Лабораторная работа №4

студента группы ИТ – 32

Курбатовой Софьи Андреевны

Выполнение:_____ Защита _____

РАЗРАБОТКА СЕТЕВЫХ ПРИЛОЖЕНИЙ

Цель работы : получение навыков в проектировании прикладных протоколов для передачи данных и реализации этих протоколов в приложениях Windows на языке C/C++ с применением Windows Sockets.

Содержание работы

1. Разработать прикладной протокол для передачи текстовых сообщений с помощью транспортного протокола UDP. Предусмотреть возможность разбиения больших сообщений на отдельные фрагменты.
2. Разработать в Visual C++ приложение Windows для обмена сообщениями по протоколу, разработанному в п. 1. Отправка сообщений должна производиться как широковещательно, так и с указанием адреса назначения (имя или IP-адрес).
3. Разработать прикладной протокол для передачи файлов с помощью транспортного протокола TCP.
4. Разработать в Visual C++ два приложения Windows для копирования файлов по протоколу, разработанному в п. 3:

− серверное приложение, которое принимает файлы от клиентских приложений и сохраняет их на диске;

− клиентское приложение, которое должно подключаться к серверному приложению для отправки файлов. Предусмотреть возможность одновременного копирования нескольких файлов.

5. Протестируйте работу приложений, разработанных в п. 2 и 4, на нескольких компьютерах под управлением Windows.

Ход работы

1. В соответствии с определенным в п.1. задании был разработан прикладной протокол для передачи текстовых сообщений с помощью транспортного протокола UDP. Структура пакета определена на рисунке 4.1. :

```
#pragma pack(1)
struct SLP_msg
{
    int filelen;          //длина сообщени¤
    int numberfrag;      //номер фрагмента
    WCHAR username[20]; //имя отправителя
    WCHAR text[10];      //текст сообщения
};
#pragma pack()
```

Рис. 4.1. Структура фрагмента: UDP

Поле filelen содержит общую длину передаваемого сообщения, которая в дальнейшем будет использована для определения количества переданных пакетов.

Поле numberfrag содержит номер передаваемого фрагмента. Каждый отправляемый фрагмент перед отправкой получает номер, начиная с 0. Получение сообщения будет производиться до тех пор, пока не будет получен последний фрагмент.

Поле username содержит в себе имя отправителя, которое будет выведено на экран вместе с полученным сообщением.

Поле text содержит в себе часть сообщения фиксированной длины. Таким образом зная длину всего сообщения и размер этого поля можно определить какое количество пакетов должно быть получено.

В соответствии с определенным в п.3. задании был разработан прикладной протокол для передачи файлов с помощью транспортного протокола TCP. Структуры описаны на рисунке

```
#pragma pack(1)
struct AdressHeader
{
    int CountOfFiles;       //количество файлов
    TCHAR adr[128] = L"";   //адрес отправителя
};
#pragma pack()

#pragma pack(1)
struct MainHeader
{
    int filesize; //размер файла
    TCHAR filename[50]; // имя файла
};
#pragma pack()
```

Рис. 4.2. Структуры фрагмента: заголовок и данные

Структура AddressHeader будет отправлена серверу перед началом отправки файлов. В содержимом структуры определены адрес отправителя и передаваемое количество файлов, которое используется при получении файлов сервером.

Структура MainHeader будет отправлена серверу после отправки первой структуры. Её содержимое включает в себя поля filesize, в котором будет записан размер передаваемого файла, а в поле filename определено его имя. Получив структуру сервер сможет создать файл в соответствии с содержимым ее полей. А затем выполнит запись полученных данных в созданный файл. Процесс будет выполняться до тех пор, пока количество полученных сегментов не станет равным содержимому поля CountOfFiles из структуры AddressHeader.

2. Блок-схемы алгоритмов представлены на рисунках:

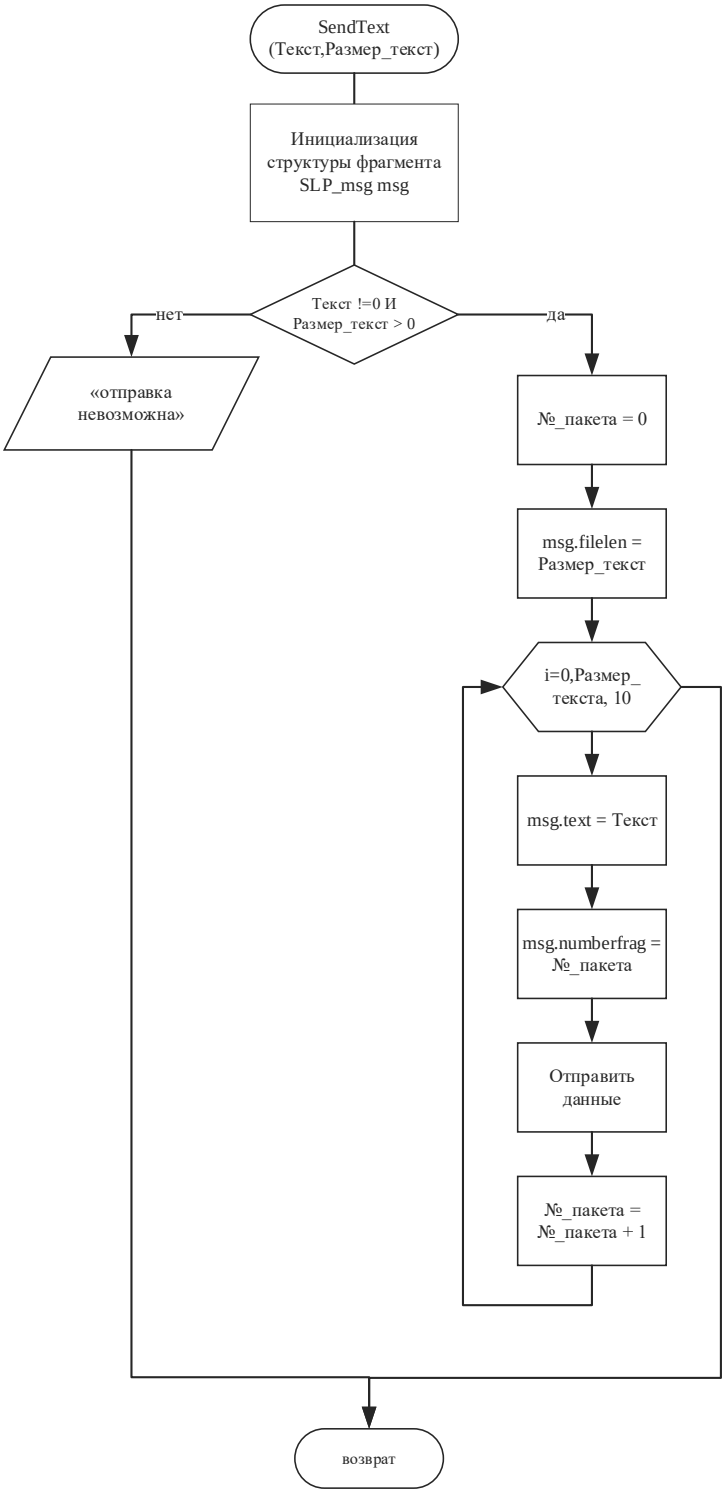

Рис. 4.3. UDP: Блок-схема алгоритма отправки сообщения

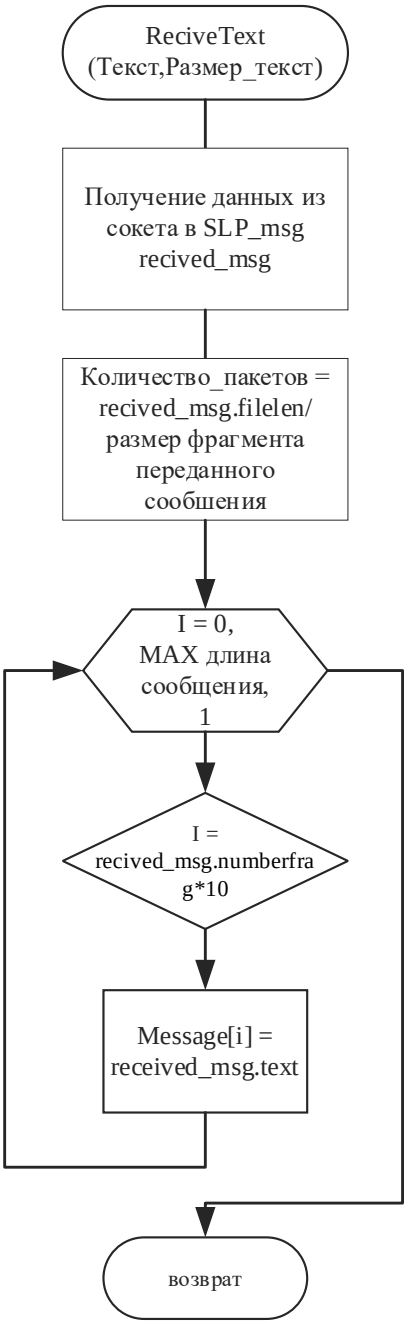

Рис. 4.4. UDP: Блок-схема алгоритма чтения сообщения

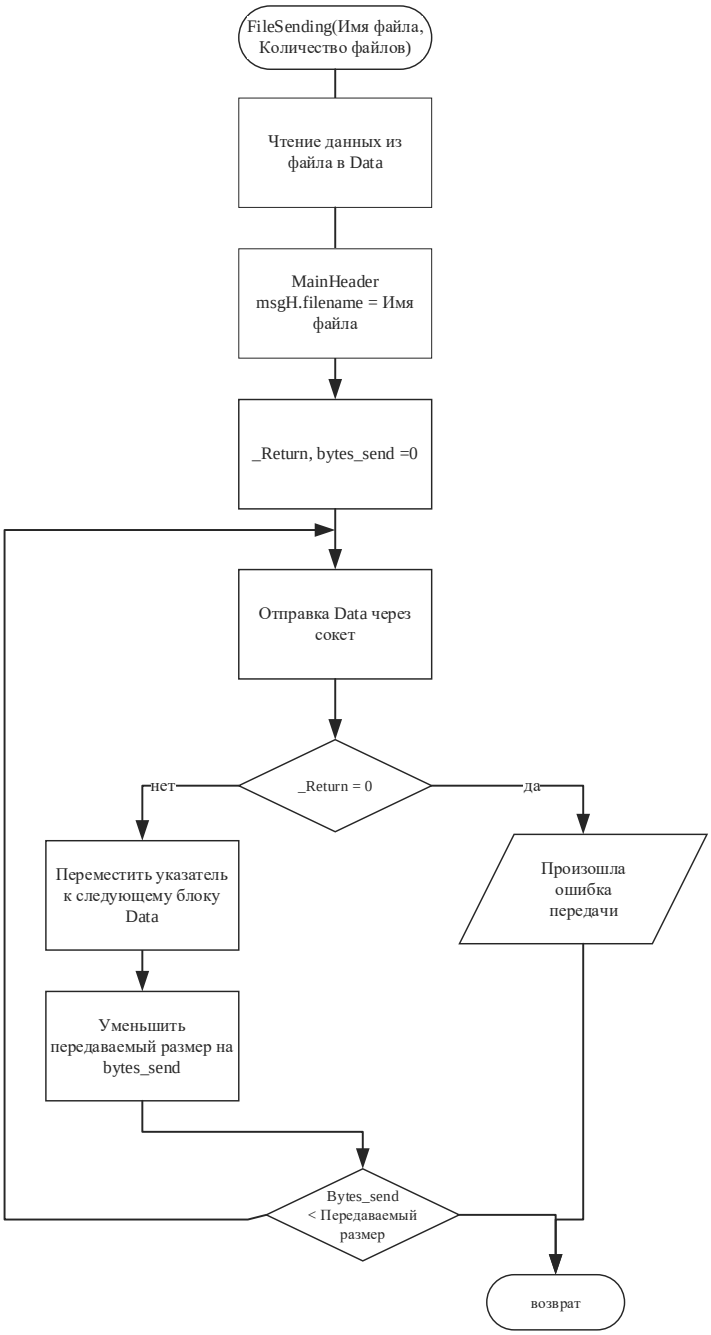

Рис. 4.5. TCP: Блок-схема алгоритма отправки файла

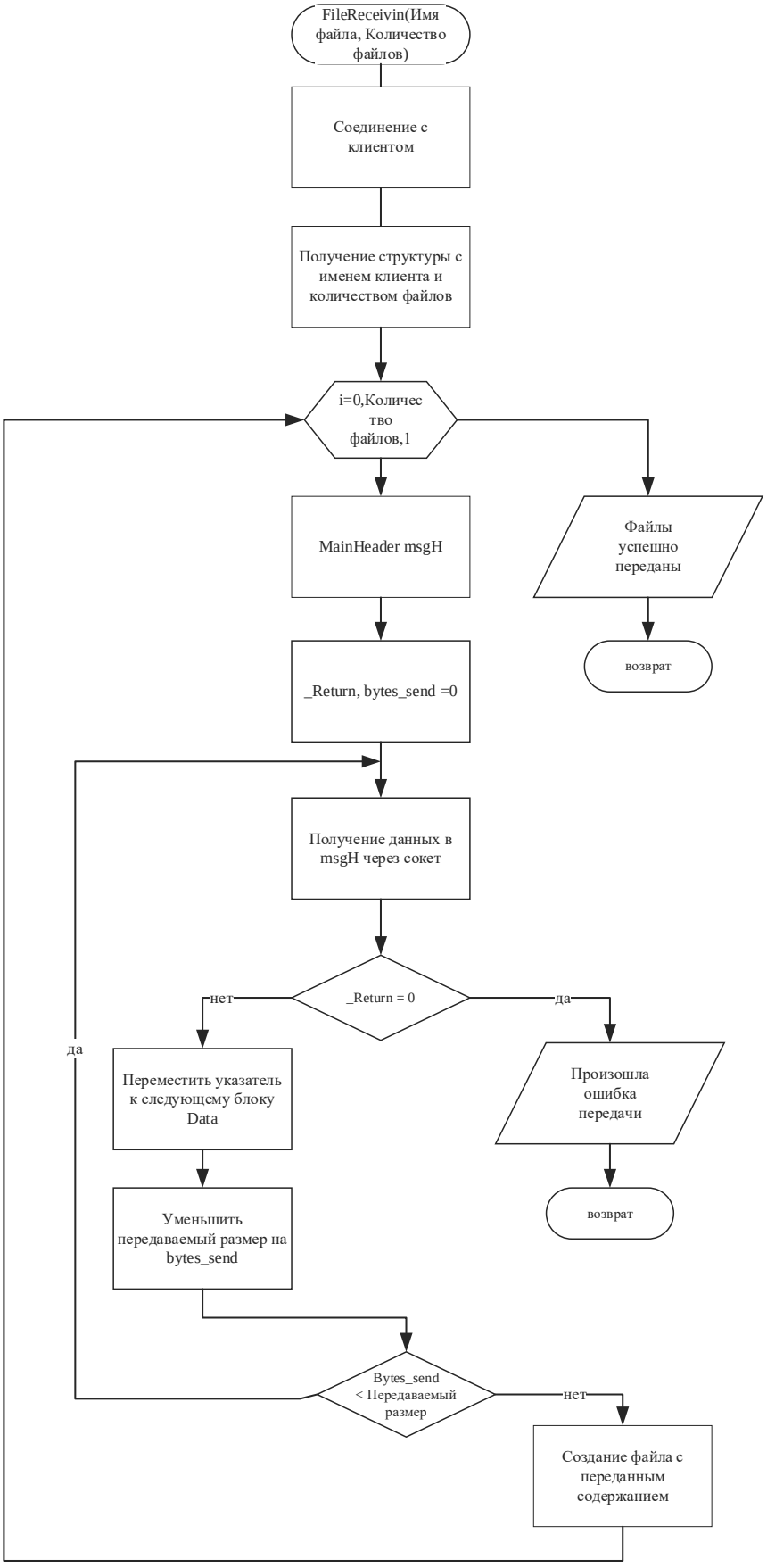

Рис. 4.6. TCP: Блок-схема алгоритма получения файла

3.

Листинг 1. Приложение 1 для обмена сообщениями с помощью UDP

```
//данные для отправки
#define IDC_EDIT_MESSAGES         2001
#define IDC_EDIT_TEXT             2002
#define IDC_EDIT_USERNAME         2003
#define IDC_CLEAR_ALL             2004 //кнопка чтобы удалить весь отправленный текст

//переключение адреса
#define IDC_IPADDR                2005 //чтобы ввести целевой IP
#define IDC_CONNECT               2006
#define IDC_DISCONNECT            2007

#define MAX_TEXT                  1024

//размеры текста с сообщением
#define MAX_MESSAGE_SIZE          255
#define MAX_USERNAME_SIZE         20
#define MAX_USERMESSAGE_SIZE      532
#define FRAFMENT_PACK_SIZE        10
/*Дескрипторы*/
HWND hwnd = NULL; // дескриптор главного окна

/*Оконные процедуры*/
LRESULT CALLBACK MainWindowProcess(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam);

BOOL PreTranslateMessage(LPMSG lpMsg);//для передачи сообщений при нажатии на Enter

/*Обработчики WM_CREATE, WM_DESTROY, WM_SIZE*/
BOOL OnCreate(HWND hwnd, LPCREATESTRUCT lpCreateStruct);
void OnDestroy(HWND hwnd);
void OnCommand(HWND hwnd, int id, HWND hwndCtl, UINT codeNotify);

HANDLE hThread; //поток для приема сообщений от экземпляров приложения

unsigned _stdcall ThreadFunc(LPVOID lParam); //чтение сообщений здесь

//пакет для передачи
#pragma pack(1)
struct SLP_msg
{
	int filelen;          //длина сообщени¤
	int numberfrag;             //номер фрагмента
	WCHAR username[20]; //имя отправителя
	WCHAR text[10];             //текст сообщения
};
#pragma pack()

//работа с сокетами
SOCKET sockets;

sockaddr_in sockSin = { 0 };
sockaddr_in sockSout = { 0 };
sockaddr_in sockSoin = { 0 };


void SendText(SLP_msg msg, LPCTSTR Send_Data, unsigned int Send_Data_Size);//отправка сообщений
void StartChat(HWND hwnd, LPCTSTR Message);//вывод имени отправителя и сообщения на экран
int WINAPI _tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR CmdLine, int CmdShow)
{
	/*Регистрация оконного класса и обработка ошибки*/
	WNDCLASSEX wcex = { sizeof(WNDCLASSEX) };
	wcex.style = CS_HREDRAW | CS_VREDRAW | CS_DBLCLKS;
	wcex.lpfnWndProc = MainWindowProcess; // оконная процедура
	wcex.hInstance = hInstance;
	wcex.hIcon = LoadIcon(NULL, IDI_APPLICATION);
```

```
	wcex.hCursor = LoadCursor(NULL, IDC_ARROW);
	wcex.hbrBackground = (HBRUSH)CreateSolidBrush(RGB(0, 100, 256));
	wcex.lpszClassName = TEXT("MainWindowProcess"); // имя класса
	wcex.hIconSm = LoadIcon(NULL, IDI_APPLICATION);
	if (0 == RegisterClassEx(&wcex))
	{
		return -1;
	}
	/*-----------------------------------------------*/
	LoadLibrary(TEXT("ComCtl32.dll"));
	LoadLibrary(TEXT("Ws2_32.dll"));//for sockets
	/*Создание главного файла и обработка ошибки */
	hwnd = CreateWindowEx(0, TEXT("MainWindowProcess"), TEXT("Chat"),
		WS_OVERLAPPEDWINDOW, CW_USEDEFAULT, 0, 900, 600, NULL, NULL, hInstance, NULL);
	if (hwnd == NULL)	{	return -1;	}
	/*-----------------------------------------------*/
	ShowWindow(hwnd, CmdShow); // отображаем главное окно

	/*Цикл обработки сообщений*/
	MSG  msg;
	BOOL RetRes;
	while ((RetRes = GetMessage(&msg, NULL, 0, 0)) != FALSE)
	{
		if (RetRes == -1)
		{
			//Error editing
		}
		else if (!PreTranslateMessage(&msg))
		{
			TranslateMessage(&msg);
			DispatchMessage(&msg);
		}
	}

	return (int)msg.wParam;
}


/*Процедура главного окна*/
LRESULT CALLBACK MainWindowProcess(HWND hWnd, UINT msg, WPARAM wParam, LPARAM lParam)
{
	switch (msg)
	{
		HANDLE_MSG(hWnd, WM_CREATE, OnCreate);
		HANDLE_MSG(hWnd, WM_DESTROY, OnDestroy);
		HANDLE_MSG(hWnd, WM_COMMAND, OnCommand);
	}
	return DefWindowProc(hWnd, msg, wParam, lParam);
}



/*Обработчик сообщений*/
BOOL PreTranslateMessage(LPMSG Msg)
{
	/*Переменные*/
	WCHAR Message[MAX_MESSAGE_SIZE]; //сообщение
	WCHAR UserName[MAX_USERNAME_SIZE] = _T("");//имя отправителя
	WCHAR UserMessage[MAX_USERMESSAGE_SIZE] = _T("");//сообщение + имя отправителя

	//объявим структуру с пакетом
	SLP_msg msg;
	memset(msg.text, NULL, 10);
	memset(msg.username, NULL, 20);

	DWORD symbols, symb_user;  //количество символов в сообщении
```

```
	if ((WM_KEYDOWN == Msg->message) && (VK_RETURN == Msg->wParam)) // нажата клавиша Enter
	{
		HWND hwndCtl = GetDlgItem(hwnd, IDC_EDIT_TEXT);
		HWND hwndUser = GetDlgItem(hwnd, IDC_EDIT_USERNAME);

		if (GetFocus() == hwndCtl) // поле ввода обладает фокусом клавиатуры
		{
			/*Чтобы можно было отправить многострочный текст*/
			/*CTRL+ENTER*/
			if (GetKeyState(VK_SHIFT) < 0) // нажата клавиша SHIFT
			{
				Edit_ReplaceSel(hwndCtl, _T("\r\n"));
			}
			else
			{
				symbols = Edit_GetText(hwndCtl, Message, _countof(Message));//
копируем текст из поля ввода
				symb_user = Edit_GetText(hwndUser, UserName, _countof(UserName));//
копируем текст из поля ввода

				if (symbols > 0)
				{
					// очищаем поле ввода
					Edit_SetText(hwndCtl, NULL);

					//скопируем введенные данные в структуру
					StringCchCat(msg.username,20, UserName);

					StartChat(hwnd,Message);//отобрзим отправляемый текст у себя

					SendText(msg, Message, wcslen(Message));

				}
			}
			return TRUE;
		}
	}
	return FALSE;
}

BOOL OnCreate(HWND hwnd, LPCREATESTRUCT lpCreateStruct)
{

	DWORD dwStyle = WS_CHILD | WS_VISIBLE | WS_VSCROLL | WS_HSCROLL | ES_MULTILINE |
ES_AUTOHSCROLL | ES_AUTOVSCROLL;

	// создаём поле вывода сообщений
	CreateWindowEx(WS_EX_STATICEDGE, TEXT("Edit"), TEXT(""), dwStyle | ES_READONLY,
		10, 10, 490, 250, hwnd, (HMENU)IDC_EDIT_MESSAGES, lpCreateStruct->hInstance, NULL);

	//Для username
	CreateWindowEx(0, TEXT("Static"), TEXT("User: "), WS_CHILD | WS_VISIBLE | SS_SIMPLE,
		10, 270, 40, 40, hwnd, NULL, lpCreateStruct->hInstance, NULL);

	CreateWindowEx(WS_EX_CLIENTEDGE, TEXT("Edit"), TEXT(""), WS_CHILD | WS_VISIBLE,
		50, 270, 450, 40, hwnd, (HMENU)IDC_EDIT_USERNAME, lpCreateStruct->hInstance, NULL);

	CreateWindowEx(0, TEXT("Button"), TEXT("Clear History"), WS_CHILD | WS_VISIBLE |
BS_PUSHBUTTON,
		520, 10, 200, 30, hwnd, (HMENU)IDC_CLEAR_ALL, lpCreateStruct->hInstance, NULL);

	// создаём поле ввода
	HWND hwndCtl = CreateWindowEx(WS_EX_CLIENTEDGE, TEXT("Edit"), TEXT(""), dwStyle,
		10, 320, 490, 140, hwnd, (HMENU)IDC_EDIT_TEXT, lpCreateStruct->hInstance, NULL);

	//Для работы с адресами
```

```
		// создаём поле ввода IP-адреса
	CreateWindowEx(0, TEXT("SysIPAddress32"), NULL, WS_CHILD | WS_VISIBLE,
		520, 50, 200, 30, hwnd, (HMENU)IDC_IPADDR, lpCreateStruct->hInstance, NULL);

	//кнопка для переключения
	CreateWindowEx(0, TEXT("Button"), TEXT("Connect"), WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON,
		730, 50, 100, 30, hwnd, (HMENU)IDC_CONNECT, lpCreateStruct->hInstance, NULL);

	//кнопка для переключения
	CreateWindowEx(0, TEXT("Button"), TEXT("Disconnect"), WS_CHILD | WS_VISIBLE |
BS_PUSHBUTTON,
		730, 90, 100, 30, hwnd, (HMENU)IDC_DISCONNECT, lpCreateStruct->hInstance, NULL);

	// задаём ограничение на ввод текста
	Edit_LimitText(hwndCtl, MAX_MESSAGE_SIZE);


	//Инициализация библиотеки WinSock
	WSADATA WsaData;
	int err = WSAStartup(MAKEWORD(2,2),&WsaData);

	// Открытие сокета
	sockets = socket(AF_INET, SOCK_DGRAM,NULL);

	//заполним структур, чтобы через можно было получать данные
	sockSout.sin_family = AF_INET; // AF_INET = IPv4 addresses
	sockSout.sin_port = htons(7581); // Little to big endian conversion
	sockSout.sin_addr.s_addr = htonl(INADDR_ANY);//0.0.0.0

	//выполним ассоциирование сокета
	err = bind(sockets, (sockaddr*)&sockSout, sizeof(sockSout));

	if (err != SOCKET_ERROR)
	{
		hThread = (HANDLE)_beginthreadex(NULL, 0, ThreadFunc, NULL, 0, NULL);
	}
	return TRUE;
}

void OnDestroy(HWND hwnd)
{
	// уведомляем о завершении приложения,
	closesocket(sockets);
	WSACleanup();
	PostQuitMessage(0); // отправляем сообщение WM_QUIT
}
void OnCommand(HWND hwnd, int id, HWND hwndCtl, UINT codeNotify)
{
	if (BN_CLICKED == codeNotify)
	{
		HINSTANCE hInstance = GetWindowInstance(hwnd);
		switch (id)
		{
			case IDC_CLEAR_ALL:
			{
				HWND hwndCtl = GetDlgItem(hwnd, IDC_EDIT_MESSAGES);
				Edit_SetText(hwndCtl, NULL);// очищаем поле ввода
			}
			break;
			case IDC_CONNECT:
			{
				// получим IP-адрес из поля ввода
				const char bufferNameIP[25] = { 0 };
				GetDlgItemTextA(hwnd, IDC_IPADDR, (LPSTR)bufferNameIP,
_countof(bufferNameIP));

		if (bufferNameIP != NULL)
```

```
		{
			sockaddr_in sin = { 0 };
			sockSin.sin_family = AF_INET;
			sockSin.sin_port = htons(7581);
			sockSin.sin_addr.s_addr = inet_addr(bufferNameIP);//"192.168.56.104"
"192.168.56.1"
			MessageBox(hwnd, L"IP-connect Enable", L"Details", MB_OK);
			}
			else
			{
				MessageBox(hwnd, L"Null IP", L"Details", MB_OK);
			}
		}
			break;
			case IDC_DISCONNECT:
			{
				// очистим поле ввода IP-адреса
				SendDlgItemMessage(hwnd, IDC_IPADDR, IPM_CLEARADDRESS,0,0);

				BOOL optval = TRUE;
				int optlen = sizeof(optval);
				int err = setsockopt(sockets, SOL_SOCKET, SO_BROADCAST,
(char*)&optval, optlen);
				sockSin.sin_family = AF_INET;
				sockSin.sin_port = htons(7581);
				sockSin.sin_addr.s_addr = htonl(INADDR_BROADCAST);

				MessageBox(hwnd, L"BroadCast Enable", L"Details", MB_OK);
			}
			break;
		}
	}
}


// ----------------------------------------------------------------------------------------------
void SendText(SLP_msg msg, LPCTSTR Send_Data, unsigned int Send_Data_Size)
{

	if ((Send_Data != NULL) && (Send_Data_Size > 0))
	{
		int packnum = 0;//номер пакета

		msg.filelen = Send_Data_Size;

		for (int i = 0; i < Send_Data_Size; i+=10)
		{
		WCHAR frag_pack[FRAFMENT_PACK_SIZE] = L""; //инициализация фрагмента пакета
		memcpy_s(frag_pack, sizeof(frag_pack), &Send_Data[i], sizeof(frag_pack));//скопируем
данные в фрагмента пакета

			StringCchCat(msg.text,sizeof(msg.text), frag_pack);//скопируем данные в
отправялемый пакет
			msg.numberfrag = packnum;//запишем номер пакета

			int result = sendto(sockets,(const char*)&msg, sizeof(msg), NULL,(struct
sockaddr*)&sockSin, sizeof(sockSin));

			packnum++;//увеличим номер пакета для следующей отправки
			ZeroMemory(msg.text, sizeof(msg.text));
		}
	}
}

void StartChat(HWND hwnd, LPCTSTR Message)
{
	MessageBox(hwnd, L"LET'S GO CHAT!", L"CHAT", MB_OK);
```

```
	WCHAR UserMessage[MAX_USERMESSAGE_SIZE] = _T("");

	HWND hwndCtl = GetDlgItem(hwnd, IDC_EDIT_MESSAGES);

	StringCchCat(UserMessage, _countof(UserMessage), _T("\r\n"));
	StringCchCat(UserMessage, _countof(UserMessage), Message);

	Edit_SetSel(hwndCtl, Edit_GetTextLength(hwndCtl), -1);// устанавливаем курсор в конец поля
вывода

	SetFocus(hwnd);

	Edit_ReplaceSel(hwndCtl, UserMessage);// вставляем текст в поле вывода
}

unsigned __stdcall ThreadFunc(void* lParam)
{
	/*Переменные*/
	WCHAR Message[MAX_MESSAGE_SIZE] = _T("");                    //сообщение
	WCHAR UserName[MAX_USERNAME_SIZE] = _T("");                  //имя отправителя
	WCHAR UserMessage[MAX_USERMESSAGE_SIZE] = _T("");     //имя отправителя+сообщение

	SLP_msg recived_msg = {0};
	int socket_len  = sizeof(sockSout);

		for (;;)
		{
			//Начнем прием сообщений через сокет
			int result =
recvfrom(sockets,(char*)&recived_msg,sizeof(recived_msg),NULL,(struct sockaddr*)&sockSout,
&socket_len);

			if(result != SOCKET_ERROR)
			{
				int struct_size = sizeof(recived_msg);//узнаем размер структуры

				int reseived_size = recived_msg.filelen / _countof(recived_msg.text);
//количество пакетов

				/*Собираем сообщение*/
				for (int i = 0; i < MAX_MESSAGE_SIZE; i++)
				{
				//будем выполнять сдвиг указателя в массиве Message
				//пока не дойдем до конца уже заполненной части массива
					if (i == recived_msg.numberfrag*FRAFMENT_PACK_SIZE)
					{
					StringCchCatW(Message, _countof(Message), recived_msg.text);
					}
			}
				//если определенный через размер номер равен номеру пакета, значит
сообщение было получено верно

				if (reseived_size == recived_msg.numberfrag)
				{
					/*Заполнение массивов для вывода имени отправителя и
сообщения*/
					StringCchCat(UserName, 20, recived_msg.username);

					StringCchCat(UserName, _countof(UserName), _T(":"));

					StringCchCat(UserMessage, _countof(UserMessage), UserName);

					StringCchCat(UserMessage, _countof(UserMessage), Message);

					StartChat(hwnd, UserMessage);
```

```
                                /*Освобождение ресурсов и т.д.*/
                                memset(UserName, NULL, 255);
                                memset(UserMessage, NULL, 255);
                                memset(Message, NULL, 255);
                        }
                }
        }
return(0);
}
```

Листинг 2. Приложение 2 для обмена сообщениями с помощью TCP: клиент

```
#define BUTTON_CONNECTION         2001 //соединение с сервером
#define BUTTON_DISCONNECT         2002 //разрыв соединения с сервером

#define BUTTON_CHOOSE_FILE        2003

#define BUTTON_SEND               2004


#define IDC_IPADDR                2005 //ввод IP-адреса
#define IDC_USERNAME              2006 //ввод имени отправителя
#define IDC_LIST                  2007 //вывод списка файлов

#pragma pack(1)
struct AdressHeader
{
	int CountOfFiles;
	TCHAR adr[128] = L""; //адрес отправителя
};
#pragma pack()

#pragma pack(1)
struct MainHeader
{
	int filesize; //размер файла
	TCHAR filename[50]; // имя файла
};
#pragma pack()


SOCKET sender_socket;//сокет для передачи данных

//volatile bool stoped = false;

AdressHeader msgA;

wchar_t FileNameTitles[MAX_PATH] = L"";//хранит указатель на папку, если выбрано более одного
файла
wchar_t FileNameTitle[MAX_PATH] = L"";//использовать если выбран один файл


BOOL FileSending(wchar_t* NameOfFiles);//функция для отправки файлов, передаем их имена

DWORD result;

LRESULT CALLBACK WindowProc(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam);

void OnIdle(HWND hWnd);
BOOL OnCreate(HWND hWnd, LPCREATESTRUCT lpCreateStruct);
void OnDestroy(HWND hWnd);
void OnCommand(HWND hwnd, int id, HWND hwndCtl, UINT codeNotify);

void sendfile(SOCKET send_socket, const char* buf, int len);
int WINAPI _tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpszCmdLine, int nCmdShow) {

	WNDCLASSEX wcex = { sizeof(WNDCLASSEX) };
```

```
	wcex.style = CS_HREDRAW | CS_VREDRAW | CS_DBLCLKS;
	wcex.lpfnWndProc = WindowProc;
	wcex.hInstance = hInstance;
	wcex.hIcon = LoadIcon(NULL, IDI_APPLICATION);
	wcex.hCursor = LoadCursor(NULL, IDC_ARROW);
	wcex.hbrBackground = (HBRUSH)(COLOR_BTNFACE + 2);
	wcex.lpszMenuName = NULL;
	wcex.lpszClassName = TEXT("WindowClass");
	wcex.hIconSm = LoadIcon(NULL, IDI_APPLICATION);

	if (0 == RegisterClassEx(&wcex))
	{
		return -1;
	}

	LoadLibrary(TEXT("ComCtl32.dll"));

	HWND hWnd = CreateWindowEx(NULL, TEXT("WindowClass"), TEXT("Client"),
		WS_OVERLAPPEDWINDOW, CW_USEDEFAULT, CW_USEDEFAULT, 700, 400, NULL, NULL, hInstance,
NULL);

	if (NULL == hWnd)
	{
		return -1;
	}

	ShowWindow(hWnd, nCmdShow);

	MSG msg;
	BOOL bRet;

	for (;;)
	{
		while (!PeekMessage(&msg, NULL, 0, 0, PM_NOREMOVE))
		{
			OnIdle(hWnd);
		}

		bRet = GetMessage(&msg, NULL, 0, 0);

		if (bRet == -1)
		{
		}
		else if (FALSE == bRet)
		{
			break;
		}
		else
		{
			TranslateMessage(&msg);
			DispatchMessage(&msg);
		}
	}

	return (int)msg.wParam;
}

BOOL FileSending(wchar_t* NamesOfFile, int CountOfFiles)
{
	std::wstring filename = L"";

	if (CountOfFiles > 1)
	{
		/*выполнение первоначального сдвига, если в начале у нас был указатель на
директорию*/
		filename = NamesOfFile;
```

```
			NamesOfFile += (filename.length() + 1);
		}


		while (NamesOfFile != L"")
		{
			std::ifstream file_open(NamesOfFile, ios::in|ios::binary); // создание входного
потока

			/*не влияет???*/
			//чтобы русский язык нормально определялся в буфере wchar_t
			std::locale loc(std::locale(), new std::codecvt_utf8<__int32>);
			file_open.imbue(loc);

			//узнаем размер файла
			file_open.seekg(0,wifstream::end);//перейдем в конец файла
			int size = file_open.tellg();//определим размер файла
			file_open.seekg(0);//вернемся в начало файла

			char* buffer = new char[size+1];//инициализация буфера

			file_open.read(buffer, size);

			if (file_open)
			{
				MainHeader msgH; //пакет с файлом

				StringCchCopy(msgH.filename, MAX_PATH, NamesOfFile);
				msgH.filesize = size;
				sendfile(sender_socket, (const char*)&msgH, sizeof(msgH));
				sendfile(sender_socket, (const char*)buffer, size);

				/*освобождение ресурсов т.д. */
				delete[] buffer;

				file_open.close();
				filename = NamesOfFile;

				NamesOfFile += (filename.length() + 1);//сдвиг к следующему файлу
			}

			else
			{
				delete[] buffer;
				file_open.close();
				return 0;
				//обработка ошибки чтения файла должна быть где-то здесь....
			}
		}
		return 0;
}
LRESULT CALLBACK WindowProc(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam) {
		switch (uMsg)
		{
			HANDLE_MSG(hWnd, WM_CREATE, OnCreate);
			HANDLE_MSG(hWnd, WM_DESTROY, OnDestroy);
			HANDLE_MSG(hWnd, WM_COMMAND, OnCommand);
			break;
		}
		return DefWindowProc(hWnd, uMsg, wParam, lParam);
}
BOOL OnCreate(HWND hWnd, LPCREATESTRUCT lpCreateStruct)
{
		/*Кнопки*/
		CreateWindowEx(0, WC_BUTTON, TEXT("Соединиться с сервером"),
			WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON, 275, 100, 240, 27, hWnd,
(HMENU)BUTTON_CONNECTION, lpCreateStruct->hInstance, NULL);
```

```
	CreateWindowEx(0, WC_BUTTON, TEXT("Завершить соединение"),
		WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON, 275, 130, 240, 27, hWnd,
(HMENU)BUTTON_DISCONNECT, lpCreateStruct->hInstance, NULL);

	/*Поле для ввода IP-адреса*/
	CreateWindowEx(0, TEXT("SysIPAddress32"),
		NULL, WS_CHILD | WS_VISIBLE, 20, 100, 240, 27, hWnd, (HMENU)IDC_IPADDR,
lpCreateStruct->hInstance, NULL);
	CreateWindowEx(0, WC_STATIC, TEXT("Адрес сервера:"), WS_CHILD | WS_VISIBLE, 20, 70, 240,
27, hWnd, NULL, lpCreateStruct->hInstance, NULL);

	/*Поле для ввода имени пользователя*/
	CreateWindowEx(0, TEXT("Edit"),
		NULL, WS_CHILD | WS_VISIBLE | WS_BORDER | ES_MULTILINE, 20, 40, 240, 27, hWnd,
(HMENU)IDC_USERNAME, lpCreateStruct->hInstance, NULL);


	CreateWindowEx(0, WC_STATIC, TEXT("Имя:"), WS_CHILD | WS_VISIBLE, 20, 10, 240, 27, hWnd,
NULL, lpCreateStruct->hInstance, NULL);

	/*Работа с файлами*/
	CreateWindowEx(0, WC_BUTTON, TEXT("Выбрать файлы/файлы"),
		WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON, 360, 230, 240, 27, hWnd,
(HMENU)BUTTON_CHOOSE_FILE, lpCreateStruct->hInstance, NULL);

	CreateWindowEx(0, WC_BUTTON, TEXT("Передать файл"),
		WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON, 360, 260, 240, 27, hWnd, (HMENU)BUTTON_SEND,
lpCreateStruct->hInstance, NULL);

	CreateWindowEx(0, TEXT("ListBox"),
		NULL, WS_CHILD | WS_VISIBLE | WS_BORDER | LBS_WANTKEYBOARDINPUT |
LBS_NOTIFY|WS_VSCROLL|WS_HSCROLL, 20, 170, 320, 150, hWnd, (HMENU)IDC_LIST, lpCreateStruct-
>hInstance, NULL);

	/*Инициализация сокета*/
	WSADATA wsaData;
	int err = WSAStartup(MAKEWORD(2, 2), &wsaData);
	sender_socket = socket(AF_INET, SOCK_STREAM, 0);

	if (sender_socket == INVALID_SOCKET)
	{
		return WSAGetLastError();
	}
	return TRUE;
}

void OnDestroy(HWND hWnd)
{
	closesocket(sender_socket);
	WSACleanup();
	PostQuitMessage(0);
}
void OnCommand(HWND hwnd, int id, HWND hwndCtl, UINT codeNotify)
{
	if (BN_CLICKED == codeNotify)
	{
		HINSTANCE hInstance = GetWindowInstance(hwnd);
		switch (id)
		{
		case BUTTON_CONNECTION:
		{
			const char bufferNameIP[25] = {0};
			GetDlgItemTextA(hwnd, IDC_IPADDR, (LPSTR)bufferNameIP,
_countof(bufferNameIP));
			sockaddr_in sin = { 0 };
			sin.sin_family = AF_INET; //семейство протоколов
```

```
                    sin.sin_port = htons(7581); //номер порта
                    sin.sin_addr.s_addr = inet_addr(bufferNameIP);
                    int result = connect(sender_socket, (sockaddr*)&sin, sizeof(sin));
                    if (result == SOCKET_ERROR)
                    {
                           closesocket(sender_socket);
                           sender_socket = INVALID_SOCKET;
                           MessageBox(NULL, TEXT("Не удалось установить соединение с сервером\n",
result), TEXT("Client"), MB_OK | MB_ICONERROR);
                    }
             }
             break;
             case BUTTON_DISCONNECT:
             {
                    //завершение соединения с сервером.
                    int result = shutdown(sender_socket, SD_SEND);//закрытие канала клиент-сервер
                    closesocket(sender_socket);

                    if (result == SOCKET_ERROR)
                    {
                           closesocket(sender_socket);
                           sender_socket = INVALID_SOCKET;
                           MessageBox(NULL, TEXT("Не удалось завершить соединение с сервером\n",
result), TEXT("Client"), MB_OK | MB_ICONERROR);
                    }
                    else
                    {
                           MessageBox(NULL, TEXT("Удалось завершить соединение с сервером\n",
result), TEXT("Client"), MB_OK | MB_ICONINFORMATION);
                    }
             }
             break;
             case BUTTON_CHOOSE_FILE:
             {
                    OPENFILENAME ofn = {sizeof(OPENFILENAME)};

                    ofn.hInstance = GetModuleHandle(NULL);
                    ofn.lpstrFile = FileNameTitles;//полный путь
                    ofn.nMaxFile = MAX_PATH;
                    ofn.lpstrFilter = TEXT("Files\0*.*");//фильтр
                    ofn.nFilterIndex = 1;
                    ofn.lpstrFileTitle = FileNameTitle;//название файла
                    ofn.nMaxFileTitle = MAX_PATH;
                    ofn.lpstrInitialDir = NULL;
                    ofn.Flags = OFN_PATHMUSTEXIST | OFN_FILEMUSTEXIST | OFN_EXPLORER |
OFN_ALLOWMULTISELECT;

                    if (GetOpenFileName(&ofn) == TRUE)
                    {
                           MessageBox(NULL, TEXT("Файлы успешно выбраны"), TEXT("Client"), MB_OK
| MB_ICONINFORMATION);
                           int nFileOffset = ofn.nFileOffset;

                           HWND hWNDctrl = GetDlgItem(hwnd, IDC_LIST);
                           ListBox_ResetContent(hWNDctrl); //очистка списка просмотра
                           /*Начало подсчета количества файлов*/
                           if (nFileOffset > lstrlen(ofn.lpstrFile))
                           {
                                  wchar_t* str = ofn.lpstrFile;//ссылка на директорию
                                  std::wstring filename = str;
                                  str += (filename.length() + 1);//сдвиг, пропускаем корневую
папку

                                  /*Было выделено более одного файла*/
                                  while (ofn.lpstrFile[nFileOffset])
                                  {
```

```
                                        nFileOffset = nFileOffset + wcslen(ofn.lpstrFile +
nFileOffset) + 1;
                                        msgA.CountOfFiles++;

                                        /*По одному добавляем в listbox*/
                                        int iItem = ListBox_AddString(hWNDctrl, str);
                                        ListBox_SetCurSel(hWNDctrl, iItem);

                                        /*сдвиг к следующему имени файла*/
                                        filename = str;
                                        str += (filename.length() + 1);
                                }
                        }
                        else
                        {
                                /*Был выделен один файл*/
                                msgA.CountOfFiles++;
                                int iItem = ListBox_AddString(hWNDctrl, nFileOffset);
                                ListBox_SetCurSel(hWNDctrl, iItem);
                        }
                }
        }
        break;

        case BUTTON_SEND:
        {
                GetDlgItemTextW(hwnd, IDC_USERNAME, (LPWSTR)msgA.adr,
sizeof(msgA.adr));//Запись имени отправителя в структуру

                sendfile(sender_socket, (const char*)&msgA, sizeof(msgA));//отправим имя
серверу

                if (msgA.CountOfFiles > 1)
                {
                        FileSending(FileNameTitles, msgA.CountOfFiles);//отправка нескольких
экземпляров файла
                }
                else
                {
                        FileSending(FileNameTitle, msgA.CountOfFiles);//единственный экземпляр
файла
                }
                msgA.CountOfFiles = 0; //обнуление счетчика файлов
        }
        break;

        }
    }
}
void sendfile(SOCKET send_socket, const char* Data, int len)
{
    if (Data == nullptr || len == 0)
    {
        return;
    }
    int _return, bytes_send = 0;
    do
    {
        _return = send(send_socket, Data + bytes_send, len - bytes_send, 0);//возвращение
количества отправленных байт
        if (_return == SOCKET_ERROR)
        {
                int err = WSAGetLastError();
                MessageBox(NULL, TEXT("Возникла ошибка", err), TEXT("Client"), MB_OK |
MB_ICONERROR);
                continue;
        }
```

```
            else
            {
                  bytes_send += _return;//сдвиг?
            }
      } while (bytes_send < len);

}
void OnIdle(HWND hWnd) {}
```

Листинг 2. Приложение 2 для обмена сообщениями с помощью TCP: сервер

```
#define BUTTON_DISCONNECT         2001
#define BUTTON_RECEIVE                  2002

#define MAX_MESSAGE_SIZE          255

#pragma pack(1)
struct AdressHeader
{
      int CountOfFiles;           //количество файлов
      TCHAR adr[128] = L"";       //адрес отправителя
};
#pragma pack()

#pragma pack(1)
struct MainHeader
{
      int filesize; //размер файла
      TCHAR filename[50]; // имя файла
};
#pragma pack()


//volatile bool stoped = false;

SOCKET data_socket;//сокет с данными от клиента
SOCKET listen_socket;//сокет для прослушивания потока


sockaddr_in sOut;//описание сокета для работы с протоколом

AdressHeader msgA;//для пакета с имененем адресата и количеством файлов


void recv_file(char* Data, int Size);

unsigned __stdcall ListenThread(LPVOID lpParameter);//для ожидания соединения


LRESULT CALLBACK WindowProc(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam);

BOOL OnCreate(HWND hWnd, LPCREATESTRUCT lpCreateStruct);
void OnDestroy(HWND hWnd);

int WINAPI _tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpszCmdLine, int nCmdShow) {

      WNDCLASSEX wcex = { sizeof(WNDCLASSEX) };

      wcex.style = CS_HREDRAW | CS_VREDRAW | CS_DBLCLKS;
      wcex.lpfnWndProc = WindowProc;
      wcex.hInstance = hInstance;
      wcex.hIcon = LoadIcon(NULL, IDI_APPLICATION);
      wcex.hCursor = LoadCursor(NULL, IDC_ARROW);
      //wcex.hbrBackground = (HBRUSH)(COLOR_WINDOW);
      wcex.hbrBackground = (HBRUSH)CreateSolidBrush(RGB(0, 100, 256));
      wcex.lpszMenuName = NULL;
      wcex.lpszClassName = TEXT("WindowClass");
```

```
	wcex.hIconSm = LoadIcon(NULL, IDI_APPLICATION);

	if (0 == RegisterClassEx(&wcex))
	{
		return -1;
	}

	LoadLibrary(TEXT("ComCtl32.dll"));

	HWND hWnd = CreateWindowEx(NULL, TEXT("WindowClass"), TEXT("Server"),
		WS_OVERLAPPEDWINDOW, CW_USEDEFAULT, CW_USEDEFAULT, 550, 200, NULL, NULL, hInstance,
NULL);

	if (NULL == hWnd)
	{
		return -1;
	}

	ShowWindow(hWnd, nCmdShow);

	MSG msg;
	BOOL bRet;

	for (;;)
	{
		bRet = GetMessage(&msg, NULL, 0, 0);

		if (bRet == -1)
		{
		}
		else if (FALSE == bRet)
		{
			break;
		}
		else
		{
			TranslateMessage(&msg);
			DispatchMessage(&msg);
		}
	}

	return (int)msg.wParam;
}

LRESULT CALLBACK WindowProc(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
	switch (uMsg)
	{

		HANDLE_MSG(hWnd, WM_CREATE, OnCreate);
		HANDLE_MSG(hWnd, WM_DESTROY, OnDestroy);

		case WM_COMMAND:
		{
			switch (LOWORD(wParam))
			{

				case BUTTON_DISCONNECT:
				{
					int err = shutdown(listen_socket, SD_BOTH);
				}
				break;
				case BUTTON_RECEIVE:
				{
					recv_file((char*)&msgA, sizeof(msgA));

					TCHAR Message[MAX_MESSAGE_SIZE] = _T(""); //сообщение
```

```
                                StringCchCat(Message, _countof(Message), _T("Отправитель:"));
                                StringCchCat(Message, _countof(msgA.adr), msgA.adr);

                                MessageBox(NULL, Message, TEXT("Server"), MB_OK |
MB_ICONINFORMATION);

        for (int i = 0; i < msgA.CountOfFiles; i++)
        {
                MainHeader msgH = {0};
                recv_file((char*)&msgH, sizeof(msgH)); //получим пакет с данными
                std::ofstream file_receive(msgH.filename, std::ios::out | std::ios::binary); //
создание выходного потока

                                        //чтобы русский язык нормально определялся в буфере
wchar_t
                                        std::locale loc(std::locale(), new
std::codecvt_utf8<__int32>);
                                        file_receive.imbue(loc);

                                char* Data = new char[msgH.filesize + 1];//инициализация буфера
                                Data[0] = {0};
                                recv_file((char*)Data, msgH.filesize);//чтение данных в буфер
                                file_receive.write((char*)Data, msgH.filesize + 1);
                                file_receive.close();
                        }
                                MessageBox(NULL, TEXT("Прием файлов завершен\n"),
TEXT("Server"), MB_OK | MB_ICONINFORMATION);
                            }
                            break;
                    return TRUE;
                    }
                return 0;
                }
        }

        return DefWindowProc(hWnd, uMsg, wParam, lParam);
}

BOOL OnCreate(HWND hWnd, LPCREATESTRUCT lpCreateStruct) {

        CreateWindowEx(0, WC_BUTTON, TEXT("Завершить соединение"),
                WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON, 150, 20, 240, 30, hWnd,
(HMENU)BUTTON_DISCONNECT, lpCreateStruct->hInstance, NULL);

        CreateWindowEx(0, WC_BUTTON, TEXT("Получить файлы"),
                WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON, 150, 60, 240, 30, hWnd,
(HMENU)BUTTON_RECEIVE, lpCreateStruct->hInstance, NULL);

        WSADATA wsaData;
        u_long argp = 1;
        int result = ioctlsocket(data_socket, FIONBIO, &argp);// перевода сокета в не блокируемое
состояние (nonblocking mode) используется команда FIONBIO

        result = WSAStartup(MAKEWORD(2, 2), &wsaData);

        if (result == 0)
        {
                listen_socket = socket(AF_INET, SOCK_STREAM, 0);

                if (listen_socket != INVALID_SOCKET)
                {
                        //начало ассоциирование сокета
                        sOut.sin_family = AF_INET;
                        sOut.sin_port = htons(7581);
                        sOut.sin_addr.s_addr = htonl(INADDR_ANY);
```

```
                    bind(listen_socket, (sockaddr*)&sOut, sizeof(sOut));//ассоциирование

                    _beginthreadex(NULL, 0, ListenThread, NULL, 0, NULL);//создание потока
прослушивания
             }
       }
       return TRUE;
}

void OnDestroy(HWND hWnd)
{

       closesocket(listen_socket);
       WSACleanup();
       PostQuitMessage(0);
}


void recv_file(char* Data, int Size)
{
       if (Data == nullptr || Size == 0)
       {
             return;
       }
       int _return, bytes_receive = 0;
       do
       {
             _return = recv(data_socket, Data + bytes_receive, Size, 0);
             if (_return == SOCKET_ERROR)
             {
                    int err = WSAGetLastError();
                    MessageBox(NULL, TEXT("Возникла ошибка", err), TEXT("Client"), MB_OK |
MB_ICONERROR);
             }
             else
             {
                    bytes_receive += _return;
                    Size -= _return;
             }
       } while (Size > 0);
}

unsigned __stdcall ListenThread(LPVOID lpParameter)
{
       int result = listen(listen_socket, 5);
       if ( result != SOCKET_ERROR)
       {
             for (;;)
             {
                    data_socket = accept(listen_socket, NULL, NULL);
                    MessageBox(NULL, TEXT("Соединение установлено"), TEXT("Server"), MB_OK |
MB_ICONINFORMATION);
                    if (INVALID_SOCKET == data_socket)
                    {
                           if (WSAEINTR == WSAGetLastError()) break;
                    }
             }
       }
       return 0;
}
```

4. Демонстрация работы приложения 1 на рисунках :

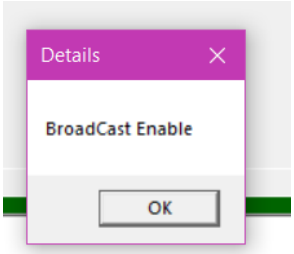

Рис. 4.7. Нажатие на кнопку Disconnect включает широковещательную передачу

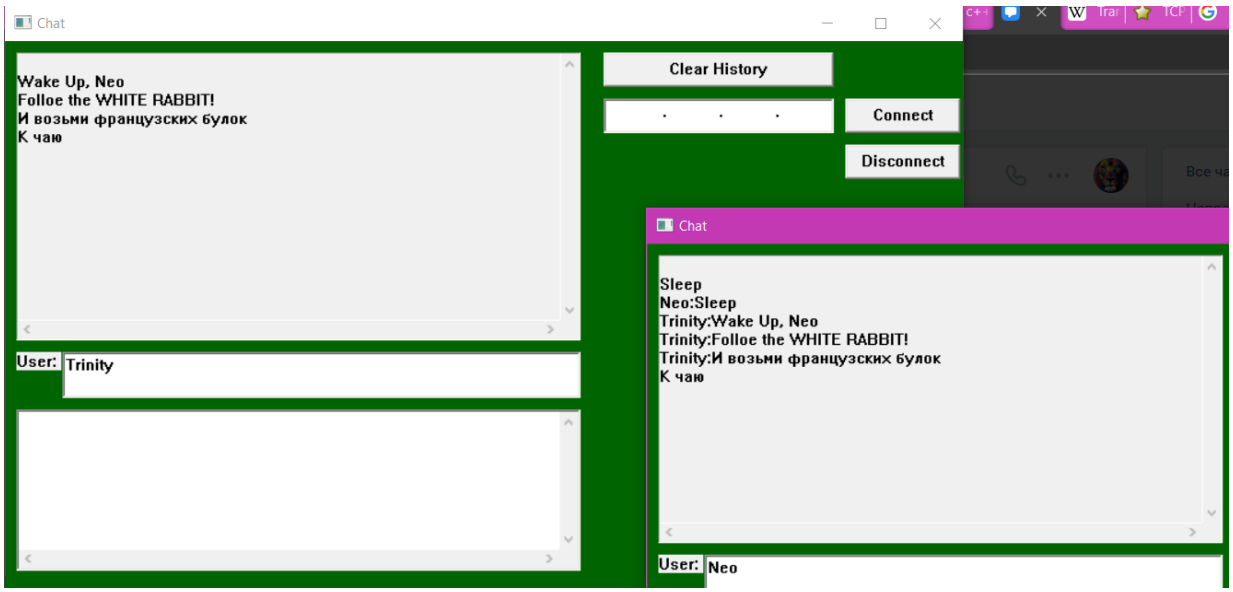

Рис. 4.8. Пример отправки сообщений

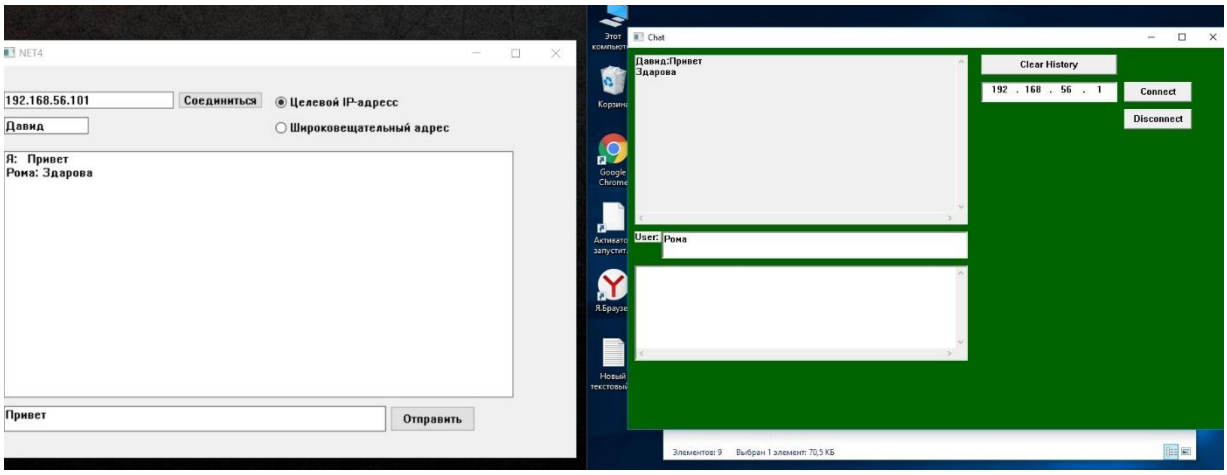

Рис. 4.9. Отправка сообщений на целевой IP-адрес

Демонстрация работа приложения 2 продемонстрирована на рисунках :

Чтобы начать передачу файлов на сервере необходимо нажать на кнопку «Получить файлы», а затем на клиенте – «Передать файл».

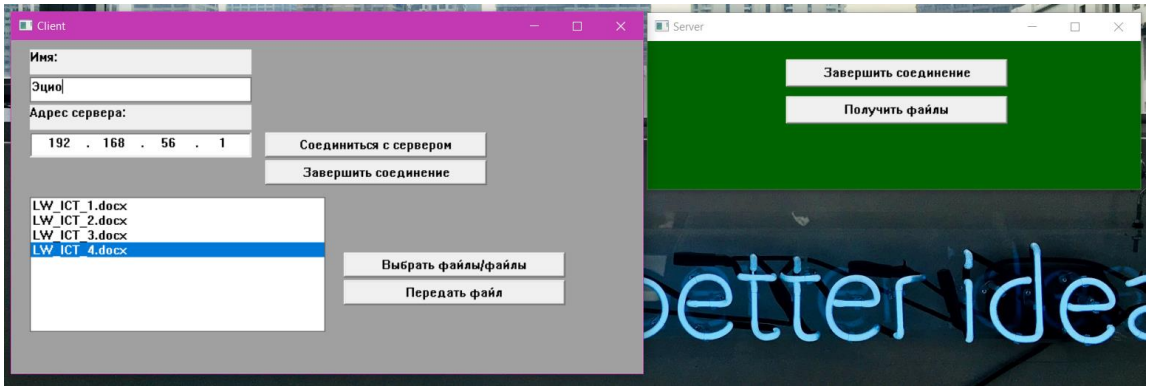

Рис. 4.10. Вид приложения перед началом отправки файлов

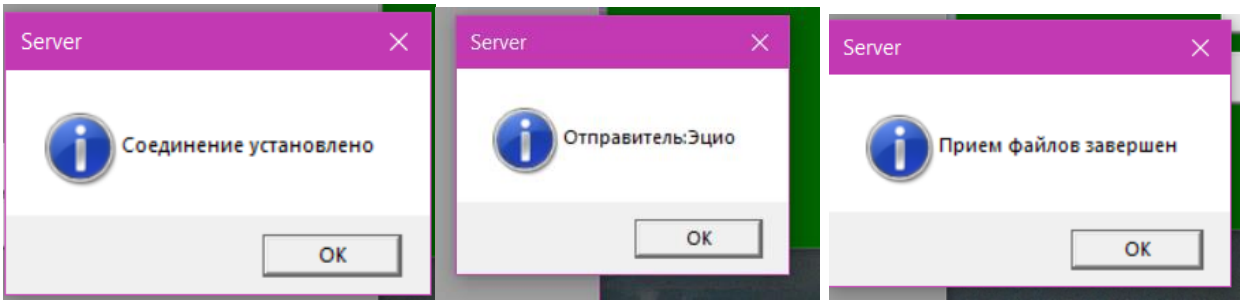

Рис. 4.11. Информационные сообщения

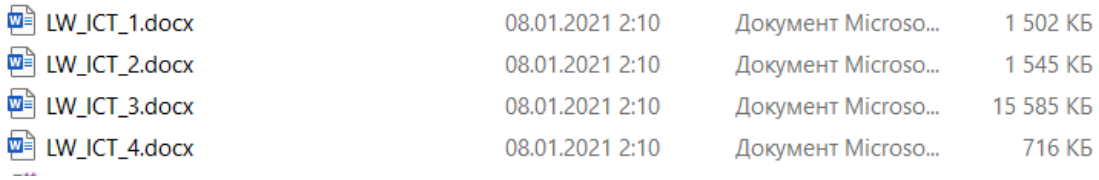

Рис. 4.12. Переданные файлы

Вывод: Таким образом, в ходе выполнения лабораторной работы были получены навыки в проектировании прикладных протоколов для передачи данных и реализации этих протоколов в приложениях Windows на языке C/C++ с применением Windows Sockets