Dynamic Memory

Dynamic memory allocation is the process of allocating memory during the execution of a program.

This is in contrast to **static memory allocation**, where memory is allocated at compile time.

There are two main ways to perform dynamic memory allocation in assembly language:

Using system calls: This involves making calls to the operating system to allocate and deallocate memory.

Implementing a heap manager: This involves implementing your own data structure and algorithms to manage memory allocation and deallocation.

The example program in the section you provided uses the first method. It makes system calls to the Windows operating system to allocate and deallocate memory.

Here is a summary of the steps involved in dynamic memory allocation using system calls:
Make a system call to allocate memory.

This will return a pointer to the allocated memory block. Use the allocated memory block.

Make a system call to deallocate the memory block when you are finished using it. The following table lists some of the Win32 API functions that can be used for dynamic memory allocation:

Function	Description
GetProcessHeap	Returns a 32-bit integer handle to the program's existing heap area in EAX. If the function succeeds, it returns a handle to the heap in EAX. If it fails, the return value in EAX is NULL.
HeapAlloc	Allocates a block of memory from a heap. If it succeeds, the return value in EAX contains the address of the memory block. If it fails, the returned value in EAX is NULL.
HeapCreate	Creates a new heap and makes it available to the calling program. If the function succeeds, it returns a handle to the newly created heap in EAX. If it fails, the return value in EAX is NULL.
HeapDestroy	Destroys the specified heap object and invalidates its handle. If the function succeeds, the return value in EAX is nonzero.
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.
HeapReAlloc	Reallocates and resizes a block of memory from a heap. If the function succeeds, the return value is a pointer to the reallocated memory block. If the function fails and you have not specified HEAP_GENERATE_EXCEPTIONS, the return value is NULL.
HeapSize	Returns the size of a memory block previously allocated by a call to HeapAlloc or HeapReAlloc. If the function succeeds, EAX contains the size of the allocated memory block, in bytes. If the function fails, the return value is SIZE_T – 1. (SIZE_T equals the maximum number of bytes to which a pointer can point.)

Here is a summary of the heap functions you provided:

GetProcessHeap() returns a handle to the current process's default heap.

HeapCreate() creates a new private heap for the current process.

HeapDestroy() destroys an existing private heap.

HeapAlloc() allocates a block of memory from a heap.

HeapFree() frees a block of memory previously allocated from a heap.

When to use which function:

Use GetProcessHeap() if you are content to use the default heap owned by the current program.

Use HeapCreate() to create a new private heap if you need more control over memory management.

Use HeapDestroy() to destroy a private heap when you are finished using it. Use HeapAlloc() to allocate memory from a heap.

Use HeapFree() to free memory that was allocated from a heap.

Here is an example of how to use the HeapAlloc() and HeapFree() functions to allocate and free a block of memory from a heap:

```
; Create a new private heap.
INVOKE HeapCreate, 0, HEAP_START, HEAP_MAX

; Allocate a block of memory from the heap.
INVOKE HeapAlloc, hHeap, 0, 1000

; Use the allocated memory block.
; ...

; Free the allocated memory block.
INVOKE HeapFree, hHeap, 0, pArray

; Destroy the private heap.
INVOKE HeapDestroy, hHeap
```

It is important to note that dynamic memory allocation should be used carefully to avoid memory leaks. A memory leak occurs when a program allocates memory but does not free it when it is finished using it. Memory leaks can lead to performance problems and eventually cause the program to crash.

Here's the complete program:

```
; Heap Test #1 (Heaptest1.asm)
INCLUDE Irvine32.inc
; This program uses dynamic memory allocation to allocate and
; fill an array of bytes.

.data
ARRAY_SIZE = 1000
FILL_VAL EQU 0FFh
hHeap HANDLE ?
; handle to the process heap
pArray DWORD ?
; pointer to block of memory

.code
main PROC
INVOKE GetProcessHeap
; get handle to the program heap
.IF eax == NULL
; if failed, display message
    call WriteWindowsMsg
    jmp quit
.ELSE
    mov hHeap,eax
    ; success
.ENDIF

```

```
1271 call allocate_array
1272 jnc arrayOk
1273 ; failed (CF = 1)?
1274 call WriteWindowsMsg
1275 call Crlf
1276 jmp quit
1277
1278 arrayOk:
1279 ; ok to fill the array
1280 call fill_array
1281 call display_array
1282 call Crlf
1283 ; free the array
1284 INVOKE HeapFree, hHeap, 0, pArray
1285
1286 quit:
1287     exit
1288
1289 main ENDP
```

```
1291 ;-----------------------------------------------------
1292 allocate_array PROC USES eax
1293 ;
1294 ; Dynamically allocates space for the array.
1295 ; Receives: EAX = handle to the program heap
1296 ; Returns: CF = 0 if the memory allocation succeeds.
1297 ;-----------------------------------------------------
1298 INVOKE HeapAlloc, hHeap, HEAP_ZERO_MEMORY, ARRAY_SIZE
1299 .IF eax == NULL
1300     stc
1301 ; return with CF = 1
1302 .ELSE
1303     mov pArray,eax
1304 ; save the pointer
1305     clc
1306 ; return with CF = 0
1307 .ENDIF
1308     ret
1309
1310 allocate_array ENDP
1311
```

```
1312 ;-----------------------------------------------------
1313 fill_array PROC USES ecx edx esi
1314 ;
1315 ; Fills all array positions with a single character.
1316 ; Receives: nothing
1317 ; Returns: nothing
1318 ;-----------------------------------------------------
1319 mov ecx,ARRAY_SIZE
1320 ; loop counter
1321 mov esi,pArray
1322 ; point to the array
1323 L1:
1324 mov BYTE PTR [esi],FILL_VAL
1325 ; fill each byte
1326 inc esi
1327 ; next location
1328 loop L1
1329 ret
1330
1331 fill_array ENDP
1332
```

```
1333 ;---------------------------------------------------
1334 display_array PROC USES eax ebx ecx esi
1335 ;
1336 ; Displays the array
1337 ; Receives: nothing
1338 ; Returns: nothing
1339 ;---------------------------------------------------
1340 mov ecx,ARRAY_SIZE
1341 ; loop counter
1342 mov esi,pArray
1343 ; point to the array
1344 L1:
1345 mov al,[esi]
1346 ; get a byte
1347 mov ebx,TYPE BYTE
1348 call WriteHexB
1349 ; display it
1350 inc esi
1351 ; next location
1352 loop L1
1353 ret
1354
1355 display_array ENDP
1356
1357 END main
```

The HeapTest1.asm program is an assembly language example that showcases dynamic memory allocation and manipulation in the Windows environment.

The code demonstrates how to allocate memory from the heap, fill that memory with specific values, and display the allocated memory's contents.

The program starts with the .data section, where constants and variables are defined. It specifies the size of the array to be allocated, which is set to 1000 bytes, and the value used to fill the array, which is 0FFh.

The .code section begins with the main procedure. In this procedure, the program performs the following tasks:

It calls the GetProcessHeap function to obtain a handle to the default heap owned by the current process.

This is where memory allocations will be made. If obtaining the heap handle fails (resulting in a NULL handle), the program calls the WriteWindowsMsg function to display an error message and then exits.

If the GetProcessHeap call is successful, the obtained heap handle is stored in the hHeap variable for later use.

The program then proceeds to allocate memory for an array by calling the allocate_array procedure.

If memory allocation fails (indicated by the Carry Flag being set), it calls the WriteWindowsMsg function to display an error message and exits.

If allocation is successful, the pointer to the allocated memory is saved in the pArray variable.

After successful allocation, the program calls the fill_array procedure, which fills the allocated memory with a specified value (0FFh in this case).

Following the memory filling, the program calls the display_array procedure to display the contents of the allocated memory in hexadecimal format.

After displaying the memory contents, the program frees the allocated memory by invoking the HeapFree function.

The program then proceeds to the quit label, where it invokes the Exit system call to terminate the program.

In summary, HeapTest1.asm demonstrates the process of dynamic memory allocation in assembly language within the Windows environment.

It allocates memory from the default process heap, fills that memory with specific values, displays the memory's contents, and finally releases the allocated memory.

The program uses the GetProcessHeap function to obtain the default heap handle and the HeapAlloc and HeapFree functions for memory

allocation and deallocation, respectively.

Let's move on to heaptest2.asm:

```
1360 ; Heap Test #2 (Heaptest2.asm)
1361 INCLUDE Irvine32.inc
1362
1363 .data
1364 HEAP_START = 2000000    ; 2 MByte
1365 HEAP_MAX = 400000000    ; 400 MByte
1366 BLOCK_SIZE = 500000     ; 0.5 MByte
1367 hHeap HANDLE ?
1368 pData DWORD ?
1369 str1 BYTE 0dh, 0ah, "Memory allocation failed", 0dh, 0ah, 0
1370
1371 .code
1372 main PROC
1373     ; Create a new heap with specified size limits
1374     INVOKE HeapCreate, 0, HEAP_START, HEAP_MAX
1375     .IF eax == NULL
1376         ; Failed to create heap
1377         call WriteWindowsMsg
1378         call Crlf
1379         jmp quit
1380     .ELSE
1381         mov hHeap, eax
1382         ; Success: store the heap handle
1383     .ENDIF
1384
1385     mov ecx, 2000    ; Loop counter
1386
```

```
1387 L1:
1388     call allocate_block
1389     ; Allocate a block
1390
1391     .IF Carry?
1392         ; Allocation failed
1393         mov edx, OFFSET str1
1394         ; Display error message
1395         call WriteString
1396         jmp quit
1397     .ELSE
1398         ; Allocation successful
1399         mov al, '.'
1400         ; Show progress with a dot
1401         call WriteChar
1402     .ENDIF
1403
1404     loop L1
1405
1406 quit:
1407     ; Destroy the heap
1408     INVOKE HeapDestroy, hHeap
1409     .IF eax == NULL
1410         ; Failed to destroy heap
1411         call WriteWindowsMsg
1412         call Crlf
1413     .ENDIF
1414
1415     exit
1416 main ENDP
1417
```

```
1417
1418 allocate_block PROC USES ecx
1419     ; Allocate a block and fill it with all zeros
1420     INVOKE HeapAlloc, hHeap, HEAP_ZERO_MEMORY, BLOCK_SIZE
1421     .IF eax == NULL
1422         stc
1423         ; Return with CF = 1 (allocation failed)
1424     .ELSE
1425         mov pData, eax
1426         ; Save the pointer to the allocated memory
1427         clc
1428         ; Return with CF = 0 (allocation succeeded)
1429     .ENDIF
1430     ret
1431 allocate_block ENDP
1432
1433 free_block PROC USES ecx
1434     ; Free a previously allocated block
1435     INVOKE HeapFree, hHeap, 0, pData
1436     ret
1437 free_block ENDP
1438
1439 END main
```

HeapTest2.asm is an assembly program that demonstrates dynamic memory allocation and usage of custom heap management.

It aims to allocate large blocks of memory repeatedly until the specified heap size limit is reached. The code is divided into sections for clarity.

The data section, defined using the .data directive, starts by declaring constants and variables.

HEAP_START is set to 2 megabytes (2MB), representing the initial heap size.

HEAP_MAX is set to 400 megabytes (400MB), indicating the maximum heap size.

BLOCK_SIZE is set to 0.5 megabytes (0.5MB), representing the size of memory blocks to be allocated.

The program uses hHeap to store the handle to the custom heap and pData to hold the pointer to the allocated memory. str1 is a string that will be used to display an error message in case of allocation failure.

The .code section contains the main procedure, labeled main PROC. It begins by invoking the HeapCreate function to create a new heap with specified initial and maximum sizes.

If the creation of the heap fails (resulting in a NULL heap handle), the program calls the WriteWindowsMsg function to display an error message and then jumps to the quit label to exit.

In case of a successful heap creation, the handle to the custom heap is stored in the hHeap variable for later use.

A loop is initiated using ecx as a loop counter, set to 2000 iterations. The purpose of this loop is to repeatedly allocate memory blocks.

Within the loop, the program calls the allocate_block procedure. This procedure uses the HeapAlloc function to allocate memory from the custom heap.

If memory allocation fails (indicated by the Carry Flag being set), the program displays an error message using str1, calls WriteString to print the message, and jumps to the quit label to exit.

If memory allocation is successful, a dot ('.') is displayed on the screen as a progress indicator, indicating a successful memory allocation.

The program continues the loop until all 2000 iterations are completed, each time allocating a memory block.

After the loop finishes, the program reaches the quit label, where it invokes HeapDestroy to destroy the custom heap.

If HeapDestroy fails (returns NULL), an error message is displayed using WriteWindowsMsg, and the program exits using the exit system call.

In summary, HeapTest2.asm showcases dynamic memory allocation using custom heap management. It repeatedly allocates memory blocks until a specified heap size limit is reached.

The program uses functions like HeapCreate, HeapAlloc, and HeapDestroy to manage custom heaps and memory allocation.

Progress is indicated by displaying dots for successful allocations, and any errors are communicated using appropriate error messages.

The program demonstrates the flexibility of heap management in assembly language within the Windows environment.