|  |  |  |
| --- | --- | --- |
|  | | |
| Федеральное государственное бюджетное  образовательное учреждение высшего образования «Новосибирский государственный технический университет» | | |
|  | | |
| Кафедра теоретической и прикладной информатики | | |
| Лабораторная работа № 3 | | |
| по дисциплине «Компьютерные сети» | | |
| **разработка приложения интерактивной переписки** | | |
|  | | |
|  | Бригада 4 | буров евгений |
| Группа ПМ-23 | Гайченко максим |
|  |  |
| Преподаватели | Кобылянский валерий григорьевич |
|  | сивак мария алексеевна |
|  |  |
|  |  |
|  |  |
| Новосибирск, 2025 | | |

1. Цель работы

Изучить основные принципы разработки многопользовательских приложений, построенных на основе технологии клиент-сервер с использованием стека протоколов TCP/IP.

1. Ход работы

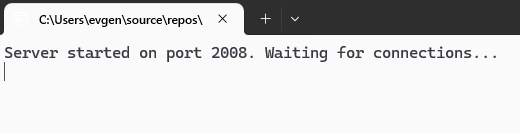
Реализовано чат-приложение. Сервер поддерживает соединение с несколькими клиентами одновременно. Обмен сообщениями происходит через сервер: при получении сообщения он выводит его в свою консоль и также пересылает его всем прочим участникам переписки. При подключении нового пользователя сервер оповещает действующих клиентов и посылает им его имя.

Для осуществления обмена сообщениями с несколькими клиентами используются потоки. Для регулируемого доступа к ограниченным ресурсам, например, консоли и массиву, содержащему информацию о клиентах, используются мутексы.

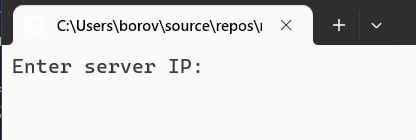
Для предотвращения конфликтов, возникающих при одновременном наборе сообщения клиентом и поступления сообщения с сервера

1. Тестирование программы

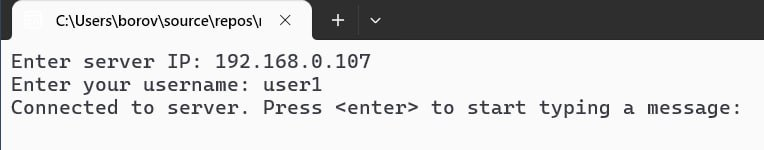
*Начало работы сервера:*

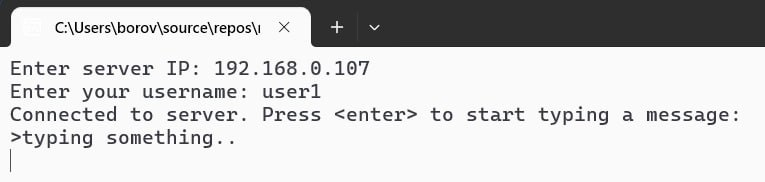


*Запуск клиента:*

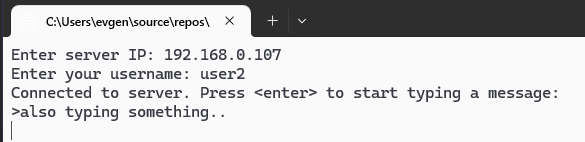


*Подключение и ввод имени:*

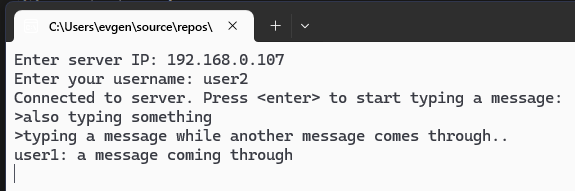


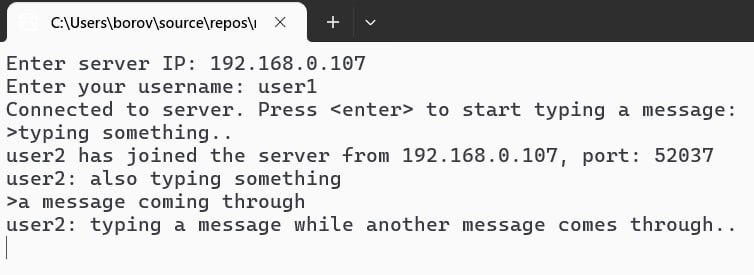


*Подключение нового клиента:*

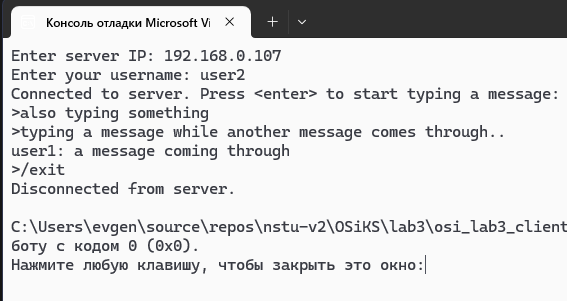


*При переходе пользователя в режим набора сообщения доступ к выводу консоли  
закрывается до тех пор, пока сообщение не будет отправлено:*





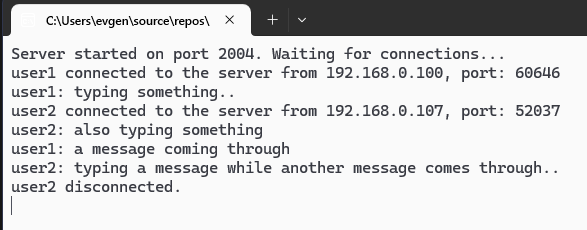
*Отключение пользователя:*



*Оповещение:*



*Журнал сервера:*



1. Текст программ

server.cpp:

#define \_WINSOCK\_DEPRECATED\_NO\_WARNINGS

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

#include <winsock2.h>

#include <ws2tcpip.h>

#include <iostream>

#include <string>

#include <mutex>

#include <thread>

#include <vector>

#include <algorithm>

const int USERNAME\_LENGTH = 256;

const int MESSAGE\_BUFFER\_SIZE = 1024;

struct ClientInfo {

SOCKET client\_socket;

std::string name;

std::string ip\_address;

unsigned short port;

};

SOCKET server\_socket;

std::vector<ClientInfo> clients;

std::mutex mtx;

void broadcast(const std::string& message, SOCKET sender\_socket = INVALID\_SOCKET, bool include\_sender = false) {

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

for (const auto& client : clients) {

// Skip if this is the sender and we shouldn't include them

if (client.client\_socket == sender\_socket && !include\_sender) {

continue;

}

// Skip clients that haven't set their name yet (except for sender)

if (client.name.empty() && client.client\_socket != sender\_socket) {

continue;

}

send(client.client\_socket, message.c\_str(), message.size() + 1, 0);

}

}

void handle\_client(ClientInfo client\_info) {

char buffer[MESSAGE\_BUFFER\_SIZE];

// Get username first

int bytes\_received = recv(client\_info.client\_socket, buffer, USERNAME\_LENGTH, 0);

if (bytes\_received <= 0) {

closesocket(client\_info.client\_socket);

return;

}

buffer[bytes\_received] = '\0';

client\_info.name = buffer;

{

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

std::cout << client\_info.name << " connected to the server from "

<< client\_info.ip\_address << ", port: " << client\_info.port << "\n";

// Update the client's name in the vector

auto it = std::find\_if(clients.begin(), clients.end(),

[&](const ClientInfo& c) { return c.client\_socket == client\_info.client\_socket; });

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

it->name = client\_info.name;

}

}

// Notify all clients about new connection (except the new client)

std::string join\_message = client\_info.name + " has joined the server from " +

client\_info.ip\_address + ", port: " + std::to\_string(client\_info.port);

broadcast(join\_message, client\_info.client\_socket);

while (true) {

bytes\_received = recv(client\_info.client\_socket, buffer, MESSAGE\_BUFFER\_SIZE, 0);

if (bytes\_received <= 0) {

std::string disconnected\_name;

{

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

auto it = std::find\_if(clients.begin(), clients.end(),

[&](const ClientInfo& c) { return c.client\_socket == client\_info.client\_socket; });

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

disconnected\_name = it->name;

std::cout << disconnected\_name << " disconnected.\n";

clients.erase(it);

}

}

if (!disconnected\_name.empty()) {

// Notify all clients about disconnection

broadcast(disconnected\_name + " has left the chat.");

}

closesocket(client\_info.client\_socket);

break;

}

buffer[bytes\_received] = '\0';

{

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

std::cout << client\_info.name << ": " << buffer << "\n";

}

// Broadcast the message to all other clients

broadcast(client\_info.name + ": " + buffer, client\_info.client\_socket);

}

}

int main() {

WSADATA wsa\_data;

if (WSAStartup(MAKEWORD(2, 2), &wsa\_data) != 0) {

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

return 1;

}

server\_socket = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);

if (server\_socket == INVALID\_SOCKET) {

std::cerr << "Socket creation failed.\n";

WSACleanup();

return 1;

}

SOCKADDR\_IN server\_addr;

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_port = htons(2004);

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

if (bind(server\_socket, (sockaddr\*)&server\_addr, sizeof(server\_addr)) == SOCKET\_ERROR) {

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

closesocket(server\_socket);

WSACleanup();

return 1;

}

if (listen(server\_socket, SOMAXCONN) == SOCKET\_ERROR) {

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

closesocket(server\_socket);

WSACleanup();

return 1;

}

std::cout << "Server started on port 2004. Waiting for connections...\n";

while (true) {

SOCKADDR\_IN client\_addr;

int client\_addr\_len = sizeof(client\_addr);

SOCKET client\_socket = accept(server\_socket, (sockaddr\*)&client\_addr, &client\_addr\_len);

if (client\_socket == INVALID\_SOCKET) {

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

continue;

}

// Get client IP and port

char client\_ip[INET\_ADDRSTRLEN];

inet\_ntop(AF\_INET, &client\_addr.sin\_addr, client\_ip, INET\_ADDRSTRLEN);

unsigned short client\_port = ntohs(client\_addr.sin\_port);

ClientInfo new\_client = { client\_socket, "", client\_ip, client\_port };

{

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

clients.push\_back(new\_client);

}

std::thread(handle\_client, new\_client).detach();

}

closesocket(server\_socket);

WSACleanup();

return 0;

}

client.cpp:

#define \_WINSOCK\_DEPRECATED\_NO\_WARNINGS

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

#include <winsock2.h>

#include <ws2tcpip.h>

#include <iostream>

#include <string>

#include <thread>

#include <queue>

#include <mutex>

#include <conio.h>

const int MESSAGE\_BUFFER\_SIZE = 1024;

std::queue<std::string> queued\_received\_messages;

std::mutex mtx;

void receive\_messages(SOCKET client\_socket) {

char buffer[MESSAGE\_BUFFER\_SIZE];

while (true) {

int bytes\_received = recv(client\_socket, buffer, MESSAGE\_BUFFER\_SIZE, 0);

if (bytes\_received <= 0) {

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

std::cout << "Disconnected from server.\n";

break;

}

buffer[bytes\_received] = '\0';

{

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

std::cout << buffer << "\n";

}

}

}

int main() {

WSADATA wsa\_data;

if (WSAStartup(MAKEWORD(2, 2), &wsa\_data) != 0) {

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

return 1;

}

SOCKET client\_socket = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);

if (client\_socket == INVALID\_SOCKET) {

std::cerr << "Socket creation failed.\n";

WSACleanup();

return 1;

}

std::string server\_ip;

std::cout << "Enter server IP: ";

std::getline(std::cin, server\_ip);

SOCKADDR\_IN server\_addr;

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_port = htons(2004);

inet\_pton(AF\_INET, server\_ip.c\_str(), &server\_addr.sin\_addr);

if (connect(client\_socket, (sockaddr\*)&server\_addr, sizeof(server\_addr)) == SOCKET\_ERROR) {

std::cerr << "Failed to connect to server.\n";

closesocket(client\_socket);

WSACleanup();

return 1;

}

std::string name;

std::cout << "Enter your username: ";

std::getline(std::cin, name);

send(client\_socket, name.c\_str(), name.size() + 1, 0);

std::cout << "Connected to server. Press <enter> to start typing a message:\n";

// Start a thread to receive messages

std::thread receiver(receive\_messages, client\_socket);

receiver.detach();

while (true) {

std::string message;

while ((\_getch()) != '\r');

{

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

std::cout << ">";

std::getline(std::cin, message);

}

if (message == "/exit") {

break;

}

if (send(client\_socket, message.c\_str(), message.size() + 1, 0) == SOCKET\_ERROR) {

std::cerr << "Failed to send message.\n";

break;

}

}

closesocket(client\_socket);

WSACleanup();

return 0;

}