北京邮电大学　计算机学院

《操作系统概念》实验报告

姓名　　王睿嘉

学号 2015211906

班级 2015211307

实验 内存管理

一、 实验内容和环境描述

1. 实验目的

从不同侧面了解 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。主要需要探究以下几方面：

1）Windows 2000/XP 虚拟存储系统的组织；

2）如何控制虚拟内存空间；

3）如何编写内存追踪和显示工具；

4）详细了解与内存相关的 API 函数的使用。

2. 实验要求

使用 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，另一个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：

1）时间：开始执行的时间；

2）块数：分配内存的粒度；

3）操作：包括保留、提交、释放、回收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。

4）大小：块的大小；

5）访问权限：共五种 PAGE_READONLY、PAGE_READWRITE、PAGE_EXECUTE、PAGE_EXECUTE_READ 和 PAGE_ EXECUTE_READWRITE。可以将这些权限编号，存放于文件中。

3. 环境描述

编程语言：C++

集成开发环境：VS2015

二、 输入输出定义

1. 输入定义

用户无需自行输入数据。

2. 输出定义

程序自动生成opfile文件，保存模拟的内存操作，并产生两个线程，一个从opfile文件里读取操作，模拟内存活动，另一个跟踪第一个的内存行为，将结果输出至out.txt文件中。与此同时，在命令行界面打印一定信息，以显示内存操作流程及状态。

三、 实验设计

1. 整体说明

应用程序有三种使用内存的方法：

1）以页为单位的虚拟内存分配方法，适合大型对象或结构数组；

2）内存映射文件方法，适合大型数据流文件及多个进程间的数据共享；

3）内存堆方法，适合大量的小型内存申请。

本次实验主要针对第一种使用方式。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。

需要区分6个内存操作的含义：

保留（reserve），指保留进程的虚拟地址空间，而不分配物理存储空间；

提交（commit），指在内存中分配物理存储空间；

释放（release），指将物理内存和虚拟地址空间全部释放；

回收（decommit），指释放物理内存空间，但虚拟地址空间仍然保留，与提交相对应；

锁（lock），指将页面锁定在物理内存中，从而防止虚拟内存管理机制将页面交换至页面文件，而引起不必要的硬盘和物理内存间的低效页面交换；

解锁（unlock），指解锁页面，从而允许系统对页面进行交换操作。

Windows进程的虚拟地址空间中存在3种状态的页面：空闲页面、保留页面和提交页面。

空闲页面，指可以保留或提交的可用页面；

保留页面，指逻辑页面已分配，但没有分配物理存储页面。设置这种状态的效果是可以保留一部分虚拟地址，若不释放这些地址，就不能被其他应用程序所使用；

提交页面：指物理存储已被分配的页面。可对它加以保护，不允许访问、只读访问或允许读写访问。提交页面可以被回收，从而变成保留页面。

在本实验中，首先随机生成输入文件，其中包含对内存要做的各种操作；然后按照输入文件的各项实现对内存的管理。

2. 关键点说明

2.1 VirtualAlloc函数

在调用进程的虚拟地址中保留或提交页面。除非设置 MEM_RESET 标志，否则被这个函数分配的内存单元自动初始化为 0。

2.2 VirtualFree 函数

释放或回收调用进程的相关页面，成功，返回一个非零值；否则，返回零值。

2.3 VirtualLock 函数

将调用进程所占用的部分物理内存加锁，以确保后续对该区域的存取操作不会失败。调用成功，返回一个非零值；否则，返回零值。

2.4 VirtualUnlock 函数

将调用进程所占用的部分物理内存解锁，从而使系统在必要时可将这些页面换出。调用成功，返回一个非零值；否则，返回零值。

2.5 主要数据结构

定义两个结构体：

operation，记录对内存的操作信息

```
struct operation{
        int time;                       //起始时间
        int block;                      //内存页数
        int oper;                       //操作
        int protection;                 //权限
};
```

trace，跟踪每次分配活动

```
struct trace {
        LPVOID start;                   //起始地址
        long size;                      //分配大小
};
```

定义两个信号量 allo 和 trac，以保证内存操作的互斥，具体利用 WaitForSingleObject 和 ReleaseSemaphore 函数来实现。

四、 文件说明

共有一个源码文件：

Memory Management.cpp：内存管理的具体实现。

共有一个文本文件：

out.txt：记录内存行为。

五、 实验结果

参考代码经更改后，运行如下：

```
0:reserve now
starting address:00E10000       size:12288
1:reserve now
starting address:00FB0000       size:4096
2:reserve now
starting address:03090000       size:20480
3:reserve now
starting address:03220000       size:16384
4:reserve now
starting address:03870000       size:20480
5:commit now
starting address:00E10000       size:12288
6:commit now
starting address:00FB0000       size:4096
7:commit now
starting address:03090000       size:20480
8:commit now
starting address:03220000       size:16384
9:commit now
starting address:03870000       size:20480
10:lock now
starting address:00E10000       size:12288      998 - 内存分配访问无效
11:lock now
starting address:00FB0000       size:4096
12:lock now
starting address:03090000       size:20480      998 - 内存分配访问无效
13:lock now
starting address:03220000       size:16384
14:lock now
starting address:03870000       size:20480      998 - 内存分配访问无效
15:unlock now
starting address:00E10000       size:12288      158 - 段已解除锁定
16:unlock now
starting address:00FB0000       size:4096
17:unlock now
starting address:03090000       size:20480      158 - 段已解除锁定
18:unlock now
starting address:03220000       size:16384
19:unlock now
starting address:03870000       size:20480      158 - 段已解除锁定
20:decommit now
starting address:00E10000       size:12288      87 - 参数错误
21:decommit now
starting address:00FB0000       size:4096
22:decommit now
starting address:03090000       size:20480      87 - 参数错误
23:decommit now
starting address:03220000       size:16384
24:decommit now
starting address:03870000       size:20480      87 - 参数错误
25:release now
starting address:00E10000       size:12288
26:release now
starting address:00FB0000       size:4096
27:release now
starting address:03090000       size:20480
28:release now
starting address:03220000       size:16384
29:release now
starting address:03870000       size:20480
请按任意键继续. . .
```

out.txt 文件截取部分如下：

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
1
dwActiveProcessorMask	255
dwAllocationGranularity	65536
dwNumberOfProcessors	8
dwOemId	0
dwPageSize	4096
dwProcessorType	586
lpMaximumApplicationAddress	7FFEFFFF
lpMinimumApplicationAddress	00010000
wProcessorArchitecture	0
wProcessorLevel	6
wProcessorRevision	24067
wReserved	0
************************************************************************************
dwAvailPageFile	3150520320
dwAvailPhys	2147483647
dwAvailVirtual	2092396544
dwLength	32
dwMemoryLoad	60
dwTotalPageFile	4294967295
dwTotalPhys	2147483647
dwTotalVirtual	2147352576
&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
AllocationBase	00000000
AllocationProtect	0
BaseAddress	00010000
Protect	1
RegionSize	12451840
State	65536
Type	0
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

系统信息解释如下：

dwActiveProcessorMask	活动处理器掩码
dwAllocationGranularity	分配粒度
dwNumberOfProcessors	处理器号
dwPageSize	页大小
dwProcessorType	处理器类型
lpMaximumApplicationAddress	最大分配地址
lpMinimumApplicationAddress	最小分配地址
wProcessorArchitecture	处理器结构
wProcessorLevel	处理器级别
wProcessorRevision	处理器修订号
wReserved	保留

内存状况信息解释如下：

dwAvailPageFile	可用页文件
dwAvailPhys	可用物理空间大小
dwAvailVirtual	可用虚拟空间大小
dwLength	长度
dwMemoryLoad	主存下载
dwTotalPageFile	总共页文件
dwTotalPhys	总共物理空间大小
dwTotalVirtual	总共虚拟空间大小

内存基本信息解释如下：

AllocationBase	分配基址
AllocationProtect	分配保护
BaseAddress	基地址
Protect	类型
RegionSize	区域大小
State	状态
Type	类型

通篇浏览文件，可发现在 5 个 reserve 阶段 dwAvailVirtual 减少相应的块大小，5 个 release 阶段 dwAvail Virtual 增加相应的块大小，其余阶段可用虚拟空间大小不变。

六、 实验心得

在本次实验中，遇到的主要问题有以下六点：

a) 参考代码的修改。生成 opfile 文件、内存管理实现这两部分代码存在明显前后关系，因而可以整合至一个程序。但在首次运行时，命令行界面仅显示“0”，便不再向下运行。再次梳理代码，发现参考代码中并未添加关闭文件的操作，所以内存管理线程读取 opfile 文件时，随机生成的内存操作信息还未保存。通过 fclose 关闭后，该问题得到解决；

b） GetLastError 函数返回值。运行时，发现在锁、解锁及回收部分内存时，会多打印一个 1~1000 的数字。查阅代码，发现是 GetLastError 函数的返回值。共存在 3 种错误：998，内存访问无效；158，段

已解除锁定；87，参数错误。更改输出格式，将错误代号及具体信息同时打印，便于分析；

c) VirtualFree 函数参数。回收和释放均使用 VirtualFree 函数，但对于该函数的第二个参数，回收过程传入为块的大小，而释放过程，参数值为 0。起初以为函数传参存在错误，更改后编译无法通过。Baidu 得知，若 dwFreeType 指定为 MEM_RELEASE，dwSize 应设置为 0，否则函数会调用失败；

d) 记录条项数量。程序生成了 30 次内存操作，文件却自 0 开始，存在 31 个记录。起初迷惑不解，最终明白：程序运行初态记录了一次内存和系统状况。在 a)运行失败时，命令行界面显示的“0”便是这次记录；

e) 错误报告。对于锁、解锁和回收操作，1、3、5 会提示相应错误信息。起初，以为是权限造成，但修改权限后，仍然报错。最终也未探究出原因……；

f) 物理内存分配。查阅系统和内存状态记录，发现提交操作后，可用物理空间大小始终没有变化，且始终等于总共物理空间大小。研究许久……无果。

实践出真知，本次内存管理实验是对课堂和书本所学知识的补充。实际情形与已了解到的原理大体一致，但又复杂许多。通过反复研究代码，加深了对内存管理的理解和记忆，收获颇丰。