# 内存管理

```
班级: 2015211306
学号: 2015211301
姓名: 魏  晓
时间: 2018.01.10
```

- **实验亮点**
  - 重新用C++实现虚拟内存管理
  - 重构核心代码段,具体见下文

---

## 实验目的

> 在本次实验中，需要从不同的侧面了解 Windows 2000/XP 的虚拟内存机制。在 Windows 2000/XP 操作系统中，可以通过一些 API 操纵虚拟内存。主要需要了解以下几方面：
> - Windows 2000/XP 虚拟存储系统的组织
> - 如何控制虚拟内存空间
> - 如何编写内存追踪和显示工具
> - 详细了解与内存相关的API 函数的使用

## Windows 2000/XP 虚拟内存机制简介

- 内存管理是Windows2000/XP 执行体的一部分，位于Ntoskrnl.exe 文件中，是整个操作系统的重要组成部分。
- 默认情况下，32 位Windows 2000/XP 上每个用户进程可以占有2GB 的私有地址空间，操作系统占有剩下的2GB。Windows 2000/XP 在x86 体系结构上利用二级页表结构来实现虚拟地址向物理地址的变换。一个32 位虚拟地址被解释为三个独立的分量——页目录索引、页表索引和字节索引——它们用于找出描述页面映射结构的索引。页面大小及页表项的宽度决定了页目录和页表索引的宽度。比如，在x86 系统中，因为一页包含4096 字节，于是字节索引被确定为12 位宽（212 = 4096）。
- 应用程序有三种使用内存方法：
  - 以页为单位的虚拟内存分配方法，适合于大型对象或结构数组；
  - 内存映射文件方法，适合于大型数据流文件以及多个进程之间的数据共享；

  - 内存堆方法，适合于大量的小型内存申请。
- 本次实验主要是针对第一种使用方式。应用程序通过API 函数 VirtualAlloc 和VirtualAllocEx 等实现以页为单位的虚拟内存分配方法。首先保留地址空间，然后向此地址空间提交物理页面，也可以同时实现保留和提交。保留地址空间是为线程将来使用保留一块虚拟地址。在已保留的区域中，提交页面必须指出将物理存储器提交到何处以及提交多少。提交页面在访问时会转变为物理内存中的有效页面。

## 相关的API 函数

> - 可以通过GetSystemInfo，GlobalMemoryStatus 和VirtualQuery 来查询进程虚空间的状态。主要的信息来源如下：
>
> ```
> VOID GetSystemInfo  （ LPSYSTEM_INFO lpSystemInfo ）；
> 结构SYSTEMINFO 定义如下：
> typedef struct _SYSTEM_INFO {
> DWORD dwOemld;
> DWORD dwPageSize;
> LPVOID lpMinimumApplicationAddress;
> LPVOID lpMaximumApplicationAddress;
> DWORD dwActiveProcessorMask;
> DWORD dwNumberOfProcessors;
> DWORD dwProcessorType;
> DWORD dwAllocationGranularity;
> DWORD dwReserved;
> } SYSTEM_INFO, *LPSYSTEM_INFO;
> ```
>
> - 函数VOID GlobalMemoryStatus （LPMEMORYSTATUS lpBuffer）；
> - 数据结构MEMORYSTATUS 定义如下：
>
> ```
> typedef struct _ MEMORYSTATUS {
> DWORD dwLength;
> DWORD dwMemoryLoad;
> DWORD dwTotalPhys;
> DWORD dwAvailPhys;
> DWORD dwTotalPageFile;
> DWORD dwAvailPageFile;
> DWORD dwTotalVirtual;
> DWORD dwAvailVirtual;
> } MEMORYSTATUS, * LPMEMORYSTATUS;
> ```

> - 函数DWORD VirtualQuery （ LPCVOLD lpAddress,PMEMORY_BASIC_INFORMATION lpBuffer, DWORD dwLength）；
> - 主要数据结构MEMORY_BASIC_INFORMATION 定义如下：
>
> ```
> typedef struct _ MEMORY_BASIC_INFORMATION {
> PVOID BaseAddress;
> PVOID AllocationBase;
> DWORD AllocationProtect;
> DWORD RegionSize;
> DWORD State;
> DWORD Protect;
> DWORD Type;
> } MEMORY_BASIC_INFORMATION;
> typedef MEMORY_BASIC_INFORMATION * PMEMORY_BASIC_INFORMATION;
> ```
>
> - 还有一些函数，例如VirtualAlloc，VirtualAllocEx，VirtualFree 和VirtualFreeEx 等，用 于虚拟内存的管理，详情请见Microsoft 的Win32 API Reference Manual。

## 实验内容

> 使用 Win32 API 函数，编写一个包含两个线程的进程，一个线程用于模拟内存分配活动，一个线程用于跟踪第一个线程的内存行为。模拟内存活动的线程可以从一个文件中读出要进行的内存操作，每个内存操作包含如下内容：
> - 时间：开始执行的时间；
> - 块数：分配内存的粒度；
> - 操作：包括保留一个区域、提交一个区域、释放一个区域、回收一个区域以及锁与解锁一个区域；可以将这些操作编号，存放于文件中。
> - 大小：指块的大小；
> - 访问权限：共五种PAGE_READONLY、PAGE_READWRITE、PAGE_EXCUTE、PAGE_EXECUTE_READ 和PAGE_ EXECUTE_READWRITE。可以将这些权限编号，存放于文件中。
> - 跟踪线程将页面大小、已使用的地址范围、物理内存总量以及虚拟内存总量等信息显示出来。

## 实验程序

- 在编译运行原实验样例程序之后,改进了其中的管理策略，重新使用C++编写了实验程序
- 代码见同文件夹下

## 核心代码

```
auto dwError = GetLastError ();
            HLOCAL hlocal = NULL;
            DWORD systemLocale = MAKELANGID (LANG_NEUTRAL, SUBLANG_NEUTRAL);
            BOOL fOk = FormatMessageA (
                FORMAT_MESSAGE_FROM_SYSTEM | FORMAT_MESSAGE_IGNORE_INSERTS |
                FORMAT_MESSAGE_ALLOCATE_BUFFER,
                NULL, dwError, systemLocale,
                (char *) &hlocal, 0, NULL);
            if (!fOk)
            {
                // Is it a network-related error?
                HMODULE hDll = LoadLibraryEx (TEXT ("netmsg.dll"), NULL,DONT_RESOLVE_DLL_REFERENCES);
                if (hDll != NULL)
                {
                    fOk = FormatMessageA (
                        FORMAT_MESSAGE_FROM_HMODULE | FORMAT_MESSAGE_IGNORE_INSERTS |
                        FORMAT_MESSAGE_ALLOCATE_BUFFER,
                        hDll, dwError, systemLocale,
                        (char *) &hlocal, 0, NULL);
                    FreeLibrary (hDll);
                }
            }

            if (fOk && (hlocal != NULL))
            {
                os << (const char *) LocalLock (hlocal) << std::endl;
                LocalFree (hlocal);
            }
```

## 实验结果

- 原实验程序使用方法
    - 首先执行makefile.exe,生成opfile文件，里面保存了模拟的内存操作。
    - 然后执行memory-op.exe,产生两个线程，一个从opfile文件里读取内存操作，模拟内存活动，另一个跟踪第一个的内存行为，将结果输出，并保存在out.txt文件中。两个线程通过信号量实现同步。
- 原实验-结果

```
0:reserve now
starting address:0x20000        size:12288
1:reserve now
starting address:0x160000       size:4096
2:reserve now
starting address:0x170000       size:20480
3:reserve now
starting address:0x190000       size:16384
4:reserve now
starting address:0x1a0000       size:20480
5:commit now
starting address:0x20000        size:12288
6:commit now
starting address:0x160000       size:4096
7:commit now
starting address:0x170000       size:20480
8:commit now
starting address:0x190000       size:16384
9:commit now
starting address:0x1a0000       size:20480
10:lock now
starting address:0x20000        size:12288
998
11:lock now
starting address:0x160000       size:4096
12:lock now
starting address:0x170000       size:20480
998
13:lock now
starting address:0x190000       size:16384
14:lock now
starting address:0x1a0000       size:20480
15:unlock now
starting address:0x20000        size:12288
158
16:unlock now
starting address:0x160000       size:4096
17:unlock now
starting address:0x170000       size:20480
158
18:unlock now
starting address:0x190000       size:16384
19:unlock now
starting address:0x1a0000       size:20480
```

```
20:decommit now
starting address:0x20000        size:12288
87
21:decommit now
starting address:0x160000       size:4096
22:decommit now
starting address:0x170000       size:20480
87
23:decommit now
starting address:0x190000       size:16384
24:decommit now
starting address:0x1a0000       size:20480
87
25:release now
starting address:0x20000        size:12288
26:release now
starting address:0x160000       size:4096
27:release now
starting address:0x170000       size:20480
28:release now
starting address:0x190000       size:16384
29:release now
starting address:0x1a0000       size:20480

Process returned 0 (0x0)   execution time : 16.191 s
Press any key to continue.
```

---

- 修改后运行结果

```
                PageSize 4096

Case: PAGE_READONLY
Reverse
             BaseAddress 00090000
          AllocationBase 00090000
       AllocationProtect PAGE_NOACCESS
   RegionSize / PageSize 1
                 Protect PAGE_UNKNOWN
                   State MEM_RESERVE
                    Type MEM_PRIVATE
Commit
             BaseAddress 00090000
          AllocationBase 00090000
       AllocationProtect PAGE_NOACCESS
   RegionSize / PageSize 1
                 Protect PAGE_READONLY
                   State MEM_COMMIT
                    Type MEM_PRIVATE
Lock
             BaseAddress 00090000
          AllocationBase 00090000
       AllocationProtect PAGE_NOACCESS
   RegionSize / PageSize 1
                 Protect PAGE_READONLY
                   State MEM_COMMIT
                    Type MEM_PRIVATE
Unlock
             BaseAddress 00090000
          AllocationBase 00090000
       AllocationProtect PAGE_NOACCESS
   RegionSize / PageSize 1
                 Protect PAGE_READONLY
                   State MEM_COMMIT
                    Type MEM_PRIVATE
Decommit
             BaseAddress 00090000
          AllocationBase 00090000
       AllocationProtect PAGE_NOACCESS
   RegionSize / PageSize 1
                 Protect PAGE_UNKNOWN
                   State MEM_RESERVE
                    Type MEM_PRIVATE
Release
             BaseAddress 00090000
          AllocationBase 00000000
       AllocationProtect PAGE_UNKNOWN
   RegionSize / PageSize 16
                 Protect PAGE_NOACCESS
                   State MEM_FREE
                    Type MEM_UNKNOWN_TYPE

Case: PAGE_READWRITE
Reverse
             BaseAddress 00090000
          AllocationBase 00090000
       AllocationProtect PAGE_NOACCESS
   RegionSize / PageSize 2
                 Protect PAGE_UNKNOWN
```

```
                   State MEM_RESERVE
                    Type MEM_PRIVATE
Commit
             BaseAddress 00090000
          AllocationBase 00090000
       AllocationProtect PAGE_NOACCESS
   RegionSize / PageSize 2
                 Protect PAGE_READWRITE
                   State MEM_COMMIT
                    Type MEM_PRIVATE
Lock
             BaseAddress 00090000
          AllocationBase 00090000
       AllocationProtect PAGE_NOACCESS
   RegionSize / PageSize 2
                 Protect PAGE_READWRITE
                   State MEM_COMMIT
                    Type MEM_PRIVATE
Unlock
             BaseAddress 00090000
          AllocationBase 00090000
       AllocationProtect PAGE_NOACCESS
   RegionSize / PageSize 2
                 Protect PAGE_READWRITE
                   State MEM_COMMIT
                    Type MEM_PRIVATE
Decommit
             BaseAddress 00090000
          AllocationBase 00090000
       AllocationProtect PAGE_NOACCESS
   RegionSize / PageSize 2
                 Protect PAGE_UNKNOWN
                   State MEM_RESERVE
                    Type MEM_PRIVATE
Release
             BaseAddress 00090000
          AllocationBase 00000000
       AllocationProtect PAGE_UNKNOWN
   RegionSize / PageSize 16
                 Protect PAGE_NOACCESS
                   State MEM_FREE
                    Type MEM_UNKNOWN_TYPE

Case: PAGE_EXECUTE
Reverse
```

```
             BaseAddress 00090000
          AllocationBase 00090000
       AllocationProtect PAGE_NOACCESS
   RegionSize / PageSize 3
                 Protect PAGE_UNKNOWN
                   State MEM_RESERVE
                    Type MEM_PRIVATE
Commit
             BaseAddress 00090000
          AllocationBase 00090000
       AllocationProtect PAGE_NOACCESS
   RegionSize / PageSize 3
                 Protect PAGE_EXECUTE
                   State MEM_COMMIT
                    Type MEM_PRIVATE
Lock
             BaseAddress 00090000
          AllocationBase 00090000
       AllocationProtect PAGE_NOACCESS
   RegionSize / PageSize 3
                 Protect PAGE_EXECUTE
                   State MEM_COMMIT
                    Type MEM_PRIVATE
Unlock
             BaseAddress 00090000
          AllocationBase 00090000
       AllocationProtect PAGE_NOACCESS
   RegionSize / PageSize 3
                 Protect PAGE_EXECUTE
                   State MEM_COMMIT
                    Type MEM_PRIVATE
Decommit
             BaseAddress 00090000
          AllocationBase 00090000
       AllocationProtect PAGE_NOACCESS
   RegionSize / PageSize 3
                 Protect PAGE_UNKNOWN
                   State MEM_RESERVE
                    Type MEM_PRIVATE
Release
             BaseAddress 00090000
          AllocationBase 00000000
       AllocationProtect PAGE_UNKNOWN
   RegionSize / PageSize 16
                 Protect PAGE_NOACCESS
```

```
                   State MEM_FREE
                    Type MEM_UNKNOWN_TYPE

Case: PAGE_EXECUTE_READ
Reverse
             BaseAddress 00090000
          AllocationBase 00090000
       AllocationProtect PAGE_NOACCESS
   RegionSize / PageSize 4
                 Protect PAGE_UNKNOWN
                   State MEM_RESERVE
                    Type MEM_PRIVATE
Commit
             BaseAddress 00090000
          AllocationBase 00090000
       AllocationProtect PAGE_NOACCESS
   RegionSize / PageSize 4
                 Protect PAGE_EXECUTE_READ
                   State MEM_COMMIT
                    Type MEM_PRIVATE
Lock
             BaseAddress 00090000
          AllocationBase 00090000
       AllocationProtect PAGE_NOACCESS
   RegionSize / PageSize 4
                 Protect PAGE_EXECUTE_READ
                   State MEM_COMMIT
                    Type MEM_PRIVATE
Unlock
             BaseAddress 00090000
          AllocationBase 00090000
       AllocationProtect PAGE_NOACCESS
   RegionSize / PageSize 4
                 Protect PAGE_EXECUTE_READ
                   State MEM_COMMIT
                    Type MEM_PRIVATE
Decommit
             BaseAddress 00090000
          AllocationBase 00090000
       AllocationProtect PAGE_NOACCESS
   RegionSize / PageSize 4
                 Protect PAGE_UNKNOWN
                   State MEM_RESERVE
                    Type MEM_PRIVATE
Release
```

```
             BaseAddress 00090000
          AllocationBase 00000000
       AllocationProtect PAGE_UNKNOWN
   RegionSize / PageSize 16
                 Protect PAGE_NOACCESS
                   State MEM_FREE
                    Type MEM_UNKNOWN_TYPE

Case: PAGE_EXECUTE_READWRITE
Reverse
             BaseAddress 00090000
          AllocationBase 00090000
       AllocationProtect PAGE_NOACCESS
   RegionSize / PageSize 5
                 Protect PAGE_UNKNOWN
                   State MEM_RESERVE
                    Type MEM_PRIVATE
Commit
             BaseAddress 00090000
          AllocationBase 00090000
       AllocationProtect PAGE_NOACCESS
   RegionSize / PageSize 5
                 Protect PAGE_EXECUTE_READWRITE
                   State MEM_COMMIT
                    Type MEM_PRIVATE
Lock
             BaseAddress 00090000
          AllocationBase 00090000
       AllocationProtect PAGE_NOACCESS
   RegionSize / PageSize 5
                 Protect PAGE_EXECUTE_READWRITE
                   State MEM_COMMIT
                    Type MEM_PRIVATE
Unlock
             BaseAddress 00090000
          AllocationBase 00090000
       AllocationProtect PAGE_NOACCESS
   RegionSize / PageSize 5
                 Protect PAGE_EXECUTE_READWRITE
                   State MEM_COMMIT
                    Type MEM_PRIVATE
Decommit
             BaseAddress 00090000
          AllocationBase 00090000
       AllocationProtect PAGE_NOACCESS
   RegionSize / PageSize 5
                 Protect PAGE_UNKNOWN
                   State MEM_RESERVE
                    Type MEM_PRIVATE
Release
             BaseAddress 00090000
          AllocationBase 00000000
       AllocationProtect PAGE_UNKNOWN
   RegionSize / PageSize 16
                 Protect PAGE_NOACCESS
                   State MEM_FREE
                    Type MEM_UNKNOWN_TYPE
```

# 实验结论

- 现在多数的计算机页大小为 4 KB (4096 Byte)
- `VirtualQuery` 可以查看虚拟内存分配情况;
  - `BaseAddress` 表示查询的内存基址（64 位）；
  - `AllocationAddress` 表示系统为内存分配的基址（64 位）；
  - `AllocationProtect` 表示申请内存时的访问保护;
  - `RegionSize` 表示申请区域的大小（字节为单位）；
  - `Protect` 表示当前的内存访问保护;
  - `State` 表示当前内存分配状态;
  - `Type` 表示当前内存分配的类型;
- `VirtualAlloc` 可以申请系统分配虚拟内存;
  - 使用 `MEM_RESERVE` 可以申请**保留**一段空间;
  - 使用 `MEM_COMMIT` 表示**提交**申请，获取保留的空间;
- `VirtualLock` 用于**锁定**虚拟内存于物理内存中，保证之后对其访问 *不引起缺页中断*;
- `VirtualUnlock` **取消**刚刚的**锁定**;
- `VirtualFree` 释放申请的虚拟内存;
  - 使用 `MEM_DECOMMIT` 可以**撤销提交**，返回保留状态;
  - 使用 `MEM_RELEASE` 用于**撤销保留**，其他进程可以申请使用这块内存。

# 实验感想

- 经过这次试验,我对win32虚拟内存机制有了更深的理解,比如内存的相关API函数的用法,详细理解了windowsXP的内存管理

- 在这次试验中,我把以前学到的计算机组成原理的知识和操作系统的知识都用到了,体会到了虚拟内存对于现代操作系统的巨大意义
- 这次试验也锻炼了我windows系统编程的能力