

# ***Gestão de memória na WIN32***

- Jeffrey Richter, Christophe Nasarre, **Windows via C/C++**, Fifth Edition, Microsoft Press, 2008  
**[cap. 13, 14, 15, 16, 17 e 18]**
- Microsoft, **Microsoft Developer's Network** (MSDN)

- No Windows existem três mecanismos para manipular a memória:
  - Memória virtual
    - Dedicada à manipulação de grandes vectores de dados
  - Ficheiros mapeados
    - Dedicados à manipulação de grande *streams* de dados e partilha de dados entre processos na mesma máquina
  - *Heaps*
    - Mais dedicados à gestão de um grande número de objectos criados dinamicamente

## Windows – Gestão de Memória - Windows 32 bits

---

- 4 GB espaço de endereçamento virtual por processo
  - 2GB espaço utilizador e 2GB espaço de sistema
    - Em algumas versões pode-se ter 3GB espaço utilizador e 1GB espaço de sistema
  - Paginação a pedido com *clustering* (uma falta de página traz a página em causa e mais algumas na vizinhança (entre 1 a 8))
  - Substituição de páginas local baseado no modelo de *working set*
- As páginas virtuais do processo podem estar no seguintes estados:
  - Free - Não atribuída ao processo. O acesso a estas páginas origina um *page fault*
  - *Reserved* - página reservada no espaço de endereçamento e para poder ser utilizada tem de existir um pedido explícito de *commit*. A referência a uma página neste estado origina um *page fault*.
  - *Committed* – página associada ao processo possuído dados ou código, pode estar em memória física ou disco

## Windows – Gestão de Memória – Espaço de endereçamento virtual

---

Partição	Windows 98
NULL-Pointer Assignment	0x00000000 0x00000FFF
DOS/16-bit Windows Application Compatibility	0x00001000 0x003FFFFFFF
User-Mode	0x00400000 0x7FFFFFFF
64-KB Off-Limits	NA
Shared Memory-Mapped File	0x80000000 0xBFFFFFFF
Kernel-Mode	0xC0000000 0xFFFFFFFF

## Windows – Gestão de Memória – Espaço de endereçamento virtual

---

Partição	32-Bit Windows 2000(x86 and Alpha)
NULL-Pointer Assignment	0x00000000 0x0000FFFF
DOS/16-bit Windows Application Compatibility	NA
User-Mode	0x00010000 0x7FFEFFFE
64-KB Off-Limits	0x7FFF0000 0x7FFFFFFF
Shared Memory-Mapped File	NA
Kernel-Mode	0x80000000 0xFFFFFFFF

## Windows – Gestão de Memória – Espaço de endereçamento virtual

---

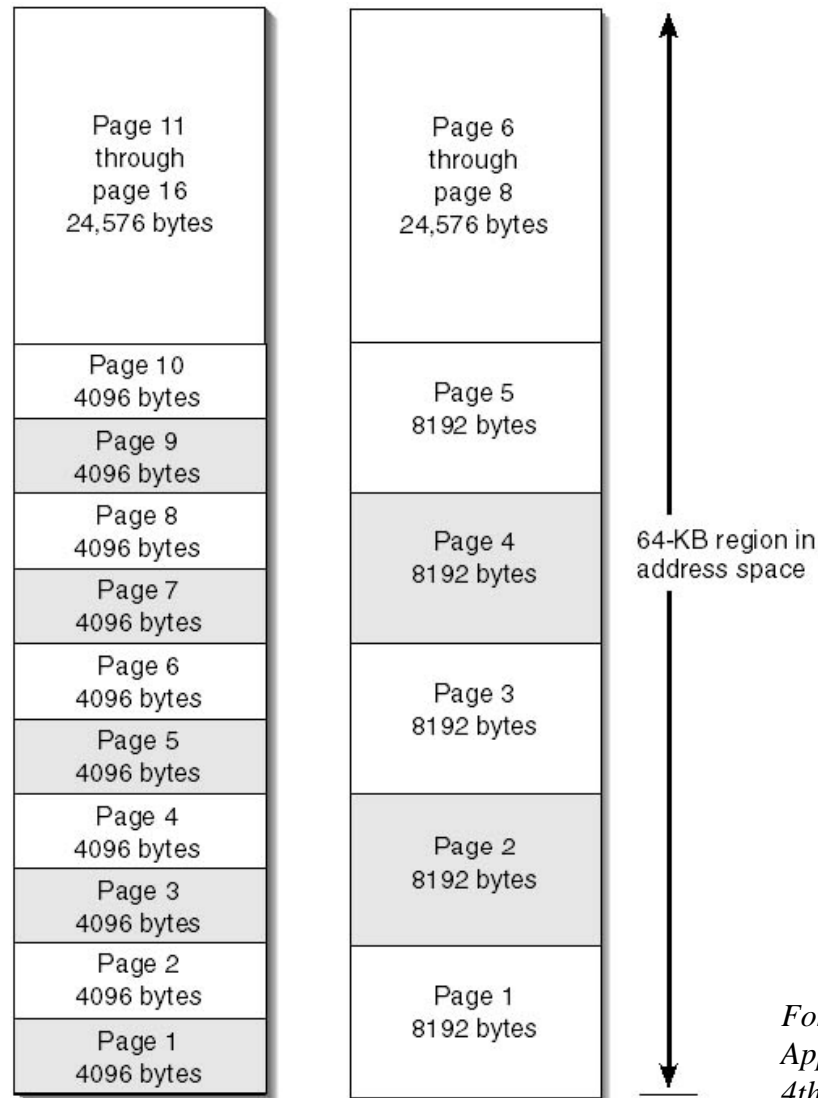
```
VOID GetSystemInfo( LPSYSTEM_INFO lpSystemInfo );
```

Permite obter informações sobre o sistema computacional como a arquitectura e tipo de processador, o número de processadores do sistema, a dimensão das paginas de memória, etc.

```
typedef struct _SYSTEM_INFO {  
    union {  
        DWORD dwOemId;  
        struct {  
            WORD wProcessorArchitecture;  
            WORD wReserved;  
        };  
    };  
  
    DWORD dwPageSize;  
    LPVOID lpMinimumApplicationAddress;  
    LPVOID lpMaximumApplicationAddress;  
    DWORD_PTR dwActiveProcessorMask;  
    DWORD dwNumberOfProcessors;  
    DWORD dwProcessorType;  
    DWORD dwAllocationGranularity;  
    WORD wProcessorLevel;  
    WORD wProcessorRevision;  
} SYSTEM_INFO;
```

## Windows – Gestão de Memória – Espaço de endereçamento virtual

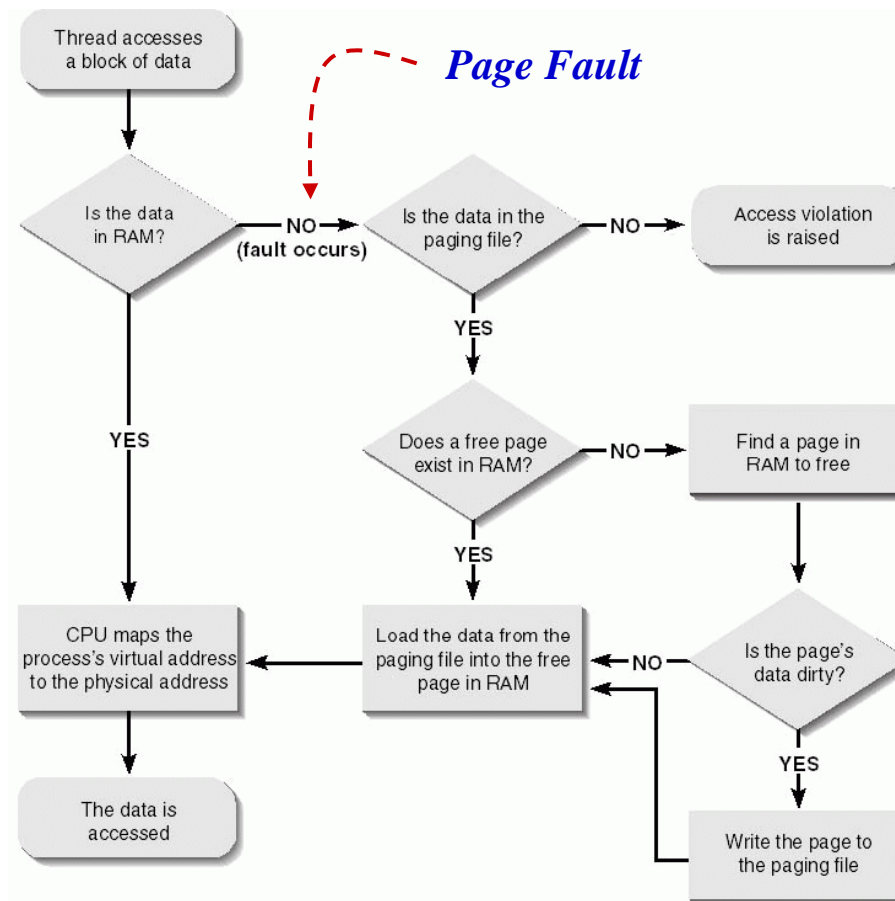
**Espaço de endereçamento de um processo para diferentes CPU**



*Fonte: Jeffrey Richter, Programming Applications for Microsoft Windows, 4th edition, cap. 13*

## Windows – Gestão de Memória – *Page Fault*

### Conversão de endereço virtual em endereço físico



Fonte: Jeffrey Richter, *Programming Applications for Microsoft Windows*, 4th edition, cap. 13

O Windows utiliza esta técnica para quase tudo. Não há obrigatoriamente apenas uma área de *Swap* ou *Paging File*, podem existir várias. Na realidade o Windows quando executa uma aplicação em vez de carregar o programa em memória apenas reserva o espaço de endereçamento para o mesmo e usa como *Paging File* o ficheiro executável em disco.

**Nota:** Esta técnica não é válida para programas situados em *Floppy Disks*.



## Windows – Gestão de Memória – VirtualAlloc

---

Quando um processo necessita de memória tem necessidade de realizar dois passos:

1. **reservar uma região de memória e**
2. **atribuir armazenamento físico na região (*Commit*)**

Para estas operações existe a a função de sistema `VirtualAlloc`

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

- ***lpAddress*** Especifica o endereço de início da região a atribuir (*allocate*). Se este argumento for NULL o sistema determina onde reservar a região.
- ***dwSize*** define a dimensão da região em bytes
- ***flAllocationType*** tipo de atribuição: MEM\_COMMIT, MEM\_RESET, MEM\_RESERVE, etc.
- ***flProtect*** tipo de atribuição de protecção de acesso: PAGE\_READONLY, PAGE\_READWRITE, PAGE\_EXECUTE, PAGE\_EXECUTE\_READ, PAGE\_EXECUTE\_READWRITE, PAGE\_GUARD, etc.

## Windows – Gestão de Memória – VirtualFree

---

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

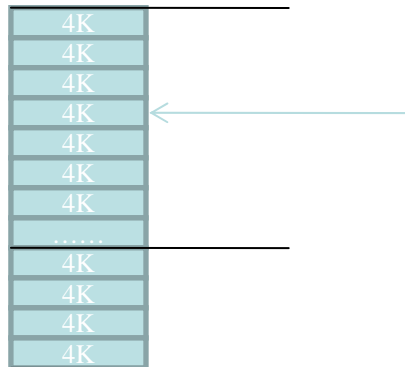
- ***lpAddress*** Especifica o endereço de início da região a libertar.
- ***dwSize*** define a dimensão da região em bytes
- ***dwFreeType*** indica o tipo de operação:
  - MEM\_DECOMMIT liberta o armazenamento associado à região indicada
  - MEM\_RELEASE Anula a reserva da região indicada. Neste caso a dimensão deve ser zero (*dwSize*) e o endereço *lpAddress* deve ser o endereço devolvido pela função *VirtualAlloc()*

## Windows – Gestão de Memória – VirtualQuery

---

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

- **lpAddress** Especifica o endereço de início da região a libertar.
- **dwSize** define a dimensão da região em bytes
- **dwFreeType** indica o tipo de operação:
  - MEM\_DECOMMIT liberta o armazenamento associado à região indicada
  - MEM\_RELEASE Anula a reserva da região indicada. Neste caso a dimensão deve ser zero (dwSize) e o endereço lpAddress deve ser o endereço devolvido pela função VirtualAlloc( )



## Windows – Gestão de Memória – *Windows Task Manager*

### Process working set

Memória física atribuída ao processo. Esta dimensão inclui as páginas de memória partilhadas

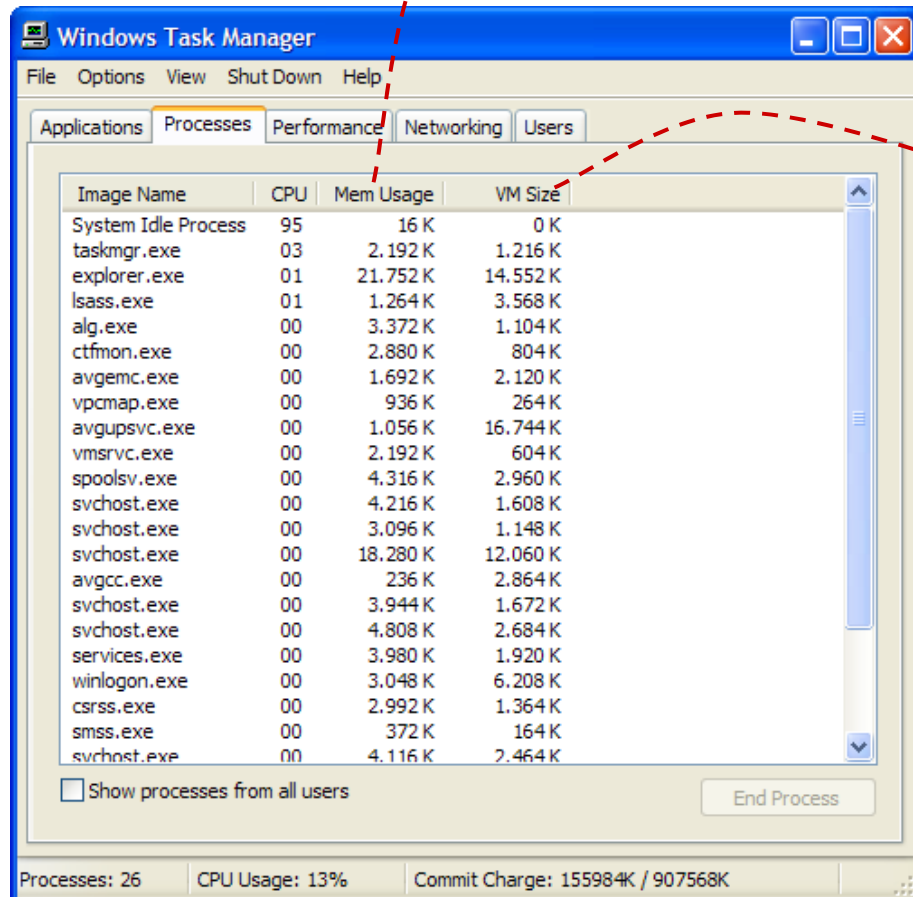


Image Name	CPU	Mem Usage	VM Size
System Idle Process	95	16 K	0 K
taskmgr.exe	03	2.192 K	1.216 K
explorer.exe	01	21.752 K	14.552 K
lsass.exe	01	1.264 K	3.568 K
alg.exe	00	3.372 K	1.104 K
ctfmon.exe	00	2.880 K	804 K
avgemc.exe	00	1.692 K	2.120 K
vpcmap.exe	00	936 K	264 K
avgupsvc.exe	00	1.056 K	16.744 K
vmsvc.exe	00	2.192 K	604 K
spoolsv.exe	00	4.316 K	2.960 K
svchost.exe	00	4.216 K	1.608 K
svchost.exe	00	3.096 K	1.148 K
svchost.exe	00	18.280 K	12.060 K
avgcc.exe	00	236 K	2.864 K
svchost.exe	00	3.944 K	1.672 K
svchost.exe	00	4.808 K	2.684 K
services.exe	00	3.980 K	1.920 K
winlogon.exe	00	3.048 K	6.208 K
csrss.exe	00	2.992 K	1.364 K
smss.exe	00	372 K	164 K
svchost.exe	00	4.116 K	2.464 K

☐ Show processes from all users

End Process

Processes: 26   CPU Usage: 13%   Commit Charge: 155984K / 907568K

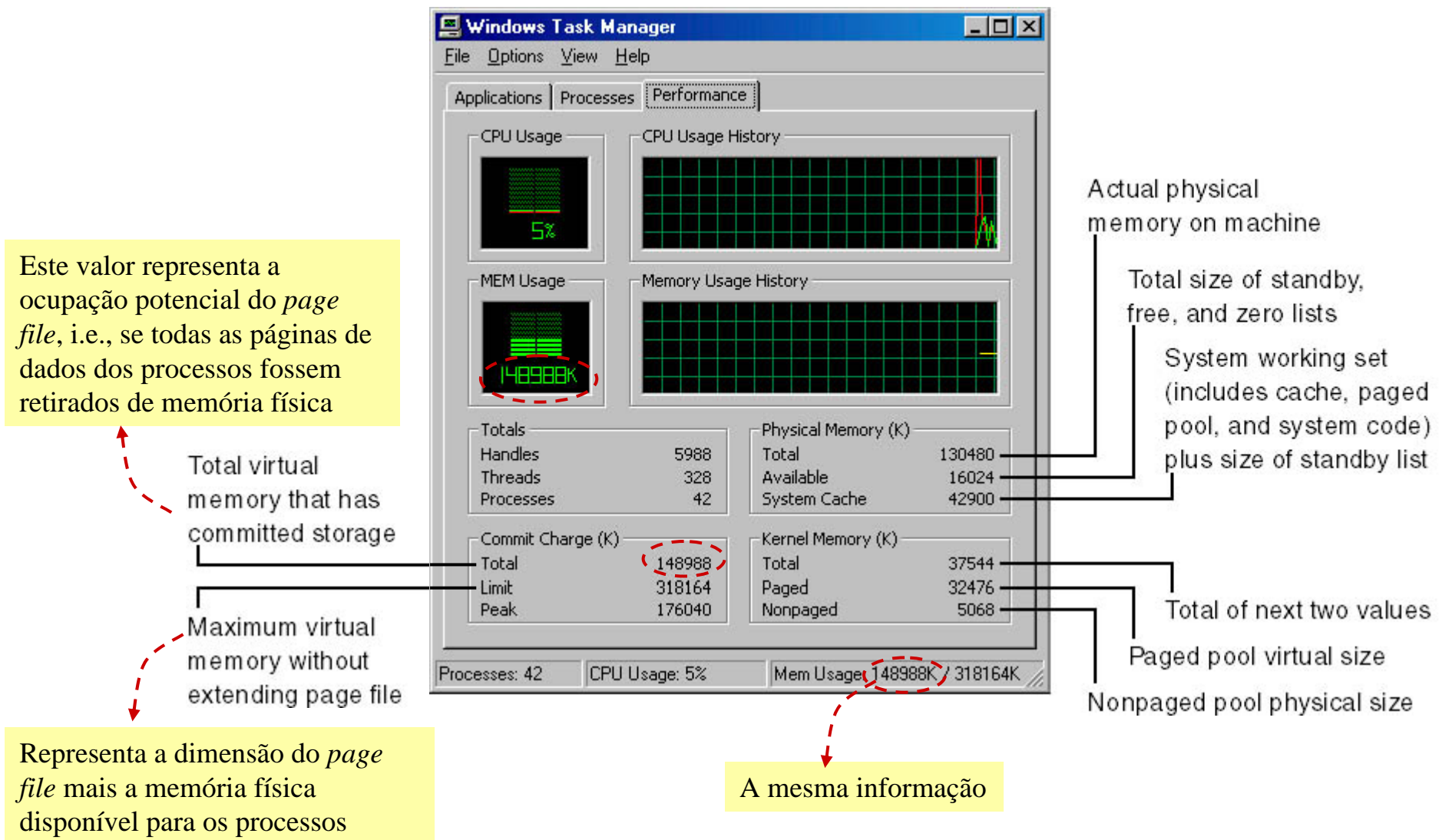
### Process private bytes

Não representa a totalidade da memória virtual do processo, apenas a dimensão da memória virtual privada (não partilhada) – Apenas os dados não inclui o código.

A mesma informação obtida através do contador de desempenho *Private Bytes* da ferramenta *performance tool*

A existência de fugas de memória (*memory leaks*) podem ser identificadas analisando este valor

## Windows – Gestão de Memória – *Windows Task Manager*



## Windows – Gestão de Memória – gestão das *frames*

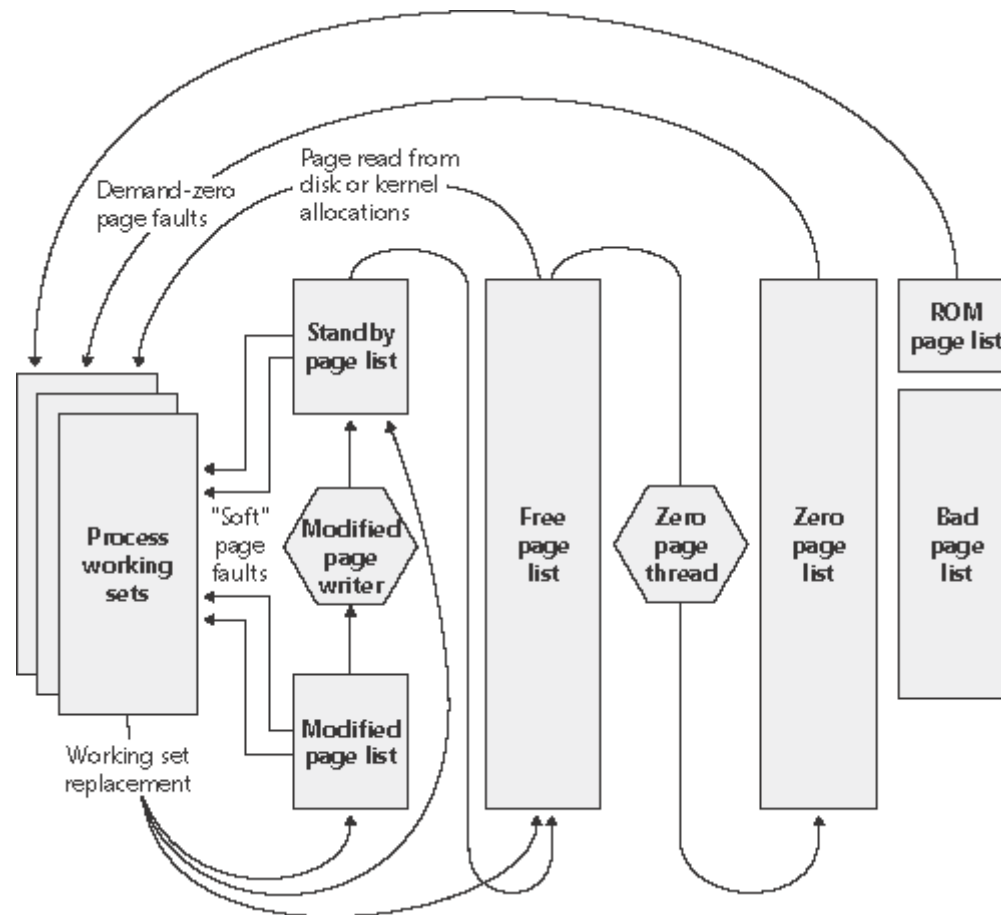




Diagrama de estado das páginas físicas

# Exemplo BigArray

*Utilize as ferramentas  
ProcessInfo, VMMap e taskmanager  
para acompanhar a execução do programa*

## Windows – Gestão de Memória – *Threads e Stack*

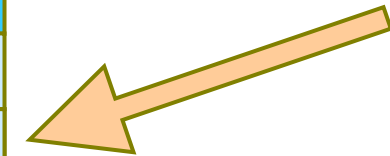
---

Endereço de Memória	Estado da página
0x080FF000	<i>Top of stack: Committed</i>
0x080FE000	<i>Committed com atributo de guarda</i>
0x080FD000	Reservada
	
	
0x08003000	Reservada
0x08002000	Reservada
0x08001000	Reservada
0x08000000	<i>Bottom of stack: Reserved</i>



## Windows – Gestão de Memória – *Threads e Stack*

Endereço de Memória	Estado da página
0x080FF000	<i>Top of stack: Committed</i>
0x080FE000	<i>Committed</i>
0x080FD000	<i>Committed com atributo de guarda</i>
0x08003000	Reservada
0x08002000	Reservada
0x08001000	Reservada
0x08000000	<i>Bottom of stack: Reserved</i>



Acesso a  
esta página...





... gera uma exceção  
(SEH) tratada pelo S.O.  
que efectua:

1. Retira o atributo de guarda desta página
2. Realize o *committed* da página seguinte com atributo de guarda

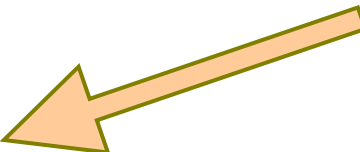
## Windows – Gestão de Memória – *Threads e Stack*

---

Endereço de Memória	Estado da página
0x080FF000	<i>Top of stack: Committed</i>
0x080FE000	<i>Committed</i>
0x080FD000	<i>Committed</i>
	
	
0x08003000	<i>Committed</i>
0x08002000	<i>Committed com atributo de guarda</i>
0x08001000	<i>Reservada</i>
0x08000000	<i>Bottom of stack: Reserved</i>

## Windows – Gestão de Memória – *Threads e Stack*

Endereço de Memória	Estado da página
0x080FF000	<i>Top of stack: Committed</i>
0x080FE000	<i>Committed</i>
0x080FD000	<i>Committed</i>
0x08003000	<i>Committed</i>
0x08002000	<i>Committed</i>
0x08001000	<i>Committed</i>
0x08000000	<i>Bottom of stack: Reserved</i>

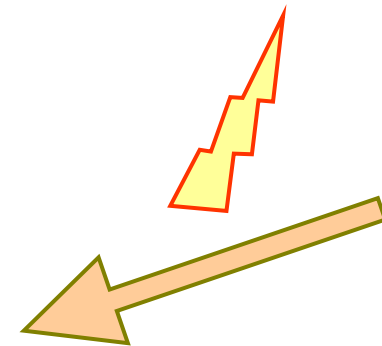


Como se trata da penúltima página o atributo de guarda não é activado  
Quando é feito o *committed* desta página o sistema gera a excepção  
EXCEPTION\_STACK\_OVERFLOW

## Windows – Gestão de Memória – *Threads e Stack*

Endereço de Memória	Estado da página
0x080FF000	<i>Top of stack: Committed</i>
0x080FE000	<i>Committed</i>
0x080FD000	<i>Committed</i>
0x08003000	<i>Committed</i>
0x08002000	<i>Committed</i>
0x08001000	<i>Committed</i>
0x08000000	<i>Bottom of stack: Reserved</i>

Gera a  
excepção (SEH)  
*access violation*  
tratada pelo S.O.  
que termina o processo



*Ficheiros Mapeados*

## Windows – Gestão de Memória - Ficheiros *Mapeados*

---

- Dedicados à manipulação de grande streams de dados e partilha de dados entre processos na mesma máquina
- Esta é a mais flexível forma de partilha de dados em memória entre vários processos, utilizada pela grande maioria das técnicas de comunicação entre processos, inclusive as fornecidos pelo próprio sistema.
- Área de Memória que é visível e partilhada por múltiplos processos.
- É possível definir a memória partilhada como “Copy-On-Write”.
- Método:
  - Criar ou abrir um ficheiro usando **CreateFile** ou **OpenFile**. O handle desse ficheiro será usado no próximo passo. Se o objectivo é apenas criar uma área de memória partilhada, não se cria nenhum ficheiro, e como handle usa-se o valor `INVALID_HANDLE_VALUE` para `CreateFileMapping`, neste caso é criado pelo sistema um ficheiro temporário.
  - Criar um objecto de sistema para mapeamento do ficheiro usando **CreateFileMapping** ou, se já criado, abri-lo usando `OpenFileMapping`. Especifica-se a dimensão do ficheiro e o tipo de acesso.
  - Mapear, parcialmente ou totalmente, o objecto a memória partilhada no espaço de memória do processo usando **MapViewOfFile**.

## Windows – Gestão de Memória - Ficheiros *Mapeados*

---

```
HANDLE CreateFileMapping( HANDLE hFile, LPSECURITY_ATTRIBUTES lpsa,  
                          DWORD fdwProtect, DWORD dwMaximumSizeHigh,  
                          DWORD dwMaximumSizeLow, LPTSTR lpszMapName );
```

- hFile : Handle do Ficheiro Aberto. Se o valor passado for INVALID\_HANDLE\_VALUE, o sistema usará um ficheiro temporário interno para a partilha residente no *paging file* do sistema
- fdwProtect
  - PAGE\_READONLY
  - PAGE\_READWRITE
  - PAGE\_WRITECOPY
- dwMaximumSizeHigh, dwMaximumSizeLow
  - If zero, MaximumSize = File Size
- lpszMapName
  - Para ser usado em OpenFileMapping, se <> NULL

## Windows – Gestão de Memória - Ficheiros *Mapeados*

---

HANDLE **OpenFileMapping**( DWORD dwDesiredAccess, BOOL bInheritHandle,  
LPTSTR lpszMapName );

- dwDesiredAccess
  - FILE\_MAP\_READ
  - FILE\_MAP\_WRITE, FILE\_MAP\_ALL\_ACCESS
  - FILE\_MAP\_COPY

LPVOID **MapViewOfFile**( HANDLE hFileMapping, DWORD dwDesiredAccess,  
DWORD dwFileOffsetHigh, DWORD dwFileOffsetLow,  
DWORD dwNumberOfBytesToMap );

- dwFileOffsetHigh, dwFileOffsetLow
  - Offset no Ficheiro onde começa o mapeamento da Memória (0 = início do ficheiro).
- dwNumberOfBytesToMap
  - N° de bytes do ficheiro mapeados em memória (0 = Todo o ficheiro).



## Windows – Gestão de Memória - Ficheiros *Mapeados*

---

BOOL UnmapViewOfFile( LPCVOID lpBaseAddress );

- Remove o mapeamento da memória, escrevendo todas as alterações novamente no ficheiro (na altura mais conveniente).

BOOL FlushViewOfFile( LPVOID lpBaseAddress,  
DWORD dwNumberOfBytesToFlush );

- Para actualizar imediatamente o ficheiro com as alterações efectuadas em memória

```
HANDLE hFile, hMapping;  
LPVOID lpBaseAddress;  
  
hFile = CreateFile (TEXT("Ficheiro.ext"), ...);  
hMapping = CreateFileMapping (hFile, NULL, PAGE_READWRITE, 0, 0, TEXT("Map_Ficheiro"));  
lpBaseAddress = MapViewOfFile (hMapping, FILE_MAP_ALL_ACCESS, 0, 0, 0);  
  
CloseHandle (hFile);      // Podemos fechar imediatamente os handles, porque MapViewOfFile  
CloseHandle (hMapping); // ja' incrementou o usage count dos objectos  
...  
UnmapViewOfFile (lpBaseAddress); // Termina o mapeamento e escreve as alterações no ficheiro
```

## Windows – Gestão de Memória - Ficheiros *Mapeados* – Exemplo

```
BOOL printFile(const char *fileName) {
    HANDLE hFile = CreateFile(
        fileName,                // file name
        GENERIC_READ,            // access mode
        0,                        // share
        NULL,                    // Security
        OPEN_EXISTING,           // how to create
        FILE_ATTRIBUTE_NORMAL,    // file attributes
        NULL                      // handle to template file
    );
    if (hFile == INVALID_HANDLE_VALUE) {
        ReportErrorSystem("Erro na abertura do ficheiro <%s>:", fileName);
        return FALSE;
    }

    HANDLE fileMapHandle = CreateFileMapping(
        hFile,                    // handle to file
        NULL,                    // security
        PAGE_READONLY,            // protection
        0,                        // high-order DWORD of size
        0,                        // low-order DWORD of size
        NULL                      // object name
    );
    if (fileMapHandle == NULL) {
        ReportErrorSystem("Erro na abertura do FileMap");
        CloseHandle(hFile);
        return FALSE;
    }
}
```

```
char *lpMapAddress = (char *)MapViewOfFile(
    fileMapHandle,                // Handle to mapping object
    FILE_MAP_READ,               // Read/write permission
    0,                           // dwFileOffsetHigh
    0,                           // dwFileOffsetLow
    0);                           // dwNumberOfBytesToMap
if (lpMapAddress == NULL) {
    ReportErrorSystem("Erro no MapViewOfFile.");
    CloseHandle(hFile); CloseHandle(fileMapHandle);
    return FALSE;
}
DWORD fileSize = GetFileSize(hFile, NULL);
if (fileSize == INVALID_FILE_SIZE) {
    ReportErrorSystem("Erro no GetFileSize.");
    UnmapViewOfFile(lpMapAddress);
    CloseHandle(hFile); CloseHandle(fileMapHandle);
    return FALSE;
}
for (DWORD i = 0; i < fileSize; ++i) putchar(lpMapAddress[i]);

// libertar recursos
UnmapViewOfFile(lpMapAddress);
CloseHandle(fileMapHandle);
CloseHandle(hFile);
return TRUE;
} // end printFile
```

*Heaps*

## Windows – Gestão de Memória – *Heaps*

---

- Os *heaps* são vocacionados para “*alocar*” muitos blocos de pequena dimensão (listas, árvores)
- Os *heaps* tem a vantagem de esconder os detalhes da granularidade de atribuição e limites das páginas
- Os *heaps* tem a desvantagem de a atribuição e libertação de memória serem mais lentas que os outros mecanismos (`VirtualAlloc`, etc) e não existe controlo do *commit* e *uncommit* de memória física
- Internamente um *heap* é uma região reservada do espaço de endereçamento não necessitando que a totalidade das suas páginas estejam *commit*. À medida que existem pedidos de atribuição de memória ao gestor do *heap* é que a memória física vai sendo *commit*.
- Quando um processo é criado o sistema cria um *heap* no seu espaço de endereçamento. Este *heap* é o *heap* por omissão de um processo (*process default heap*). O *handle* para o *heap* de omissão do processo pode ser obtido através da função `GetProcessHeap()`
- O acesso às funções dos *heaps* são *thread safe* (embora se possa desactivar esta característica no caso do *heap* só ser utilizado por um *thread* bem determinado)

## Windows – Gestão de Memória – *Heaps*: HeapCreate

---

```
HANDLE HeapCreate( DWORD fOptions,  
                  SIZE_T dwInitialSize,  
                  SIZE_T dwMaximumSize );
```

- ***fOptions*** altera o comportamento das operações no *heap*. Pode ter o valor zero ou uma combinação do seguintes valores:
  - HEAP\_GENERATE\_EXCEPTIONS é gerada um exceção (SEH) quando são pedidos blocos ao gestor do *heap* e esse não pode ser satisfeito
  - HEAP\_NO\_SERIALIZE Não é utilizado o acesso exclusivo nas operações sobre o *heap*
- ***dwInitialSize*** dimensão inicial em bytes
- ***dwMaximumSize*** dimensão máxima em bytes. No caso de ser zero o *heap* torna-se num *heap* dinâmico que pode crescer.

## Windows – Gestão de Memória – *Heaps*: HeapAlloc

---

```
LPVOID HeapAlloc( HANDLE hHeap,  
                  DWORD dwFlags,  
                  SIZE_T dwBytes );
```

- **hHeap** *handle* do *heap* onde se quer alocar o bloco de memória
- **dwFlags** sobrepõe-se ao indicando na função HeapCreate e pode ser uma combinação dos seguintes valores
  - HEAP\_GENERATE\_EXCEPTIONS Em situação de erro pode ser gerada uma destas excepções:
    - STATUS\_NO\_MEMORY - a “alocação” falhou devido a falta de memória
    - STATUS\_ACCESS\_VIOLATION - a “alocação” falhou devido a uma corrupção do *heap* ou parâmetros errados passados a esta função
  - HEAP\_NO\_SERIALIZE
  - HEAP\_ZERO\_MEMORY a memória “alocada” é iniciada a zero.
- **dwBytes** dimensão do bloco

**Nota:** Existem as funções

- HeapReAlloc que permite alterar a dimensão de um *heap*. Para mais detalhes consulte o MSDN.
- HeapSize que permite determinar a dimensão de um bloco atribuído

## Windows – Gestão de Memória – *Heaps*: HeapFree

---

```
BOOL HeapFree( HANDLE hHeap,  
               DWORD dwFlags,  
               LPVOID lpMem );
```

- ***hHeap*** handle do heap
- ***dwFlags*** se for utilizado `HEAP_NO_SERIALIZE` a função acede ao *heap* sem utilizar exclusão mútua
- ***lpMem*** apontador de memória devolvido pela função `HeapAlloc` e `HeapReAlloc`.

```
BOOL HeapDestroy( HANDLE hHeap );
```

Elimina o *heap* referenciado pelo *handle* *hHeap*.

- Razões para a criação de heaps adicionais
  - Protecção entre componentes
  - Gestão de memória mais eficiente
  - Acesso local
  - Evitar a necessidade de sincronização entre tarefas
  - Rápida libertação da memória