Лабораторная работа №6 Асылбек уулу Бакыта из группы ИСП-308

1. Доработать изученный на лекции код таким образом, чтобы он работал одновременно с двумя клиентами. Таким образом Вы дадите возможность клиентам общаться между собой и получите что-то вроде «чатика».

Клиент

#include <stdio.h>

#include <stdlib.h>

#include <WinSock2.h>

#include <Windows.h>

#include <ws2tcpip.h>

#include <wspiapi.h>

#define BUFFER\_SIZE 100

#define PORT "10203"

#define SIGNAL\_BYE "bye"

#pragma comment (lib, "Ws2\_32.lib")

#pragma comment (lib, "Mswsock.lib")

#pragma comment (lib, "AdvApi32.lib")

int result = E\_FAIL;

WSADATA wsa = { 0 };

SOCKET connect\_socket = INVALID\_SOCKET;

ADDRINFOA client = { 0 };

ADDRINFOA\* full\_client = NULL;

BOOL is\_running = TRUE;

int main()

{

result = WSAStartup(0x0202, &wsa);

printf("WSAStartup returned: %d\n", result);

if (result != ERROR\_SUCCESS)

{

goto cleanup;

}

client.ai\_family = AF\_INET;

client.ai\_socktype = SOCK\_STREAM;

client.ai\_protocol = IPPROTO\_TCP;

result = getaddrinfo("localhost", PORT, &client, &full\_client);

printf("Functoin getaddrinfo returned: %d\n", result);

if (result != ERROR\_SUCCESS)

{

goto cleanup;

}

connect\_socket = socket(full\_client->ai\_family, full\_client->ai\_socktype, full\_client->ai\_protocol);

if (connect\_socket == INVALID\_SOCKET)

{

printf("Failed to create socket\n");

result = WSAGetLastError();

goto cleanup;

}

printf("Create socket successfully\n");

result = connect(connect\_socket, full\_client->ai\_addr, full\_client->ai\_addrlen);

if (result == SOCKET\_ERROR)

{

goto cleanup;

}

is\_running = TRUE;

while (is\_running)

{

char buffer[MAX\_PATH] = { 0 };

printf("Enter a message: ");

fgets(buffer, sizeof(buffer), stdin);

result = send(connect\_socket, buffer, strlen(buffer), 0);

memset(buffer, 0, sizeof(buffer));

result = recv(connect\_socket, buffer, BUFFER\_SIZE, 0);

if (result > 0)

{

if (strncmp(buffer, SIGNAL\_BYE, strlen(SIGNAL\_BYE)) == 0)

{

printf("Server closed connection :)\n");

is\_running = FALSE;

result = ERROR\_SUCCESS;

}

}

else if (result == 0)

{

printf("Connection closed!\n");

}

else

{

printf("Failed to recv with code %d\n", WSAGetLastError());

}

}

cleanup:

if (result != ERROR\_SUCCESS)

{

printf("WSAGetLastError returned: %d\n", WSAGetLastError());

}

if (full\_client) freeaddrinfo(full\_client);

if (connect\_socket != INVALID\_SOCKET) closesocket(connect\_socket);

return result;

}

Сервер

#undef UNICODE

#include <stdio.h>

#include <stdlib.h>

#include <WinSock2.h>

#include <Windows.h>

#include <ws2tcpip.h>

#include <wspiapi.h>

#define BUFFER\_SIZE 100

#define PORT "10203"

#define SIGNAL\_BYE "bye"

#pragma comment (lib, "Ws2\_32.lib")

#pragma comment (lib, "Mswsock.lib")

#pragma comment (lib, "AdvApi32.lib")

int m = 0;

int result = E\_FAIL;

WSADATA wsa = { 0 };

SOCKET listen\_socket = INVALID\_SOCKET;

ADDRINFOA server = { 0 };

ADDRINFOA\* full\_server = NULL;

BOOL is\_running = TRUE;

char buffer[BUFFER\_SIZE] = { 0 };

SOCKET client[1] = { INVALID\_SOCKET };

DWORD message(LPVOID t)

{

int h;

if ((int)t == 0)

{

h = 0;

}

else

{

h = 1;

}

printf\_s(" Client %d\n", (int)t);

do

{

result = recv(client[(int)t], buffer, BUFFER\_SIZE, 0);

if (result > 0)

{

printf(" Client %d writes a message: %s\n", (int)t, buffer);

send(client[h], buffer, result, 0);

if (strncmp(buffer, SIGNAL\_BYE, strlen(SIGNAL\_BYE)) == 0)

{

result = 0;

}

}

else if (result == 0)

{

printf("Connection closing...\n");

}

memset(buffer, 0, sizeof(buffer));

} while (result > 0);

return 0;

}

int main()

{

result = WSAStartup(0x0202, &wsa);

printf("WSAStartup returned: %d\n", result);

if (result != ERROR\_SUCCESS)

{

goto cleanup;

}

server.ai\_family = AF\_INET;

server.ai\_socktype = SOCK\_STREAM;

server.ai\_protocol = IPPROTO\_TCP;

result = getaddrinfo("localhost", PORT, &server, &full\_server);

printf("Functoin getaddrinfo returned: %d\n", result);

if (result != ERROR\_SUCCESS)

{

goto cleanup;

}

listen\_socket = socket(full\_server->ai\_family, full\_server->ai\_socktype, full\_server->ai\_protocol);

if (listen\_socket == INVALID\_SOCKET)

{

printf("Failed to create socket\n");

result = WSAGetLastError();

goto cleanup;

}

printf("Create socket successfully\n\n");

result = bind(listen\_socket, full\_server->ai\_addr, full\_server->ai\_addrlen);

if (result == SOCKET\_ERROR)

{

goto cleanup;

}

result = listen(listen\_socket, SOMAXCONN);

if (result == SOCKET\_ERROR)

{

goto cleanup;

}

while (is\_running)

{

client[m] = accept(listen\_socket, NULL, NULL);

if (client[m] == INVALID\_SOCKET)

{

printf("Failed to accept with code %d\n", WSAGetLastError());

continue;

}

CreateThread(NULL, 0, message, (void\*)m, 0, NULL);

m++;

}

result = shutdown(client[0], SD\_BOTH);

if (result == SOCKET\_ERROR)

{

printf("Failed to shutdown with code %d\n", WSAGetLastError());

is\_running = FALSE;

closesocket(client[0]);

}

result = shutdown(client[1], SD\_BOTH);

if (result == SOCKET\_ERROR)

{

printf("Failed to shutdown with code %d\n", WSAGetLastError());

is\_running = FALSE;

closesocket(client[1]);

}

cleanup:

if (result != ERROR\_SUCCESS)

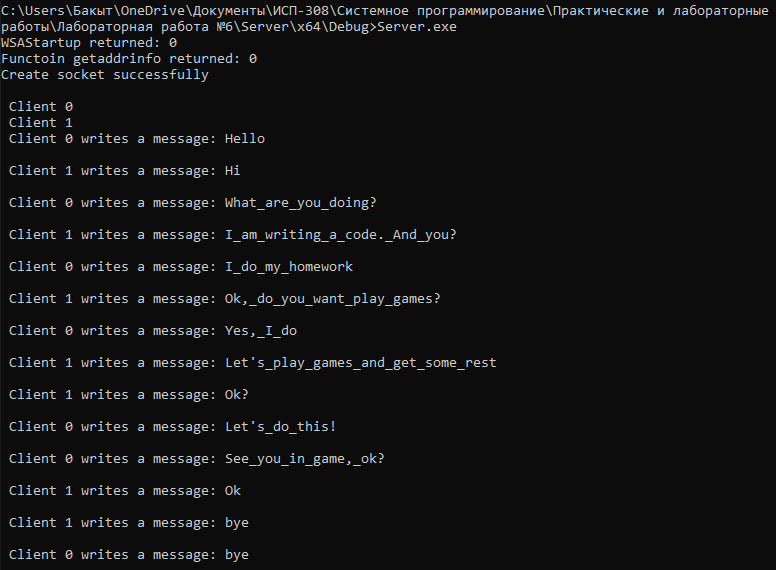
{

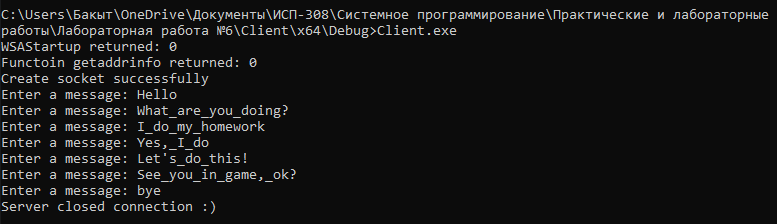
printf("WSAGetLastError returned: %d\n", WSAGetLastError());

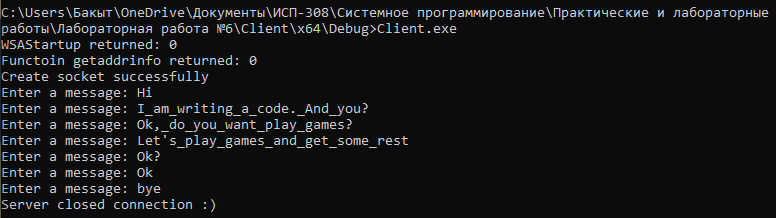
}

return result;

}







1. Реализовать аналогичный «чатик» через pipe

Клиент

#define \_NO\_CRT\_STDIO

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <windows.h>

#include <stdio.h>

#include <string.h>

#include <WinBase.h>

#define BUFSIZE 512

char name[10] = {0};

HANDLE Mutex;

DWORD WINAPI ReadThread(LPVOID lpPipe)

{

WaitForSingleObject(Mutex, INFINITE);

char buf[BUFSIZE] = { 0 };

DWORD Read = NULL;

BOOL Success;

HANDLE Pipe = (HANDLE)lpPipe;

// Подготовка буфера к чтению

memset(buf, 0, sizeof(buf));

Read = 0;

// Считывание

Success = ReadFile(Pipe, // Дескриптор pipe'а

buf, // Буфер для получения ответа

BUFSIZE \* sizeof(char), // Размер буфера

&Read, // Количество считываемых байтов

NULL); // Не перекрывающийся

if (Read > 0)

{

// Вывод сообщения

printf(" %s\n", buf);

}

// Освободить мьютекс

ReleaseMutex(Mutex);

return 0;

}

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

{

HANDLE Pipe, Thread = 0;

BOOL Success = FALSE;

DWORD ToWrite, Written, Mode, Idt;

char Message[BUFSIZE] = { 0 };

LPCSTR Pipename = "\\\\.\\pipe\\mynamedpipe";

//Ввод имени клиента

printf("Enter your name: ");

scanf\_s("%s", name, 10);

name[strlen(name)] = 0;

// Попробуйте открыть именованный канал, дождитесь его, если это необходимо

while (1) {

Pipe = CreateFileA(Pipename, // Название pipe'a

GENERIC\_READ | // Чтение и запись доступны

GENERIC\_WRITE,

0, // Запрещает другим процессам открывать файл или устройство, если они запрашивают доступ на удаление, чтение или запись

NULL, // Дескриптор не может быть унаследован

OPEN\_EXISTING, // Открывает файл или устройство, только если оно существует

0, // Атрибуты по умолчанию

NULL); // Нет шаблона файла

// Остановить, если дескриптор трубы действителен

if (Pipe != INVALID\_HANDLE\_VALUE)

break;

// Завершите работу, если возникает ошибка, отличная от ERROR\_PIPE\_BUSY

if (GetLastError() != ERROR\_PIPE\_BUSY) {

printf("Could not open pipe.\n");

return -1;

}

// Все экземпляры канала заняты, поэтому подождите 20 секунд

if (!WaitNamedPipeA(Pipename, 20000)) {

printf("The pipe could not be opened. Wait 20 seconds.");

return -1;

}

}

// Труба подсоединилась. Переключитесь в режим чтения сообщений

Mode = PIPE\_READMODE\_MESSAGE;

Success = SetNamedPipeHandleState(Pipe, // handle pipe'a

&Mode, // Новый режим pipe'a

NULL, // Счетчик коллекций не задается

NULL); // Счетчик коллекций не задается

if (!Success)

{

printf("SetNamedPipeHandleState failed.\n");

return -1;

}

// Создать мьютекс для блокировки ресурсов

Mutex = CreateMutexW(0, FALSE, 0);

// Отправка сообщения на сервер канала

while (1)

{

Thread = CreateThread(0, 0, ReadThread, (LPVOID)Pipe, 0, &Idt);

if (Thread == NULL)

return -1;

WaitForSingleObject(Mutex, INFINITE);

// Подготовка записи сообщения

memset(Message, 0, sizeof(Message));

printf("\nEnter the message to server: ");

scanf\_s("%s", Message, BUFSIZE);

if (strncmp(Message, "bye", strlen(Message)) == 0)

break;

char temp\_buffer[BUFSIZE] = { 0 };

strcpy\_s(temp\_buffer, BUFSIZE, name);

strcat\_s(temp\_buffer, BUFSIZE, ": ");

strcat\_s(temp\_buffer, BUFSIZE, Message);

memset(Message, 0, sizeof(Message));

strcpy\_s(Message, BUFSIZE, temp\_buffer);

ToWrite = ((strlen(Message) + 1) \* sizeof(char)) - ((strlen(name) + 3) \* sizeof(char));//Размер байтов отправляемого сообщения

printf("Sending %d byte message.\n", ToWrite);

// Отправка...

Success = WriteFile(Pipe, // handle pipe'a

Message, // Сообщение

BUFSIZE \* sizeof(char), // Количество байтов

&Written, // Количество байтов

NULL); // Не перекрывающийся

//Проверка

if (!Success)

{

printf("WriteFile to pipe failed.\n");

return -1;

}

ReleaseMutex(Mutex);

}

printf("\nGoodbye!:)");

if (Thread != NULL)

CloseHandle(Thread);

CloseHandle(Pipe);

return 0;

}

Сервер

#include <windows.h>

#include <stdio.h>

#include <tchar.h>

#include <strsafe.h>

#define CONNECTING\_STATE 0

#define READING\_STATE 1

#define WRITING\_STATE 2

#define INSTANCES 4

#define PIPE\_TIMEOUT 5000

#define BUFSIZE 512

typedef struct

{

OVERLAPPED Overlap;

HANDLE PI;

TCHAR Request[BUFSIZE];

DWORD Read;

TCHAR Reply[BUFSIZE];

DWORD ToWrite;

DWORD State;

BOOL PendingIO;

} PIPEINST, \* LPPIPEINST;

VOID DisconnectAndReconnect(DWORD);

BOOL ConnectToNewClient(HANDLE, LPOVERLAPPED);

VOID GetAnswerToRequest(LPPIPEINST);

PIPEINST Pipe[INSTANCES];

HANDLE Events[INSTANCES];

int \_tmain(VOID)

{

DWORD i, wait, Ret;

BOOL Success;

LPTSTR Pipename = NULL;

Pipename = (LPTSTR)TEXT("\\\\.\\pipe\\mynamedpipe");

// Начальный цикл создает несколько экземпляров именованного канала вместе с объектом события для каждого экземпляра.

// Для каждого экземпляра запускается перекрывающаяся операция ConnectNamedPipe.

for (i = 0; i < INSTANCES; i++)

{

// Создать объект события для каждого экземпляра

Events[i] = CreateEvent(NULL, // Дескриптор безопасности по умолчанию

TRUE, // Объект события сброса вручную

TRUE, // Сигнализируется начальное состояние объекта события

NULL); // Объект события создается без имени

if (Events[i] == NULL)

{

printf("CreateEvent failed.\n");

CloseHandle(Events[i]);

return 0;

}

Pipe[i].Overlap.hEvent = Events[i];

Pipe[i].PI = CreateNamedPipe(Pipename, // Имя канала

PIPE\_ACCESS\_DUPLEX | // Труба двунаправленная. Как серверные, так и клиентские процессы могут считывать данные и записывать в канал

FILE\_FLAG\_OVERLAPPED, // Режим перекрытия

PIPE\_TYPE\_MESSAGE | // Данные записываются в канал в виде потока сообщений

PIPE\_READMODE\_MESSAGE | // Данные считываются из канала в виде потока сообщений

PIPE\_WAIT, // Режим блокировки

INSTANCES, // Количество экземпляров

BUFSIZE \* sizeof(TCHAR), // Размер выводимого буфера

BUFSIZE \* sizeof(TCHAR), // Размер вводимого буфера

PIPE\_TIMEOUT, // Перерыв клиента

NULL); // Дескриптор безопасности по умолчанию

if (Pipe[i].PI == INVALID\_HANDLE\_VALUE)

{

printf("CreateNamedPipe failed.\n");

return 0;

}

// Вызов функции для соединения нового клиента

Pipe[i].PendingIO = ConnectToNewClient(Pipe[i].PI, &Pipe[i].Overlap);

Pipe[i].State = Pipe[i].PendingIO ? CONNECTING\_STATE : READING\_STATE;

}

while (1)

{

// Ждать сигнала об объекте события, завершения перекрывающейся операции чтения, записи или подключения.

wait = WaitForMultipleObjects(INSTANCES, // Количество объектов события

Events, // Массив объектов события

FALSE, // Ждёт хотя-бы одного

INFINITE); // Ждёт бесконечно

// wait показывает, какая труба завершила операцию.

i = wait - WAIT\_OBJECT\_0; // определяет какой канал

if (i < 0 || i >(INSTANCES - 1))

{

printf("Index out of range.\n");

return 0;

}

// Получить результат, если операция была отложена

if (Pipe[i].PendingIO)

{

Success = GetOverlappedResult(Pipe[i].PI, // Дескриптор pipe'a

&Pipe[i].Overlap, // Структура OVERLAPPED

&Ret, // Количество байтов, которые были переданы операцией чтения или записи

FALSE); // Не ждёт завершения функции

switch (Pipe[i].State)

{

// Ожидающая операция подключения

case CONNECTING\_STATE:

if (!Success)

{

printf("Error.\n");

return 0;

}

Pipe[i].State = READING\_STATE;

break;

// Ожидающая операция чтения

case READING\_STATE:

if (!Success || Ret == 0)

{

DisconnectAndReconnect(i);

continue;

}

Pipe[i].Read = Ret;

Pipe[i].State = WRITING\_STATE;

break;

// Ожидающая операция записи

case WRITING\_STATE:

if (!Success || Ret != Pipe[i].ToWrite)

{

DisconnectAndReconnect(i);

continue;

}

Pipe[i].State = READING\_STATE;

break;

default:

{

printf("Invalid pipe state.\n");

return 0;

}

}

}

// Состояние канала определяет, какую операцию выполнить следующей

switch (Pipe[i].State)

{

// Экземпляр канала подключен к клиенту и готов прочитать запрос от клиента

case READING\_STATE:

{

Success = ReadFile(Pipe[i].PI,

Pipe[i].Request,

BUFSIZE \* sizeof(TCHAR),

&Pipe[i].Read,

&Pipe[i].Overlap);

printf(" %s\n", Pipe[i].Request);

}

// Операция чтения успешно завершена

if (Success && Pipe[i].Read != 0)

{

Pipe[i].PendingIO = FALSE;

Pipe[i].State = WRITING\_STATE;

continue;

}

// Операция чтения все еще находится на рассмотрении.

if (!Success && (GetLastError() == ERROR\_IO\_PENDING))

{

Pipe[i].PendingIO = TRUE;

continue;

}

// Произошла ошибка. Отключитесь от клиента

DisconnectAndReconnect(i);

break;

// Запрос был успешно прочитан от клиента. Получите данные ответа и запишите их клиенту

case WRITING\_STATE:

GetAnswerToRequest(&Pipe[i]);

Success = WriteFile(Pipe[i == 0 ? 1 : 0].PI,

Pipe[i].Reply,

Pipe[i].ToWrite,

&Ret,

&Pipe[i].Overlap);

// Операция записи успешно завершена

if (Success && Ret == Pipe[i].ToWrite)

{

Pipe[i == 0 ? 1 : 0].PendingIO = FALSE;

Pipe[i == 0 ? 1 : 0].State = READING\_STATE;

continue;

}

// Операция записи все еще находится на рассмотрении

if (!Success && (GetLastError() == ERROR\_IO\_PENDING))

{

Pipe[i].PendingIO = TRUE;

continue;

}

// Произошла ошибка. Отключитесь от клиента

DisconnectAndReconnect(i);

break;

default:

{

printf("Invalid pipe state.\n");

return 0;

}

}

}

return 0;

}

// DisconnectAndReconnect(DWORD). Эта функция вызывается при возникновении ошибки или когда клиент закрывает свой дескриптор канала

// Отключитесь от этого клиента, затем вызовите ConnectNamedPipe, чтобы дождаться подключения другого клиента

VOID DisconnectAndReconnect(DWORD i)

{

// Отсоединение экземпляра трубы

if (!DisconnectNamedPipe(Pipe[i].PI))

{

printf("DisconnectNamedPipe failed.\n");

}

// Вызовите функцию для подключения нового клиента

Pipe[i].PendingIO = ConnectToNewClient(Pipe[i].PI, &Pipe[i].Overlap);

Pipe[i].State = Pipe[i].PendingIO ? CONNECTING\_STATE : READING\_STATE;

}

// ConnectToNewClient(HANDLE, LPOVERLAPPED) Эта функция вызывается для запуска операции перекрытого подключения.

// Он возвращает значение TRUE, если операция находится в ожидании, или значение FALSE, если соединение было завершено.

BOOL ConnectToNewClient(HANDLE hPipe, LPOVERLAPPED lpo)

{

BOOL Connected, fPendingIO = FALSE;

// Запустите перекрытое соединение для этого экземпляра канала

Connected = ConnectNamedPipe(hPipe, lpo);

// Перекрытый ConnectNamedPipe должен возвращать ноль

if (Connected)

{

printf("ConnectNamedPipe failed.\n");

return 0;

}

switch (GetLastError())

{

// Выполняется перекрытое соединение

case ERROR\_IO\_PENDING:

fPendingIO = TRUE;

break;

// Клиент уже подключен, поэтому сигнализируйте о событии

case ERROR\_PIPE\_CONNECTED:

if (SetEvent(lpo->hEvent))

break;

// Если во время операции подключения возникает ошибка...

default:

{

printf("ConnectNamedPipe failed.\n");

return 0;

}

}

return fPendingIO;

}

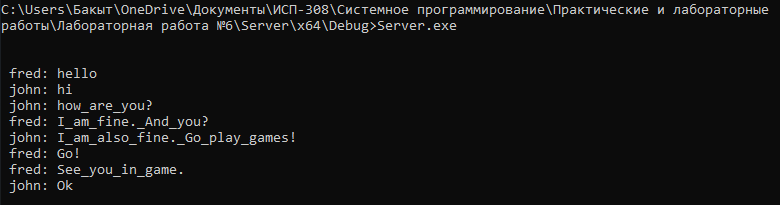
VOID GetAnswerToRequest(LPPIPEINST pipe)

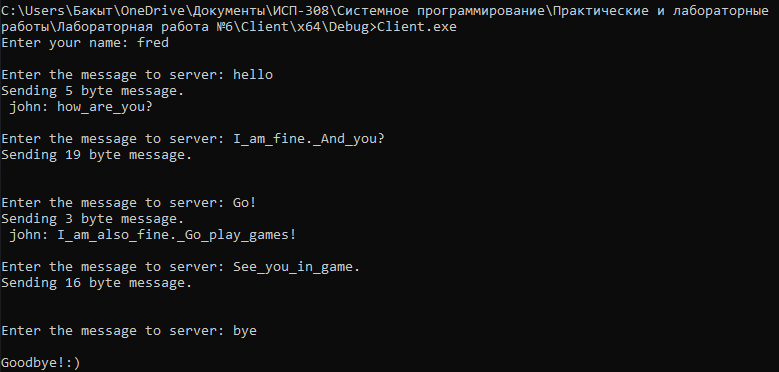
{

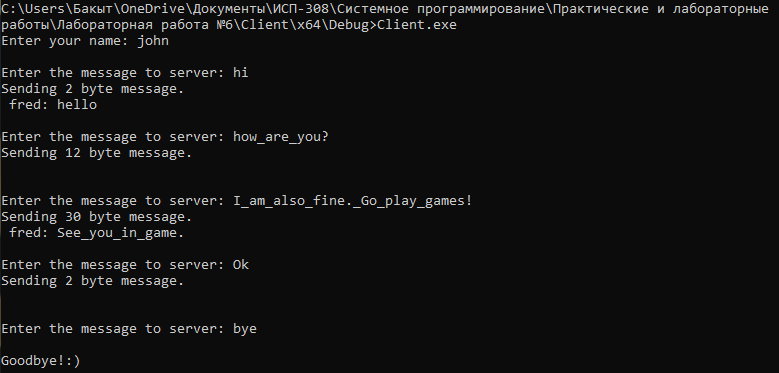
StringCchCopy(pipe->Reply, BUFSIZE, pipe->Request);

pipe->ToWrite = (lstrlen(pipe->Reply) + 1) \* sizeof(TCHAR);

}







1. Продемонстрировать работу с mailslot’ами. Тут «чатик» не нужен. Но взаимодействие клиента и сервера должно быть показано.

Клиент

#include <windows.h>

#include <stdio.h>

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

BOOL WriteSlot(HANDLE slot, LPCTSTR lpszMessage)

{

BOOL result;

DWORD write;

// Запись

result = WriteFile(slot,

lpszMessage,

(DWORD)(lstrlen(lpszMessage) + 1) \* sizeof(TCHAR),

&write,

(LPOVERLAPPED)NULL);

// Проверка

if (!result)

{

printf("WriteFile failed.\n");

return 0;

}

printf("\n Slot was recorded successfully.");

return 1;

}

int main()

{

HANDLE f;

f = CreateFile(SlotName,

GENERIC\_WRITE,

FILE\_SHARE\_READ,

(LPSECURITY\_ATTRIBUTES)NULL,

OPEN\_EXISTING,

FILE\_ATTRIBUTE\_NORMAL,

(HANDLE)NULL);

// Проверка

if (f == INVALID\_HANDLE\_VALUE)

{

printf("CreateFile failed.\n");

return 0;

}

// Вывод сообщений

WriteSlot(f, TEXT("Hi everyone."));

WriteSlot(f, TEXT("Sorry, I have to go! Good luck!"));

CloseHandle(f);

return 1;

}  
Сервер

#include <windows.h>

#include <tchar.h>

#include <stdio.h>

#include <strsafe.h>

HANDLE slot;

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

BOOL ReadSlot()

{

DWORD sotnm, read;

BOOL result;

LPTSTR buffer;

TCHAR achID[80];

HANDLE event;

OVERLAPPED ol;

sotnm = read = 0;

event = CreateEventW(NULL, FALSE, FALSE, TEXT("ExampleSlot"));

// Проверка

if (NULL == event)

return 0;

ol.Offset = 0;

ol.OffsetHigh = 0;

ol.hEvent = event;

// Извлечение информации о mailslot'е.

result = GetMailslotInfo(slot, // Дескриптор mailslot'a

(LPDWORD)NULL, // Размер сообщения в байтах

&sotnm, // Размер следующего сообщения

NULL, // Количество сообщений

(LPDWORD)NULL); // Не ожидает сообщения

// Проверка

if (!result)

{

printf("GetMailslotInfo failed.\n");

return 0;

}

while (1) // Для извлечения всех сообщений

{

// Кем было написано это сообщение и сколько оно весит

StringCchPrintf((LPTSTR)achID,

80,

TEXT("\n Message was sent by Client.\n Sending %d byte message."), sotnm/2);

// Выделение памяти для сообщения

buffer = (LPTSTR)GlobalAlloc(GPTR, lstrlen((LPTSTR)achID) \* sizeof(TCHAR) + sotnm);

if (buffer == NULL)

return 0;

buffer[0] = '\0';

// Чтение

result = ReadFile(slot,

buffer,

sotnm,

&read,

&ol);

// Проверка

if (!result)

{

printf("ReadFile failed.\n");

GlobalFree((HGLOBAL)buffer);

return 0;

}

// Объединение сообщение и строку номера сообщения

StringCbCat(buffer, lstrlen((LPTSTR)achID) \* sizeof(TCHAR) + sotnm, (LPTSTR)achID);

// Представление сообщения.

\_tprintf(TEXT(" Contents of the mailslot: %s\n"), buffer);

//Освобождает память и делает дескриптор недействительным

GlobalFree((HGLOBAL)buffer);

// Извлечение информации об указанном mailslot'е.

result = GetMailslotInfo(slot, // Дескриптор mailslot'a

(LPDWORD)NULL, // Размер сообщения в байтах

&sotnm, // Размер следующего сообщения

NULL, // Количество сообщений

(LPDWORD)NULL); // Не ожидает сообщения

// Проверка

if (!result)

{

printf("GetMailslotInfo failed.\n");

return 0;

}

}

// Закрываем дескриптор

CloseHandle(event);

return 1;

}

BOOL WINAPI MakeSlot(LPCTSTR lpszSlotName)

{

slot = CreateMailslot(lpszSlotName,

0, // Сообщение может иметь любой размер

MAILSLOT\_WAIT\_FOREVER, // Всегда ожидает сообщения

(LPSECURITY\_ATTRIBUTES)NULL); // безопасность по умолчанию

// Проверка

if (slot == INVALID\_HANDLE\_VALUE)

{

printf("CreateMailslot failed.\n");

return 0;

}

return 1;

}

int main()

{

// Создание mailslot'a

MakeSlot(SlotName);

while (1)

{

Sleep(5 \* 1000);

// Чтение из mailslot'a

ReadSlot();

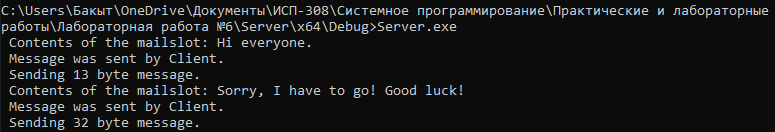
}

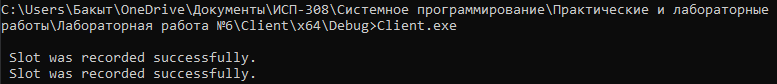
// Закрываем дескриптор

CloseHandle(slot);

return 1;

}





1. Доработать чатик с pipe или socket (на Ваш выбор) таким образом, чтобы была возможность отправки файлов друг другу.

Архитектура взаимодействия Клиент-Сервера

Начало

Установить соединения

listen\_socket = socket(full\_server->ai\_family, full\_server->ai\_socktype, full\_server->ai\_protocol);

Установить соединения

connect\_socket = socket(full\_client->ai\_family, full\_client->ai\_socktype, full\_client->ai\_protocol);

Сервер

Клиент

Получить сообщение

result = recv(client[(int)t], buffer, BUFFER\_SIZE, 0);

Отправить сообщение

result = send(connect\_socket, buffer, strlen(buffer), 0);

Отправить сообщение

send(client[h], buffer, result, 0);

Получить сообщение

result = recv(connect\_socket, buffer, BUFFER\_SIZE, 0);

Закрыть сокет

closesocket(client[0]);

closesocket(client[1]);

Закрыть сокет

closesocket(connect\_socket);

Конец