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

**ОТЧЕТ**

по лабораторной работе №4

по дисциплине «Основы операционных систем»

на тему «Передача данных с использованием почтовых ящиков и таймеров ожидания»

Выполнили студенты группы 22ВВП1

Демин М. С.

Беляев Д. И.

Приняли:

Егоров В. Ю.

Федюнин Р. Н.

Пенза 2024

**Название**

Передача данных с использованием почтовых ящиков и таймеров ожидания

**Цель работы**

Изучение построения клиент-серверного взаимодействия процессов с использованием почтовых ящиков. Изучение особенностей работы нитей при использовании таймеров ожидания.

**Лабораторное задание**

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. hMailSlot:

- Тип: HANDLE

- Описание: Дескриптор, который используется для взаимодействия с созданным почтовым слотом (mailslot). Почтовые слоты позволяют обмениваться сообщениями между процессами.

2. MAILSLOT\_OVERLAPPED mailslot\_ol:

- Тип: MAILSLOT\_OVERLAPPED

- Описание: Структура, используемая для асинхронного ввода-вывода в почтовом слоте. В данном коде она инициализируется с помощью функции SecureZeroMemory, чтобы избежать случайных данных в памяти.

3. hTimer:

- Тип: HANDLE

- Описание: Дескриптор таймера, который используется для выполнения периодических операций. Таймер может вызывать функцию обратного вызова (APC) по истечении заданного времени.

4. LARGE\_INTEGER liDueTime:

- Тип: LARGE\_INTEGER

- Описание: Структура, представляющая время ожидания для таймера. Здесь используется для задания времени, через которое таймер должен сработать.

**Описание структуры программы**

1. Создание почтового слота:

- Программа начинает с создания почтового слота с помощью функции CreateMailslot. Если создание не удалось, выводится сообщение об ошибке, и программа завершает выполнение.

2. Инициализация структуры для асинхронного ввода-вывода:

- Структура MAILSLOT\_OVERLAPPED инициализируется нулями для предотвращения использования неинициализированной памяти.

3. Создание таймера:

- Далее программа создает таймер с помощью CreateWaitableTimer. Если создание таймера не удалось, выводится сообщение об ошибке, и программа закрывает дескриптор почтового слота перед завершением.

4. Настройка таймера:

- Программа задает время ожидания для таймера с помощью SetWaitableTimer. В данном случае таймер настроен на срабатывание через 1 секунду (10000000 наносекунд). Если установка таймера не удалась, выводится сообщение об ошибке, и дескрипторы закрываются.

5. Основной цикл:

- Программа входит в бесконечный цикл, где она вызывает SleepEx с параметрами INFINITE и TRUE. Это позволяет программе ожидать событий и обрабатывать их асинхронно.

6. Закрытие дескрипторов:

- После выхода из основного цикла (что в данном коде никогда не происходит), программа должна закрыть дескрипторы почтового слота и таймера для освобождения ресурсов.

**Текст программы**

**CLIENT main.c**

// CLIENT

#include <stdio.h>

#include <windows.h>

#define CHECK\_INTERVAL 10000

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());

}

}

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

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 1;

}

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 1;

}

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 1;

}

while (TRUE) {

SleepEx(INFINITE, TRUE);

}

CloseHandle(hMailSlot);

CloseHandle(hTimer);

return 0;

}

**SERVER main.c**

// SERVER

#include <stdio.h>

#include <windows.h>

#define CHECK\_INTERVAL 500

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());

}

}

}

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

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 1;

}

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 1;

}

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 1;

}

while (TRUE) {

SleepEx(INFINITE, TRUE);

}

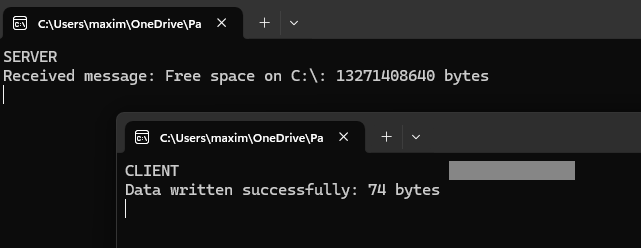
CloseHandle(hMailSlot);

CloseHandle(hTimer);

return 0;

}

**Результат работы программы**



**Вывод**

Изучили построение клиент-серверного взаимодействия процессов с использованием почтовых ящиков. Изучили особенности работы нитей при использовании таймеров ожидания.