```cpp
// HookTester.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"
#include <windows.h>

int main(int argc, char* argv)
{

    MessageBox(
        NULL,
        (LPCWSTR)L"Sleeping for 3 seconds...",
        (LPCWSTR)L"FAILURE",
        MB_ICONASTERISK | MB_OK
    );

    Sleep(3000);

    MessageBox(
        NULL,
        (LPCWSTR)L"Done sleeping.",
        (LPCWSTR)L"FAILURE",
        MB_ICONASTERISK | MB_OK
    );

    HANDLE hFile;
    char DataBuffer[] = "This is some test data to write to the file.";
    DWORD dwBytesToWrite = (DWORD)strlen(DataBuffer);
    DWORD dwBytesWritten = 0;
    BOOL bErrorFlag = FALSE;

    hFile = CreateFile((LPCWSTR)L"C:\\Users\\John Smith\\Desktop\\Testfile.txt",                // name of the write
                        GENERIC_WRITE,          // open for writing
                        0,                      // do not share
                        NULL,                   // default security
                        CREATE_NEW,             // create new file only
                        FILE_ATTRIBUTE_NORMAL,  // normal file
                        NULL);                  // no attr. template

     WriteFile(hFile,           // open file handle
                DataBuffer,      // start of data to write
                dwBytesToWrite,  // number of bytes to write
                &dwBytesWritten, // number of bytes that were written
                NULL);            // no overlapped structure

    CloseHandle(hFile);

    return 0;
}


```

```cpp
// Lab_07_B.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"
#include <iostream>
#include <windows.h>
#include <tlhelp32.h>
#include <cstdio>

int main(int argc, char* argv[])
{
    if (argc != 3)
    {
        std::cout << "Usage: Lab_07_B Evil.dll <PID>" << std::endl;
    }

    PROCESSENTRY32 entry;
    entry.dwSize = sizeof(PROCESSENTRY32);

    HANDLE snapshot = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, NULL);

    if (Process32First(snapshot, &entry) == TRUE)
    {
        while (Process32Next(snapshot, &entry) == TRUE)
        {
            size_t i = 0;
            if (entry.th32ProcessID == std::atoi(argv[2]))
            {
                HANDLE hProcess = OpenProcess(PROCESS_ALL_ACCESS, FALSE, entry.th32ProcessID);
                HANDLE hThread;
                char*   szLibPath = argv[1];
                void*   pLibRemote;   // The address (in the remote process) where szLibPath will be copied to;
                DWORD   hLibModule;   // Base address of loaded module (==HMODULE);
                HMODULE hKernel32 = GetModuleHandle(L"Kernel32");

                // 1. Allocate memory in the remote process for szLibPath
                // 2. Write szLibPath to the allocated memory
                pLibRemote = VirtualAllocEx(hProcess, NULL, strlen(szLibPath), MEM_COMMIT, PAGE_READWRITE);
                WriteProcessMemory(hProcess, pLibRemote, (void*)szLibPath, strlen(szLibPath), NULL);

                // Load "Evil.dll" into the remote process
                hThread = CreateRemoteThread(hProcess, NULL, 0, (LPTHREAD_START_ROUTINE)GetProcAddress(hKernel32, "LoadLibraryA"), pLibRemote, 0, NULL);
                WaitForSingleObject(hThread, INFINITE);

                // Get handle of the loaded module
```

```
                    GetExitCodeThread(hThread, &hLibModule);

                    // Clean up
                    CloseHandle(hThread);
                    VirtualFreeEx(hProcess, pLibRemote, sizeof(szLibPath),
                      MEM_RELEASE);
                }
            }
        }

        CloseHandle(snapshot);

        return 0;
}
```

```cpp
#pragma comment(lib, "detours.lib")

#include <stdio.h>
#include <windows.h>
#include "detours.h"

static VOID(WINAPI * TrueSleep)(DWORD dwMilliseconds) = Sleep;
static INT(WINAPI * TrueMessageBox)(HWND hWnd, LPCTSTR lpText, LPCTSTR lpCaption, UINT uType) = MessageBox;
static HANDLE(WINAPI * TrueCreateFile)(LPCTSTR lpFileName,
                                    DWORD dwDesiredAccess,
                                    DWORD dwShareMode,
                                    LPSECURITY_ATTRIBUTES lpSecurityAttributes,
                                    DWORD dwCreationDisposition,
                                    DWORD dwFlagsAndAttributes,
                                    HANDLE hTemplateFile) = CreateFile;

VOID WINAPI NoSleep(DWORD dwMilliseconds)
{
    printf("Program attempted to sleep for %d milliseconds.\n", dwMilliseconds);
    return TrueSleep(0);
}

INT WINAPI ChangeMessageBoxTitle(HWND hWnd, LPCTSTR lpText, LPCTSTR lpCaption, UINT uType)
{
    LPCTSTR new_title = L"SUCCESS!";
    wprintf(L"Changing MessageBox title from %s to %s\n", lpCaption, new_title);
    return TrueMessageBox(hWnd, lpText, new_title, uType);
}

HANDLE WINAPI LogFileCreation(LPCTSTR lpFileName,
    DWORD dwDesiredAccess,
    DWORD dwShareMode,
    LPSECURITY_ATTRIBUTES lpSecurityAttributes,
    DWORD dwCreationDisposition,
    DWORD dwFlagsAndAttributes,
    HANDLE hTemplateFile)
{
    wprintf(L"Program attempted to create file %s\n", lpFileName);
    return TrueCreateFile(lpFileName,
        dwDesiredAccess,
        dwShareMode,
        lpSecurityAttributes,
        dwCreationDisposition,
        dwFlagsAndAttributes,
        hTemplateFile);
}

BOOL WINAPI DllMain(HINSTANCE hinst, DWORD dwReason, LPVOID reserved)
{
    LONG error;
    (void)hinst;
    (void)reserved;

    if (DetourIsHelperProcess()) {
        return TRUE;
    }

    if (dwReason == DLL_PROCESS_ATTACH) {
        DetourRestoreAfterWith();

        printf("Starting.\n");
        fflush(stdout);

        DetourTransactionBegin();
        DetourUpdateThread(GetCurrentThread());
        DetourAttach(&(PVOID&)TrueSleep, NoSleep);
        error = DetourTransactionCommit();

        if (error == NO_ERROR) {
            printf("Detoured Sleep().\n");
        }
        else {
            printf("Error detouring Sleep(): %d\n", error);
        }

        DetourTransactionBegin();
        DetourUpdateThread(GetCurrentThread());
        DetourAttach(&(PVOID&)TrueMessageBox, ChangeMessageBoxTitle);
        error = DetourTransactionCommit();

        if (error == NO_ERROR) {
            printf("Detoured MessageBox().\n");
        }
        else {
            printf("Error detouring MessageBox(): %d\n", error);
        }

        DetourTransactionBegin();
        DetourUpdateThread(GetCurrentThread());
        DetourAttach(&(PVOID&)TrueCreateFile, LogFileCreation);
        error = DetourTransactionCommit();

        if (error == NO_ERROR) {
            printf("Detoured CreateFile().\n");
        }
        else {
            printf("Error detouring CreateFile(): %d\n", error);
        }

    }
    else if (dwReason == DLL_PROCESS_DETACH) {
        DetourTransactionBegin();
        DetourUpdateThread(GetCurrentThread());
        DetourDetach(&(PVOID&)TrueSleep, NoSleep);
        error = DetourTransactionCommit();
        printf("Removed Sleep() (result=%d)\n", error);
        fflush(stdout);

        DetourTransactionBegin();
        DetourUpdateThread(GetCurrentThread());
        DetourDetach(&(PVOID&)TrueMessageBox, ChangeMessageBoxTitle);
        error = DetourTransactionCommit();
        printf("Removed MessageBox() (result=%d)\n", error);
        fflush(stdout);

        DetourTransactionBegin();
        DetourUpdateThread(GetCurrentThread());
        DetourDetach(&(PVOID&)TrueCreateFile, LogFileCreation);
        error = DetourTransactionCommit();
        printf("Removed CreateFile() (result=%d)", error);
        fflush(stdout);
    }

    return TRUE;
}

extern "C" __declspec(dllexport) void dummy(void) {
    return;
}

```