 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

DosAllocSeg

Syntax

```
DosAllocSeg (Size, Selector, AllocFlags)
```

Parameters

- Size (USHORT) - input: Number of bytes requested. The value specified must be less than or equal to 65535. A value of zero indicates 65536 bytes.
- Selector (PSEL) - output: Address where the selector of the segment allocated is returned.
- AllocFlags (USHORT) - input: Bit indicators describing the characteristics of the segment being allocated. The bits that can be set and their meanings are:

Bit	Description
15-4	Reserved and must be set to zero
3	If segment is shared, it can be decreased in size by DosReallocSeg
2	Segment may be discarded by the system in low memory situations
1	Segment is shareable through DosGetSeg
0	Segment is shareable through DosGiveSeg

Return Code

rc (USHORT) - return: Return code

- 0 NO_ERROR
- 8 ERROR_NOT_ENOUGH_MEMORY
- 87 ERROR_INVALID_PARAMETER

Remarks

DosAllocSeg allows a process to allocate a data segment up to 64KB in size, which is movable and swappable by the system. If your application needs to accommodate a large data structure that exceeds the 64KB limit, DosAllocHuge may be issued to allocate multiple segments as one huge block

of memory.

A segment allocated by DosAllocSeg with AllocFlags bit 2 set can be discarded by the system to remedy a low memory situation when the segment is not in use. Upon allocation, a discardable segment is locked and ready for access. The caller issues DosUnlockSeg when it is finished using the segment. The next time the caller needs to access the segment, it must issue DosLockSeg. During the time a segment is locked, it cannot be discarded, but it can still be swapped.

Allocate memory as discardable when it is needed to hold data for only short periods of time; for example, saved bit map images for obscured windows. Once the system discards a segment, the caller must reallocate the segment with DosReallocSeg and regenerate the data. Reallocating the segment automatically locks it for the first access.

A segment may also be designated as shared with another process. If a process issues DosAllocSeg with AllocFlags bit 0 set, then the segment allocated is shareable through DosGiveSeg. To share the segment in this manner, the owning process can then issue DosGiveSeg to obtain a selector for the sharer to use. The owning process then passes the selector to the sharer using some means of interprocess communication. The sharing process can use the selector to access the shared segment. If the shared segment has been designated discardable (AllocFlags bit 2 is also set), the sharer must issue DosLockSeg to lock the segment.

Memory allocated with DosAllocSeg is freed by a call to DosFreeSeg.

'Note:' This request may be issued from privilege level 2. However, the segment is allocated as a privilege level 3 segment.

Family API Considerations

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

- Requested Size value is rounded up to the next paragraph (16-byte).
- Selector is the actual segment address allocated.
- AllocFlags must be set to zero.

Bindings

C

```
#define INCL_DOSMEMMGR

USHORT  rc = DosAllocSeg(Size, Selector, AllocFlags);

USHORT  Size;        /* Number of bytes requested */
PSEL    Selector;    /* Selector allocated (returned) */
USHORT  AllocFlags;  /* Allocation flags */
```

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

MASM

```
EXTRN  DosAllocSeg:FAR
INCL_DOSMEMMGR      EQU 1

PUSH   WORD    Size          ;Number of bytes requested
PUSH@  WORD    Selector      ;Selector allocated (returned)
PUSH   WORD    AllocFlags    ;Allocation flags
CALL   DosAllocSeg

Returns WORD
```

Example Code

This example requests a segment of memory with 30,128 bytes. The segment can be shared with a DosGetSeg API call.

```
#define INCL_DOSMEMMGR

#define NUMBER_OF_BYTES 30128
#define ALLOC_FLAG SEG_GETTABLE

SEL    Selector;
USHORT rc;

   rc = DosAllocSeg(NUMBER_OF_BYTES,     /* # of bytes requested */
                    &Selector,           /* Selector allocated */
                    ALLOC_FLAG);         /* Allocation flags */
```

The following example requests a segment of memory with 4,000 bytes. The following example also 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 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 */
#define  ALLOCFLAGS    0      /* Segment allocation flags - no sharing */
#define  SLEEPSHORT    5L     /* Sleep interval - 5 milliseconds */
#define  SLEEPLONG     75L    /* Sleep interval - 75 milliseconds */
#define  RETURN_CODE   0      /* 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 */
  SEL        ThreadStackSel;       /* Segment selector for thread stack */
  PBYTE      StackEnd;             /* Ptr. to end of thread stack */
  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 */
                   &ThreadStackSel,      /* Segment selector (returned) */
                   ALLOCFLAGS);          /* Allocation flags - no sharing */
  StackEnd = MAKEP(ThreadStackSel, SEGSIZE-1);

  /** Start Thread2 **/
  if(!(rc=DosCreateThread((PFNTHREAD) Thread2,    /* Thread address */
                       &ThreadID,                 /* Thread ID (returned) */
                       StackEnd)))   /* End of thread stack */
    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 */
}
```

Note

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

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