实验五：Windows虚拟存储

5.1 实验目的

了解Windows 2000/XP 的内存管理机制，掌握页式虚拟存储技术。

理解内存分配原理，特别是以页面为单位的虚拟内存分配方法。

掌握下Windows 2000/XP内存管理的基本API.

5.2 实验要求

通过本实验，要求学生能够了解在Windows 2000/XP中如何控制存储空间。具体的要求如下：

（1）在程序中利用一个线程模拟各种虚存活动，比如虚存的保留、提交等。

（2）在程序中运行另一个监控线程来实时监视系统当前所进行的虚存操作，并将监测到的这些操作的信息从控制台输出；要求本线程和上一个虚存活动模拟线程保持同步，即模拟线程一旦有某一模拟活动则监控线程就应当随即监控到该活动的信息。

（3）在利用监控线程监视内存活动的同时汇报整个存储系统的使用情况。

本实验中参考程序的运行结果如图2 -1 所示。

5.3 相关基础知识

5.3.1 Windows 中的虚拟存储技术

Windows 采用了分页存储技术，关于分页技术的详细知识，可以从任何一本操作系统的教材中得到。windows 在实现虚拟存储技术的时候，利用页面文件（P aging file ）来实现物理内存的扩展。所谓的页面文件就是Windows 2000 /XP 在硬盘上分配的用来存储没有装入内存的程序和数据文件部分的磁盘文件。这个文件是一个名叫pagefile.sys 的系统隐藏文件，当系统安装时，会在安装系统盘的根目录下创建该文件，其默认值大于计算机中 RAM 的1.5 倍。需要时，Windows 2000 / XP 将数据从页面文件移至内存，并将数据从内存移至页面文件以便为新数据释放空问。页面文件也称为交换文件。

页面文件和物理内存或RAM 构成“虚拟内存”。如果系统要求的内存量超过了虚拟内存的大小，则系统就会出现提示，发出虚拟内存不足的警告。我们可以根据需要设置虚拟内存的大小，方法是：右击“我的电脑”，依次选择“属性”->“高级”-> “性能选项”-> “更改”；在这个设置功能下，我们还可以在其他分区或者磁盘下新增页面文件（默认情况下，非系统盘上没有设置页面文件），这样相应的磁盘根目录下也会出现一个系统隐藏文件pagefile.sys。设置界面（Windows 2000）如图2-2 所示。

图2-2 Windows2000 虚拟1内存的设置

5.3.2 虚存页面的状态

1 .页面的种类

每一个进程的虚拟地址空间中的页面根据其所处状态可以分为三种：提交页面、保留页面和空闲页面。下面逐一解释它们的含义。

l）提交（Committed ）页面

提交页面是已分得物理存储的虚拟地址页面，通过设定该区域的属性可对它加以保护，例如设为“只读”。系统在第一次读写页面时进行初始化并将提交的页面装入物理内存；当进程结束时系统将释放提交页面的存储空问，当然也可以使用VirtualFree 函数进行存储空间的释放。

2）保留（Reserved ）页面

保留页面是逻辑页面已分配，但没有分配物理存储页面。这样可以在进程中保留一部分虚拟地址，如果没有释放这些地址，则进程中进行的其他内存分配操作就不能使用该段虚拟地址空问。可以使用VirtualFree 函数将提交页面转换为保留页面。

3）空闲（Free ）页面

空闲页面是指那些可以保留或提交的可用页面，对当前的进程是不可存取的。可以使用系统函数VirtualFree将提交页面或保留页面转换为空闲页面。

2 ．页面的操作

针对上述几种虚拟内存页面所处的几种不同状态，可以对其进行不同的操作，这些操作包括：保留一个区域、提交一个区域、回收一个区域、释放一个区域和对一个虚拟内存区域加锁或解锁等。各操作的含义解释如下。

* 保留：保留进程的虚拟地址空间，而不分配物理存储空间。
* 提交：在内存中为进程的虚拟地址分配物理存储空间。我们不但可以对空闲状态或者处于保留状态的页面进行提交操作，也可以对已经提交的页面进行提交操作。
* 回收：释放物理内存空问，但是虚拟地址空间仍然保留，它与提交相对应，即可以回收已经提交的内存块，有时又称为除配。
* 释放：将物理存储和虚拟地址空间全部释放，它与保留相对应，即可以释放已经保留的内存块。
* 加锁：可以对已经提交的页面进行加锁操作，这样就使得这些页面常驻内存而不会产生通常的缺页现象。可以是对已经加锁的页面进行的解锁操作。

5.3.3 存储系统的统计指标

系统中维护结构体MEMORYSTATUS ，我们可以通过这个结构体来看系统的虚拟和物理内存的指标。结构体格式如下，后面给出了其中各项的含义。

注意：除了最后两项是与各个不同进程相关的，其他的指标是关于整个系统的信息，对所有进程都是一样。

格式

typedef struct\_MEMORYSTATUS{

DWORD Dwlength;

DWORD dwMemoryload;

SIZE\_T dwAvailPhys;

SIZE\_T dwTotalPagefile;

SIZE\_T dwAvailPagefile;

SIZE\_T dwTotalVirtual;

SIZE\_T dwAvailVirtual;

}MEMORYSTATUS,\*LPMEMORYSTATUS;

参数说明

dwlength ：指明本结构所占的空间大小，在使用适当的函数GlobalMemoryStatus 从系统中获取这个结构的数据时，该系统函数会给这个域设置正确的值。

dwMemoryload ：物理存储使用负荷指数，使用一个百分数表示当前物理内存已经被占用的比率。在利用Win 32 API 查询得到的此结构中，这个比率只是精确到个位，而且是选择进一的原则，例如使用了78.2 ％的物理内存，则显示占用79 ％。

dwTotalPhys ：系统中安装的物理内存总数，以Byte（字节）计数。

dwAvailPhys ：可用物理内存数，以Bytpe 计数。

dwTotalPagefile ：页面文件总数，也就是系统在外存上为虚拟内存系统分配的页面文件( paging file ）的总量，以Byte 计数。

dwAvailPagefile ：可用页面文件数，以Byte 计数。

dwTotalVirtual ：本进程中用户可以访问的虚存空间总数。注意目前32 位WindowS 系统中，在总共4GB 的空间中 ，高端的2 GB 是系统占用的，只有低端的2GB 才是用户可以访问的。此处以Byte计数，也就是说在Windows 2000 / XP 该数字应该显示为2147352576 。

dwAvailVirtual：在本进程中用户可以访问的虚存空间中可用部分的数量，也就是还没有被程序分配的用户虚拟空间大小，以Byte 计数。

除了采用上述的结构来表示关于系统虚拟存储系统的信息以外，我们还关心在进程的某一段具体的空间上的虚拟存储空间的状态。为此系统也有一个相应的MEMORY\_BASIC\_INFORMATION 结构用来说明我们对一段具体的虚存空间可以关注哪些方面的属性。可以通过win32 函数VirtualQuery 查询从某一虚存地址开始的虚存也面的一些属性，并以本结构返回结果，该结构在windows.h 中定义。解释如下：

格式

typedef struct\_MEMORY\_BASIC\_INFORMATION {

PVOID BaseAddress ;

PVOID AllocationBase ;

DWORD AllocationProtect ;

DWORD Regionsize ;

DWORD State ;

DWORD Protect ;

DWORD Type ;

｝MEMORY\_BASIC\_INFORMATION;

参数说明

BaseAddress ：一个虚存地址，该结构所包含的信息就是从这个地址开始的、属性相同的、虚存地址的属性信息。

AllocationBase ：表示用VirtualAlloc 分配包括该段内存在内的内存块时，该分配动作的基地址（即起始地址 ）。

AllocationProtect ：代表分配该段地址空间时的页面属性，如PAGE\_READWRITE （可读可写）、PAGE\_ ExECUTE （可执行）等（其他属性可参考Platform SDK ，在virtualAlloc 函数中也有过介绍）。

RegionSize ：从BaseAddress 开始，具有相同属性（这里列出的属性）的地址空间的大小。

State ：当前这片虚存页面的状态，如上面讲到有三种取值：MEM\_COMMIT 、MEM \_FREE 和MEM\_RESERVE 。这个参数对我们来说是最重要的，从中我们便可知道指定内存页面的状态。

Protect ：第6 个参数是页面的属性，其可能的取值与AllocationProtect 相同。

Type ：最后一个参数指明了该内存块的类型，有三种可能值：MEM\_IMAGE 、MEM \_MAPPED 和MEM\_PRIVATE 。

5.4 程序的实现

5.4.1 实验环境

本实验是在Windows 2000 + VC 6 . 0 上实现的，主要利用Windows SDK 提供的API 来完成程序的功能。实验需要在Windows 下安装VC 后进行。VC 是一个集成开发环境，其中 包含了Windows SDK 的所有定义和工具。实验中所用的API ，即应用程序接口，是操作系统提供的用来进行应用程序设计的系统功能接口。要使用这些API ，需要包含对这终函数进行说明的SDK 头文件，最常见的就是windows . h 。

这里由于用到了多线程编程，我们还需要设置一下编译环境，在VC 环境中选中菜单project –>settings ，在左边列表中选中工程名，在右边选中C / C ++项，将其下project options 中的默认值MLd 改为MTd 。

注意：上述环境设置需要在打开某项工程后才能进行。随后在2 . 4 . 4 节将介绍如何建立相应工程。

另外，除了需要包含头文件windows.h 之外，process.h 头文件也是必需的，因为我们在程序中使用了beginthreadex ( C 运行时函数)。

5.4.2 程序的结构

本程序是结构化的C 代码，程序仅包含一个单进程，在主线程中派生两个线程，一个线程用来仿真存储器的活动，另一个线程用来监控第l 个线程的内存行为，因此两个线程之间存在同步关系。程序中各函数说明如下。

1 . main 函数

用\_beginthreadex 函数启动两个线程。

2 . simulator 线程（模拟内存分配）

随机地进行各种虚存操作，这些操作包括：虚存的保留与提交、虚存的注销、虚存的注销并释放虚存空间、改变虚拟内存页的保护、锁定虚拟内存页和虚存的保留。每一个活动完成后，程序的全局变量Actnum 就被设置为一个特定的整数值。线程随即被阻塞直至Actnum 被置回零。

3 . inspector 线程（跟踪内存分配情况）

通过Actnum 的值获取上一个虚存动作的类型。并通过BASE\_PTR 的值获得该动作发生的虚存地址，打印相关信息。最后设置Actnum 为零，通知模拟线程继续下一次动作，从而实现两个线程的同步。

当程序的主线程完成两个主要工作线程的派生后，便处于等待状态，用户此时可以观察模拟和监控线程的活动。如果想结束程序，只需键入中止程序的命令即可。实际上，本程序中止命令就是键入任何一个键，如图2-3 所示。其中带阴影的部分是程序的流程，虚线框表示这些流程部分各自对应的函数体。

5.4.3 数据结构

1 ．指示器Actnum

该指示器的初值为零。当其为零时，模拟线程处于活动状态。在一个随机数的控制下，模拟线程进行各种虚存操作活动。当一个模拟活动成功完成，模拟线程就会设置指示器Actnun 为一个非零值并自动阻塞直至Actnum 被置回零（由监控线程完成该步骤）。监控线程在Actnum 为零时阻塞等待，当监控到Actnum 变化到非零，则通过Actnum 的值确定该动作的类型，并通过全局量BASE\_PTR 的值获得该动作发生的虚存地址。在这两个信息的帮助下，监控线程可完成对该动作的监控，最后设置Actnum 为零。模拟线程因此得以继续模拟。概括地讲，通过Actnum 实现了两个线程的同步和信息传递。

2 ．地址指针BASE\_PTR

这个指针在进行虚存分配操作时用于记录返回的虚存起始地址，在进行其他虚存操作时用于指示进行该项操作的目标虚存的起始地址。它在程序初始执行时并没有被赋初值，所以在头几次随机的虚存模拟活动中可能导致动作失败，直到BASE \_PTR 由虚存分配函数VirtulAlloc 来赋予一个已分配的虚存空间区块的起始地址。但是由于对失败的动作程序采取忽略策略，所以这并不影响程序的运行。在程序运行过程中任意时刻，该指针的值都是上一次成功的虚存控制动作所进行的虚存地址。

5.4.4 实现步骤

（1） 打开VC ，选择菜单项File ->New ，选择projects 选项卡并建立一个名为VIRTUMEM 的win32 console applicatoin工程。创建时注意指定创建该工程的目录。

（2） 在工程中创建源文件VIRTUMEM.cpp ：选择菜单项Project ->Add to project ->Files , 此时将打开个新窗口，在其中的“文件名”输入栏中输入自己想要创建的文件名，这里是VIRTUMEM . cpp 。接着询问是否创建新文件，回答yes 。然后，出现如图2 -4 所示窗口，通过Workspace->Fileview->Source Files打开该文件，在其中编辑源文件并保存。

图2 -4 创建并编辑源程序文件

（3） 设置编译环境（参考2 . 4 . 1 节）,然后调用菜单项Buils->Rebuild all进行编辑连接，可以在指定的工程目录下得到debug ->VIRTUMEN.exe程序，现在即可在控制台以指定的参数运行VIRTUMEM . exe.

5.4.5 相关API 函数介绍

1.\_beginthreadex 的用法

功能

用来创建新线程的C runtime 函数。所创建的线程执行指定的可执行模块（在本实验中我们在程序中给出了threadwork 函数供各线程执行）。

格式

unsigned long \_beginthreadex （

VOID \* security ,

UNSIGNED stack size ,

UNSIGNED (\_stdcall \* start\_address ) （void \* ) ,

VOID \* arglist ,

UNSIGNED initflag ,

UNSIGNED \* thrdaddr

) ;

参数说明

security : SECURITY\_ATTRIBUTES 类型的指针，指出返回的句柄是否可被子线程继承，指定为NULL 时，表示不可继承。

stack\_size ：指定新的线程所用的堆栈的大小（字节），指定为“0 ”时表示使用默认值，即其大小与当前线程一样。

unsigned ( \_stdcall \* start\_address )( void \* ) ：指向新的线程将要执行的函数模块，该函数必须声明为\_stdcan 标准，具体做法见程序。

arglist ：执行的函数模块的参数表，没有任何参数则取NULL 。

initflag ：用于指定线程的初始状态（取“0”表示运行状态，否则取CREATE\_ SUSPEND 表示挂起）

thrdaddr : 32 位的一个整数，是代表新线程的标志号，在本线程被创建后就可以通过它来访问。

2 . sleep 的用法

功能

用于使当前线程休眠的API 。

格式

VOID Sleep{

WDORD dWmilliseconds

};

参数说明

dwmiuiseconds ：指定休眠的微秒数。

3 . VirtualAlloc 函数

功能

保留（或）提交某一范围的虚拟地址。当在一个进程中保留一段虚拟地址时，并没有物理内存页被提交。而且，保留一个地址范围将不会保证将来会有可用的物理内存来提交给这些地址。要使用保留的地址，内存首先必须被提交给该地址，当内存被提交时，内存物理页被分配。

格式

LPVOID VirtualAllc (

LPVOID lpAddress ,

SIZE\_T dwsize ,

DWORD flAllocateType ,

DWORD flProtect

) :

参数说明

lpAddress ：如果使用的是NULL 值，则该函数在某一个最合适的位置保留地址范围。否则本参数为要保留的地址范围指定一个准确的起始地址。本函数成功执行后，返回值指示实际被保留的地址的开始位置。如果无法完成请求，则返回一个错误代码。

dwsize ：该参数指示函数待分配的地址范围，即需要分配多大的虚拟地址空间。该值的大小可以是小于2 GB 的任意值，但是VirtualAlloc 函数实际上被限制为一个较小的范围。能够被保留的最大值为该进程中最大的连续自由地址空间。请求2 GB 的范围会导致失败，因为加载一个应用程序时已经使用了整个2 GB 地址空间的一部分，这导致在任何给定的时间里，不可能有2GB 那么多的可用地址空间。如果第一个参数不是NULL ，则实际上请求分配的范围是lpAddress ~( lpAddress + dwsize ）这个地址范围跨越的所有页面。

fLAllocationType 、fLProtect ：被用来决定以何种方式分配地址以及与它们相关联的页面的属性。地址可被分配为MEM\_RESERVE 或者MEM\_COMMIT 类型，前者用于保留虚存，后者用于提交己保留的虚存，如果使用二者的并运算，则表示保留并提交一块尚未保留的虚存。页面属性可以指定为如下一些值： READONLY （只读）、 PAGE \_READWRITE （读写）、PAGE\_EXECUTE （执行）、PAGE\_EXECUTE \_READ （执行和读取）、PAGE\_EXECXTE \_READWRITE （执行和读写）和PAGE \_NOACCESS （不允许任何存取）。值得注意的是，无沦上述哪种值被传递到该函数，被保留的地址总是PAGE\_NOACCESS ，这是系统强制的默认值。已提交的页可以是只读的、可读写的、可执行的等等。

4 . VirtualFree 函数

功能

解除已被提交的虚存或者释放被保留或者提交的进程虚拟地址空间。一旦地址页面被保留或者已被提交，则VirtualFree 函数是惟一可以释放它们的函数，即将它们返回到自由地址状态的方法。VirtualFree还可以用来对己提交的虚拟地址页解除提交，使得该段地址空间变成保留状态。当解除地址的提交时，所有与该地址相关的物理内存都会被释放。

格式

B00L VirtualFree（

LPV0lD lpAddress ,

DWORD dwsize ,

DWORD dwFreeType

）；

参数说明

lpAddress ：表示要释放的虚拟内存的起始地址。

dwsize ：表示要释放的区块大小（B yte ）。当进行虚存释放操作时，必须将它设置为“0 " , 否则操作的范围是包含在lpAddress ~（IpAddress + dwsize ）范围内的所有页面。

dwFreeType ：该参数可取为MEM\_DECOMMIT 或者MEM\_RELEASE 。前者表示解除提交，后者表示释放空间。

注意：只有已提交的地址才能被解除提交。当需要对一个大范围的地址解除提交时牢记这一点是非常重要的。

例如，假设你有一定范围的地址，在其中多个地址的子集己被提交，而其他部分被保留。要想使全部范围的地址保留，惟一方法就是对离散的各个被提交的地址段分别解除提交。试图对整个范围的地址解除提交将会失败，因为保留的地址无法解除提交。但是在进行存储空间释放时，指定的相关页面都会被释放，而不会出现上述解除提交操作中提到的问题。因为在地址被释放时，一个地址的状态并不妨碍对它进行一次释放操作，哪怕它本身就处在空闲状态。

5 . VirtualProtect 函数

功能

改变虚拟内存页的保护方式。例如，一个应用程序可以按PAGE\_READWRITE 保护方式来提交一个页的地址，并且立即将数据填写到该页中。然后，该页的保护将被改变为PAGE\_READONLY ，这样可以有效地保护数据不被该进程中的任何线程重写。

格式

B00L VirtualProtect (

LPVOID lpAddress ,

DWORD dwsize ,

DWORD flNewProtect ,

PDWORD f1OldProtect

);

参数说明

lpAddress ：表示要改变页面保护方式的起始地址。

dwsize ：表示想要进行操作的区块的大小（B yte ）。实际上操作的范围是包含在lpAddress ~（lpAddress + dwsize ）范围内的所有页面。

注意：该函数操作的所有区块必须是由同一次分配动作保留或提交的区块。

flNewProtect ：用于指定新的保护方式。

flOldProtect ：用于获得作用区域的第1 页旧的保护方式，如果指定为NULL 或者不是PDWOD 类型的变量，则会导致调用不成功。

6 . VirtualLock 和VirtualUnlock 函数

功能

锁定和解锁虚拟内存页。一个进程可以分配一些页并将它锁定到工作组中或者解锁，这样可保证对它们的使用不会出现缺页现象。但是如果一个应用程序将提交的内存页锁定在工作组中 ，这样做可能利大于弊，因为这样做可能会迫使该进程中的其他关键页被替代。如果那样的话，这些关键页可能会被映射到磁盘，在被访问的时候会发生缺页，从而导致该进程将花费许多的CPU 时间将关键页映射到内存和映射出内存。

格式

BOOL Virtua1Lock (

LPVOID lpAddress ,

DWORD dwsize

);

B00L VirtualUnlock (

LPV0lD lpAddress,

DWORD dwsize

);

参数说明

lpAdrress ：表示操作的起始地址。

dwSize ：表示要锁定和解锁的范围。

同样，函数执行的结果是上述参数指定的虚拟地址空间所跨越的所有页面都被锁定到内存或被解锁。

注意：解锁操作的作用范围内的区块必须是经过锁定的页面；被锁定的页面必须是被提交过的，并且保护方式不是PAGE\_NOACCESS ；最后，当一个进程终止时，被锁定的页面会自动解除锁定。

7 . virtualQuery 函数

功能

查询一个进程的虚拟内存。给定一个进程的地址空间为2GB ，如果没有查询地址信息的能力，那么想要管理地址的全部范围将是困难的。因为地址本身代表独立的内存，对于它们，这些内存有可能被提交，或者不被提交，对它们进行查询，就是要读取保存虚存空间状态的数据结构。在Windows 2000 中，该结构是“虚拟地址描述符树”。Win32 在VirtualQuecy 函数中提供了“遍历VAD 结构”的能力。

格式

DWORD VirtualQuery (

LPCVOID lpAddress ,

PMEMORY\_BASIC\_INFORMATION lpBuffer ,

DWORD dwLenqth

) ;

参数说明

IpAddress ：指明开始查询状态的虚存空间的开始地址点。

IpBuffer ：一个系统定义的MEMORY\_BASIC\_INFORMATION 结构的指针，前面已经解释了该结构的含义。

dwLength ：传入上面提到的结构所占空间的字节数。

8 . GlobalMemorystatus 函数

功能

该函数用于获取程序存储空间的使用状况以及系统的使用概况。

格式

VOID Globa1Memorystatus (

LPMEMORYSTATUS lpBuffer

);

参数说明

lPBuffer ：这个参数是一个指向MEMORYSTATUS结构的一个指针，用于函数返回相关的信息。该指针由系统定义，这在基础知识的介绍中已经讲到了。

5.5 实验分析

程序运行时，模拟线程进行各种虚存操作，这些活动导致了程序虚存空间和系统存储资源的变化。监控线程给出了监控到的这些活动和变化的信息。程序给出了整个内存系统的各个全局统计量，包括物理内存使用量和页面文件使用情况等。通过这些统计量的变化可以分析当前的虚存活动对存储系统带来的变化。在分析的过程中可以帮助学生理解一些重要的内容，比如程序在进行一次虚存提交过后，显示的可用物理存储反而变多了，这是因为Windows 的内存管理发现某个进程在一段时间没有运行后，会将它的部分页面转移到页面文件中去。所以虽然本实验程序的进程目前分配了一些内存，但是总的可用物理内存数量还是增多了。作为验证，可以检查一下此时可用页面文件的数量是否减少。

5.6 实验报告要求：

## 1）分析和总结5.7 给出的程序实现的过程和功能

\_beginthreadex函数功能：创建新的线程

主函数创建两个线程，用于启动模拟虚拟活动和进行监控的线程

simulator 模拟虚存活动的独立线程

inspector 来监视另一线程的模拟活动

VitualAlloc函数功能：在调用进程的虚拟地址中保留、提交内存区域

VirtualFree 函数功能：释放或注销调用进程虚拟空间中的页面。在本实验中，执行对内存块的释放和回收。

VirtualProtect 函数功能：改变虚拟内存页的保护方式。

VirtualLock 函数功能：将调用进程虚拟空间中的内存加锁

VirtualUnlock 函数功能：将调用进程虚拟空间中的内存解锁

VirtualQuery 函数功能：查询进程的虚拟内存

Globalmemorystatus 函数功能：获取计算机系统中当前使用的物理内存和虚拟内存的信息。

dwlength ：指明本结构所占的空间大小，在使用适当的函数GlobalMemoryStatus 从系统中获取这个结构的数据时，该系统函数会给这个域设置正确的值。

dwMemoryload ：物理存储使用负荷指数，使用一个百分数表示当前物理内存已经被占用的比率。在利用Win 32 API 查询得到的此结构中，这个比率只是精确到个位，而且是选择进一的原则，例如使用了78.2 ％的物理内存，则显示占用79 ％。

dwTotalPhys ：系统中安装的物理内存总数，以Byte（字节）计数。

dwAvailPhys ：可用物理内存数，以Bytpe 计数。

dwTotalPagefile ：页面文件总数，也就是系统在外存上为虚拟内存系统分配的页面文件( paging file ）的总量，以Byte 计数。

dwAvailPagefile ：可用页面文件数，以Byte 计数。

dwTotalVirtual ：本进程中用户可以访问的虚存空间总数。注意目前32 位WindowS 系统中，在总共4GB 的空间中 ，高端的2 GB 是系统占用的，只有低端的2GB 才是用户可以访问的。此处以Byte计数，也就是说在Windows 2000 / XP 该数字应该显示为2147352576 。

dwAvailVirtual：在本进程中用户可以访问的虚存空间中可用部分的数量，也就是还没有被程序分配的用户虚拟空间大小，以Byte 计数。

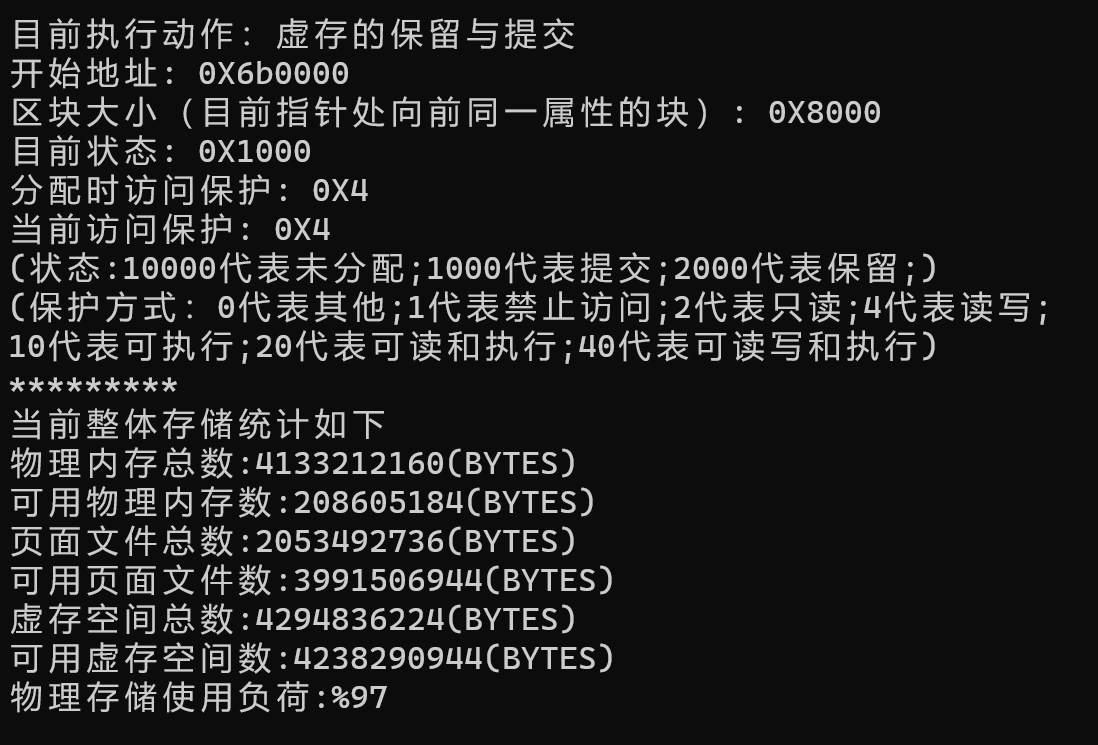
除了采用上述的结构来表示关于系统虚拟存储系统的信息以外，我们还关心在进程的某一段具体的空间上的虚拟存储空间的状态。为此系统也有一个相应的MEMORY\_BASIC\_INFORMATION 结构用来说明我们对一段具体的虚存空间可以关注哪些方面的属性。可以通过win32 函数VirtualQuery 查询从某一虚存地址开始的虚存也面的一些属性，并以本结构返回结果，该结构在windows.h 中定义。

## 2）给出运行结果截图

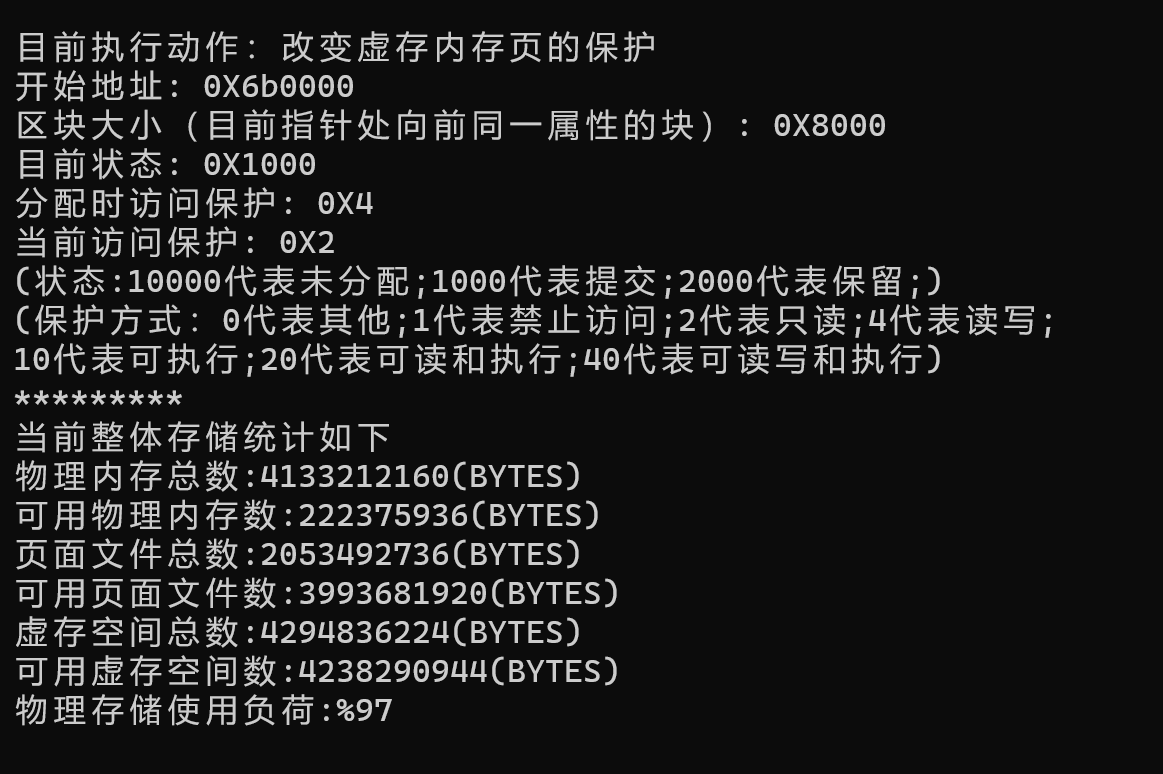
虚存的保留



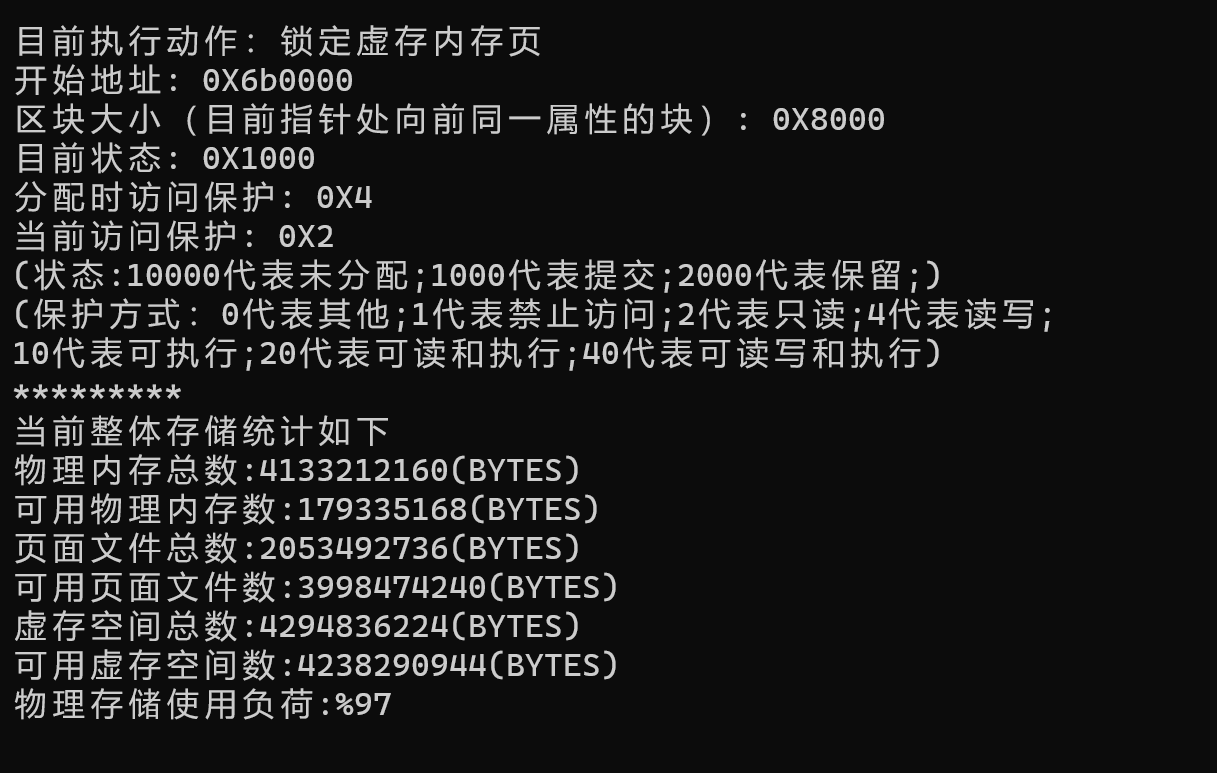
虚存的保留和提交



虚存内存页的保护



虚存内存页的锁定（枷锁）



虚存的释放

## 3）根据程序的功能，对运行结果进行解释。

Windows在实现虚拟存储技术的时候，利用页面文件来实现物理内存的扩展。虚拟页面状态分为三种，提交页面、保留页面、空闲页面；页面操作分为五种，保留、提交、回收、释放、枷锁。正如上面的截图所展示的一样。

当程序运行时，模拟线程会执行各种虚存操作，这些操作会导致程序虚拟内存空间和系统存储资源的变化。监控线程会提供这些活动和变化的信息。程序还会提供整个内存系统的各种全局统计量，如物理内存使用量和页面文件使用情况等。通过分析这些统计量的变化，可以评估当前虚拟内存活动对存储系统的影响。在分析过程中，这些统计量可以帮助我们理解一些重要的概念。例如，程序执行虚存提交后，显示的可用物理存储反而增多的现象是由于Windows的内存管理发现某个进程在一段时间没有运行后，将其部分页面转移到了页面文件中。因此，虽然本实验程序的进程目前分配了一些内存，但总的可用物理内存数量仍然增多了。

## 4）实验心得

通过本次实验，我对于内存分配原理和虚拟内存分配方法有了更深入的理解。同时，通过老师提供的代码，学习了Windows内存管理的基本API，并且了解了Windows的内存管理机制和页式虚拟存储技术。这些知识对于开发高效、稳定的程序非常重要，也可以帮助更好地理解操作系统的工作原理。希望在以后的学习和实践中能够应用这些知识。

5.7 源程序

内容:

源程序

#include <windows.h>

#include <stdio.h>

#include <process.h>

#include <time.h>

unsigned \_stdcall simulator(void \*);

unsigned \_stdcall inspector(void \*);

LPVOID BASE\_PTR;

int Actnum=0;

//主函数，主要是用于启动用于模拟虚存活动和进行监控的两个线程

int main(int argc, char\* argv[])

{

unsigned ThreadID[2];

\_beginthreadex(NULL,0,simulator,NULL,0,&ThreadID[0]);

\_beginthreadex(NULL,0,inspector,NULL,0,&ThreadID[1]);

getchar();

return 0;

}

//模拟一系列的虚存活动，作为一个独立的线程运行

unsigned \_stdcall simulator(void \*)

{

DWORD OldProtect;

int randnum;

printf("Now the simulator procedure has been started.\n");

//产生一个随机数种子

srand( (unsigned)time (NULL) );

//在一个死循环中，用随机数控制，不断进行虚存操作活动

while(1)

{

Sleep(500);//控制整个模拟和监控的速度

while(Actnum!=0)

{

Sleep(500);//等待，直到监控线程捕捉到上一个模拟动作后再继续下一个动作

}

randnum = 7&rand();

switch(randnum) //各个动作中的虚存指针均使用BASE\_PTR；它在过程中由虚存分配函数动态调整，如果某动作不成功，则不会被监控线程监控到

{

case 0:

if (BASE\_PTR=VirtualAlloc(NULL,1024\*32,MEM\_RESERVE|MEM\_COMMIT,PAGE\_READWRITE))

{

Actnum=1;//虚存的保留与提交

}

break;

case 1:

if (VirtualFree(BASE\_PTR,1024\*32,MEM\_COMMIT))

{

Actnum=2;//虚存的注销

}

break;

case 2:

if (VirtualFree(BASE\_PTR,0,MEM\_RESERVE))

{

Actnum=3;//虚存的注销并释放虚存空间

}

break;

case 3:

if (VirtualProtect(BASE\_PTR,1024\*32,PAGE\_READONLY,&OldProtect))

{

Actnum=4;//改变虚存内存页的保护

}

break;

case 4:

if (VirtualLock(BASE\_PTR,1024\*12))

{

Actnum=5;//锁定虚拟内存页

}

break;

case 5:

if (BASE\_PTR=VirtualAlloc(NULL,1024\*32,MEM\_RESERVE,PAGE\_READWRITE))

{

Actnum=6;//虚存的保留

}

break;

default:

break;

} //end of 'switch'

} //end of 'while'

return 0;

}

//通过一个全局变量来监视另一模拟线程的模拟活动，并通过适当的信息查询函数，将存储的使用和活动情况打出报告

unsigned \_stdcall inspector(void \*)

{

int QuOut=0;

char para1[3000];

MEMORYSTATUS Vmeminfo;

char tempstr[100];

MEMORY\_BASIC\_INFORMATION inspectorinfo1;

int structsize = sizeof(MEMORY\_BASIC\_INFORMATION);

printf("Hi , now inspector begin to work\n");

//在一个死循环中不断通过一个全局变量（监视器），来监控模拟线程是否由新的动作，如果有，通过的API函数将相应虚存处（通过公用BASE\_PTR实现）的信息进行检查，从而验证该动作对存储使用的影响

while(1)

{

Sleep(1000);

if(Actnum!=0)

{

//通过全局变量（监视器）Actnum来获取上一个虚存动作的类型及相应构造监控信息的头部

switch(Actnum)

{

case 1:

memset (&inspectorinfo1, 0 , structsize);

VirtualQuery((LPVOID)BASE\_PTR,&inspectorinfo1,structsize);

strcpy(para1,"目前执行动作：虚存的保留与提交\n");

break;

case 2:

memset (&inspectorinfo1, 0 , structsize);

VirtualQuery((LPVOID)BASE\_PTR,&inspectorinfo1,structsize);

strcpy(para1,"目前执行动作：虚存的除配\n");

break;

case 3:

memset (&inspectorinfo1, 0 , structsize);

VirtualQuery((LPVOID)BASE\_PTR,&inspectorinfo1,structsize);

strcpy(para1,"目前执行动作：虚存的除配并释放虚存空间\n");

break;

case 4:

memset (&inspectorinfo1, 0 , structsize);

VirtualQuery((LPVOID)BASE\_PTR,&inspectorinfo1,structsize);

strcpy(para1,"目前执行动作：改变虚存内存页的保护\n");

break;

case 5:

memset (&inspectorinfo1, 0 , structsize);

VirtualQuery((LPVOID)BASE\_PTR,&inspectorinfo1,structsize);

strcpy(para1,"目前执行动作：锁定虚存内存页\n");

break;

case 6:

memset (&inspectorinfo1, 0 , structsize);

VirtualQuery((LPVOID)BASE\_PTR,&inspectorinfo1,structsize);

strcpy(para1,"目前执行动作：虚存的保留\n");

break;

default:

break;

}

//实时显示固定格式的相关材料；通过目前监控到的动作所发生的虚存地址，监控该活动对相应存储空间的影响

sprintf(tempstr,"开始地址: 0X%x\n",inspectorinfo1.BaseAddress);

strcat(para1,tempstr);

sprintf(tempstr,"区块大小（目前指针处向前同一属性的块）: 0X%x\n",inspectorinfo1.RegionSize);

strcat(para1,tempstr);

sprintf(tempstr,"目前状态: 0X%x\n",inspectorinfo1.State);

strcat(para1,tempstr);

sprintf(tempstr,"分配时访问保护: 0X%x\n",inspectorinfo1.AllocationProtect);

strcat(para1,tempstr);

sprintf(tempstr,"当前访问保护: 0X%x\n",inspectorinfo1.Protect);

strcat(para1,tempstr);

strcat(para1,"(状态:10000代表未分配;1000代表提交;2000代表保留;)\n");

strcat(para1,"(保护方式：0代表其他;1代表禁止访问;2代表只读;4代表读写;\n10代表可执行;20代表可读和执行;40代表可读写和执行)\n\*\*\*\*\*\*\*\*\*\n");

//全局信息，报告目前系统和当前进程的存储使用总体情况

GlobalMemoryStatus(&Vmeminfo);

strcat(para1,"当前整体存储统计如下\n");

sprintf(tempstr,"物理内存总数:%lu(BYTES)\n",Vmeminfo.dwTotalPhys);//书中的数据类型为d，改为lu

strcat(para1,tempstr);

sprintf(tempstr,"可用物理内存数:%lu(BYTES)\n",Vmeminfo.dwAvailPhys); //书中的数据类型为d，改为lu

strcat(para1,tempstr);

sprintf(tempstr,"页面文件总数:%lu(BYTES)\n",Vmeminfo.dwTotalPageFile);//书中的数据类型为d，改为lu

strcat(para1,tempstr);

sprintf(tempstr,"可用页面文件数:%lu(BYTES)\n",Vmeminfo.dwAvailPageFile);//书中的数据类型为d，改为lu

strcat(para1,tempstr);

sprintf(tempstr,"虚存空间总数:%lu(BYTES)\n",Vmeminfo.dwTotalVirtual);//书中的数据类型为d，改为lu

strcat(para1,tempstr);

sprintf(tempstr,"可用虚存空间数:%lu(BYTES)\n",Vmeminfo.dwAvailVirtual);//书中的数据类型为d，改为lu

strcat(para1,tempstr);

sprintf(tempstr,"物理存储使用负荷:%%%lu\n\n\n\n",Vmeminfo.dwMemoryLoad);//书中的数据类型为d，改为lu

strcat(para1,tempstr);

printf("%s",para1);//显示报告内容

//(这里可以同时将报告内容记录进日志文件)

Actnum=0;//通知模拟线程可以进行下一个模拟动作

Sleep(500);//调节模拟和监控的总体速度

}//for if

} //for while

return 0;

}