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 №5

на тему

**ЭЛЕМЕНТЫ СЕТЕВОГО ПРОГРАММИРОВАНИЯ**

Выполнил: студент гр.253505 Павлович В.Ю.

Проверил: ассистент кафедры информатики Гриценко Н.Ю.

Минск 2024

**СОДЕРЖАНИЕ**

[1 Постановка задачи 3](#_Toc184406693)

[2 Краткие теоретические сведения 4](#_Toc184406694)

[3 Описание работы програмы 5](#_Toc184406695)

[3.1 Серверное приложение 5](#_Toc184406696)

[3.2 Клиентское приложение 5](#_Toc184406697)

[Заключение 7](#_Toc184406698)

[Список использованных источников 8](#_Toc184406699)

[Приложение А (обязательное) Исходный код программы 9](#_Toc184406700)

# **1 ПОСТАНОВКА ЗАДАЧИ**

Целью лабораторной работы является изучение основ построения и функционирования сетей, стеков протоколов, программных интерфейсов. Организация взаимодействия между процессами в сетях.

В качестве задачи необходимо разработать упрощенный чат для нескольких пользователей с использованием сетевых сокетов.

Разработанное приложение должно удовлетворять следующим требованиям:

– архитектура: централизованная (выделенный процесс-сервер и процессы клиенты) или децентрализованная (процессы-клиенты с «серверными» функциями);

– сервер: создание сокета для приема соединений или отдельных сообщений; прием и временное хранение сообщений; передача сообщений адресно одному или нескольким клиентам; поддержание списка актуальных клиентов;

– клиент: обнаружение сервера и соединение с ним; ввод пользовательских сообщений и передача их серверу либо напрямую соответствующему клиенту; прием и отображение сообщений.

2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Сокет – это конечная точка связи – объект, через который приложение сокетов Windows отправляет или получает пакеты данных в сети. Сокет имеет тип и связан с запущенным процессом, и он может иметь имя. В настоящее время сокеты обычно обмениваются данными только с другими сокетами в том же "домене связи", который использует Набор протоколов Интернета.

Оба типа сокетов являются двунаправленными; они представляют собой потоки данных, которые можно обмениваться данными в обоих направлениях одновременно (полно дуплексный).

Доступны два типа сокета:

– потоковые сокеты – обеспечивают поток данных без границ записей: поток байтов. Гарантируется, что данные будут доставлены, правильно упорядочены и не продублированы;

– датаграммные сокеты – поддерживают ориентированный на записи поток данных, который не гарантирует доставку и может не соответствовать порядку отправки или дублироваться.

"Упорядоченность" означает, что пакеты доставляются в порядке отправки. "Непродублированный" означает, что вы получаете определенный пакет только один раз.[1]

3 ОПИСАНИЕ РАБОТЫ ПРОГРАММЫ

В этом разделе будут рассмотрены основные функции программы, разработанной программы. Отдельно будут рассмотрены клиентское и серверное приложения.

## **3.1 Серверное приложение**

Серверное приложение при запуске создает сокет и начинает прослушивать подключения к этому сокету. При подключении клиента в отдельном потоке запускается функция обработки данного клиента, в которой задается идентификатор конкретного клиента и происходит обработка всех полученный от него сообщений (отправка сообщения либо всем клиентам, либо только конкретному при указании его идентификатора). Также сервер способен корректно обрабатывать отключение клиента от сокета. Пользовательский интерфейс серверного приложения представлен на рисунке 3.1.

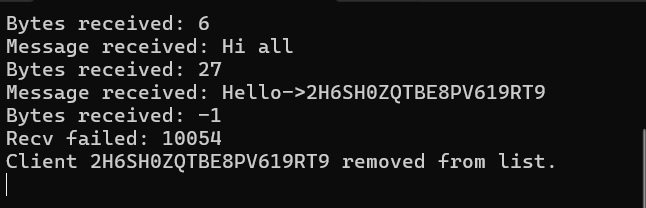


Рисунок 3.1 – Пользовательский интерфейс серверного приложения

## **3.2 Клиентское приложение**

При запуске клиентское приложение подключается к сокету на сервере, после чего пользователь может отправлять и получать сообщения. Для отправки сообщения конкретному пользователю, после текста сообщения нужно указать ->name, где name – идентификатор пользователя, выданный сервером. При получении сообщения пользователь видит идентификатор отправителя и текст сообщения, а также (private), если сообщение было отправлено конкретно ему. Пользовательский интерфейс клиентского приложения представлен на рисунке 3.2.

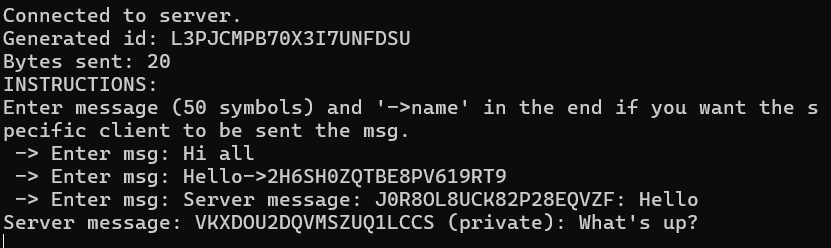


Рисунок 3.2 – Пользовательский интерфейс клиентского приложения

Таким образом, разработанные приложения представляю собой простейший клиент-серверный чат на сокетах, позволяющий отправлять сообщения либо всем подключенным клиентам, либо какому-то конкретному.

# **ЗАКЛЮЧЕНИЕ**

В ходе выполнения лабораторной работы была реализован простейший клиент-серверный чат. Для обмена сообщениями использовалось подключение клиентов к серверу по сокетам.

Данное приложение демонстрирует возможности сокетов для реализации клиент-серверных приложений. Из-за дуплексной природы сокетов, каждый клиент в реальном времени может получать сообщения от других пользователей без отправки каких-либо дополнительный запросов на сервер.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Сокеты Windows. Фон – Win32 apps [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/ru-ru/cpp/mfc/windows-sockets-background>?

view=msvc-170.

ПРИЛОЖЕНИЕ А

(обязательное)

**Исходный код программы**

Листинг А1 – Исходный код клиентского приложения

#define WIN32\_LEAN\_AND\_MEAN

#include <windows.h>

#include <winsock2.h>

#include <ws2tcpip.h>

#include <stdlib.h>

#include <stdio.h>

#include <ctime>

#include <iostream>

#include <thread>

#include <string>

#include <atomic>

#include <stdlib.h>

// Need to link with Ws2\_32.lib, Mswsock.lib, and Advapi32.lib

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

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

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

#define DEFAULT\_BUFLEN 512

#define DEFAULT\_PORT "27015"

std::atomic<bool> isRunning(true); // flag for ending sending

SOCKET ConnectSocket = INVALID\_SOCKET;

std::string static GenRandom(const int len) {

srand(time(0));

static const char alphanum[] = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ";

std::string tmp\_s;

tmp\_s.reserve(len);

for (int i = 0; i < len; ++i) {

tmp\_s += alphanum[rand() % (sizeof(alphanum) - 1)];

}

return tmp\_s;

}

void static ReceiveMessages() {

int iResult;

char recvbuf[DEFAULT\_BUFLEN];

int recvbuflen = DEFAULT\_BUFLEN;

while (isRunning) {

iResult = recv(ConnectSocket, recvbuf, recvbuflen, 0);

if (iResult > 0) {

recvbuf[iResult] = '\0';

printf("Server message: %s\n", recvbuf);

}

else if (iResult == 0) {

printf("Connection closed by server.\n");

break;

}

else {

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

break;

}

}

}

void static SendMessages() {

int iResult;

char sendbuf[DEFAULT\_BUFLEN];

while (true) {

printf(" -> Enter msg: ");

fgets(sendbuf, sizeof(sendbuf), stdin);

size\_t len = strlen(sendbuf);

if (len > 0 && sendbuf[len - 1] == '\n') {

sendbuf[len - 1] = '\0';

}

if (strlen(sendbuf) == 0) {

sendbuf[0] = '\0';

iResult = send(ConnectSocket, sendbuf, 1, 0);

if (iResult == SOCKET\_ERROR) {

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

isRunning = false;

break;

}

isRunning = false;

break;

}

else {

iResult = send(ConnectSocket, sendbuf, (int)strlen(sendbuf), 0);

if (iResult == SOCKET\_ERROR) {

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

break;

}

}

}

}

int main()

{

WSADATA wsaData;

int iResult;

// Initialize Winsock

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

if (iResult != 0) {

printf("WSAStartup failed with error: %d\n", iResult);

return 1;

}

struct addrinfo\* result = NULL, \* ptr = NULL, hints;

ZeroMemory(&hints, sizeof(hints));

hints.ai\_family = AF\_INET;

hints.ai\_socktype = SOCK\_STREAM;

hints.ai\_protocol = IPPROTO\_TCP;

// Resolve the server address and port

iResult = getaddrinfo("127.0.0.1", DEFAULT\_PORT, &hints, &result);

if (iResult != 0) {

printf("getaddrinfo failed: %d\n", iResult);

WSACleanup();

return 1;

}

// Create a SOCKET for connecting to server

ConnectSocket = socket(result->ai\_family, result->ai\_socktype, result->ai\_protocol);

// Check for errors to ensure that the socket is a valid socket

if (ConnectSocket == INVALID\_SOCKET) {

printf("Error at socket(): %ld\n", WSAGetLastError());

freeaddrinfo(result);

WSACleanup();

return 1;

}

// Connect to server.

iResult = connect(ConnectSocket, result->ai\_addr, (int)result->ai\_addrlen);

if (iResult == SOCKET\_ERROR) {

closesocket(ConnectSocket);

ConnectSocket = INVALID\_SOCKET;

}

// Free the resources returned by getaddrinfo and print an error message

freeaddrinfo(result);

if (ConnectSocket == INVALID\_SOCKET) {

printf("Unable to connect to server!\n");

WSACleanup();

return 1;

}

printf("Connected to server.\n");

std::string sendbufID = GenRandom(20);

printf("Generated id: %s\n", sendbufID.c\_str());

iResult = send(ConnectSocket, sendbufID.c\_str(), 20, 0);

if (iResult == SOCKET\_ERROR) {

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

closesocket(ConnectSocket);

WSACleanup();

return 1;

}

printf("Bytes sent: %d\n", iResult);

printf("INSTRUCTIONS:\nEnter message (50 symbols) and '->name' in the end if you want the specific client to be sent the msg.\n");

std::thread recvThread(ReceiveMessages);

std::thread sendThread(SendMessages);

sendThread.join();

recvThread.join();

// Cleanup

closesocket(ConnectSocket);

WSACleanup();

return 0;

}

Листинг А2 – Исходный код серверного приложения

#include <winsock2.h>

#include <ws2tcpip.h>

#include <stdio.h>

#include <iostream>

#include <thread>

#include <vector>

#include <mutex>

#include <string>

#include <unordered\_map>

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

#define DEFAULT\_PORT "27015"

#define DEFAULT\_BUFLEN 512

// Struct to store client information

struct ClientInfo {

SOCKET socket;

char name[21];

std::vector<std::string> msgs;

// Конструктор для удобной инициализации

ClientInfo(SOCKET s, const char\* n)

: socket(s), msgs() {

strncpy\_s(name, n, sizeof(name) - 1);

name[sizeof(name) - 1] = '\0'; // Ensure null termination

}

};

struct addrinfo\* result = NULL, \* ptr = NULL, hints;

std::vector<ClientInfo> clientSockets; // List of connected client sockets

std::mutex clientMutex; // Mutex to synchronize access to the clientSockets vector

// Function to broadcast a message to all clients

void static BroadcastMessage(const std::string& message, SOCKET senderSocket) {

std::lock\_guard<std::mutex> lock(clientMutex);

for (const auto& client : clientSockets) {

if (client.socket != senderSocket) { // Don't send the message back to the sender

send(client.socket, message.c\_str(), int(message.size()), 0);

}

}

}

void HandleClient(SOCKET ClientSocket) {

char recvbuf[DEFAULT\_BUFLEN];

int iResult, iSendResult;

int recvbuflen = DEFAULT\_BUFLEN;

iResult = recv(ClientSocket, recvbuf, DEFAULT\_BUFLEN, 0);

if (iResult <= 0) {

closesocket(ClientSocket);

return;

}

recvbuf[iResult] = '\0'; // Null-terminate the received name

char clientName[21];

// Safely copy the received name to clientName

strncpy\_s(clientName, recvbuf, sizeof(clientName) - 1);

clientName[sizeof(clientName) - 1] = '\0'; // Ensure null termination

// Add client to the list

{

std::lock\_guard<std::mutex> lock(clientMutex);

clientSockets.push\_back(ClientInfo(ClientSocket, clientName));

}

std::cout << "Client connected: " << recvbuf << std::endl;

// Receive until the peer shuts down the connection

do {

iResult = recv(ClientSocket, recvbuf, recvbuflen, 0); // iResult - number of received bytes

printf("Bytes received: %d\n", iResult);

if (iResult > 0) {

recvbuf[iResult] = '\0';

std::string message = recvbuf;

// Check for empty msg

if (strlen(recvbuf) == 0) {

printf("Client %s disconnected.\n", clientName);

break;

}

else {

printf("Message received: %s\n", recvbuf);

}

// Check if the message contains "->" for directed messaging

size\_t arrowPos = message.find("->");

if (arrowPos != std::string::npos) {

std::string actualMessage = message.substr(0, arrowPos); // Extract the actual message

std::string targetID = message.substr(arrowPos + 2); // Extract the target client ID

// Save the message in the client's message list

{

std::lock\_guard<std::mutex> lock(clientMutex);

for (auto& client : clientSockets) {

if (client.socket == ClientSocket) {

client.msgs.push\_back(actualMessage); // Add the message to the client's message list

break;

}

}

}

// Find the target client

bool clientFound = false;

{

std::lock\_guard<std::mutex> lock(clientMutex);

for (const auto& client : clientSockets) {

if (client.name == targetID) {

clientFound = true;

std::string name(clientName);

std::string directedMessage = name + " (private): " + actualMessage;

send(client.socket, directedMessage.c\_str(), static\_cast<int>(directedMessage.size()), 0);

break;

}

}

}

if (!clientFound) {

std::string errorMsg = "Error: Client with ID '" + targetID + "' not found.\n";

send(ClientSocket, errorMsg.c\_str(), static\_cast<int>(errorMsg.size()), 0);

}

}

else {

// If no "->" found, treat as a broadcast

std::string name(clientName);

std::string broadcastMessage = name + ": " + message;

BroadcastMessage(broadcastMessage, ClientSocket);

}

}

else {

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

break;

}

} while (iResult > 0);

// Remove the client from the list after disconnection

{

std::lock\_guard<std::mutex> lock(clientMutex);

clientSockets.erase(std::remove\_if(clientSockets.begin(), clientSockets.end(),

[ClientSocket](const ClientInfo& client) {

return client.socket == ClientSocket;

}),

clientSockets.end());

}

printf("Client %s removed from list.\n", clientName);

closesocket(ClientSocket);

return;

}

int main() {

WSADATA wsaData; // The WSADATA structure contains information about the Windows Sockets implementation.

int iResult;

// Initialize Winsock (MAKEWORD - initializing version, )

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

if (iResult != 0) {

printf("WSAStartup failed: %d\n", iResult);

return 1;

}

// clear hints structure

ZeroMemory(&hints, sizeof(hints));

hints.ai\_family = AF\_INET; // specify the IPv4 address family

hints.ai\_socktype = SOCK\_STREAM; // specify a stream socket

hints.ai\_protocol = IPPROTO\_TCP; // specify the TCP protocol

hints.ai\_flags = AI\_PASSIVE; // caller intends to use the returned socket address structure in a call to the bind function

// Resolve the local address and port to be used by the server

iResult = getaddrinfo("0.0.0.0", DEFAULT\_PORT, &hints, &result);

if (iResult != 0) {

printf("getaddrinfo failed: %d\n", iResult);

WSACleanup();

return 1;

}

// Create a SOCKET

SOCKET ListenSocket = socket(result->ai\_family, result->ai\_socktype, result->ai\_protocol);

if (ListenSocket == INVALID\_SOCKET) {

printf("Error at socket(): %ld\n", WSAGetLastError());

freeaddrinfo(result);

WSACleanup();

return 1;

}

// Bind exsisting socket to address

iResult = bind(ListenSocket, result->ai\_addr, (int)result->ai\_addrlen);

if (iResult == SOCKET\_ERROR) {

printf("bind failed with error: %d\n", WSAGetLastError());

freeaddrinfo(result); // sock adrr no longer needed

closesocket(ListenSocket);

WSACleanup();

return 1;

}

// Define listening mode for socket

// SOMAXCONN - maximum reasonable number of pending connections in the queue

if (listen(ListenSocket, SOMAXCONN) == SOCKET\_ERROR) {

printf("Listen failed with error: %ld\n", WSAGetLastError());

closesocket(ListenSocket);

WSACleanup();

return 1;

}

printf("Socket is listening\n");

while (true) {

SOCKET ClientSocket = INVALID\_SOCKET;

ClientSocket = accept(ListenSocket, NULL, NULL);

if (ClientSocket == INVALID\_SOCKET) {

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

continue; // Don't terminate the server; just skip this iteration

}

printf("New client connected.\n");

// Add the new client

std::thread(HandleClient, ClientSocket).detach();

}

closesocket(ListenSocket);

WSACleanup();

return 0;

}