|  |  |  |
| --- | --- | --- |
|  | | |
| Федеральное государственное бюджетное  образовательное учреждение высшего образования  FPMI_ngtu_neti_rgb_polya«Новосибирский государственный технический университет» | | |
|  | | |
| Кафедра теоретической и прикладной информатики | | |
| Лабораторная работа № 3 | | |
| по дисциплине «Операционные системы и компьютерные сети» | | |
| Разработка приложения интерактивной переписки | | |
|  | | |
|  | Бригада | ВесЕлый Денис |
| №1 | Ворончук Илья |
|  |  |
| Группа | ПМИ-32 |
|  |  |
|  |  |
| Преподаватели | кобылянский в.г. |
|  | сивак м.а. |
| Новосибирск, 2025 | | |

# Цель работы

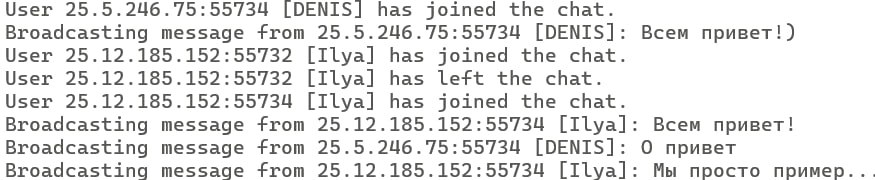
Изучить основные принципы разработки многопользовательских приложений, построенных на основе технологии клиент-сервер с использованием стека протоколов TCP/IP.

# Описание хода выполнения работы

Для выполнения задания были разработаны два приложения: многопоточный сервер и клиент для интерактивной переписки (чата).

**Серверная часть** реализована с использованием многопоточности: при подключении нового клиента сервер создает для него отдельный поток обработки. Это позволяет принимать сообщения от одного клиента и рассылать их всем остальным подключенным пользователям в режиме реального времени. При подключении клиент отправляет свой никнейм. Если никнейм не указан, сервер автоматически присваивает имя "User #N". Сервер ведет лог подключений, отключений и пересылаемых сообщений в своей консоли.

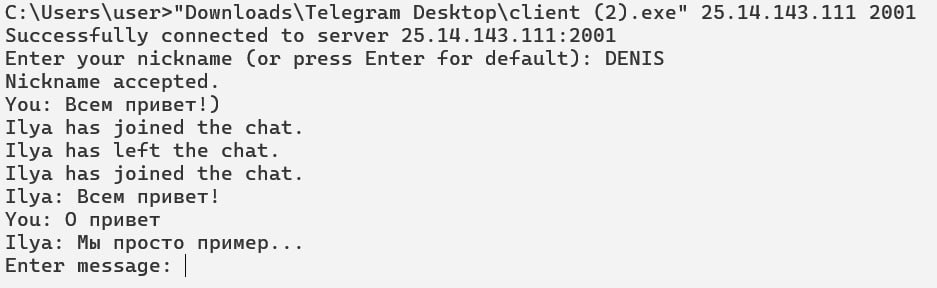
**Пример работы приложения со стороны сервера:**

*Запуск сервера происходил с указанием номера порта (2001) протокола TCP.*

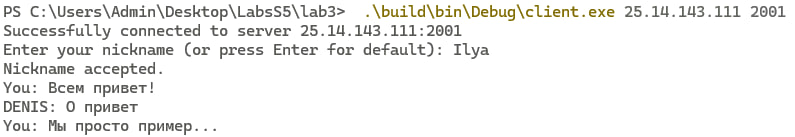
**Клиентская часть** также является многопоточной для обеспечения одновременной отправки и приема сообщений. Основной поток ожидает ввод пользователя, а фоновый поток постоянно принимает данные от сервера и асинхронно выводит их на экран.

**Пример работы со стороны двух клиентов (Клиент 1 – DENIS, Клиент 2 – Ilya):**

Окно Клиента 1 (DENIS)



Окно Клиента 2 (Ilya)



Анализ результатов:

Как видно из примеров, система работает корректно.

1. Сервер одновременно обслуживает двух клиентов.
2. При подключении (и отключении) нового пользователя (Ilya), все остальные участники (DENIS) получают системное уведомление.
3. Сообщение, отправленное одним клиентом (DENIS: О привет), принимается сервером и немедленно рассылается всем остальным участникам чата (Ilya).
4. Реализована проверка уникальности никнеймов и улучшенный пользовательский интерфейс клиента.

# Вывод

В результате выполнения лабораторной работы были успешно освоены фундаментальные принципы построения многопользовательских клиент-серверных приложений.

На практике была реализована многопоточная схема сетевого взаимодействия, включающая следующие ключевые этапы:

1. Создание многопоточного сервера, способного одновременно обслуживать множество TCP-соединений, выделяя для каждого клиента отдельный поток выполнения.
2. Организация синхронизации доступа к общим ресурсам (списку клиентов) с помощью мьютексов для предотвращения состояния гонки.
3. Разработка многопоточного клиента, который может одновременно отправлять (в основном потоке) и получать (в фоновом потоке) сообщения, обеспечивая интерактивность чата.
4. Реализация протокола прикладного уровня для обмена служебной информацией (никнеймы, системные сообщения) между клиентом и сервером.

Эта работа позволила закрепить теоретические знания о работе сокетов и многопоточности, а также получить практические навыки, необходимые для разработки сложных сетевых приложений.

# Приложение

### **common.h**

#pragma once

#ifdef \_WIN32

#define \_WINSOCK\_DEPRECATED\_NO\_WARNINGS

#include <winsock2.h>

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

#else

#include <arpa/inet.h>

#include <errno.h>

#include <netinet/in.h>

#include <sys/socket.h>

#include <sys/types.h>

#include <unistd.h>

#define SOCKET int

#define INVALID\_SOCKET -1

#define SOCKET\_ERROR -1

#define closesocket(s) close(s)

#endif

### **server.cpp**

#include "common.h"

#ifdef \_WIN32

#include <ws2tcpip.h>

#endif

#include <algorithm>

#include <atomic>

#include <cstring> // Для strerror в Linux/macOS

#include <iostream>

#include <map>

#include <mutex>

#include <set> // Добавлено для хранения свободных номеров

#include <string>

#include <thread>

#include <vector>

const int BUFFER\_SIZE = 4096;

// ------ Глобальные переменные для управления клиентами ------

// Карта для хранения сокетов клиентов и их адресов

std::map<SOCKET, std::string> clients;

// Мьютекс для синхронизации доступа к карте клиентов

std::mutex clients\_mutex;

// Атомарный счетчик для уникальных User #N

std::atomic<int> user\_counter = 0;

// Новая структура для хранения освободившихся номеров

std::set<int> released\_user\_numbers;

// ------ Функции ------

#include <cctype>

#include <string>

/\*\*

\* @brief Проверяет, что строка начинается с префикса "User #" и после '#' идёт положительное целое

\* число.

\* @param nickname Никнейм пользователя.

\* @return Если начинается, то true, иначе false.

\*/

bool starts\_with\_user\_and\_positive\_int(const std::string &nickname) {

const std::string prefix = "User #";

if (nickname.rfind(prefix, 0) != 0) return false; // не начинается с "User #"

size\_t pos = prefix.size();

if (pos >= nickname.size()) return false; // нет символов после '#'

// всё оставшееся — цифры и образуют положительное число (без ведущих знаков плюс)

for (size\_t i = pos; i < nickname.size(); ++i)

if (!std::isdigit((unsigned char)nickname[i])) return false;

// исключаем "0" и строки, состоящие только из нулей

for (size\_t i = pos; i < nickname.size(); ++i)

if (nickname[i] != '0') return true;

return false;

}

/\*\*

\* @brief Обрезает пробельные символы (пробел, \r, \n, \t) с начала и конца

\* строки.

\* @param str Строка для обработки.

\* @return Очищенная строка.

\*/

std::string trim\_string(const std::string &str) {

const std::string WHITESPACE = " \n\r\t";

size\_t first = str.find\_first\_not\_of(WHITESPACE);

if (std::string::npos == first) {

return "";

}

size\_t last = str.find\_last\_not\_of(WHITESPACE);

return str.substr(first, (last - first + 1));

}

/\*\*

\* @brief Проверяет, занят ли уже указанный никнейм.

\* @param nickname Никнейм для проверки.

\* @return true, если ник занят, иначе false.

\*/

// Важно: эта функция НЕ должна блокировать мьютекс сама,

// так как она будет вызываться из кода, где мьютекс уже захвачен.

bool is\_nickname\_taken\_unsafe(const std::string &nickname) {

return std::any\_of(clients.begin(), clients.end(),

[&nickname](const auto &pair) { return pair.second == nickname; });

}

/\*\*

\* @brief Рассылает сообщение всем подключенным клиентам, кроме отправителя.

\* @param message Сообщение для рассылки.

\* @param sender\_socket Сокет клиента-отправителя.

\*/

void broadcast\_message(const std::string &message, SOCKET sender\_socket) {

std::lock\_guard<std::mutex> lock(clients\_mutex);

for (auto const &[socket, nickname] : clients) {

if (socket != sender\_socket) {

send(socket, message.c\_str(), message.length(), 0);

}

}

}

/\*\*

\* @brief Функция для обработки сообщений от одного клиента. Выполняется в

\* отдельном потоке.

\* @param client\_socket Сокет подключенного клиента.

\* @param client\_ip\_port IP-адрес и порт клиента для логов сервера.

\*/

void handle\_client(SOCKET client\_socket, std::string client\_ip\_port) {

char buffer[BUFFER\_SIZE];

std::string nickname;

// Цикл получения уникального никнейма от клиента

while (true) {

int bytes\_received = recv(client\_socket, buffer, BUFFER\_SIZE - 1, 0);

if (bytes\_received <= 0) {

// Клиент отключился, не выбрав ник

std::cout << "Client " << client\_ip\_port << " disconnected before setting a nickname."

<< std::endl;

closesocket(client\_socket);

return;

}

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

std::string proposed\_nickname = trim\_string(std::string(buffer));

// Блокируем мьютекс на все время выбора ника, чтобы избежать "гонок"

std::lock\_guard<std::mutex> lock(clients\_mutex);

if (proposed\_nickname.empty()) {

// Логика генерации имени при пустом вводе

if (!released\_user\_numbers.empty()) {

// Переиспользуем освобожденный номер

int reused\_number = \*released\_user\_numbers.begin();

nickname = "User #" + std::to\_string(reused\_number);

released\_user\_numbers.erase(released\_user\_numbers.begin());

} else {

// Генерируем новый номер, проверяя на случай ручного ввода

do {

nickname = "User #" + std::to\_string(++user\_counter);

} while (is\_nickname\_taken\_unsafe(nickname));

}

send(client\_socket, "#NICK\_OK#", 10, 0);

break;

}

// Логика проверки ника, введенного пользователем

if (is\_nickname\_taken\_unsafe(proposed\_nickname)) {

// Отправляем сигнал, что ник занят - мьютекс будет освобожден

// и клиент сможет отправить новый вариант

send(client\_socket, "#NICK\_TAKEN#", 13, 0);

} else {

// Ник уникален

nickname = proposed\_nickname;

send(client\_socket, "#NICK\_OK#", 10, 0);

break;

}

}

// Сохраняем никнейм клиента

{

std::lock\_guard<std::mutex> lock(clients\_mutex);

clients[client\_socket] = nickname;

}

// Оповещение всех о новом пользователе

std::string server\_join\_msg =

"User " + client\_ip\_port + " [" + nickname + "] has joined the chat.";

std::cout << server\_join\_msg << std::endl;

std::string client\_join\_msg = nickname + " has joined the chat.";

broadcast\_message(client\_join\_msg, client\_socket);

// Цикл приема сообщений от клиента

int bytes\_received;

while ((bytes\_received = recv(client\_socket, buffer, BUFFER\_SIZE - 1, 0)) > 0) {

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

bool data\_truncated = (bytes\_received == BUFFER\_SIZE - 1);

std::string trimmed\_msg = trim\_string(std::string(buffer));

if (trimmed\_msg.empty()) continue;

std::string received\_message = nickname + ": " + trimmed\_msg;

if (data\_truncated) {

received\_message += " [message was too long and has been truncated]";

}

std::cout << "Broadcasting message from " << client\_ip\_port << " [" << nickname

<< "]: " << trimmed\_msg << std::endl;

broadcast\_message(received\_message, client\_socket);

}

// Обработка отключения клиента

std::cout << "User " << client\_ip\_port << " [" + nickname + "] has left the chat." << std::endl;

std::string disconnect\_message = nickname + " has left the chat.";

{

std::lock\_guard<std::mutex> lock(clients\_mutex);

clients.erase(client\_socket);

// Если ник был системным, возвращаем его номер в пул

if (starts\_with\_user\_and\_positive\_int(nickname)) {

try {

int number = std::stoi(nickname.substr(6));

released\_user\_numbers.insert(number);

std::cout << "System nickname ID " << number << " has been released." << std::endl;

} catch (...) {

// Игнорируем ошибки парсинга на всякий случай

}

}

}

broadcast\_message(disconnect\_message, client\_socket);

closesocket(client\_socket);

}

/\*\*

\* @brief Инициализирует Winsock.

\* @return true в случае успеха, false в случае ошибки.

\*/

bool initialize\_sockets() {

#ifdef \_WIN32

WSADATA wsa\_data;

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

std::cerr << "WSAStartup failed." << std::endl;

return false;

}

#endif

std::cout << "Sockets initialized." << std::endl;

return true;

}

void cleanup\_sockets() {

#ifdef \_WIN32

WSACleanup();

#endif

}

/\*\*

\* @brief Создает и настраивает слушающий сокет.

\* @param port Порт для прослушивания.

\* @return Дескриптор слушающего сокета или INVALID\_SOCKET в случае ошибки.

\*/

SOCKET create\_listen\_socket(int port) {

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

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

std::cerr << "Socket creation failed." << std::endl;

return INVALID\_SOCKET;

}

sockaddr\_in server\_addr;

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

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

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

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

std::cerr << "Bind failed." << std::endl;

closesocket(listen\_socket);

return INVALID\_SOCKET;

}

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

std::cerr << "Listen failed." << std::endl;

closesocket(listen\_socket);

return INVALID\_SOCKET;

}

std::cout << "Server started on port " << port << ". Listening for connections..." << std::endl;

return listen\_socket;

}

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

if (argc != 2) {

std::cerr << "Usage: " << argv[0] << " <port>" << std::endl;

return 1;

}

int port = atoi(argv[1]);

if (!initialize\_sockets()) {

return 1;

}

SOCKET listen\_socket = create\_listen\_socket(port);

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

cleanup\_sockets();

return 1;

}

while (true) {

sockaddr\_in client\_addr;

#ifdef \_WIN32

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

#else

socklen\_t client\_addr\_size = sizeof(client\_addr);

#endif

SOCKET client\_socket = accept(listen\_socket, (sockaddr \*)&client\_addr, &client\_addr\_size);

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

std::cerr << "Accept failed." << std::endl;

continue;

}

char client\_ip[INET\_ADDRSTRLEN];

// В поле client\_addr.sin\_addr - сетевой порядок битов (Big-Endian - старший

// байт находится по младшему адресу) Используем функцию inet\_ntop, а не

// inet\_ntoa, так как она позволяет контролировать размер буфера

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

std::string client\_ip\_port =

std::string(client\_ip) + ":" + std::to\_string(ntohs(client\_addr.sin\_port));

// Передаем в поток сокет и IP:порт для логгирования

std::thread client\_thread(handle\_client, client\_socket, client\_ip\_port);

// Используем detach для независимой параллельной обработки клиентов

client\_thread.detach();

}

closesocket(listen\_socket);

cleanup\_sockets();

return 0;

}

### **client.cpp**

#include <atomic>

#include <cstring> // For strerror on Linux/macOS

#include <iostream>

#include <string>

#include <thread>

#include "common.h"

const int CLEANER\_COUNT = 80;

const int BUFFER\_SIZE = 4096;

// Атомарный флаг для потокобезопасного завершения

std::atomic<bool> should\_exit = false;

#ifdef \_WIN32

#include <windows.h>

void enable\_virtual\_terminal\_processing() {

HANDLE hOut = GetStdHandle(STD\_OUTPUT\_HANDLE);

if (hOut == INVALID\_HANDLE\_VALUE) return;

DWORD dwMode = 0;

if (!GetConsoleMode(hOut, &dwMode)) return;

dwMode |= ENABLE\_VIRTUAL\_TERMINAL\_PROCESSING;

SetConsoleMode(hOut, dwMode);

}

#endif

/\*\*

\* @brief Очищает текущую строку в консоли и перерисовывает приглашение к вводу.

\*/

void redraw\_prompt() {

std::cout << "\r\033[K";

std::cout << "Enter message: ";

std::cout.flush();

}

/\*\*

\* @brief Функция для приема сообщений от сервера. Выполняется в отдельном

\* потоке.

\* @param client\_socket Сокет для связи с сервером.

\*/

void receive\_messages(SOCKET client\_socket) {

char buffer[BUFFER\_SIZE];

int bytes\_received;

while (!should\_exit) {

bytes\_received = recv(client\_socket, buffer, BUFFER\_SIZE - 1, 0);

if (bytes\_received > 0) {

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

// Перемещаем курсор в начало строки, очищаем ее и выводим сообщение

std::cout << "\r\033[K";

std::cout << buffer << std::endl;

redraw\_prompt();

} else {

// Если recv вернул 0 или ошибку, значит соединение потеряно

if (!should\_exit) {

std::cout << "\rConnection to server lost. Press Enter to exit." << std::endl;

}

should\_exit = true;

break;

}

}

}

/\*\*

\* @brief Инициализирует Winsock.

\* @return true в случае успеха, false в случае ошибки.

\*/

bool initialize\_sockets() {

#ifdef \_WIN32

WSADATA wsa\_data;

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

std::cerr << "WSAStartup failed." << std::endl;

return false;

}

#endif

return true;

}

void cleanup\_sockets() {

#ifdef \_WIN32

WSACleanup();

#endif

}

/\*\*

\* @brief Создает сокет и подключается к серверу.

\* @param server\_ip IP-адрес сервера.

\* @param port Порт сервера.

\* @return Дескриптор сокета или INVALID\_SOCKET в случае ошибки.

\*/

SOCKET connect\_to\_server(const char\* server\_ip, int port) {

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

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

std::cerr << "Socket creation failed." << std::endl;

return INVALID\_SOCKET;

}

sockaddr\_in server\_addr;

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

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

server\_addr.sin\_addr.s\_addr = inet\_addr(server\_ip);

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

std::cerr << "Connection to server failed." << std::endl;

closesocket(client\_socket);

return INVALID\_SOCKET;

}

std::cout << "Successfully connected to server " << server\_ip << ":" << port << std::endl;

return client\_socket;

}

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

#ifdef \_WIN32

enable\_virtual\_terminal\_processing(); // Включаем поддержку ANSI кодов в

// Windows

#endif

if (argc != 3) {

std::cerr << "Usage: " << argv[0] << " <server\_ip> <port>" << std::endl;

return 1;

}

if (!initialize\_sockets()) {

return 1;

}

SOCKET client\_socket = connect\_to\_server(argv[1], atoi(argv[2]));

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

cleanup\_sockets();

return 1;

}

// Цикл запроса и проверки никнейма

char response\_buffer[BUFFER\_SIZE];

while (!should\_exit) {

std::string nickname;

std::cout << "Enter your nickname (or press Enter for default): ";

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

if (nickname == "/exit") {

should\_exit = true; // Сигнализируем потоку-приемнику, что мы выходим

break;

}

std::string nick\_to\_send = nickname.empty() ? " " : nickname;

if (send(client\_socket, nick\_to\_send.c\_str(), nick\_to\_send.length(), 0) == SOCKET\_ERROR) {

std::cerr << "Send failed." << std::endl;

should\_exit = true; // Ошибка отправки, выходим

break;

}

int bytes = recv(client\_socket, response\_buffer, BUFFER\_SIZE - 1, 0);

if (bytes <= 0) {

should\_exit = true; // Сервер закрыл соединение, выходим

break;

}

response\_buffer[bytes] = '\0';

if (strcmp(response\_buffer, "#NICK\_OK#") == 0) {

std::cout << "Nickname accepted." << std::endl;

break; // Ник принят, выходим из цикла

} else if (strcmp(response\_buffer, "#NICK\_TAKEN#") == 0) {

std::cout << "This nickname is already taken. Please choose another one." << std::endl;

} else {

std::cout << "Unknown response from server. Disconnecting." << std::endl;

should\_exit = true;

break;

}

}

if (should\_exit) {

std::cout << "Could not set nickname or exiting. Connection closed." << std::endl;

closesocket(client\_socket);

cleanup\_sockets();

return 1;

}

// Запускаем поток для приема сообщений

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

// Основной цикл для отправки сообщений

std::string message;

while (!should\_exit) {

redraw\_prompt();

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

// Если should\_exit стал true, пока мы ждали getline, выходим

if (should\_exit) {

break;

}

// Перемещаем курсор на одну строку вверх и очищаем ее

std::cout << "\033[A\033[K"; // Вверх на строку, очистить ее

std::cout << "You: " << message << std::endl;

if (message == "/exit") {

should\_exit = true; // Сигнализируем потоку-приемнику, что мы выходим

break;

}

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

std::cerr << "Send failed." << std::endl;

should\_exit = true;

}

}

#ifdef \_WIN32

shutdown(client\_socket, SD\_SEND);

#else

shutdown(client\_socket, SHUT\_WR);

#endif

// Главный поток и receiver\_thread должны завершиться вместе, поэтому

// дожидаемся его

if (receiver\_thread.joinable()) {

receiver\_thread.join();

}

closesocket(client\_socket);

cleanup\_sockets();

std::cout << "Connection closed. Press Enter to exit." << std::endl;

return 0;

}