Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет   
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Операционные среды и системное программирование

ОТЧЁТ

к лабораторной работе №8

на тему

Интерфейс сокетов и основы сетевого программирования (Windows). Программирование взаимодействия через сеть с использованием интерфейса сокетов. Реализация сетевых протоколов: собственных или стандартных

Студент: гр.153502

Макаренко А.И.

Проверил: Гриценко Н.Ю.

Минск 2023

**СОДЕРЖАНИЕ**

[1 Цель работы 3](file:///C:\Users\alyona\Downloads\Telegram%20Desktop\ОСиСП%201%20ЛР.docx#_Toc147495008)

[2 Теоретические сведения 4](file:///C:\Users\alyona\Downloads\Telegram%20Desktop\ОСиСП%201%20ЛР.docx#_Toc147495009)

[3 Описание функций программы 6](file:///C:\Users\alyona\Downloads\Telegram%20Desktop\ОСиСП%201%20ЛР.docx#_Toc147495010)

[Список использованных источников 8](file:///C:\Users\alyona\Downloads\Telegram%20Desktop\ОСиСП%201%20ЛР.docx#_Toc147495011)

[Приложение А 9](file:///C:\Users\alyona\Downloads\Telegram%20Desktop\ОСиСП%201%20ЛР.docx#_Toc147495012)

1 ЦЕЛЬ РАБОТЫ

Целью выполнения лабораторной работы является создание приложения на *Win32 API*, которое использует сокеты для обмена данными и реализует собственный или стандартный протокол.

В качестве задачи необходимо построить клиент-серверное приложение для обмена текстовыми сообщениями с использованием сокетов *TCP*.

2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Операционная система *Windows* предоставляет механизмы для упрощения обмена данными и обмена данными между приложениями. В совокупности действия, включенные этими механизмами, называются межпроцессными коммуникациями (*IPC*).

Как правило, приложения могут использовать *IPC*, классифицированные как клиенты или серверы. Клиент – это приложение или процесс, запрашивающий службу из какого-либо другого приложения или процесса. Сервер – это приложение или процесс, который отвечает на запрос клиента.

К основным методам обмена данных относят:

1 буфер обмена;

2 *COM*;

3 копирование данных;

4 динамический обмен данными;

5 сопоставление файлов;

6 *Mailslots*;

7 Каналы;

8 *RPC*;

9 сокеты.

Программный компонент, использующий *COM*, может взаимодействовать с множеством других компонентов, даже тех, которые еще не были написаны. Компоненты взаимодействуют как объекты и клиенты. Распределенная модель *COM* расширяет модель программирования *COM*, чтобы она работала по сети.

Сопоставление файлов позволяет процессу обрабатывать содержимое файла как блок памяти в адресном пространстве процесса. Процесс может использовать простые операции указателя для проверки и изменения содержимого файла. Если два или более процессов обращаются к одному сопоставлению файлов, каждый процесс получает указатель на память в своем адресном пространстве, который он может использовать для чтения или изменения содержимого файла. Процессы должны использовать объект синхронизации, например семафор, чтобы предотвратить повреждение данных в многозадачности.

Почтовые слои обеспечивают односторонний обмен данными. Любой процесс, создающий *mailslot*, является сервером *mailslot*. Другие процессы, называемые клиентами *mailslot*, отправляют сообщения на сервер *mailslot*, записывая сообщения в его *mailslot*. Входящие сообщения всегда добавляются к почтовому слою. *Mailslot* сохраняет сообщения до тех пор, пока сервер *mailslot* не прочитает их. Процесс может быть как сервером *mailslot*, так и клиентом *mailslot*, поэтому двусторонняя связь возможна с использованием нескольких почтовых слотов.

Существует два типа каналов для двусторонней связи: анонимные и именованные каналы. Анонимные каналы позволяют связанным процессам передавать информацию друг другу. Как правило, анонимный канал используется для перенаправления стандартных входных или выходных данных дочернего процесса, чтобы он смог обмениваться данными с родительским процессом. Для обмена данными в обоих направлениях (дуплексная операция) необходимо создать два анонимных каналов. Родительский процесс записывает данные в один канал с помощью дескриптора записи, а дочерний процесс считывает данные из этого канала с помощью дескриптора чтения. Аналогичным образом дочерний процесс записывает данные в другой канал, а родительский процесс считывает из него. Анонимные каналы не могут использоваться по сети, а также между несвязанными процессами.

Именованные каналы используются для передачи данных между процессами, которые не являются связанными, и между процессами на разных компьютерах. Как правило, процесс сервера с именованным каналом создает именованный канал с известным именем или именем, которое должно быть передано клиентам. Клиентский процесс с именованным каналом, который знает имя канала, может открыть его другой конец с ограничениями доступа, заданными серверным процессом именованного канала. После подключения сервера и клиента к каналу они могут обмениваться данными, выполняя операции чтения и записи в канале [1].

Буфер обмена – это набор функций и сообщений, которые позволяют приложениям передавать данные. Так как все приложения имеют доступ к буферу обмена, данные можно легко передавать между приложениями или внутри приложения. Буфер обмена управляется пользователем. Окно должно передавать данные в буфер обмена или из нее только в ответ на команду пользователя. Окно не должно использовать буфер обмена для передачи данных без знаний пользователя. [2]

Копирование данных реализуется через команду *WM\_COPYDATA* и структуру данных *COPYDATASTRUCT*.

Протокол *DDE* (динамического обмена данными) – это набор сообщений и рекомендаций. Он отправляет сообщения между приложениями, которые совместно используют данные, и использует общую память для обмена данными между приложениями. Приложения могут использовать протокол *DDE* для однократной передачи данных и для непрерывного обмена, в ходе которого приложения отправляют обновления друг другу по мере того, как новые данные становятся доступными

*DDEML* – это библиотека динамической компоновки (*DLL*), которую приложения могут использовать для совместного использования данных. *DDEML* предоставляет функции и сообщения, упрощающие задачу добавления возможности *DDE* в приложение [3].

Сокет – это виртуальная конструкция из *IP*-адреса и номера порта, служащая для передачи данных [4].

Обмен данными, ориентированный на соединения, может использовать надежную связь, для обеспечения которой протокол уровня 4 посылает подтверждения о получении данных и запрашивает повторную передачу, если данные не получены или искажены. Протокол *TCP* использует именно такую надежную связь. *TCP* используется в таких прикладных протоколах, как *HTTP*, *FTP*.

Протокол *TCP* требует, чтобы перед отправкой сообщения было открыто соединение. Серверное приложение должно выполнить так называемое пассивное открытие (*passive open*), чтобы создать соединение с известным номером порта, и, вместо того чтобы отправлять вызов в сеть, сервер переходит в ожидание поступления входящих запросов. Клиентское приложение должно выполнить активное открытие (*active open*), отправив серверному приложению синхронизирующий порядковый номер (*SYN*), идентифицирующий соединение. Клиентское приложение может использовать динамический номер порта в качестве локального порта. Сервер должен отправить клиенту подтверждение (*ACK*) вместе с порядковым номером (*SYN*) сервера. В свою очередь клиент отвечает *АСК*, и соединение устанавливается. После этого может начаться процесс отправки и получения сообщений. При получении сообщения в ответ всегда отправляется сообщение *АСК*. Если до получения *АСК* отправителем истекает тайм-аут, сообщение помещается в очередь на повторную передачу.

3 ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММЫ

Для реализации задачи было выбрано приложение обмена текстовыми сообщениями между клиентами по локальной сети с использованием сокетов. Согласно формулировке задачи, были спроектированы следующие функции программы:

– выбор логина;

– отправление сообщения и отображение сообщений в чате;

– сообщение о новом участнике чата.

Приложение состоит из двух частей. Первая отвечает за работу сервера, к которому подключаются пользователи. Вторая часть отвечает за доступ клиента к чату. В серверной части обрабатываются два процесса: ожидание нового подключения и ожидание сообщения от пользователя. Новые пользователи для установления соединения отправляют на сервер свой логин, который сохраняется вместе с полученным сокетом.

Клиентская часть помимо функций обращения к серверу создаёт пользовательский интерфейс (рисунок 3.1).

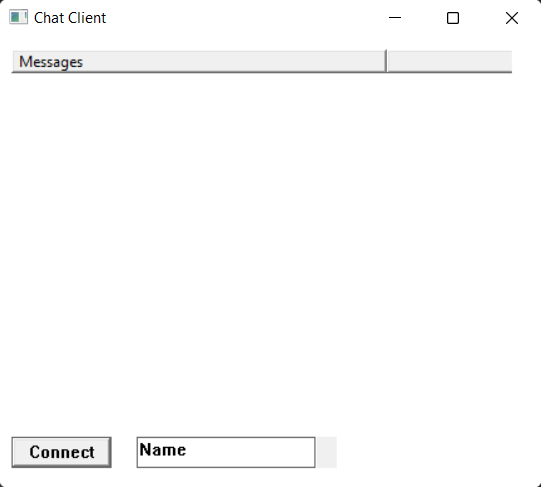


Рисунок 3.1 – Окно приложения

Окно приложения содержит список сообщений, кнопку для соединения и поле ввода. Список сообщений изначально пуст. Чтобы подключиться, необходимо ввести логин и нажать на кнопку. После этого кнопка изменится на *Send* и по её нажатию будут отправляться сообщения в чат (рисунок 3.2).



Рисунок 3.2 – Кнопка отправки сообщений

После подключения также появится сообщение у всех участников чата о входе нового человека (рисунок 3.3).

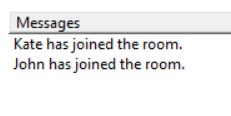


Рисунок 3.3 – Сообщение о входе в чат

При выходе из чата будет отображено сообщение для всех оставшихся пользователей (рисунок 3.5).

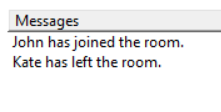


Рисунок 3.4 – Сообщение о выходе из чата

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Взаимодействие между процессами [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/ipc/interprocess-communications.

[2] Сведения о буфере обмена [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/dataxchg/about-the-clipboard.

[3] Сведения о динамическом обмене данными [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/dataxchg/about-dynamic-data-exchange.

[4] Что такое сокет и зачем он нужен [Электронный ресурс]. – Режим доступа: https://thecode.media/socket/.

[5] Протоколы *TCP* и *UDP* [Электронный ресурс]. – Режим доступа: https://professorweb.ru/my/csharp/web/level1/1\_4.php.

ПРИЛОЖЕНИЕ А

**Исходный код программы**

**Файл LAB78SERVER.cpp**

#include <iostream>

#include <string>

#include <vector>

#include <winsock2.h>

#pragma comment(lib, "ws2\_32.lib")

#include <ws2tcpip.h>

class ChatRoom {

public:

void join(SOCKET clientSocket, const std::string& username)

{

clients.push\_back({ clientSocket, username });

broadcast(username + " has joined the room.\n");

}

void leave(SOCKET clientSocket, const std::string& username)

{

auto it = std::find\_if(clients.begin(), clients.end(), [clientSocket](const ClientInfo& info) {

return info.socket == clientSocket;

});

if (it != clients.end()) {

broadcast(username + " has left the room.\n");

clients.erase(it);

}

}

void broadcast(const std::string& message, SOCKET excludeClient = INVALID\_SOCKET)

{

for (const auto& client : clients) {

if (client.socket != excludeClient) {

send(client.socket, message.c\_str(), message.size(), 0); // sends data on a connected socket socket descriptor, buffer, length of the buffer, flags

}

}

}

private:

struct ClientInfo {

SOCKET socket;

std::string username;

};

std::vector<ClientInfo> clients;

};

ChatRoom globalRoom;

DWORD WINAPI ClientThread(LPVOID lpParam)

{

SOCKET clientSocket = reinterpret\_cast<SOCKET>(lpParam);

char buffer[1024];

int recvResult;

recvResult = recv(clientSocket, buffer, sizeof(buffer), 0); // returns the number of bytes received

//socket descriptor, pointer to buffer, len of buffer, flags

std::string username;

if (recvResult > 0) {

buffer[recvResult] = '\0';

username = std::string(buffer);

globalRoom.join(clientSocket, username);

}

while (true) {

recvResult = recv(clientSocket, buffer, sizeof(buffer), 0);

if (recvResult > 0) {

buffer[recvResult] = '\0';

std::string message(buffer);

globalRoom.broadcast(username + ": " + message);

} else if (recvResult == 0) {

globalRoom.leave(clientSocket, username);

closesocket(clientSocket);

std::cout << "Client disconnected\n";

break;

} else {

std::cerr << "Recv failed\n";

globalRoom.leave(clientSocket, username);

closesocket(clientSocket);

break;

}

}

return 0;

}

int main()

{

WSADATA wsaData; // information about the Windows Sockets implementation

if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) { // initiates use of the Winsock DLL by a process like a load library

//version and wsadata

std::cerr << "WSAStartup failed\n";

return -1;

}

SOCKET serverSocket = socket(AF\_INET, SOCK\_STREAM, 0);

// af\_inet - ipv4 (adress family specification), sock\_stream - for stream socket

if (serverSocket == INVALID\_SOCKET) {

std::cerr << "Error creating socket\n";

WSACleanup();

return -1;

}

sockaddr\_in serverAddr; //specifies a transport address and port for the AF\_INET address family

serverAddr.sin\_family = AF\_INET;

serverAddr.sin\_port = htons(8888); // converts port to big-endian

serverAddr.sin\_addr.s\_addr = INADDR\_ANY;

if (bind(serverSocket, reinterpret\_cast<sockaddr\*>(&serverAddr), sizeof(serverAddr)) == SOCKET\_ERROR) { // bind local address with a socket

// socket descriptor, pointer to sockaddr, length of sockaddr

std::cerr << "Bind failed\n";

closesocket(serverSocket);

WSACleanup();

return -1;

}

if (listen(serverSocket, SOMAXCONN) == SOCKET\_ERROR) { //somaxconn - max

std::cerr << "Listen failed\n";

closesocket(serverSocket);

WSACleanup();

return -1;

}

std::cout << "Server is listening on port 8888...\n";

while (true) {

SOCKET clientSocket = accept(serverSocket, nullptr, nullptr);

if (clientSocket == INVALID\_SOCKET) {

std::cerr << "Accept failed\n";

break;

}

std::cout << "Client connected\n";

HANDLE threadHandle = CreateThread(nullptr, 0, ClientThread, reinterpret\_cast<LPVOID>(clientSocket), 0, nullptr);

//pointer to a SECURITY\_ATTRIBUTES (null - cannot be inherited), size (0 - default value is used), function, pointer to params to pass, flags (0 - runs immidiately)

//pointer to a variable that recieves thread identifier

if (threadHandle == nullptr) {

std::cerr << "Failed to create thread\n";

closesocket(clientSocket);

break;

}

}

closesocket(serverSocket);

WSACleanup();

return 0;

}

**Файл LAB78CLIENT.cpp**

#include <WS2tcpip.h>

#include <windows.h>

#include <string>

#include <commctrl.h>

#pragma comment(lib, "ws2\_32.lib")

#pragma comment(lib, "comctl32.lib")

HWND g\_hListView;

SOCKET g\_clientSocket;

HANDLE g\_hThread;

HWND hwndEdit;

HWND hwndBtn;

void SendMessageToServer(const std::wstring& message)

{

std::string s(message.begin(), message.end());

send(g\_clientSocket, s.c\_str(), s.size(), 0);

}

void AddMessageToListView(const std::string& message)

{

auto normalMessage = std::wstring(message.begin(), message.end());

LVITEM lvItem;

lvItem.mask = LVIF\_TEXT;

lvItem.iItem = ListView\_GetItemCount(g\_hListView);

lvItem.iSubItem = 0;

lvItem.pszText = (LPWSTR)normalMessage.c\_str();

ListView\_InsertItem(g\_hListView, &lvItem);

}

DWORD WINAPI MessageListener(LPVOID lpParam)

{

char buffer[1024];

while (true) {

int recvResult = recv(g\_clientSocket, buffer, sizeof(buffer), 0);

if (recvResult > 0) {

buffer[recvResult] = '\0';

std::string message(buffer);

AddMessageToListView(message);

} else if (recvResult == 0) {

closesocket(g\_clientSocket);

MessageBox(nullptr, L"Server disconnected", L"Info", MB\_OK | MB\_ICONINFORMATION);

break;

} else {

MessageBox(nullptr, L"Error receiving message", L"Error", MB\_OK | MB\_ICONERROR);

closesocket(g\_clientSocket);

break;

}

}

}

bool isConnected;

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

{

switch (uMsg) {

case WM\_CREATE: {

INITCOMMONCONTROLSEX icex;

icex.dwSize = sizeof(INITCOMMONCONTROLSEX);

icex.dwICC = ICC\_LISTVIEW\_CLASSES;

InitCommonControlsEx(&icex);

g\_hListView = CreateWindowEx(

0, WC\_LISTVIEW, L"", WS\_CHILD | WS\_VISIBLE | LVS\_REPORT,

10, 10, 400, 300, hwnd, nullptr, nullptr, nullptr);

//exstyle, className, window name, style, x, y, width, height, parent, id to manipulate, instance of assiciated module, pointer to passed params

LVCOLUMN lvCol;

lvCol.mask = LVCF\_TEXT | LVCF\_WIDTH | LVCF\_SUBITEM;

lvCol.cx = 300;

lvCol.pszText = (LPWSTR)L"Messages";

lvCol.iSubItem = 0;

ListView\_InsertColumn(g\_hListView, 0, &lvCol);

hwndBtn= CreateWindow(

L"BUTTON", L"Connect",

WS\_TABSTOP | WS\_VISIBLE | WS\_CHILD | BS\_DEFPUSHBUTTON,

10, 320, 80, 25,

hwnd, (HMENU)1, nullptr, nullptr);

hwndEdit = CreateWindow(

L"Edit", L"Name",

WS\_CHILD | WS\_VISIBLE | WS\_VSCROLL | ES\_LEFT | WS\_BORDER,

110, 320, 160, 25, hwnd, (HMENU)2, // edit control ID

(HINSTANCE)GetWindowLongPtr(hwnd, GWLP\_HINSTANCE), NULL); // application instance

break;

}

case WM\_COMMAND: {

if (LOWORD(wParam) == 1) {

if (!isConnected) {

g\_clientSocket = socket(AF\_INET, SOCK\_STREAM, 0);

if (g\_clientSocket == INVALID\_SOCKET) {

MessageBox(hwnd, L"Error creating socket", L"Error", MB\_OK | MB\_ICONERROR);

return 0;

}

sockaddr\_in serverAddr;

serverAddr.sin\_family = AF\_INET;

serverAddr.sin\_port = htons(8888);

inet\_pton(AF\_INET, "127.0.0.1", &serverAddr.sin\_addr); // ip to binary

//family, address, buffer

if (connect(g\_clientSocket, reinterpret\_cast<sockaddr\*>(&serverAddr), sizeof(serverAddr)) == SOCKET\_ERROR) {

MessageBox(hwnd, L"Error connecting to server", L"Error", MB\_OK | MB\_ICONERROR);

closesocket(g\_clientSocket);

return 0;

}

WCHAR buff[1024];

GetWindowText(hwndEdit, buff, 1024);

SendMessageToServer(buff);

isConnected = true;

SetWindowText(hwndEdit, L"");

g\_hThread = CreateThread(nullptr, 0, MessageListener, nullptr, 0, nullptr);

SetWindowText(hwndBtn, L"Send");

} else {

WCHAR buff[1024];

GetWindowText(hwndEdit, buff, 1024);

SendMessageToServer(buff);

SetWindowText(hwndEdit, L"");

}

}

break;

}

case WM\_DESTROY: {

PostQuitMessage(0);

break;

}

default: {

return DefWindowProc(hwnd, uMsg, wParam, lParam);

}

}

return 0;

}

int WINAPI wWinMain(\_In\_ HINSTANCE hInstance, \_In\_opt\_ HINSTANCE hPrevInstance, \_In\_ LPWSTR lpCmdLine, \_In\_ int nCmdShow)

{

UNREFERENCED\_PARAMETER(hPrevInstance);

UNREFERENCED\_PARAMETER(lpCmdLine);

WSADATA wsaData;

if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {

return -1;

}

WNDCLASS wc = {};

wc.lpfnWndProc = WindowProc;

wc.hInstance = hInstance;

wc.lpszClassName = L"ChatClientClass";

if (RegisterClass(&wc)) {

HWND hwnd = CreateWindowEx(

0, L"ChatClientClass", L"Chat Client",

WS\_OVERLAPPEDWINDOW, CW\_USEDEFAULT, CW\_USEDEFAULT, 450, 400,

nullptr, nullptr, hInstance, nullptr);

if (hwnd) {

ShowWindow(hwnd, nCmdShow);

MSG msg;

while (GetMessage(&msg, nullptr, 0, 0)) {

TranslateMessage(&msg);

DispatchMessage(&msg);

}

}

}

return 0;

}