

This call returns the current process ID, thread ID, and the process ID of the parent process.

Syntax

DosGetPID (ProcessIDs)

Parameters

;ProcessIDs (PPIDINFO) - output : Address of the structure where the ID information is returned. ::pid (PID) : Current process identifier. ::tid (TID) : Thread (of the current process) identifier. ::pidParent (PID) : Parent process (of the current process) identifier.

Return Code

;rc (USHORT) - return:Return code description is: *0 NO_ERROR

Remarks

The process ID may be used to generate uniquely named temporary files, or for communication with signals. For more information on signals, see [DosFlagProcess](#) and [DosSendSignal](#).

In the OS/2 environment, thread IDs are used with calls that manipulate threads in the current process. For more information, see [DosSuspendThread](#), [DosResumeThread](#), [DosGetPrtty](#), and [DosSetPrtty](#).

If the application is executing in the OS/2 environment, it is more efficient to obtain these variables by calling [DosGetInfoSeg](#) instead of [DosGetPID](#). However, applications written to the family API cannot depend on the availability of [DosGetInfoSeg](#).

To get an ID for a process other than the current process or its parent process, issue [DosGetPPID](#).

Bindings

MASM

<PRE> PIDINFO struc

```
pidi_pid      dw  ? ;current process' process ID
pidi_tid      dw  ? ;current process' thread ID
pidi_pidParent dw  ? ;process ID of the parent
```

PIDINFO ends

EXTRN DosGetPID:FAR INCL_DOSPROCESS EQU 1

PUSH@ OTHER ProcessIDsArea ;Process IDs (returned) CALL DosGetPID

Returns NONE </PRE>

C

```
<PRE> typedef struct _PIDINFO { /* pidi */
```

```
PID pid;           /* current process' process ID */
TID tid;           /* current process' thread ID */
PID pidParent;     /* process ID of the parent */
```

```
} PIDINFO;
```

```
#define INCL_DOSPROCESS
```

```
USHORT rc = DosGetPID(ProcessIDsArea);
```

```
PPIDINFO ProcessIDsArea; /* Process IDs (returned) */ USHORT rc; /* return code */ </PRE>
```

Example

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. <PRE> /* --- 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");

} </PRE> <PRE> /* --- 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 */

} </PRE>

```

[Dos16](#)

From:

<https://osfree.su/doku/> - **osFree wiki**

Permanent link:

<https://osfree.su/doku/doku.php?id=en:docs:fapi:dosgetpid&rev=1632030507>

Last update: **2021/09/19 05:48**

