Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

отчёт по дисциплине

**«Основы операционных систем»**

Лабораторная работа №5

**«Динамически загружаемые библиотеки»**

Выполнили:

студенты группы 19ВВ2

Вожжов И.В.

Анохин А.А.

Принял:

к.т.н. Егоров В.Ю.

Пенза 2022

**Цель работы**

Изучить связывание процесса с динамически загружаемыми библиотеками на этапе загрузки и на этапе выполнения.

**Задание**

Необходимо разработать программу, состоящую из головной программы и двух динамически загружаемых библиотек. Одна библиотека должна загружаться с использованием динамического связывания, а другая – с использованием функции LoadLibrary. Библиотеки должны выполнять действия по вариантам заданий из лабораторной работы №4.

**Ход работы:**

Для подключения статической библиотеки необходимо предоставить данные для разрешения вызова функций из DLL.

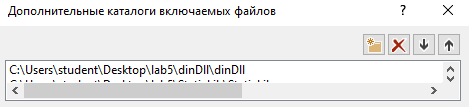


Рисунок 1 –указали в настройках путь до заголовочных файлов DLL

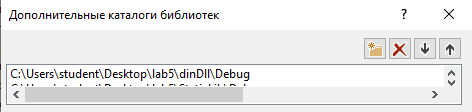


Рисунок 2 – путь до файла с расшерением .lib

**Листинг**

**Динамическая библиотека**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "pch.h"

#include <cstdio>

#include <windows.h>

#include <stdio.h>

#include <string.h>

#include <locale>

#include <Windows.h>

#include <conio.h>

#include <iostream>

#include <time.h>

#include <ctime>

#include "Header.h"

HANDLE hMailSlot;

//char szBuf1[512];

DWORD cbWritten;

BOOL fReturnCode;

DWORD cbMessages;

DWORD cbMsgNumber;

HANDLE hMailslot;

HANDLE ringers[5];

LPCWSTR lpszMailSlotName = L"\\\\.\\mailslot\\mailslot";

char szBuf[512];

DWORD cbRead;

int initialCount = 0;

int lock = 0;

LARGE\_INTEGER ringer(int h, int m);

void startClient();

VOID CALLBACK TimerAPCProc(

LPVOID lpArgToCompletionRoutine,

DWORD dwTimerLowValue,

DWORD dwTimerHighValue

);

extern "C" \_\_declspec(dllexport) void AssyncFileCopyTaskDynamic()

{

hMailslot = CreateMailslot(lpszMailSlotName, 0, MAILSLOT\_WAIT\_FOREVER, NULL);

if (hMailslot == INVALID\_HANDLE\_VALUE)

{

printf("Error MailSlot Creating!\n");

\_getch();

return;

}

printf("This is server! VT ver:2.0.15\n");

startClient();

while (1)

{

fReturnCode = GetMailslotInfo(hMailslot, NULL, &cbMessages, &cbMsgNumber, NULL);

if (!fReturnCode)

{

printf("Get MailSlotInfo Error!\n");

\_getch();

return;

}

if (cbMsgNumber != 0)

{

if (ReadFile(hMailslot, szBuf, 512, &cbRead, NULL))

{

printf("Received %d bytes: <%s>\n", cbRead, szBuf);

#pragma region setTimer

int hour = 0, minute = 0;

LARGE\_INTEGER lnTime;

sscanf\_s(szBuf, "%d %d", &hour, &minute);

lnTime = ringer(hour, minute);

ringers[initialCount] = CreateWaitableTimer(NULL, TRUE, L"SPO\_Timer");

if (!SetWaitableTimer(ringers[initialCount], &lnTime, 0, TimerAPCProc, NULL, FALSE))

{

printf("Error at set timer\n");

return;

}

initialCount++;

#pragma endregion

}

else

printf("Error Data Transfer!\n");

}

if (initialCount != lock) {

while (1)

{

static int nCount = 0;

// Set thread to alertable state

SleepEx(INFINITE, TRUE);

break;

}

lock++;

}

break;

}

CloseHandle(hMailslot);

}

void startClient() {

hMailSlot = CreateFile(lpszMailSlotName, GENERIC\_WRITE, FILE\_SHARE\_READ, NULL, OPEN\_EXISTING, 0, NULL);

if (hMailSlot == INVALID\_HANDLE\_VALUE)

{

printf("CreateFile Error!\n");

\_getch();

return;

}

while (1) {

//scanf\_s("%s %s", &szBuf1);

printf("Enter time to set ringer: ");

gets\_s(szBuf);

//strcpy\_s(szBuf1, "Test message from console");

if (!WriteFile(hMailSlot, szBuf, strlen(szBuf) + 1, &cbWritten, NULL))

printf("Error Data Transfer!\n");

else {

printf("Transferred %d bytes: <%s>\n", cbWritten, szBuf);

break;

}

}

//CloseHandle(hMailSlot);

return;

}

VOID CALLBACK TimerAPCProc(

LPVOID lpArgToCompletionRoutine,

DWORD dwTimerLowValue,

DWORD dwTimerHighValue

)

{

MessageBeep(10);

printf("good morning!");

return;

}

LARGE\_INTEGER ringer(int h, int m) {

struct tm\* u = new tm;

const time\_t timer = time(NULL);

localtime\_s(u, &timer);

const int nTimerUnitsPerSecond = 10000000;

LARGE\_INTEGER ring;

long time;

time = (h - u->tm\_hour) \* 60 \* 60

+ (m - u->tm\_min) \* 60;

ring.QuadPart = -(time \* nTimerUnitsPerSecond);

return ring;

}

#include "pch.h"

#include <cstdio>

#include <windows.h>

#include <stdio.h>

#include <string.h>

#include <locale>

#include <Windows.h>

#include <conio.h>

#include <iostream>

#include <time.h>

#include <ctime>

#include "Header1.h"

using namespace std;

HANDLE hMailSlot;

//char szBuf1[512];

DWORD cbWritten;

//LPCWSTR lpszMailSlotName = L"\\\\.\\mailslot\\mailslot";

/// <summary>

/// //////////////////////////////////////////////////////////

/// </summary>

BOOL fReturnCode;

DWORD cbMessages;

DWORD cbMsgNumber;

HANDLE hMailslot;

HANDLE ringers[5];

LPCWSTR lpszMailSlotName = L"\\\\.\\mailslot\\mailslot";

char szBuf[512];

DWORD cbRead;

int initialCount = 0;

int lock = 0;

LARGE\_INTEGER ringer(int h, int m);

void startClient();

VOID CALLBACK TimerAPCProc(

LPVOID lpArgToCompletionRoutine,

DWORD dwTimerLowValue,

DWORD dwTimerHighValue

);

extern "C" \_\_declspec(dllexport) void AssyncFileCopyTaskStatic()

{

hMailslot = CreateMailslot(lpszMailSlotName, 0, MAILSLOT\_WAIT\_FOREVER, NULL);

if (hMailslot == INVALID\_HANDLE\_VALUE)

{

printf("Error MailSlot Creating!\n");

\_getch();

return ;

}

printf("This is server! VT ver:2.0.15\n");

startClient();

while (1)

{

fReturnCode = GetMailslotInfo(hMailslot, NULL, &cbMessages, &cbMsgNumber, NULL);

if (!fReturnCode)

{

printf("Get MailSlotInfo Error!\n");

\_getch();

return;

}

if (cbMsgNumber != 0)

{

if (ReadFile(hMailslot, szBuf, 512, &cbRead, NULL))

{

printf("Received %d bytes: <%s>\n", cbRead, szBuf);

#pragma region setTimer

int hour = 0, minute = 0;

LARGE\_INTEGER lnTime;

sscanf\_s(szBuf, "%d %d", &hour, &minute);

lnTime = ringer(hour, minute);

ringers[initialCount] = CreateWaitableTimer(NULL, TRUE, L"SPO\_Timer");

if (!SetWaitableTimer(ringers[initialCount], &lnTime, 0, TimerAPCProc, NULL, FALSE))

{

printf("Error at set timer\n");

return;

}

initialCount++;

#pragma endregion

}

else

printf("Error Data Transfer!\n");

}

if (initialCount != lock) {

while (1)

{

static int nCount = 0;

// Set thread to alertable state

SleepEx(INFINITE, TRUE);

break;

}

lock++;

}

break;

}

CloseHandle(hMailslot);

}

void startClient() {

hMailSlot = CreateFile(lpszMailSlotName, GENERIC\_WRITE, FILE\_SHARE\_READ, NULL, OPEN\_EXISTING, 0, NULL);

if (hMailSlot == INVALID\_HANDLE\_VALUE)

{

printf("CreateFile Error!\n");

\_getch();

return ;

}

while (1) {

//scanf\_s("%s %s", &szBuf1);

printf("Enter time to set ringer: ");

gets\_s(szBuf);

//strcpy\_s(szBuf1, "Test message from console");

if (!WriteFile(hMailSlot, szBuf, strlen(szBuf) + 1, &cbWritten, NULL))

printf("Error Data Transfer!\n");

else {

printf("Transferred %d bytes: <%s>\n", cbWritten, szBuf);

break;

}

}

//CloseHandle(hMailSlot);

return ;

}

VOID CALLBACK TimerAPCProc(

LPVOID lpArgToCompletionRoutine,

DWORD dwTimerLowValue,

DWORD dwTimerHighValue

)

{

MessageBeep(10);

printf("good morning!");

return;

}

LARGE\_INTEGER ringer(int h, int m) {

struct tm\* u = new tm;

const time\_t timer = time(NULL);

localtime\_s(u, &timer);

const int nTimerUnitsPerSecond = 10000000;

LARGE\_INTEGER ring;

long time;

time = (h - u->tm\_hour) \* 60 \* 60

+ (m - u->tm\_min) \* 60;

ring.QuadPart = -(time \* nTimerUnitsPerSecond);

return ring;

}

**Lab5.cpp**

#include <iostream>

#include <Windows.h>

#include <conio.h>

#include "Header.h"

#include "Header1.h"

#include "pch.h"

typedef void (\*funclass) ();

int main()

{

printf("Static\n");

AssyncFileCopyTaskStatic();

printf("Static library func is cancel working\n");

system("pause");

HMODULE hDynamicLib = LoadLibrary(L"D:/ПТУ/Операционки/Lab5/dinDll/Debug/dinDll.dll");

if (!hDynamicLib) {

printf("Error while load dynamic lib");

}

funclass AssyncFileCopyTaskDynamic = (funclass) GetProcAddress(hDynamicLib, "AssyncFileCopyTaskDynamic");

if (!AssyncFileCopyTaskDynamic) {

printf("Get proc address err");

} //подгрузка функции в класс обертку

printf("Dynamic\n");

AssyncFileCopyTaskDynamic();

printf("Dynamic lib fun is cancel working\n");

FreeLibrary(hDynamicLib);

//освобождение библиотеки

return 0;

}

**Результат**

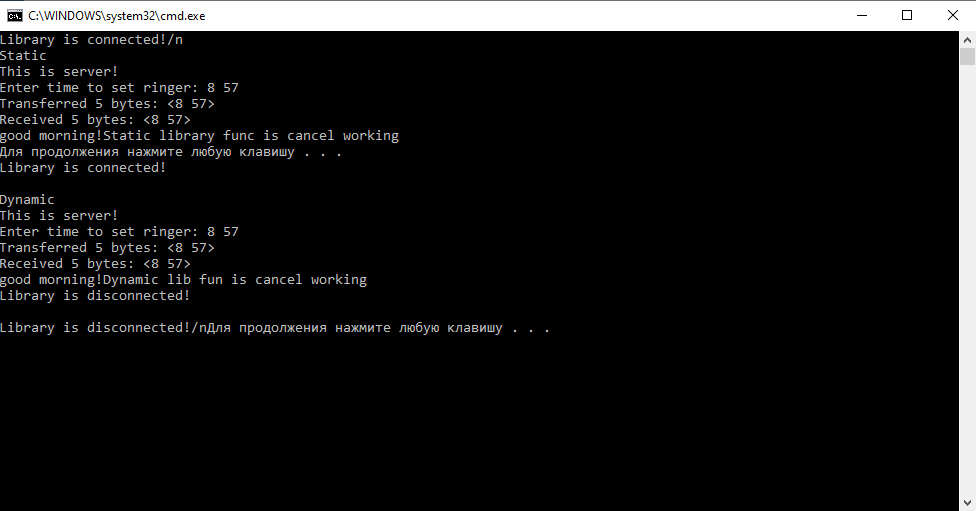


Рисунок 1

**Вывод**

Из данной лабораторной работы мы научились связывать процессы с динамически загружаемыми библиотеками на этапе загрузки и на этапе выполнения.