1. **实验描述及实验目的**

线程是现代操作系统中最重要的概念，通过多线程程序设计，操作系统能够充分利用计算机系统中的各种硬件资源，极大的提高系统效率。本实验主要考查对多线程概念的理论，以及利用C/C++语言编程实现多线程程序，掌握信号量的使用，以实现多线程的同步。

1. **实验报告要求**

(1) 实验报告排版请注意字体、字号、段间距、样式等格式的统一。模板中原有的内容保留“楷体\_GB2312g”格式，自行输入的内容可以采用“宋体”+“Times New Roman”小四号字体。可以在模板的基础上自行拓展大标题、小标题等格式，无强制要求，注意美观易辨识即可。

(2) 请按照实验内容题目提供核心代码、注释、必要的说明与实验结果，其余的设计、分析、思考、总结可以自行拓展。请注意完成实验内容题目的全部要求。请注意截图的清晰程度。报告中的代码，请注意其排版，如有 IDE 的高亮请保留高亮。

1. **实验内容题目**

（1）编写程序，在程序中根据用户输入的可执行程序名称，创建一个进程来运行该可执行程序，并输出该进程的PID。

（2）假设有四个线程，第一个线程输出字符串 “This”，第二个线程输出字符串 “is”, 第三个线程输出字符串“Jinan”, 第四个线程输出字符串 “University！”。编制C/C++程序，在主程序main函数中创建四个线程并依次启动，设计信号量(Semaphore)同步机制，当主程序运行时，屏幕输出的结果是字符串“This is Jinan University!”

（3）本题基于实验题目2。在主函数中依次启动四个线程，修改主程序，使得给定用户任意输入的整数n，程序输出n个同样的字符串“This is Jinan University!”

1. **实验过程与代码实现**
2. 编写程序，在程序中根据用户输入的可执行程序名称，创建一个进程来运行该可执行程序，并输出该进程的PID。

**实验代码：**

#include <windows.h>

#include <stdio.h>

#include <wchar.h>

#include <io.h> //为了使控制台能打印UTF-16宽字符

#include <fcntl.h> //为了使控制台能打印UTF-16宽字符

int main()

{

wchar\_t exeName[256];

\_setmode(\_fileno(stdout), \_O\_U16TEXT); //设置控制台为Unicode模式

wprintf(L"请输入可执行程序的路径（例如： C:\\Windows\\System32\\notepad.exe）：\n");

wscanf\_s(L"%s", exeName, (unsigned int)sizeof(exeName) / sizeof(wchar\_t));

PROCESS\_INFORMATION pi; //进程信息

STARTUPINFO si; //启动信息

// 初始化结构体

ZeroMemory(&si, sizeof(si));

si.cb = sizeof(si);

ZeroMemory(&pi, sizeof(pi));

// 定义创建进程是否成功

BOOL success = CreateProcessW(

exeName, // 可执行程序名称（宽字符）

NULL, // 命令行参数

NULL, // 进程安全属性

NULL, // 线程安全属性

FALSE, // 不继承句柄

0, // 创建标志

NULL, // 使用父进程环境变量

NULL, // 使用父进程当前目录

&si, // 启动信息

&pi // 进程信息

);

if (!success)

{

DWORD errorCode = GetLastError();

if (errorCode == ERROR\_FILE\_NOT\_FOUND)

{

wprintf(L"错误: 找不到指定的文件 - %s\n", exeName);

wprintf(L"请检查路径是否正确，确保文件存在\n");

}

else if (errorCode == ERROR\_PATH\_NOT\_FOUND)

{

wprintf(L"错误: 找不到指定的路径 - %s\n", exeName);

wprintf(L"请检查路径是否正确\n");

}

else

{

wprintf(L"创建进程失败，错误代码: %d\n", errorCode);

}

return 1;

}

// 输出进程ID

wprintf(L"成功创建进程，PID: %d\n", pi.dwProcessId);

// 等待进程结束

WaitForSingleObject(pi.hProcess, INFINITE);

wprintf(L"进程已结束\n");

// 关闭句柄

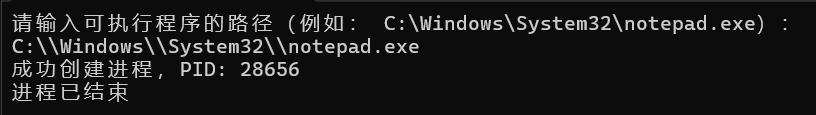
CloseHandle(pi.hProcess);

CloseHandle(pi.hThread);

return 0;

}

**实验结果：**



1. 假设有四个线程，第一个线程输出字符串 “This”，第二个线程输出字符串 “is”, 第三个线程输出字符串“Jinan”, 第四个线程输出字符串 “University！”。编制C/C++程序，在主程序main函数中创建四个线程并依次启动，设计信号量(Semaphore)同步机制，当主程序运行时，屏幕输出的结果是字符串“This is Jinan University!”

**实验代码：**

#include <windows.h>

#include <stdio.h>

// 定义四个信号量的句柄

HANDLE sem1, sem2, sem3, sem4;

// 线程1

DWORD WINAPI Thread1(LPVOID lpParam)

{

// 等待第一个信号量

WaitForSingleObject(sem1, INFINITE); //INFINITE：无限等待直到信号量到来

printf("This");

// 释放第二个信号量，允许线程2执行

ReleaseSemaphore(sem2, 1, NULL);

return 0;

}

// 线程2

DWORD WINAPI Thread2(LPVOID lpParam)

{

WaitForSingleObject(sem2, INFINITE);

printf(" is");

// 释放第三个信号量，允许线程3执行

ReleaseSemaphore(sem3, 1, NULL);

return 0;

}

// 线程3

DWORD WINAPI Thread3(LPVOID lpParam)

{

WaitForSingleObject(sem3, INFINITE);

printf(" Jinan");

// 释放第四个信号量，允许线程4执行

ReleaseSemaphore(sem4, 1, NULL);

return 0;

}

// 线程4

DWORD WINAPI Thread4(LPVOID lpParam)

{

WaitForSingleObject(sem4, INFINITE);

printf(" University!");

return 0;

}

int main()

{

//存储四个线程的句柄和ID

HANDLE threads[4];

DWORD threadIds[4];

// 创建信号量

// sem1初始值为1，允许第一个线程先执行

sem1 = CreateSemaphore(NULL, 1, 1, NULL);

// 其他信号量初始值为0，等待前一个线程释放

sem2 = CreateSemaphore(NULL, 0, 1, NULL);

sem3 = CreateSemaphore(NULL, 0, 1, NULL);

sem4 = CreateSemaphore(NULL, 0, 1, NULL);

// 创建四个线程

threads[0] = CreateThread(NULL, 0, Thread1, NULL, 0, &threadIds[0]);

threads[1] = CreateThread(NULL, 0, Thread2, NULL, 0, &threadIds[1]);

threads[2] = CreateThread(NULL, 0, Thread3, NULL, 0, &threadIds[2]);

threads[3] = CreateThread(NULL, 0, Thread4, NULL, 0, &threadIds[3]);

// 等待所有线程执行完毕

WaitForMultipleObjects(4, threads, TRUE, INFINITE);

// 打印换行

printf("\n");

// 关闭线程和信号量句柄，释放资源

for (int i = 0; i < 4; i++)

{

CloseHandle(threads[i]);

}

CloseHandle(sem1);

CloseHandle(sem2);

CloseHandle(sem3);

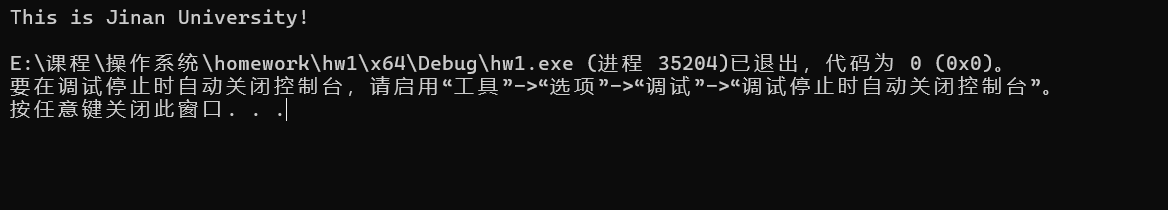
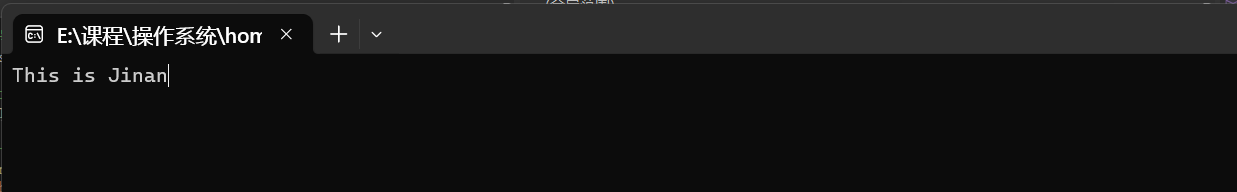
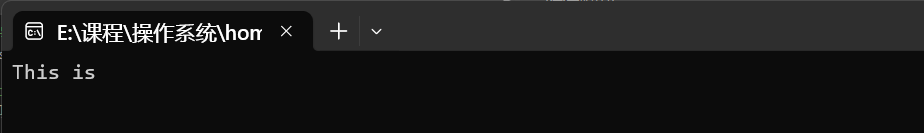
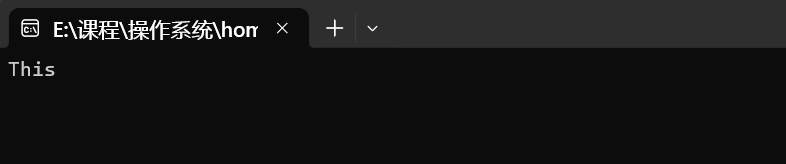
CloseHandle(sem4);

return 0;

}

**实验结果：**

分别在四个线程函数执行设置程序断点进行调试，发现四个线程依次执行：



1. 本题基于实验题目2。在主函数中依次启动四个线程，修改主程序，使得给定用户任意输入的整数n，程序输出n个同样的字符串“This is Jinan University!”

**实验代码：（由于线程定义函数与实验（2）一致，在此只展示main函数）**

int main()

{

int n = 0;

printf("请输入语句重复次数：\n");

scanf\_s("%d", &n);

HANDLE threads[4];

DWORD threadIds[4];

sem1 = CreateSemaphore(NULL, 1, 1, NULL);

sem2 = CreateSemaphore(NULL, 0, 1, NULL);

sem3 = CreateSemaphore(NULL, 0, 1, NULL);

sem4 = CreateSemaphore(NULL, 0, 1, NULL);

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

{

// 每次循环重新创建线程

threads[0] = CreateThread(NULL, 0, Thread1, NULL, 0, &threadIds[0]);

threads[1] = CreateThread(NULL, 0, Thread2, NULL, 0, &threadIds[1]);

threads[2] = CreateThread(NULL, 0, Thread3, NULL, 0, &threadIds[2]);

threads[3] = CreateThread(NULL, 0, Thread4, NULL, 0, &threadIds[3]);

// 释放第一个信号量，启动线程 1

ReleaseSemaphore(sem1, 1, NULL);

// 等待所有线程完成一次输出

WaitForMultipleObjects(4, threads, TRUE, INFINITE);

printf("\n");

// 关闭线程句柄

for (int j = 0; j < 4; j++)

{

CloseHandle(threads[j]);

}

}

CloseHandle(sem1);

CloseHandle(sem2);

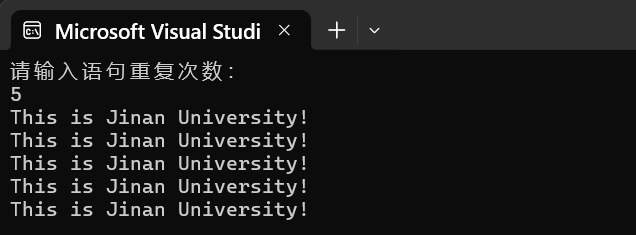
CloseHandle(sem3);

CloseHandle(sem4);

return 0;

}

**实验结果：**



1. **实验总结**

在实验（3），我遇到了一个问题，开始时，我的main函数是：

int main()

{

int n = 0;

printf("请输入语句重复次数：\n");

scanf\_s("%d", &n);

HANDLE threads[4];

DWORD threadIds[4];

sem1 = CreateSemaphore(NULL, 1, 1, NULL);

sem2 = CreateSemaphore(NULL, 0, 1, NULL);

sem3 = CreateSemaphore(NULL, 0, 1, NULL);

sem4 = CreateSemaphore(NULL, 0, 1, NULL);

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

{

// 每次循环重新创建线程

threads[0] = CreateThread(NULL, 0, Thread1, NULL, 0, &threadIds[0]);

threads[1] = CreateThread(NULL, 0, Thread2, NULL, 0, &threadIds[1]);

threads[2] = CreateThread(NULL, 0, Thread3, NULL, 0, &threadIds[2]);

threads[3] = CreateThread(NULL, 0, Thread4, NULL, 0, &threadIds[3]);

// 等待所有线程完成一次输出

WaitForMultipleObjects(4, threads, TRUE, INFINITE);

printf("\n");

}

// 关闭线程句柄

for (int j = 0; j < 4; j++)

{

CloseHandle(threads[j]);

}

CloseHandle(sem1);

CloseHandle(sem2);

CloseHandle(sem3);

CloseHandle(sem4);

return 0;

}

结果如下：



后来了解到：我的for循环中在打印完第一遍语句后，sem1的计数值变为0，启动下一次输出需要重新给出第一个信号量来初始化线程。

于是在for循环中增加代码：

ReleaseSemaphore(sem1, 1, NULL);

通过本次实验，我了解了在C语言环境中一些线程函数的基本使用方法，加深了信号量对线程间同步作用的印象。