《**Windows**核心编程》实验报告

课程： Windows核心编程

实验名称：Windows内存，虚拟内存和内存映射文件

姓名： 杨佳伲 班级： 信安实验221

# 学号： 2022132006 实验报告日期： 2024/12/XX

一、实验名称

**Windows内存、虚拟内存和内存映射文件**

二、实验环境（详细说明运行的系统、平台及代码等）

1．操作系统：Windows；

2. 运行环境：vs2019。

## 三、实验目的

１、掌握windows内存体系结构，理解进程虚拟地址，虚拟地址空间分区，地址

空间中区域，给区域调拨物理存储器的概念和基本过程

２、学会使用VirtualAlloc，VirtualFree，以及VirtualQuery虚拟内存相关函数使用

３、 掌握内存映射文件基本概念，实现步骤和相关api函数应用，采用内存映射文

件进行数据共享和文件分割。

## 四、实验内容、步骤及结果

### １．实验内容

1、采用虚拟内存函数VirtualAlloc，以及虚拟内存相关函数使用，分配一个60kb地址

预订和物理调拨，并在分配地址空间写入数据；采用VirtualQuery进行查询虚拟内存情况，

并然后调用VirtualFree释放虚拟内存。

2、在给得课件代码基础上，采用内存映射文件实现两个进程数据共享，写进程，通

过键盘输入数据，然后写入数据。然后读进程，则把写进程写入内存的数据读取出来。

3、采用内存文件映射，对一个大文件（大于2G的文件）进行读，并把该文件分割成

多个子文件，保存在磁盘中。

### ２．实验步骤

1、编写虚拟内存分配和调拨，查询和释放程序

1）采用SDK或mfc或控制台创建一个应用程序。

2）然后添加代码，采用虚拟内存函数VirtualAlloc，以及虚拟内存相关函数使用，分

配一个64kb地址预订和物理调拨，并将字符串“202200000”**（各自学号）**赋值到虚拟内存；

采用VirtualQuery进行查询虚拟内存情况，并然后调用VirtualFree释放虚拟内存。

虚拟内存

// experience3.cpp : 定义控制台应用程序的入口点。

//

#include "stdafx.h"

#include <string.h>

#include <Windows.h>

int \_tmain(int argc, \_TCHAR\* argv[]) {

char szText[] = "2022132006"; //分配一个64kb地址预订和物理调拨

char\* pszData = (char\*)VirtualAlloc(NULL, 64 \* 1024, MEM\_RESERVE | MEM\_COMMIT, PAGE\_READWRITE);

if (pszData == NULL)

{

printf("分配失败.\n");

}

else

{

strcpy(pszData, szText); //将字符串“2022132006”（各自学号）赋值到虚拟内存

printf("已将学号2022132006赋值到虚拟内存.\n");

}

MEMORY\_BASIC\_INFORMATION mbi; //查询虚拟内存分配

VirtualQuery(pszData, &mbi, sizeof(mbi));

printf("使用VirtualQuery获得的信息：\n"

"BaseAddress:0x%.8x\nAllocationBase:0x%.8x\n"

"AllocationProtect:0x%.8x\nRegionSize:%u\n"

"State:0x%.8x\nProtect:0x%.8x\nType:0x%.8x\n",

mbi.BaseAddress, mbi.AllocationBase,

mbi.AllocationProtect, mbi.RegionSize,

mbi.State, mbi.Protect, mbi.Type

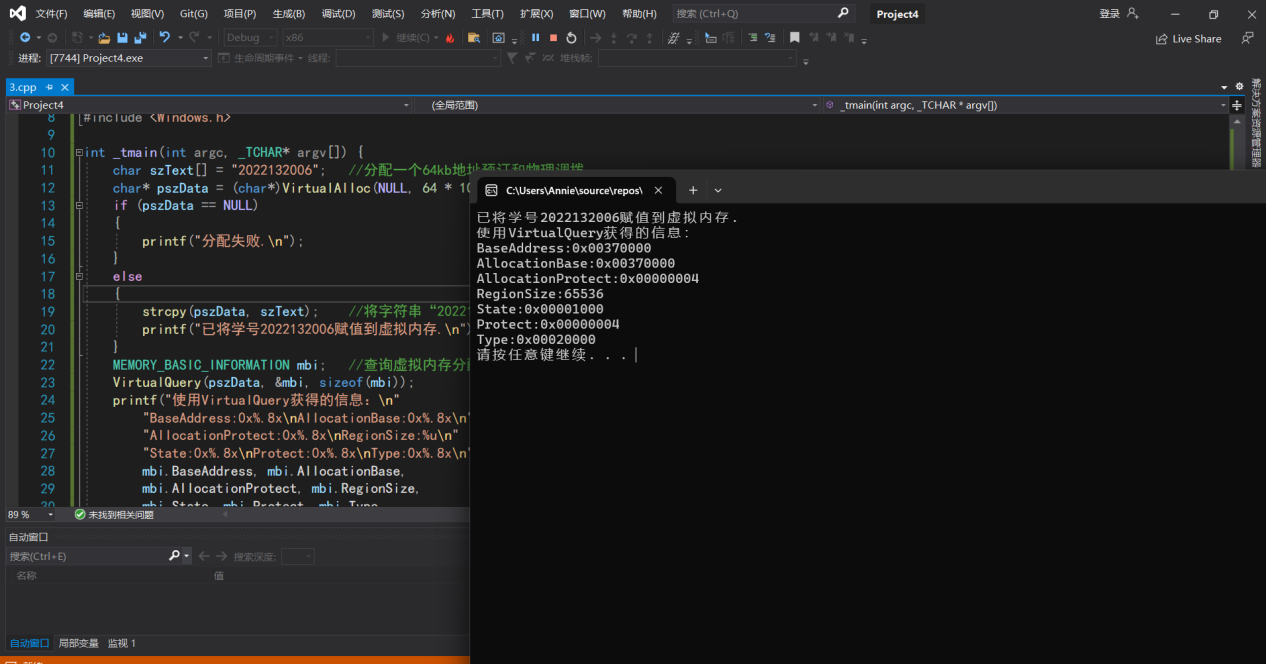
); //释放虚拟内存

VirtualFree(pszData, 0, MEM\_RELEASE);

system("pause");

return 0;

}



2、完成两进程的数据共享，

首先写进程端内存映射文件的过程,具体步骤如下：

1）调用CreateFileMapping(INVALID\_HANDLE\_VALUE, …….)完成文件映射内核对象

2）调用MapViewofFile函数将内存映射文件进程地址空间

3）通过MapViewofFile函数返回值，地址指针值获得对共享内存块的操作，写入需要

写入的数据

4）当读进程读取数据之后，然后调用UnmapViewofFile()函数释放进程空间地址，然

后调用CloseHandle函数释放文件映射对象和文件对象

然后读进程则完成数据读取和显示，具体步骤如下：

1） 调用OpenFileMapping函数打开写进程的文件映射对象

2） 调用MapViewofFile函数映射进程地址空间，并获得内存的指针

3） 利用MapViewofFile函数获得指针，对共享内存块数据进行读操作，并输出显示

4）当读取完进程数据，然后调用UnmapViewofFile()函数释放进程空间地址，然后调用

CloseHandle函数释放文件映射对象和文件对象

设计一个内存监视器

#include "stdafx.h"

#include <windows.h>

#include <Lmcons.h>

void ShowSystemInfo(); //获取并显示硬件相关信息

void GlobalMemoryStatusx(); //获取并显示内存信息

void GetNames(); //获取并显示计算机名、用户名等信息

int \_tmain(int argc, \_TCHAR\* argv[])

{

ShowSystemInfo();

GlobalMemoryStatusx();

GetNames();

getchar();

return 0;

}

void ShowSystemInfo() //获取并显示硬件相关信息

{

SYSTEM\_INFO si;

GetSystemInfo(&si);

printf("cpu页面大小：0x%.8X\n", si.dwPageSize);

printf("进程的可用地址空间的最小内存地址:0x%.8X\n进程的可用地址空间的最大内存地址;0x%.8X\n", si.lpMinimumApplicationAddress, si.lpMaximumApplicationAddress);

printf("\n处理器架构:");

switch (si.wProcessorArchitecture)

{

case PROCESSOR\_ARCHITECTURE\_INTEL:

printf("intel");

printf(" CPU vendor is %d", si.wProcessorLevel);

break;

case PROCESSOR\_ARCHITECTURE\_IA64:

printf("64 bits intel");

break;

case PROCESSOR\_ARCHITECTURE\_AMD64:

printf("64 bits AMD");

break;

case PROCESSOR\_ARCHITECTURE\_UNKNOWN:

printf("UNKNOWN");

break;

}

printf("\n");

}

void GetNames() //获取计算机名、用户名等信息

{

DWORD dwComputerNameLen = MAX\_COMPUTERNAME\_LENGTH + 1;

DWORD dwUserNameLen = UNLEN + 1;

TCHAR szComputerName[MAX\_COMPUTERNAME\_LENGTH + 1];

TCHAR szUserName[UNLEN + 1];

if (!SetComputerName("My\_Computer"))

{

printf("Set Error %d", GetLastError());

}

GetComputerName(szComputerName, &dwComputerNameLen);

printf("计算机名：%s\n", szComputerName);

dwComputerNameLen = MAX\_COMPUTERNAME\_LENGTH + 1;

GetComputerNameEx(ComputerNameDnsHostname, szComputerName, &dwComputerNameLen);

printf("ComputerNameDnsHostname: %s\n", szComputerName);

dwComputerNameLen = MAX\_COMPUTERNAME\_LENGTH + 1;

GetComputerNameEx(ComputerNamePhysicalNetBIOS, szComputerName, &dwComputerNameLen);

printf("ComputerNamePhysicalNetBIOS: %s\n", szComputerName);

GetUserName(szUserName, &dwUserNameLen);

printf("用户名：%s\n", szUserName);

}

void GlobalMemoryStatusx()

{

MEMORYSTATUS status; //定义存放内存信息的变量

GlobalMemoryStatus(&status); //调用GlobalMemoryStatus函数获取内存信息

//对得到的内存信息处理现实处理

printf("总内存%d KB\n", status.dwTotalPhys / 1000); //现实内存大小

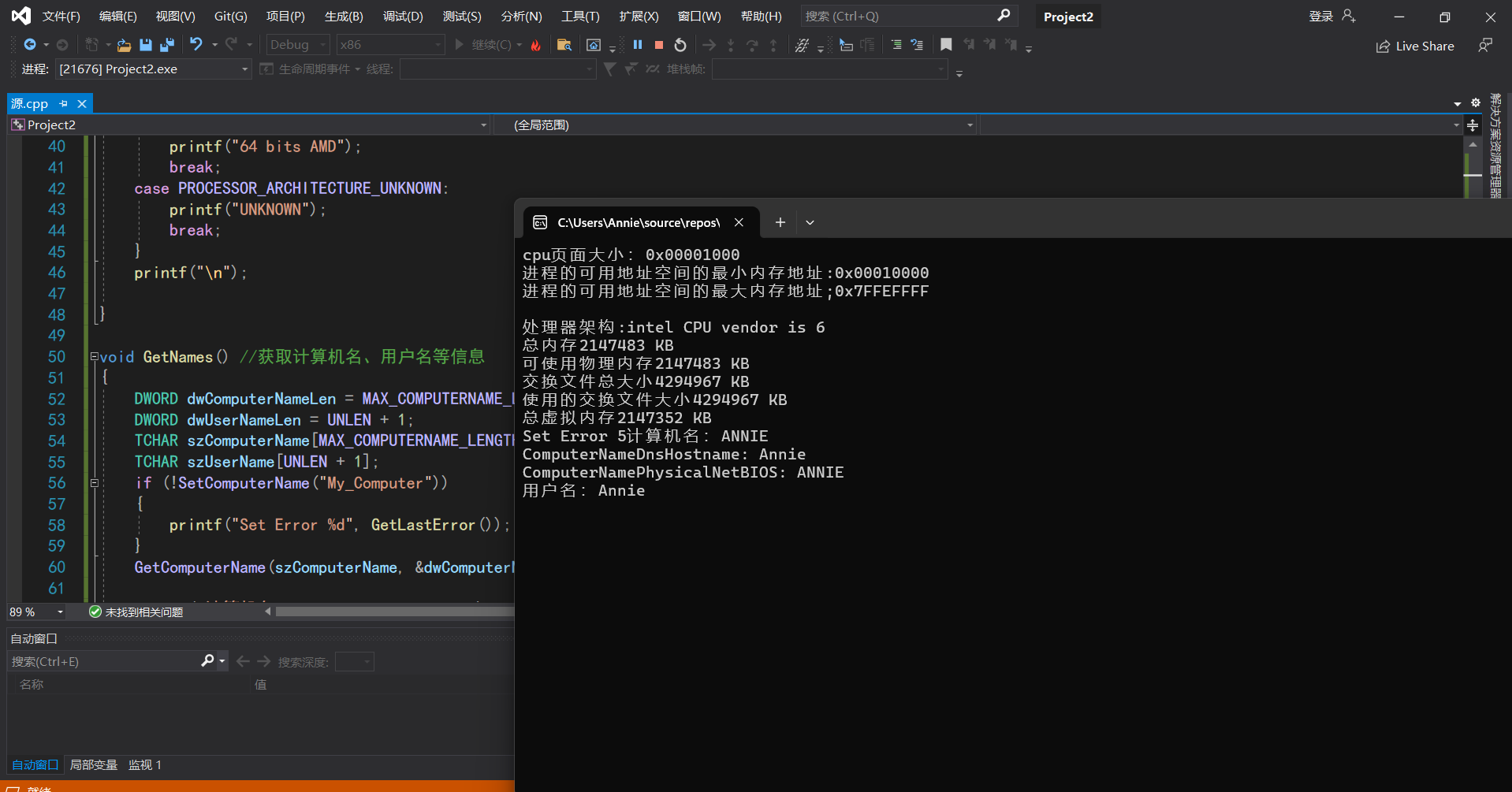
printf("可使用物理内存%d KB\n", status.dwAvailPhys / 1000);

printf("交换文件总大小%d KB\n", status.dwTotalPageFile / 1000);

printf("使用的交换文件大小%d KB\n", status.dwAvailPageFile / 1000);

printf("总虚拟内存%d KB\n", status.dwTotalVirtual / 1000);

}



3、实现对大文件读取和分割，具体步骤如下：1） 如下是分割大文件，读取大文件代码

//读大文件

SYSTEM\_INFO sinf;

GetSystemInfo(&sinf);

HANDLE

hFile=CreateFile(TEXT("d://huge.txt"),GENERIC\_WRITE|GENERIC\_READ,0,NULL,O

PEN\_EXISTING,FILE\_ATTRIBUTE\_NORMAL,NULL);//打开文件

DWORD dwFileSizeHigh;

\_int64 qwFileSize=GetFileSize(hFile,&dwFileSizeHigh);//获得文件大小

qwFileSize+=(((\_int64)dwFileSizeHigh)<<32);//将高32位加到文件低32位上

HANDLE

hFileMap=CreateFileMapping(hFile,NULL,PAGE\_READWRITE,0,0,NULL);

//创建一个文件内核对象，大小默认为文件大小

CloseHandle(hFile);

\_int64 qwFileOffset=0;//每次映射文件大小，初始值为0

while(qwFileSize>0){

DWORD dwBytesInBlock=sinf.dwAllocationGranularity;//预定空间的分配

if(qwFileSize<sinf.dwAllocationGranularity)

dwBytesInBlock=(DWORD)qwFileSize;

PCHAR pbFile=(PCHAR)MapViewOfFile(hFileMap,

FILE\_MAP\_WRITE,

(DWORD)(qwFileOffset>>32),//高32位

(DWORD)(qwFileOffset&0xFFFFFFFF),//低32位

dwBytesInBlock);

//为文件的数据预定一块地址空间区域并将文件的数据作为物理存储器调拨给区域

cout<<"content: "<<pbFile<<endl;

UnmapViewOfFile(pbFile);//从进程空间撤销对文件数据的关联

qwFileOffset+=dwBytesInBlock;

qwFileSize-=dwBytesInBlock;

}

CloseHandle(hFileMap);

1. 分割之后，采用内存映射文件写入到子文件，代码类似

采用内存映射，实现进程数据共享

#include <Windows.h>

#include <stdio.h>

void main()

{

TCHAR szName[] = TEXT("2015122075"); // 内存映射对象的名称

char szData[100]; // 共享内存中的数据

LPVOID pBuffer; // 共享内存指针

HANDLE hMap = OpenFileMapping(FILE\_MAP\_ALL\_ACCESS,0,szName);

if (hMap != NULL)

{

pBuffer = MapViewOfFile(hMap, FILE\_MAP\_ALL\_ACCESS, 0, 0, 0); // 打开成功

printf("读到的共享内存中的数据是：%s \n", (char \*)pBuffer);

}

else

{

printf("请输入数据: "); //打开失败

scanf("%s", szData);

fflush(stdin);

hMap = CreateFileMapping(INVALID\_HANDLE\_VALUE,

NULL,

PAGE\_READWRITE,

0,

strlen(szData) + 1,

szName

);

pBuffer = MapViewOfFile(hMap, FILE\_MAP\_ALL\_ACCESS, 0, 0, 0); //映射对象的一个视图，得到指向共享内存的指针，设置里面的数据。

strcpy((char \*)pBuffer,szData);

printf("写入共享内存的数据：%s \n", (char \*)pBuffer);

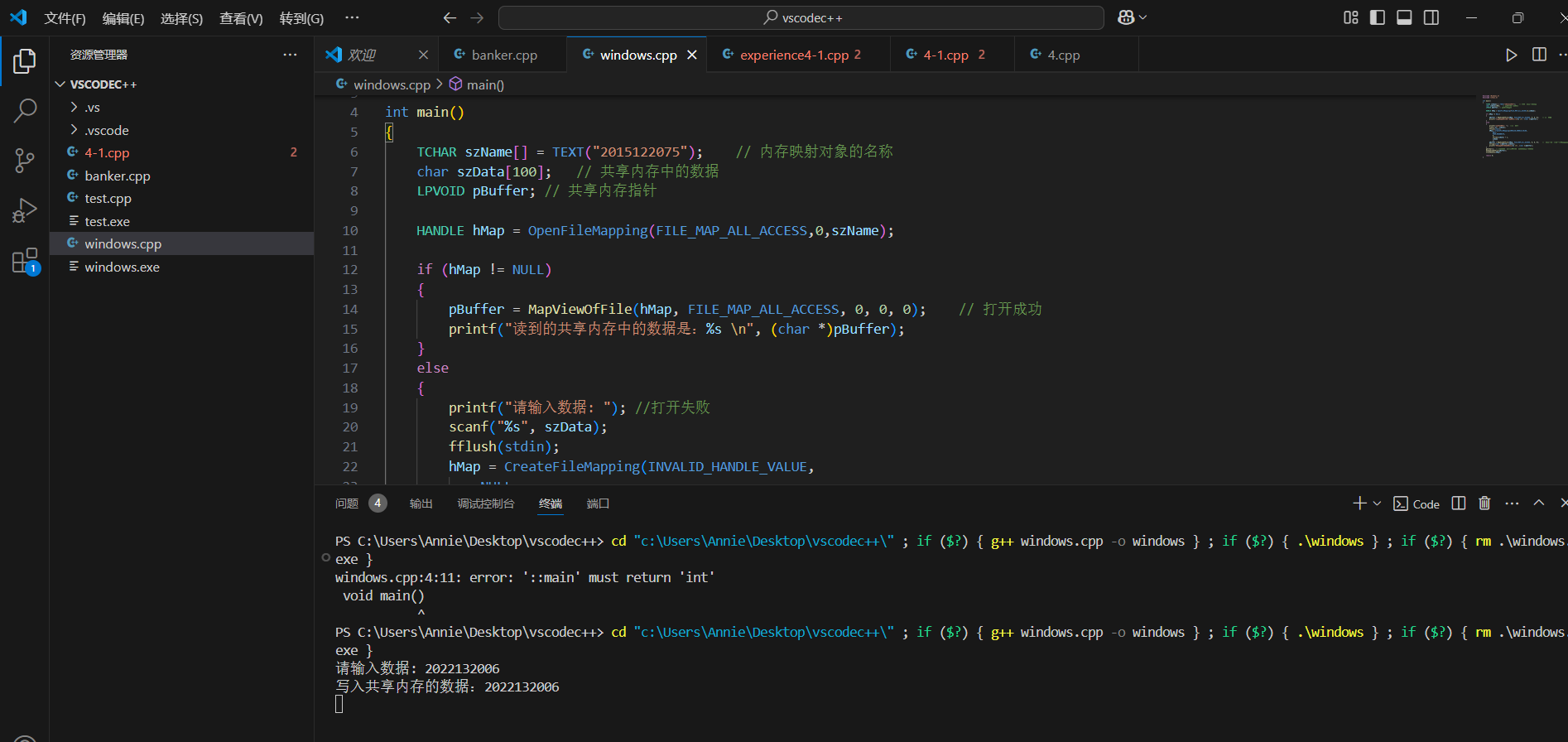
}

getchar(); // 解除文件映射，关闭内存映射文件的对象句柄。

UnmapViewOfFile(pBuffer);

CloseHandle(hMap);

}



采用内存映射，对大文件进行分割

// experience4-1.cpp : 定义控制台应用程序的入口点。

//

#include "stdafx.h"

#include <Windows.h>

#include <iostream>

#include <sstream>

#include <fstream>

#define iNumberOfFile 1000

using namespace std;

stringstream sstr1[iNumberOfFile];

string szFileName[iNumberOfFile];

string str[iNumberOfFile];

int main()

{

SYSTEM\_INFO sinf; //读大文件

GetSystemInfo(&sinf);

HANDLE hFile = CreateFile(TEXT("d://1//experience4-1.txt"), GENERIC\_WRITE | GENERIC\_READ, 0, NULL, OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, NULL);//打开文件

DWORD dwFileSizeHigh;

\_int64 qwFileSize = GetFileSize(hFile, &dwFileSizeHigh); //获得文件大小

qwFileSize += (((\_int64)dwFileSizeHigh) << 32); //将高32位加到文件低32位上

HANDLE hFileMap = CreateFileMapping(hFile, NULL, PAGE\_READWRITE, 0, 0, NULL); //创建一个文件内核对象，大小默认为文件大小

CloseHandle(hFile);

\_int64 qwFileOffset = 0; //每次映射文件大小，初始值为0

int iFileName = 1;

string szFilePath = "experience4-1\_";

while (qwFileSize > 0)

{

DWORD dwBytesInBlock = sinf.dwAllocationGranularity; //预定空间的分配

if (qwFileSize < sinf.dwAllocationGranularity)

dwBytesInBlock = (DWORD)qwFileSize;

PCHAR pbFile = (PCHAR)MapViewOfFile(hFileMap,

FILE\_MAP\_WRITE,

(DWORD)(qwFileOffset >> 32), //高32位

(DWORD)(qwFileOffset & 0xFFFFFFFF), //低32位

dwBytesInBlock); //为文件的数据预定一块地址空间区域并将文件的数据作为物理存储器调拨给区域

sstr1[iFileName] << iFileName;

sstr1[iFileName] >> str[iFileName];

szFileName[iFileName] = szFilePath + str[iFileName] + ".txt";

ofstream fout(szFileName[iFileName]);

if (fout)

{

cout << "Create: " << szFileName[iFileName] << " Success!" << endl;

fout << pbFile << endl;

}

else

cout << "Create: " << szFileName[iFileName] << " Error !" << endl;

fout.close();

iFileName++;

UnmapViewOfFile(pbFile); //从进程空间撤销对文件数据的关联

qwFileOffset += dwBytesInBlock;

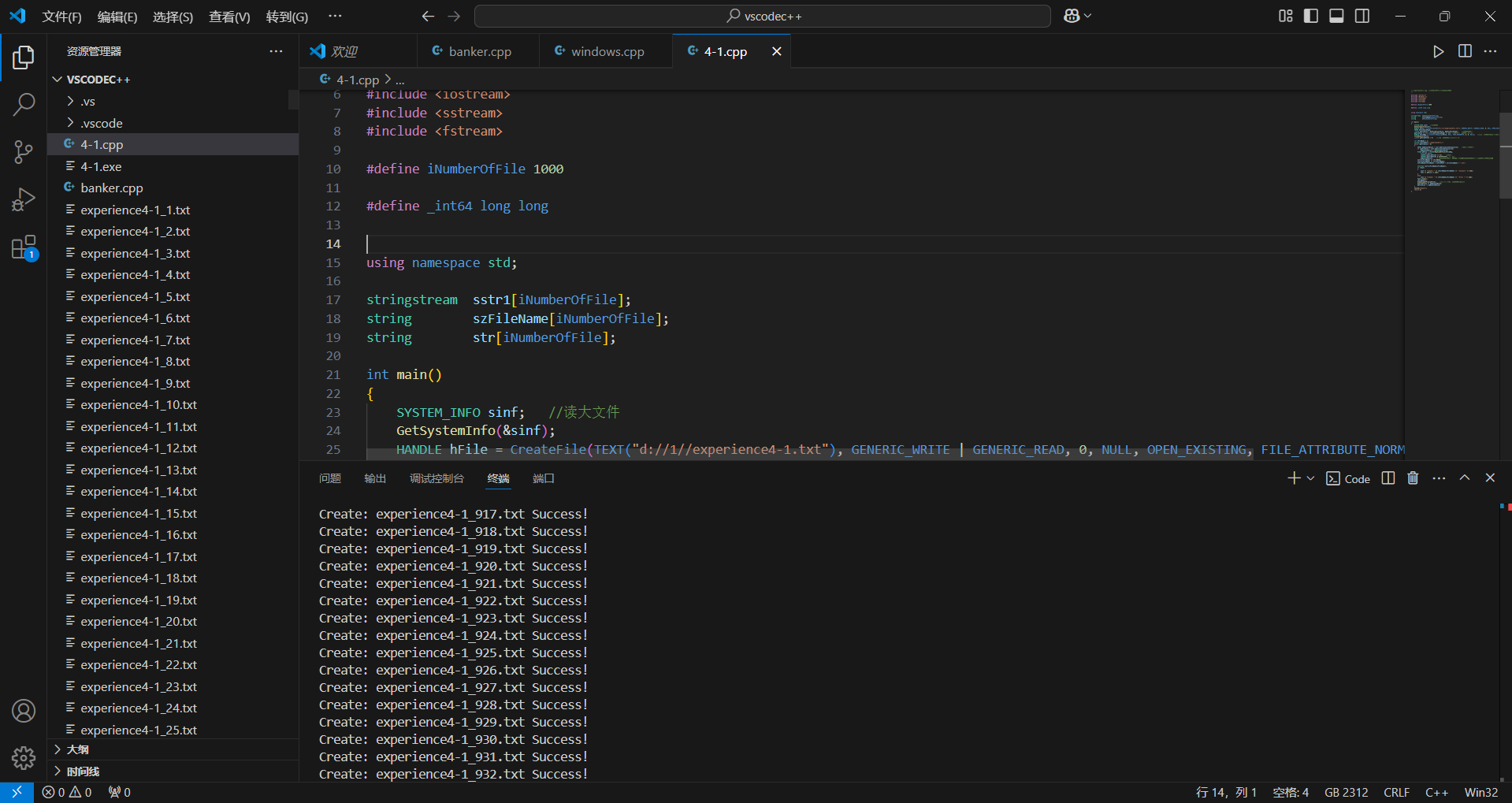
qwFileSize -= dwBytesInBlock;

}

system("pause");

return 0;

}



五、实验中的问题及解决办法。

掌握了关于内存和虚拟内存的相关函数，以及获取计算机信息的内容的一些函数。

在编写代码中遇到一些参数的类型不正确，后面后面将项目属性的字符集改为多字节字符集即可运行。

在查询完虚拟内存后需要将虚拟内存释放掉，不然会出错。

2