第十课　内存管理

一、内存空间

1. 地址空间

地址空间是程序可寻址的最大范围。
32位操作系统的地址空间为[0, 4G)，2^32=4G。
地址空间越大，编写程序越容易。

2. 地址空间的划分

1) 用户地址空间：[0, 2G)

A. 存放用户的程序代码和数据。
　　用户空间中的程序代码不能直接访问(可以通过WIN32 APIs间接访问)
　　内核空间中的程序代码和数据。

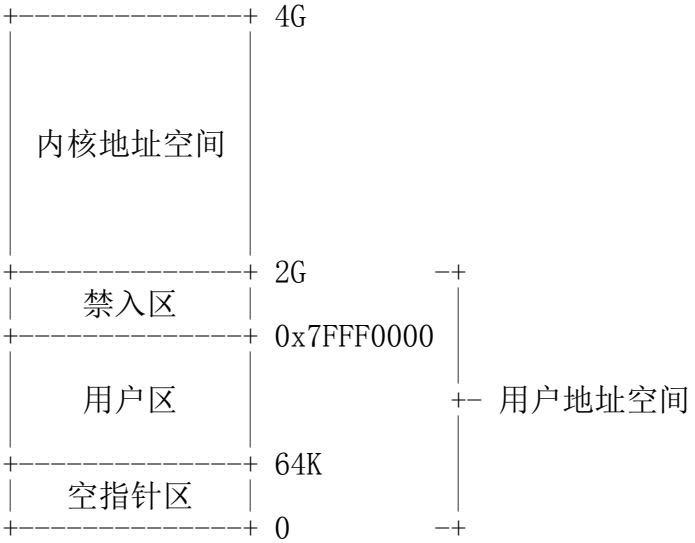
B. 空指针区：NULL区，[0, 64K)
　　系统将地址值小于64K的指针都认作空指针。

C. 用户区：[64K, 0x7FFF0000)

D. 禁入区：[0x7FFF0000, 2G)，2G-0x7FFF0000=64K

2) 内核地址空间：[2G, 4G)

存放内核程序代码和数据，如系统驱动等。
内核地址空间的程序代码可以访问用户地址空间的程序代码和数据。



3. 区域

区域就是一块连续的内存。
区域的大小一般为64K或64K的整数倍。
每个区域都有特定的状态：

1) 空闲：未被使用
2) 私有：已被预定
3) 映像：存放代码
4) 映射：存放数据

4. 物理内存

半导体内存，内存条。
系统可以使用的实际内存。
CPU可以直接访问的内存。

5. 分页文件

将磁盘文件(pagefile.sys)虚拟成内存使用。
CPU如果要访问其中的数据，必须先将该数据通过换页读到物理内存中。
物理内存空间不够时，同样通过换页将暂时不用的数据写到分页文件中。

6. 虚拟内存

相对于每个进程而言的，独立且连续的虚拟地址空间。
具体每个地址单元，究竟映射到物理内存或分页文件的哪个字节，
完全由操作系统动态管理和维护，对程序员完全透明。

7. 内存页

系统管理内存的最小单位。内存页大小为4K字节。
每个内存页都有特定的权限。

8. 页目表

32位地址：

```
31       22 21       12 11          0
XXXXXXXXXX XXXXXXXXXX XXXXXXXXXXXX
   10位        10位         12位
 2^10=1K     2^10=1K      2^12=4K
   页目         页表          页
```

页目包含1K项，每项对应一个页表，页表包含1K项，每项对应一个页，
页包含4K项，每项对应一个字节。32位地址空间：1K*1K*4K=4G。

9. 内存访问

1) 首先根据虚拟内存地址在物理内存中查找对应的位置。
   如果找到，则访问其中的数据，否则执行2)；

2) 其次根据虚拟内存地址在分页文件中查找对应的位置。
   如果没找到(野指针)，则返回错误，否则执行3)；

3) 将与该虚拟地址对应的，分页文件中的内存页换入物理内存，
   同时将原物理内存页换出到分页文件，执行4)；

4) 访问物理内存中的数据。

10. 内存分配

1) 虚拟内存分配：分配大内存(1M以上)。

并非特定在物理内存或分页文件中分配。
小内存也可以用这种方式分配，但常用于分配大内存。

2) 堆内存分配：分配小内存(1M以下)。

malloc/new。
大内存也可以用这种方式分配，
但内存块之间会产生内存空洞，造成浪费。

3) 栈内存分配：操作系统自动维护。

二、虚拟内存
---

1. 特点

速度快，分配大内存效率高。将内存和地址的分配分别执行。
可以先分配地址，直到需要时再绑定(提交)到内存。
常用于大型电子表格等的处理。

2. 分配虚拟内存

```
LPVOID VirtualAlloc (
    LPVOID lpAddress,         // NULL或绑定(提交)地址
    SIZE_T dwSize,            // 内存大小(以字节为单位)
    DWORD  flAllocationType, // 分配方式
    DWORD  flProtect          // 访问方式，一般使用PAGE_READWRITE
);
```

成功返回虚拟内存地址，失败返回NULL。

flAllocationType取值：
    MEM_COMMIT - 在物理内存或分页文件中分配地址同时绑定(提交)到内存
                  如买现房，有地址有房子
    MEM_RESERVE - 只分配地址不绑定(提交)到内存
                  如买期房，有地址无房子

获取内存状态：

```
void GlobalMemoryStatus (
    LPMEMORYSTATUS lpBuffer // 内存状态信息
```

```
);

typedef struct _MEMORYSTATUS {
    DWORD  dwLength;        // 结构体字节数
    DWORD  dwMemoryLoad;    // 内存使用率，百分之几
    SIZE_T dwTotalPhys;     // 物理内存总字节数
    SIZE_T dwAvailPhys;     // 空闲物理内存字节数
    SIZE_T dwTotalPageFile; // 分页文件总字节数
                            // 并非pagefile.sys文件大小，
                            // 最多可写入多少字节
    SIZE_T dwAvailPageFile; // 空闲分页文件字节数
    SIZE_T dwTotalVirtual;  // 虚拟内存(地址空间)总字节数
    SIZE_T dwAvailVirtual;  // 空闲虚拟内存(地址空间)字节数
}   MEMORYSTATUS, *LPMEMORYSTATUS;
```

3. 释放虚拟内存

```
BOOL VirtualFree (
    LPVOID lpAddress, // 虚拟内存地址
    SIZE_T dwSize,    // 释放字节数，0表示全部释放
    DWORD  dwFreeType // 释放方式
);
```

成功返回TRUE，失败返回FALSE。

dwFreeType取值：
    MEM_DECOMMIT - 解绑定(反提交)，只释放内存，不释放地址
    MEM_RELEASE - 同时释放内存和地址，dwSize参数必须取0

注意：内存绑定(提交)以页(4096字节)为单位。

```
+------------+------------+
|            |            |
|    4096    |    4096    |
|            |            |
+------------+------------+
^ ^          ^^ ^          ^
A B          CD E          F
```

在A/B/C处提交，VirtualAlloc函数返回的地址与A一样。
在D/E/F处提交，VirtualAlloc函数返回的地址比A大4096。

范例：WinVirtual

三、堆内存

1. 特点

适合小内存分配，一般小于1M内存。
一般每个程序都有自己的堆，默认大小为1M，会根据使用情况进行调整。

每个进程都有默认的堆空间，malloc/new从这些堆中分配堆内存。

2. 获取调用进程的首个堆

```
HANDLE GetProcessHeap (void);
```

成功返回调用进程首个堆的句柄，失败返回NULL。

3. 获取调用进程的所有堆

```
DWORD GetProcessHeaps (
    DWORD   NumberOfHeaps, // 堆句柄数组容量，
                           // 即最多能容纳几个句柄
    PHANDLE ProcessHeaps   // 堆句柄数组
);
```

成功返回调用进程堆的个数，堆句柄放在ProcessHeaps数组中，
失败返回0。

4. 创建堆

```
HANDLE HeapCreate (
    DWORD flOptions,      // 创建选项
    DWORD dwInitialSize, // 初始字节数
    DWORD dwMaximumSize  // 最大字节数，0表示无限大
);
```

成功返回堆句柄，失败返回NULL。

flOptions取值：
    HEAP_GENERATE_EXCEPTIONS - 从堆中分配内存失败抛出异常
    HEAP_NO_SERIALIZE   - 支持对堆内存的不连续存取

5. 分配堆内存

```
LPVOID HeapAlloc (
    HANDLE hHeap,   // 堆句柄
    DWORD  dwFlags, // 分配方式
    DWORD  dwBytes  // 内存大小(以字节为单位)
);
```

成功返回堆内存地址，失败返回NULL。

dwFlags为以下值的位或：
    HEAP_GENERATE_EXCEPTIONS
        - 从堆中分配内存失败抛出异常 \
    HEAP_NO_SERIALIZE   > 调HeapCreate函数时
        - 支持对堆内存的不连续存取   / 已指定的此处可略
    HEAP_ZERO_MEMORY
        - 初始化清零

6. 释放堆内存

```
BOOL HeapFree (
    HANDLE hHeap,    // 堆句柄
    DWORD  dwFlags, // 释放方式，只能取HEAP_NO_SERIALIZE
    LPVOID lpMem    // 堆内存地址
);
```

成功返回TRUE，失败返回FALSE。

7. 销毁堆

```
BOOL HeapDestroy (
    HANDLE hHeap // 堆句柄
);
```

成功返回TRUE，失败返回FALSE。

当堆被销毁时，其中的所有堆内存自动被释放。

8. Windows平台的malloc/new实现实际就是调用了上述堆函数

范例：WinHeap

四、栈内存
---

1. 每个线程都有自己的栈，默认大小为1M字节。

2. 操作系统自动维护栈内存的分配与释放。

3. Windows提供了_alloca函数，用于在栈上分配内存。

五、内存映射文件
---

1. 基本概念

1) 将文件映射成内存来使用，即以访问内存的方式操作文件。

2) 常用于进程间通信，比直接通过文件I/O进行进程间通信更高效。

2. 创建文件

用CreateFile创建，可读可写GENERIC_READ | GENERIC_WRITE。

3. 创建映射

```
HANDLE CreateFileMapping (
    HANDLE  hFile,              // 文件句柄
    LPSECURITY_ATTRIBUTES lpAttributes,
                                // 安全属性，NULL
    DWORD   flProtect,          // 访问方式，PAGE_READWRITE
```

```
                                // 可读可写
    DWORD   dwMaximumSizeHigh, // 映射总字节数高32位
    DWORD   dwMaximumSizeLow,  // 映射总字节数低32位
    LPCTSTR lpName             // 映射名，NULL表示匿名映射，
                               // 其它进程无法访问
);
```

成功返回映射句柄，失败返回NULL。

4. 加载映射

```
LPVOID MapViewOfFile (
    HANDLE hFileMappingObject,  // 映射句柄
    DWORD  dwDesiredAccess,     // 访问方式，FILE_MAP_ALL_ACCESS
                                // 可读可写
    DWORD  dwFileOffsetHigh,    // 偏移量高32位 \
                                                > 合成值必须是
    DWORD  dwFileOffsetLow,     // 偏移量低32为 /  区域粒度(64K)的整数倍
    SIZE_T dwNumberOfBytesToMap // 映射的字节数，
                                // 不能超过映射总字节数
);
```

成功返回映射地址，失败返回NULL。

5. 使用映射

以内存方式使用映射。

6. 卸载映射

```
BOOL UnmapViewOfFile (
    LPCVOID lpBaseAddress // 映射地址
);
```

成功返回TRUE，失败返回FALSE。

7. 关闭映射

```
CloseHandle
```

8. 关闭文件

```
CloseHandle
```

9. 打开映射

```
HANDLE OpenFileMapping (
    DWORD   dwDesiredAccess, // 访问方式，FILE_MAP_ALL_ACCESS
                             // 可读可写
    BOOL    bInheritHandle,  // 子进程是否可以继承
```

```
                                    // 此函数返回的映射句柄
    LPCTSTR lpName                  // 映射名
);
```

成功返回映射句柄，失败返回NULL。

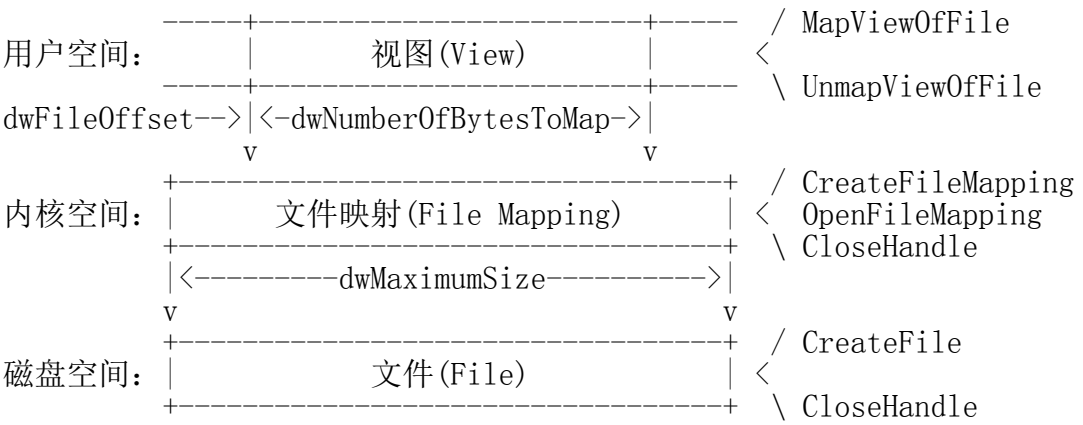
10. 基于内存映射文件的进程间通信

1) 写进程

创建文件->创建映射->加载映射->写入映射->卸载映射->关闭映射->关闭文件

2) 读进程

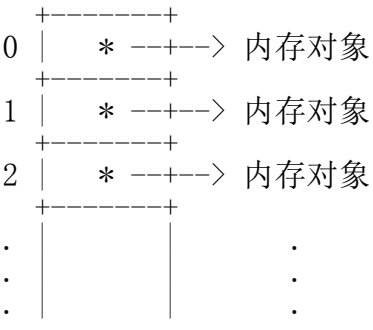
打开映射->加载映射->读取映射->卸载映射->关闭映射



范例：WinMap、WinRead

六、句柄

句柄表：



句柄就是内存对象地址在句柄表中的索引。
通过句柄不能直接访问内存，只能通过APIs函数操作其所标识的对象。