 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

DosHoldSignal

This call temporarily disables or enables signal processing for the current process.

Syntax

```
DosHoldSignal (ActionCode)
```

Parameters

- ActionCode (USHORT) - input : Disables or enables signals intended for the current process.

Value	Definition
0	Signals are enabled
1	Signals are disabled

Return Code

rc (USHORT) - return

Return code descriptions are:

- 0 NO_ERROR
- 1 ERROR_INVALID_FUNCTION

Remarks

DosHoldSignal with ActionCode = 1 causes signal processing (except central processing errors and numeric processor errors) to be postponed until a DosHoldSignal with ActionCode = 0 is issued. Any signals that occur while processing is disabled are recognized, but not accepted until signal recognition is enabled.

To allow for nesting of requests, a count of the number of outstanding DosHoldSignal requests with ActionCode = 1 are maintained.

DosHoldSignal is used by library routines, subsystems, and similar code that lock critical sections or temporarily reserve resources needed to prevent a signal from terminating a task. A process can have only one signal handling address for each signal. Dynalink routines should not have a signal handler (which might override a handler established by a calling process).

Signals can be held for a short period and should be released and re-held, if necessary. Their guidelines for proper use are similar to hardware interrupt counterparts such as the CLI/STI instructions.

Family API Considerations

Some options operate differently in the DOS mode than in the OS/2 mode. Therefore, the following restriction applies to DosHoldSignal when coding for the DOS mode:

The only signal recognized in the DOS mode is SIGINTR (Ctrl-C) and SIGBREAK. Only SIGINTR and SIGBREAK are turned off by this call.

Bindings

C Binding

```
#define INCL_DOSSIGNALS

USHORT  rc = DosHoldSignal(ActionCode);

USHORT           ActionCode;    /* Indicate to Disable/Enable Signals */

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

MASM Binding

```
EXTRN  DosHoldSignal:FAR
INCL_DOSSIGNALS     EQU 1

PUSH   WORD    ActionCode    ;Indicate to Disable/Enable Signals
CALL   DosHoldSignal

Returns NONE
```

Example

The following example illustrates the use of the Ctrl-C (SIGINTR) signal to signal time-critical events. Process1 invokes process2, which establishes a signal handler named CtrlC_Handler() and waits, by blocking on a reserved RAM semaphore, for a signal from process1. A portion of process2 is immune

to signalling.

```
#define INCL_DOSPROCESS
#define INCL_DOSSIGNALS

#include <os2.h>

#define SLEEPTIME       200L              /* Sleep interval */
#define START_PROGRAM   "process2.exe"    /* Program name */


main()
{
  CHAR          ObjFail[50];
  PSZ           Args;
  PSZ           Envs;
  RESULTCODES   ReturnCodes;
  USHORT        rc;

  /* Start process2 and check its PID */
  if(!(DosExecPgm(ObjFail,                /* Object name buffer */
                  sizeof(ObjFail),        /* Length of obj. name buffer */
                  EXEC_ASYNC,             /* Execution flag */
                  Args,                   /* Ptr. to argument string */
                  Envs,                   /* Ptr. to environment string */
                  &ReturnCodes,           /* Ptr. to resultcodes struct.*/
                  START_PROGRAM)))        /* Name of program file */
    printf("Process2 started.\n");
  printf("Process2 ID is %d\n", ReturnCodes.codeTerminate);

  /* Sleep to give time slice to process2 */
  DosSleep(SLEEPTIME);                    /* Sleep interval */

  /*** After process2 sets signal handler, send process2 a signal ***/
  if(!(rc = DosSendSignal(ReturnCodes.codeTerminate,  /* PID of process2 */
                          SIG_CTRLC)))                /* Signal to send*/
    printf("Ctrl-C signal sent from Process1 to Process2.\n");
}

/* ----- process2.c ----- */

#define INCL_DOSPROCESS
#define INCL_DOSSIGNALS
#define INCL_DOSERRORS

#include <os2.h>

#define SLEEPTIME         50L
#define TIMEOUT           5000L
```

```
VOID APIENTRY CtrlC_Handler(arg1, arg2)    /** Define signal handler **/
  USHORT      arg1;
  USHORT      arg2;
{
  printf("Handler for Ctrl-C now running.\n");
  return;
}

main()
{
  ULONG            RamSem = 0L;   /* Allocate and initialize Ram
                                     Semaphore */
  ULONG far        *RamSemHandle = &RamSem;  /* Ram Semaphore handle */
  USHORT           rc;

  /* Establish signal handler */
  if(!(rc=DosSetSigHandler((PFNSIGHANDLER) CtrlC_Handler,
                           NULL,          /* Previous handler - ignored */
                           NULL,          /* Previous action - ignored */
                           SIGA_ACCEPT,   /* Request type */
                           SIG_CTRLC)))   /* Signal number */
    printf("Process2 has set Ctrl-C handler.\n");
  else

    /* Error processing on rc */;
  /* Get semaphore for first time */
  if(!(rc=DosSemRequest(RamSemHandle,       /* Semaphore handle */
                        TIMEOUT)))          /* Timeout interval */
    printf("Semaphore obtained.\n");

  /*** Disable and then enable signal-handling ***/
  if(!(rc=DosHoldSignal(HLDSIG_DISABLE)))   /** Action code - disable **/
  {
    printf("Signalling DISABLED.\n");

    /* Do signal-proof work here */
    if(!(rc=DosHoldSignal(HLDSIG_ENABLE)))  /** Action code - enable **/
      printf("Signalling ENABLED.\n");
  }
  /* At this point, process1 may have sent a Ctrl-C signal. */
  /* Try to obtain semaphore again -- resulting in Timeout. */
  /* The Timeout, however, may be interrupted by the signal. */

  printf("Process2 will now wait on a Ramsem for a while.\n");
  if((rc=DosSemRequest(RamSemHandle,     /* Semaphore handle */
                       TIMEOUT))         /* Timeout interval */
     == ERROR_INTERRUPT)
  printf("Process2 interrupted while waiting, rc is %d.\n", rc);
}
```

Note

Text based on http://www.edm2.com/index.php/DosHoldSignal

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