2025/11/02 23:46 1/4 DosExit

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

# **Syntax**

DosExit (ActionCode, ResultCode)

### **Parameters**

ActionCode (USHORT) - input: Terminates the process and all its threads.

Value	Definition
0	The current thread ends.
1	All threads in the process end.

• ResultCode (USHORT) - input : Program's completion code. It is passed to any thread that issues DosCwait for this process.

#### **Remarks**

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

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

Do not terminate thread 1 without terminating 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 specified with ActionCode=1 to ensure the process ends.

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

# **Family API Considerations**

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

- There is no thread support in DOS 3.3; therefore DosExit exits the currently executing program.
- If ActionCode = 0 this option is ignored. It is equivalent to an ActionCode = 1.

### **Example Code**

# **C Binding**

```
#define INCL_DOSPROCESS

VOID    DosExit(ActionCode, ResultCode);
USHORT    ActionCode;    /* Indicates end thread or process */
USHORT    ResultCode;    /* Result Code to save for DosCwait */
```

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
#define START_PROGRAM "simple.exe"
#define RETURN OK 3
            LoadError[100];
CHAR
PSZ
            Args;
PSZ
            Envs;
RESULTCODES ReturnCodes;
USHORT
            rc:
   if(!DosExecPgm(LoadError,
                                             /* Object name buffer */
                                             /* Length of object name buffer
                   sizeof(LoadError),
*/
                                             /* Asynchronous/Trace flags */
                  EXEC_SYNC,
                                             /* Argument string */
                  Args,
                                             /* Environment string */
                  Envs,
                                            /* Termination codes */
                  &ReturnCodes,
                  START PROGRAM))
                                             /* Program file name */
      if (ReturnCodes.codeResult == RETURN OK) /* Check result code */
         printf("things are ok..");
      else
         printf("something is wrong...");
```

\_\_\_\_\_simple.exe\_\_\_\_\_

The following example shows how to suspend and resume execution of a thread within a process. The main thread creates Thread2 and allows it to begin executing. Thread2 iterates through a loop that

2025/11/02 23:46 3/4 DosExit

prints a line and then sleeps, relinquishing its time slice to the main thread. After one iteration by Thread2, the main thread suspends Thread2 and then resumes it. Subsequently, Thread2 completes the remaining three iterations.

```
#define INCL DOSPROCESS
#include <os2.h>
#define
          SEGSIZE
                        4000
                               /* Number of bytes requested in segment */
                               /* Segment allocation flags - no sharing */
#define
          ALLOCFLAGS
                        0
                               /* Sleep interval - 5 milliseconds */
#define
          SLEEPSHORT
                        5L
                               /* Sleep interval - 75 milliseconds */
#define
          SLEEPLONG
                        75L
#define
          RETURN CODE
                               /* Return code for DosExit() */
VOID APIENTRY Thread2()
{
 USHORT
             i;
 /* Loop with four iterations */
  for(i=1; i<5; i++)
    printf("In Thread2, i is now %d\n", i);
    /* Sleep to relinquish time slice to main thread */
    DosSleep(SLEEPSHORT);
                                   /* Sleep interval */
  }
 DosExit(EXIT_THREAD,
                                   /* Action code - end a thread */
                                   /* Return code */
          RETURN CODE);
}
main()
{
 TID
             ThreadID;
                                   /* Thread identification */
             ThreadStackSel;
                                   /* Segment selector for thread stack */
  SEL
                                   /* Ptr. to end of thread stack */
  PBYTE
             StackEnd;
 USHORT
             rc;
  /** Allocate segment for thread stack; make pointer to end of stack. **/
       We must allocate a segment in order to preserve segment **/
       protection for the thread.
  rc = DosAllocSeg(SEGSIZE,
                                        /* Number of bytes requested */
                                        /* Segment selector (returned) */
                   &ThreadStackSel,
                                        /* Allocation flags - no sharing */
                   ALLOCFLAGS);
 StackEnd = MAKEP(ThreadStackSel, SEGSIZE-1);
  /** Start Thread2 **/
  if(!(rc=DosCreateThread((PFNTHREAD) Thread2,
                                                   /* Thread address */
                                                   /* Thread ID (returned) */
                       &ThreadID,
                                     /* End of thread stack */
                       StackEnd)))
    printf("Thread2 created.\n");
```

```
/* Sleep to relinquish time slice to Thread2 */
 if(!(DosSleep(SLEEPSHORT)))
                                               /* Sleep interval */
   printf("Slept a little to let Thread2 execute.\n");
  /**** Suspend Thread2, do some work, then resume Thread2 *****/
  if(!(rc=DosSuspendThread(ThreadID)))
                                               /* Thread ID */
    printf("Thread2 SUSPENDED.\n");
  printf("Perform work that will not be interrupted by Thread2.\n");
  if(!(rc=DosResumeThread(ThreadID)))
                                               /* Thread ID */
    printf("Thread2 RESUMED.\n");
  printf("Now we may be interrupted by Thread2.\n");
 /* Sleep to allow Thread2 to complete */
 DosSleep(SLEEPLONG);
                                               /* Sleep interval */
}
```

## **MASM Binding**

```
EXTRN DosExit:FAR
INCL_DOSPROCESS EQU 1

PUSH WORD ActionCode ;Indicates end thread or process
PUSH WORD ResultCode ;Result Code to save for DosCwait
CALL DosExit
```

### Note

This text based on http://www.edm2.com/index.php/DosExit (FAPI)

From:

http://www.osfree.org/doku/ - osFree wiki

Permanent link:

http://www.osfree.org/doku/doku.php?id=en:docs:fapi:dosexit&rev=1535272406

Last update: 2018/08/26 08:33

