Windows操作系统

C/C++ 程序实验

首都师范大学 信息工程学院

系统分析与管理实验室

## 实验八 Windows 内存管理

### 一、背景知识

  使用Windows 的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，而且要求两个线程之间通过信号量实现同步。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，每个内存操作包括如下内容：

1、时间：操作等待时间。

2、块数：分配内存的粒度。

3、操作：包括保留(reserve)一个区域、提交(commit)一个区域、释放(release)一个区域、回收(decommit)一个区域和加锁(lock)与解锁(unlock)一个区域，可以将这些操作编号存放于文件。

a、保留是指保留进程的虚拟地址空间，而不分配物理存储空间。

b、提交在内存中分配物理存储空间。

c、回收是指释放物理内存空间，但在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。

d、释放是指将物理存储和虚拟地址空间全部释放，它与保留(reserve)相对应．即可以 释放已经保留的内存块。

4、大小：块的大小。

5、访问权限：共五种，分别为PAGE\_READONLY，PAGE\_READWRITE，PAGE\_EXECUTE，PAGE\_EXECUTE\_READ和PAGE\_EXETUTE\_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。

### 二、实验目的

本练习的目的是从不同侧面了解Windows 对用户进程的虚拟内存空间的管理、分配方法。同时需要了解跟踪程序的编写方法（与被跟踪程序保持同步，使用Windows提供的信号量）。

对Windows分配虚拟内存、改变内存状态，以及对物理内存(physical memory)和页面文件(pagefile)状态查询的API函数的功能、参数限制、使用规则要进一步了解。

默认情况下，32位Windows 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 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。

Windows进程的虚拟地址空间中也有三种状态的页面：空闲页面、保留页面和提交页面。

1、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。

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的页面。设置这种状态的效果是可以保留一部分虚拟地址，这样，如果不预先释放这些地址，就不能被其他应用程序（如malloc，localalloc等）的操作所使用。

试图读或写空闲页面或保留页面将导致页面出错异常。保留页面可被释放或提交。

3、提交(Committed)页面：提交页面是物理存储（在内存中或磁盘上）已被分配的页面。可对它加以保护，不许访问或允许只读访问，或允许读写访问。提交也可以被回收以释放存储空间，从而变成保留页面。

### 三、工具/准备工作

在开始本实验之前，请回顾教科书的相关内容。

需要做以下准备：

1) 一台运行Windows 操作系统的计算机。

2) 计算机中需安装Microsoft Visual Studio Code。

### 四、实验内容与步骤

#### 1. 内存管理

**步骤1**：登录进入Windows 。

**步骤2**：在“开始”菜单中单击“程序”-“Microsoft Visual Studio Code”。

**步骤3：**新建项目名为“8-1”，并且新建项“8-1.cpp”。

**清单8-1 创建文件**

#include <fstream>

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

using namespace std;

struct operation

{

int time; //起始时间

int block; //内存页数

int oper; //操作

int protection; //权限

};

int main()

{

FILE \* file;

file = fopen("opfile", "wb"); //"opfile"为二进制用确定内存操作

operation op;

srand((unsigned)time(NULL));

for (int j = 0; j < 6; j++)

{

for (int i = 0; i <5; i++)

{

//0-PAGE\_READONLY;

//1-PAGE\_READWRITE;

//2-PAGE\_EXECUTE;

//3-PAGE\_ECECUTE\_READ;

//4-PAGE\_EXECUTE\_READWRITE;

op.time = rand() % 1000; //随即生成等待时间

printf("op.time=%d", op.time);

op.block = rand() % 5 + 1; //随即生成块大小

printf("op.block=%d\n", op.block);

op.oper = j;

op.protection = i;

fwrite(&op, sizeof(operation), 1, file); //将生成的结构写入文件

}

}

system("pause");

return 0;

}

**步骤4：**单按“F5”开始调试，注意路径里不要含有中文。

**步骤5：**按暂停按钮可暂停程序的执行，按终止按钮可终止程序的执行。



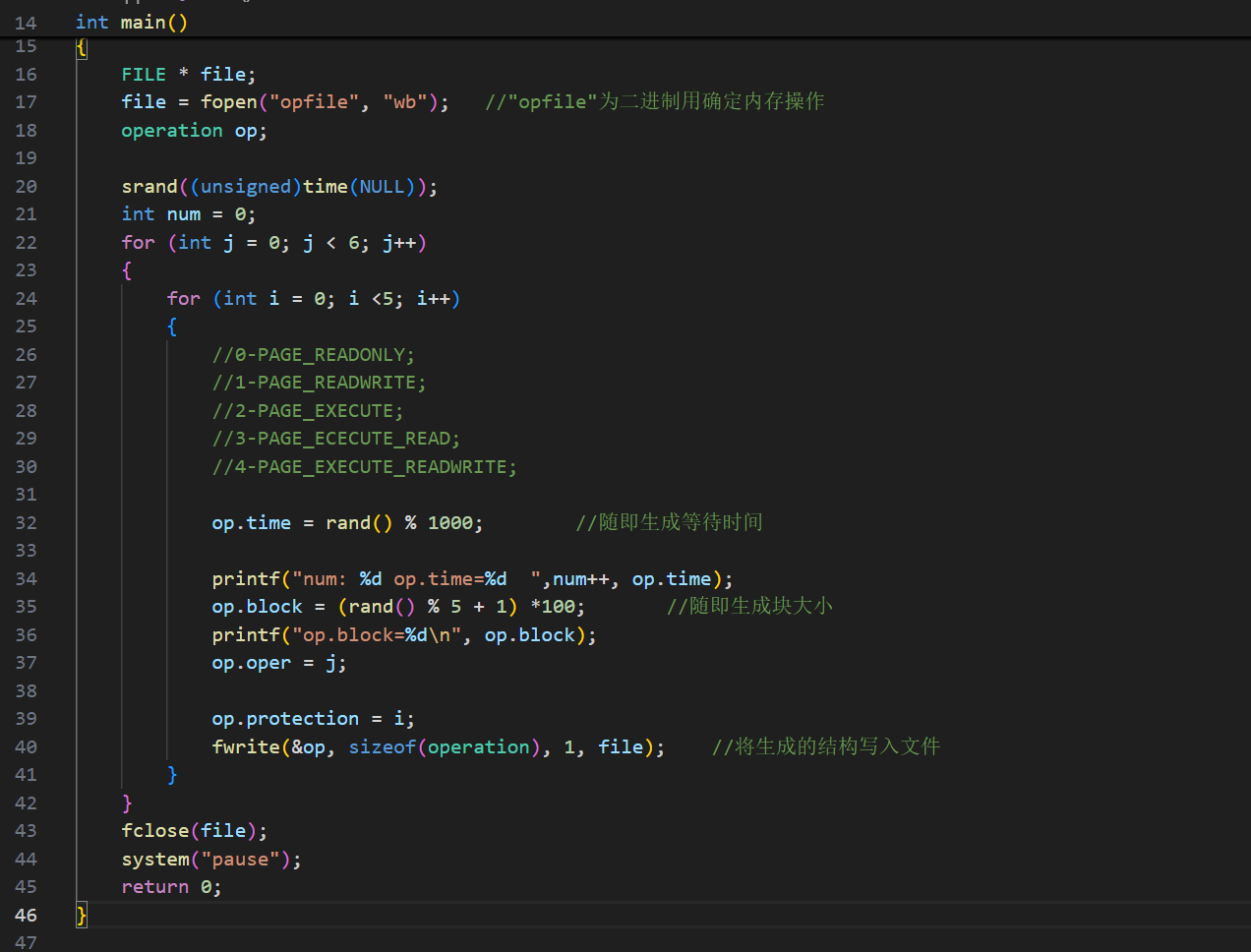
操作能否正常进行？如果不行，则可能的原因是什么？

操作能够正常进行，如果不行，可能是因为文件中包含中文字符，或文件路径包含中文。

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

运行结果是：



图片 1 修改块大小，并且加上fclose(file)

文本

低可信度描述已自动生成

图片 2 控制台输出结果如上图所示

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

图片 3 文件夹中多出一个opfile文件

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**步骤6：**新建项目名为“8-2”，并且新建项“8-2.cpp”。

**清单8-2 内存管理**

#include <stdio.h>

#include <stdlib.h>

#include <windows.h>

#include <iostream>

#include <fstream>

using namespace std;

struct operation

{

int time; //起始时间

int block; //内存页数

int oper; //操作

int protection; //权限

};

struct trace //跟踪每一次分配活动的数据结构

{

LPVOID start; //起始地址

long size; //分配的大小

};

HANDLE allo, trac; //信号量句柄

DWORD Tracker(LPDWORD lpdwparm) //跟踪allocator线程的内存行为，并输出必要信息

{

ofstream outfile; //输出文件

outfile.open("out.txt");

for (int i = 0; i <= 30; i++)

{

WaitForSingleObject(trac, INFINITE); //等待allocator一次内存分配活动结束

//打印内存情况和系统状况

outfile << i << endl;

//以下一段显示系统信息，每次执行操作后系统信息不变

//如果要查看系统信息，可以取消注释

/\*\*/ SYSTEM\_INFO info;

GetSystemInfo(&info);

outfile << "dwActiveProcessorMask" << '\t' << info.dwActiveProcessorMask << endl;

outfile << "dwAllocationGranularity" << '\t' << info.dwAllocationGranularity << endl;

outfile << "dwNumberOfProcessors" << '\t' << info.dwNumberOfProcessors << endl;

outfile << "dwOemId" << '\t' << info.dwOemId << endl;

outfile << "dwPageSize" << '\t' << info.dwPageSize << endl;

outfile << "dwProcessorType" << '\t' << info.dwProcessorType << endl;

outfile << "lpMaximumApplicationAddress"<<'\t'<<

info.lpMaximumApplicationAddress << endl;

outfile<<"lpminimemApplicationAddress"<<'\t'<< info.lpMinimumApplicationAddress << endl;

outfile << "wProcessorArchitecture" << '\t' << info.wProcessorArchitecture << endl;

outfile << "wProcessorLevel" << '\t' << info.wProcessorLevel << endl;

outfile << "wProcessorRevision" << '\t' << info.wProcessorRevision << endl;

outfile << "wReserved" << '\t' << info.wReserved << endl;

outfile << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

/\*\*/

//内存状况

/\*\*/ MEMORYSTATUS status; //内存状况

GlobalMemoryStatus(&status);

outfile << "dwAvailPageFile" << '\t' << status.dwAvailPageFile << endl;

outfile << "dwAvailPhys" << '\t' << status.dwAvailPhys << endl;

outfile << "dwAvailVirtual" << '\t' << status.dwAvailVirtual << endl;

outfile << "dwLength" << '\t' << status.dwLength << endl;

outfile << "dwMemoryLoad" << '\t' << status.dwMemoryLoad << endl;

outfile << "dwTotalPageFile" << '\t' << status.dwTotalPageFile << endl;

outfile << "dwTotalPhy" << '\t' << status.dwTotalPhys << endl;

outfile << "dwTotalVirtual" << '\t' << status.dwTotalVirtual << endl;

outfile<<"&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&" << endl;

//以下一段显示内存基本信息，每次操作后内存基本信息不变

//如果要查看内存基本信息，可以取消注释

MEMORY\_BASIC\_INFORMATION mem; //内存基本信息

VirtualQuery(info.lpMinimumApplicationAddress,&mem, sizeof(MEMORY\_BASIC\_INFORMATION));

outfile << "AllocationBase" << '\t' << mem.AllocationBase << endl;

outfile << "AllocationProtect" << '\t' << mem.AllocationProtect << endl;

outfile << "BaseAddress" << '\t' << mem.BaseAddress << endl;

outfile << "Protect" << '\t' << mem.Protect << endl;

outfile << "RegionSize" << '\t' << mem.RegionSize << endl;

outfile << "State" << '\t' << mem.State << endl;

outfile << "Type" << '\t' << mem.Type << endl;

outfile<< "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;

/\*\*/

//释放信号量通知allocator可以执行下次内存分配活动

ReleaseSemaphore(allo, 1, NULL);

}

return 0;

}

void Allocator()

{

trace traceArray[5];

int index = 0;

FILE \* file;

#pragma warning(disable:4996)

file = fopen("opfile", "rb");

operation op;

SYSTEM\_INFO info;

DWORD temp;

GetSystemInfo(&info);

for (int i = 0; i < 30; i++)

{

WaitForSingleObject(allo, INFINITE); //等待tracker打印结果结束的信号量

cout << i << ':';

fread(&op, sizeof(operation), 1, file);

Sleep(op.time); //执行时间，如果想在指定时间执行可以取消注释

GetSystemInfo(&info);

switch (op.protection) //根据文件内容确定权限

{

case 0:

{

index = 0;

temp = PAGE\_READONLY;

break;

}

case 1:

temp = PAGE\_READWRITE;

break;

case 2:

temp = PAGE\_EXECUTE;

break;

case 3:

temp = PAGE\_EXECUTE\_READ;

break;

case 4:

temp = PAGE\_EXECUTE\_READWRITE;

break;

default:

temp = PAGE\_READONLY;

}

switch (op.oper)

{

case 0: //保留一个区域

{

cout << "reserve now" << endl;

traceArray[index].start = VirtualAlloc(NULL, op.block \* info.dwPageSize, MEM\_RESERVE, PAGE\_NOACCESS);

traceArray[index++].size = op.block \* info.dwPageSize;

cout << "starting address:" << traceArray[index - 1].start << '\t'

<< "size:" << traceArray[index - 1].size << endl;

break;

}

case 1: //提交一个区域

{

cout << "commit now" << endl;

traceArray[index].start=VirtualAlloc(traceArray[index].start, traceArray[index].size, MEM\_COMMIT, temp);

index++;

cout << "starting address:" << traceArray[index - 1].start << '\t' << "size:" << traceArray[index - 1].size << endl;

break;

}

case 2: //锁一个区域

{

cout << "lock now" << endl;

cout << "starting address:" << traceArray[index].start << '\t' << "size:" << traceArray[index].size << endl;

if (!VirtualLock(traceArray[index].start, traceArray[index++].size))

cout << GetLastError() << endl; //GetLastError()函数返回错误号

break;

}

case 3: //解锁一个区域

{

cout << "unlock now" << endl;

cout << "starting address:" << traceArray[index].start << '\t' << "size:" << traceArray[index].size << endl;

if (!VirtualUnlock(traceArray[index].start, traceArray[index++].size))

cout << GetLastError() << endl;

break;

}

case 4: //回收一个区域

{

cout << "decommit now" << endl;

cout << "start address:" << traceArray[index].start << '\t' << "size:" << traceArray[index].size << endl;

if(!VirtualFree(traceArray[index].start,traceArray[index++].size,

MEM\_DECOMMIT))

cout << GetLastError() << endl;

break;

}

case 5: //释放一个区域

{

cout << "release now" << endl;

cout << "starting address:" << traceArray[index].start << '\t' << "size:" << traceArray[index].size << endl;

if (!VirtualFree(traceArray[index++].start, 0, MEM\_RELEASE))

cout << GetLastError() << endl;

break;

}

default:

cout << "error" << endl;

}

ReleaseSemaphore(trac, 1, NULL); //释放信号量通知tracker可以打印信息

}

}

int main()

{

DWORD dwThread;

HANDLE handle[2];

//生成两个线程

handle[0] = CreateThread(NULL, 0, (LPTHREAD\_START\_ROUTINE)Tracker, NULL, 0, &dwThread);

handle[1] = CreateThread(NULL, 0, (LPTHREAD\_START\_ROUTINE)Allocator, NULL, 0, &dwThread);

//生成两个信号量

allo = CreateSemaphore(NULL, 0, 1, "allo");

trac = CreateSemaphore(NULL, 1, 1, "trac");

//等待线程执行的执行结果，在退出

WaitForMultipleObjects(2, handle, TRUE, INFINITE);

return 0;

}

**步骤7：**将“opfile”文件复制到该项目文件夹下。

**步骤8：**按“F5”开始调试，注意路径里不要含有中文。

**步骤9：**按暂停按钮可暂停程序的执行，按终止按钮可终止程序的执行。



操作能否正常进行？如果不行，则可能的原因是什么？

操作能够正常进行，如果不行，可能是因为文件中包含中文字符，或文件路径包含中文。

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

运行结果是：

文本

描述已自动生成

图片 4 运行结果1

文本

描述已自动生成

图片 5 运行结果2

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

结果分析：依次分析保留、提交、锁、解锁、回收、释放一个区域的操作引起的内存状态的变化。

保留内存主要影响虚拟内存空间。它会在地址空间中分配区域，但未实际分配物理内存或页面文件。

表格

描述已自动生成

图片 6 如图0号dwAvailVirtual – 1号dwAvailVirtual== 1,638,400,000，保留的虚拟空间大小和我们在前面opfile里要的空间块空间大小一样

表格

描述已自动生成

图片 7 如图1号dwAvailVirtual – 2号dwAvailVirtual== 1,638,400,000，保留的虚拟空间大小和我们在前面opfile里要的空间块空间大小一样，

图2号dwAvailVirtual – 3号dwAvailVirtual== 409,600,000，保留的虚拟空间大小和我们在前面opfile里要的空间块空间大小一样，

图3号dwAvailVirtual – 4号dwAvailVirtual== 1,228,800,000，保留的虚拟空间大小和我们在前面opfile里要的空间块空间大小一样。

表格

描述已自动生成

图片 8 4号dwAvailVirtual – 5号dwAvailVirtual== 2,048,000,000，保留的虚拟空间大小和我们在前面opfile里要的空间块空间大小一样。

**0-4号为止，对虚拟内存的reserve操作结束。5-9号，开始commit操作，开始对物理内存提交分配申请。**

5号dwAvailPhys – 6号dwAvailPhys == 2,420,736，

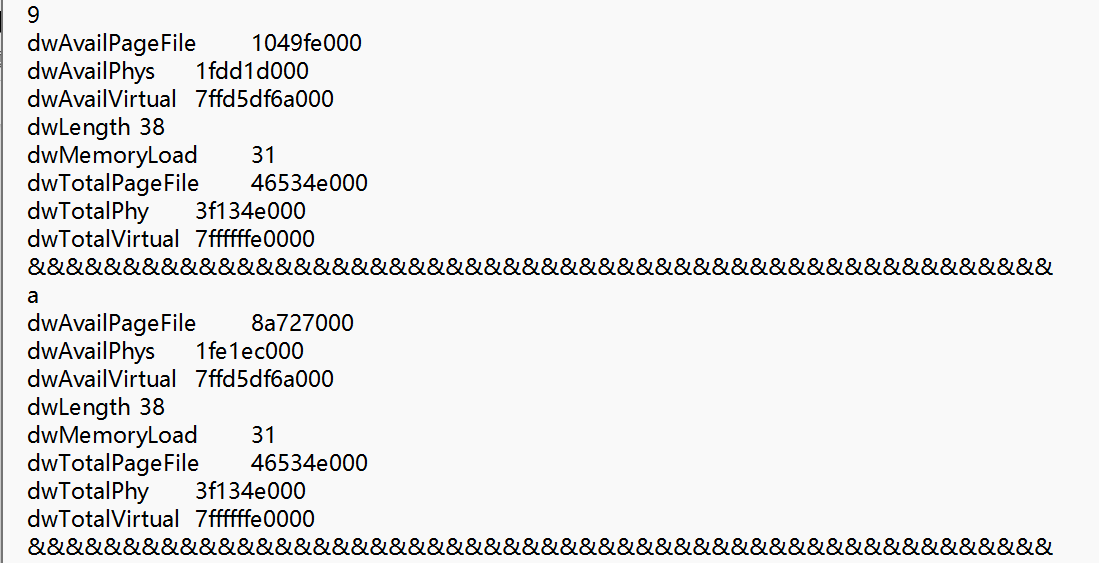
5号dwAvailPageFile– 6号dwAvailPageFile== 1,642,594,304，

6号dwAvailPhys – 7号dwAvailPhys == 6,701,056，

6号dwAvailPageFile– 7号dwAvailPageFile== 1,640,632,320，

7号dwAvailPhys – 8号dwAvailPhys == 6,467,584

7号dwAvailPageFile– 8号dwAvailPageFile== 411,803,648。



图片 9 8号dwAvailPhys – 9号dwAvailPhys == -3,231,744，

8号dwAvailPageFile– 9号dwAvailPageFile== 1,228,800,000，

9号dwAvailPhys – a号dwAvailPhys == -5,042,176

9号dwAvailPageFile– a号dwAvailPageFile== 2,049,798,144。

提交内存将保留区域的内存分配给物理页面或页面文件。这一步使保留的内存可用，但会减少物理内存和页面文件的可用空间，这时

dwAvailPhys 减少：物理内存用于支持提交的页面。

dwAvailPageFile 减少：页面文件分配更多空间。

**5-9号为止，对虚拟内存的commit操作结束。a-e号，开始lock操作，开始对物理内存锁定。**

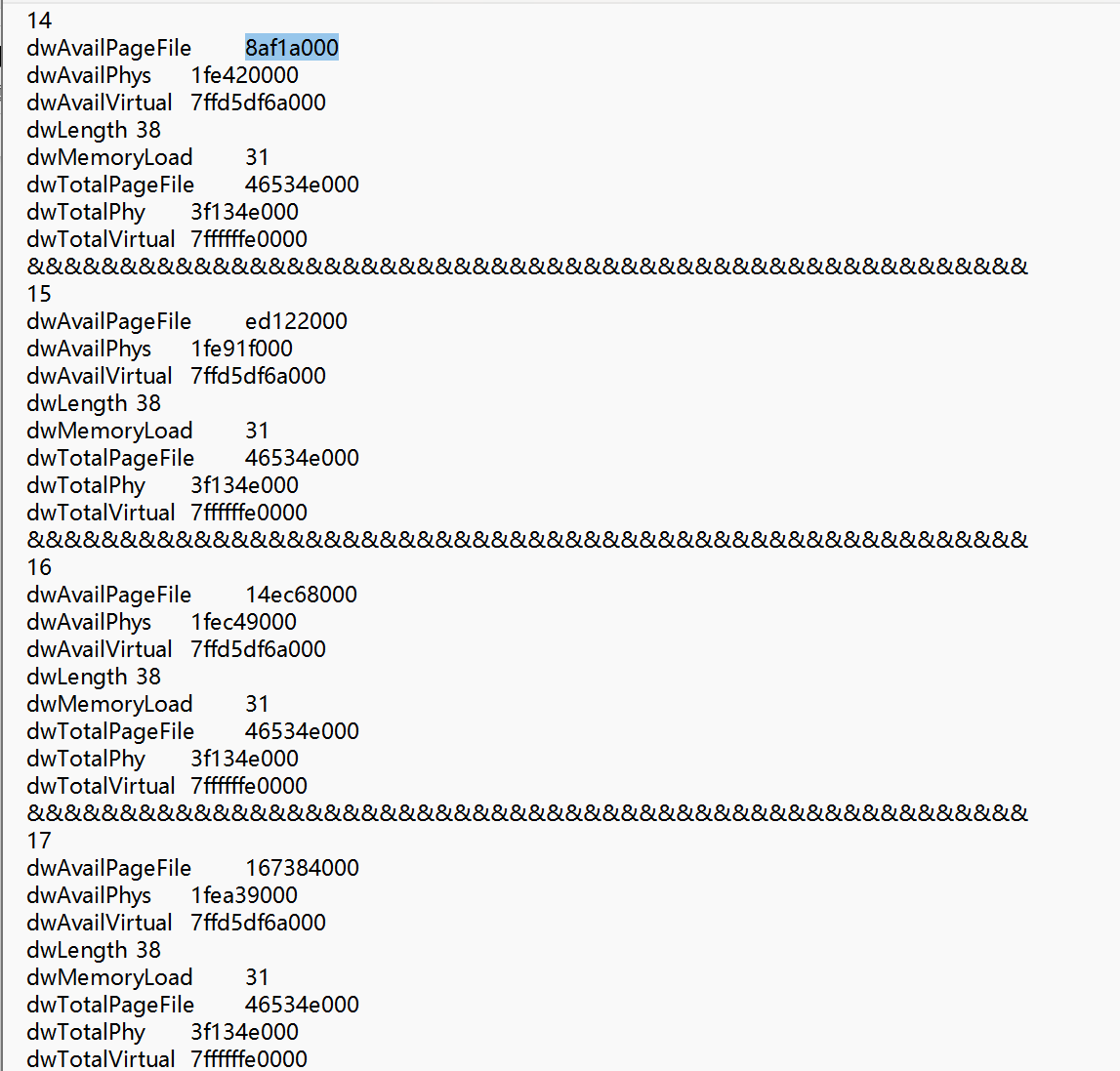
锁定内存会确保被锁定的内存始终驻留在物理内存中而不会被交换到页面文件：

* **dwAvailPhys 几乎不变**：物理内存被锁定。
* **dwAvailPageFile 不变**，因为锁定内存不会影响页面文件。
* **dwAvailVirtual 不变**，因为没有新增保留的虚拟内存。

**a-e号为止，对虚拟内存的lock操作结束。F-13号，开始unlock操作，开始对物理内存解锁。**

解锁内存会将内存的锁定状态解除，允许其被交换至页面文件：

* **dwAvailPhys 几乎不变**。
* **dwAvailPageFile 不变**。
* **dwAvailVirtual 不变**。



图片 10 15号dwAvailPhys – 14号dwAvailPhys == 5,238,784，

15号dwAvailPageFile– 14号dwAvailPageFile== 1,646,297,088，

16号dwAvailPhys – 15号dwAvailPhys == 3,317,760，

16号dwAvailPageFile– 15号dwAvailPageFile== 1,639,211,008。

**F-13号为止，对虚拟内存的unlock操作结束。14-18号，开始decommit操作，开始对物理内存撤销。**

回收内存会释放提交的页面，保持保留状态。回收后：

* **dwAvailPhys 增加**：释放了提交的物理页面。
* **dwAvailPageFile 增加**：页面文件也释放相应空间。
* **dwAvailVirtual 不变**：保留的虚拟内存仍然存在。

表格

低可信度描述已自动生成

图片 11 1a号dwAvailVirtual – 19号dwAvailVirtual== 1,638,400,000，

1b号dwAvailVirtual – 1a号dwAvailVirtual== 1,638,400,000，

1c号dwAvailVirtual – 1b号dwAvailVirtual== 409,600,000，

1d号dwAvailVirtual – 1c号dwAvailVirtual== 1,228,800,000，

1e号dwAvailVirtual – 1d号dwAvailVirtual== 2,049,048,576。

**14-18号为止，对虚拟内存的decommit操作结束。19-1d号，开始release操作，开始对虚拟保留内存释放。**

释放内存会完全释放保留的虚拟地址空间，归还给系统：

* **dwAvailVirtual 增加**：虚拟地址空间恢复。
* **dwAvailPhys 和 dwAvailPageFile 几乎不变**，因为只影响虚拟地址。

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

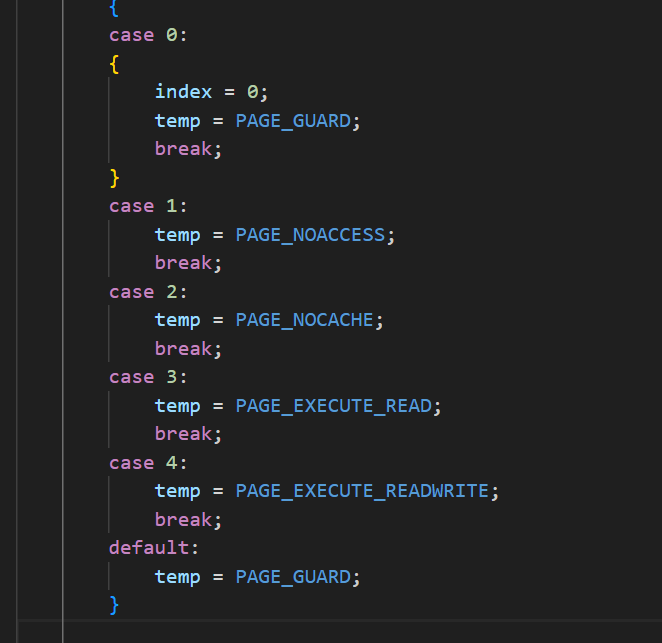
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**选作：**在以上源代码的基础上，编写一个程序。

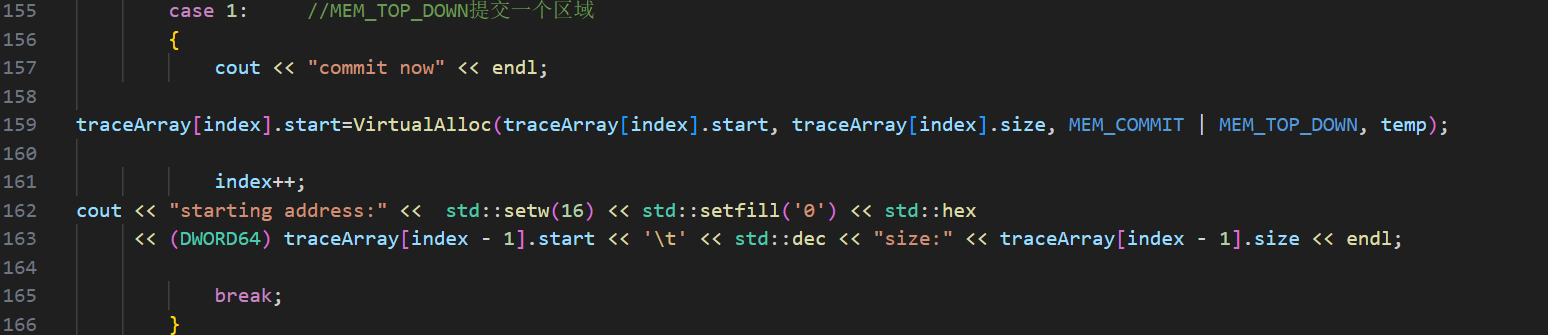
创建两个线程，一个用于内存分配，另一个用于跟踪内存的分配情况并打印信息。将virtualalloc函数的参数ftallocationtype分别改为MEM\_RESET或MEM\_TOP\_DOWN，将flprotect参数分别改为PAGE\_GUARD、PAGE\_NOACCESS或PAGE\_NOCACHE，再进行本练习的各项操作，再查看内存分配的各个结果，分析原因。尝试调换分配、回收、内存复位、加锁、解锁、提交、回收的次序，查看结果，井分析原因。

MEM\_RESET、MEM\_TOP\_DOWN、PAGE\_GUARD、PAGE\_NOACCESS、PAGE\_NOCACHE属性的含义在MSDN中均有详细介绍，请读者自行查阅。

请描述你所做的工作：



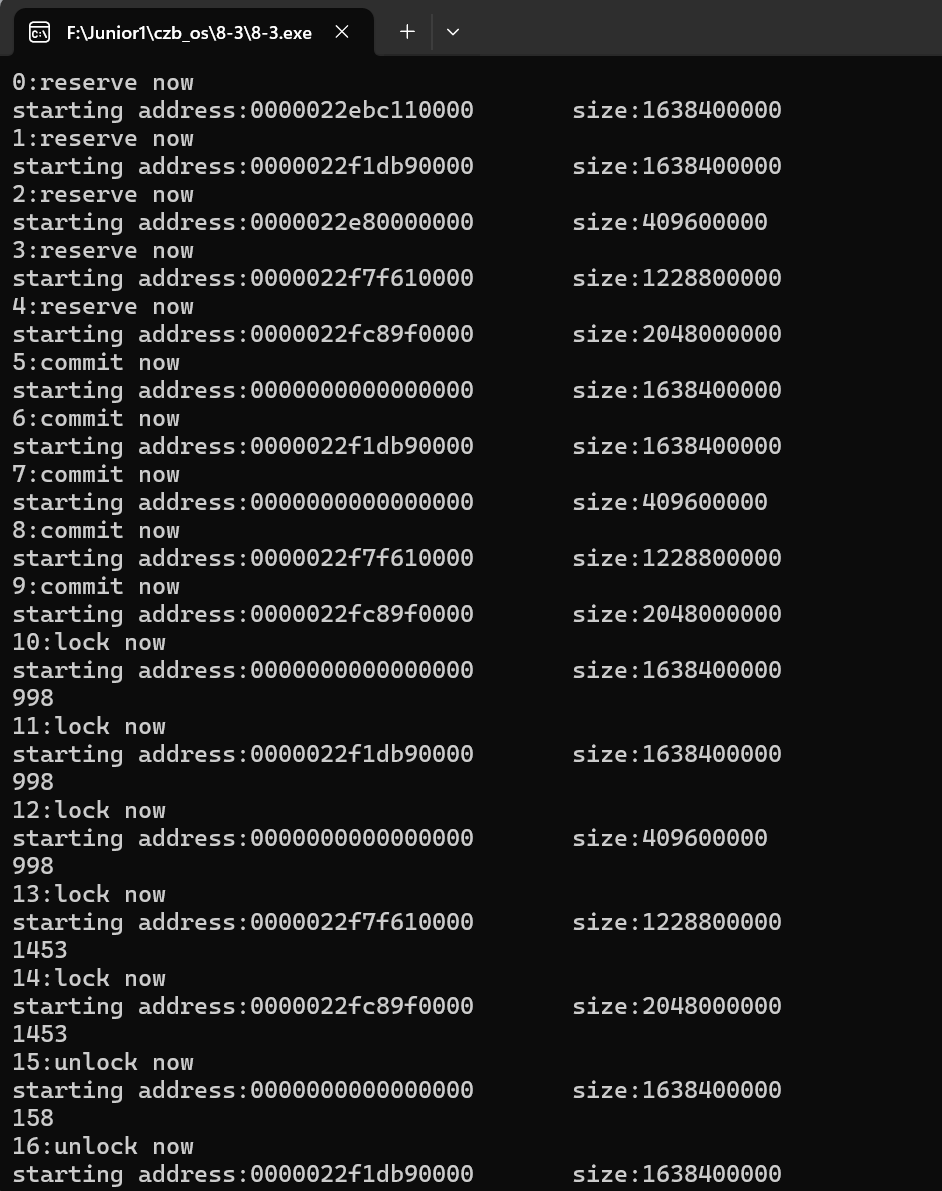
图片 12 修改代码将原本的flprotect参数更改为PAGE\_GUARD、PAGE\_NOACCESS、PAGE\_NOCACHE



图片 13 将commit更改为TOP DOWN形式



图片 14 将release更改为reset



图片 15 控制台输出1  
commit操作时部分操作返回起始地址为 0000000000000000，说明 VirtualAlloc 在提交时失败了，因为PAGE\_GUARD和PAGE\_NOCACHE状态下无法commit。  
部分锁定失败，返回错误码 998 或 1453

错误 998 可能是因为提交失败导致锁定的起始地址为 0000000000000000。

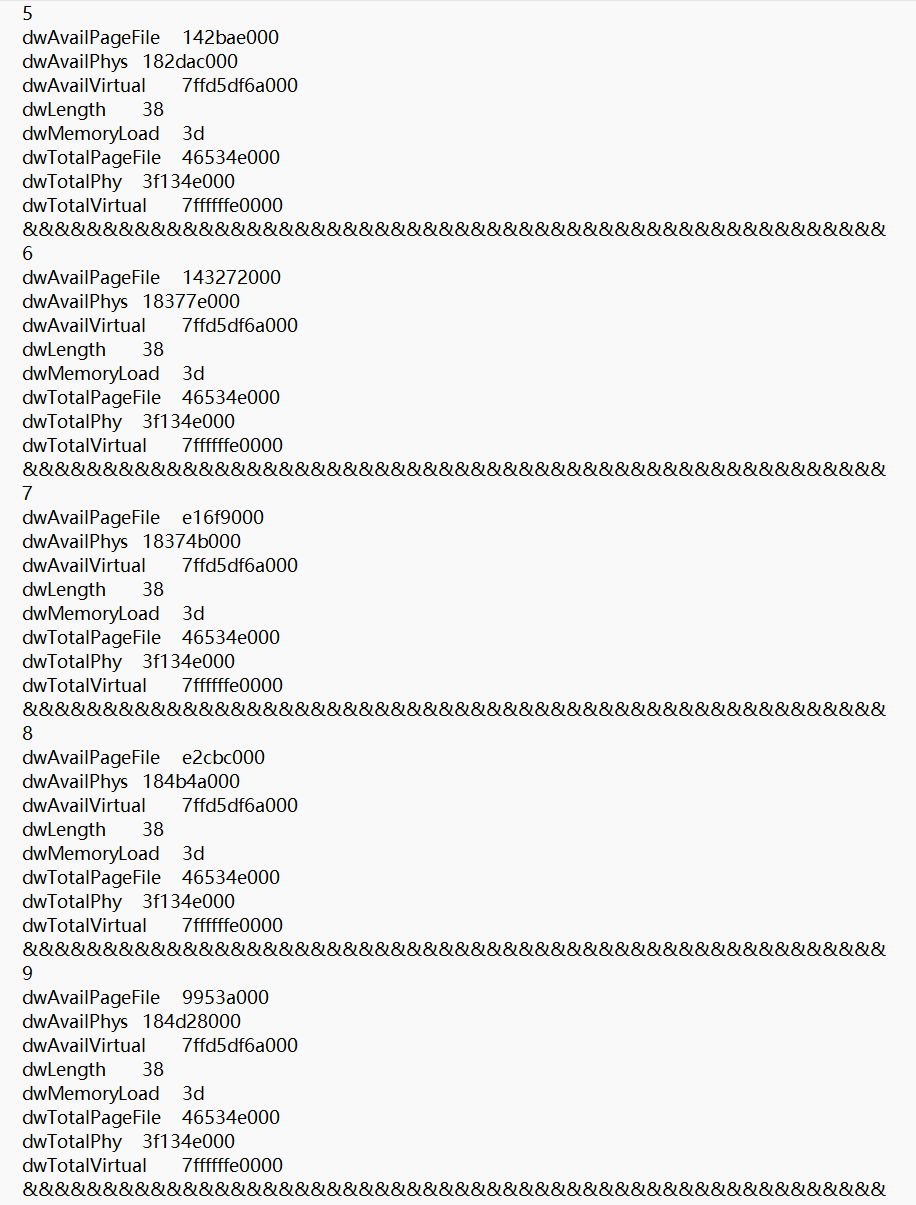
 错误 1453 表明锁定的内存超出了系统的工作集限制。

文本

中度可信度描述已自动生成

图片 16 控制台输出2  
unlock，操作大部分成功，但仍然对无效地址进行了操作。  
decommit操作，部分回收失败，返回错误码 487，地址为 0000000000000000 的区域在之前的提交阶段已经失败，因此回收自然会失败。  
reset操作返回错误码 87，出现参数无效。



图片 17 观察输出文件out.txt可以发现虚拟地址被正确保留。  


图片 18 commit操作有成功和失败的，主要观察PageFile变换大小，PAGE\_GUARD和PAGE\_NOCACHE状态下无法commit。

加锁也有因同样原因无法lock的，但unlock操作均正常。



图片 19 MEM\_RESET与MEM\_RELEASE不同，在调用MEM\_RESET时，最后虚拟内存不会被释放。

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_