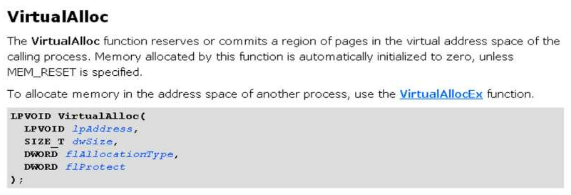
Memoria 3 Mappatura Windows

mercoledì 19 novembre 2025 16:25

Vediamo ora l'allocazione delle pagine (mappandole) in windows (cambiando struttura interna dell'address space)



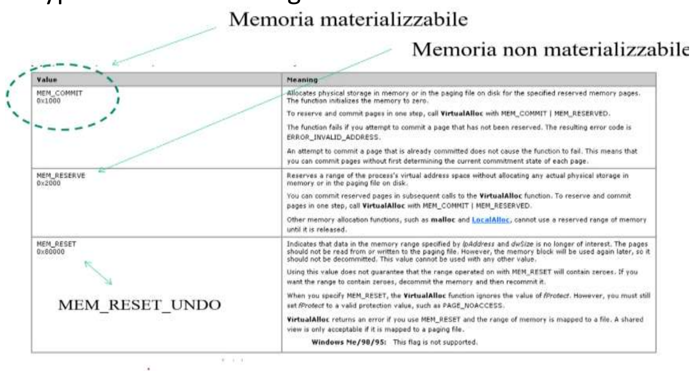
VirtualAlloc

The **VirtualAlloc** function reserves or commits a region of pages in the virtual address space of the calling process. Memory allocated by this function is automatically initialized to zero, unless MEM_RESET is specified.

To allocate memory in the address space of another process, use the VirtualAllocEx function.

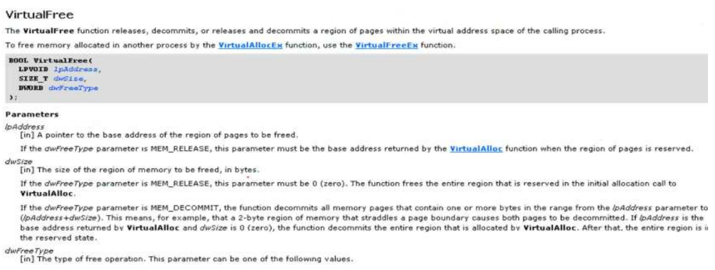
```
LPVOID VirtualAlloc(
  LPVOID lpAddress,
  SIZE_T dwSize,
  DWORD flAllocationType,
  DWORD flProtect
);
```

Dove per fAllocationtype si intendono i seguenti valori:


Memoria materializzabile
Memoria non materializzabile
MEM_RESET_UNDO

Value	Meaning
MEM_COMMIT 0x1000	Allocates physical storage in memory or in the paging file on disk for the specified reserved memory pages. The function initializes the memory to zero. To reserve and commit pages in one step, call **VirtualAlloc** with MEM_COMMIT | MEM_RESERVED. The function fails if you attempt to commit a page that has not been reserved. The resulting error code is ERROR_INVALID_ADDRESS. An attempt to commit a page that is already committed does not cause the function to fail. This means that you can commit pages without first determining the current commitment state of each page.
MEM_RESERVE 0x2000	Reserves a range of the process's virtual address space without allocating any actual physical storage in memory or in the paging file on disk. You can commit reserved pages in subsequent calls to the **VirtualAlloc** function. To reserve and commit pages in one step, call **VirtualAlloc** with MEM_COMMIT | MEM_RESERVED. Other memory allocation functions, such as **malloc** and LocalAlloc, cannot use a reserved range of memory until it is released.
MEM_RESET 0x80000	Indicates that data in the memory range specified by *lpAddress* and *dwSize* is no longer of interest. The pages should not be read from or written to the paging file. However, the memory block will be used again later, so it should not be decommitted. This value cannot be used with any other value. Using this value does not guarantee that the range operated on with MEM_RESET will contain zeroes. If you want the range to contain zeroes, decommit the memory and then recommit it. When you specify MEM_RESET, the **VirtualAlloc** function ignores the value of *flProtect*. However, you must still set *flProtect* to a valid protection value, such as PAGE_NOACCESS. **VirtualAlloc** returns an error if you use MEM_RESET and the range of memory is mapped to a file. A shared view is only acceptable if it is mapped to a paging file. **Windows Me/98/95:** This flag is not supported.

Invece per liberare pagine allocate :



VirtualFree

The **VirtualFree** function releases, decommits, or releases and decommits a region of pages within the virtual address space of the calling process.

To free memory allocated in another process by the VirtualAllocEx function, use the VirtualFreeEx function.

```
BOOL VirtualFree(
  LPVOID lpAddress,
  SIZE_T dwSize,
  DWORD dwFreeType
);
```

Parameters

lpAddress
[in] A pointer to the base address of the region of pages to be freed.

If the *dwFreeType* parameter is MEM_RELEASE, this parameter must be the base address returned by the VirtualAlloc function when the region of pages is reserved.

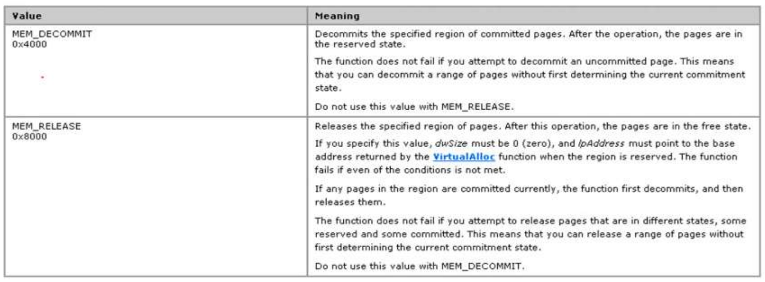
dwSize
[in] The size of the region of memory to be freed, in bytes.

If the *dwFreeType* parameter is MEM_RELEASE, this parameter must be 0 (zero). The function frees the entire region that is reserved in the initial allocation call to **VirtualAlloc**.

If the *dwFreeType* parameter is MEM_DECOMMIT, the function decommits all memory pages that contain one or more bytes in the range from the *lpAddress* parameter to (*lpAddress*+*dwSize*). This means, for example, that a 2-byte region of memory that straddles a page boundary causes both pages to be decommitted. If *lpAddress* is the base address returned by **VirtualAlloc** and *dwSize* is 0 (zero), the function decommits the entire region that is allocated by **VirtualAlloc**. After that, the entire region is i the reserved state.

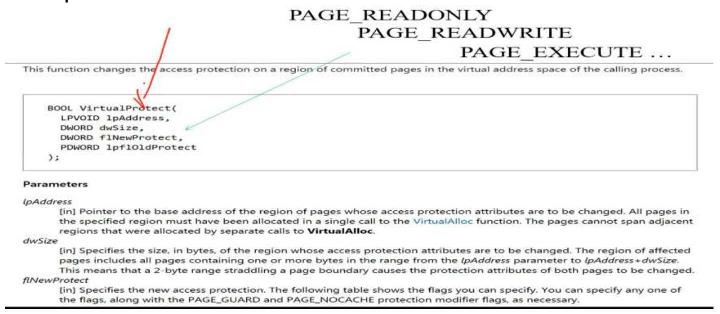
dwFreeType
[in] The type of free operation. This parameter can be one of the following values.

Dove il tipo di gestione può essere :



Value	Meaning
MEM_DECOMMIT 0x4000	Decommits the specified region of committed pages. After the operation, the pages are in the reserved state. The function does not fail if you attempt to decommit an uncommitted page. This means that you can decommit a range of pages without first determining the current commitment state. Do not use this value with MEM_RELEASE.
MEM_RELEASE 0x8000	Releases the specified region of pages. After this operation, the pages are in the free state. If you specify this value, *dwSize* must be 0 (zero), and *lpAddress* must point to the base address returned by the VirtualAlloc function when the region is reserved. The function fails if even of the conditions is not met. If any pages in the region are committed currently, the function first decommits, and then releases them. The function does not fail if you attempt to release pages that are in different states, some reserved and some committed. This means that you can release a range of pages without first determining the current commitment state. Do not use this value with MEM_DECOMMIT.

Mentre per cambiare i permessi :



This function changes the access protection on a region of committed pages in the virtual address space of the calling process.

```
BOOL VirtualProtect(
  LPVOID lpAddress,
  DWORD dwSize,
  DWORD flNewProtect,
  PDWORD lpflOldProtect
);
```

Parameters

lpAddress
[in] Pointer to the base address of the region of pages whose access protection attributes are to be changed. All pages in the specified region must have been allocated in a single call to the VirtualAlloc function. The pages cannot span adjacent regions that were allocated by separate calls to **VirtualAlloc**.

dwSize
[in] Specifies the size, in bytes, of the region whose access protection attributes are to be changed. The region of affected pages includes all pages containing one or more bytes in the range from the *lpAddress* parameter to *lpAddress*+*dwSize*. This means that a 2-byte range straddling a page boundary causes the protection attributes of both pages to be changed.

flNewProtect
[in] Specifies the new access protection. The following table shows the flags you can specify. You can specify any one of the flags, along with the PAGE_GUARD and PAGE_NOCACHE protection modifier flags, as necessary.

Mentre per quanto la struttura degli address space in windows sono le seguenti :

32-bit version default
  2GB user level pages
  2GB kernel level pages

32-bit version with 4GT (4-giga byte tuning)
  3GB user level pages
  1GB kernel level pages

64-bit versions
  altamente variabile dipendendo dalla specifica release/configurazione di Windows

---

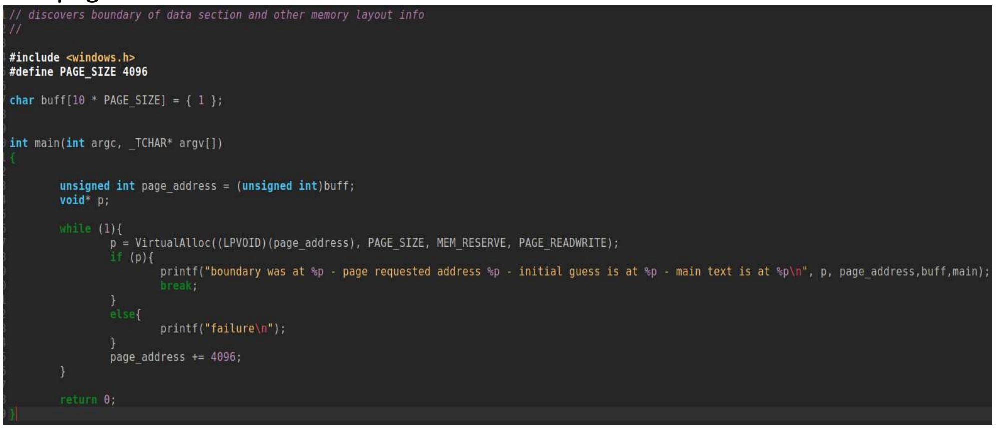
Quindi in generale si ha che :

`brk`, `mmap` e `VirtualAlloc` sono i servizi di back-end (di sistema) su cui si appoggiano **istanze** della libreria `malloc`

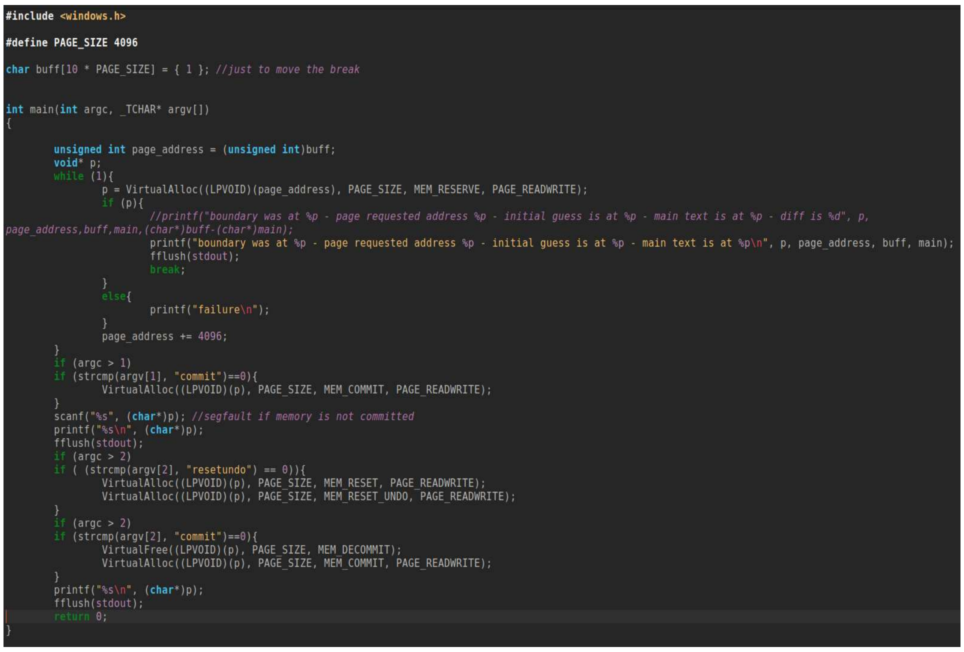
lo schema che malloc adotta e' di <u>prereserving di indirizzi logici (ovvero di aree di memoria logica)</u>

le aree preriservate vengono poi gestite come buffer su cui
- ✓ individuare quali porzioni possono essere date in uso su chiamate alla funzione `malloc`
- ✓ installare i metadati che la libreria `malloc` mantiene per tener traccia di porzioni libere ed occupate – gia' consegnate su chiamate precedenti alla funzione `malloc`

Vediamo ora gli esempi :

1. Creazione pagine
   a. 

```c
// discovers boundary of data section and other memory layout info
//

#include <windows.h>
#define PAGE_SIZE 4096

char buff[10 * PAGE_SIZE] = { 1 };


int main(int argc, _TCHAR* argv[])
{

        unsigned int page_address = (unsigned int)buff;
        void* p;

        while (1){
                p = VirtualAlloc((LPVOID)(page_address), PAGE_SIZE, MEM_RESERVE, PAGE_READWRITE);
                if (p){
                        printf("boundary was at %p - page requested address %p - initial guess is at %p - main text is at %p\n", p, page_address,buff,main);
                        break;
                }
                else{
                        printf("failure\n");
                }
                page_address += 4096;
        }

        return 0;
}
```

   b. Inizialmente fallisce , ma poi una volta raggiunto il boundary <u>delle pagine già allocate</u>, ne restituisce una ulteriore
2. Mem reserve commit
   a. 

```c
#include <windows.h>

#define PAGE_SIZE 4096

char buff[10 * PAGE_SIZE] = { 1 }; //just to move the break


int main(int argc, _TCHAR* argv[])
{

        unsigned int page_address = (unsigned int)buff;
        void* p;
        while (1){
                p = VirtualAlloc((LPVOID)(page_address), PAGE_SIZE, MEM_RESERVE, PAGE_READWRITE);
                if (p){
                        //printf("boundary was at %p - page requested address %p - initial guess is at %p - main text is at %p - diff is %d", p,
page_address,buff,main,(char*)buff-(char*)main);
                        printf("boundary was at %p - page requested address %p - initial guess is at %p - main text is at %p\n", p, page_address, buff, main);
                        fflush(stdout);
                        break;
                }
                else{
                        printf("failure\n");
                }
                page_address += 4096;
        }
        if (argc > 1)
        if (strcmp(argv[1], "commit")==0){
                VirtualAlloc((LPVOID)(p), PAGE_SIZE, MEM_COMMIT, PAGE_READWRITE);
        }
        scanf("%s", (char*)p); //segfault if memory is not committed
        printf("%s\n", (char*)p);
        fflush(stdout);
        if (argc > 2)
        if ( (strcmp(argv[2], "resetundo") == 0)){
                VirtualAlloc((LPVOID)(p), PAGE_SIZE, MEM_RESET, PAGE_READWRITE);
                VirtualAlloc((LPVOID)(p), PAGE_SIZE, MEM_RESET_UNDO, PAGE_READWRITE);
        }
        if (argc > 2)
        if (strcmp(argv[2], "commit")==0){
                VirtualFree((LPVOID)(p), PAGE_SIZE, MEM_DECOMMIT);
                VirtualAlloc((LPVOID)(p), PAGE_SIZE, MEM_COMMIT, PAGE_READWRITE);
        }
        printf("%s\n", (char*)p);
        fflush(stdout);
        return 0;
}
```

   b. Se solo commit come prima e poi appena crea la pagina leggo e scrivo .
   c. Se passo come secondo parametro "commit" prima decommit -> anonima a contenuto nullo e poi ci scrivo