 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

DosSetSigHandler

This call notifies OS/2 of a handler for a signal. It may also be used to ignore a signal or install a default action for a signal.

Syntax

```
DosSetSigHandler (Routine, PrevAddress, PrevAction, Action, SigNumber)
```

Parameters

;Routine (PFNSIGHANDLER) - input : Address of the entry point of routine that receives control when a signal equal to SigNumber is received. ;PrevAddress (PFNSIGHANDLER FAR *) - output: Address of the previous signal handler. This operand may be coded as null (= 0), then it is ignored. ;PrevAction (PUSHORT) - output : Address of the previous signal action. Only values 0 to 3 are returned. This operand may be coded as null (= 0), then it is ignored. ;Action (USHORT) - input : Indicates what action to take when the specified signal is received: `'Value Definition'` 0 The system default action is installed for the signal. 1 The signal is to be ignored. 2 The routine receives control when the SigNumber occurs. 3 It is an error for any program to signal this SigNumber to this process. 4 The current signal is reset without affecting the disposition of the signal. ; SigNumber (USHORT) - input : Signal number to be intercepted by this signal handler. The signal numbers defined are: `'Value Definition'` 1 Ctrl-C (SIGINTR) 3 Program terminated (SIGTERM) 4 Ctrl-Break (SIGBREAK) 5 Process flag A (SIGPFA) 6 Process flag B (SIGPFB) 7 Process flag C (SIGPFC) `'Note:'` Presentation Manager applications may not establish signal handlers for Ctrl-C and Ctrl-Break. Establishing a signal handler for Ctrl-C and Ctrl-Break is supported for VIO-Windowable and full-screen applications.

The following chart indicates what signal to specify to cause the signal handler to get control for the CTRL-C and CTRL-Break key sequences in each of the keyboard modes (ASCII and Binary):

	ASCII Mode	**BINARY Mode**
CTRL-C	SIGINTR	
CTRL-Break	SIGINTR	SIGBREAK

Return Code

rc (USHORT) - return:Return code descriptions are:

- 0 NO_ERROR
- ERROR_INVALID_FUNCTION
- 209 ERROR_INVALID_SIGNAL_NUMBER
- 210 ERROR_THREAD_1_INACTIVE

Remarks

When the signal indicated by SigNumber occurs, the signal handling routine receives control with:

(SS:SP) Far return address (SS:SP+4) SigNumber being processed (SS:SP+6) SigArg furnished on the DosFlagProcess request, if appropriate.

Other than SS, SP, CS, IP and flags, all registers contain the same values they contained at the time the signal was received. The handler may exit by executing an intersegment return instruction, or by manually setting the stack frame to some known state and jumping to some known location. If the former option is selected, execution resumes where it was interrupted, and all registers are restored to their values at the time of the interruption.

The signal handler is given control under the first thread of a process, not a thread created by the DosCreateThread system request. It is invalid to issue this system call when thread 1 has terminated. If thread 1 terminates with other threads still active, all signals are reset to the default action.

To return from the signal, the handler must remove the signal number and signal argument passed as parameters. For handlers written in most high-level languages, this is done automatically. A handler written in assembler language must execute a Far RET 4 instruction or its equivalent, to return to the caller. The signal handler may also reset the stack pointer to some previous valid stack frame and jump to some other part of the program.

The values returned in PrevAddress and PrevAction are to be used for restoring the previous signal handler when the current process no longer wishes to intercept this signal. For Action = 4, no values are returned for PrevAddress or PrevAction.

When a signal is issued from the base keyboard device driver in response to a Ctrl-C or Ctrl-Break key press, the default action terminates the process if the application did not install a signal handler for any signal numbers 1-4.

For signals of type SIGINTR or SIGBREAK, a call to DosSetSigHandler also determines which process within the current session is signalled as a result of a device driver call to Device Helper Services for the SendEvent function and CTRL-C (or CTRL-BREAK) event type. (See the IBM Operating System/2 Version 1.2 I/O Subsystems And Device Support Volume 1, Device Helper Services discussion). This process is known as the `signal focus“ for SIGINTR (or SIGBREAK) within its session. The signal focus for SIGINTR need not be the same process as the signal focus for SIGBREAK. The determination for signal focus follows.

Initially, a session has no signal focus for SIGINTR (or SIGBREAK). A process becomes the signal focus

for SIGINTR (or SIGBREAK) within its session if it calls DosSetSigHandler with ActionCode equal to 1, 2, or 3. A process remains the signal focus until:

- The process terminates.
- The process calls DosSetSigHandler with ActionCode equal to zero.
- Another process calls DosSetSigHandler with ActionCode equal to 1, 2, or 3.

In the first two cases, the parent or its closest related ancestor process that has a handler installed for the appropriate signal becomes the focus. If no eligible process exists, the session ceases to have a signal focus for that signal.

If a device driver makes a SendEvent call for CTRL-C or CTRL-BREAK and the current session has no focus for the corresponding signal, all processes in the session are signaled with SIGTERM to terminate.

Family API Considerations

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

- The only signals recognized in DOS are SIGINTR (Ctrl-C) and SIGBREAK.
- The option Action=3 generates an “invalid signal number” error.
- If SigNumber is any value other than SIGINTR or SIGBREAK, then an “invalid signal number” error is generated.

SIGINTR is fully supported, and SIGBREAK is related to SIGINTR. Therefore, if SIGINTR is specified, both SIGINTR and SIGBREAK are transferred to the SIGINTR handler. SIGBREAK is permitted as a coded value, but the request to set SIGBREAK is ignored. To be compatible in all environments, SIGBREAK and SIGINTR should be considered together in all cases.

Bindings

C Binding

```
#define INCL_DOSSIGNALS

USHORT  rc = DosSetSigHandler(Routine, PrevAddress, PrevAction, Action,
                              SigNumber);

PFNSIGHANDLER       Routine;     /* Signal handler */
PFNSIGHANDLER FAR * PrevAddress; /* Previous handler (returned) */
PUSHORT             PrevAction;  /* Previous action (returned) */
USHORT              Action;      /* Indicate request type */
USHORT              SigNumber;   /* Signal number of interest */

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

MASM Binding

```
EXTRN  DosSetSigHandler:FAR
INCL_DOSSIGNALS     EQU 1

PUSH@  DWORD   Routine        ;Signal handler
PUSH@  DWORD   PrevAddress    ;Previous handler (returned)
PUSH@  WORD    PrevAction     ;Previous action (returned)
PUSH   WORD    Action         ;Indicate request type
PUSH   WORD    SigNumber      ;Signal number of interest
CALL   DosSetSigHandler

Returns WORD
```

Example

The following example illustrates the use of a user-defined flag to signal time-critical events. The main thread installs a routine, named FlagA_Handler(), as the signal handler for user-defined Flag A. It then creates a thread and blocks on a reserved RAM semaphore; this thread obtains its process ID and signals the main thread via Flag A. The main thread responds by executing the signal handler.

```
#define INCL_DOSPROCESS
#define INCL_DOSSIGNALS
#define INCL_DOSERRORS

#include <os2.h>

#define TIMEOUT           5000L

TID         ThreadID;
BYTE        ThreadStack[4000];

VOID APIENTRY FlagA_Handler(arg1, arg2)       /* Define signal handler */
  USHORT      arg1;
  USHORT      arg2;
{
  printf("Handler for Flag A now running.\n");
  return;
}

VOID APIENTRY Thread_A()
{
  PIDINFO     PidInfo;
  USHORT      FlagArg;
  USHORT      rc;

  DosGetPID(&PidInfo);
  printf("Process ID is %d\n", PidInfo.pid);
```

```
  if(!(rc = DosFlagProcess(PidInfo.pid,
                           FLGP_PID,
                           PFLG_A,
                           FlagArg)))
    printf("FlagA signal sent from ThreadA to main thread.\n");
  else
    printf("FlagProcess rc is %d\n", rc)/* Error processing on rc */;
  DosExit(EXIT_THREAD,                  /* Action Code */
          RETURN_CODE);                 /* Result Code */

}

main()
{
  ULONG             RamSem = 0L;
  ULONG far         *RamSemHandle = &RamSem;
  USHORT            rc;

  if(!(rc=DosSetSigHandler((PFNSIGHANDLER) FlagA_Handler,
                           NULL,
                           NULL,
                           SIGA_ACCEPT,
                           SIG_PFLG_A)))
    printf("Main thread has set FlagA handler.\n");
  else
    /* Error processing on rc */;
  if(!(rc=DosSemRequest(RamSemHandle,
                        TIMEOUT)))
    printf("Semaphore obtained.\n");
  if(!(DosCreateThread((PFNTHREAD) Thread_A,
                        &ThreadID,
                        &ThreadStack[3999])))
    printf("ThreadA created.\n");
  printf("Main thread will now wait on a Ramsem for a while.\n");
  if((rc=DosSemRequest(RamSemHandle,
                       TIMEOUT))
      == ERROR_INTERRUPT)
  printf("Main thread interrupted while waiting, rc is %d.\n", rc);
}
```

Note

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

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