**软件逆向工程 上机作业 2：Windows 系统编程实验**

21009201006 夏雨轩

一、程序1：该程序能够创建一个本机的notepad 进程（使用CreateProcess，显式指定

notepad可执行文件的路径）

1. 设计思路：

使用 CreateProcess()执行 notepad.exe

1. 程序代码与含义：

#include <Windows.h>

#include <tchar.h>

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

{

// 定义PROCESS\_INFORMATION结构体，用于接收新进程的信息

PROCESS\_INFORMATION processInfo;

// 定义STARTUPINFO结构体，用于指定新进程的主窗口外观

STARTUPINFO startupInfo;

// 初始化STARTUPINFO结构体

ZeroMemory(&startupInfo, sizeof(startupInfo));

startupInfo.cb = sizeof(startupInfo);

// 创建新进程

if (CreateProcess(

\_T("C:\\Windows\\System32\\notepad.exe"), // Notepad可执行文件的路径

NULL, // 命令行参数

NULL, // 进程的安全描述符

NULL, // 线程的安全描述符

FALSE, // 指定是否继承句柄

0, // 创建标志

NULL, // 使用父进程的环境变量

NULL, // 使用父进程的当前目录

&startupInfo, // STARTUPINFO结构体

&processInfo // PROCESS\_INFORMATION结构体

))

{

// 等待新进程结束

WaitForSingleObject(processInfo.hProcess, INFINITE);

// 关闭进程和线程句柄

CloseHandle(processInfo.hProcess);

CloseHandle(processInfo.hThread);

}

else

{

// 处理创建进程失败的情况

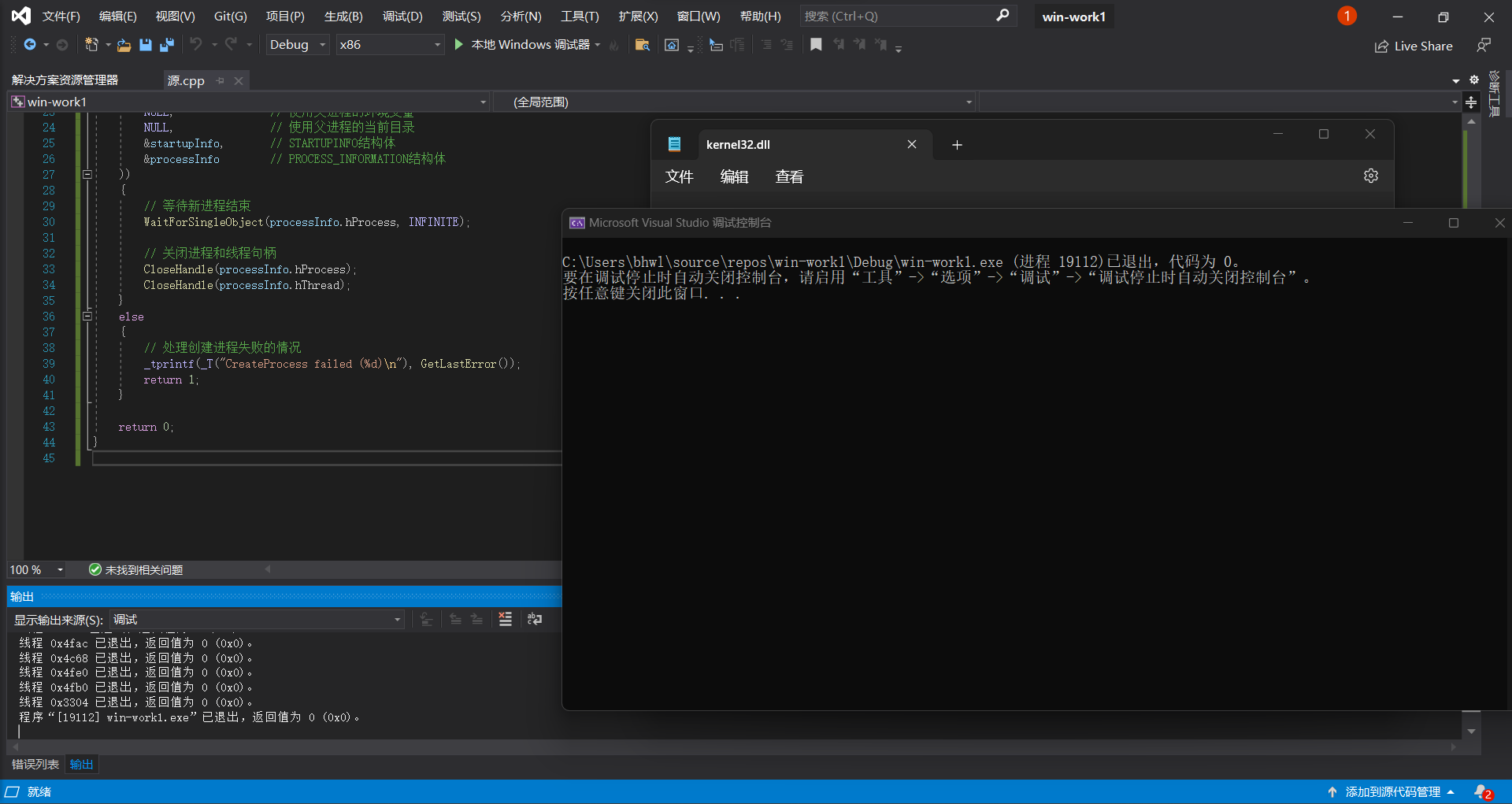
return 1;

}

return 0;

}

1. 程序运行结果：



二、程序 2：

a) 该程序能够创建一个线程，显示 MessageBox

b) 在以上子线程中，编程获得 kernel32.dll 在当前系统中的路径信息，作为

内容显示在以上的 MessageBox 中

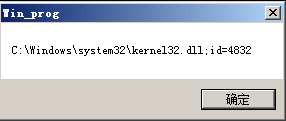
c) 在以上子线程中， 编程获得子线程所加载 kernel32.dll 中的

GetCurrentThreadId()函数的地址，调用该函数，获得子线程的线程编号，

将线程编号连接到上一问的 kernel32.dll 路径后面，再将连接结果字符串

显示在 MessageBox 中

d) 最终显示结果如下图所示：



1. 设计思路：

在主函数中，使用 CreateThread 创建了一个子线程，并通过 WaitForSingleObject 等待子线程执行完毕。最后，通过 CloseHandle 关闭线程句柄。

在线程函数中，首先获取 kernel32.dll 模块的句柄，然后获取模块的路径。接着，通过 GetProcAddress 获取 GetCurrentThreadId 函数的地址，并通过函数指针调用该函数，获取当前线程的ID。最后，构建一个信息字符串，使用 MessageBox 弹出一个消息框，显示线程信息。

1. 程序代码与含义：

#include <windows.h>

#include <iostream>

#include <string>

#include <sstream>

// 函数声明

DWORD WINAPI ThreadFunction(LPVOID lpParam);

int main() {

// 创建子线程

HANDLE hThread = CreateThread(NULL, 0, ThreadFunction, NULL, 0, NULL);

if (hThread == NULL) {

std::cerr << "Error creating thread: " << GetLastError() << std::endl;

return 1;

}

// 等待子线程完成

WaitForSingleObject(hThread, INFINITE);

// 关闭线程句柄

CloseHandle(hThread);

return 0;

}

DWORD WINAPI ThreadFunction(LPVOID lpParam) {

// 获取 kernel32.dll 模块的句柄

HMODULE hModule = GetModuleHandle(L"kernel32.dll");

if (hModule == NULL) {

std::cerr << "Error getting module handle: " << GetLastError() << std::endl;

return 1;

}

// 获取 kernel32.dll 模块的路径

wchar\_t modulePath[MAX\_PATH];

GetModuleFileName(hModule, modulePath, MAX\_PATH);

// 获取 GetCurrentThreadId 函数的地址

FARPROC pGetCurrentThreadId = GetProcAddress(hModule, "GetCurrentThreadId");

if (pGetCurrentThreadId == NULL) {

std::cerr << "Error getting address of GetCurrentThreadId: " << GetLastError() << std::endl;

return 1;

}

// 调用 GetCurrentThreadId 函数获取线程编号

DWORD threadId = reinterpret\_cast<DWORD(\_\_stdcall\*)()>(pGetCurrentThreadId)();

// 构建要显示的信息字符串

wchar\_t buffer[MAX\_PATH + 50];

wsprintf(buffer, L"kernel32.dll 路径: %s\nThread ID: %lu", modulePath, threadId);

// 显示 MessageBox

MessageBox(NULL, buffer, L"Thread Information", MB\_OK);

return 0;

}

3. 程序运行结果：

