Министерство науки и высшего образования Российской Федерации  
Пензенский государственный университет  
Кафедра вычислительная техника

**ОТЧЕТ**

по лабораторной работе №5

по дисциплине «Основы операционных систем»

на тему «Динамически загружаемые библиотеки»

Выполнили студенты группы 22ВВП1

Демин М. С.

Беляев Д. И.

Приняли:

Егоров В. Ю.

Федюнин Р. Н.

Пенза 2024

**Название**

Динамически загружаемые библиотеки

**Цель работы**

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

**Лабораторное 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. (Разработать программу периодической проверки свободного места на жестком диске (функция GetDiskFreeSpace) и сбора данной информации на едином сервере с использованием почтового ящика.)

**Описание данных**

1. Библиотеки:

• windows.h: Заголовочный файл, содержащий определения для работы с Windows API.

• serverLib.h: Заголовочный файл, который, предположительно, содержит определения функций для работы с серверной частью программы.

2. Типы данных:

• typedef void (\*StartClientFunc)();: Определение типа для указателя на функцию, которая не принимает аргументов и не возвращает значения. Этот указатель будет использоваться для вызова функции StartClient из загруженной динамической библиотеки.

3. Переменные:

• HMODULE hClientLib: Переменная, которая хранит дескриптор загруженной библиотеки clientLib.dll.

• StartClientFunc startClient: Указатель на функцию StartClient, получаемый через GetProcAddress.

**Описание структуры программы**

1. Функция StartClient:

• Загружает динамическую библиотеку clientLib.dll с помощью функции LoadLibrary.

• Проверяет, успешно ли была загружена библиотека. Если нет, выводит сообщение об ошибке и завершает выполнение.

• Получает адрес функции StartClient из загруженной библиотеки с помощью GetProcAddress.

• Проверяет, удалось ли получить адрес функции. Если нет, выводит сообщение об ошибке, освобождает библиотеку и завершает выполнение.

• Если все прошло успешно, вызывает функцию startClient, а затем освобождает библиотеку с помощью FreeLibrary.

2. Функция main:

• Принимает аргументы командной строки. Ожидается, что будет передан один аргумент.

• Проверяет количество аргументов. Если их не два (имя программы и аргумент), программа завершает выполнение с кодом 1.

• В зависимости от первого символа второго аргумента ('c' или 's'):

• Если 'c', вызывается функция StartClient, чтобы запустить клиентскую часть.

• Если 's', вызывается функция StartServer, чтобы запустить серверную часть (функция StartServer должна быть определена в подключенной библиотеке serverLib.h).

• Завершает выполнение программы с кодом 0.

**Текст программы**

**main.c**

#include <stdio.h>

#include <windows.h>

#include "serverLib.h"

typedef void (\*StartClientFunc)();

void StartClient()

{

HMODULE hClientLib = LoadLibrary(TEXT("clientLib.dll"));

if (!hClientLib) {

printf("Failed to load client library. Error: %d\n", GetLastError());

return 1;

}

StartClientFunc startClient = (StartClientFunc)GetProcAddress(hClientLib, "StartClient");

if (!startClient) {

printf("Failed to get StartClient function address. Error: %d\n", GetLastError());

FreeLibrary(hClientLib);

return 1;

}

startClient();

FreeLibrary(hClientLib);

}

int main(int argc, char\* argv[])

{

if (argc != 2) {

return 1;

}

if (argv[1][0] == 'c') {

StartClient();

} else if (argv[1][0] == 's') {

StartServer();

}

return 0;

}

**clientLib.c**

#include <stdio.h>

#include <windows.h>

#include "clientLib.h"

LPCTSTR MAILBOX\_NAME = TEXT("\\\\.\\mailslot\\sample\_mailslot");

typedef struct {

OVERLAPPED ol;

HANDLE hMailSlot;

WCHAR buffer[256];

} MAILSLOT\_OVERLAPPED;

VOID CALLBACK WriteCompletionRoutine(DWORD dwErrorCode, DWORD dwNumberOfBytesTransfered, LPOVERLAPPED lpOverlapped) {

if (dwErrorCode == ERROR\_SUCCESS) {

wprintf(L"Data written successfully: %d bytes\n", dwNumberOfBytesTransfered);

}

else {

wprintf(L"Failed to write file. Error: %d\n", dwErrorCode);

}

}

VOID CALLBACK TimerAPCProc(LPVOID lpArg, DWORD dwTimerLowValue, DWORD dwTimerHighValue) {

WCHAR drive[] = L"C:\\";

ULARGE\_INTEGER freeBytesAvailable, totalNumberOfBytes, totalNumberOfFreeBytes;

if (GetDiskFreeSpaceEx(drive, &freeBytesAvailable, &totalNumberOfBytes, &totalNumberOfFreeBytes)) {

MAILSLOT\_OVERLAPPED\* mailslot\_ol = (MAILSLOT\_OVERLAPPED\*)lpArg;

WCHAR\* message = mailslot\_ol->buffer;

swprintf(message, sizeof(mailslot\_ol->buffer) / sizeof(WCHAR), L"Free space on %s: %llu bytes\n", drive, freeBytesAvailable.QuadPart);

BOOL result = WriteFileEx(mailslot\_ol->hMailSlot, message, (DWORD)(wcslen(message) \* sizeof(WCHAR)), &mailslot\_ol->ol, WriteCompletionRoutine);

if (!result) {

wprintf(L"WriteFileEx failed. Error: %d\n", GetLastError());

}

}

else {

wprintf(L"Failed to get disk space. Error: %d\n", GetLastError());

}

}

void StartClient() {

wprintf(L"CLIENT\n");

HANDLE hMailSlot = CreateFile(MAILBOX\_NAME, GENERIC\_WRITE, FILE\_SHARE\_READ, NULL, OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, NULL);

if (hMailSlot == INVALID\_HANDLE\_VALUE) {

wprintf(L"Failed to open mailslot. Error: %d\n", GetLastError());

return;

}

MAILSLOT\_OVERLAPPED mailslot\_ol;

SecureZeroMemory(&mailslot\_ol, sizeof(mailslot\_ol));

mailslot\_ol.hMailSlot = hMailSlot;

HANDLE hTimer = CreateWaitableTimer(NULL, FALSE, NULL);

if (!hTimer) {

wprintf(L"CreateWaitableTimer failed. Error: %d\n", GetLastError());

CloseHandle(hMailSlot);

return;

}

LARGE\_INTEGER liDueTime;

liDueTime.QuadPart = -10000000LL;

if (!SetWaitableTimer(hTimer, &liDueTime, CHECK\_INTERVAL, TimerAPCProc, &mailslot\_ol, FALSE)) {

wprintf(L"SetWaitableTimer failed. Error: %d\n", GetLastError());

CloseHandle(hMailSlot);

CloseHandle(hTimer);

return;

}

while (TRUE) {

SleepEx(INFINITE, TRUE);

}

CloseHandle(hMailSlot);

CloseHandle(hTimer);

}

**serverLib.c**

#include <stdio.h>

#include <windows.h>

// #define SERVERLIBRARY\_API \_\_declspec(dllexport)

#include "serverLib.h"

LPCTSTR MAILBOX\_NAME = TEXT("\\\\.\\mailslot\\sample\_mailslot");

typedef struct {

OVERLAPPED ol;

HANDLE hMailSlot;

WCHAR buffer[256];

} MAILSLOT\_OVERLAPPED;

VOID CALLBACK ReadCompletionRoutine(DWORD dwErrorCode, DWORD dwNumberOfBytesTransfered, LPOVERLAPPED lpOverlapped) {

MAILSLOT\_OVERLAPPED\* mailslot\_ol = (MAILSLOT\_OVERLAPPED\*)lpOverlapped;

WCHAR\* buffer = mailslot\_ol->buffer;

buffer[dwNumberOfBytesTransfered / sizeof(WCHAR)] = L'\0';

wprintf(L"Received message: %s", buffer);

}

VOID CALLBACK TimerAPCProc(LPVOID lpArg, DWORD dwTimerLowValue, DWORD dwTimerHighValue) {

MAILSLOT\_OVERLAPPED\* mailslot\_ol = (MAILSLOT\_OVERLAPPED\*)lpArg;

DWORD dwMaxMessageSize, dwNextSize, dwMessageCount, dwReadTimeout;

if (!GetMailslotInfo(mailslot\_ol->hMailSlot, &dwMaxMessageSize, &dwNextSize, &dwMessageCount, &dwReadTimeout)) {

wprintf(L"GetMailslotInfo failed. Error: %d\n", GetLastError());

return;

}

if (dwNextSize != MAILSLOT\_NO\_MESSAGE) {

if (!ReadFileEx(mailslot\_ol->hMailSlot, mailslot\_ol->buffer, sizeof(mailslot\_ol->buffer), &mailslot\_ol->ol, ReadCompletionRoutine)) {

wprintf(L"ReadFileEx failed. Error: %d\n", GetLastError());

}

}

}

void StartServer() {

wprintf(L"SERVER\n");

HANDLE hMailSlot = CreateMailslot(MAILBOX\_NAME, 0, MAILSLOT\_WAIT\_FOREVER, NULL);

if (hMailSlot == INVALID\_HANDLE\_VALUE) {

wprintf(L"Failed to create mailslot. Error: %d\n", GetLastError());

return;

}

MAILSLOT\_OVERLAPPED mailslot\_ol;

SecureZeroMemory(&mailslot\_ol, sizeof(mailslot\_ol));

mailslot\_ol.hMailSlot = hMailSlot;

HANDLE hTimer = CreateWaitableTimer(NULL, FALSE, NULL);

if (!hTimer) {

wprintf(L"CreateWaitableTimer failed. Error: %d\n", GetLastError());

CloseHandle(hMailSlot);

return;

}

LARGE\_INTEGER liDueTime;

liDueTime.QuadPart = -10000000LL;

if (!SetWaitableTimer(hTimer, &liDueTime, CHECK\_INTERVAL, TimerAPCProc, &mailslot\_ol, FALSE)) {

wprintf(L"SetWaitableTimer failed. Error: %d\n", GetLastError());

CloseHandle(hMailSlot);

CloseHandle(hTimer);

return;

}

while (TRUE) {

SleepEx(INFINITE, TRUE);

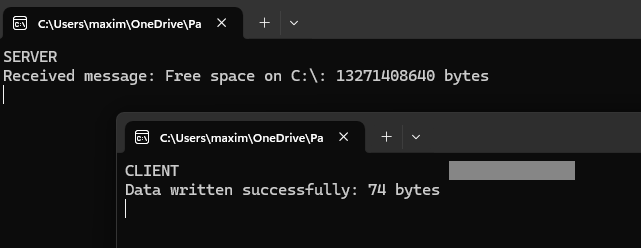
}

CloseHandle(hMailSlot);

CloseHandle(hTimer);

}

**Результат работы программы**



**Вывод**

Изучили как делать связывание процесса с динамически загружаемыми библиотеками на этапе загрузки и на этапе выполнения.