《**Windows**核心编程》实验报告

课程： Windows核心编程 实验名称：进程和线程编程

姓名： 杨佳伲 班级： 信安实验221

# 学号： 2022132006 实验报告日期： 2024/12/27

一、实验名称

# 进程和线程编程

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：Windows7；

2. 运行环境：vc6。

## 三、实验目的

1、 完成一个Windows窗口应用程序，熟悉Windows窗口应用结构，消息驱动

2、 掌握WNDCLASSEX结构体、CreateWindows等API函数，图标、光标。了解GDI中 DC概念，以及绘图函数

3、 并掌握进程和线程的概念、进程的状态

4、 掌握线程同步技术，包括线程用户模式下同步，例如临界区（关键段）线程同步，以及采用内核对象模式线程同步，例如事件，互斥量，信号量

5、 熟悉进程和线程API函数编程，要求能够完成进程创建，以及应用线程相关函数完成多线程编程，并完成多线程同步。

6、 熟悉process Explorer软件和spy++软件，使用这两款软件工具对内核对象，进程和线程相关信息进行查看。

## 四、实验内容、步骤及结果

### １．实验内容

1、 使用process Explorer软件和Spy++，并学会使用软件，查看内核对象，理解内核对象，进程和线程概念。

2、 1) 创建一个窗口应用程序，完成自定义图标（最小化图标和小图标），以及窗口光标，窗口标题名字;

2) 采用自创画笔和画刷，绘制一个多边形，长方形，椭圆，直线等图形 ；

3）在窗口应用程序添加菜单子项“打开记事本”，然后点击菜单实现，调用 C r e a t e T o o l h e l p 3 2 S n a p s h o t记事本应用程序是否打开，若没有，则调用 CreateProcess创建子进程，打开记事本功能。

4) 窗口应用程序点击鼠标左键

，则调用FindWindows找到记事本窗口，分别在记事本和应用程序窗口输出鼠标左键的坐标，并在记事本和窗口应用程序鼠标左键坐标点为圆心，输出圆形，并用自创画笔和画刷填充。

5)然后菜单项新建菜单子项“退出记事本”，退出记事本，调用结束进程函数TerminateProcess结束记事本进程。

4、 程序创建三个线程，一个主线程，主线程创建两个附加线程，采用事件或者其他线程同步机制实现线程同步：

1）第一个附加线程功能为：屏幕输入字符串，写入文件，字符串写入完，通知第二个附加线程。

2）第二个附加线程功能为：读取文件中字符串中，查找一个单词，若找到了该单词，通知主线程把找到的单词显示出来；

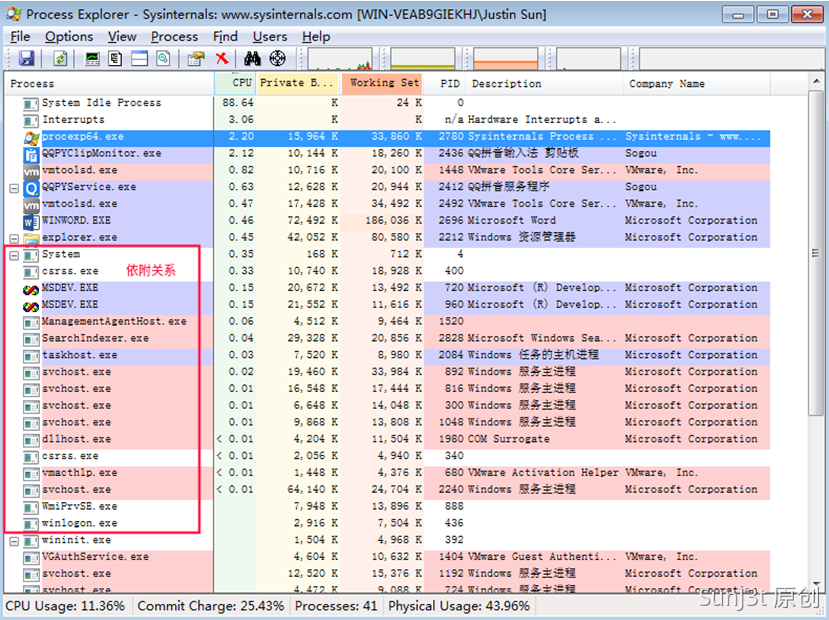
3）若附加线程退出时，还未查找到此单词，则通知主线程，打印出未找到此单词。

### ２．实验步骤

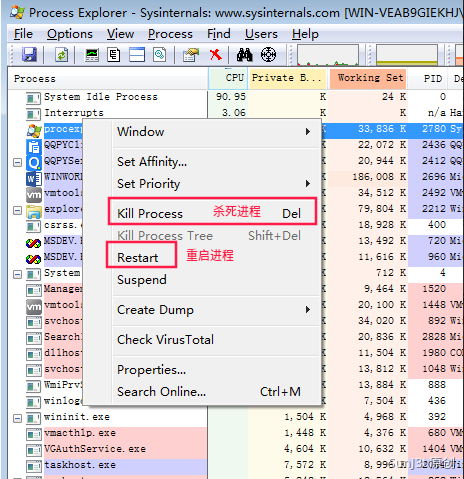
1、用process Explorer软件查看进程相关限制信息。以及process Explorer工具其他作

用 。 具 体 应 用 可 以 参 考 文 档 “ P R O C E X P 进 程 管 理 器 简 明 使 用 教 程 ” ， 下 载 地 址

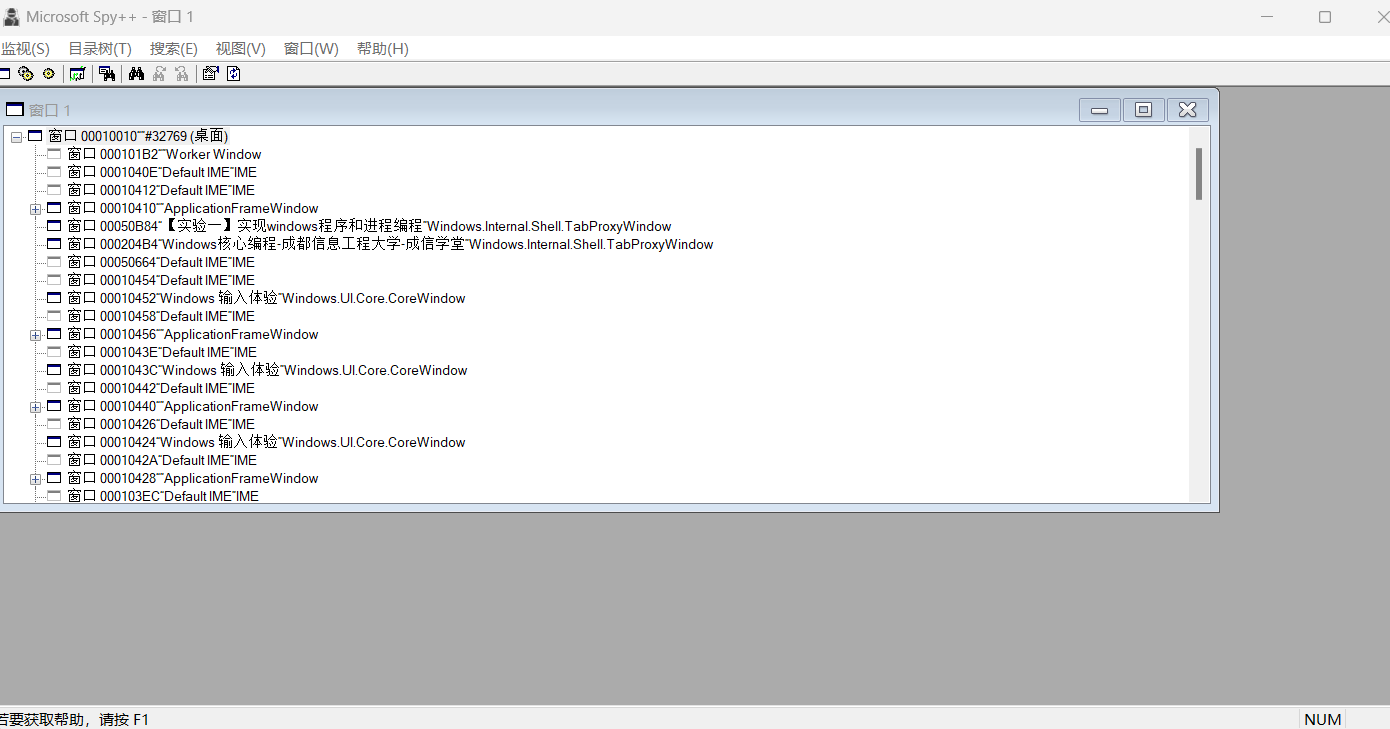
http://wenku.baidu.com/view/d57b2a97dd88d0d233d46a96.html?re=view###



Procexp除了可以查看进程，还可以拿来结束进程，重启进程等



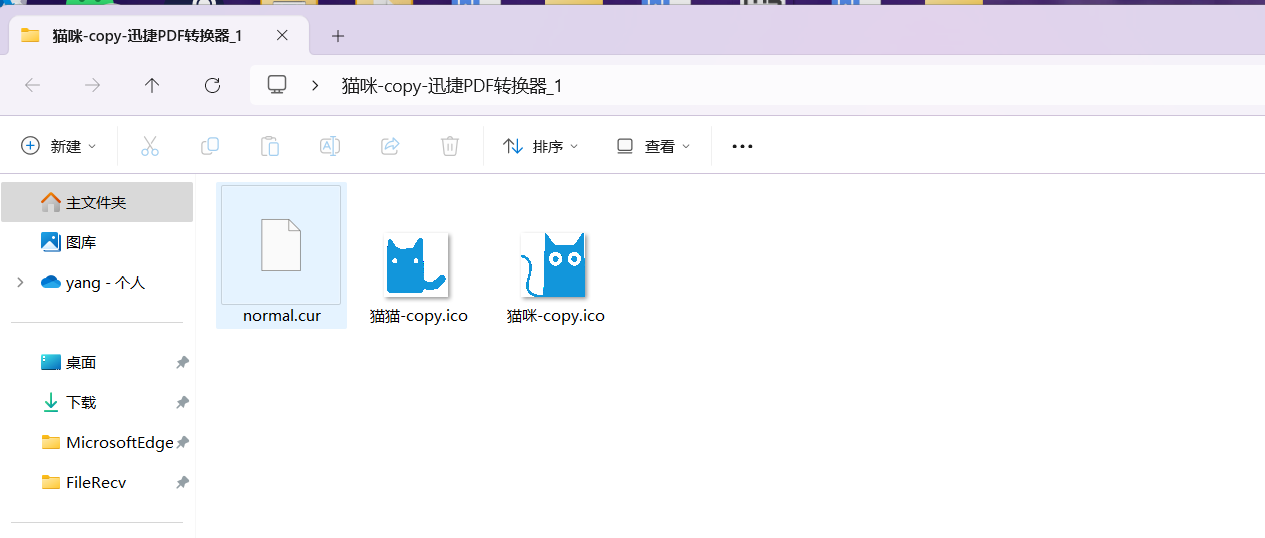
2、使用Visual studio tools的Spy++查看窗口相关信息，学会使用Spy++软件



3、1) 创建一个窗口应用程序，完成自定义图标（最小化图标和小图标），以及窗

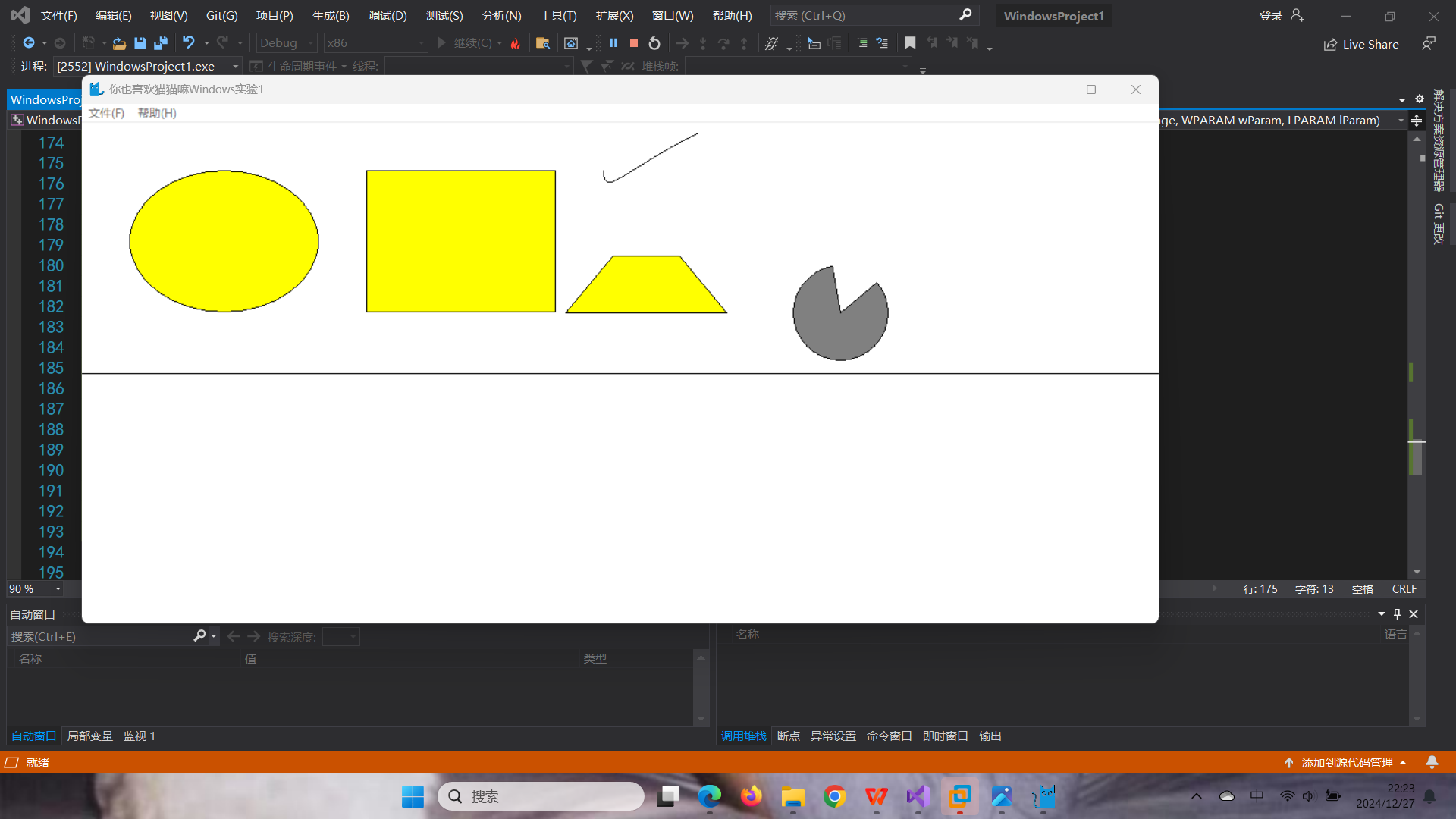
口光标，窗口标题名字;

网上自行找素材自定义，注意图标转换为icon格式，鼠标指针静态为cur格式，用的vs不支持动态指针ani格式的



2)采用自创画笔和画刷，绘制一个多边形，长方形，椭圆，直线，弧线，以及饼状

图等图形 ；



关键代码实现如下：

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)

{

HDC hdc;

PAINTSTRUCT ps;

RECT rect;

HBRUSH hBrush, hOldBrush;

HPEN hPen, hPen4, hOldPen;

POINT apt1[4] = { 510, 200,560, 140,630, 140,680, 200 };

static POINT bapt[4] = { 550,50,550,80,570,50,650,10 };//4个点

switch (message)

{

case WM\_COMMAND:

{

int wmId = LOWORD(wParam);

// 分析菜单选择:

switch (wmId)

{

case IDM\_ABOUT:

DialogBox(hInst, MAKEINTRESOURCE(IDD\_ABOUTBOX), hWnd, About);

break;

case IDM\_EXIT:

DestroyWindow(hWnd);

break;

default:

return DefWindowProc(hWnd, message, wParam, lParam);

}

}

break;

case WM\_PAINT:

{

PAINTSTRUCT ps;

HDC hdc = BeginPaint(hWnd, &ps);

// TODO: 在此处添加使用 hdc 的任何绘图代码...

/\*TextOut(hdc, 100, 200, L"helloworld", 10);\*/

GetClientRect(hWnd, &rect);

hPen = CreatePen(PS\_INSIDEFRAME, 100, RGB(255, 0, 0));

hBrush = CreateSolidBrush(RGB(255, 255, 0));

hOldBrush = HBRUSH(SelectObject(hdc, hBrush));

MoveToEx(hdc, rect.left, (rect.bottom - rect.top) / 2, NULL);

LineTo(hdc, rect.right, (rect.bottom - rect.top) / 2);

Rectangle(hdc, 300, 50, 500, 200); // 3

Ellipse(hdc, 50, 50, 250, 200); // 4

Polygon(hdc, apt1, 4);

DeleteObject(SelectObject(hdc, hOldBrush));

//弧线

hPen4 = (HPEN)GetStockObject(BLACK\_PEN);//获得Windows预设的画笔画刷

hOldPen = (HPEN)SelectObject(hdc, hPen4);//选入画笔

PolyBezier(hdc, bapt, 4);// 4个点: 1和4是端点, 2.3点是控制点

// 7个点 : 1.4.7是端点, 其余是控制点

SelectObject(hdc, hOldPen);//恢复先前画笔

DeleteObject(hPen4);//记得自创画笔一定删除

//饼图

hdc = GetDC(hWnd);

BeginPath(hdc);

SelectObject(hdc, GetStockObject(GRAY\_BRUSH));

MoveToEx(hdc, 800, 200, (LPPOINT)NULL);

AngleArc(hdc, 800, 200, 50, 100, 300);

LineTo(hdc, 800, 200);

EndPath(hdc);

StrokeAndFillPath(hdc);

ReleaseDC(hWnd, hdc);

EndPaint(hWnd, &ps);

}

break;

case WM\_DESTROY:

PostQuitMessage(0);

break;

default:

return DefWindowProc(hWnd, message, wParam, lParam);

}

return 0;

}

3）按下键盘上某键，则调用CreateProcess创建子进程，打开记事本程序，则向记

事 本 中 输 出 子 进 程 和 线 程 I D 然 后 显 示 出 来 。

4) 窗口应用程序点击鼠标左键，则分别在记事本和应用程序窗口输出鼠标左键的

坐标，并在记事本和窗口应用程序鼠标左键坐标点为圆心，输出半径为100的圆，

并用自创画笔和画刷填充

5) 然后菜单项新建菜单子项“退出记事本”，退出记事本，调用结束进程函数

TerminateProcess结束记事本进程。

实现代码：

#include <windows.h>

#include <iostream>

#include <tlhelp32.h>

BOOL FindFirst(DWORD dwValue);

BOOL FindNext(DWORD dwValue);

DWORD g\_arList[1024];

int g\_nListCnt;

HANDLE g\_hProcess; // 目标进程句柄

BOOL WriteMemory(DWORD dwAddr, DWORD dwValue);

void ShowList();

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

{

TCHAR szFileName[] = TEXT("Notepad");

STARTUPINFO si = { sizeof(si) };

PROCESS\_INFORMATION pi;

::CreateProcess(NULL, szFileName, NULL, NULL, FALSE,CREATE\_NEW\_CONSOLE, NULL, NULL, &si, &pi); // 关闭线程句柄，既然我们不使用它

::CloseHandle(pi.hThread);

g\_hProcess = pi.hProcess;

printf("进程ID：%d，线程ID：%d\n", pi.dwProcessId, pi.dwThreadId);

Sleep(100);

HWND nphWnd = ::FindWindow("Notepad", NULL);

if (nphWnd)

{

int iVal;

printf(" Input val = "); // 输入修改的值

scanf("%d", &iVal);

FindFirst(iVal); // 进行第一次查找

ShowList(); // 打印出搜索的结果

while (g\_nListCnt > 1)

{

printf(" Input val = ");

scanf("%d", &iVal);

FindNext(iVal); // 进行下次搜索

ShowList(); // 显示搜索结果

}

printf(" New value = "); // 取得新值

scanf("%d", &iVal); // 写入新值

if (WriteMemory(g\_arList[0], iVal))

printf(" Write data success \n");

}

else

{

::MessageBox(NULL, TEXT("please open notepad"), TEXT("error"), MB\_OK);

return 0;

}

::CloseHandle(g\_hProcess);

return 0;

}

BOOL CompareAPage(DWORD dwBaseAddr, DWORD dwValue)

{

BYTE arBytes[4096]; // 读取1页内存

if (!::ReadProcessMemory(g\_hProcess, (LPVOID)dwBaseAddr, arBytes, 4096, NULL))

return FALSE; // 此页不可读

DWORD\* pdw; // 在这1页内存中查找

for (int i = 0; i<(int)4 \* 1024 - 3; i++)

{

pdw = (DWORD\*)&arBytes[i];

if (pdw[0] == dwValue) // 等于要查找的值？

{

if (g\_nListCnt >= 1024)

return FALSE;

// 添加到全局变量中

g\_arList[g\_nListCnt++] = dwBaseAddr + i;

}

}

return TRUE;

}

BOOL FindFirst(DWORD dwValue)

{

const DWORD dwOneGB = 1024 \* 1024 \* 1024; // 1GB

const DWORD dwOnePage = 4 \* 1024; // 4KB

if (g\_hProcess == NULL)

return FALSE;

DWORD dwBase; // 查看操作系统类型，以决定开始地址

OSVERSIONINFO vi = { sizeof(vi) };

::GetVersionEx(&vi);

if (vi.dwPlatformId == VER\_PLATFORM\_WIN32\_WINDOWS)

dwBase = 4 \* 1024 \* 1024; // Windows 98系列，4MB

else

dwBase = 640 \* 1024; // Windows NT系列，64KB

for (; dwBase < 2 \* dwOneGB; dwBase += dwOnePage) // 在开始地址到2GB的地址空间进行查找

{

CompareAPage(dwBase, dwValue); // 比较1页大小的内存

}

return TRUE;

}

BOOL FindNext(DWORD dwValue)

{

int nOrgCnt = g\_nListCnt; // 保存m\_arList数组中有效地址的个数，初始化新的m\_nListCnt值

g\_nListCnt = 0; // 在m\_arList数组记录的地址处查找

BOOL bRet = FALSE; // 假设失败

DWORD dwReadValue;

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

{

if (::ReadProcessMemory(g\_hProcess, (LPVOID)g\_arList[i], &dwReadValue, sizeof(DWORD), NULL))

{

if (dwReadValue == dwValue)

{

g\_arList[g\_nListCnt++] = g\_arList[i];

bRet = TRUE;

}

}

}

return bRet;

}

void ShowList() // 打印出搜索到的地址

{

for (int i = 0; i< g\_nListCnt; i++)

{

printf("%08lX \n", g\_arList[i]);

}

}

BOOL WriteMemory(DWORD dwAddr, DWORD dwValue)

{

return ::WriteProcessMemory(g\_hProcess, (LPVOID)dwAddr, &dwValue, sizeof(DWORD), NULL);

}

4、 程序创建三个线程，一个主线程，主线程创建两个附加线程，采用事件或者其他

线程同步机制实现线程同步：

1）第一个附加线程功能为：屏幕输入字符串，写入文件，字符串写入完，通知

第二个附加线程。

2）第二个附加线程功能为：读取文件中字符串中，查找一个单词，若找到了该单词，通知主线程把找到的单词显示出来；

3）若附加线程退出时，还未查找到此单词，则通知主线程，打印出未找到此单

词。

关键代码：

#include <windows.h>

#include <iostream>

#include <process.h>

#include <string>

using namespace std;

string szStr = "This is second experience in windows programming.";

string szUserChar;

HANDLE hThread;

int flag = -1;

DWORD WINAPI ThreadFunc(LPVOID lpParam)

{

int i = (int)lpParam;

cout << "来自主线程 ： " << i << endl;

cout << "\n请出入想要查找的单词:\n";

cin >> szUserChar;

flag = szStr.find(szUserChar);

if (flag != -1)

{

return 0;

}

else

{

flag = -1;

return 0;

}

}

int main(int argc, char argv[])

{

DWORD dwThreadID; //创建附加线程

DWORD dwExitCode;

cout << "主线程的字符串为: " << szStr << endl;

cout << "\n创建附加线程\n";

cout << "Loading......\n\n";

hThread = CreateThread(NULL, 0, ThreadFunc, (LPVOID)11, 0, &dwThreadID);

WaitForSingleObject(hThread, INFINITE); //等待线程结束

cout << "\n返回主线程\n";

if(::GetExitCodeThread(hThread, &dwExitCode)) //判断线程是否结束运行

{

if(dwExitCode == STILL\_ACTIVE)

{

cout << "\n线程还在运行中\n";

}

else

{

if (flag == -1)

{

cout << "附加线程退出，没有找到该单词" << endl;

}

else

{

cout << "附加线程退出，找到单词：" << szUserChar << endl;

}

}

}

cout << endl;

return 0;

}

#include <Windows.h>

#include <process.h>

#include <stdio.h>

#define num 4

int flag = 0;

char FileName[4][6] = { "A.txt","B.txt","C.txt","D.txt" };

HANDLE g\_hThreadParameter1, g\_hThreadParameter2,g\_hThreadParameter3,g\_hThreadParameter4; //信号量与关键段

HANDLE g\_hMain;

CRITICAL\_SECTION g\_csThreadCode;

HANDLE hFileA;

HANDLE hFileB;

HANDLE hFileC;

HANDLE hFileD;

DWORD WINAPI Thread1(LPVOID);

DWORD WINAPI Thread2(LPVOID);

DWORD WINAPI Thread3(LPVOID);

DWORD WINAPI Thread4(LPVOID);

int main(int argc, char \*argv)

{

DWORD dwThreadID;

HANDLE h[4];

hFileA = CreateFile(TEXT("A.txt"), GENERIC\_WRITE, 0, NULL, CREATE\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, NULL); //创建 A B C D　4个文件

hFileB = CreateFile(TEXT("B.txt"), GENERIC\_WRITE, 0, NULL, CREATE\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, NULL);

hFileC = CreateFile(TEXT("C.txt"), GENERIC\_WRITE, 0, NULL, CREATE\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, NULL);

hFileD = CreateFile(TEXT("D.txt"), GENERIC\_WRITE, 0, NULL, CREATE\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, NULL);

CloseHandle(hFileA);

CloseHandle(hFileB);

CloseHandle(hFileC);

CloseHandle(hFileD);

printf("%s\n", FileName[0]);

printf("%s\n", FileName[1]);

printf("%s\n", FileName[2]);

printf("%s\n", FileName[3]);

g\_hMain = CreateSemaphore(NULL, 0, 1, NULL); //初始化信号量和关键段

g\_hThreadParameter1 = CreateSemaphore(NULL, 0, 1, NULL); //当前1个资源，最大允许1个同时访问

g\_hThreadParameter2 = CreateSemaphore(NULL, 0, 1, NULL); //当前0个资源，最大允许1个同时访问

g\_hThreadParameter3 = CreateSemaphore(NULL, 0, 1, NULL); //当前0个资源，最大允许1个同时访问

g\_hThreadParameter4 = CreateSemaphore(NULL, 0, 1, NULL); //当前0个资源，最大允许1个同时访问

InitializeCriticalSection(&g\_csThreadCode);

h[0] = CreateThread(NULL, NULL, Thread1, NULL, 0, &dwThreadID);

h[1] = CreateThread(NULL, NULL, Thread2, NULL, 0, &dwThreadID);

h[2] = CreateThread(NULL, NULL, Thread3, NULL, 0, &dwThreadID);

h[3] = CreateThread(NULL, NULL, Thread4, NULL, 0, &dwThreadID);

ReleaseSemaphore(g\_hMain, 1, NULL); //信号量++

WaitForMultipleObjects(4, h, TRUE, INFINITE); //等待线程结束

DeleteCriticalSection(&g\_csThreadCode); //销毁信号量和关键段

CloseHandle(g\_hThreadParameter1);

CloseHandle(g\_hThreadParameter2);

CloseHandle(g\_hThreadParameter3);

CloseHandle(g\_hThreadParameter4);

CloseHandle(h[0]);

CloseHandle(h[1]);

CloseHandle(h[2]);

CloseHandle(h[3]);

return 0;

}

DWORD WINAPI Thread1(LPVOID)

{

int i, j;

FILE \*stream;

int k = 0;

for (i = 1, j = 0; k < 8 ; k++,j--)

{

if (j < 0) j = 3;

if (flag == 0)

{

WaitForSingleObject(g\_hMain, INFINITE); //等待信号量>0

flag++;

}

else

{

WaitForSingleObject(g\_hThreadParameter1, INFINITE); //等待信号量>0

}

stream = fopen(FileName[j], "at+");

int ac = GetLastError();

fprintf(stream, "%s", "1");

fclose(stream);

ReleaseSemaphore(g\_hThreadParameter2, 1, NULL); //信号量++

}

return 0;

}

DWORD WINAPI Thread2(LPVOID)

{

int i, j;

FILE \*stream;

int k = 0;

for ( i = 2, j = 1; k < 8; k++, j-- )

{

if (j < 0) j = 3;

WaitForSingleObject(g\_hThreadParameter2, INFINITE); //等待信号量>

stream = fopen(FileName[j], "at+");

int ac = GetLastError();

fprintf(stream, "%s", "2");

fclose(stream);

ReleaseSemaphore(g\_hThreadParameter3, 1, NULL); //信号量++

}

return 0;

}

DWORD WINAPI Thread3(LPVOID)

{

int i, j;

FILE \*stream;

int k = 0;

for ( i = 3, j = 2; k < 8; k++, j--)

{

if (j < 0 ) j = 3;

WaitForSingleObject(g\_hThreadParameter3, INFINITE); //等待信号量>0

stream = fopen(FileName[j], "at+");

int ac = GetLastError();

fprintf(stream, "%s", "3");

fclose(stream);

ReleaseSemaphore(g\_hThreadParameter4, 1, NULL); //信号量++

}

return 0;

}

DWORD WINAPI Thread4(LPVOID)

{

int i, j;

FILE \*stream;

int k = 0;

for ( i = 4, j = 3; k < 8; k++, j--)

{

if (j < 0) j = 3;

WaitForSingleObject(g\_hThreadParameter4, INFINITE); //等待信号量>0

stream = fopen(FileName[j], "at+");

int ac = GetLastError();

fprintf(stream, "%s", "4");

fclose(stream);

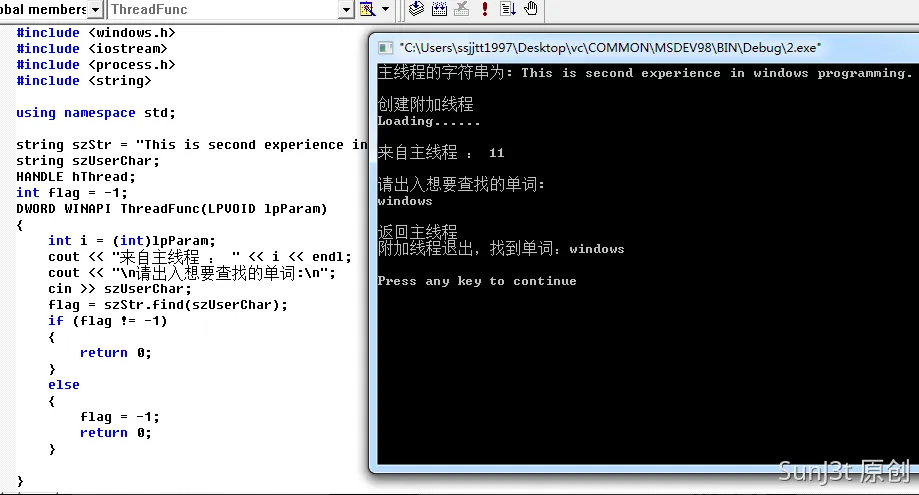
ReleaseSemaphore(g\_hThreadParameter1, 1, NULL); //信号量++

}

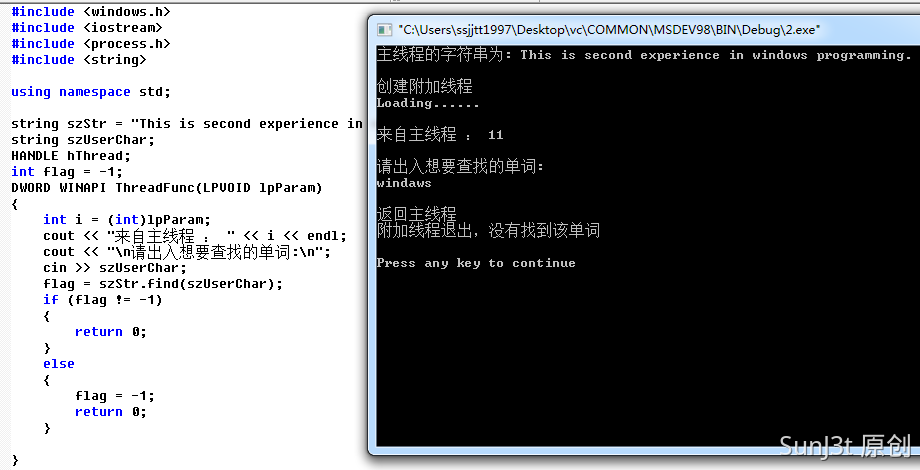
return 0;

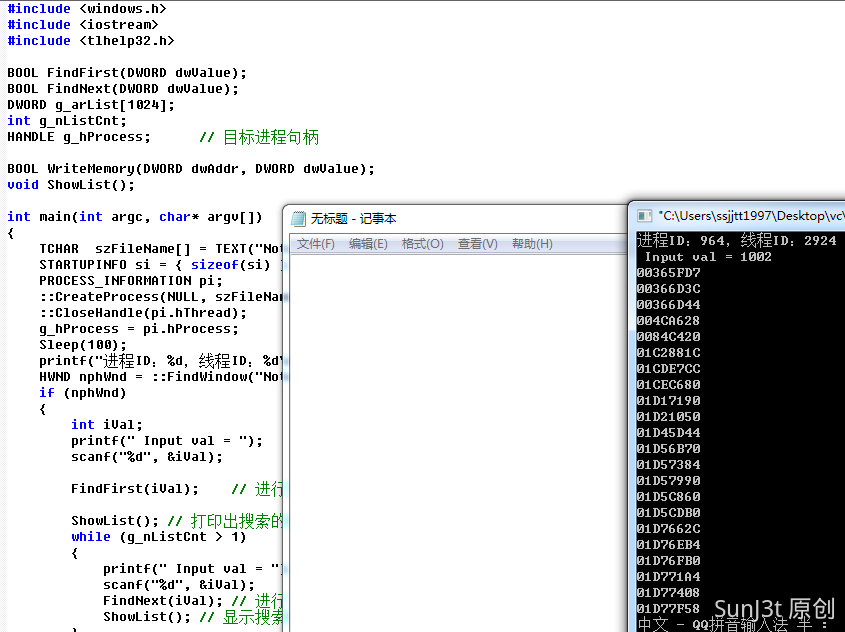
}

1. 实验结果

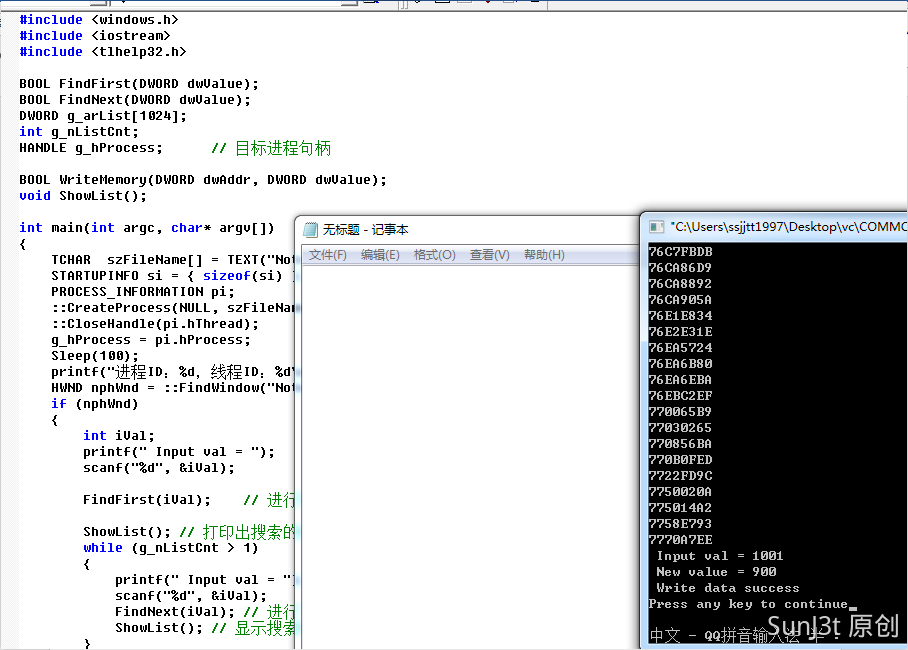


线程退出时仍然没有查到字符





取得新值并写入新值：



五、实验中的问题及解决办法。

很多实验的东西是具体的代码设计，课上讲的很少，解决办法是下来自己查阅资料。。

一开始使用VS2015去编译老师给的相关代码和书上的代码，发现会爆很多错误，后面发现是因为版本太高，好一些函数在新版本并不能使用。所以最后我还是使用的vc作为编译器。

如果不在查找程序之前让程序稍停一会儿，那么FindWindow()函数会直接执行完毕而导致判断nphWnd时会认为程序并没有打开。所以需要在FindWindow()函数前加上sleep()函数。

对windows的API调用时要注意每个函数的参数的作用，以及参数的类型，如果类型不匹配的话也是不能够成功调用的。

2