一、实验目标

1. 熟悉 Windows 核心的 API 用法
2. 了解 Windows 进程和线程的创建

二、实验内容

使用 Windows 核心 API 实现以下程序：
1. 程序 1：该程序能够创建一个本机的 OllyDbg 进程（使用 CreateProcess，显式指定 ollyDbg 可执行文件的路径）
2. 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。

三、实验过程

1.程序 1：

1.1 初始化函数变量：

```
char szCommandLine[] = "\"D:\\Program Files\\odbg201\\ollydbg.exe\"";
STARTUPINFO si = { sizeof(si) };
PROCESS_INFORMATION pi;
si.dwFlags = STARTF_USESHOWWINDOW; // 指定wShowWindow成员有效
si.wShowWindow = TRUE; // 此成员设为TRUE的话则显示新建进程的主窗口
```

这里边 szCommandLine 变量存储的是可执行文件的绝对路径。

1.2 创建新的进程：

```
BOOL bRet = CreateProcess (
    NULL,   // 不在此指定可执行文件的文件名
    szCommandLine,// 命令行参数
    NULL,   // 默认进程安全性
    NULL,   // 默认进程安全性
    FALSE,  // 指定当前进程内句柄不可以被子进程继承
    CREATE_NEW_CONSOLE, // 为新进程创建一个新的控制台窗口
    NULL,   // 使用本进程的环境变量
    NULL,   // 使用本进程的驱动器和目录
    &si,
    &pi);
```

1.3 打印新创建的进程信息：

```
if(bRet)
{
    // 不使用的句柄关掉
    CloseHandle(pi.hThread);
    CloseHandle(pi.hProcess);
    printf("新进程的ID号：%d\n",pi.dwProcessId);
    printf("新进程的主线程ID号：%d\n",pi.dwThreadId);
}
return 0;
```

1.4 代码清单：

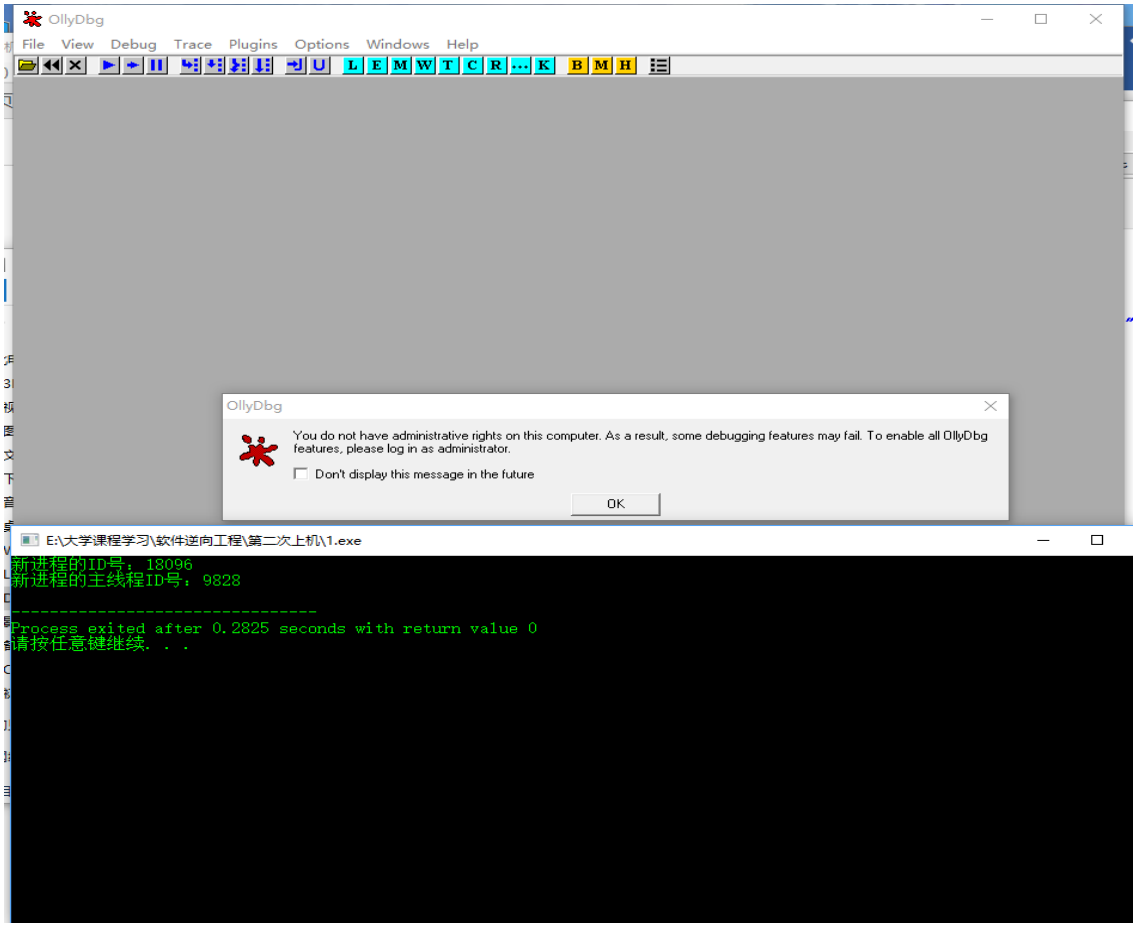
[*] 1.c

```c
#include <stdio.h>
#include <windows.h>

int main(int argc,char *argv[])
{
    char szCommandLine[] = "\"D:\\Program Files\\odbg201\\ollydbg.exe\"";
    STARTUPINFO si = { sizeof(si) };
    PROCESS_INFORMATION pi;
    si.dwFlags = STARTF_USESHOWWINDOW; // 指定wShowWindow成员有效
    si.wShowWindow = TRUE; // 此成员设为TRUE的话则显示新建进程的主窗口

    BOOL bRet = CreateProcess (
        NULL,   // 不在此指定可执行文件的文件名
        szCommandLine,// 命令行参数
        NULL,   // 默认进程安全性
        NULL,   // 默认进程安全性
        FALSE,  // 指定当前进程内句柄不可以被子进程继承
        CREATE_NEW_CONSOLE, // 为新进程创建一个新的控制台窗口
        NULL,   // 使用本进程的环境变量
        NULL,   // 使用本进程的驱动器和目录
        &si,
        &pi);
    if(bRet)
    {
        // 不使用的句柄关掉
        CloseHandle(pi.hThread);
        CloseHandle(pi.hProcess);
        printf("新进程的ID号：%d\n",pi.dwProcessId);
        printf("新进程的主线程ID号：%d\n",pi.dwThreadId);
    }
    return 0;
}
```

1.5 运行结果：



结果我们看出，成功实现了要求的功能。

2. 程序 2：

2.1 编写线程函数，实现获取本地 dll 文件路径：

```
//获取本DLL的文件路径
TCHAR szCurrent[100] = { 0 };
HMODULE hModule = GetModuleHandle("kernel32.dll");

if (hModule)
{
    GetModuleFileName(hModule/*NULL*/, szCurrent, 519);
}
```

GetModuleHandle 功能是获取一个应用程序或动态链接库的模块句柄。只有在当前进程的场景中，这个句柄才会有效。GetModuleFileName 功能是获取当前进程已加载模块的文件的完整路径，该模块必须由当前进程加载。

2.2 获取线程 id：

使用 GetCurrentThreadId()函数获取当前线程号，返回值是整数，然后使用 sprintf 来进行类型转化、拼接。

```
id = GetCurrentThreadId();
sprintf(str,"%d",id);
sprintf(szCurrent,"%s%s%s",szCurrent,";id=",str);
```

2.3 MessageBox 输出：

```
MessageBox(NULL,TEXT(szCurrent),TEXT("Win_prog"),MB_YESNO|MB_ICONQUESTION);
```

2.4 创建一个线程：

使用 CreateThread 函数即可，具体用法课本上有。

```
  //让主线程进入循环，主线程若退出，子线程1，2会被系统“杀死”
  //创建线程1
    CreateThread(
        NULL,                   // default security attributes
        0,                      // use default stack size
        ThreadProc1,            // thread function
        NULL,                   // argument to thread function
        0,                      // use default creation flags
        NULL);                  // returns the thread identifier
    while(1);
```

2.5 完整代码：

[*] 2.cpp

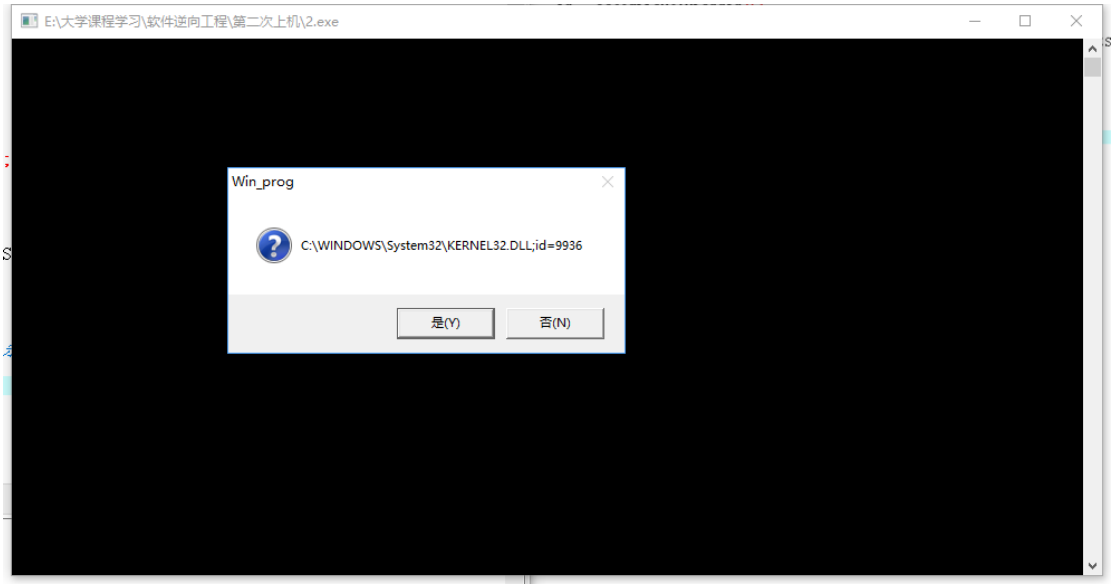
```cpp
#include <windows.h>
#include <stdio.h>
#include<time.h>
#include<string.h>
#include<stdlib.h>

DWORD WINAPI ThreadProc1( LPVOID lpParam )
{
    int len,id;
    TCHAR str[100] = { 0 };
    //获取本DLL的文件路径
    TCHAR szCurrent[100] = { 0 };
    HMODULE hModule = GetModuleHandle("kernel32.dll");

    if (hModule)
    {
        GetModuleFileName(hModule/*NULL*/, szCurrent, 519);
    }
    id = GetCurrentThreadId();
    sprintf(str,"%d",id);
    sprintf(szCurrent,"%s%s%s",szCurrent,";id=",str);
    MessageBox(NULL,TEXT(szCurrent),TEXT("Win_prog"),MB_YESNO|MB_ICONQUESTION);

}
int main()
{
    //让主线程进入循环，主线程若退出，子线程1，2会被系统“杀死”
  //创建线程1
    CreateThread(
        NULL,                   // default security attributes
        0,                      // use default stack size
        ThreadProc1,            // thread function
        NULL,                   // argument to thread function
        0,                      // use default creation flags
        NULL);                  // returns the thread identifier
    while(1);
    return 0;
}
```

2.6 运行结果：



成功实现了要求的功能。

四、实验总结

本次实验通过两个程序的编写，对 Windows 的 API 接口有了很多了解，总体来说还是比较有收获的。中间也遇到了很多问题，比如忘记了转义字符\的问题，路径的反斜杠没有转义导致运行出错，还有不知道调用哪个函数获取 dll 文件的路径信息，最后都经过自己查阅百度谷歌相关信息而解决。