE
8	ERROR_NOT_ENOUGH_MEMORY
95	ERROR_INTERRUPT
640	ERROR_TIMEOUT

Remarks

DosWaitEventSem enables a thread to wait for an event semaphore to be posted.

This function can be called by any thread in the process that created the semaphore. Threads in other processes can also call this function, but they must first gain access to the semaphore by calling DosOpenEventSem.

Related Functions

- DosCloseEventSem
- DosCreateEventSem
- DosOpenEventSem
- DosPostEventSem
- DosQueryEventSem
- DosResetEventSem

DosWaitEventSem — Wait Event Semaphore

Example Code

This example causes the calling thread to wait until the specified event semaphore is posted. Assume that the handle of the semaphore has been placed into *hev* already.

ulTimeout is the number of milliseconds that the calling thread will wait for the event semaphore to be posted. If the specified event semaphore is not posted during this time interval, the request times out.

```
#define INCL_DOSSEMAPHORES   /* Semaphore values */
#include <os2.h>
#include <stdio.h>

#ifndef ERROR_TIMEOUT
    #define  ERROR_TIMEOUT    640
    #define  ERROR_INTERRUPT   95
#endif

HEV     hev;        /* Event semaphore handle */
ULONG   ulTimeout;  /* Number of milliseconds to wait */
APIRET  rc;         /* Return code */

   ulTimeout = 60000;  /* Wait for a maximum of 1 minute */

   rc = DosWaitEventSem(hev, ulTimeout);

   if (rc == ERROR_TIMEOUT)
     {
       printf("DosWaitEventSem call timed out");
       return;
     }

   if (rc == ERROR_INTERRUPT)
     {
       printf("DosWaitEventSem call was interrupted");
       return;
     }

   if (rc != 0)
     {
       printf("DosWaitEventSem error: return code = %ld", rc);
       return;
     }
```

DosWaitMuxWaitSem – Wait MuxWait Semaphore

#define INCL_DOSSEMAPHORES

APIRET DosWaitMuxWaitSem (HMUX hmux, ULONG ulTimeout, PULONG pUser)

DosWaitMuxWaitSem waits for a muxwait semaphore to clear.

Parameters

hmux (HMUX) – input

The handle of the muxwait semaphore to wait for.

ulTimeout (ULONG) – input

The time-out in milliseconds. This is the maximum amount of time the user wants to allow the thread to be blocked.

This parameter can also have the following values:

Value	Definition
0	(SEM_IMMEDIATE_RETURN) DosWaitMuxWaitSem returns without blocking the calling thread.
–1	(SEM_INDEFINITE_WAIT) DosWaitMuxWaitSem blocks the calling thread indefinitely.

pUser (PULONG) – output

A pointer to receive the user field (from the muxwait semaphore data structure) of the semaphore that was posted or released.

If DCMW_WAIT_ANY was specified in the *flAttr* parameter when the muxwait semaphore was created, this will be the user field of the semaphore that was posted or released. If the muxwait semaphore consists of mutex semaphores, any mutex semaphore that is released is owned by the caller.

If DCMW_WAIT_ALL was specified in the *flAttr* parameter when the muxwait semaphore was created, this will be the user field of the *last* semaphore that was posted or released. (If the thread did not block, the last semaphore that was posted or released will also be the last semaphore in the muxwait-semaphore list.) If the muxwait semaphore consists of mutex semaphores, all of the mutex semaphores that are released are owned by the caller.

Returns

Return Code.

DosWaitMuxWaitSem returns the following values:

0	NO_ERROR
6	ERROR_INVALID_HANDLE
8	ERROR_NOT_ENOUGH_MEMORY
87	ERROR_INVALID_PARAMETER
95	ERROR_INTERRUPT
103	ERROR_TOO_MANY_SEM_REQUESTS
105	ERROR_SEM_OWNER_DIED
286	ERROR_EMPTY_MUXWAIT
287	ERROR_MUTEX_OWNED
292	ERROR_WRONG_TYPE
640	ERROR_TIMEOUT

DosWaitMuxWaitSem – Wait MuxWait Semaphore

Remarks

DosWaitMuxWaitSem enables a thread to wait for a muxwait semaphore to clear.

This function can be issued by any thread in the process that created the semaphore. Threads in other processes can also issue this function, but they must first gain access to the semaphore by issuing DosOpenMuxWaitSem.

Related Functions

- DosAddMuxWaitSem
- DosCloseMuxWaitSem
- DosCreateMuxWaitSem
- DosDeleteMuxWaitSem
- DosOpenMuxWaitSem
- DosQueryMuxWaitSem

Example Code

This example waits for a muxwait semaphore to clear. Assume that the handle of the semaphore has been placed into *hmux* already.

ulTimeout is the number of milliseconds that the calling thread will wait for the muxwait semaphore to clear. If the specified muxwait semaphore is not cleared during this time interval, the request times out.

```
#define INCL_DOSSEMAPHORES   /* Semaphore values */
#include <os2.h>
#include <stdio.h>

#ifndef ERROR_TIMEOUT
    #define  ERROR_TIMEOUT    640
    #define  ERROR_INTERRUPT   95
#endif

HMUX     hmux;      /* Muxwait semaphore handle */
ULONG    ulTimeout; /* Number of milliseconds to wait */
ULONG    ulUser;    /* User field for the semaphore that was
                       posted or released (returned) */
APIRET   rc;        /* Return code */

   ulTimeout = 60000;  /* Wait for a maximum of 1 minute */

   rc = DosWaitMuxWaitSem(hmux, ulTimeout, &ulUser);
                       /* On successful return, the ulUser    */
                       /*   variable contains the user        */
                       /*   identifier of the semaphore       */
                       /*   that caused the wait to           */
                       /*   terminate.  If the caller had     */
                       /*   to wait for all the semaphores    */
                       /*   within the muxwait semaphore to   */
                       /*   clear, then the value corresponds */
                       /*   to the last semaphore within the  */
                       /*   muxwait semaphore to clear.  If   */
                       /*   the caller had to wait for any    */
                       /*   semaphore with the muxwait        */
                       /*   semaphore to clear, then the      */
                       /*   value corresponds to that         */
                       /*   semaphore.                        */

   if (rc == ERROR_TIMEOUT)
     {
       printf("DosWaitMuxWaitSem call timed out");
```

```
    return;
  }

if (rc == ERROR_INTERRUPT)
  {
    printf("DosWaitMuxWaitSem call was interrupted");
    return;
  }

if (rc != 0)
  {
    printf("DosWaitMuxWaitSem error: return code = %ld", rc);
    return;
  }
```

DosWaitNPipe – Wait Named Pipe Instance

#define INCL_DOSNMPIPES

APIRET DosWaitNPipe (PSZ pszFileName, ULONG ulTimeOut)

DosWaitNPipe waits for a named-pipe instance to become available.

Parameters

pszFileName (PSZ) – input

A pointer to the ASCIIZ name of the pipe to be opened.

ulTimeOut (ULONG) – input

The maximum time, in milliseconds, to wait for a named-pipe instance to become available. When a value of 0 is specified, DosWaitNPipe uses the value of *TimeOut* that was specified when the pipe was created with DosCreateNPipe. When a value of -1 is specified, DosWaitNPipe waits indefinitely.

Returns

Return Code.

DosWaitNPipe returns the following values:

0	NO_ERROR
2	ERROR_FILE_NOT_FOUND
95	ERROR_INTERRUPT
231	ERROR_PIPE_BUSY

Remarks

DosWaitNPipe enables a client process to wait for a named-pipe instance to become available when all instances are busy. It should be used only when ERROR_PIPE_BUSY is returned from a call to DosOpen.

The *TimeOut* parameter of DosWaitNPipe places a limit on the amount of time the calling process waits for a named-pipe instance to become available, as follows:

- If the time limit is reached before a pipe instance becomes available, ERROR_PIPE_BUSY is returned.
- If a time limit of 0 is specified, the system uses the default time-out value that was specified when the pipe was created with DosCreateNPipe.
- If a time limit of -1 is specified, DosWaitNPipe waits indefinitely.

If DosWaitNPipe is successful, the client must again call DosOpen to gain access to the pipe instance.

If more than one client process is blocked on a DosWaitNPipe request, the system gives the next available pipe instance to the process whose thread has the highest priority. If all of the waiting threads have the same priority, the thread that has been waiting the longest receives the next pipe instance.

Note: The priority of a thread can be changed by calling DosSetPriority.

Related Functions

- DosCallNPipe
- DosConnectNPipe
- DosCreateNPipe
- DosDisConnectNPipe
- DosPeekNPipe
- DosQueryNPHState
- DosQueryNPipeInfo
- DosQueryNPipeSemState
- DosSetNPHState
- DosSetNPipeSem
- DosTransactNPipe
- DosClose
- DosDupHandle
- DosOpen
- DosRead
- DosResetBuffer
- DosWrite

Example Code

This example waits for an available instance of a named pipe. The example assumes that an attempt to connect to a named pipe through a call to DosOpen ended with a "pipe busy" error return code. In such cases, a process can wait for an available instance of the named pipe by issuing DosWaitNPipe.

```
#define INCL_DOSNMPIPES   /* Named-pipe values */
#include <os2.h>
#include <stdio.h>

UCHAR   FileName[40]; /* Pipe name */
ULONG   TimeOut;      /* Maximum wait time */
APIRET  rc;           /* Return code */

   strcpy(FileName,"\\PIPE\\PIPE1");

   TimeOut = 30000;   /* Wait for up to thirty seconds to   */
                      /*   connect to the named pipe (units */
                      /*   are in milliseconds)             */

   rc = DosWaitNPipe(FileName, TimeOut);

   if (rc != 0)
     {
       printf("DosWaitNPipe error: return code = %ld",rc);
       return;
     }
```

DosWaitThread –
Place Current Thread into a Wait State

#define INCL_DOSPROCESS

APIRET DosWaitThread (PTID ppThreadID, ULONG ulWaitOption)

DosWaitThread places the current thread into a wait state until another thread in the current process has ended. It then returns the thread identifier of the ending thread.

Parameters

ppThreadID (PTID) – input/output

On input, the address of the ThreadID of the thread of interest. If *ThreadID* is zero, the current thread waits until the next thread in the process has ended. If *ThreadID* is non-zero, the current thread waits until the indicated thread has ended.

On output, the ThreadID of the ended thread is returned in this field.

ulWaitOption (ULONG) – input

Indicates whether to return if no thread has ended. The values of this field are as follows:

Value	Definition
0	(DCWW_WAIT): The current thread waits until a thread ends. If a thread has already ended, the call returns immediately with the *ThreadID*.
1	(DCWW_NOWAIT): The current thread does not wait if no threads have ended.

Returns

Return Code.

DosWaitThread returns the following values:

0	NO_ERROR
294	ERROR_THREAD_NOT_TERMINATED
309	ERROR_INVALID_THREADID

Remarks

DosWaitThread is used to wait for termination of threads within a process. It is usually used so that thread resources (for example, the stack) can be recovered when a thread ends. DosWaitThread waits on any thread within the current process, or on a specific thread within the process, based on the *ThreadID* parameter's contents. *WaitOption* allows the caller the option of waiting until a thread ends, or getting immediate return and status. If no thread has ended and the *NoWait* option is specified, the *ThreadID* field is preserved.

If DosWaitThread is called with the input *ThreadID* set to the current thread (the thread attempts to wait on its own termination), the ERROR_INVALID_THREADID error code is returned. ERROR_INVALID_THREADID is also returned if a caller attempts to wait on the termination of the thread with a *ThreadID* of 1.

Related Functions

- DosCreateThread
- DosKillThread
- DosWaitChild

Example Code

This example places the calling thread into a wait state until a specified thread has ended. Assume that the thread ID of the specified thread has been placed into *ThreadID* already.

```
#define INCL_DOSPROCESS     /* Process and thread values */
#include <os2.h>
#include <stdio.h>

TID     ThreadID;     /* Identifier of the thread
                         to wait on */
ULONG   WaitOption;   /* Wait options */
APIRET  rc;           /* Return code */

   WaitOption = 0;    /* Indicate that the calling thread  */
                      /*   intends to wait until the       */
                      /*   specified thread has ended      */

   rc = DosWaitThread(&ThreadID, WaitOption);

   if (rc != 0)
     {
       printf("DosWaitThread error: return code = %ld", rc);
       return;
     }
```

DosWrite –
Write to a File from a Buffer

#define INCL_DOSFILEMGR

```
APIRET DosWrite (HFILE FileHandle, PVOID pBufferArea, ULONG ulBufferLength,
                 PULONG pBytesWritten)
```

DosWrite writes a specified number of bytes from a buffer to the specified file.

Parameters

FileHandle (HFILE) – input

File handle from DosOpen.

pBufferArea (PVOID) – input

Address of the buffer that contains the data to write.

ulBufferLength (ULONG) – input

Number of bytes to write.

pBytesWritten (PULONG) – output

Address of the variable to receive the number of bytes actually written.

Returns

Return Code.

DosWrite returns the following values:

0	NO_ERROR
5	ERROR_ACCESS_DENIED
6	ERROR_INVALID_HANDLE
19	ERROR_WRITE_PROTECT
26	ERROR_NOT_DOS_DISK
29	ERROR_WRITE_FAULT
33	ERROR_LOCK_VIOLATION
109	ERROR_BROKEN_PIPE

Remarks

DosWrite begins to write at the current file-pointer position. The file pointer is automatically moved by read and write operations. It can be moved to a desired position by issuing DosSetFilePtr.

If the specified file has been opened using the write-through flag, DosWrite writes the data to the disk before returning. Upon return to the caller, *BytesWritten* contains the number of bytes actually written.

If there is not enough space on the disk or diskette to write all of the bytes specified by *BufferLength*, then DosWrite does not write any bytes. Upon return to the caller, *BytesWritten* contains zero.

A value of zero for *BufferLength* is not considered an error. No data transfer occurs, and there is no effect on the file or the file pointer.

If the file is read-only, the write operation to the file is not performed.

If you issue DosOpen with the Direct Open flag set to 1 in the *OpenMode* parameter, you have direct access to an entire disk or diskette volume, independent of the file system. You must lock the logical volume before accessing it, and you must unlock the logical volume when you are finished accessing it. Issue DosDevIOCtl for Category 8, Function 0 to lock the logical volume, and for Category 8,

Function 1 to unlock the logical volume. While the logical volume is locked, no other process can access it.

Named-Pipe Considerations

DosWrite also is used to write bytes or messages to a named pipe.

Each write operation to a message pipe writes a message whose size is the length of the write operation. DosWrite automatically encodes message lengths in the pipe, so applications need not encode this information in the buffer being written.

Write operations in blocking mode always write all requested bytes before returning.

In nonblocking mode, DosWrite returns either with all bytes written or none written. DosWrite returns with no bytes written when it would have to divide the message into blocks in order to complete the request. This can occur when there is not enough space left in the pipe, or when the pipe is currently being written to by another client. If this occurs, DosWrite returns immediately with a value of zero for *BytesWritten*, indicating that no bytes were written.

For a byte pipe, if the number of bytes to be written exceeds the space available in the pipe, DosWrite writes as many bytes as it can, and returns with the number of bytes actually written in *BytesWritten*.

An attempt to write to a pipe whose other end has been closed returns ERROR_BROKEN_PIPE.

Related Functions

- DosOpen
- DosRead
- DosSetFilePtr

Example Code

This example opens a file, and writes data to the file.

```
#define INCL_DOSFILEMGR   /* File Manager values */
#include <os2.h>
#include <stdio.h>

#define OPEN_FILE 0x01
#define CREATE_FILE 0x10
#define FILE_ARCHIVE 0x20
#define FILE_EXISTS OPEN_FILE
#define FILE_NOEXISTS CREATE_FILE
#define DASD_FLAG 0
#define INHERIT 0x80
#define WRITE_THRU 0
#define FAIL_FLAG 0
#define SHARE_FLAG 0x10
#define ACCESS_FLAG 0x02

#define FILE_NAME "test.dat"
#define FILE_SIZE 800L
#define FILE_ATTRIBUTE FILE_ARCHIVE
#define EABUF 0L

HFILE   FileHandle;
ULONG   Wrote;              /* Number of bytes written */
ULONG   Action;             /* Action taken by DosOpen */
UCHAR   FileData[100];      /* Data to write */
APIRET  rc;                 /* Return code */
```

DosWrite –
Write to a File from a Buffer

```
Action = 2;
strcpy(FileData, "Data...");

 rc = DosOpen(FILE_NAME,                 /* File path name */
              &FileHandle,              /* File handle */
              &Action,                  /* Action taken */
              FILE_SIZE,                /* File primary allocation */
              FILE_ATTRIBUTE,           /* File attribute */
              FILE_EXISTS | FILE_NOEXISTS,    /* Open function type */
              DASD_FLAG | INHERIT |           /* Open mode of the file */
              WRITE_THRU | FAIL_FLAG |
              SHARE_FLAG | ACCESS_FLAG,
              EABUF);                   /* No extended attributes */

 if (rc != 0)
   {
     printf("DosOpen error: return code = %ld",rc);
     return;
   }

 rc = DosWrite(FileHandle,           /* File handle */
              (PVOID) FileData,     /* User buffer */
              sizeof(FileData),     /* Buffer length */
              &Wrote);              /* Bytes written */

 if (rc != 0)
   {
     printf("DosWrite error: return code = %ld",rc);
     return;
   }
```

DosWriteQueue — Write Queue

```
#define INCL_DOSQUEUES

APIRET DosWriteQueue (HQUEUE QueueHandle, ULONG ulRequest, ULONG ulDataLength,
                      PVOID pDataBuffer, ULONG ulElemPriority)
```

DosWriteQueue adds an element to a queue.

Parameters

QueueHandle (HQUEUE) — input

The handle of the queue to which an element is to be added.

ulRequest (ULONG) — input

A doubleword value to be passed with the queue element. It is used by an application to code an event. The data is understood by the thread that is adding the element to the queue, as well as by the thread that receives the queue element. There is no special meaning to this data, and the operating system does not alter it.

ulDataLength (ULONG) — input

The length, in bytes, of the data that is being sent to the queue.

pDataBuffer (PVOID) — input

A pointer to the buffer that contains the data to be placed into the queue.

ulElemPriority (ULONG) — input

The priority value of the element that is being added to the queue. This parameter is valid only for queues that were created as priority-based queues, as specified in the *QueueFlags* parameter of DosCreateQueue. *ElemPriority* is a numerical value in the range of 0 to 15, with 15 being the highest priority.

- If the priority value is 15, the element is added to the top of the queue.
- If the priority value is 0, the element is added as the last element in the queue.
- Elements with the same priority value are grouped together in FIFO (first in, first out) order.

If you assign a value greater than 15 to *ElemPriority*, the system sets *ElemPriority* to 15. No error code is returned for this condition.

Returns

Return Code.

DosWriteQueue returns the following values:

0	NO_ERROR
334	ERROR_QUE_NO_MEMORY
337	ERROR_QUE_INVALID_HANDLE

Remarks

DosWriteQueue adds an element to the specified queue. A client process must request access to the queue by calling DosOpenQueue before it can issue this function. The server process and its threads do not need to issue DosOpenQueue, because they already have access to the queue.

If the queue was created as a priority-based queue (as specified in the *QueueFlags* parameter of DosCreateQueue), then the priority of the element that is being added must be specified.

DosWriteQueue — Write Queue

If the server process has closed the queue before this request is issued, ERROR_QUE_INVALID_HANDLE is returned.

Related Functions

- DosCloseQueue
- DosCreateQueue
- DosOpenQueue
- DosPeekQueue
- DosPurgeQueue
- DosQueryQueue
- DosReadQueue

Example Code

This example adds an element to a queue. Assume that the caller has placed the handle of the queue into *QueueHandle* already. Assume that *DataBuffer* has been set to point to a data element in shared memory, and that *DataLength* has been set to contain the length of the data element in shared memory.

```
#define INCL_DOSQUEUES   /* Queue values */
#include <os2.h>
#include <stdio.h>

HQUEUE   QueueHandle;   /* Queue handle */
ULONG    Request;       /* Request-identification data */
ULONG    DataLength;    /* Length of element being added */
PVOID    DataBuffer;    /* Element being added */
ULONG    ElemPriority;  /* Priority of element being added */
APIRET   rc;            /* Return code */

   Request = 0;         /* Assume that no special data is being */
                        /*   sent along with this write request */

   ElemPriority = 0;    /* For priority-based queues: add the */
                        /*   new queue element at the logical */
                        /*   "end" of the queue               */

   rc = DosWriteQueue(QueueHandle, Request, DataLength, DataBuffer,
                      ElemPriority);

   if (rc != 0)
     {
       printf("DosWriteQueue error: return code = %ld", rc);
       return;
     }
```

Appendix A. Data Types

APIRET

Unsigned integer in the range 0 through 4 294 967 295.

```
typedef unsigned long APIRET;
```

BOOL

Boolean.

Valid values are FALSE, which is 0, and TRUE, which is 1.

```
typedef unsigned long BOOL;
```

BOOL32

Boolean.

Valid values are FALSE, which is 0, and TRUE, which is 1.

```
typedef unsigned long BOOL32;
```

BYTE

Byte.

```
typedef unsigned char BYTE;
```

CHAR

Single-byte character.

```
#define CHAR char
```

COLOR

Color value.

```
typedef long COLOR;
```

COUNTRYCODE

Country code and code page.

```
typedef struct _COUNTRYCODE {
ULONG    ulcountry;
ULONG    ulcodepage;
 } COUNTRYCODE;
```

ulcountry (ULONG)

ulcodepage (ULONG)

COUNTRYINFO

Country information.

```
typedef struct _COUNTRYINFO {
ULONG     ulcountry;
ULONG     ulcodepage;
ULONG     ulfsDateFmt;
CHAR      chszCurrency;
CHAR      chszThousandsSeparator;
CHAR      chszDecimal;
CHAR      chszDateSeparator;
CHAR      chszTimeSeparator;
UCHAR     ucfsCurrencyFmt;
UCHAR     uccDecimalPlace;
UCHAR     ucfsTimeFmt;
USHORT    usabReserved;
CHAR      chszDataSeparator;
USHORT    usabReserved2;
 } COUNTRYINFO;
```

ulcountry (ULONG)

Country code.

ulcodepage (ULONG)

Code page.

ulfsDateFmt (ULONG)

Date format.

chszCurrency (CHAR)

Currency indicator.

chszThousandsSeparator (CHAR)

Thousands separator.

chszDecimal (CHAR)

Decimal separator.

chszDateSeparator (CHAR)

Date separator.

chszTimeSeparator (CHAR)

Time separator.

ucfsCurrencyFmt (UCHAR)

Bit fields for currency format.

uccDecimalPlace (UCHAR)

Currency decimal places.

ucfsTimeFmt (UCHAR)

Time format (AM/PM or 24 hr)

usabReserved (USHORT)

Reserved (0).

chszDataSeparator (CHAR)

Data list separator.

usabReserved2 (USHORT)

Reserved (0).

DATETIME DateTime data structure.

```
typedef struct _DATETIME {
UCHAR     uchours;
UCHAR     ucminutes;
UCHAR     ucSeconds;
UCHAR     uchundredths;
UCHAR     ucday;
UCHAR     ucmonth;
USHORT    usyear;
SHORT     stimezone;
UCHAR     ucweekday;
 } DATETIME;
```

uchours (UCHAR)

Current hour, using values 0 through 23.

ucminutes (UCHAR)

Current minute, using values 0 through 59.

ucSeconds (UCHAR)

Current second, using values 0 through 59.

uchundredths (UCHAR)

Current hundredths of a second, using values 0 through 99.

ucday (UCHAR)

Current day of the month, using values 1 through 31.

ucmonth (UCHAR)

Current month of the year, using values 1 through 12.

usyear (USHORT)

Current year.

stimezone (SHORT)

The difference in minutes between the current time zone and Greenwich Mean Time (GMT). This value is positive for time zones west of Greenwich, England, and negative for time zones east of Greenwich. A value of -1 indicates that the time zone is undefined.

ucweekday (UCHAR)

Current day of the week, using values 0 through 6. (Sunday is equal to 0.)

DosDebug Buffer Structure DosDebug buffer structure.

```
typedef struct _DosDebug Buffer Structure {
ULONG     ulPid;
ULONG     ulTid;
LONG      lCmd;
LONG      lValue;
ULONG     ulAddr;
ULONG     ulBuffer;
ULONG     ulLen;
ULONG     ulIndex;
ULONG     ulMTE;
ULONG     ulEAX;
ULONG     ulECX;
ULONG     ulEDX;
ULONG     ulEBX;
ULONG     ulESP;
ULONG     ulEBP;
ULONG     ulESI;
ULONG     ulEDI;
ULONG     ulEFlags;
ULONG     ulEIP;
ULONG     ulCSLim;
ULONG     ulCSBase;
UCHAR     ucCSAcc;
UCHAR     ucCSAtr;
USHORT    usCS;
ULONG     ulDSLim;
ULONG     ulDSBase;
UCHAR     ucDSAcc;
UCHAR     ucDSAtr;
USHORT    usDS;
ULONG     ulESLim;
ULONG     ulESBase;
UCHAR     ucESAcc;
UCHAR     ucESAtr;
USHORT    usES;
ULONG     ulFSLim;
ULONG     ulFSBase;
UCHAR     ucFSAcc;
UCHAR     ucFSAtr;
USHORT    usFS;
ULONG     ulGSLim;
ULONG     ulGSBase;
UCHAR     ucGSAcc;
UCHAR     ucGSAtr;
USHORT    usGS;
ULONG     ulSSLim;
ULONG     ulSSBase;
UCHAR     ucSSAcc;
UCHAR     ucSSAtr;
USHORT    usSS;
 } DosDebug Buffer Structure;
```

ulPid (ULONG)

Debuggee Process ID

ulTId (ULONG)

Debuggee Thread ID

lCmd (LONG)

Command or Notification

lValue (LONG)

Generic Data Value

ulAddr (ULONG)

Debuggee Address

ulBuffer (ULONG)

Debugger Buffer Address

ulLen (ULONG)

Length of Range

ulIndex (ULONG)

Generic Identifier Index

ulMTE (ULONG)

Module Table Entry Handle

ulEAX (ULONG)

Register Set

ulECX (ULONG)

ulEDX (ULONG)

ulEBX (ULONG)

ulESP (ULONG)

ulEBP (ULONG)

ulESI (ULONG)

ulEDI (ULONG)

ulEFlags (ULONG)

ulEIP (ULONG)

ulCSLim (ULONG)

Byte Granular Limits

ulCSBase (ULONG)

Byte Granular Base

ucCSAcc (UCHAR)

Access Bytes

ucCSAtr (UCHAR)
Attribute Bytes
usCS (USHORT)

ulDSLim (ULONG)

ulDSBase (ULONG)

ucDSAcc (UCHAR)

ucDSAtr (UCHAR)

usDS (USHORT)

ulESLim (ULONG)

ulESBase (ULONG)

ucESAcc (UCHAR)

ucESAtr (UCHAR)

usES (USHORT)

ulFSLim (ULONG)

ulFSBase (ULONG)

ucFSAcc (UCHAR)

ucFSAtr (UCHAR)

usFS (USHORT)

ulGSLim (ULONG)

ulGSBase (ULONG)

ucGSAcc (UCHAR)

ucGSAtr (UCHAR)

usGS (USHORT)

ulSSLim (ULONG)

ulSSBase (ULONG)

ucSSAcc (UCHAR)

ucSSAtr (UCHAR)

usSS (USHORT)

EAOP2 EAOP2 data structure.

```
typedef struct _EAOP2 {
PGEA2LIST    ppfpGEA2List;
PFEA2LIST    ppfpFEA2List;
ULONG        uloError;
 } EAOP2;
```

ppfpGEA2List (PGEA2LIST)

GEA set.

ppfpFEA2List (PFEA2LIST)

FEA set.

uloError (ULONG)

Offset of FEA error.

ERRORID Error identity.

```
typedef ULONG ERRORID;
```

EXCEPTIONREGISTRATIONRECORD These structures are linked together to form a chain of exception handlers that are dispatched upon receipt of an exception. Exception handlers should not be registered directly from a high level language such as 'C'. This is the responsibility of the language runtime routine.

```
typedef struct _EXCEPTIONREGISTRATIONRECORD {
STRUCT    _EXCEPTIONREGISTRATIONRECORD *prev_structure;
ULONG     ul(_cdecl *ExceptionHandler) (PEXCEPTIONREPORTRECORD,
                                        struct _EXCEPTIONREGISTRATIONRECORD *,
                                        PCONTEXTRECORD, PVOID);
 } EXCEPTIONREGISTRATIONRECORD;
typedef struct _EXCEPTIONREGISTRATIONRECORD EXCEPTIONREGISTRATIONRECORD;
typedef struct _EXCEPTIONREGISTRATIONRECORD *PEXCEPTIONREGISTRATIONRECORD;
```

_EXCEPTIONREGISTRATIONRECORD *prev_structure (STRUCT)

Nested ExceptionRegistrationRecord structure.

EXCEPTIONREPORTRECORD This structure contains machine-independent information about an exception or unwind. No system exception will ever have more parameters than the value of EXCEPTION_MAXIMUM_PARAMETERS. User exceptions are not bound to this limit.

```
typedef struct _EXCEPTIONREPORTRECORD {
ULONG     ulExceptionNum;
ULONG     ulfHandlerflags;
STRUCT    _EXCEPTIONREPORTRECORD ;
PVOID     pExceptionAddress;
ULONG     ulcParameters;
ULONG     ulExceptionInfo[EXCEPTION_MAXIMUM_PARAMETERS];
 } EXCEPTIONREPORTRECORD;
```

ulExceptionNum (ULONG)

Exception number.

ulfHandlerflags (ULONG)

Handler flags.

_EXCEPTIONREPORTRECORD (STRUCT)

Nested ExceptionReportRecord structure.

pExceptionAddress (PVOID)

Address of the exception.

ulcParameters (ULONG)

Size of exception specific information.

ulExceptionInfo[EXCEPTION_MAXIMUM_PARAMETERS] (ULONG)

Exception specific information.

FDATE

Date data structure for file-system functions.

```
typedef struct _FDATE {
USHORT    usday;
USHORT    usmonth;
USHORT    usyear;
 } FDATE;
```

usday (USHORT)

Binary day for directory entry.

usmonth (USHORT)

Binary month for directory entry.

usyear (USHORT)

Binary year for directory entry.

FEA2

32-bit FEA2 data structure.

```
typedef struct _FEA2 {
ULONG     uloNextEntryOffset;
BYTE      bfEA;
BYTE      bcbName;
USHORT    uscbValue;
CHAR      chszName[1];
 } FEA2;
```

uloNextEntryOffset (ULONG)

Offset to next entry.

bfEA (BYTE)

Flags.

bcbName (BYTE)

Name length not including NULL.

uscbValue (USHORT)

Value length.

chszName[1] (CHAR)

FEA2LIST

FEA2 data structure.

```
typedef struct _FEA2LIST {
ULONG    ulcbList;
FEA2     list[1];
 } FEA2LIST;
```

ulcbList (ULONG)

Total bytes of structure including full list.

list[1] (FEA2)

Variable length FEA2 structures.

FILEFINDBUF3

32-bit level 1 information (used without EAs).

```
typedef struct _FILEFINDBUF3 {
ULONG    uloNextEntryOffset;
FDATE    fdateCreation;
FTIME    ftimeCreation;
FDATE    fdateLastAccess;
FTIME    ftimeLastAccess;
FDATE    fdateLastWrite;
FTIME    ftimeLastWrite;
ULONG    ulcbFile;
ULONG    ulcbFileAlloc;
ULONG    ulattrFile;
UCHAR    uccchName;
CHAR     chachName[CCHMAXPATHCOMP];
 } FILEFINDBUF3;
```

uloNextEntryOffset (ULONG)

fdateCreation (FDATE)

ftimeCreation (FTIME)

fdateLastAccess (FDATE)

ftimeLastAccess (FTIME)

fdateLastWrite (FDATE)

ftimeLastWrite (FTIME)

ulcbFile (ULONG)

ulcbFileAlloc (ULONG)

ulattrFile (ULONG)

uccchName (UCHAR)

chachName[**CCHMAXPATHCOMP**] (CHAR)

FILEFINDBUF4

32-bit level 2 information (used with EAs).

```
typedef struct _FILEFINDBUF4 {
ULONG    uloNextEntryOffset;
FDATE    fdateCreation;
FTIME    ftimeCreation;
FDATE    fdateLastAccess;
FTIME    ftimeLastAccess;
FDATE    fdateLastWrite;
FTIME    ftimeLastWrite;
ULONG    ulcbFile;
ULONG    ulcbFileAlloc;
ULONG    ulattrFile;
ULONG    ulcbList;
UCHAR    uccchName;
CHAR     chachName[CCHMAXPATHCOMP];
 } FILEFINDBUF4;
```

uloNextEntryOffset (ULONG)

fdateCreation (FDATE)

ftimeCreation (FTIME)

fdateLastAccess (FDATE)

ftimeLastAccess (FTIME)

fdateLastWrite (FDATE)

ftimeLastWrite (FTIME)

ulcbFile (ULONG)

ulcbFileAlloc (ULONG)

ulattrFile (ULONG)

ulcbList (ULONG)

uccchName (UCHAR)

chachName[CCHMAXPATHCOMP] (CHAR)

FILELOCK

FILELOCK data structure.

```
typedef struct _FILELOCK {
LONG    l1Offset;
LONG    l1Range;
 } FILELOCK;
```

l1Offset (LONG)

Offset to the beginning of the lock range.

l1Range (LONG)

Length, in bytes, of the lock range.

FILESTATUS3

32-bit level 1 information.

```
typedef struct _FILESTATUS3 {
FDATE    fdateCreation;
FTIME    ftimeCreation;
FDATE    fdateLastAccess;
FTIME    ftimeLastAccess;
FDATE    fdateLastWrite;
FTIME    ftimeLastWrite;
ULONG    ulcbFile;
ULONG    ulcbFileAlloc;
ULONG    ulattrFile;
 } FILESTATUS3;
```

fdateCreation (FDATE)

Date of file creation.

ftimeCreation (FTIME)

Time of file creation.

fdateLastAccess (FDATE)

Date of last access.

ftimeLastAccess (FTIME)

Time of last access.

fdateLastWrite (FDATE)

Date of last write.

ftimeLastWrite (FTIME)

Time of last write.

ulcbFile (ULONG)

File size (end of data).

ulcbFileAlloc (ULONG)

File allocated size.

ulattrFile (ULONG)

Attributes of the file.

FILESTATUS4

32-bit level 2 information.

```
typedef struct _FILESTATUS4 {
FDATE    fdateCreation;
FTIME    ftimeCreation;
FDATE    fdateLastAccess;
FTIME    ftimeLastAccess;
FDATE    fdateLastWrite;
FTIME    ftimeLastWrite;
ULONG    ulcbFile;
ULONG    ulcbFileAlloc;
ULONG    ulattrFile;
ULONG    ulcbList;
 } FILESTATUS4;
```

fdateCreation (FDATE)

Date of file creation.

ftimeCreation (FTIME)

Time of file creation.

fdateLastAccess (FDATE)

Date of last access.

ftimeLastAccess (FTIME)

Time of last access.

fdateLastWrite (FDATE)

Date of last write.

ftimeLastWrite (FTIME)

Time of last write.

ulcbFile (ULONG)

File size (end of data).

ulcbFileAlloc (ULONG)

File allocated size.

ulattrFile (ULONG)

Attributes of the file.

ulcbList (ULONG)

Length of entire EA set.

FSQBUFFER2

Data structure for information about an attached file system (local or remote), or about a character device or pseudocharacter device attached to the file system.

```
typedef struct _FSQBUFFER2 {
USHORT    usiType;
USHORT    uscbName;
USHORT    uscbFSDName;
USHORT    uscbFSAData;
UCHAR     ucszName;
UCHAR     ucszFSDName;
UCHAR     ucrgFSAData;
 } FSQBUFFER2;
```

usiType (USHORT)

Type of item.

Value	Definition
1	(FSAT_CHARDEV) Resident character device
2	(FSAT_PSEUDODEV) Pseudocharacter device
3	(FSAT_LOCALDRV) Local drive
4	(FSAT_REMOTEDRV) Remote drive attached to the file-system driver.

uscbName (USHORT)

Length, in bytes, of the item name, not counting null.

uscbFSDName (USHORT)

Length, in bytes, of the file-system driver name, not counting null.

uscbFSAData (USHORT)

Length, in bytes, of the file-system driver Attach data returned by the file-system driver.

ucszName (UCHAR)

Item name. The name is an ASCIIZ string.

ucszFSDName (UCHAR)

Name of the file-system driver that the item is attached to. The name is an ASCIIZ string.

ucrgFSAData (UCHAR)

File-system driver Attach data returned by the file-system driver.

FTIME

Time data structure for file-system functions.

```
typedef struct _FTIME {
USHORT    ustwosecs;
USHORT    usminutes;
USHORT    ushours;
 } FTIME;
```

ustwosecs (USHORT)

A binary number of two-second increments.

usminutes (USHORT)

A binary number of minutes.

ushours (USHORT)

A binary number of hours.

GEA2

32-bit Level 3 File Information - Get Extended Attributes.

```
typedef struct _GEA2 {
ULONG     uloNextEntryOffset;
BYTE      bcbName;
CHAR      chszname[1];
 } GEA2;
```

uloNextEntryOffset (ULONG)

Offset to next entry.

bcbName (BYTE)

Name length not including NULL.

chszname[1] (CHAR)

Attribute name.

GEA2LIST Get Extended Attributes list.

```
typedef struct _GEA2LIST {
ULONG    ulcbList;
GEA2     list[1];
 } GEA2LIST;
```

ulcbList (ULONG)

Total bytes of structure including full list.

list[1] (GEA2)

Variable-length GEA2 structures.

HDC Device-context handle.

```
typedef LHANDLE HDC;
```

HDIR 32-bit value used as a directory handle.

```
typedef VOID *HDIR;
```

HEV 32-bit value used as an event handle.

```
typedef ULONG *HEV;
```

HFILE Resource handle.

```
typedef LHANDLE HFILE;
```

HMF Metafile handle.

```
typedef LHANDLE HMF;
```

HMODULE Module handle.

```
typedef LHANDLE HMODULE;
```

HMONITOR 32-bit value used as a system monitor handle.

```
typedef VOID *HMONITOR;
```

HMTX 32-bit value used as a mutex-semaphore handle.

```
typedef ULONG *HMTX;
```

HMUX 32-bit value used as a muxwait semaphore handle.

```
typedef ULONG *HMUX;
```

HPIPE 32-bit value used as a pipe handle.

```
typedef VOID *HPIPE;
```

HPS Presentation-space handle.

```
typedef LHANDLE HPS;
```

HSYSSEM 32-bit value used as a system semaphore handle.

```
typedef VOID *HSYSSEM;
```

HQUEUE 32-bit value used as a system queue handle.

```
typedef VOID *HQUEUE;
```

HRGN Region handle.

```
typedef LHANDLE HRGN;
```

HSEM Semaphore handle.

```
typedef VOID *HSEM;
```

HTIMER — 32-bit value used as a timer handle.

```
typedef VOID *HTIMER;
```

HVDD — 32-bit value used as a virtual device driver handle.

```
typedef ULONG *HVDD;
```

HWND — Window handle.

```
typedef LHANDLE HWND;
```

LONG — Signed integer in the range –2 147 483 648 through 2 147 483 647.

Note: Where this data type represents a graphic coordinate in world or model space, its value is restricted to –134 217 728 through 134 217 727.

A graphic coordinate in device or screen coordinates is restricted to –32 768 through 32 767.

The value of a graphic coordinate may be further restricted by any transforms currently in force, including the positioning of the origin of the window on the screen. In particular, coordinates in world or model space must not generate coordinate values after transformation (that is, in device or screen space) outside the range –32 768 through 32 767.

```
#define LONG long
```

NID — A 32-bit value to hold a name identifier.

```
typedef ULONG NID;
```

NPCH — 32-bit pointer to a value or array of values.

```
typedef CHAR *NPCH;
```

NPFN — 32-bit pointer to a function with *pascal* calling type.

```
typedef CHAR *NPFN;
```

NPSZ — 32-bit pointer to a null-terminated string.

```
typedef CHAR *NPSZ;
```

PAVAILDATA — Pointer to the 4-byte buffer in which the system returns the number of bytes that were available.

```
typedef AVAILDATA *PAVAILDATA;
```

PBOOL — Pointer to BOOL.

```
typedef BOOL *PBOOL;
```

PBOOL32 — Pointer to BOOL32.

```
typedef BOOL32 *PBOOL32;
```

PBYTE — Pointer to a data area.

```
typedef BYTE *PBYTE;
```

PCHAR — Pointer to CHAR.

```
typedef CHAR *PCHAR;
```

PCOLOR — Pointer to COLOR.

```
typedef COLOR *PCOLOR;
```

PCOUNTRYCODE — Pointer to COUNTRYCODE.

```
typedef CHAR *PCOUNTRYCODE;
```

PCOUNTRYINFO — Pointer to COUNTRYINFO.

```
typedef CHAR *PCOUNTRYINFO;
```

PDATETIME — Pointer to DATETIME.

```
typedef DATETIME *PDATETIME;
```

PEAOP2 Pointer to EAOP2.

```
typedef EAOP2 *PEAOP2;
```

PERRORID Pointer to ERRORID.

```
typedef ERRORID *PERRORID;
```

PEXCEPTIONREGISTRATIONRECORD Pointer to EXCEPTIONREGISTRATIONRECORD.

```
typedef EXCEPTIONREGISTRATIONRECORD *PEXCEPTIONREGISTRATIONRECORD;
```

PEXCEPTIONREPORTRECORD Pointer to EXCEPTIONREPORTRECORD.

```
typedef EXCEPTIONREPORTRECORD *PEXCEPTIONREPORTRECORD;
```

PFILEFINDBUF3 Pointer to FILEFINDBUF3.

```
typedef FILEFINDBUF3 *PFILEFINDBUF3;
```

PFILEFINDBUF4 Pointer to FILEFINDBUF4.

```
typedef FILEFINDBUF4 *PFILEFINDBUF4;
```

PFILELOCK Pointer to FILELOCK.

```
typedef FILELOCK *PFILELOCK;
```

PFN Pointer to procedure.

```
typedef int *PFN();
```

PFNSIGHANDLER 32-bit pointer to a function with *pascal* calling type.

```
typedef CHAR *PFNSIGHANDLER;
```

PFNTHREAD Address of the code to be executed when the thread begins execution.

```
typedef VOID *PFNTHREAD;
```

PFNEXITLIST Address of a routine to be executed.

```
typedef PVOID *PFNEXITLIST;
```

PFEA2LIST Pointer to FEA2LIST.

```
typedef CHAR *PFEA2LIST;
```

PFSQBUFFER2 Pointer to FSQBUFFER2.

```
typedef FSQBUFFER2 *PFSQBUFFER2;
```

PGEA2LIST Pointer to GEA2LIST.

```
typedef CHAR *PGEA2LIST;
```

PHDC Pointer to HDC.

```
typedef HDC *PHDC;
```

PHDIR Pointer to HDIR.

```
typedef HDIR *PHDIR;
```

PHEV Pointer to HEV.

```
typedef HEV *PHEV;
```

PHFILE Pointer to HFILE.

```
typedef HFILE *PHFILE;
```

PHMF Pointer to HMF.

```
typedef HMF *PHMF;
```

PHMODULE Pointer to HMODULE.

```
typedef HMODULE *PHMODULE;
```

PHMONITOR Pointer to HMONITOR.

```
typedef HMONITOR *PHMONITOR;
```

PHMTX Pointer to HMTX.

```
typedef HMTX *PHMTX;
```

PHMUX Pointer to HMUX.

```
typedef HMUX *PHMUX;
```

PHPIPE Pointer to HPIPE.

```
typedef HPIPE *PHPIPE;
```

PHPS Pointer to HPS.

```
typedef HPS *PHPS;
```

PHQUEUE Pointer to HQUEUE.

```
typedef HQUEUE *PHQUEUE;
```

PHRGN Pointer to HRGN.

```
typedef HRGN *PHRGN;
```

PHSEM Pointer to HSEM.

```
typedef HSEM *PHSEM;
```

PHTIMER Pointer to HTIMER.

```
typedef HTIMER *PHTIMER;
```

PHVDD Pointer to HVDD.

```
typedef HVDD *PHVDD;
```

PIB Process Information Block structure.

```
typedef struct _PIB {
ULONG    ulpib_ulpid;
ULONG    ulpib_ulppid;
ULONG    ulpib_hmte;
PCHAR    pppib_pchcmd;
PCHAR    pppib_pchenv;
ULONG    ulpib_flstatus;
ULONG    ulpib_ultype;
 } PIB;
```

ulpib_ulpid (ULONG)

Process identifier.

ulpib_ulppid (ULONG)

Parent process identifier.

ulpib_hmte (ULONG)

Module handle of executable program.

pppib_pchcmd (PCHAR)

Command line pointer.

pppib_pchenv (PCHAR)

Environment pointer.

ulpib_flstatus (ULONG)

Process status bits.

ulpib_ultype (ULONG)

Process type code.

PID Process identity.

```
typedef ULONG PID;
```

PIPESEMSTATE Data structure for the status of a named pipe that is attached to a semaphore.

```
typedef struct _PIPESEMSTATE {
BYTE      bfStatus;
BYTE      bfFlag;
USHORT    usKey;
USHORT    usAvail;
 } PIPESEMSTATE;
```

bfStatus (BYTE)

A coded value that indicates the status of the named pipe.

Value	Definition
0	(NPSS_EOI) End of information buffer. No more information records follow, and subsequent fields in this information record have no defined value.
1	(NPSS_RDATA) Read data is available.
2	(NPSS_WSPACE) Write space is available.
3	(NPSS_CLOSE) The pipe is closed.

bfFlag (BYTE)

A bit field that indicates additional information about the state of the named pipe.

Bit	Description
7–1	Reserved
0	(NPSS_WAIT) If set, a thread is waiting at the other end of the pipe.

usKey (USHORT)

The value specified for *KeyHandle* when DosSetNPipeSem was issued.

usAvail (USHORT)

If **fStatus** has a value of 1, this field contains the number of bytes of data that are available to read from the pipe. If **fStatus** has a value of 2, this field contains the number of bytes of write space that are available in the pipe.

PLONG Pointer to LONG.

```
typedef LONG *PLONG;
```

PPIB Pointer to PIB.

```
typedef PIB *PPIB;
```

PPID Pointer to PID.

```
typedef PID *PPID;
```

PPIPESEMSTATE Pointer to PIPESEMSTATE.

```
typedef PIPESEMSTATE *PPIPESEMSTATE;
```

PPPIB Pointer to PPIB.

```
typedef PPIB *PPPIB;
```

PPTIB Pointer to PTIB.

```
typedef PTIB *PPTIB;
```

PPVOID Pointer to PVOID.

```
typedef PVOID *PPVOID;
```

PREQUESTDATA Pointer to REQUESTDATA.

```
typedef REQUESTDATA *PREQUESTDATA;
```

PRESULTCODES Pointer to RESULTCODES.

```
typedef RESULTCODES *PRESULTCODES;
```

PSEMRECORD Pointer to SEMRECORD.

```
typedef SEMRECORD *PSEMRECORD;
```

PSTARTDATA Pointer to STARTDATA.

```
typedef STARTDATA *PSTARTDATA;
```

PSTATUSDATA Pointer to STATUSDATA.

```
typedef STATUSDATA *PSTATUSDATA;
```

PSZ Pointer to a null-terminated string.

```
typedef char *PSZ;
```

PTIB Pointer to TIB.

```
typedef TIB *PTIB;
```

PTIB2 Pointer to TIB2.

```
typedef TIB2 *PTIB2;
```

PTID Pointer to TID.

```
typedef TID *PTID;
```

PULONG Pointer to ULONG.

```
typedef ULONG *PULONG;
```

PVOID Pointer to a data type of undefined format.

```
typedef VOID *PVOID;
```

REQUESTDATA REQUESTDATA structure.

```
typedef struct _REQUESTDATA {
PID      idpid;
ULONG    ulData;
 } REQUESTDATA;
```

idpid (PID)

Process identifier of the process that placed the element into the queue.

ulData (ULONG)

User-defined value.

RESULTCODES RESULTCODES data structure.

```
typedef struct _RESULTCODES {
ULONG    ulcodeTerminate;
ULONG    ulcodeResult;
 } RESULTCODES;
```

ulcodeTerminate (ULONG)

Termination code or process identifier.

ulcodeResult (ULONG)

Exit code.

SEMRECORD Muxwait semaphore data structure.

```
typedef struct _SEMRECORD {
HSEM     hsemCur;
ULONG    ulUser;
 } SEMRECORD;
```

hsemCur (HSEM)

Handle of the semaphore.

ulUser (ULONG)

User-defined value.

SGID

32-bit value used as a session identifier.

```
typedef ULONG *SGID;
```

SHORT

Signed integer in the range -32 768 through 32 767.

```
#define SHORT short
```

STARTDATA

Start session data structure.

```
typedef struct _STARTDATA {
USHORT    uscb;
USHORT    usRelated;
USHORT    usFgBg;
USHORT    usTraceOpt;
PSZ       pszPgmTitle;
PSZ       pszPgmName;
PSZ       pszPgmInputs;
PSZ       pszTermQ;
PSZ       pszEnvironment;
USHORT    usInheritOpt;
USHORT    usSessionType;
PSZ       pszIconFile;
ULONG     ulPgmHandle;
USHORT    usPgmControl;
USHORT    usInitXPos;
USHORT    usInitYPos;
USHORT    usInitXSize;
USHORT    usInitYSize;
USHORT    usReserved;
PSZ       pszObjectBuffer;
ULONG     ulObjectBuffLen;
 } STARTDATA;
```

uscb (USHORT)

The length of the data structure in bytes.

usRelated (USHORT)

A 0 equals an independent session and a 1 equals a child session.

usFgBg (USHORT)

A 0 equals a start in foreground and a 1 equals a start in background.

usTraceOpt (USHORT)

A 0 equals no trace and a 1 equals a trace.

pszPgmTitle (PSZ)

The address of the program title.

pszPgmName (PSZ)

The address of the program name.

pszPgmInputs (PSZ)

Input arguments.

pszTermQ (PSZ)

The address of the program queue name.

pszEnvironment (PSZ)

The address of the environment string.

usInheritOpt (USHORT)

The inherit option (shell of the program).

usSessionType (USHORT)

The session type.

pszIconFile (PSZ)

The address of the icon definition.

ulPgmHandle (ULONG)

The program handle.

usPgmControl (USHORT)

The initial state of the windowed application.

usInitXPos (USHORT)

The x coordinate of the initial session window.

usInitYPos (USHORT)

The y coordinate of the initial session window.

usInitXSize (USHORT)

The initial size of x.

usInitYSize (USHORT)

The initial size of y.

usReserved (USHORT)

Reserved area which must be 0.

pszObjectBuffer (PSZ)

The address of the name of the object that contributed to the failure of DosExecPgm.

ulObjectBuffLen (ULONG)

The length of the object buffer in bytes.

STATUSDATA

Status data structure.

```
typedef struct _STATUSDATA {
USHORT    usLength;
USHORT    usSelectInd;
USHORT    usBondInd;
 } STATUSDATA;
```

usLength (USHORT)

The length of the data structure in bytes, including **Length** itself.

usSelectInd (USHORT)

An indicator that specifies whether the target session should be flagged as selectable or non-selectable.

usBondInd (USHORT)

BondInd specifies which session to bring to the foreground the next time the parent session is selected.

STRUCT

Dummy data structure to be able to nest structures.

```
typedef struct _STRUCT {
```

TIB

Thread Information Block structure.

```
typedef struct _TIB {
PVOID    ptib_pexchain;
PVOID    ptib_pstack;
PVOID    ptib_pstacklimit;
PTIB2    pptib_ptib2;
ULONG    ultib_version;
PVOID    ptib_arbpointer;
 } TIB;
```

ptib_pexchain (PVOID)

Head of exception handler chain.

ptib_pstack (PVOID)

Pointer to the base of the stack.

ptib_pstacklimit (PVOID)

Pointer to the end of the stack.

pptib_ptib2 (PTIB2)

Pointer to a system-specific thread information block.

ultib_version (ULONG)

Version number for this Thread Information Block.

ptib_arbpointer (PVOID)

Thread ordinal number.

TIB2

System-specific Thread Information Block structure.

```
typedef struct _TIB2 {
ULONG     ultib2_ultid;
ULONG     ultib2_ulpri;
ULONG     ultib2_version;
USHORT    ustib2_usMCCount;
USHORT    ustib2_fMCForceFlag;
 } TIB2;
```

ultib2_ultid (ULONG)

Current thread identifier.

ultib2_ulpri (ULONG)

Current thread priority.

ultib2_version (ULONG)

Version number for this system-specific Thread Information Block.

ustib2_usMCCount (USHORT)

Must-complete count.

ustib2_fMCForceFlag (USHORT)

Must-complete force flag.

TID

Thread identity.

```
typedef ULONG TID;
```

UCHAR

Unsigned integer in the range 0 through 255.

```
typedef unsigned char UCHAR;
```

ULONG

Unsigned integer in the range 0 through 4 294 967 295.

```
typedef unsigned long ULONG;
```

USHORT

Unsigned integer in the range 0 through 65 535.

```
typedef unsigned short USHORT;
```

Appendix B. Errors

The following shows the numerical value of an error, its symbolic name, and a brief description of the error.

0 NO_ERROR
No error occurred.

1 ERROR_INVALID_FUNCTION
Invalid function number.

2 ERROR_FILE_NOT_FOUND
File not found.

3 ERROR_PATH_NOT_FOUND
Path not found.

4 ERROR_TOO_MANY_OPEN_FILES
Too many open files (no handles left).

5 ERROR_ACCESS_DENIED
Access denied.

6 ERROR_INVALID_HANDLE
Invalid handle.

7 ERROR_ARENA_TRASHED
Memory control blocks destroyed.

8 ERROR_NOT_ENOUGH_MEMORY
Insufficient memory.

9 ERROR_INVALID_BLOCK
Invalid memory-block address.

10 ERROR_BAD_ENVIRONMENT
Invalid environment.

11 ERROR_BAD_FORMAT
Invalid format.

12 ERROR_INVALID_ACCESS
Invalid access code.

13 ERROR_INVALID_DATA
Invalid data.

14 Reserved.

15 ERROR_INVALID_DRIVE
Invalid drive specified.

16 ERROR_CURRENT_DIRECTORY
Attempting to remove current directory.

17 ERROR_NOT_SAME_DEVICE
Not same device.

18 ERROR_NO_MORE_FILES
No more files.

19 ERROR_WRITE_PROTECT
Attempt to write on write-protected diskette.

20 ERROR_BAD_UNIT
Unknown unit.

21 ERROR_NOT_READY
Drive not ready.

22 ERROR_BAD_COMMAND
Unknown command.

23 ERROR_CRC
Data error - cyclic redundancy check.

24 ERROR_BAD_LENGTH
Invalid request structure length.

25 ERROR_SEEK
Seek error.

26 ERROR_NOT_DOS_DISK
Unknown media type.

27 ERROR_SECTOR_NOT_FOUND
Sector not found.

28 ERROR_OUT_OF_PAPER
Printer is out of paper.

29 ERROR_WRITE FAULT
Write fault.

30 ERROR_READ_FAULT
Read fault.

31 ERROR_GEN_FAILURE
General failure.

32 ERROR_SHARING_VIOLATION
Sharing violation.

33 ERROR_LOCK_VIOLATION
Lock violation.

34 ERROR_WRONG_DISK
Invalid disk change.

35 ERROR_FCB_UNAVAILABLE
FCB unavailable.

36 ERROR_SHARING_BUFFER_EXCEEDED
Sharing buffer overflow.

37 ERROR_CODE_PAGE_MISMATCHED
Code page does not match.

38 ERROR_HANDLE_EOF
End of file reached.

39 ERROR_HANDLE_DISK_FULL
Disk is full.

40 – 49 Reserved.

50 ERROR_NOT_SUPPORTED
Network request not supported.

51 ERROR_REM_NOT_LIST
Remote network node is not online.

52 ERROR_DUP_NAME
Duplicate file name in network.

53 ERROR_BAD_NETPATH
Network path not found.

54 ERROR_NETWORK_BUSY
Network is busy.

55 ERROR_DEV_NOT_EXIST
Device is not installed in network.

56 ERROR_TOO_MANY_CMDS
Network command limit reached.

57 ERROR_ADAP_HDW_ERR
Network adapter hardware error.

58 ERROR_BAD_NET_RESP
Incorrect response in network.

59 ERROR_UNEXP_NET_ERR
Unexpected error in network.

60 ERROR_BAD_REM_ADAP
Remote network adapter error.

61 ERROR_PRINTQ_FULL
Network printer queue is full.

62 ERROR_NO_SPOOL_SPACE
No space in print spool file.

63 ERROR_PRINT_CANCELLED
Print spool file deleted.

64 ERROR_NETNAME_DELETED
Network name deleted.

65 ERROR_NETWORK_ACCESS_DENIED
Access to network denied.

66 ERROR_BAD_DEV_TYPE
Device type invalid for network.

67 ERROR_BAD_NET_NAME
Network name not found.

68 ERROR_TOO_MANY_NAMES
Network name limit exceeded.

69 ERROR_TOO_MANY_SESS
Network session limit exceeded.

70 ERROR_SHARING_PAUSED
Temporary pause in network.

71 ERROR_REQ_NOT_ACCEP
Network request denied.

72 ERROR_REDIR_PAUSED
Pause in network print disk redirection.

73 ERROR_SBCS_ATT_WRITE_PROT
Attempted write on protected disk.

74 ERROR_SBCS_GENERAL_FAILURE
General failure, single-byte character set.

75 – 79 Reserved.

80 ERROR_FILE_EXISTS
File exists.

81 ERROR_DUP_FCB
Reserved.

82 ERROR_CANNOT_MAKE
Cannot make directory entry.

83 ERROR_FAIL_I24
Failure on INT 24.

84 ERROR_OUT_OF_STRUCTURES
Too many redirections.

85 ERROR_ALREADY_ASSIGNED
Duplicate redirection.

86 ERROR_INVALID_PASSWORD
Invalid password.

87 ERROR_INVALID_PARAMETER
Invalid parameter.

88 ERROR_NET_WRITE_FAULT
Network device fault.

89 ERROR_NO_PROC_SLOTS
No process slots available.

90 ERROR_NOT_FROZEN
System error.

91 ERR_TSTOVFL
Timer service table overflow.

92 ERR_TSTDUP
Timer service table duplicate.

93 ERROR_NO_ITEMS
No items to work on.

95 ERROR_INTERRUPT
Interrupted system call.

99 ERROR_DEVICE_IN_USE
Device in use.

100 ERROR_TOO_MANY_SEMAPHORES
User/system open semaphore limit reached.

101 ERROR_EXCL_SEM_ALREADY_OWNED
Exclusive semaphore already owned.

102 ERROR_SEM_IS_SET
DosCloseSem found semaphore set.

103 ERROR_TOO_MANY_SEM_REQUESTS
Too many exclusive semaphore requests.

104 ERROR_INVALID_AT_INTERRUPT_TIME
Operation invalid at interrupt time.

105 ERROR_SEM_OWNER_DIED
Previous semaphore owner terminated without freeing semaphore.

106 ERROR_SEM_USER_LIMIT
Semaphore limit exceeded.

107 ERROR_DISK_CHANGE
Insert drive B disk into drive A.

108 ERROR_DRIVE_LOCKED
Drive locked by another process.

109 ERROR_BROKEN_PIPE
Write on pipe with no reader.

110 ERROR_OPEN_FAILED
Open/create failed due to explicit fail command.

111 ERROR_BUFFER_OVERFLOW
Buffer passed to system call too small to hold return data.

112 ERROR_DISK_FULL
Not enough space on the disk.

113 ERROR_NO_MORE_SEARCH_HANDLES
Cannot allocate another search structure and handle.

114 ERROR_INVALID_TARGET_HANDLE
Target handle in DosDupHandle invalid.

115 ERROR_PROTECTION_VIOLATION
Invalid user virtual address.

116 ERROR_VIOKBD_REQUEST
Error on display write or keyboard read.

117 ERROR_INVALID_CATEGORY
Category for DevIOCtl not defined.

118 ERROR_INVALID_VERIFY_SWITCH
Invalid value passed for verify flag.

119 ERROR_BAD_DRIVER_LEVEL
Level four driver not found.

120 ERROR_CALL_NOT_IMPLEMENTED
Invalid function called.

121 ERROR_SEM_TIMEOUT
Time-out occurred from semaphore API function.

122 ERROR_INSUFFICIENT_BUFFER
Data buffer too small.

123 ERROR_INVALID_NAME
Illegal character or invalid file-system name.

124 ERROR_INVALID_LEVEL
Non-implemented level for information retrieval or setting.

125 ERROR_NO_VOLUME_LABEL
No volume label found with DosQueryFSInfo function.

126 ERROR_MOD_NOT_FOUND
Module handle not found with getprocaddr, getmodhandle.

127 ERROR_PROC_NOT_FOUND
Procedure address not found with getprocaddr.

128 ERROR_WAIT_NO_CHILDREN
DosWaitChild finds no children.

129 ERROR_CHILD_NOT_COMPLETE
DosWaitChild children not terminated.

130 ERROR_DIRECT_ACCESS_HANDLE
Handle operation invalid for direct disk-access handles.

131 ERROR_NEGATIVE_SEEK
Attempting seek to negative offset.

132 ERROR_SEEK_ON_DEVICE
Application trying to seek on device or pipe.

133 ERROR_IS_JOIN_TARGET
Drive has previously joined drives.

134 ERROR_IS_JOINED
Drive is already joined.

135 ERROR_IS_SUBSTED
Drive is already substituted.

136 ERROR_NOT_JOINED
Cannot delete drive that is not joined.

137 ERROR_NOT_SUBSTED
Cannot delete drive that is not substituted.

138 ERROR_JOIN_TO_JOIN
Cannot join to a joined drive.

139 ERROR_SUBST_TO_SUBST
Cannot substitute to a substituted drive.

140 ERROR_JOIN_TO_SUBST
Cannot join to a substituted drive.

141 ERROR_SUBST_TO_JOIN
Cannot substitute to a joined drive.

142 ERROR_BUSY_DRIVE
Specified drive is busy.

143 ERROR_SAME_DRIVE
Cannot join or substitute a drive to a directory on the same drive.

144 ERROR_DIR_NOT_ROOT
Directory must be a subdirectory of the root.

145 ERROR_DIR_NOT_EMPTY
Directory must be empty to use join command.

146 ERROR_IS_SUBST_PATH
Path specified is being used in a substitute.

147 ERROR_IS_JOIN_PATH
Path specified is being used in a join.

148 ERROR_PATH_BUSY
Path specified is being used by another process.

149 ERROR_IS_SUBST_TARGET
Cannot join or substitute a drive that has a directory that is the target of a previous substitute.

150 ERROR_SYSTEM_TRACE
System trace error.

151 ERROR_INVALID_EVENT_COUNT
DosWaitMuxWaitSem errors.

152 ERROR_TOO_MANY_MUXWAITERS
System limit of 100 entries reached.

153 ERROR_INVALID_LIST_FORMAT
Invalid list format.

154 ERROR_LABEL_TOO_LONG
Volume label too big.

155 ERROR_TOO_MANY_TCBS
Cannot create another TCB.

156 ERROR_SIGNAL_REFUSED
Signal refused.

157 ERROR_DISCARDED
Segment is discarded.

158 ERROR_NOT_LOCKED
Segment is not locked.

159 ERROR_BAD_THREADID_ADDR
Invalid thread-identity address.

160 ERROR_BAD_ARGUMENTS
Invalid environment pointer.

161 ERROR_BAD_PATHNAME
Invalid path name passed to exec.

162 ERROR_SIGNAL_PENDING
Signal already pending.

163 ERROR_UNCERTAIN_MEDIA
Error with INT 24 mapping.

164 ERROR_MAX_THRDS_REACHED
No more process slots.

165 ERROR_MONITORS_NOT_SUPPORTED
Error with INT 24 mapping.

166 ERROR_UNC_DRIVER_NOT_INSTALLED
Default redirection return code.

167 ERROR_LOCK_FAILED
Locking failed.

168 ERROR_SWAPIO_FAILED
Swap I/O failed.

169 ERROR_SWAPIN_FAILED
Swap in failed.

170 ERROR_BUSY
Segment is busy.

171 – 172 Reserved.

173 ERROR_CANCEL_VIOLATION
A lock request is not outstanding for the specified file range, or the range length is zero.

174 ERROR_ATOMIC_LOCK_NOT_SUPPORTED
The file-system driver (FSD) does not support atomic lock operations. Versions of OS/2 prior to version 2.00 do not support atomic lock operations.

175 ERROR_READ_LOCKS_NOT_SUPPORTED
The file system driver (FSD) does not support shared read locks.

176 – 179 Reserved.

180 ERROR_INVALID_SEGMENT_NUMBER
Invalid segment number.

181 ERROR_INVALID_CALLGATE
Invalid call gate.

182 ERROR_INVALID_ORDINAL
Invalid ordinal.

183 ERROR_ALREADY_EXISTS
Shared segment already exists.

184 ERROR_NO_CHILD_PROCESS
No child process to wait for.

185 ERROR_CHILD_ALIVE_NOWAIT
NoWait specified and child alive.

186 ERROR_INVALID_FLAG_NUMBER
Invalid flag number.

187 ERROR_SEM_NOT_FOUND
Semaphore does not exist.

188 ERROR_INVALID_STARTING_CODESEG
Invalid starting code segment, incorrect END (label) directive.

189 ERROR_INVALID_STACKSEG
Invalid stack segment.

190 ERROR_INVALID_MODULETYPE
Invalid module type – dynamic-link library file cannot be used as an application. Application cannot be used as a dynamic-link library.

191 ERROR_INVALID_EXE_SIGNATURE
Invalid EXE signature – file is a DOS mode program or an improper program.

192 ERROR_EXE_MARKED_INVALID
EXE marked invalid – link detected errors when the application was created.

193 ERROR_BAD_EXE_FORMAT
Invalid EXE format – file is a DOS mode program or an improper program.

194 ERROR_ITERATED_DATA_EXCEEDS_64K
Iterated data exceeds 64KB – there is more than 64KB of data in one of the segments of the file.

195 ERROR_INVALID_MINALLOCSIZE
Invalid minimum allocation size – the size is specified to be less than the size of the segment data in the file.

196 ERROR_DYNLINK_FROM_INVALID_RING
Dynamic link from invalid privilege level – privilege level 2 routine cannot link to dynamic-link libraries.

197 ERROR_IOPL_NOT_ENABLED
IOPL not enabled – IOPL set to NO in CONFIG.SYS.

198 ERROR_INVALID_SEGDPL
Invalid segment descriptor privilege level – can only have privilege levels of 2 and 3.

199 ERROR_AUTODATASEG_EXCEEDS_64k
Automatic data segment exceeds 64KB.

200 ERROR_RING2SEG_MUST_BE_MOVABLE
Privilege level 2 segment must be movable.

201 ERROR_RELOC_CHAIN_XEEDS_SEGLIM
Relocation chain exceeds segment limit.

202 ERROR_INFLOOP_IN_RELOC_CHAIN
Infinite loop in relocation chain segment.

203 ERROR_ENVVAR_NOT_FOUND
Environment variable not found.

204 ERROR_NOT_CURRENT_CTRY
Not current country.

205 ERROR_NO_SIGNAL_SENT
No signal sent – no process in the command subtree has a signal handler.

206 ERROR_FILENAME_EXCED_RANGE
File name or extension is greater than 8.3 characters.

207 ERROR_RING2_STACK_IN_USE
Privilege level 2 stack is in use.

208 ERROR_META_EXPANSION_TOO_LONG
Meta (global) expansion is too long.

209 ERROR_INVALID_SIGNAL_NUMBER
Invalid signal number.

210 ERROR_THREAD_1_INACTIVE
Inactive thread.

211 ERROR_INFO_NOT_AVAIL
File system information is not available for this file.

212 ERROR_LOCKED
Locked error.

213 ERROR_BAD_DYNALINK
Attempted to execute a non-family API in DOS mode.

214 ERROR_TOO_MANY_MODULES
Too many modules.

215 ERROR_NESTING_NOT_ALLOWED
Nesting is not allowed.

217 ERROR_ZOMBIE_PROCESS
Zombie process.

218 ERROR_STACK_IN_HIGH_MEMORY
Stack is in high memory.

219 ERROR_INVALID_EXITROUTINE_RING
Invalid exit routine ring.

220 ERROR_GETBUF_FAILED
Get buffer failed.

221 ERROR_FLUSHBUF_FAILED
Flush buffer failed.

222 ERROR_TRANSFER_TOO_LONG
Transfer is too long.

224 ERROR_SMG_NO_TARGET_WINDOW
The application window was created without the FCF_TASKLIST style, or the application window not yet been created or has already been destroyed.

228 ERROR_NO_CHILDREN
No child process.

229 ERROR_INVALID_SCREEN_GROUP
Invalid session.

230 ERROR_BAD_PIPE
Non-existent pipe or invalid operation.

231 ERROR_PIPE_BUSY
Pipe is busy.

232 ERROR_NO_DATA
No data available on non-blocking read.

233 ERROR_PIPE_NOT_CONNECTED
Pipe was disconnected by server.

234 ERROR_MORE_DATA
More data is available.

240 ERROR_VC_DISCONNECTED
Session was dropped due to errors.

250 ERROR_CIRCULARITY_REQUESTED
Renaming a directory that would cause a circularity problem.

251 ERROR_DIRECTORY_IN_CDS
Renaming a directory that is in use.

252 ERROR_INVALID_FSD_NAME
Trying to access nonexistent FSD.

253 ERROR_INVALID_PATH
Invalid pseudo device.

254 ERROR_INVALID_EA_NAME
Invalid character in name, or invalid cbName.

255 ERROR_EA_LIST_INCONSISTENT
List does not match its size, or there are invalid EAs in the list.

256 ERROR_EA_LIST_TOO_LONG
FEAList is longer than 64K–1 bytes.

257 ERROR_NO_META_MATCH
String does not match expression.

259 ERROR_NO_MORE_ITEMS
DosQueryFSAttach ordinal query.

260 ERROR_SEARCH_STRUC_REUSED
DOS mode findfirst/next search structure reused.

261 ERROR_CHAR_NOT_FOUND
Character not found.

262 ERROR_TOO_MUCH_STACK
Stack request exceeds system limit.

263 ERROR_INVALID_ATTR
Invalid attribute.

264 ERROR_INVALID_STARTING_RING
Invalid starting ring.

265 ERROR_INVALID_DLL_INIT_RING
Invalid DLL INIT ring.

266 ERROR_CANNOT_COPY
Cannot copy.

267 ERROR_DIRECTORY
Used by DOSCOPY in doscall1.

268 ERROR_OPLOCKED_FILE
Oplocked file.

269 ERROR_OPLOCK_THREAD_EXISTS
Oplock thread exists.

270 ERROR_VOLUME_CHANGED
Volume changed.

271 – 273 Reserved.

274 ERROR_ALREADY_SHUTDOWN
System is already shut down.

275 ERROR_EAS_DIDNT_FIT
Buffer is not big enough to hold the EAs.

276 ERROR_EA_FILE_CORRUPT
EA file has been damaged.

277 ERROR_EA_TABLE_FULL
EA table is full.

278 ERROR_INVALID_EA_HANDLE
EA handle is invalid.

279 ERROR_NO_CLUSTER
No cluster.

280 ERROR_CREATE_EA_FILE
Cannot create the EA file.

281 ERROR_CANNOT_OPEN_EA_FILE
Cannot open the EA file.

282 ERROR_EAS_NOT_SUPPORTED
Destination file system does not support EAs.

283 ERROR_NEED_EAS_FOUND
Destination file system does not support EAs, and the source file's EAs contain a need EA.

284 ERROR_DUPLICATE_HANDLE
The handle already exists.

285 ERROR_DUPLICATE_NAME
The name already exists.

286 ERROR_EMPTY_MUXWAIT
The list of semaphores in a muxwait semaphore is empty.

287 ERROR_MUTEX_OWNED
The calling thread owns one or more of the mutex semaphores in the list.

288 ERROR_NOT_OWNER
Caller does not own the semaphore.

289 ERROR_PARAM_TOO_SMALL
Parameter is not large enough to contain all of the semaphore records in the muxwait semaphore.

290 ERROR_TOO_MANY_HANDLES
Limit reached for number of handles.

291 ERROR_TOO_MANY_OPENS
There are too many files or semaphores open.

292 ERROR_WRONG_TYPE
Attempted to create wrong type of semaphore.

293 ERROR_UNUSED_CODE
Code is not used.

294 ERROR_THREAD_NOT_TERMINATED
Thread has not terminated.

295 ERROR_INIT_ROUTINE_FAILED
Initialization routine failed.

296 ERROR_MODULE_IN_USE
Module is in use.

297 ERROR_NOT_ENOUGH_WATCHPOINTS
There are not enough watchpoints.

298 ERROR_TOO_MANY_POSTS
Post count limit was reached for an event semaphore.

299 ERROR_ALREADY_POSTED
Event semaphore is already posted.

300 ERROR_ALREADY_RESET
Event semaphore is already reset.

301 ERROR_SEM_BUSY
Semaphore is busy.

302 Reserved

303 ERROR_INVALID_PROCID
Invalid process identity.

304 ERROR_INVALID_PDELTA
Invalid priority delta.

305 ERROR_NOT_DESCENDANT
Not descendant.

306 ERROR_NOT_SESSION_MANAGER
Requestor not session manager.

307 ERROR_INVALID_PCLASS
Invalid P class.

308 ERROR_INVALID_SCOPE
Invalid scope.

309 ERROR_INVALID_THREADID
Invalid thread identity.

310 ERROR_DOSSUB_SHRINK
Cannot shrink segment – DosSubSetMem.

311 ERROR_DOSSUB_NOMEM
No memory to satisfy request – DosSubAllocMem.

312 ERROR_DOSSUB_OVERLAP
Overlap of the specified block with a block of allocated memory – DosSubFreeMem.

313 ERROR_DOSSUB_BADSIZE
Invalid size parameter – DosSubAllocMem or DosSubFreeMem.

314 ERROR_DOSSUB_BADFLAG
Invalid flag parameter – DosSubSetMem.

315 ERROR_DOSSUB_BADSELECTOR
Invalid segment selector.

316 ERROR_MR_MSG_TOO_LONG
Message too long for buffer.

317 ERROR_MR_MID_NOT_FOUND
Message identity number not found.

318 ERROR_MR_UN_ACC_MSGF
Unable to access message file.

319 ERROR_MR_INV_MSGF_FORMAT
Invalid message file format.

320 ERROR_MR_INV_IVCOUNT
Invalid insertion variable count.

321 ERROR_MR_UN_PERFORM
Unable to perform function.

322 ERROR_TS_WAKEUP
Unable to wake up.

323 ERROR_TS_SEMHANDLE
Invalid system semaphore.

324 ERROR_TS_NOTIMER
No timers available.

326 ERROR_TS_HANDLE
Invalid timer handle.

327 ERROR_TS_DATETIME
Date or time invalid.

328 ERROR_SYS_INTERNAL
Internal system error.

329 ERROR_QUE_CURRENT_NAME
Current queue name does not exist.

330 ERROR_QUE_PROC_NOT_OWNED
Current process does not own queue.

331 ERROR_QUE_PROC_OWNED
Current process owns queue.

332 ERROR_QUE_DUPLICATE
Duplicate queue name.

333 ERROR_QUE_ELEMENT_NOT_EXIST
Queue element does not exist.

334 ERROR_QUE_NO_MEMORY
Inadequate queue memory.

335 ERROR_QUE_INVALID_NAME
Invalid queue name.

336 ERROR_QUE_INVALID_PRIORITY
Invalid queue priority parameter.

337 ERROR_QUE_INVALID_HANDLE
Invalid queue handle.

338 ERROR_QUE_LINK_NOT_FOUND
Queue link not found.

339 ERROR_QUE_MEMORY_ERROR
Queue memory error.

340 ERROR_QUE_PREV_AT_END
Previous queue element was at end of queue.

341 ERROR_QUE_PROC_NO_ACCESS
Process does not have access to queues.

342 ERROR_QUE_EMPTY
Queue is empty.

343 ERROR_QUE_NAME_NOT_EXIST
Queue name does not exist.

344 ERROR_QUE_NOT_INITIALIZED
Queues not initialized.

345 ERROR_QUE_UNABLE_TO_ACCESS
Unable to access queues.

346 ERROR_QUE_UNABLE_TO_ADD
Unable to add new queue.

347 ERROR_QUE_UNABLE_TO_INIT
Unable to initialize queues.

349 ERROR_VIO_INVALID_MASK
Invalid function replaced.

350 ERROR_VIO_PTR
Invalid pointer to parameter.

351 ERROR_VIO_APTR
Invalid pointer to attribute.

352 ERROR_VIO_RPTR
Invalid pointer to row.

353 ERROR_VIO_CPTR
Invalid pointer to column.

354 ERROR_VIO_LPTR
Invalid pointer to length.

355 ERROR_VIO_MODE
Unsupported screen mode.

356 ERROR_VIO_WIDTH
Invalid cursor width value.

357 ERROR_VIO_ATTR
Invalid cursor attribute value.

358 ERROR_VIO_ROW
Invalid row value.

359 ERROR_VIO_COL
Invalid column value.

360 ERROR_VIO_TOPROW
Invalid TopRow value.

361 ERROR_VIO_BOTROW
Invalid BotRow value.

362 ERROR_VIO_RIGHTCOL
Invalid right column value.

363 ERROR_VIO_LEFTCO
Invalid left column value.

364 ERROR_SCS_CALL
Call issued by other than session manager.

365 ERROR_SCS_VALUE
Value is not for save or restore.

366 ERROR_VIO_WAIT_FLAG
Invalid wait flag setting.

367 ERROR_VIO_UNLOCK
Screen not previously locked.

368 ERROR_SGS_NOT_SESSION_MGR
Caller not session manager.

369 ERROR_SMG_INVALID_SGID
Invalid session identity.

369 ERROR_SMG_INVALID_SESSION_ID
Invalid session ID.

370 ERROR_SMG_NOSG
No sessions available.

370 ERROR_SMG_NO_SESSIONS
No sessions available.

371 ERROR_SMG_GRP_NOT_FOUND
Session not found.

371 ERROR_SMG_SESSION_NOT_FOUND
Session not found.

372 ERROR_SMG_SET_TITLE
Title sent by shell or parent cannot be changed.

373 ERROR_KBD_PARAMETER
Invalid parameter to keyboard.

374 ERROR_KBD_NO_DEVICE
No device.

375 ERROR_KBD_INVALID_IOWAIT
Invalid I/O wait specified.

376 ERROR_KBD_INVALID_LENGTH
Invalid length for keyboard.

377 ERROR_KBD_INVALID_ECHO_MASK
Invalid echo mode mask.

378 ERROR_KBD_INVALID_INPUT_MASK
Invalid input mode mask.

379 ERROR_MON_INVALID_PARMS
Invalid parameters to DosMon.

380 ERROR_MON_INVALID_DEVNAME
Invalid device name string.

381 ERROR_MON_INVALID_HANDLE
Invalid device handle.

382 ERROR_MON_BUFFER_TOO_SMALL
Buffer too small.

383 ERROR_MON_BUFFER_EMPTY
Buffer is empty.

384 ERROR_MON_DATA_TOO_LARGE
Data record is too large.

385 ERROR_MOUSE_NO_DEVICE
Mouse device closed (invalid device handle).

386 ERROR_MOUSE_INV_HANDLE
Mouse device closed (invalid device handle).

387 ERROR_MOUSE_INV_PARMS
Parameters invalid for display mode.

388 ERROR_MOUSE_CANT_RESET
Function assigned and cannot be reset.

389 ERROR_MOUSE_DISPLAY_PARMS
Parameters invalid for display mode.

390 ERROR_MOUSE_INV_MODULE
Module not valid.

391 ERROR_MOUSE_INV_ENTRY_PT
Entry point not valid.

392 ERROR_MOUSE_INV_MASK
Function mask invalid.

393 NO_ERROR_MOUSE_NO_DATA
No valid data.

394 NO_ERROR_MOUSE_PTR_DRAWN
Pointer drawn.

395 ERROR_INVALID_FREQUENCY
Invalid frequency for beep.

396 ERROR_NLS_NO_COUNTRY_FILE
Cannot find COUNTRY.SYS file.

397 ERROR_NLS_OPEN_FAILED
Cannot open COUNTRY.SYS file.

398 ERROR_NLS_NO_CTRY_CODE
Country code not found.

398 ERROR_NO_COUNTRY_OR_CODEPAGE
Country code not found.

399 ERROR_NLS_TABLE_TRUNCATED
Table returned information truncated, buffer is too small.

400 ERROR_NLS_BAD_TYPE
Selected type does not exist.

401 ERROR_NLS_TYPE_NOT_FOUND
Selected type is not in file.

402 ERROR_VIO_SMG_ONLY
Valid from session manager only.

403 ERROR_VIO_INVALID_ASCIIZ
Invalid ASCIIZ length.

404 ERROR_VIO_DEREGISTER
VioDeRegister not allowed.

405 ERROR_VIO_NO_POPUP
Pop-up window not allocated.

406 ERROR_VIO_EXISTING_POPUP
Pop-up window on screen (NoWait).

407 ERROR_KBD_SMG_ONLY
Valid from session manager only.

408 ERROR_KBD_INVALID_ASCIIZ
Invalid ASCIIZ length.

409 ERROR_KBD_INVALID_MASK
Invalid replacement mask.

410 ERROR_KBD_REGISTER
KbdRegister not allowed.

411 ERROR_KBD_DEREGISTER
KbdDeRegister not allowed.

412 ERROR_MOUSE_SMG_ONLY
Valid from session manager only.

413 ERROR_MOUSE_INVALID_ASCIIZ
Invalid ASCIIZ length.

414 ERROR_MOUSE_INVALID_MASK
Invalid replacement mask.

415 ERROR_MOUSE_REGISTER
Mouse register not allowed.

416 ERROR_MOUSE_DEREGISTER
Mouse deregister not allowed.

417 ERROR_SMG_BAD_ACTION
Invalid action specified.

418 ERROR_SMG_INVALID_CALL
INIT called more than once, or invalid session identity.

419 ERROR_SCS_SG_NOTFOUND
New session number.

420 ERROR_SCS_NOT_SHELL
Caller is not shell.

421 ERROR_VIO_INVALID_PARMS
Invalid parameters passed.

422 ERROR_VIO_FUNCTION_OWNED
Save/restore already owned.

423 ERROR_VIO_RETURN
Non-destruct return (undo).

424 ERROR_SCS_INVALID_FUNCTION
Caller invalid function.

425 ERROR_SCS_NOT_SESSION_MGR
Caller not session manager.

426 ERROR_VIO_REGISTER
Vio register not allowed.

427 ERROR_VIO_NO_MODE_THREAD
No mode restore thread in SG.

428 ERROR_VIO_NO_SAVE_RESTORE_THD
No save/restore thread in SG.

429 ERROR_VIO_IN_BG
Function invalid in background.

430 ERROR_VIO_ILLEGAL_DURING_POPUP
Function not allowed during pop-up window.

431 ERROR_SMG_NOT_BASESHELL
Caller is not the base shell.

432 ERROR_SMG_BAD_STATUSREQ
Invalid status requested.

433 ERROR_QUE_INVALID_WAIT
NoWait parameter out of bounds.

434 ERROR_VIO_LOCK
Error returned from Scroll Lock.

435 ERROR_MOUSE_INVALID_IOWAIT
Invalid parameters for IOWait.

436 ERROR_VIO_INVALID_HANDLE
Invalid VIO handle.

437 ERROR_VIO_ILLEGAL_DURING_LOCK
Function not allowed during screen lock.

438 ERROR_VIO_INVALID_LENGTH
Invalid VIO length.

439 ERROR_KBD_INVALID_HANDLE
Invalid KBD handle.

440 ERROR_KBD_NO_MORE_HANDLE
Ran out of handles.

441 ERROR_KBD_CANNOT_CREATE_KCB
Unable to create kcb.

442 ERROR_KBD_CODEPAGE_LOAD_INCOMPL
Unsuccessful code-page load.

443 ERROR_KBD_INVALID_CODEPAGE_ID
Invalid code-page identity.

444 ERROR_KBD_NO_CODEPAGE_SUPPORT
No code page support.

445 ERROR_KBD_FOCUS_REQUIRED
Keyboard focus required.

446 ERROR_KBD_FOCUS_ALREADY_ACTIVE
Calling thread has an outstanding focus.

447 ERROR_KBD_KEYBOARD_BUSY
Keyboard is busy.

448 ERROR_KBD_INVALID_CODEPAGE
Invalid code page.

449 ERROR_KBD_UNABLE_TO_FOCUS
Focus attempt failed.

450 ERROR_SMG_SESSION_NON_SELECT
Session is not selectable.

451 ERROR_SMG_SESSION_NOT_FOREGRND
Parent/child session is not foreground.

452 ERROR_SMG_SESSION_NOT_PARENT
Not parent of requested child.

453 ERROR_SMG_INVALID_START_MODE
Invalid session start mode.

454 ERROR_SMG_INVALID_RELATED_OPT
Invalid session start related option.

455 ERROR_SMG_INVALID_BOND_OPTION
Invalid session bond option.

456 ERROR_SMG_INVALID_SELECT_OPT
Invalid session select option.

457 ERROR_SMG_START_IN_BACKGROUND
Session started in background.

458 ERROR_SMG_INVALID_STOP_OPTION
Invalid session stop option.

459 ERROR_SMG_BAD_RESERVE
Reserved parameters are not zero.

460 ERROR_SMG_PROCESS_NOT_PARENT
Session parent process already exists.

461 ERROR_SMG_INVALID_DATA_LENGTH
Invalid data length.

462 ERROR_SMG_NOT_BOUND
Parent is not bound.

463 ERROR_SMG_RETRY_SUB_ALLOC
Retry request block allocation.

464 ERROR_KBD_DETACHED
This call is not allowed for a detached PID.

465 ERROR_VIO_DETACHED
This call is not allowed for a detached PID.

466 ERROR_MOU_DETACHED
This call is not allowed for a detached PID.

467 ERROR_VIO_FONT
No font is available to support the mode.

468 ERROR_VIO_USER_FONT
User font is active.

469 ERROR_VIO_BAD_CP
Invalid code page specified.

470 ERROR_VIO_NO_CP
System displays do not support code page.

471 ERROR_VIO_NA_CP
Current display does not support code page.

472 ERROR_INVALID_CODE_PAGE
Invalid code page.

473 ERROR_CPLIST_TOO_SMALL
Code page list is too small.

474 ERROR_CP_NOT_MOVED
Code page was not moved.

475 ERROR_MODE_SWITCH_INIT
Mode switch initialization error.

476 ERROR_CODE_PAGE_NOT_FOUND
Code page was not found.

477 ERROR_UNEXPECTED_SLOT_RETURNED
Internal error.

478 ERROR_SMG_INVALID_TRACE_OPTION
Invalid start session trace indicator.

479 ERROR_VIO_INTERNAL_RESOURCE
VIO internal resource error.

480 ERROR_VIO_SHELL_INIT
VIO shell initialization error.

481 ERROR_SMG_NO_HARD_ERRORS
No session manager hard errors.

482 ERROR_CP_SWITCH_INCOMPLETE
DosSetProcessCp is unable to set a KBD or VIO code page.

483 ERROR_VIO_TRANSPARENT_POPUP
Error during VIO pop-up window.

484 ERROR_CRITSEC_OVERFLOW
Critical section overflow.

485 ERROR_CRITSEC_UNDERFLOW
Critical section underflow.

486 ERROR_VIO_BAD_RESERVE
Reserved parameter is not zero.

487 ERROR_INVALID_ADDRESS
Invalid physical address.

488 ERROR_ZERO_SELECTORS_REQUESTED
At least one selector must be requested.

489 ERROR_NOT_ENOUGH_SELECTORS_AVA
Not enough GDT selectors to satisfy request.

490 ERROR_INVALID_SELECTOR
Not a GDT selector.

491 ERROR_SMG_INVALID_PROGRAM_TYPE
Invalid program type.

492 ERROR_SMG_INVALID_PGM_CONTROL
Invalid program control.

493 ERROR_SMG_INVALID_INHERIT_OPT
Invalid inherit option.

494 ERROR_VIO_EXTENDED_SG

495 ERROR_VIO_NOT_PRES_MGR_SG

496 ERROR_VIO_SHIELD_OWNED

497 ERROR_VIO_NO_MORE_HANDLES

498 ERROR_VIO_SEE_ERROR_LOG

499 ERROR_VIO_ASSOCIATED_DC

500 ERROR_KBD_NO_CONSOLE

501 ERROR_MOUSE_NO_CONSOLE

502 ERROR_MOUSE_INVALID_HANDLE

503 ERROR_SMG_INVALID_DEBUG_PARMS

504 ERROR_KBD_EXTENDED_SG

505 ERROR_MOU_EXTENDED_SG

506 ERROR_SMG_INVALID_ICON_FILE

507 ERROR_TRC_PID_NON_EXISTENT

508 ERROR_TRC_COUNT_ACTIVE

509 ERROR_TRC_SUSPENDED_BY_COUNT

510 ERROR_TRC_COUNT_INACTIVE

511 ERROR_TRC_COUNT_REACHED

512 ERROR_NO_MC_TRACE

513 ERROR_MC_TRACE

514 ERROR_TRC_COUNT_ZERO

515 ERROR_SMG_TOO_MANY_DDS

516 ERROR_SMG_INVALID_NOTIFICATION

517 ERROR_LF_INVALID_FUNCTION

518 ERROR_LF_NOT_AVAIL

519 ERROR_LF_SUSPENDED

520 ERROR_LF_BUF_TOO_SMALL

521 ERROR_LF_BUFFER_CORRUPTED

521 ERROR_LF_BUFFER_FULL

522 ERROR_LF_INVALID_DAEMON

522 ERROR_LF_INVALID_RECORD

523 ERROR_LF_INVALID_TEMPL

523 ERROR_LF_INVALID_SERVICE

524 ERROR_LF_GENERAL_FAILURE

525 ERROR_LF_INVALID_ID

526 ERROR_LF_INVALID_HANDLE

527 ERROR_LF_NO_ID_AVAIL

528 ERROR_LF_TEMPLATE_AREA_FULL

529 ERROR_LF_ID_IN_USE

530 ERROR_MOU_NOT_INITIALIZED

531 ERROR_MOUINITREAL_DONE

532 ERROR_DOSSUB_CORRUPTED

533 ERROR_MOUSE_CALLER_NOT_SUBSYS

534 ERROR_ARITHMETIC_OVERFLOW

535 ERROR_TMR_NO_DEVICE

536 ERROR_TMR_INVALID_TIME

537 ERROR_PVW_INVALID_ENTITY

538 ERROR_PVW_INVALID_ENTITY_TYPE

539 ERROR_PVW_INVALID_SPEC

540 ERROR_PVW_INVALID_RANGE_TYPE

541 ERROR_PVW_INVALID_COUNTER_BLK

542 ERROR_PVW_INVALID_TEXT_BLK

543 ERROR_PRF_NOT_INITIALIZED

544 ERROR_PRF_ALREADY_INITIALIZED

545 ERROR_PRF_NOT_STARTED

546 ERROR_PRF_ALREADY_STARTED

547 ERROR_PRF_TIMER_OUT_OF_RANGE

548 ERROR_PRF_TIMER_RESET

639 ERROR_VDD_LOCK_USEAGE_DENIED

640 ERROR_TIMEOUT

641 ERROR_VDM_DOWN

642 ERROR_VDM_LIMIT

643 ERROR_VDD_NOT_FOUND

644 ERROR_INVALID_CALLER

645 ERROR_PID_MISMATCH

646 ERROR_INVALID_VDD_HANDLE

647 ERROR_VLPT_NO_SPOOLER

648 ERROR_VCOM_DEVICE_BUSY

649 ERROR_VLPT_DEVICE_BUSY

650 ERROR_NESTING_TOO_DEEP

651 ERROR_VDD_MISSING

691 ERROR_IMP_INVALID_PARM

692 ERROR_IMP_INVALID_LENGTH

693 MSG_HPFS_DISK_ERROR_WARN

730 ERROR_MON_BAD_BUFFER

731 ERROR_MODULE_CORRUPTED

2055 ERROR_LF_TIMEOUT

2057 ERROR_LF_SUSPEND_SUCCESS

2058 ERROR_LF_RESUME_SUCCESS

2059 ERROR_LF_REDIRECT_SUCCESS

2060 ERROR_LF_REDIRECT_FAILURE

32768 ERROR_SWAPPER_NOT_ACTIVE

32769 ERROR_INVALID_SWAPID

32770 ERROR_IOERR_SWAP_FILE

32771 ERROR_SWAP_TABLE_FULL

32772 ERROR_SWAP_FILE_FULL

32773 ERROR_CANT_INIT_SWAPPER

32774 ERROR_SWAPPER_ALREADY_INIT

32775 ERROR_PMM_INSUFFICIENT_MEMORY

32776 ERROR_PMM_INVALID_FLAGS

32777 ERROR_PMM_INVALID_ADDRESS

32778 ERROR_PMM_LOCK_FAILED

32779 ERROR_PMM_UNLOCK_FAILED

32780 ERROR_PMM_MOVE_INCOMPLETE

32781 ERROR_UCOM_DRIVE_RENAMED

32782 ERROR_UCOM_FILENAME_TRUNCATED

32783 ERROR_UCOM_BUFFER_LENGTH

32784 ERROR_MON_CHAIN_HANDLE

32785 ERROR_MON_NOT_REGISTERED

32786 ERROR_SMG_ALREADY_TOP

32787 ERROR_PMM_ARENA_MODIFIED

32788 ERROR_SMG_PRINTER_OPEN

32789 ERROR_PMM_SET_FLAGS_FAILED

32790 ERROR_INVALID_DOS_DD

32791 ERROR_BLOCKED

32792 ERROR_NOBLOCK

32793 ERROR_INSTANCE_SHARED

32794 ERROR_NO_OBJECT

32795 ERROR_PARTIAL_ATTACH

32796 ERROR_INCACHE

32797 ERROR_SWAP_IO_PROBLEMS

32798 ERROR_CROSSES_OBJECT_BOUNDARY

32799 ERROR_LONGLOCK

32800 ERROR_SHORTLOCK

32801 ERROR_UVIRTLOCK

32802 ERROR_ALIASLOCK

32803 ERROR_ALIAS

32804 ERROR_NO_MORE_HANDLES

32805 ERROR_SCAN_TERMINATED

32806 ERROR_TERMINATOR_NOT_FOUND

32807 ERROR_NOT_DIRECT_CHILD

32808 ERROR_DELAY_FREE

32809 ERROR_GUARDPAGE

32900 ERROR_SWAPERROR

32901 ERROR_LDRERROR

32902 ERROR_NOMEMORY

32903 ERROR_NOACCESS

32904 ERROR_NO_DLL_TERM

65026 ERROR_CPSIO_CODE_PAGE_INVALID

65027 ERROR_CPSIO_NO_SPOOLER

65028 ERROR_CPSIO_FONT_ID_INVALID

65033 ERROR_CPSIO_INTERNAL_ERROR

65034 ERROR_CPSIO_INVALID_PTR_NAME

65037 ERROR_CPSIO_NOT_ACTIVE

65039 ERROR_CPSIO_PID_FULL

65040 ERROR_CPSIO_PID_NOT_FOUND

65043 ERROR_CPSIO_READ_CTL_SEQ

65045 ERROR_CPSIO_READ_FNT_DEF

65047 ERROR_CPSIO_WRITE_ERROR

65048 ERROR_CPSIO_WRITE_FULL_ERROR

65049 ERROR_CPSIO_WRITE_HANDLE_BAD

65074 ERROR_CPSIO_SWIT_LOAD

65077 ERROR_CPSIO_INV_COMMAND

65078 ERROR_CPSIO_NO_FONT_SWIT

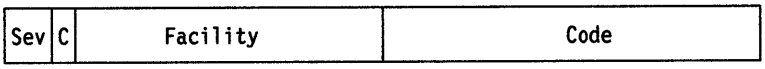
65079 ERROR_ENTRY_IS_CALLGATE

Appendix C. System Exceptions

The operating system defines a class of error conditions called *exceptions*, and specifies the default actions that are taken when these exceptions occur. The system default action in most cases is to terminate the thread that caused the exception.

Exception values have the following 32-bit format:

```
3 3 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1
1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
```



"Sev" is the severity code, as follows:

Value	Meaning
00	Success
01	Informational
10	Warning
11	Error

"C" is the Customer code flag.

"Facility" is the facility code.

"Code" is the facility's status code.

Exceptions that are specific to OS/2 Version 2.00 (for example, XCPT_SIGNAL) have a facility code of 1.

System exceptions include both synchronous and asynchronous exceptions. Synchronous exceptions are caused by events that are internal to a thread's execution. For example, synchronous exceptions could be caused by invalid parameters, or by a thread's request to end its own execution.

Asynchronous exceptions are caused by events that are external to a thread's execution. For example, an asynchronous exception can be caused by a user's entering a Ctrl+C or Ctrl+Break key sequence, or by a process' issuing DosKillProcess to end the execution of another process.

The Ctrl+Break and Ctrl+C exceptions are also known as *signals*, or as *signal exceptions*.

The following tables show the symbolic names of system exceptions, their numerical values, and related information fields.

Table C-1. Portable, Non-Fatal, Software-Generated Exceptions

Exception Name	**Value**
XCPT_GUARD_PAGE_VIOLATION ExceptionInfo[0] - R/W flag ExceptionInfo[1] - FaultAddr	0x80000001
XCPT_UNABLE_TO_GROW_STACK	0x80010001

Table C-2. Portable, Fatal, Hardware-Generated Exceptions		
Exception Name	**Value**	**Related Trap**
XCPT_ACCESS_VIOLATION ExceptionInfo[0] - Flags XCPT_UNKNOWN_ACCESS 0x0 XCPT_READ_ACCESS 0x1 XCPT_WRITE_ACCESS 0x2 XCPT_EXECUTE_ACCESS 0x4 XCPT_SPACE_ACCESS 0x8 XCPT_LIMIT_ACCESS 0x10 ExceptionInfo[1] - FaultAddr	0xC0000005	0x09, 0x0B, 0x0C, 0x0D, 0x0E
XCPT_INTEGER_DIVIDE_BY_ZERO	0xC000009B	0
XCPT_FLOAT_DIVIDE_BY_ZERO	0xC0000095	0x10
XCPT_FLOAT_INVALID_OPERATION	0xC0000097	0x10
XCPT_ILLEGAL_INSTRUCTION	0xC000001C	0x06
XCPT_PRIVILEGED_INSTRUCTION	0xC000009D	0x0D
XCPT_INTEGER_OVERFLOW	0xC000009C	0x04
XCPT_FLOAT_OVERFLOW	0xC0000098	0x10
XCPT_FLOAT_UNDERFLOW	0xC000009A	0x10
XCPT_FLOAT_DENORMAL_OPERAND	0xC0000094	0x10
XCPT_FLOAT_INEXACT_RESULT	0xC0000096	0x10
XCPT_FLOAT_STACK_CHECK	0xC0000099	0x10
XCPT_DATATYPE_MISALIGNMENT ExceptionInfo[0] - R/W flag ExceptionInfo[1] - Alignment ExceptionInfo[2] - FaultAddr	0xC000009E	0x11
XCPT_BREAKPOINT	0xC000009F	0x03
XCPT_SINGLE_STEP	0xC00000A0	0x01

Table C-3. Portable, Fatal, Software-Generated Exceptions		
Exception Name	**Value**	**Related Trap**
XCPT_IN_PAGE_ERROR ExceptionInfo[0] - FaultAddr	0xC0000006	0x0E
XCPT_PROCESS_TERMINATE	0xC0010001	
XCPT_ASYNC_PROCESS_TERMINATE ExceptionInfo[0] - TID of terminating thread	0xC0010002	
XCPT_NONCONTINUABLE_EXCEPTION	0xC0000024	
XCPT_INVALID_DISPOSITION	0xC0000025	

Table C-4. Non-Portable, Fatal Exceptions		
Exception Name	**Value**	**Related Trap**
XCPT_INVALID_LOCK_SEQUENCE	0xC000001D	
XCPT_ARRAY_BOUNDS_EXCEEDED	0xC0000093	0x05

Table C-5. Unwind Operation Exceptions	
Exception Name	**Value**
XCPT_UNWIND	0xC0000026
XCPT_BAD_STACK	0xC0000027
XCPT_INVALID_UNWIND_TARGET	0xC0000028

Table C-6. Fatal Signal Exceptions	
Exception Name	**Value**
XCPT_SIGNAL ExceptionInfo[0] - Signal Number	0xC0010003

System Exception Descriptions

XCPT_ACCESS_VIOLATION

Exception Description

Access Violation

An access violation exception is generated when an attempt is made either to load or store data in an inaccessible location, or to execute an inaccessible instruction. This exception corresponds to both the Intel 80386 general protection fault (#13), caused by an invalid access attempt; and the page fault (#14), caused by an attempt to access an uncommitted page or a page with incorrect attributes for the desired operation.

Exception Code:

XCPT_ACCESS_VIOLATION (0xC0000005)

Handler Information:

The ExceptionAddress field in the ExceptionReportRecord points to the instruction that caused the exception. This exception is continuable.

Default Action:

The process is ended.

Additional Parameters (2):

Exception Info[0] - Flags

XCPT_UNKNOWN_ACCESS (0x0)
XCPT_READ_ACCESS (0x1)
XCPT_WRITE_ACCESS (0x2)
XCPT_EXECUTE_ACCESS (0x4)
XCPT_SPACE_ACCESS (0x8)
XCPT_LIMIT_ACCESS (0x10)

Exception Info[1] - FaultAddr

The virtual address (if available) of the data that is not accessible, or XCPT_DATA_UNKNOWN.

XCPT_BREAKPOINT

Exception Description

Breakpoint

A breakpoint exception occurs when a breakpoint instruction is executed. This exception is intended for use by debuggers. This exception is continuable.

Exception Code:
XCPT_BREAKPOINT (0xC0000006)

Default Action:
The process is ended.

Additional Parameters:
None.

XCPT_ARRAY_BOUNDS_EXCEEDED

Exception Description

Bounds Check

The bounds check exception corresponds to the Intel 80386 bounds check fault (#5), caused by a BOUND instruction that fails.

Exception Code:
XCPT_ARRAY_BOUNDS_EXCEEDED (0xC0000093)

Handler Information:
The CS:EIP in the exception context structure points to the instruction that caused the exception. This exception is continuable.

Default Action:
The process is ended.

Additional Parameters:
None.

XCPT_DATATYPE_MISALIGNMENT

Exception Description

Data-Type Misalignment

A data-type misalignment exception is generated when an attempt is made to load or store data in an address that is not naturally aligned on a hardware architecture that does not provide alignment hardware. For example, 16-bit entities must be aligned on two-byte boundaries, and 32-bit entities must be aligned on four-byte boundaries. This exception does not occur on the Intel 80386 processor. This exception is continuable.

Exception Code:
XCPT_DATATYPE_MISALIGNMENT (0xC000009E)

Default Action:
The process is ended.

Additional Parameters (3):
Exception Info[0] - Read/Write Flag

XCPT_READ_ACCESS, or
XCPT_WRITE_ACCESS.

Exception Info[1] - Data-type Mask

A data-type mask that specifies how many low-address bits must be zero. For example, the data-type mask for a 16-bit entity is one, a 32-bit entity three, and so on.

Exception Info[2] - Virtual Address

The virtual address of the misaligned data.

XCPT_FLOAT_DIVIDE_BY_ZERO

Exception Description

Floating Divide-by-Zero

A floating divide-by-zero exception is generated when an attempt is made to divide a floating-point dividend by a floating-point divisor of zero. This exception is continuable.

Exception Code:
XCPT_FLOAT_DIVIDE_BY_ZERO (0xC0000095)

Default Action:
The process is ended.

Additional Parameters:
None.

XCPT_FLOAT_OVERFLOW

Exception Description

Floating Overflow

A floating overflow exception is generated when the resulting exponent of a floating-point operation is greater than the magnitude allowed for the respective floating point data type. This exception is continuable.

Exception Code:
XCPT_FLOAT_OVERFLOW (0xC0000098)

Default Action:
The process is ended.

Additional Parameters:
None.

XCPT_FLOAT_UNDERFLOW

Exception Description

Floating Underflow

A floating underflow exception is generated when the resulting exponent of a floating-point operation is less than the magnitude provided for the respective floating-point data type. This exception is continuable.

Exception Code:
XCPT_FLOAT_UNDERFLOW (0xC000009A)

Default Action:
The process is ended.

Additional Parameters:
None.

XCPT_FLOAT_INVALID_OPERATION

Exception Description

Invalid Floating-Point Operation

This exception usually indicates a programming error corresponding to the invalid floating-point operations defined in IEEE Standard 754. The Intel 80386 processor raises trap #16. This exception is continuable.

Default Action:
The process is ended.

Exception Code:
XCPT_FLOAT_INVALID_OPERATION (0xC0000097)

Additional Parameters:
None.

XCPT_FLOAT_DENORMAL_OPERAND

Exception Description

Denormalized Operand

A denormalized operand exception occurs when the 80387 NPX processor attempts an arithmetic operation on a denormal operand, and the user has not masked off denormal operations. This exception is continuable.

Exception Code:
XCPT_FLOAT_DENORMAL_OPERAND (0xC0000094)

Default Action:
The process is ended.

Additional Parameters:
None.

XCPT_FLOAT_INEXACT_RESULT

Exception Description

Loss of Precision

A loss of precision exception occurs when the result of an operation is not exactly representable in the destination format. For example, the fraction 1/3 cannot be exactly represented in binary form. For the Intel 80386 and 80387 processors, this corresponds to one of the class of exceptions for which the 80387 processor signals the 80386 processor to raise trap #16. This exception is continuable.

Default Action:
The process is ended.

Exception Code:
XCPT_FLOAT_INEXACT_RESULT (0xC0000096)

Additional Parameters:
None.

XCPT_FLOAT_STACK_CHECK

Exception Description

Invalid Floating-Point Stack Operation

An invalid floating-point stack check is raised when a floating-point processor attempts an illegal operation on a private stack. The Intel 80387 processor maintains eight internal 10-byte "registers" that are individually addressable and yet behave as a push-down stack under the influence of the FLD (push real) and FST (pop real to destination) instructions. Overflow and underflow are checked with each instruction, and this exception is raised when appropriate. This is one of the class of exceptions for which the Intel 80387 processor signals the Intel 80386 processor to raise trap #16. This exception is continuable.

Exception Code:
XCPT_FLOAT_STACK_CHECK (0xC0000099)

Default Action:
The process is ended.

Additional Parameters:
None.

XCPT_ILLEGAL_INSTRUCTION

Exception Description

Illegal Instruction

An illegal instruction exception is generated when an attempt is made to execute an instruction whose operation is not defined for the host machine architecture. On the Intel 80386 processor, this corresponds to the invalid opcode fault (#6), caused by any invalid instruction. This exception is continuable.

Exception Code:
XCPT_ILLEGAL_INSTRUCTION (0xC000001C)

Default action:
The process is ended.

Additional Parameters:
None.

XCPT_PRIVILEGED_INSTRUCTION

Exception Description

Privileged Instruction

A privileged instruction exception is generated when an attempt is made to execute an instruction whose operation is not allowed in the current machine mode. For example, an attempt is made to execute an instruction in user mode that is only allowed in kernel mode. This exception is continuable.

Exception Code:
XCPT_PRIVILEGED_INSTRUCTION (0xC000009D)

Default Action:
The process is ended.

Additional Parameters:
None.

XCPT_INVALID_LOCK_SEQUENCE

Exception Description

Invalid Lock Sequence

An invalid lock sequence exception is generated when an attempt is made to execute an operation within an interlocked section of code, and the sequence is invalid for the host machine architecture. This exception is continuable.

Exception Code:
XCPT_INVALID_LOCK_SEQUENCE (0xC000001D)

Default Action:
The process is ended.

Additional Parameters:
None.

XCPT_INTEGER_DIVIDE_BY_ZERO

Exception Description

Integer Divide-by-Zero

An integer divide-by-zero exception is generated when an attempt is made to divide an integer dividend by an integer divisor of zero. On the Intel 80387 processor, this is a divide by zero fault (#0), caused by a DIV or IDIV by zero operation. This exception is continuable.

Exception Code:
XCPT_INTEGER_DIVIDE_BY_ZERO (0xC000009B)

Default action:
The process is ended.

Additional Parameters:
None.

XCPT_INTEGER_OVERFLOW

Exception Description

Integer Overflow

An integer overflow exception is generated when the result of an integer operation causes a carry-out of the the most significant bit of the result, which is not the same as the carry-into of the most significant bit of the result. For example, the addition of two positive integers produces a negative result. On the Intel 80387 processor, this corresponds to overflow trap (#4), caused by executing an INTO instruction with the OF flag set. This exception is continuable.

Exception Code:
XCPT_INTEGER_OVERFLOW (0xC000009C)

Default action:
The process is ended.

Additional Parameters:
None.

XCPT_SINGLE_STEP

Exception Description

Single Step

A single-step exception is generated when a trace trap or other single instruction execution mechanism signals that one instruction has been executed. This exception is intended for use by debuggers. This exception is continuable.

Default Action:
The process is ended

Exception Code:
XCPT_SINGLE_STEP (0xC00000A0)

Additional Parameters:
None.

XCPT_GUARD_PAGE_VIOLATION

Exception Description

Guard Page Violation

A guard page violation exception is generated when an attempt is made to load or store data in a location that is contained within a guard page. Memory management software immediately turns the guard page into a demand zero page and initiates a guard page violation exception.

Exception Code:
XCPT_GUARD_PAGE_VIOLATION (0x800001)

Default Action:
Execution continues. If possible, the memory page immediately below the guard page is allocated and marked as a guard page. The higher guard page is marked to no longer be a guard page, and the instruction is restarted. This allows for dynamic stack growth. If it is not possible to allocate another page below the faulting page, an XCPT_UNABLE_TO_GROW_STACK exception is raised. This exception is continuable.

Additional Parameters (2):

ExceptionInfo[0] - Read/Write Flag

XCPT_READ_ACCESS, or
XCPT_WRITE_ACCESS.

ExceptionInfo[1] - Virtual Address

The virtual address of the data within a guard page.

XCPT_UNABLE_TO_GROW_STACK

Exception Description

Unable to Grow Stack

The default action for a guard page violation is to attempt to allocate another page of memory immediately below the page on which the fault occurred, thereby implementing dynamic stack growth. If this attempt fails, XCPT_UNABLE_TO_GROW_STACK is generated, indicating that the thread has, at most, one more page of stack space available. This exception is continuable.

Exception Code:

XCPT_UNABLE_TO_GROW_STACK (0x80010001)

Default Action:

Execution continues.

Additional Parameters:

None.

XCPT_BAD_STACK

Exception Description

Bad Stack

This exception is raised when an ExceptionRegistrationRecord is reached that is not properly aligned or is not within the current stack boundaries. It is also raised if an unwind target is specified that does not point to an ExceptionRegistrationRecord. This exception is noncontinuable.

Exception Code:

XCPT_BAD_STACK (0xC0000027)

Default Action:

The process is ended.

Additional Parameters:

None.

XCPT_INVALID_UNWIND_TARGET

Exception Description

Invalid Unwind Target

This exception is raised when the address of the target ExceptionRegistrationRecord is below the current stack pointer. This exception is noncontinuable.

Exception Code:

XCPT_INVALID_UNWIND_TARGET

Default Action:

The process is ended.

Additional Parameters:

None.

XCPT_IN_PAGE_ERROR

Exception Description

Page Read Error

A page read error exception is generated when an attempt is made to read a page into memory and an I/O error is encountered. This exception is continuable.

Exception Code:
XCPT_IN_PAGE_ERROR (0xC0000006)

Default Action:
The process is ended.

Additional Parameters (1):
ExceptionInfo[0] - Virtual Address

A virtual address within the page that was being read.

XCPT_INVALID_DISPOSITION

Exception Description

Invalid Disposition

This exception is raised when an exception handler returns anything except XCPT_CONTINUE_EXECUTION or XCPT_CONTINUE_SEARCH. This exception is *not* continuable.

Execution Code:
XCPT_INVALID_DISPOSITION (0xC0000025)

Default Action:
The process is ended.

Additional Parameters:
None.

XCPT_NONCONTINUABLE_EXCEPTION

Exception Description

Continuing a Noncontinuable Exception

This exception is raised when an exception handler returns XCPT_CONTINUE_EXECUTION in response to a noncontinuable exception. This exception is *not* continuable.

Execution Code:
XCPT_NONCONTINUABLE_EXCEPTION (0xC0000024)

Default Action:
The process is ended.

Additional Parameters:
None.

XCPT_PROCESS_TERMINATE

Exception Description

Process Termination

There are two exceptions a thread may receive when it is about to end:

XCPT_PROCESS_TERMINATE, and
XCPT_ASYNC_PROCESS_TERMINATE.

A thread receives XCPT_PROCESS_TERMINATE after it calls DosExit to end itself or the entire process. This exception is *not* continuable.

Additional Parameters:
None.

XCPT_ASYNC_PROCESS_TERMINATE

Exception Description

There are two exceptions a thread may receive when it is about to end:

XCPT_PROCESS_TERMINATE, and
XCPT_ASYNC_PROCESS_TERMINATE.

A thread receives XCPT_ASYNC_PROCESS_TERMINATE when another thread in the process has caused it to end. For example, another thread has called DosExit to end the process, or has not handled a fatal exception, and so on. This exception is continuable.

Additional Parameters (1):
ExceptionInfo[0] - TID

The thread identification of the terminating thread.

XCPT_UNWIND

Exception Description

Unwinding

The system fills in an exception number for an unwind if the user chooses not to do so. Note that an ExceptionReportRecord containing XCPT_UNWIND does not indicate that an exception has occurred, but rather that an unwind is in progress.

Exception Code:
XCPT_UNWIND (0xC0000026)

Default Action:
Does not apply.

Additional Parameters:
None.

XCPT_SIGNAL

Exception Description

Signal Exceptions

An OS/2 Version 2.00 application may receive three signals:

XCPT_SIGNAL_INTR (Ctrl+C)
XCPT_SIGNAL_KILLPROC (DosKillProcess)
XCPT_SIGNAL_BREAK (Ctrl+Break).

The signal being sent may be determined by examining the exception information in the ExceptionReportRecord.

Exception Code:
XCPT_SIGNAL (0xC0010003)

Default Action:
The process is ended.

Additional Parameters (1):
ExceptionInfo[0] - Signal Number

Number	Signal
1	XCPT_SIGNAL_INTR
3	XCPT_SIGNAL_KILLPROC
4	XCPT_SIGNAL_BREAK

XCPT_SIGNAL is called a "signal exception" and is sent only to thread 1 in the process receiving the exception. This is consistent with 16-bit signals, and provides greater consistency in the environment of the process for handling the various asynchronous exceptions. For example, since a repeated typematic Ctrl+C could possibly cause the thread to recursively process the exception and consume stack space without ever being able to handle the first "signal", the exception dispatcher "holds" each exception of the same type until a handler either returns XCPT_CONTINUE_EXECUTION to the exception dispatcher, or the process calls DosAcknowledgeSignalException for that signal. Only one signal or exception is actually held (they are not queued by the system).

DosSetSignalExceptionFocus indicates to the system that the process wants to receive the XCPT_SIGNAL_INTR and XCPT_SIGNAL_BREAK signals. Previously, when a process called DosSetSigHandler, the system noted that the process was aware of the particular signal for which it was registering the handler. When a process called DosSetSigHandler, it became a candidate for the "signal focus" for its session. At any point in time, the focus for a session is the last process to register a signal handler for that signal. When the user presses Ctrl+C on the keyboard, the system delivers an XCPT_SIGNAL_INTR signal to the current keyboard focus. The user could also press Ctrl+Break to deliver an XCPT_SIGNAL_BREAK signal, but this would only work if input were in raw mode.

Note that all exception handlers (on thread 1) must be prepared to "see" signal exceptions. It is always possible that a previous handler has issued DosSetSignalExceptionFocus or that a Dos16SetSigHandler has been issued by some 16-bit code in the path. They can always be ignored by returning XCPT_CONTINUE_SEARCH to the exception dispatcher. Note that signals result in a call to the 16-bit signal handler (if installed) if all the 32-bit exception handlers return XCPT_CONTINUE_SEARCH.

DosSetSignalExceptionFocus performs the function of assigning the signal focus exactly as if the application had called DosSetSigHandler twice, once for each signal. The process calls DosSetSignalExceptionFocus when it wants to indicate that it expects to receive XCPT_SIGNAL_INTR or XCPT_SIGNAL_BREAK after it has registered an exception handler to process the signal when it comes. Each call to DosSetSignalExceptionFocus increments a counter in the PTDA of the process. When the system attempts to send XCPT_SIGNAL_INTR or XCPT_SIGNAL_BREAK to a process, it first checks to see if either this counter is greater than zero, or if the process has registered a 16-bit signal handler for that signal. If either of these is true, the signal will be sent. If the process has registered both 16-bit and 32-bit handlers, the 32-bit handlers are called first. If they do not handle the signal, the 16-bit handlers are called. If the 32-bit handlers are called and do not handle the signal, and there are no 16-bit handlers, the process is terminated.

Appendix D. DosDebug Commands

Not all fields must be defined for every DosDebug command. The same field can have a different meaning in different DosDebug commands. For each command, fields in the Debug Buffer structure that are not listed are not useful for that command, but may be modified by DosDebug as required.

Error cases for commands are not listed. The listed return values from commands are valid only if the DBG_N_Success notification is given.

DBG_C_Null - Debug Command 0

Null Command

Parameters

Pid = Process ID of debuggee
Cmd = DBG_C_Null

No operation is performed on the debuggee. You can issue this command to verify the process ID of the debuggee, and to check if the debuggee is active.

Pid must be valid, or an error is returned.

DBG_C_ReadMem - Debug Command 1 and 2

DBG_C_ReadMem_I - Debug Command 1 and 2

DBG_C_ReadMem_D - Debug Command 1 and 2

Read Word Command

Parameters

Pid = Process ID of debuggee
Addr = Address to read from
Cmd = DBG_C_ReadMem_I, or DBG_C_ReadMem_D, or DBG_C_ReadMem

The commands DBG_C_ReadMem_I, DBG_C_ReadMem_D, and DBG_C_ReadMem are identical.

Returns

The word at the desired address is read, and stored into Value.

Value = Word read from the specified address.

Restrictions

You are unable to read from any memory outside user space.

The high-order word of Value is set to zero.

DBG_C_ReadReg - Debug Command 3

Read Register Set Command

Parameters

Pid = Process ID of debuggee
Tid = Thread ID of register set to read
Cmd = DBG_C_ReadReg

If Tid is zero and the debuggee is stopped, the register set comes from the active debuggee thread.
If Tid is zero and the debuggee is executing, ERROR_INVALID_THREADID is returned.

Returns

The register set in the Debug Buffer is updated, including the selector information as follows:

Tid = Thread ID corresponding to the register set
MTE = Program module's MTE (Module Table Entry) handle

DBG_C_WriteMem - Debug Command 4 and 5

DBG_C_WriteMem_I - Debug Command 4 and 5

DBG_C_WriteMem_D - Debug Command 4 and 5

Write Word Command

Parameters

Pid = Process ID of debuggee
Addr = Address to write to
Value = Word to write
Cmd = DBG_C_WriteMem_I, or DBG_C_WriteMem_D, or DBG_C_WriteMem

The commands DBG_C_WriteMem_I, DBG_C_WriteMem_D, and DBG_C_WriteMem are identical.

Returns

The word in Value is written to the specified address.

In the case of a write to shared read-only memory, the memory is converted to private, and any set dynamic RAS tracepoints are removed from that memory, before the write is performed.

The area will continue to be shared by other processes, if any. In this way, breakpoints may be set in the debuggee without affecting the other modules.

Restrictions

You are unable to write to any memory outside user space.

The high-order word of Value is ignored.

DBG_C_WriteReg - Debug Command 6

Write Register Set Command

Parameters

Pid = Process ID of debuggee
Tid = Nonzero Thread ID of register set to write
Cmd = DBG_C_WriteReg

The register set in the Debug Buffer should contain the desired values.

Returns

Tid = Thread ID corresponding to the register set.

All registers are updated to the stored values. The access rights, limits, and Eflags are also updated to match the current values.

An error is returned if the selectors are not accessible by user space code, or are not valid memory segments.

Restrictions

Reserved system or processor flags bits are not modified via this method, but are masked to their correct values. The selector access rights and limits cannot be modified. The flags, access rights, and limits in the Debug Buffer are updated to the actual values.

DBG_C_Go - Debug Command 7

Go Command

Parameters

Pid = Process ID of debuggee
Cmd = DBG_C_Go

Returns

All non-frozen threads of the debuggee are allowed to execute user code at once. If all of the debuggee threads are frozen, an error is returned.

The Go command completes when a DosDebug event (such as a Breakpoint) occurs. This event can be any one of the DosDebug notifications. See Appendix E, "DosDebug Notifications" on page E-1.

When the next DosDebug event occurs, all threads in the debuggee process are marked to not execute any additional user code until the next Go command is issued. This provides a stable environment for debugging.

When the Go command returns, the register set is automatically updated to reflect the thread that detected the event.

DBG_C_Term - Debug Command 8

Terminate Command

Parameters

Pid = Process ID of debuggee
Cmd = DBG_C_Term

Returns

The debuggee process is terminated immediately.

No additional DosDebug commands or notifications will be allowed to this process. Outstanding memory aliases and watchpoints will be invalidated automatically.

Debuggee DosExecPgm processing will be attempted, but any unexpected DosDebug event (such as a Breakpoint) during this period will cause the process to terminate without completing DosExitList processing. For this reason, data watchpoints will automatically be cleared before attempting DosExitList processing.

If the Terminate command is issued during DosExitList processing, DosExitList processing will terminate immediately, without continuing the DosExitList routines.

DBG_C_SStep - Debug Command 9

Single Step Command

Parameters

Pid = Process ID of debuggee
Tid = Thread ID of thread to single-step
Cmd = DBG_C_SStep

If Tid is zero, all threads will be marked to single-step at once, and the first thread to be scheduled to execute user-space code will single-step. No other threads will single-step.

Returns

Usually, the Exception notification is returned, but any notification may be returned. See Appendix E, DosDebug Notifications.

Callgates that result in a privilege level transition to ring 0 will appear to single-step as a single instruction, with the single-step occurring just after the function completes. This hides ring 0 execution from debuggers.

Attempting to single-step any thread that is frozen results in an error.

Restrictions

The Single Step command has two modes of operation, as follows:

- If Tid is zero, the current thread is single-stepped while allowing all other threads to execute.
- If Tid is nonzero, a specific thread is selected for single-stepping. Only that thread is executed, even if it is single-stepping a kernel function that can potentially cause a deadlock condition.

The single-step notification is not lost if the single-step operation causes a notification to be sent to DosDebug. In this case, the single-step notification is queued.

The single-step operation is not lost if other notifications were queued before the Single Step command was issued. The Debug Continue command will clear the notifications one at a time until DosDebug has been completely notified. On the last Continue command, the single-step operation will take place as originally requested.

When a single-step operation is interrupted by an exception, the EIP (instruction pointer) should be moved to the next RING3 instruction. This may be in ring 3 system code. The single-step notification will be issued at this time.

The Single Step command correctly single-steps most instructions. Single-stepping some REP instructions may not work correctly due to errors in the 80386 processor.

DBG_C_Stop - Debug Command 10

Stop Command

Parameters

Pid = Process ID of debuggee
Cmd = DBG_C_Stop

Returns

The function performed by this command depends on the current state of the debuggee process, as follows:

- If the debuggee is already stopped:

  If there is a pending notification from the current thread, it is returned. See Appendix E, DosDebug Notifications for information about pending notifications.

  If there is no pending notification from the current thread, DBG_N_Success is returned.

- If the debuggee is executing user code:

  The debuggee is marked to stop before the next time it is ready to execute user-space (ring 2 or 3) code. This is known as an Asynchronous Stop command.

  Kernel operations will not be interrupted for this Asynchronous Stop command. That is, threads blocked in the kernel (via a semaphore or internal operation) will not be interrupted. However, an infinite loop in user space will be stopped.

  Note that the AsyncStop variation of this command implies a debugger with a minimum of two threads - one waiting for a Go or SStep command to finish, and another executing the Stop command.

DBG_C_Freeze - Debug Command 11

Freeze Thread Command

Parameters

Pid = Process ID of debuggee
Tid = Thread ID of thread to freeze
Cmd = DBG_C_Freeze

If Tid is zero, all debuggee threads will be frozen.

Returns

The desired threads are prevented from executing user code on a Go or SStep command.

By using the Freeze and Resume commands, a given set of threads can be executed at once, while keeping the other threads suspended.

No error is returned if the thread was previously frozen; it just remains frozen. Freeze and Resume commands cannot be nested.

If the Tid is specified as zero, it will be set to the thread ID of the debuggee thread most recently scheduled to execute.

DBG_C_Resume - Debug Command 12

Resume Thread Command

Parameters

Pid = Process ID of debuggee
Tid = Thread ID of thread to thaw
Cmd = DBG_C_Resume

If Tid is zero, all debuggee threads will be thawed.

Returns

The Resume command complements the Freeze command. A thread that has been resumed will function as if it were never frozen.

No error is returned if the thread was not previously frozen.

If the Tid is specified as zero, it will be set to the thread ID of the debuggee thread most recently scheduled to execute.

DBG_C_NumToAddr - Debug Command 13

Convert Object Number to Address Command

Parameters

Pid = Process ID of debuggee
Cmd = DBG_C_NumToAddr
Value = Logical object number in module
MTE = Module handle of module of interest

Returns

Addr = Starting address of object
Value = Logical object number
MTE = Module handle of module of interest

The logical object number in Value is converted into an address that points to the starting address of the desired logical object of the specified module in the debuggee's memory space. This address is then stored in the Addr field, in the form of a linear address.

The Value and MTE fields are left unchanged.

The logical object numbers for a module are generated at link time. By using this function, a debugger can discern the relationship between addresses and logical object numbers. Once the logical object number is known, symbols can be generated for an address via a map or symbol file, for symbolic debugging.

DBG_C_ReadCoRegs - Debug Command 14

Read Coprocessor Registers Command

Parameters

Pid = Process ID of debuggee processor
Tid = Thread ID of Coprocessor register set to read
Cmd = DBG_C_ReadCoRegs
Value = Coprocessor Type Identifier

Buffer = Pointer to Coprocessor Register Context Buffer
Len = Size of Coprocessor Register Context Buffer
Index = Reserved, must be zero

If Tid is zero and the debuggee is stopped, the register set comes from the active debuggee thread. If Tid is zero and the debuggee is executing, ERROR_INVALID_THREADID is returned.

The coprocessor type identifier is a number that identifies the format of the coprocessor register context buffer. The buffer length must correspond exactly to the requested buffer format. The supported coprocessor types, formats and lengths include the following:

For the Intel 80387 NPX processor:

Value = DBG_CO_387 = 1
Len = 108

The coprocessor register context buffer format is the same as that defined by the fsave/frestore instructions as executed by the appropriate processor.

Returns

The debugger's coprocessor register context buffer is filled in with a copy of the registers read from the appropriate coprocessor, for the thread specified in the Tid field.

If an error occurs while attempting to access the coprocessor context during this command, the DBG_N_CoError notification is returned.

Restrictions

An error is returned if any one of the following occurs:

- The debuggee process is emulating the coprocessor.
- The specified debuggee thread has not yet attempted to use the coprocessor.
- The wrong coprocessor type is used.
- Index is not zero.

DBG_C_WriteCoRegs - Debug Command 15

Write Coprocessor Registers Command

Parameters

Pid = Process ID of debuggee processor
Tid = Nonzero Thread ID of Coprocessor register set to read
Cmd = DBG_C_WriteCoRegs
Value = Coprocessor Type Identifier
Buffer = Pointer to Coprocessor Register Context Buffer
Len = Size of Coprocessor Register Context Buffer
Index = Reserved, must be zero

The coprocessor type identifier is a number that identifies the format of the coprocessor register context buffer. The buffer length must correspond exactly to the requested buffer format.

See "DBG_C_ReadCoRegs - Debug Command 14" on page D-6 for the supported coprocessor types, formats, and lengths.

The coprocessor register context buffer format is the same as that defined by the fsave/frestore instructions as executed by the appropriate processor.

Returns

The debuggee thread's coprocessor registers are filled with the values passed via the coprocessor register context buffer, for the thread specified in the Tid field.

If an error occurs while attempting to access the coprocessor context during this command, the DBG_N_CoError notification is returned.

Restrictions

An error is returned if any one of the following occurs:

- The debuggee process is emulating the coprocessor.
- The specified debuggee thread has not yet attempted to use the coprocessor.
- The wrong coprocessor type is used.
- Index is not zero.

The coprocessor may adjust some control register bits, but DosDebug will not return an error if a modification is attempted, nor will it mask the values. Because of internal coprocessor management, this adjustment may be delayed until the thread actually uses the coprocessor again.

DBG_C_ThrdStat - Debug Command 17

Get Thread Status Command

Parameters

Pid = Process ID of debuggee
Tid = Thread ID of thread of interest
Cmd = DBG_C_ThrdStat
Buffer = Pointer to Thread Status buffer
Len = Length of Thread Status buffer, in bytes. This value is 4.

If Tid is zero, the status of the debuggee thread most recently scheduled to run will be returned.

Returns

Value = Thread ID of 'next' active thread to examine
Tid = Thread ID of thread whose status is returned
Thread Status buffer filled in.

Thread Status buffer format is as follows:

```
TStat   struc
        DbgState        db      ?
        TState          db      ?
        TPriority       dw      ?
TStat   ends
```

DbgState in the Thread Status buffer contains information about the current state of debugging, and will have one of the following values upon return:

0 DBG_D_Thawed
1 DBG_D_Frozen

TState in the Thread Status buffer contains information about the scheduling state of the thread, and will have one of the following values upon return:

0 DBG_T_Runnable
1 DBG_T_Suspended
2 DBG_T_Blocked
3 DBG_T_CritSec

TPriority in the Thread Status buffer contains the thread's base scheduling priority. This priority will be expressed as scheduling class and delta values upon return.

The Value field will be filled in with the Thread ID of the 'next' thread to look at when traversing threads.

By repeatedly calling the Thread Status command, replacing the Tid with the last returned Value until a thread ID is repeated, all threads in the process can be traversed. When used in this way, the Tids returned by the Thread Status command form a loop of the debuggee's thread IDs.

DBG_C_MapROAlias - Debug Command 18 and 19

DBG_C_MapRWAlias - Debug Command 18 and 19

Map Read-Only or Read-Write Memory Alias Command

Parameters

Pid = Process ID of debuggee
Cmd = DBG_C_MapROAlias (Read Only) (*Not always supported*)
Cmd = DBG_C_MapRWAlias (Read Write)
Buffer = Reserved, must be zero.
Addr = Start of debuggee region to alias (Page-Aligned)
Len = Requested length of alias region (Page-Multiple)

Returns

Buffer = Address of the start of the debugger alias region

An alias to the debuggee's memory region of the requested length is mapped into the debugger's memory space. This region is reserved for use as an alias region until it is unmapped.

The access rights for the alias area are determined by the command number. The DBG_C_MapROAlias command maps a read-only alias region, while the DBG_C_MapRWAlias command maps a read-write alias region.

For read-write aliases, if the region is shared and read-only in the debuggee's context, a private copy of the aliased pages will be created in the debuggee's context, and dynamic RAS tracepoints will be removed from that region. This prevents debugging from affecting other areas of the system, while allowing access to shared memory areas, and proper disassembly of regions where dynamic RAS tracepoints are in use.

Because the read-write aliases may convert objects to private, using up system resources, it is recommended that read-only aliases be used when simply perusing memory. See the following **Restrictions** regarding read-only aliases on the 80386 processor.

Because the entire aliased region may map both valid and invalid regions of memory, debuggers should issue DosQueryMem just before accessing the alias region to determine if the region is valid. Debuggers should not access this region while the debuggee is executing, as portions of this region may become invalid without notifying the debugger. It is possible that no valid pages will exist in the alias region.

When the debuggee frees an aliased object, or shrinks the underlying object such that the alias would span a region outside the resultant object, an alias-free notification is returned to the debugger. This notification will be returned before the alias is invalidated. See Appendix E, DosDebug Notifications for details.

These commands may be performed while the debuggee is executing code via a Go command.

Restrictions

Because debuggers can execute code at ring 2, and the read-only bit in the page tables entries is effective only at ring 3, the read-only aliases cannot be supported. When the read-only bit becomes effective at all rings, as is expected on later processors, the read-only aliases will again be supported.

Most memory management calls may not be used on these aliases. DosQueryMem is permitted, but for interrogation only.

The passed starting addresses must be aligned on a page boundary, and the length of the aliased region must be a multiple of the page size. This restriction is due to the underlying hardware.

Aliased regions must be completely contained within a single debuggee memory object.

Unlike the DosPTrace interface for previous versions of the operating system, no interface is available for moving an alias to point to another section of debuggee memory. To move an alias, the debugger must free an existing alias, and then map a new alias to the desired region.

Aliases will only be permitted to the the user space memory region of the debuggee. No aliases will be provided to system space.

The alias region will only be provided at the linear level. No debugger LDT (Local Descriptor Table) selector will be available to access the alias region.

DBG_C_UnMapAlias - Debug Command 20

UnMap Memory Alias Command

Parameters

Pid = Process ID of debuggee
Cmd = DBG_C_UnMapAlias
Buffer = Address of the debugger alias region to unmap

Returns

The UnMapAlias command is used when the debugger has finished using an alias region. Both read-only and read-write aliases may be freed in this way.

Regions returned from other memory management calls may not be used.

The debugger may issue this command while the debuggee is executing code via a Go command.

When the debuggee process terminates, all aliases to its memory space will be invalidated. When a debugger program terminates, all aliases from its memory space will also be invalidated.

DBG_C_Connect - Debug Command 21

Connect To Debuggee Command

Parameters

Pid = Process ID of debuggee
Tid = Reserved, must be zero
Cmd = DBG_C_Connect
Value = Debugging Level Number

The only permitted debugging level number is:

1 DBG_L_386

This must be the first DosDebug command. No other DosDebug command will be accepted until the debugging connection has been established.

Returns

This command establishes a debugging connection. It must be the initial command, since it verifies the buffer format for the rest of the connection.

Because DosDebug usually cannot be ported to new machines without changing the format of the buffer, this command is needed to establish that the debugger is capable of handling the desired buffer format.

If the requested debugging level is not supported, an error is returned, and the connection is not made. This gives the debugger a chance to try again, or to automatically start a different debugger process that uses a different buffer format.

For this command, only the machine-independent portion of the buffer is examined. This portion includes the Pid, Tid, Cmd, and Value fields. This makes it possible to port the DosDebug buffer from one machine to another, without returning an error to the debugger on the initial DosDebug command.

The only DosDebug notifications that are returned by this command are DBG_N_Success and DBG_N_Error.

Restrictions

If the connection to the debuggee is not established within a reasonable ammount of time, it is assumed that the connection will never be established, and the debuggee process is terminated automatically.

The current format level may or may not be supported in future versions. This is due to the machine dependence of the DosDebug function.

DBG_C_ReadMemBuf - Debug Command 22

Read Memory Buffer Command

Parameters

Pid = Process ID of debuggee
Cmd = DBG_C_ReadMemBuf
Addr = Debuggee address to read from
Buffer = Debugger address to copy to
Len = Number of bytes to read

Returns

The number of bytes specified by Len is copied from the debuggee's user memory space starting at Addr into the debugger's Buffer.

This command is not serialized with respect to the Go command.

Restrictions

You are unable to read from any memory outside user space.

Both specified memory regions must be currently valid.

DBG_C_WriteMemBuf - Debug Command 23

Write Memory Buffer Command

Parameters

Pid = Process ID of debuggee
Cmd = DBG_C_WriteMemBuf

Addr = Debuggee address to write to
Buffer = Debugger address to copy from
Len = Number of bytes to write

Returns

The number of bytes specified by Len is copied from the debugger's Buffer into the debuggee's memory space starting at Addr.

This command is not serialized with respect to the Go command.

In the case of a write to shared read-only memory, the memory is first converted to private, and any set dynamic RAS logging points are removed from that memory, before the write is performed.

Dynamic RAS logging will continue to function in that area, in the context of other processes. The area will continue to be shared by other processes, if any.

In this way, breakpoints may be set in the debuggee without affecting the other modules.

Restrictions

You are unable to write to any memory outside user space.

Both specified memory regions must be currently valid.

DBG_C_SetWatch - Debug Command 24

Set Watchpoint Command

Parameters

Pid = Process ID of debuggee
Cmd = DBG_C_SetWatch
Addr = Starting Address of Watchpoint
Len = Length of Watchpoint, in bytes
Index = Reserved, must be zero
Value = Watchpoint Type and Scope

The Watchpoint Type and Scope is a combination of a Scope number and a Type number. Both the Scope and Type must be specified. For example, to set a local watchpoint for either read or write access, Value should be set to (DBG_W_Local + DBG_W_ReadWrite).

The Watchpoint Scopes are:

```
DBG_W_Global   equ   00000001h
DBG_W_Local    equ   00000002h
```

The Watchpoint Types are:

```
DBG_W_Execute   equ   00010000h
DBG_W_Write     equ   00020000h
DBG_W_ReadWrite equ   00030000h
```

Returns

Index = Watchpoint ID Number

This command sets a code or data watchpoint of the desired scope and type to cover the specified range of addresses.

The Watchpoint Scope controls the context in which the watchpoint is actually effective. DBG_W_Local watchpoints are effective only in the context of the debuggee process, while DBG_W_Global watchpoints are effective in the context of any process.

Both DBG_W_Local and DBG_W_Global watchpoints remain effective at interrupt time, and while executing kernel code. However, the DBG_W_Local watchpoints may miss interrupt time accesses, depending on the process context in which the interrupt occurred.

Watchpoints are disabled as soon as they are hit, so that they can only be hit once.

The resources used by a watchpoint will not be freed until the debugger is finally notified of the hit, or the debugger terminates. The debugger should use the Stop command to free resources held by any pending watchpoint hits prior to setting a watchpoint, so that these held resources will not prevent setting a new watchpoint.

DBG_W_Global watchpoints should be used sparingly, as they restrict the watchpoint resources available to all processes at once. Watchpoint resources are very limited.

Restrictions

The watchpoints are restricted by the hardware. In the case of the 80386 processor, where debug registers are used, the available watchpoint lengths are 1, 2, and 4 bytes. The 2-byte data watchpoints must be aligned on a word boundary, and the 4-byte data watchpoints must be aligned on a doubleword boundary. DBG_W_Execute watchpoints must be exactly 1 byte in length, and they must begin on an instruction boundary to be effective.

Global watchpoints are effective in v86 mode, but cannot detect DMA (direct memory access) device accesses.

Global watchpoints may be set only in the shared memory region of the linear address space. Global watchpoints will remain effective even if the underlying memory has been converted to private memory via a DosDebug memory write operation.

DBG_C_ClearWatch - Debug Command 25

Clear Watchpoint Command

Parameters

Pid = Process ID of debuggee
Cmd = DBG_C_ClearWatch
Index = Watchpoint ID Number

Returns

This command clears a currently set or hit watchpoint.

If the watchpoint is not currently set, an error is returned.

If a debugger wishes to move a watchpoint from one location to another, it should clear the old watchpoint before setting the new one, to free any resources used by currently set watchpoints.

The watchpoint will be cleared even if it is currently hit, and a notification is pending. To prevent missing the watchpoint hit in this way, you should issue the Stop command just before clearing the watchpoint, to pick up any pending watchpoint hit notifications.

DBG_C_RangeStep - Debug Command 26

Range Step Command

Parameters

Pid = Process ID of debuggee
Tid = Thread ID of thread to range-step
Cmd = DBG_C_RangeStep

Value = Linear address denoting start of range (exclusive)
Addr = Linear address denoting end of range (exclusive)

Returns

The RangeStep notification is usually returned, but any Debug notification may be returned. See Appendix E, DosDebug Notifications.

This command allows a debugger to specify a range of addresses (bounded by the linear addresses in the Value and Addr fields) through which a debuggee thread should single-step until one of the following conditions occurs:

- The debuggee thread's linear EIP (instruction pointer) is outside the range.
- The linear EIPs of consecutive debuggee threads are the same.
- Some other notification occurs.

When the RangeStep command returns, the register set is automatically updated to reflect the thread that detected the event.

Callgates that result in a privilege level transition to ring 0 will appear to range-step as a single instruction, with the range-step continuing after the function completes. This hides ring 0 execution from debuggers.

Attempting to range-step a thread that is frozen results in an error.

Restrictions

To accomplish callgate single-stepping, the single-step must be simulated because the flags (specifically, the TF bit) are not stored in the ring 0 callgate stack frame. Because of this, a range-step that results in leaving a ring 0 callgate will sometimes not execute any user-space code. The following range-step should function normally.

Range-stepping some REP instructions may not work correctly due to errors in the 80386 processor.

DBG_C_Continue - Debug Command 27

Continue Command

Parameters

Pid = Process ID of debuggee
Tid = Thread ID
Cmd = DBG_C_Continue
Value = XCPT_CONTINUE_EXECUTION, or XCPT_CONTINUE_SEARCH, or XCPT_CONTINUE_STOP

Returns

You must issue the Debug Continue command to continue after DosDebug has been given preemptive notifications or an exception notification. For such notifications, the Continue command is the only Debug command that will start the child process again. You can issue other Debug commands, but you must eventually issue the Debug Continue command.

If you issue the Debug Continue command and there is no pre-existing notification or exception, the Continue command acts like a Debug Go command.

In single-step mode, XCPT_CONTINUE_STOP has the same effect as XCPT_CONTINUE_EXECUTION. That is, execution is always stopped after a single-step operation when DBG_N_Success is returned.

Handling Preemptive Notifications

The DBG_C_Continue command is used to either continue or stop the child process after a preemptive notification has been received from DosDebug.

The XCPT_CONTINUE_STOP parameter can be used to stop the child process after a preemptive notification has been received. Any pending notifications will be held until execution of the child process is resumed using subsequent DosDebug commands. While the child process is stopped, you can issue other DosDebug commands, such as DBG_C_ReadMem.

The XCPT_CONTINUE_SEARCH parameter allows the child process to execute until the next notification is received.

The following is a list of preemptive notifications.

"DBG_N_ModuleLoad - Debug Notification no. –8" on page E-3
"DBG_N_ModuleFree - Debug Notification no. –16" on page E-6
"DBG_N_NewProc - Debug Notification no. –12" on page E-5
"DBG_N_ProcTerm - Debug Notification no. –6" on page E-2
"DBG_N_ThreadCreate - Debug Notification no. –15" on page E-6
"DBG_N_ThreadTerm - Debug Notification no. –10" on page E-4
"DBG_N_AliasFree - Debug Notification no. –13" on page E-5
"DBG_N_Exception - Debug Notification no. –7" on page E-2

Handling the DBG_N_Exception Notification

Note: XCPT_BREAKPOINT and XCPT_SINGLE_STEP are pre-first chance exception notifications.

The XCPT_CONTINUE_STOP parameter serves two purposes. It stops the child process, and it tells DosDebug that the debugger handled the exception.

The XCPT_CONTINUE_EXECUTION parameter tells DosDebug to restore the execution context of the thread that received the exception, and then continue execution of the child process. This implies that the debugger has handled the exception.

The XCPT_CONTINUE_SEARCH parameter tells DosDebug to pass the exception to the exception handler because the debugger will not handle it. After receiving an exception notification other than XCPT_BREAKPOINT or XCPT_SINGLE_STEP, the DBG_C_Continue command with the XCPT_CONTINUE_SEARCH parameter resumes execution of the child process.

DBG_C_AddrToObject - Debug Command 28

Get Memory Object Information Command

Parameters

Pid = Process ID of debuggee
Cmd = DBG_C_AddrToObject
Addr = Debuggee Linear Address

Returns

Buffer = Starting address of object
Len = Size of object in bytes
Value = Object flags
MTE = Module Table Entry handle of module if DBG_O_OBJMTE is set in the Value field object flags.

The object flags are defined as follows:

```
DBG_O_OBJMTE   equ   10000000h    ; Object is part of a module
```

Information about the memory object containing the linear address is returned. If the linear address is not part of an object, DBG_N_Error is returned with ERROR_INVALID_OBJECT in the Value field.

The Addr field will be left unchanged.

DBG_C_XchngOpcode - Debug Command 29

Exchange Opcode Command

Parameters

Pid = Process ID of debuggee
Tid = Thread ID of thread
Cmd = DBG_C_XchngOpcode
Value = Opcode 1 for Single Step
Addr = Opcode 2 for Go

Returns

The sequence of operations for this Debug command is:

1. Replace the code at the EIP (instruction pointer) with opcode 1.
2. Single-step the thread specified by the Tid field. Do not execute other threads. If the single-step operation goes into ring 0 code, consider the single-step operation complete at the first ring 0 instruction.
3. Replace the code at the original EIP with opcode 2.
4. Issue a Debug Go command on all non-frozen threads.

If an exception that DosDebug is to be notified about occurs during the single-step operation of this Debug command, opcode 2 is placed at the original EIP, and DosDebug is notified of the exception. When the debugger issues the Debug Continue command, the child process continues execution.

Note: If an exception that DosDebug is *not* to be notified about occurs, then the Debug Exchange Opcode command executes as if no exception took place.

If opcode 1 and opcode 2 are identical, this Debug command executes only operations 3 and 4 above. There is no need to single-step the thread specified by the Tid field. This would be a 'replace opcode and go' sequence using only one DosDebug function instead of two.

DBG_C_LinToSel - Debug Command 30

Translate Linear Address to Selector:Offset Command

Parameters

Pid = Process ID of debuggee
Cmd = DBG_C_LinToSel
Addr = Linear address to be translated

Returns

Value = Selector
Index = Offset

The Addr field will be left unchanged.

DBG_C_SelToLin - Debug Command 31

Translate Selector:Offset to Linear Address Command

Parameters

Pid = Process ID of debuggee
Cmd = DBG_C_SelToLin
Value = Selector from address to be translated
Index = Offset from address to be translated

Returns

Addr = Linear address

The Value and Index fields will be left unchanged.

Appendix E. DosDebug Notifications

Note: References to "IP" in the data return descriptions refer to the instruction pointer address. This is the 32-bit equivalent of the CS:EIP instruction pointer, regardless of the CS selector. This is also known as a linearized instruction pointer.

Some notifications (such as DBG_N_ModuleLoad and DBG_N_Watchpoint) may require multiple returns to the debugger. These additional pending notifications will be returned before the process being debugged can execute any more user code, and will be returned on the Go, SStep, or Stop commands.

Note that more notifications might be pending at any time, so a debugger should be ready to handle any notification at any time that a Go, SStep, or Stop command is issued.

If DosDebug returns ERROR_INTERRUPT after a command, the next notification might have been lost. If the process being debugged was executing code at that time (via a Go, SStep, or RangeStep command), it will be stopped automatically. To prevent this, DosDebug should not be used by thread 1 while signals are being used, or the debugger should issue DosEnterMustComplete before issuing the command.

DBG_N_SUCCESS - Debug Notification no. 0

Success Notification

This notification returns:

Cmd = DBG_N_Success

The DosDebug command was successful. Returned values depend on the command just executed.

DBG_N_ERROR - Debug Notification no. –1

Error Notification

This notification returns:

Cmd = DBG_N_Error
Value = Any standard error code

An error was detected while attempting a DosDebug command.

Error codes are returned from DosDebug in different ways:

1. Errors can be returned via the standard method (EAX = ERROR_CODE).

   This is only done when the DosDebug command failed to execute at all. An example of this would be ERROR_INTERRUPT when the debugger receives a signal. If DosDebug returns a nonzero value in EAX, the returned Debug buffer is invalid.

2. Errors are also returned via the DBG_N_Error notification.

   This is used when the DosDebug command was attempted, but failed due to some internal DosDebug failure. Examples of error codes returned via this interface are ERROR_INVALID_PROCID and ERROR_INVALID_DATA. Generally, these errors are detected while in the context of the debuggee process, and may include ERROR_INTERRUPT.

DBG_N_ProcTerm - Debug Notification no. –6

Process Termination Notification

This notification returns:

Cmd = DBG_N_ProcTerm
Value = Process Exit Code
Index = Process Exit Type
Addr = 0

The debuggee process is about to terminate.

The debugger is still allowed to examine the debuggee's final register and memory contents at this time. Note that when the debugger has completed this examination, it should finish terminating the debuggee process with a final Go, SStep, or Terminate command.

Value and Index contain the same information as that returned via a subsequent DosWaitChild call. The act of collecting this information does not terminate the process.

DBG_N_Exception - Debug Notification no. –7

General Exception Notification

This notification returns:

- For the pre-first chance for a breakpoint exception:

  Cmd = DBG_N_Exception
  Value = 0 (DBG_X_PRE_FIRST_CHANCE)
  Addr = Linear address of breakpoint
  Buffer = Exception number (XCPT_BREAKPOINT)

- For the pre-first chance for a single-step exception:

  Cmd = DBG_N_Exception
  Value = 0 (DBG_X_PRE_FIRST_CHANCE)
  Addr = Linear address of instruction after Single Step
  Buffer = Exception number (XCPT_SINGLE_STEP)

- For the first chance for all exceptions:

  Cmd = DBG_N_Exception
  Value = 1 (DBG_X_FIRST_CHANCE)
  Addr = Linear address of exception
  Buffer = Pointer to Exception Report Record in debuggee's context
  Len = Pointer to Exception Context Record in debuggee's context

- For the last chance for all exceptions:

  Cmd = DBG_N_Exception
  Value = 2 (DBG_X_LAST_CHANCE)
  Addr = Linear address of exception
  Buffer = Pointer to Exception Report Record in debuggee's context
  Len = Pointer to Exception Context Record in debuggee's context

- For an invalid stack notification:

  Cmd = DBG_N_Exception
  Value = 3 (DBG_X_STACK_INVALID)
  Addr = Linear address of exception
  Buffer = Exception number

The scenarios under which a debug exception is reported are pre-first, first, and last chance, and invalid stack notification. The Value field of the user debug buffer indicates the scenario.

DosDebug has detected an exception (a trap or a fault) at the specified address. The exception number in the exception structure identifies the exception that was detected.

Exception notifications are always returned from the context of the thread that detected the exception. That is, the exception structure reflects the state of the thread that caused the exception, at the time the exception was detected.

The debugger is given a maximum of two chances to handle exceptions other than single-step or breakpoint exceptions, which have a maximum of three chances. The order of operations for handling an exception is as follows:

1. Debugger has the pre-first chance to handle the exception (for breakpoint and single-step exceptions).
2. Debugger has the first chance to handle the exception, or to invoke an exception handler if it is present.
3. Debugger has the last chance to handle the exception, or to invoke an exception handler if it is present.

An exception notification is returned for all exceptions, including those raised by the user via DosRaiseException.

An exception can have an informational, warning, or fatal severity. The severity is coded in the high-order three bits of the exception number for user-raised and system exceptions.

The debugger may dismiss the exception by returning XCPT_CONTINUE_EXECUTION, so that the user's context is restored, and execution continues at the point where the exception occurred. Otherwise, the debugger may return XCPT_CONTINUE_SEARCH. This causes the exception to be passed to the user's exception handlers (after the debugger's first chance), or causes the default action for the exception to occur (after the debugger's last chance).

For performance reasons, the single-step and breakpoint exceptions cause a "pre-first" notification. This is faster than the ordinary first exception notification. At the time of the notification, the debugger may decide if the single-step or breakpoint exception was an anticipated event. If it was anticipated, the debugger may return XCPT_CONTINUE_EXECUTION, as for an ordinary first notification. If it was not anticipated, the debugger may return XCPT_CONTINUE_SEARCH in order to raise an ordinary first notification for the single-step or breakpoint exception. With the second notification, this allows a maximum of three notifications for the single-step and breakpoint exceptions.

For breakpoint exceptions, the EIP (instruction pointer) of the debuggee is decremented to point to the breakpoint instruction.

Note: Do not confuse the family of floating point exceptions with the DBG_N_CoError error notification.

Restrictions

The error code may not be reliable in some situations for the page fault exception, due to hardware errors.

DBG_N_ModuleLoad - Debug Notification no. –8

Module Load Notification

This notification returns:

Cmd = DBG_N_ModuleLoad
Value = MTE (Module Table Entry) handle of newly attached module
Addr = 0

A module has just been loaded. This occurs either at program startup, or during a call to DosLoadModule.

The newly attached module's handle is returned via Value. You can use this handle with DosQueryModuleName, or with the Debug DBG_C_NumToAddr - Debug Command 13 command, for symbolic debugging. A debugger should save these handles for future reference.

There may be many module attachments done at one time, but DosDebug is only able to communicate a single load during any one notification. In this case, the additional library load notifications become pending. The debugger should issue repeated Stop commands to be notified of these additional library loads, until Success is returned from the Stop command. If the Go, SStep, or RangeStep commands are issued instead of the Stop command, the pending notifications will be returned immediately, until there are no further notifications.

DBG_N_CoError - Debug Notification no. –9

Coprocessor Error Notification

This notification returns:

Cmd = DBG_N_CoError
Value = Any standard Error Code

An error was detected while attempting a DosDebug command that attempted to access a Coprocessor.

DBG_N_CoError is similar to the DBG_N_Error notification, but is returned only after attempting an access to the coprocessor registers.

DBG_N_CoError is returned if any one of the following occurs:

- The debuggee process is emulating the coprocessor.
- The specified debuggee thread has not yet attempted to use the coprocessor.
- The wrong coprocessor type is used.
- The requested coprocessor type is not supported or available.

This notification should not be confused with any of the floating point exception notifications.

DBG_N_ThreadTerm - Debug Notification no. –10

Thread Termination Notification

This notification returns:

Cmd = DBG_N_ThreadTerm
Value = Thread's proposed return code (from DosExit)
Addr = 0

A debuggee thread is about to terminate.

DosExitList processing has not yet been started.

The debugger is still allowed to examine the debuggee thread's final register contents at this time. When the debugger has completed this examination, it should finish terminating the debuggee thread with a final Go or SStep command.

Value contains the return code from the thread. This is only a proposed return code passed from DosExit. Only when the process actually terminates is the return code that is passed to DosWaitChild finally known.

DBG_N_AsyncStop - Debug Notification no. –11

Asynchronous Stop Notification

This notification returns:

Cmd = DBG_N_AsyncStop
Value = 0
Addr = 0

An Asynchronous Stop request has been detected.

The asynchronous stop command is used to get the attention of some debuggee thread, so that the debugger can again control the process. Because any notification results in the debuggee's coming under control of the debugger again, the asynchronous stop notification becomes redundant in that case, and is not returned.

The asynchronous stop request will be overridden and ignored if another notification can be performed instead, at the same time. An asynchronous stop notification never becomes a 'pending' notification, in the sense that a library load notification becomes pending.

DBG_N_NewProc - Debug Notification no. –12

New Process Notification

This notification returns:

Cmd = DBG_N_NewProc
Value = Process ID of the new process
Addr = 0

The debuggee process has just started a child process, and that child process needs to be debugged.

Note: This notification occurs only if descendant debugging was specified in the DosExecPgm call that started the process tree in which the debuggee is executing.

DBG_N_AliasFree - Debug Notification no. –13

Alias Free Notification

This notification returns:

Cmd = DBG_N_AliasFree
Buffer = 32-bit offset to debugger alias region
Addr = 0

An object that contains an aliased region is about to be freed by the debuggee. This can also occur if the underlying memory object is about to be shrunk such that the alias would span memory outside the resultant object.

The alias region must be unmapped before the debugger may execute the debuggee again.

The UnMapAlias command is the proper response to an alias free notification.

If a debugger creates an alias to memory in another debugger, and that memory is itself an alias to the second debugger's debuggee, then the first debugger will not receive an alias free notification when the memory of the second debugger's debuggee is freed. The alias will be freed automatically. The second debugger will receive an alias free notification.

DBG_N_Watchpoint - Debug Notification no. –14

Watchpoint Hit Notification

This notification returns:

Cmd = DBG_N_Watchpoint
Addr = Linearized instruction pointer of watchpoint hit
Value = Process ID of process that hit the watchpoint
Len = Thread ID of thread that hit the watchpoint
MTE = Module Table Entry handle of process that hit the watchpoint
Index = Watchpoint ID number

A watchpoint has been hit. The Watchpoint ID number identifies the watchpoint that was hit.

Multiple watchpoint hits become pending notifications that are returned on subsequent Stop, Go, or SStep commands. A watchpoint may be hit at any time, and more than one watchpoint may be hit at the same time.

If a watchpoint notification is pending, the resources used by the watchpoint will not be freed until the watchpoint notification is complete, or the watchpoint is cleared.

A watchpoint notification is not always returned by the same thread that caused the hit. A watchpoint may be hit by one thread, but another thread may return the notification. The thread ID of the thread that hit the watchpoint is not necessarily the value passed in the Tid field.

Data Watchpoint hits are treated as faults, rather than as traps, by the 80386 processor. Therefore, the linearized instruction pointer may not point to the exact instruction that caused the fault.

If a watchpoint is hit at interrupt time, the Value, Addr, MTE, and Len fields are all returned as zero.

DBG_N_ThreadCreate - Debug Notification no. –15

Thread Creation Notification

This notification returns:

Cmd = DBG_N_ThreadCreate
Tid = Thread ID of new thread
Addr = 0

A debuggee thread has just been created.

The new thread has not executed any user code yet.

DBG_N_ModuleFree - Debug Notification no. –16

Module Free Notification

This notification returns:

Cmd = DBG_N_ModuleFree
Value = MTE (Module Table Entry) handle of freed module
Addr = 0

A module has just been freed. This occurs either at program termination, or during execution of the DosFreeModule.

The newly attached module's handle is returned via Value. You can use this handle with DosQueryModuleName, or with the DosDebug DBG_C_NumToAddr - Debug Command 13 command, for symbolic debugging. A debugger should save these handles for future reference.

There may be many modules freed at one time, but DosDebug is only able to communicate the freeing of a single module during any one notification. In this case, the additional notifications of freed modules become pending. The debugger should issue repeated Stop commands to be notified of these additional module freeing operations, until Success is returned from the Stop command. If the Go or SStep commands are issued instead of the Stop command, the pending notifications will be returned immediately, until there are no further notifications.

DBG_N_RangeStep - Debug Notification no. –17

Range Step Notification

This notification returns:

Cmd = DBG_N_RangeStep
Addr = Linearized instruction pointer at exception
Value = Linearized instruction pointer of last user instruction executed

The debuggee stopped range-stepping because its linearized instruction pointer was outside the original range, or because the current linearized instruction pointer is equal to the linearized instruction pointer of the previous user instruction.

The DBG_N_RangeStep notification is returned independently of the DBG_N_Watchpoint notification, even though the Watchpoint fault and the RangeStep may have occurred at the exact same time.

Glossary

A

accelerator. A single key stroke that invokes an application-defined function.

accelerator table. Used to define which key strokes are treated as *accelerators* and the commands they are translated into.

access permission. All access rights that a user has regarding an object.

action. One of a set of defined tasks that a computer performs. Users request the application to perform an action in several ways, such as typing a command, pressing a function key, or selecting the action name from an action bar or menu.

action bar. The area at the top of a window that contains the choices currently available in the application program.

action point. The current position on the screen at which the pointer is pointing. (Contrast with *hot spot* and *input focus*.)

active program. A program currently running on the computer. See also *interactive program*, *noninteractive program*, and *foreground program*.

active window. The window with which the user is currently interacting.

address space. (1) The range of addresses available to a program. (2) The area of virtual storage available for a particular job.

alphanumeric video output. Output to the logical video buffer when the video adapter is in text mode and the logical video buffer is addressed by an application as a rectangular array of character cells.

anchor block. An area of Presentation Manager-internal resources allocated to a process or thread that calls WinInitialize.

anchor point. A point in a window used by a program designer or by a window manager to position a subsequently appearing window.

ANSI. American National Standards Institute.

APA. All points addressable.

API. Application programming interface. The formally-defined programming language that is between an IBM application program and the user of the program. See also *GPI*.

area. In computer graphics, a filled shape such as a solid rectangle.

ASCII. American National Standard Code for Information Interchange. A coded character set consisting of 7-bit coded characters (8 bits including parity check), used for information interchange among data processing systems, data communications systems, and associated equipment.

ASCIIZ. A string of ASCII characters that is terminated with a byte containing the value 0.

aspect ratio. In computer graphics, the width-to-height ratio of an area, symbol, or shape.

asynchronous. (1) Without regular time relationship. (2) Unexpected or unpredictable with respect to the execution of a program's instructions. See also *synchronous*.

atom. A constant that represents a string. Once a string has been defined as an atom, the atom can be used in place of the string to save space. Strings are associated with their respective atoms in an *atom table*. See also *integer atom*.

atom table. Used to relate *atoms* with the strings that they represent. Also in the table is the mechanism by which the presence of a string can be checked.

attributes. Characteristics or properties that can be controlled, usually to obtain a required appearance; for example, the color of a line. See also *graphics attributes* and *segment attributes*.

AVIO. Advanced Video Input/Output.

B

background color. The color in which the background of a graphic primitive is drawn.

background mix. An attribute that determines how the background of a graphic primitive is combined with the existing color of the graphics presentation space. Contrast with *mix*.

background program. In multiprogramming, a program that executes with a low priority. Contrast with *foreground program*.

Bézier curves. A mathematical technique of specifying smooth continuous lines and surfaces, which require a starting point and a finishing point with several intermediate points that influence or control the path of the linking curve. Named after Dr. P. Bézier.

bit map. A representation in memory of the data displayed on an APA device, usually the screen.

block. (1) A string of data elements recorded or transmitted as a unit. The elements may be characters, words, or logical records. (2) To combine two or more data elements in one block.

border. A visual indication (for example, a separator line or a background color) of the boundaries of a window.

breakpoint. (1) An instruction in a program for halting execution. Breakpoints are usually established at positions in a program where halts, caused by external

intervention, are convenient for restarting. (2) A place in a program, specified by a command or a condition, where the system halts execution and gives control to the workstation user or to a specified program.

bucket. One or more fields in which the result of an operation is kept.

buffer. (1) A portion of storage used to hold input or output data temporarily. (2) To allocate and schedule the use of buffers.

button. A mechanism on a *pointing device*, such as a mouse, used to request or initiate an action. Contrast with *pushbutton* and *radio button*.

C

cache. A high-speed buffer storage that contains frequently accessed instructions and data; it is used to reduce access time.

cached micro presentation space. A presentation space from a Presentation Manager-owned store of micro presentation spaces. It can be used for drawing to a window only, and must be returned to the store when the task is complete.

call. (1) The action of bringing a computer program, a routine, or a subroutine into effect, usually by specifying the entry conditions and jumping to an entry point. (2) To transfer control to a procedure, program, routine, or subroutine.

calling order. A sequence of instructions together with any associated data necessary to perform a call. Also known as *calling sequence*.

cancel. An action that removes the current window or menu without processing it, and returns the previous window.

CASE statement. In C, provides the body of a window procedure. There is one CASE statement for each message type written to take specific actions.

cell. See *character cell*.

CGA. Color graphics adapter.

chained list. A list in which the data elements may be dispersed but in which each data element contains information for locating the next. Synonym for *linked list*.

character. A letter, digit, or other symbol.

character box. In computer graphics, the boundary that defines, in world coordinates, the horizontal and vertical space occupied by a single character from a character set. See also *character mode*. Contrast with *character cell*.

character cell. The physical, rectangular space in which any single character is displayed on a screen or printer device. Position is addressed by row and column coordinates. Contrast with *character box*.

character code. The means of addressing a character in a character set, sometimes called *code point*.

character mode. The character mode, in conjunction with the font type, determines the extent to which graphics characters are affected by the character box, shear, and angle attributes.

check box. A control window, shaped like a square button on the screen, that can be in a checked or unchecked state. It is used to select one or more items from a list. Contrast with *radio button*.

check mark. The symbol that is used to indicate a selected item on a pull-down.

child process. A process that is loaded and started by another process. Contrast with *parent process*.

child window. A window that is positioned relative to another window (either a main window or another child window). Contrast with *parent window*.

choice. An option that can be selected. The choice can be presented as text, as a symbol (number or letter), or as an icon (a pictorial symbol).

class. See *window class*.

class style. The set of properties that apply to every window in a window class.

client area. The area in the center of a window that contains the main information of the window.

clipboard. An area of main storage that can hold data being passed from one Presentation Manager application to another. Various data formats can be stored.

clipping. In computer graphics, removing those parts of a display image that lie outside a given boundary.

clip limits. The area of the paper that can be reached by a printer or plotter.

clipping path. A clipping boundary in world-coordinate space.

CLOCK$. Character-device name reserved for the system clock.

code page. An assignment of graphic characters and control-function meanings to all code points.

code point. Synonym for *character code*.

code segment. An executable section of programming code within a load module.

color dithering. See *dithering*.

command. The name and parameters associated with an action that a program can perform.

command area. An area composed of a command field prompt and a command entry field.

command entry field. An entry field in which users type commands.

command line. On a display screen, a display line usually at the bottom of the screen, in which only commands can be entered.

command prompt. A field prompt showing the location of the command entry field in a panel.

Common Programming Interface (CPI). A consistent set of specifications for languages, commands, and calls to enable applications to be developed across all SAA environments. See also *Systems Application Architecture*.

Common User Access (CUA). A set of rules that define the way information is presented on the screen, and the techniques for the user to interact with the information.

compile. To translate a program written in a higher-level programming language into a machine language program.

COM1, COM2, COM3. Character-device names reserved for serial ports 1 through 3.

CON. Character-device name reserved for the console keyboard and screen.

contiguous. Touching or joining at a common edge or boundary, for example, an unbroken consecutive series of storage locations.

control. The means by which an operator gives input to an application. A *choice* corresponds to a control.

Control Panel. In the Presentation Manager, a program used to set up user preferences that act globally across the system.

Control Program. The basic function of OS/2, including DOS emulation and the support for keyboard, mouse, and video input/output.

control window. A class of window used to handle a specific kind of user interaction. Radio buttons and check boxes are examples.

correlation. The action of determining which element or object within a picture is at a given position on the display. This follows a *pick* operation.

CPI. Common Programming Interface.

critical extended attribute. An extended attribute that is necessary for the correct operation of the system or a particular application.

CUA. Common User Access.

current position. The point from which the next primitive will be drawn.

cursor. A symbol displayed on the screen and associated with an input device. The cursor indicates where input from the device will be placed. Types of cursors include text cursors, graphics cursors, and selection cursors. Contrast with *pointer* and *input focus*.

D

data structure. (ISO) The syntactic structure of symbolic expressions and their storage-allocation characteristics.

DBCS. See *double-byte character set*.

deadlock. (1) Unresolved contention for the use of a resource. (2) An error condition in which processing cannot continue because each of two elements of the process is waiting for an action by, or a response from, the other. (3) An impasse that occurs when multiple processes are waiting for the availability of a resource that will not become available because it is being held by another process that is in a similar wait state.

debug. To detect, diagnose, and eliminate errors in programs.

decipoint. In printing, one tenth of a point. There are 72 points in an inch.

default procedure. Function provided by the Presentation Interface that may be used to process standard messages from dialogs or windows.

default value. A value used when no value is explicitly specified by the user. For example, in the graphics programming interface, the default line-type is 'solid'.

descendant. A process or session that is loaded and started by a parent process or parent session.

Desktop Manager. In the Presentation Manager, a window that displays a list of groups of programs, each of which can be started or stopped.

desktop window. The window, corresponding to the physical device, against which all other types of windows are established.

device context. A logical description of a data destination such as memory, metafile, display, printer, or plotter. See also *direct device context*, *information device context*, *memory device context*, *metafile device context*, *queued device context*, and *screen device context*.

device driver. A file that contains the code needed to attach and use a device such as a display, printer, or plotter.

device space. Coordinate space in which graphics are assembled after all GPI transformations have been applied. Device space is defined in device-specific units.

dialog. The interchange of information between a computer and its user through a sequence of requests by the user and the presentation of responses by the computer.

dialog box. A type of window that contains one or more controls for the formatted display and entry of data. Also known as a *pop-up window*. A modal dialog box is used to implement a pop-up window.

Dialog Box Editor. A *WYSIWYG* editor that creates dialog boxes for communicating with the application user.

dialog item. A component (for example, a menu or a button) of a dialog box. Dialog items are also used when creating dialog templates.

dialog tag language. A markup language used by the DTL compiler to create dialog objects.

dialog template. The definition of a dialog box, which contains details of its position, appearance, and window ID, and the window ID of each of its child windows.

direct device context. A logical description of a data destination that is a device other than the screen (for example, a printer or plotter), and where the output is not to go through the spooler. Its purpose is to satisfy queries. See also *device context*.

direct manipulation. The action of using the mouse to move objects around the screen. For example, moving files and directories around in the *File Manager*.

direct memory access (DMA). The transfer of data between main storage and input/output devices without intervention by the processor.

directory. A type of file containing the names and controlling information for other files or other directories.

display point. Synonym for *pel*.

dithering. The process used in color displays whereby every other pel is set to one color, and the intermediate pels are set to another. Together they produce the effect of a third color at normal viewing distances. This process can only be used on solid areas of color; it does not work on narrow lines, for example.

DMA. Direct memory access.

double-byte character set (DBCS). A set of characters in which each character is represented by two bytes. Languages such as Japanese, Chinese, and Korean, which contain more characters than can be represented by 256 code points, require double-byte character sets. Since each character requires two bytes, the entering, displaying, and printing of DBCS characters requires hardware and software that can support DBCS.

doubleword. A contiguous sequence of bits or characters that comprises two computer words and is capable of being addressed as a unit.

dragging. In computer graphics, moving an object on the display screen as if it were attached to the pointer.

drawing chain. See *segment chain*.

drop. To fix the position of an object that is being dragged, by releasing the select button of the pointing device.

DTL. See *dialog tag language*.

dual-boot function. A feature of OS/2 that allows the user to start DOS from within OS/2, or OS/2 from within DOS.

duplex. Pertaining to communication in which data can be sent and received at the same time. Synonymous with *full duplex*.

dynamic linking. The process of resolving external references in a program module at load time or run time rather than during linking.

dynamic-link library. A collection of executable programming code and data that is bound to an application at load time or run time, rather than during linking. The programming code and data in a dynamic link library can be shared by several applications simultaneously.

dynamic-link module. A module that is linked at load time or run time.

dynamic segments. Graphics segments drawn in exclusive-OR mix mode so that they can be moved from one screen position to another without affecting the rest of the displayed picture.

dynamic storage. (1) A device that stores data in a manner that permits the data to move or vary with time such that the specified data is not always available for recovery. (2) A storage in which the cells require repetitive application of control signals in order to retain stored data. Such repetitive application of the control signals is called a refresh operation. A dynamic storage may use static addressing or sensing circuits. (3) See also *static storage*.

E

EBCDIC. Extended binary-coded decimal interchange code. A coded character set consisting of 8-bit coded characters (9 bits including parity check), used for information interchange among data processing systems, data communications systems, and associated equipment.

EGA. Extended graphics adapter.

8.3 file-name format. A file-naming convention in which file names are limited to eight characters before and three characters after a single dot. Usually pronounced "eight-dot-three." See also *non-8.3 file-name format*.

element. An entry in a graphics segment that comprises one or more graphics orders and that is addressed by the element pointer.

entry field. An area on the screen, usually highlighted in some manner, in which users type information.

entry-field control. The means by which the application receives data entered by the user in an entry field. When it has the input focus, it displays a flashing pointer at the position where the next typed character will go.

entry panel. A defined panel type containing one or more entry fields and protected information such as headings, prompts, and explanatory text.

exception. An abnormal condition such as an I/O error encountered in processing a data set or a file.

exclusive system semaphore. A system semaphore that can be modified only by threads within the same process.

exit. The action that terminates the current function and returns the user to a higher level function. Repeated exit requests return the user to the point from which all functions provided to the system are accessible. Contrast with *cancel*.

extended attribute. An additional piece of information about a file object, such as its data format or category. It consists of a name and a value. A file object may have more than one extended attribute associated with it.

extended-choice selection. A mode that allows the user to select more than one item from a window. Not all

windows allow extended choice selection. Contrast with *multiple-choice selection*.

extended help. A facility that provides users with information about an entire application panel rather than a particular item on the panel.

extent. Continuous space on a disk or diskette that is occupied by or reserved for a particular data set, data space, or file.

F

family-mode application. An application program that can run in the OS/2 environment and in the DOS environment. However, it cannot take advantage of many of the OS/2-mode facilities, such as multitasking, interprocess communication, and dynamic linking.

FAT. File allocation table.

FEA. Full extended attribute.

field-level help. Information specific to the field on which the cursor is positioned. This help function is "contextual" because it provides information about a specific item as it is currently used; the information is dependent upon the context within the work session.

file. A named set of records stored or processed as a unit.

file allocation table (FAT). In IBM personal computers, a table used by the operating system to allocate space on a disk for a file, and to locate and chain together parts of the file that may be scattered on different sectors so that the file can be used in a random or sequential manner.

file attribute. Any of the attributes that describe the characteristics of a file.

File Manager. In the Presentation Manager, a program that displays directories and files, and allows various actions on them.

file specification. The full identifier for a file, which includes its drive designation, path, file name, and extension.

file system driver (FSD). A program that manages file I/O and controls the format of information on the storage media.

fillet. A curve that is tangential to the end points of two adjoining lines. See also *polyfillet*.

flag. (1) An indicator or parameter that shows the setting of a switch. (2) A character that signals the occurrence of some condition, such as the end of a word.

focus. See *input focus*.

font. A particular size and style of typeface that contains definitions of character sets, marker sets, and pattern sets.

foreground program. The program with which the user is currently interacting. Also known as *interactive program*. Contrast with *background program*.

frame. The part of a window that can contain several different visual elements specified by the application, but drawn and controlled by the Presentation Manager. The frame encloses the client area.

frame styles. Different standard window layouts provided by the Presentation Manager.

FSD. File system driver.

full duplex. Synonym for *duplex*.

full-screen application. An application program that occupies the whole screen.

function. (1) In a programming language, a block, with or without formal parameters, whose execution is invoked by means of a call. (2) A set of related control statements that cause one or more programs to be performed.

function key. A key that causes a specified sequence of operations to be performed when it is pressed, for example, F1 and Alt-K.

function key area. The area at the bottom of a window that contains function key assignments such as F1=Help.

G

GDT. Global Descriptor Table.

general protection fault. An exception condition that occurs when a process attempts to use storage or a module that has some level of protection assigned to it, such as I/O privilege level. See also *IOPL code segment*.

Global Descriptor Table (GDT). Defines code and data segments available to all tasks in an application.

global dynamic-link module. A dynamic-link module that can be shared by all processes in the system that refer to the module name.

global file-name character. A special character used to refer to a set of file objects with a common base name. The asterisk (*) and question mark (?) are used as global file-name characters. For example, *.EXE can be used to refer to a set of files with the extension EXE.

glyph. A graphic symbol whose appearance conveys information.

GPI. Graphics programming interface. The formally-defined programming language that is between an IBM graphics program and the user of the program. See also *API*.

graphics. A picture defined in terms of graphic primitives and graphics attributes.

graphics attributes. Attributes that apply to graphic primitives. Examples are color, line type, and shading-pattern definition. See also *segment attributes*.

graphics field. The clipping boundary that defines the visible part of the presentation-page contents.

graphics model space. The conceptual coordinate space in which a picture is constructed after any model

transforms have been applied. Also known as *model space*.

graphic primitive. A single item of drawn graphics, such as a line, arc, or graphics text string. See also *graphics segment*.

graphics segment. A sequence of related graphic primitives and graphics attributes. See also *graphic primitive*.

graying. The indication that a choice on a pull-down is unavailable.

group. A collection of logically-connected controls. For example, the buttons controlling paper size for a printer. See also *program group*.

H

handle. An identifier that represents an object, such as a device or window, to the Presentation Interface.

hard error. An error condition on a network that requires either that the system be reconfigured, or that the source of the error be removed before the system can resume reliable operation.

header. (1) System-defined control information that precedes user data. (2) The portion of a message that contains control information for the message, such as one or more destination fields, name of the originating station, input sequence number, character string indicating the type of message, and priority level for the message.

help. A function that provides information about a specific field, an application panel, or information about the help facility.

help index. A facility that allows the user to select topics for which help is available.

help panel. A panel with information to assist users that is displayed in response to a help request from the user.

help window. A Common User Access-defined secondary window that displays information when the user requests help.

heap. An area of free storage available for dynamic allocation by an application. Its size varies according to the storage requirements of the application.

hit testing. The means of identifying which window is associated with which input device event.

hook. A mechanism by which procedures are called when certain events occur in the system. For example, the filtering of mouse and keyboard input before it is received by an application program.

hook chain. A sequence of hook procedures that are "chained" together so that each event is passed, in turn, to each procedure in the chain.

hot spot. The part of the pointer that must touch an object before it can be selected. This is usually the tip of the pointer. Contrast with *action point*.

I

icon. A pictorial representation of an item the user can select. Icons can represent items (such as a document file) that the user wants to work on, and actions that the user wants to perform. In the Presentation Manager, icons are used for data objects, system actions, and minimized programs.

icon area. In the Presentation Manager, the area at the bottom of the screen that is normally used to display the icons for minimized programs.

Icon Editor. The Presentation Manager-provided tool for creating icons.

image font. A set of symbols, each of which is described in a rectangular array of pels. Some of the pels in the array are set to produce the image of the symbol. Contrast with *outline font*.

information device context. A logical description of a data destination other than the screen (for example, a printer or plotter), but where no output will occur. Its purpose is to satisfy queries. See also *device context*.

information panel. A defined panel type characterized by a body containing only protected information.

input focus. The area of the screen that will receive input from an input device (typically the keyboard).

input router. An internal OS/2 process that removes messages from the system queue.

integer atom. A special kind of *atom* that represents a predefined system constant and carries no storage overhead. For example, names of window classes provided by Presentation Manager are expressed as integer atoms.

interactive graphics. Graphics that can be moved or manipulated by a user at a terminal.

interactive program. A program that is running (active) and is ready to receive (or is receiving) input from the user. Compare with *active program* and contrast with *noninteractive program*.

Also known as a *foreground program*.

interchange file. Data that can be sent from one Presentation Interface application to another.

interval timer. (1) A timer that provides program interruptions on a program-controlled basis. (2) An electronic counter that counts intervals of time under program control.

IOCtl. A device-specific command that requests a function of a device driver through the *DosDevIOCtl* function.

I/O operation. An input operation to, or output operation from a device attached to a computer.

IOPL. Input/output privilege level.

IOPL code segment. An IOPL executable section of programming code that enables an application to directly

manipulate hardware interrupts and ports without replacing the device driver. See also *privilege level*.

J

journal. A special-purpose file that is used to record changes made in the system.

K

Kanji. A graphic character set used in Japanese ideographic alphabets.

KBD$. Character-device name reserved for the keyboard.

kernel. The part of an operating system that performs basic functions, such as allocating hardware resources.

kerning. The design of graphics characters so that their character boxes overlap. Used to space text proportionally.

keys help. A facility that gives users a listing of all the key assignments for the current application.

L

label. In a graphics segment, an identifier of one or more elements that is used when editing the segment.

language support procedure. Function provided by the Presentation Interface for applications that do not, or cannot (as in the case of COBOL and FORTRAN programs), provide their own dialog or window procedures.

LDT. Local Descriptor Table.

LIFO stack. A data stack from which data is retrieved in last-in, first-out order.

linked list. Synonym for *chained list*.

list box. A control window containing a vertical list of selectable descriptions.

list panel. A defined panel type that displays a list of items from which users can select one or more choices and then specify one or more actions to work on those choices.

load-on-call. A function of a linkage editor that allows selected segments of the module to be disk resident while other segments are executing. Disk resident segments are loaded for execution and given control when any entry point that they contain is called.

load time. The point in time at which a program module is loaded into main storage for execution.

local area network (LAN). A data network located on the user's premises in which serial transmission is used for direct data communication among data stations.

Local Descriptor Table (LDT). Defines code and data segments specific to a single task.

lock. A serialization mechanism by means of which a resource is restricted for use by the holder of the lock.

LPT1, LPT2, LPT3. Character-device names reserved for parallel printers 1 through 3.

M

main window. The window that is positioned relative to the *desktop window*.

map. (1) A set of values having a defined correspondence with the quantities or values of another set. (2) To establish a set of values having a defined correspondence with the quantities or values of another set.

marker box. In computer graphics, the boundary that defines, in world coordinates, the horizontal and vertical space occupied by a single marker from a marker set.

marker symbol. A symbol centered on a point. Graphs and charts can use marker symbols to indicate the plotted points.

maximize. A window-sizing action that makes the window the largest size possible.

media window. The part of the physical device (display, printer, or plotter) on which a picture is presented.

memory device context. A logical description of a data destination that is a memory bit map. See also *device context*.

memory management. A feature of the operating system for allocating, sharing, and freeing main storage.

menu. A type of panel that consists of one or more selection fields. Also called a *menu panel*.

message. (1) In the Presentation Manager, a packet of data used for communication between the Presentation Interface and windowed applications. (2) In a user interface, information not requested by users but presented to users by the computer in response to a user action or internal process.

message filter. The means of selecting which messages from a specific window will be handled by the application.

message queue. A sequenced collection of messages to be read by the application.

metafile. The generic name for the definition of the contents of a picture. Metafiles are used to allow pictures to be used by other applications.

metafile device context. A logical description of a data destination that is a metafile, which is used for graphics interchange. See also *device context*.

metalanguage. A language used to specify another language. For example, data types can be described using a metalanguage so as to make the descriptions independent of any one computer language.

mickey. A unit of measurement for physical mouse motion whose value depends on the mouse device driver currently loaded.

micro presentation space. A graphics presentation space in which a restricted set of the GPI function calls is available.

minimize. A window-sizing action that makes the window the smallest size possible. In the Presentation Manager, minimized windows are represented by icons.

mix. An attribute that determines how the foreground of a graphic primitive is combined with the existing color of graphics output. Also known as *foreground mix*. Contrast with *background mix*.

mixed character string. A string containing a mixture of one-byte and *Kanji* or Hangeul (two-byte) characters.

mnemonic. A method of selecting an item on a pull-down by means of typing the highlighted letter in the menu item.

modal dialog box. The type of control that allows the operator to perform input operations on only the current dialog box or one of its child windows. Also known as a *serial dialog box*. Contrast with *parallel dialog box*.

modeless dialog box. The type of control that allows the operator to perform input operations on any of the application's windows. Also known as a *parallel dialog box*. Contrast with *modal dialog box*.

model space. See *graphics model space*.

module definition file. A file that describes the code segments within a load module. For example, it indicates whether a code segment is loadable before module execution begins (preload), or loadable only when referred to at run time (load-on-call).

mouse. A hand-held device that is moved around to position the pointer on the screen.

MOUSE$. Character-device name reserved for a mouse.

multiple-choice selection. A mode that allows users to select any number of choices, including none at all. See also *check box*. Contrast with *extended-choice selection*.

multitasking. The concurrent processing of applications or parts of applications. A running application and its data are protected from other concurrently running applications.

N

named pipe. A named buffer that provides client-to-server, server-to-client, or full duplex communication between unrelated processes. Contrast with *unnamed pipe*.

noncritical extended attribute. An extended attribute that is not necessary for the function of an application.

nondestructive read. A read process that does not erase the data in the source location.

non-8.3 file-name format. A file-naming convention in which path names can consist of up to 255 characters. See also *8.3 file-name format*.

noninteractive program. A program that is running (active) but is not ready to receive input from the user. Compare with *active program*, and contrast with *interactive program*.

nonretained graphics. Graphic primitives that are not remembered by the Presentation Interface once they have been drawn. Contrast with *retained graphics*.

NUL. Character-device name reserved for a nonexistent (dummy) device.

null-terminated string. A string of (n + 1) characters where the (n + 1)th character is the 'null' character (X'00'), and is used to represent an n-character string with implicit length. Also known as 'zero-terminated' string and 'ASCIIZ' string.

O

object window. A window that does not have a parent, but which may have child windows. An object window cannot be presented on a device.

open. To start working with a file, directory, or other object.

outline font. A set of symbols, each of which is created as a series of lines and curves. Synonymous with *vector font*. Contrast with *image font*.

output area. The area of the output device within which the picture is to be displayed, printed, or plotted.

owner window. A window into which specific events that occur in another (owned) window are reported.

owning process. The process that owns the resources that may be shared with other processes.

P

page. A 4KB segment of contiguous physical memory.

page viewport. A boundary in device coordinates that defines the area of the output device in which graphics are to be displayed. The presentation-page contents are transformed automatically to the page viewport in device space.

paint. The action of drawing or redrawing the contents of a window.

panel. A particular arrangement of information grouped together for presentation to the user in a window.

panel area. An area within a panel that contains related information. The three major Common User Access-defined panel areas are the action bar, the function key area, and the panel body.

panel body. The portion of a panel not occupied by the action bar, function key area, title or scroll bars. The panel body may contain protected information, selection fields, and entry fields. The layout and content of the panel body determine the panel type.

panel body area. The part of a window not occupied by the action bar or function key area. The panel body area

may contain information, selection fields, and entry fields. Also known as *client area*.

panel body area separator. A line or color boundary that provides users with a visual distinction between two adjacent areas of a panel.

panel definition. A description of the contents and characteristics of a panel. A panel definition is the application developer's mechanism for predefining the format to be presented to users in a window.

panel ID. A panel element located in the upper left-hand corner of a panel body that identifies that particular panel within the application.

panel title. A panel element that identifies the information in the panel.

paper size. The size of paper, defined in either standard U.S. or European names (for example, A, B, A4), and measured in inches or millimeters respectively.

parallel dialog box. See *modeless dialog box*.

parent process. A process that loads and starts other processes. Contrast with *child process*.

parent window. The window relative to which one or more child windows are positioned. Contrast with *child window*.

partition. (1) A fixed-size division of storage. (2) On an IBM personal computer fixed disk, one of four possible storage areas of variable size; one may be accessed by DOS, and each of the others may be assigned to another operating system.

path. The part of a file specification that lists a series of directory names. Each directory name is separated by the backslash character. In the file specification C:\MYFILES\MISC\GLOSSARY.SCR, the path consists of MYFILES\MISC\.

pel. The smallest area of a display screen capable of being addressed and switched between visible and invisible states. Synonym for *display point*, *pixel*, and *picture element*.

pick. To select part of a displayed object using the pointer.

picture chain. See *segment chain*.

picture element. Synonym for *pel*.

PID. Process identification.

pipe. A named or unnamed buffer used to pass data between processes. A process reads from or writes to a pipe as if the pipe were a standard-input or standard-output file. See also *named pipe* and *unnamed pipe*.

pixel. Synonym for *pel*.

plotter. An output device that uses pens to draw its output on paper or on transparency foils.

PM. Presentation Manager.

pointer. (1) The symbol displayed on the screen that is moved by a pointing device, such as a *mouse*. The pointer is used to point at items that users can select. Contrast with *cursor*. (2) A data element that indicates the location of another data element.

POINTER$. Character-device name reserved for a pointer device (mouse screen support).

pointing device. A device (such as a mouse) used to move a pointer on the screen.

pointings. Pairs of x-y coordinates produced by an operator defining positions on a screen with a pointing device, such as a *mouse*.

polyfillet. A curve based on a sequence of lines. It is tangential to the end points of the first and last lines, and tangential also to the midpoints of all other lines. See also *fillet*.

polyline. A sequence of adjoining lines.

pop. To retrieve an item from a last-in-first-out stack of items. Contrast with *push*.

pop-up window. A window that appears on top of another window in a dialog. Each pop-up window must be completed before returning to the underlying window.

Presentation Manager (PM). The visual component of OS/2 that presents, in windows, a graphics-based interface to applications and files installed and running in OS/2.

presentation page. The coordinate space in which a picture is assembled for display.

presentation space (PS). Contains the device-independent definition of a picture.

primary window. The window in which the main dialog between the user and the application takes place. In a multiprogramming environment, each application starts in its own primary window. The primary window remains for the duration of the application, although the panel displayed will change as the user's dialog moves forward. See also *secondary window*.

primitive. See *graphic primitive*.

primitive attribute. A specifiable characteristic of a graphic primitive. See *graphics attributes*.

print job. The result of sending a document or picture to be printed.

Print Manager. In the Presentation Manager, the part of the spooler that manages the spooling process. It also allows users to view print queues and to manipulate print jobs.

privilege level. A protection level imposed by the hardware architecture of the IBM personal computer. There are four privilege levels (number 0 through 3). Only certain types of programs are allowed to execute at each privilege level. See also *IOPL code segment*.

procedure call. In programming languages, a language construct for invoking execution of a procedure.

process. An instance of an executing application and the resources it is using.

program details. Information about a program that is specified in the *Program Manager* window and is used when the program is started.

program group. In the Presentation Manager, several programs that can be acted upon as a single entity.

program name. The full file specification of a program. Contrast with *program title*.

program title. The name of a program as it is listed in the *Program Manager* window. Contrast with *program name*.

prompt. A displayed symbol or message that requests input from the user or gives operational information. The user must respond to the prompt in order to proceed.

protocol. A set of semantic and syntatic rules that determines the behavior of functional units in achieving communication.

pseudocode. An artificial language used to describe computer program algorithms without using the syntax of any particular programming language.

pull-down. An *action bar* extension that displays a list of choices available for a selected action bar choice. After users select an action bar choice, the pull-down appears with the list of choices. Additional *pop-up windows* may appear from pull-down choices to further extend the actions available to users.

push. To add an item to a last-in-first-out stack of items. Contrast with *pop*.

pushbutton. A control window, shaped like a round-cornered rectangle and containing text, that invokes an immediate action, such as 'enter' or 'cancel'.

Q

queue. A linked list of elements waiting to be processed. For example, a queue may be a list of print jobs waiting to be printed.

queued device context. A logical description of a data destination (for example, a printer or plotter) where the output is to go through the spooler. See also *device context*.

R

radio button. A control window, shaped like a round button on the screen, that can be in a checked or unchecked state. It is used to select a single item from list. Contrast with *check box*.

RAS. Reliability, availability, and serviceability.

raster. (1) In computer graphics, a predetermined pattern of lines that provides uniform coverage of a display space. (2) The coordinate grid that divides the display area of a display device.

read-only file. A file that may be read from but not written to.

realize. To cause the system to ensure, wherever possible, that the physical color table of a device is set to the closest possible match in the logical color table.

recursive routine. A routine that can call itself or be called by another routine called by the recursive routine.

reentrant. The attribute of a program or routine that allows the same copy of the program or routine to be used concurrently by two or more tasks.

reference phrase. A word or phrase that is emphasized in a device-dependent manner to inform the user that additional information for the word or phrase is available.

reference phrase help. Provides help information for a selectable word or phrase.

refresh. To update a window, with changed information, to its current status.

region. A clipping boundary in device space.

register. A storage device having a specified storage capacity such as a bit, byte, or computer word, and usually intended for a special purpose.

remote file system. A file-system driver that gains access to a remote system without a block device driver.

resource. The means of providing extra information used in the definition of a window. A resource can contain definitions of fonts, templates, accelerators, and mnemonics; the definitions are held in a resource file.

resource file. A file containing information used in the definition of a window. Definitions can be of fonts, templates, accelerators, and mnemonics.

restore. To return a window to its original size or position following a sizing or moving action.

retained graphics. Graphic primitives that are remembered by the Presentation Interface after they have been drawn. Contrast with *nonretained graphics*.

return code. (1) A code used to influence the execution of succeeding instructions. (2) A value returned to a program to indicate the results of an operation requested by that program.

reverse video. A form of alphanumeric highlighting for a character, field, or cursor, in which its color is exchanged with that of its background. For example, changing a red character on a black background to a black character on a red background.

RGB. Red-green-blue. For example, "RGB display".

roman. Relating to a type style with upright characters.

root segment. In a hierarchical database, the highest segment in the tree structure.

run time. (1) Any instant at which a program is being executed. (2) The time during which an instruction in an instruction register is decoded and performed.

S

SAA. Systems Application Architecture.

scheduler. A computer program designed to perform functions such as scheduling, initiation, and termination of jobs.

screen. The physical surface of a work station or terminal upon which information is presented to users.

screen device context. A logical description of a data destination that is a particular window on the screen. See also *device context.*

SCREEN$. Character-device name reserved for the display screen.

scroll bar. A control window, horizontally or vertically aligned, that allows the user to scroll additional data into an associated panel area.

scrollable entry field. An entry field larger than the visible field.

scrollable selection field. A selection field that contains more choices than are visible.

scrolling. Moving a display image vertically or horizontally in a manner such that new data appears at one edge, as existing data disappears at the opposite edge.

secondary window. A type of window associated with the primary window in a dialog. A secondary window begins a secondary and parallel dialog that runs at the same time as the primary dialog.

sector. An addressable subdivision of a track used to record one block of program code or data on a disk or diskette.

segment. See *graphics segment.*

segment attributes. Attributes that apply to the segment as an entity, as opposed to the individual primitives within the segment. For example, the visibility or detectability of a segment.

segment chain. All segments in a graphics presentation space that are defined with the 'chained' attribute. Synonym for *picture chain.*

segment priority. The order in which segments are drawn.

segment store. An area in a normal graphics presentation space where retained graphics segments are stored.

select. To mark or choose an item. Note that *select* means to mark or type in a choice on the screen; *enter* means to send all selected choices to the computer for processing.

select button. The button on a pointing device, such as a mouse, that is pressed to select a menu choice. Also known as button 1.

selection cursor. A type of cursor used to indicate the choice or entry field users want to interact with. It is represented by highlighting the item that it is currently positioned on.

selection field. A field containing a list of choices from which the user can select one or more.

semaphore. An object used by multi-threaded applications for signalling purposes and for controlling access to serially reusable resources.

separator. See *panel body area separator.*

serial dialog box. See *modal dialog box.*

serialization. The consecutive ordering of items.

serialize. To ensure that one or more events occur in a specified sequence.

serially reusable resource (SRR). A logical resource or object that can be accessed by only one task at a time.

session. A routing mechanism for user interaction via the console; a complete environment that determines how an application runs and how users interact with the application. OS/2 can manage more than one session at a time, and more than one process can run in a session. Each session has its own set of environment variables that determine where OS/2 looks for dynamic-link libraries and other important files.

shadow box. The area on the screen that follows mouse movements and shows what shape the window will take if the mouse button is released.

shared data. Data that is used by two or more programs.

shared memory. Memory that is used by two or more programs.

shear. The tilt of graphics text when each character leans to the left or right while retaining a horizontal baseline.

shell. (1) A software interface between a user and the operating system of a computer. Shell programs interpret commands and user interactions on devices such as keyboards, pointing devices, and touch-sensitive screens, and communicate them to the operating system. (2) Software that allows a kernel program to run under different operating-system environments.

Shutdown. The procedure required before the computer is switched off to ensure that data is not lost.

sibling processes. Child processes that have the same parent process.

sibling windows. Child windows that have the same parent window.

slider box. An area on the scroll bar that indicates the size and position of the visible information in a panel area in relation to the information available. Also known as *thumb mark.*

source file. A file that contains source statements for items such as high-level language programs and data description specifications.

source statement. A statement written in a programming language.

specific dynamic-link module. A dynamic-link module created for the exclusive use of an application.

spline. A sequence of one or more Bézier curves.

spooler. A program that intercepts the data going to printer devices and writes it to disk. The data is printed or plotted when it is complete, and the required device is available. The spooler prevents output from different sources from being intermixed.

stack. A list constructed and maintained so that the next data element to be retrieved is the most recently stored. This method is characterized as last-in-first-out (LIFO).

standard window. A collection of window elements that form a panel. The standard window can include one or more of the following window elements: sizing borders, system menu icon, title bar, maximize/minimize/restore icons, action bar and pull-downs, scroll bars, and client area.

static control. The means by which the application presents descriptive information (for example, headings and descriptors) to the user. The user cannot change this information.

static storage. (1) A read/write storage unit in which data is retained in the absence of control signals. Static storage may use dynamic addressing or sensing circuits. (2) Storage other than *dynamic storage*.

style. See *window style*.

suballocation. The allocation of a part of one extent for occupancy by elements of a component other than the one occupying the remainder of the extent.

subdirectory. In an IBM personal computer, a file referred to in a root directory that contains the names of other files stored on the diskette or fixed disk.

swapping. (1) A process that interchanges the contents of an area of real storage with the contents of an area in auxiliary storage. (2) In a system with virtual storage, a paging technique that writes the active pages of a job to auxiliary storage and reads pages of another job from auxiliary storage into real storage. (3) The process of temporarily removing an active job from main storage, saving it on disk, and processing another job in the area of main storage formerly occupied by the first job.

switch. (1) An action that moves the input focus from one area to another. This can be within the same window or from one window to another. (2) In a computer program, a conditional instruction and an indicator to be interrogated by that instruction. (3) A device or programming technique for making a selection, for example, a toggle, a conditional jump.

switch list. See *Task List*.

symbolic identifier. A text string that equates to an integer value in an include file, that is used to identify a programming object.

synchronous. Pertaining to events or operations that are predictable or occur at the same time. See also *asynchronous*.

System Menu. In the Presentation Manager, the pull-down in the top left corner of a window that allows it to be moved and sized with the keyboard.

system queue. This is the master queue for all pointer device or keyboard events.

Systems Application Architecture (SAA). A formal set of rules that enables applications to be run without modification in different computer environments.

T

tag. One or more characters attached to a set of data that defines the formatting or other characteristics of the set, including its definition.

Task List. In the Presentation Manager, the list of programs that are active. The list can be used to switch to a program and to stop programs.

template. An ASCII-text definition of an action bar and pull-down menu, held in a resource file, or as a data structure in program memory.

text. Characters or symbols.

text cursor. A symbol displayed in an entry field that indicates where typed input will appear.

text window. Also known as the VIO window.

text-windowed application. The environment in which the operating system perfoms advanced-video input and output operations.

thread. A unit of execution within a process. It uses the resources of the process.

thumb mark. The portion of the scroll bar that describes the range and properties of the data that is currently visible in a window. Also known as a *slider box*.

tilde. A mark used to denote the character that is to be used as a mnemonic when selecting text items within a menu.

time slice. (1) An interval of time on the processing unit allocated for use in performing a task. After the interval has expired, processing-unit time is allocated to another task, so a task cannot monopolize processing-unit time beyond a fixed limit. (2) In systems with time sharing, a segment of time allocated to a terminal job.

title bar. The area at the top of a window that contains the window title. The title bar is highlighted when that window has the input focus. Contrast with *panel title*.

transaction. An exchange between a workstation and another device that accomplishes a particular action or result.

transform. (1) The action of modifying a picture by scaling, shearing, reflecting, rotating, or translating. (2) The object that performs or defines such a modification; also referred to as a *transformation*.

Tree. In the Presentation Manager, the window in the *File Manager* that shows the organization of drives and directories.

truncate. (1) To end a computational process in accordance with some rule. (2) To remove the beginning or ending elements of a string. (3) To drop data that cannot be printed or displayed in the line width specified or available. (4) To shorten a field or statement to a specified length.

U

unnamed pipe. A circular buffer, created in memory, used by related processes to communicate with one another. Contrast with *named pipe*.

update region. A system-provided area of dynamic storage containing one or more (not necessarily contiguous) rectangular areas of a window, that are visually invalid or incorrect, and therefore in need of repainting.

user interface. Hardware, software, or both that allows a user to interact with and perform operations on a system, program, or device.

User Shell. A component of OS/2 that uses a graphics-based, windowed interface to allow the user to manage applications and files installed and running under OS/2.

utility program. (1) A computer program in general support of computer processes; for example, a diagnostic program, a trace program, a sort program. (2) A program designed to perform an everyday task such as copying data from one storage device to another.

V

vector font. A set of symbols, each of which is created as a series of lines and curves. Synonymous with *outline font*. Contrast with *image font*.

VGA. Video graphics array.

viewing pipeline. The series of transformations applied to a graphic object to map the object to the device on which it is to be presented.

viewing window. Clipping boundary that defines the visible part of model space.

VIO. Video Input/Output.

virtual memory (VM). Addressable space that is apparent to the user as the processor storage space, but not having a fixed physical location.

virtual storage. Synonymous with *virtual memory*.

visible region. A window's presentation space, clipped to the boundary of the window and the boundaries of any overlying window.

volume. (1) A file-system driver that uses a block device driver for input and output operations to a local or remote device. (2) A portion of data, together with its data carrier, that can be handled conveniently as a unit.

W

wild-card character. The global file-name characters asterisk (*) and question mark (?).

window. A rectangular area of the screen with visible boundaries within which information is displayed. A window can be smaller than or the same size as the screen. Windows can appear to overlap on the screen.

window class. The grouping of windows whose processing needs conform to the services provided by one window procedure.

window coordinates. The means by which a window position or size is defined; measured in device units, or *pels*.

window procedure. Code that is activated in response to a message. The procedure controls the appearance and behavior of its associated windows.

window rectangle. The means by which the size and position of a window is described in relation to the desktop window.

window style. The set of properties that influence how events related to a particular window will be processed.

workstation. A display screen together with attachments such as a keyboard, a local copy device, or a tablet.

world coordinates. Application-convenient coordinates used for drawing graphics.

world-coordinate space. Coordinate space in which graphics are defined before transformations are applied.

WYSIWYG. What You See Is What You Get. A capability that enables text to be displayed on a screen in the same way it will be formatted on a printer.

Z

z-order. The order in which sibling windows are presented. The topmost sibling window obscures any portion of the siblings that it overlaps; the same effect occurs down through the order of lower sibling windows.

zooming. In graphics applications, the process of increasing or decreasing the size of picture.

Index

E

F

Q

R

S

T

U

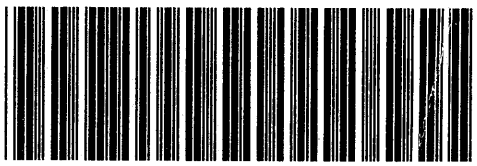
V

W

International Business
Machines Corporation

Printed in the
United States of America

10G6263



S10G-6263-00

P10G6263