Using Heap Allocation in Intel Assembly Language

Dynamic memory allocation is a feature we take for granted in high-level languages such as C++ and Java. Behind the scenes, such languages have a runtime heap manager handling program requests for storage allocation and deallocation. Generally, the heap managers work the same way: on startup, they request a large block of memory from the operating system. They create a *free list* of pointers to storage blocks. When an allocation request is received, the heap manager marks an appropriately sized block of memory as reserved and returns a pointer to the block. Later, when a delete request for the same block is received, the heap frees up the block, returning it to the free list. Whenever a new allocation request is received, the heap manager scans the free list, looking for the first available block large enough to grant the request.

Assembly language programs can perform dynamic allocation in a couple of ways. First, they can make system calls to get blocks of memory from the operating system. Second, they can implement their own heap managers that serve requests for smaller objects. In this article, we show how to implement the first method. The second method (heap manager), will be left for another article. The example program is a 32-bit protected mode applications running under Microsoft Windows.

You can request multiple blocks of memory of varying sizes from MS-Windows, using three simple Windows API functions: GetProcessHeap, HeapAlloc, and HeapFree.

GetProcessHeap returns a 32-bit integer handle to the program’s heap area. Save this handle and use it when calling other memory-related functions. Using this function, you can request (allocate) memory without having to create your own heap.

HeapAlloc returns the address of block of memory from an existing heap, identified by a heap handle. The allocated memory cannot be moved. If the memory cannot be allocated, the function returns NULL (0).

HeapFree frees a block of memory previously allocated from a heap, identified by its address and heap handle. If the block is freed successfully, the return value is nonzero. If the block cannot be freed, the function returns zero and you can call the GetLastError API function to get more information about the error.

Here's a good URL to begin reading about these functions. If the URL changes, search for *Memory Management Reference* on the Microsoft MSDN Web site:

```
http://www.msdn.microsoft.com/library/default.asp?url=/library/en-
us/memory/base/memory_management_reference.asp.
```

Example Program Here's a short program named heaptest.asm that uses dynamic memory allocation to create and fill a 1000-byte array:

```
.data
ARRAY_SIZE = 1000
NULL = 0

pArrayDWORD ?                    ; pointer to block of memory

hHeap   DWORD ?                  ; handle to the process heap
dwFlags DWORD HEAP_ZERO_MEMORY   ; set memory bytes to all zeros

str1 BYTE "Cannot allocate heap memory!",0dh,0ah,0
str2 BYTE "Writing data into the array...",0dh,0ah,0

.code
main PROC
   INVOKE GetProcessHeap         ; get handle heap
   mov hHeap,eax

   ; allocate the array's memory
   INVOKE HeapAlloc, hHeap, dwFlags, ARRAY_SIZE
   .IF eax == NULL
     mov  edx,OFFSET str1        ; "Cannot allocate..."
     call WriteString
     jmp quit
   .ELSE
     mov pArray,eax              ; save the pointer
   .ENDIF
   ; Fill the array with all "FFh"
   mov  edx,OFFSET str2          ; "Writing data into..."
   call WriteString
   mov  ecx,ARRAY_SIZE
   mov  esi,pArray               ; point to the array
L1:
   mov  BYTE PTR [esi],0FFh      ; insert a byte in the array
   inc  esi                      ; next location
   loop L1

   ; free the array
   INVOKE HeapFree, hHeap, dwFlags, pArray
quit:
   exit
main ENDP

END main
```

Linked List Example A student at Florida International University named Gabriel Perez used the Windows heap application API to create a menu-driven program that creates a linked list of names and id numbers (see *LinkedList.asm*). It inserts, finds, and removes list nodes, showing how dynamic allocation can be used in a more practical way than our first example.

```
TITLE Linked List Example        (LinkedList.asm)

; Uses dynamic allocation to interactively build a linked list
; of STRUCT objects. Has a nice interactive menu also.
; Written by Gabriel Perez, a Computer Science student at Florida
; International University, 11/18/2002.
; Used by permission.

INCLUDE Irvine32.inc

ID_MAX = 10
LASTNAME_MAX = 20
TRUE = 1
FALSE = 0

CUSTOMER STRUCT
   idNum         BYTE ID_MAX DUP(0)
   lastNam       BYTE LASTNAME_MAX DUP(0)
   nextNod       DWORD 0
CUSTOMER ENDS

sumOfEntryFields = (SIZEOF customer.lastNam + SIZEOF
customer.idNum)
```

```
.data
hHeap   DWORD ?
dwBytes DWORD ?
dwFlags DWORD HEAP_ZERO_MEMORY

progTitle BYTE "DYNAMIC MEMORY ALLOCATION VIA API CALLS",0
optA BYTE "A) DISPLAY CURRENT LIST OF CUSTOMERS",0
optB BYTE "B) SEARCH CUSTOMER",0
optC BYTE "C) ADD NEW CUSTOMER",0
optD BYTE "D) UPDATE CURRENT CUSTOMER",0
optE BYTE "E) DELETE EXISTING CUSTOMER",0
optF BYTE "F) EXIT PROGRAM",0
selection BYTE "PLEASE ENTER YOUR DESIRED SELECTION: ",0

newCustTitle    BYTE " --- ENTER A NEW CUSTOMER --- ",0
createNodMsg    BYTE "DO YOU WANT TO ENTER A NEW CUSTOMER ? Y/N: ",0
custIdMsg       BYTE "ENTER THE CUST ID: ",0
custLastMsg     BYTE "ENTER THE LAST NAME OF CUSTOMER: ",0

displayNothing   BYTE " --- THERE ARE NO CUSTOMERS IN MEMORY ---",0
titleMsg            BYTE  "---- LINK LIST CONTENTS ----",0
custIdInfo          BYTE "CUSTOMER ID: ",0
custLNameInfo       BYTE "CUSTOMER LAST NAME: ",0
spacer              BYTE "--------------------------------",0
custSearchMsg       BYTE " --- CUSTOMER SEARCH --- ",0
getSearchId         BYTE "PLEASE ENTER THE CUST ID TO DISPLAY: ",0

foundMsg            BYTE " --- CUSTOMER FOUND ---",0
deletedMsg          BYTE " ---  AND REMOVED   ---",0
notFoundMsg         BYTE " !!!! CUSTOMER NOT FOUND !!!!",0
newInfoMsg          BYTE " --- NEW CUSTOMER INFO ---",0
custUpdtMsg         BYTE " --- CUSTOMER SUCCESFULLY UPDATED --- ",0
row                 BYTE  0
column              BYTE  24

idNumber            BYTE  (ID_MAX+1) DUP(0)
lastName            BYTE  (LASTNAME_MAX+1) DUP(0)
response            BYTE  ?
searchId            BYTE  (ID_MAX+1) DUP(0)

head                DWORD ?
tail                DWORD ?
currNod             DWORD ?
prevNod             DWORD ?
nextNod             DWORD ?
foundVar            BYTE  FALSE

thisCust CUSTOMER {}
```

```
.code
main PROC
   INVOKE GetProcessHeap
   mov hHeap,eax

   mov dwBytes,SIZEOF customer

   mov  eax,yellow+(blue*16)
   call SetTextColor

   call createNewNode
   mov head,eax

   ENTRYPOINT:
   mov eax,tail
   mov currNod,eax
   call programMenu
   call getAndCallSelection
   cmp response, 'F'
   jne ENTRYPOINT

   call crlf
   call waitMsg

   ENDOFPROGRAM: exit
main ENDP

createNewNode PROC
   INVOKE heapAlloc, hHeap, dwFlags, dwBytes
   mov tail,eax
   ret
createNewNode ENDP

addTwoColumn PROC
   add row,2
   mov dl,column
   mov dh,row
   call gotoxy
   ret
addTwoColumn ENDP

programMenu PROC
   call Clrscr
   mov row,0
   mov dl,20
   mov dh,0
   call Gotoxy
```

```
    mov edx,OFFSET progTitle
    call writeString

    call addTwoColumn
    mov edx,OFFSET optA
    call writeString
    call addTwoColumn

    mov edx,OFFSET optB
    call writeString

    call addTwoColumn
    mov edx,OFFSET optC
    call writeString

    call addTwoColumn
    mov edx,OFFSET optD
    call writeString

    call addTwoColumn
    mov edx,OFFSET optE
    call writeString

    call addTwoColumn
    mov edx,OFFSET optF
    call writeString

    ret
programMenu ENDP

getAndCallSelection PROC
    mov dl,0
    mov dh,24
    call gotoxy
    mov edx, OFFSET selection
    call writeString
    call readChar
    mov response,al

    INVOKE Str_ucase, ADDR response
    mov al,'A'
    mov ah,'B'
    mov bl,'C'
    mov bh,'D'
    mov cl,'E'

    .IF (al == response)
```

```
      call showContents
    .ELSEIF (ah == response)
      call getSearch
      call waitMsg
    .ELSEIF (bl == response)
      call getData
      call moveToHeap
      call waitMsg
    .ELSEIF (bh == response)
      call getSearch
      .IF (foundVar == 1)
        call update
      .ENDIF
      call waitMsg
    .ELSEIF (cl == response)
      call getSearch
      .IF (foundVar == 1)
        call deleteNode
      .ENDIF
      call waitMsg
    .ENDIF

    ret
getAndCallSelection ENDP

showContents PROC
    mov edi,head
    mov ebx,00h

    call Clrscr

    mov edx,OFFSET titleMsg
    call writeString
    call Crlf
    Call Crlf

DISPLAYSTART:
    cmp [edi+sumOfEntryFields],ebx
    je NOMORE
    mov eax,[edi]
    mov prevNod,eax
    call displayCustomer

    add edi,SIZEOF thisCust.lastNam
    mov edi,[edi]
    mov currNod,edi

    JMP DISPLAYSTART
```

```
NOMORE: call waitMsg
   ret
showContents ENDP

displayCustomer PROC
   call Crlf
   mov edx,OFFSET custIdInfo
   call writeString
   mov edx,edi
   call writeString
   call Crlf

   mov edx,OFFSET custLNameInfo
   call writeString
   add edi,SIZEOF thisCust.IdNum
   mov edx,edi
   call writeString
   call Crlf

   mov edx,OFFSET spacer
   call writeString
   call Crlf

   ret
displayCustomer ENDP

getData PROC
   call Clrscr
   mov edx,OFFSET newCustTitle
   call writeString
   call Crlf
   call Crlf

   mov edx,OFFSET custIdMsg
   call writeString

   mov edx,OFFSET thisCust.idNum
   mov ecx,ID_MAX
   call readString

   mov edx,OFFSET custLastMsg
   call writeString

   mov edx,OFFSET thisCust.lastNam
   mov ecx,LASTNAME_MAX
   call readString
   call createNewNode
   mov eax,tail
   mov thisCust.nextNod,eax

   ret
getData ENDP

moveToHeap PROC
   mov esi,OFFSET thisCust
   mov edi,currNod

   INVOKE Str_copy, ADDR thisCust.idNum, edi

   add edi,SIZEOF thisCust.idNum
   INVOKE Str_copy, ADDR thisCust.lastNam, edi

   add edi,SIZEOF thisCust.lastNam
   mov eax,(CUSTOMER PTR [esi]).nextNod

   mov [edi],eax
   ret
moveToHeap ENDP

getSearch PROC
   call ClrScr
   mov ebx,00h
   mov edi,head
   cmp [edi+sumOfEntryFields],ebx
   je  NOTHING

   mov edx,OFFSET custSearchMsg
   call writestring
   call Crlf

   mov edx,OFFSET getSearchId
   call writeString

   mov edx,OFFSET searchId
   mov ecx,ID_MAX
   call readString

   call searchList
   jmp endproc

NOTHING:
   mov foundVar,FALSE
   mov edx,OFFSET displayNothing
   call writeString
```

```
ENDPROC:
   call crlf
   ret
getSearch ENDP

searchList PROC
   call ClrScr
   mov edi,head
   mov ebx,00h
   mov prevNod,edi

SEARCHLOOP:
   mov eax,[edi]
   INVOKE Str_compare, ADDR searchId, edi
   je  FOUND
   mov prevNod,edi
   add edi,sumOfEntryFields
   mov edi,[edi]
   cmp [edi+sumOfEntryFields],ebx
   je NOTFOUND
   jmp SEARCHLOOP

FOUND:
   mov foundVar,TRUE
   mov currNod,edi
   Call Crlf
   mov edx,OFFSET foundMsg
   call writeString
   call displayCustomer
   jmp AWAYWITHYOU

   NOTFOUND:
   mov foundVar,FALSE
   call Crlf
   mov edx,OFFSET notFoundMsg
   call writeString
   call Crlf

AWAYWITHYOU: ret
searchList ENDP

update PROC
   mov dl,0
   mov dh,6
   call Gotoxy

   mov edx,OFFSET newInfoMsg
```

```
    call WriteString
    call Crlf

    mov edx,OFFSET custIdMsg
    call writeString

    mov edx,OFFSET thisCust.idNum
    mov ecx,ID_MAX
    call readString

    mov edx,OFFSET custLastMsg
    call writeString

    mov edx,OFFSET thisCust.lastNam
    call readString

    mov edi,currNod
    INVOKE Str_copy, ADDR thisCust.idNum, edi

    add edi,SIZEOF thisCust.idNum
    INVOKE Str_copy, ADDR thisCust.lastNam, edi

    add edi,SIZEOF thisCust.lastNam

    call crlf
    mov edx,OFFSET custUpdtMsg
    call writestring
    call crlf

    ret
update ENDP

deleteNodePROC
    mov edi,currNod
    add edi,sumOfEntryFields
    mov eax,[edi]
    mov nextNod,eax

    mov edi,currNod
    .if(edi == head)
                           mov head,eax
    .endif

    mov edi,prevNod
    add edi,sumOfEntryFields
    mov eax,nextNod
    mov [edi],eax
    mov edi,currNod
    INVOKE heapFree, hHeap, dwFlags, edi

    call Crlf
    mov edx,OFFSET deletedMsg
    call writeString
    call Crlf

    ret
deleteNode ENDP
end main
```