 This is part of **Family API** which allow to create dual-os version of program runs under OS/2 and DOS

Note: This is legacy API call. It is recommended to use 32-bit equivalent

2021/09/17 04:47 · prokushev · 0 Comments
2021/08/20 03:18 · prokushev · 0 Comments

DosExecPgm

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

Syntax

```
DosExecPgm (ObjNameBuf, ObjNameBufL, ExecFlags, ArgPointer, EnvPointer,
ReturnCodes, PgmPointer)
```

Parameters

- ObjNameBuf (PCHAR) - output : Address of the name of the object that contributed to the failure of DosExecPgm is returned.
- ObjNameBufL (SHORT) - input : Length, in bytes, of the buffer described by ObjNameBuf.
- ExecFlags (USHORT) - input : Indicates how the program executes in relation to the requestor and whether execution is under conditions for debugging.
    - 0 - Execution is synchronous to the parent process. The termination code and result code are stored in the two-word structure.
    - 1 - Execution is asynchronous to the parent process. When the child process terminates, its result code is discarded. The process ID is stored in the first word of the two-word structure ReturnCodes.
    - 2 - Execution is asynchronous to the parent process. When the child process terminates, its result code is saved for examination by a DosCWait request. The process ID is stored in the first word of the two-word structure ReturnCodes.
    - 3 - Execution is the same as if ExecFlags=2 is specified, plus debugging conditions are present for the child process.
    - 4 - Execution is asynchronous to and detached from the parent process session. When the detached process is started, it is not affected by the ending of the parent process.
    - 5 - The program is loaded into storage and made ready to execute, but is not placed into execution until the session manager dispatches the threads belonging to the process.
    - 6 - Execution is the same as if ExecFlag=2 is specified, with the addition of debugging conditions being present for the child process and any of its descendants.

Some memory is consumed for uncollected result codes. Issue DosCWait to release this memory. If result codes are not collected, then ExecFlags=0 or 1 should be used.

- ArgPointer (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 parameters to the program name. The second ASCIIZ string is followed by an additional byte of zeros. A value of 0 for the address of ArgPointer means that no arguments are to be passed.
- EnvPointer(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 path name 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 filename
                       ; of program to run.
           .
           :
ao:   ASCIIZ           ; argument string 1
      ASCIIZ           ; argument string 2
      Byte of 0
```

The beginning of the environment segment is “eo” and “ao” is the offset of the first argument string in that segment. Register BX contains “ao” on entry to the target program. The address to the environment segment can also be obtained by issuing DosGetInfoSeg.

- ReturnCodes (PRESULTCODES) - output : Address of the structure containing the process ID or termination code and the result code indicating the reason for the child's termination. This structure is also used by a DosCWait request, which waits for an asynchronous child process to end.
- termcodepid (USHORT) : 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 terminated.
    - 0 - EXIT (normal)

    - 1 - Hard error abort
    - 2 - Trap operation
    - 3 - Unintercepted DosKillProcess
- resultcode (USHORT) : Result code specified by the terminating synchronous process on its last DosExit call.
- PgmPointer (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 path (it contains a “:” in the second position - or it contains a “\” - or both), then the file name must include the extension .COM or .EXE, 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.

Return Code

rc (USHORT) - return

Return code descriptions are:

- 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
- 180 ERROR_INVALID_SEGMENT_NUMBER
- 182 ERROR_INVALID_ORDINAL
- 188 ERROR_INVALID_STARTING_CODESEG
- 189 ERROR_INVALID_STACKSEG
- 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_RELOC_CHAIN_XEEDS_SEGLIM

Remarks

The target program is located and loaded into storage if necessary. A process is created and executed for the target program. The new process is created with an address space separate from its parent; that is, a new Local Descriptor Table (LDT) is built for the process.

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 pending completion of the child process. Other threads in the requesting process may continue to run.

If asynchronous execution is indicated, DosExecPgm returns with the process ID of the started child process. Specifying ExecFlags=2 allows the parent process to issue a DosCWait request after the DosExecPgm request, so it can examine the result code returned when the child process terminates. If ExecFlags=1 is specified, the result code of the asynchronous child process is not returned to the parent process.

A child process inherits file handles obtained by its parent with DosOpen calls that indicated inheritance . The child process also inherits handles to pipes created by the parent process with DosMakePipe.

Because a child process has the ability to 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 a file.

For example, a parent process can create two pipes with DosMakePipe requests. It can issue DosDupHandle to redefine the read handle of one pipe as standard input (0000H), and the write handle of the other pipe as standard output (0001H). 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 writes to one pipe, and the parent process reads what the child 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 started with ExecFlags=3 or ExecFlags=6 is provided a trace flag facility. This facility and the Ptrace buffer provided by DosPtrace 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 a DosExecPgm or a DosStartSession request.

A detached process is treated as an orphan of the parent process and executes in the background. Thus, it cannot make any VIO, KBD, or MOU calls, except 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.

Note: If the target program's entry point is in a segment that has IOPL indicated, this causes a general protection fault and the process is terminated. If any dynamic link module being used by the new

process has an initialization routine specified in a segment that has IOPL indicated, general protection fault occurs and the process is terminated.

Family API Considerations

Some options operate differently in DOS mode than in OS/2 mode. Therefore, the following restrictions apply to DosExecPgm when coding in DOS mode:

- ExecFlags must be set to zero. This value is not related to the PID of the program being executed. If ExecFlags <> 0, DosExecPgm returns the error code ERROR_INVALID_DATA.
- The ObjNameBuf field is used to provide additional information in the OS/2 mode environment as to why the DosExecPgm failed. The information is not relevant or available in DOS 2.X or DOS 3.X. Therefore, the buffer is filled in with blanks.
- The ReturnCodes two-word structure is very similar to the OS/2 mode environment. The first word is a termination code with the following meanings:
  - 0 - Exit (normal exit and termination by call INT 21H AH=31H)
  - 1 - Hard error abort
  - 2 - Not returned
  - 3 - Termination by Ctrl -Break.

The second word contains the ResultCode specified by the terminating process on its DosExit call (or INT 21H AH=4CH call).

Application Type Considerations

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.

Bindings

C

```
typedef struct _RESULTCODES {  /* resc */

  USHORT codeTerminate;        /* Termination Code -or- Process ID */
  USHORT codeResult;           /* Exit Code */

} RESULTCODES;

#define INCL_DOSPROCESS
```

```
USHORT  rc = DosExecPgm(ObjNameBuf, ObjNameBufL, ExecFlags, ArgPointer,
                        EnvPointer, ReturnCodes, PgmPointer);

PCHAR        ObjNameBuf;    /* Address of object name buffer (returned) */
SHORT        ObjNameBufL;   /* Length of object name buffer */
USHORT       ExecFlags;     /* Execute asynchronously/trace */
PSZ          ArgPointer;    /* Address of argument string */
PSZ          EnvPointer;    /* Address of environment string */
PRESULTCODES ReturnCodes;   /* Address of termination codes (returned) */
PSZ          PgmPointer;    /* Address of program file name */

USHORT       rc;            /* return code */
```

MASM

```
RESULTCODES struc

resc_codeTerminate dw  ? ;Termination Code -or- Process ID
resc_codeResult    dw  ? ;Exit Code

RESULTCODES ends

EXTRN  DosExecPgm:FAR
INCL_DOSPROCESS     EQU 1

PUSH@  OTHER   ObjNameBuf    ;Object name buffer (returned)
PUSH   WORD    ObjNameBufL   ;Length of object name buffer
PUSH   WORD    ExecFlags     ;Execute asynchronously/trace
PUSH@  ASCIIZ  ArgPointer    ;Address of argument string
PUSH@  ASCIIZ  EnvPointer    ;Address of environment string
PUSH@  DWORD   ReturnCodes   ;Termination codes (returned)
PUSH@  ASCIIZ  PgmPointer    ;Program file path name string
CALL   DosExecPgm
```

Returns WORD

Example Code

This example starts up the program simple.exe and then waits for it to finish. Then the termination and return codes are printed.

```
#define INCL_DOSPROCESS
#define START_PROGRAM "simple.exe"

CHAR        LoadError[100];
PSZ         Args;
PSZ         Envs;
RESULTCODES ReturnCodes;
```

```
   USHORT      rc;

     if(!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 */
        printf("Termination Code %d  Return Code %d \n",
               ReturnCodes.codeTerminate,
               ReturnCodes.codeResult);
-- simple.exe --

  #define INCL_DOSPROCESS
  #define RETURN_CODE 0

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

The following example demonstrates how to create a process, obtain process ID information, and kill a process. Process1 invokes process2 to run asynchronously. It obtains and prints some PID information, and then kills process2.

```
  /* ---- process1.c ---- */
  #define INCL_DOSPROCESS
  #include <os2.h>
  #define START_PROGRAM "process2.exe"   /* Program pointer */

  main()
  {
   CHAR          ObjFail [50];       /* Object name buffer */
   RESULTCODES   ReturnCodes;        /* */
   PIDINFO       PidInfo;
   PID           ParentID;           /* */
   USHORT        rc;

   printf("Process1 now running. \n");

   /** Start a child process. **/
   if(!(DosExecPgm(ObjFail,          /* Object name buffer */
                   sizeof(ObjFail),  /* Length of obj. name buffer */
                   EXEC_ASYNC,       /* Execution flag - asynchronous */
                   NULL,             /* No args. to pass to process2*/
                   NULL,             /* Process2 inherits process1's
environment */
                   &ReturnCodes,     /* Ptr. to resultcodes struct. */
```

```
                    START_PROGRAM)))  /* Name of program file */
      printf("Process2 started. \n");

    /** Obtain Process ID information and print it **/
    if(!(rc=DosGetPID(&PidInfo)))       /* Process ID's (returned) */
      printf("DosGetPID: current process ID is %d; thread ID is %d; parent
process ID is %d.\n",
              PidInfo.pid, PidInfo.tid, PidInfo.pidParent);
    if(!(rc=DosGetPPID(
             ReturnCodes.codeTerminate, /* Process whose parent is wanted */
             &ParentID)))               /* Address to put parent's PID */
      printf("Child process ID is %d; Parent process ID is %d.\n",
              ReturnCodes.codeTerminate, ParentID);

    /** Terminate process2 **/
    if(!(rc=DosKillProcess(DKP_PROCESSTREE,     /* Action code - kill process
and descendants */
                  ReturnCodes.codeTerminate)))  /* PID of root of process
tree */
      printf("Process2 terminated by process1.\n");
  }


  /* ---- process2.c ---- */
  #define INCL_DOSPROCESS
  #include <os2.h>
  #define SLEEPTIME   500L
  #define RETURN_CODE 0

  main()
  {
   printf("Process2 now running.\n");

   /* Sleep to allow process1 to kill it */
   DosSleep(SLEEPTIME);                 /* Sleep interval */
   DosExit(EXIT_PROCESS,                /* Action Code */
           RETURN_CODE);                /* Result Code */
  }
```

Note

This text based on http://www.edm2.com/index.php?title=DosExecPgm_(FAPI)

Family API		
DOS	Process Manager	DosBeep DosExit DosSleep DosExecPgm
	File Manager	DosChDir DosChgFilePtr DosClose DosDelete DosDupHandle DosMkDir DosMove DosQCurDir DosQCurDisk DosSetFileMode DosOpen DosQFileInfo DosRead DosQFileMode DosQFSInfo DosQVerify DosRmDir DosSelectDisk DosFindClose DosFindFirst DosFindNext DosSetFileInfo DosSetVerify DosWrite DosFileLocks DosSetFHandState DosNewSize DosBufReset DosQFHandState DosSetFSinfo DosShutdown
	Memory Manager	DosFreeSeg DosSubAlloc DosSubFree DosSubSet DosAllocHuge DosAllocSeg DosReallocHuge DosReallocSeg DosGetHugeShift DosCreateCSAlias
	NLS	DosCaseMap DosGetCtryInfo DosGetDBCSEv DosSetCtryCode DosGetCollate DosGetMessage DosInsMessage DosPutMessage
	Date and Time	DosSetDateTime DosGetDateTime
	Devices	DosDevConfig DosDevIOCtl DosDevIOCtl2
	Signals	DosHoldSignal DosSetSigHandler
	Misc	BadDynLink DosGetEnv DosGetMachineMode DosGetVersion DosError DosErrClass DosSetVec
KBD		KbdCharIn KbdFlushBuffer KbdGetStatus KbdSetStatus KbdStringIn KbdPeek
VIO		VioGetBuf VioGetConfig VioGetCurPos VioGetCurType VioGetPhysBuf VioReadCellStr VioReadCharStr VioScrollUp VioScrollDn VioScrollLf VioScrollRt VioScrUnLock VioSetCurPos VioSetCurType VioSetMode VioGetMode VioShowBuf VioWrtCellStr VioWrtCharStr VioWrtCharStrAtt VioWrtNAttr VioWrtNCell VioWrtNChar VioWrtTTY VioScrLock VioPopUp
Tools		BIND
Modules		DOSCALLS.DLL VIOCALLS.DLL KBDCALLS.DLL MSG.DLL
Libraries		API.LIB OS2386.LIB FAPI.LIB DOSCALLS.LIB SUBCALLS.LIB

2018/08/25 15:05 · prokushev · 0 Comments