|  |  |  |
| --- | --- | --- |
|  | | |
| Федеральное государственное бюджетное  образовательное учреждение высшего образования «Новосибирский государственный технический университет» | | |
|  | | |
| Кафедра прикладной математики | | |
| Лабораторная работа № 3 | | |
| по дисциплине «Операционные системы и компьютерные сети » | | |
| **Разработка приложения интерактивной переписки** | | |
|  | | |
|  |  |  |
| Группа ПМ-21 | порсин данил |
| Место для ввода текста. |  |
|  |  |
|  |  |
|  |  |
| Преподаватели | сивак м.а. |
|  |  |
| Новосибирск, 2025 | | |

1. **Цель работы**

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

1. **Задание**

С помощью API-интерфейса реализовать простой чат**.** Каждая бригада должна написать chat-сервер и chat-клиента. Сервер должен поддерживать соединение сразу от нескольких клиентов. Обмен между клиентами осуществляется через сервер. При получении сообщения от какого-либо клиента, сервер дублирует его на своем экране и оповещает всех подсоединенных клиентов, отправляя каждому из них данное сообщение. При подсоединении нового клиента к chat-серверу, сервер оповещает каждого клиента о новом пользователе, посылая им его IP-адрес, номер порта и имя. При отключении также необходимо сформировать оповещение.

Тестирование сервера провести в режиме удаленного доступа при одновременном подключении не менее двух клиентов, загруженных на разных компьютерах. Также необходимо проверить корректность работы клиента, осуществляющего ввод данных, при одновременной отправке данных другим клиентом.

1. **Анализ задачи**
2. Инициализация сервера (функция server())

* Создание сокета:
* Инициализируется Winsock (WSAStartup).
* Создаётся TCP-сокет (AF\_INET, SOCK\_STREAM).
* Привязка сокета:
* Указывается IP-адрес (192.168.0.100) и порт (2011).
* Сокет привязывается к адресу (bind).
* Ожидание подключений:
* Сервер переходит в режим прослушивания (listen).
* При подключении клиента создается новый сокет (accept).

1. Работа сервера с клиентом

* Обработка подключившегося клиента:
* Клиент добавляется в список клиентов (clients).
* Запускается поток listen\_client для обработки сообщений от клиента.
* Получение сообщений от клиента:
* В потоке listen\_client вызывается recv() для приёма данных.
* Если сообщение начинается с '/', вызывается process\_command():
* /register <имя> — регистрация клиента (запоминается имя).
* /scroll\_up <смещение> — отправка сообщения выше по логу.
* /scroll\_down <смещение> — отправка сообщения ниже по логу.
* /exit — удаление клиента и уведомление остальных.
* Иное — сообщение "Unknown command".
* Если сообщение не начинается с '/':
* Добавляется в лог messages\_log.
* Отправляется всем клиентам через send\_message\_to\_all().

1. Работа клиента

* Подключение к серверу:
* Пользователь вводит IP и порт.
* Инициализируется Winsock, создаётся сокет.
* Выполняется подключение к серверу (connect).
* Отправляется команда /register <имя>.
* Отправка сообщений:
* В основном потоке вызывается client\_input().
* При нажатии Enter сообщение отправляется серверу (send).
  + При нажатии стрелок вверх/вниз — отправляются команды /scroll\_up или /scroll\_down.
* Получение сообщений:
* В отдельном потоке запускается receiving\_server\_messages().
* Полученные сообщения отображаются на экране.
* Если включена прокрутка — клиент загружает нужные сообщения из истории.

1. Завершение сеанса

* При вводе /exit:
* Клиент завершает соединение и выходит из программы.
* Сервер удаляет клиента из списка и рассылает уведомление остальным.

1. **Используемый функции**

|  |  |
| --- | --- |
| Имя функции | Назначение |
| WSAStartup | Инициализация библиотеки Winsock для сетевого взаимодействия |
| socket | Создание сокета |
| connect | Подключение к серверному сокету |
| closesocket | Закрытие сокета |
| WSACleanup | Освобождение ресурсов, занятых Winsock |
| inet\_pton | Преобразование IP-адреса из текстового вида в числовой (для sockaddr) |
| htons | Преобразование порта из host byte order в network byte order |
| send | Отправка данных через сокет |
| recv | Получение данных через сокет |
| listen | Ожидание входящих соединений |
| accept | Принятие входящего соединения |
| bind | Привязка сокета к IP-адресу и порту |
| setsockopt | Установка параметров сокета |
| GetStdHandle | Получение дескриптора консоли (для ввода/вывода) |
| ReadConsoleInput | Чтение событий ввода с консоли |
| SetConsoleCursorPosition | Установка позиции курсора в окне консоли |
| GetConsoleScreenBufferInfo | Получение информации о буфере экрана консоли |
| SetConsoleOutputCP | Установка кодовой страницы вывода консоли |
| SetConsoleCP | Установка кодовой страницы ввода консоли |
| setlocale | Установка локали (для корректного отображения Юникода и локализации) |
| system("cls") | Очистка экрана консоли (Windows) |
| std::getline | Чтение строки из потока |
| std::cin, std::cout | Стандартный ввод и вывод |
| std::thread, detach | Создание и отсоединение потока выполнения |
| std::lock\_guard / mutex | Синхронизация потоков (мьютексы) |
| std::wstring\_convert | Конвертация между UTF-8 и UTF-16 |
| std::stoi, std::to\_string | Преобразование строк в числа и обратно |
| std::stringstream / istringstream | Разбор строк на команды и параметры |
| std::fstream | Работа с файлами (лог сообщений) |
| inet\_ntop | Преобразование IP-адреса из числового в строковый формат |

1. **Текст программы**

Серверная часть

#include <ws2tcpip.h>

#include <iostream>

#include <sstream>

#include <thread>

#include <vector>

#include <mutex>

#include <unordered\_map>

#include <ranges>

#include <fstream>

#include <iomanip>

#include <codecvt>

#include <Windows.h>

#pragma comment(lib, "ws2\_32.lib")

enum {

CLIENT\_EXIT = 0,

CLIENT\_REGISTER,

CLIENT\_SCROLL\_UP,

CLIENT\_SCROLL\_DOWN,

SUCCESS

};

struct CLIENT {

SOCKET socket;

std::string name;

std::string ip;

std::string port;

int window\_scroll\_offset = 0;

CLIENT(SOCKET socket, const char\* ip, const std::string& port) : socket(socket), ip(ip), port(port) {};

};

struct SERVER {

SOCKET socket;

SOCKADDR\_IN sockaddr;

std::unordered\_map<SOCKET,CLIENT> clients;

std::mutex message\_rec\_mtx;

std::fstream messages\_log;

SERVER() {

messages\_log.open("messages\_log.txt", std::ios::in | std::ios::out);

}

int message\_count = 0;

int display\_message\_count = 20;

};

SERVER server;

std::string utf16\_to\_utf8(const std::wstring& utf16\_string) {

int size\_needed = WideCharToMultiByte(CP\_UTF8, 0, utf16\_string.c\_str(), -1, nullptr, 0, nullptr, nullptr);

std::string utf8\_string(size\_needed - 1, 0);

WideCharToMultiByte(CP\_UTF8, 0, utf16\_string.c\_str(), -1, &utf8\_string[0], size\_needed, nullptr, nullptr);

return utf8\_string;

}

std::wstring utf8\_to\_utf16(const std::string& utf8\_string) {

int size\_needed = MultiByteToWideChar(CP\_UTF8, 0, utf8\_string.c\_str(), -1, nullptr, 0);

std::wstring utf16\_string(size\_needed - 1, 0);

MultiByteToWideChar(CP\_UTF8, 0, utf8\_string.c\_str(), -1, &utf16\_string[0], size\_needed);

return utf16\_string;

}

void server\_echo(const std::wstring& message, const CLIENT& client) {

std::cout << "Message from " << client.ip << ":" << client.port << " - ";

std::wcout << message << std::endl;

}

int send\_message\_to\_sender(std::string message, CLIENT& client) {

int ret\_val = send(client.socket, message.data(), message.size(), 0);

std::cout << "message to send " << message << std::endl;

if (ret\_val == SOCKET\_ERROR) {

std::cout << "Sending failed with error: " << WSAGetLastError();

closesocket(client.socket);

return 0;

}

}

int send\_message\_to\_all(std::string message, CLIENT\* except\_client = nullptr) {

int ret\_val;

for (const auto& client : std::views::values(server.clients)) {

if (&client != except\_client) {

if (client.window\_scroll\_offset == 0) {

ret\_val = send(client.socket, message.data(), message.size(), 0);

if (ret\_val == SOCKET\_ERROR) {

std::cout << "Sending failed with error: " << WSAGetLastError();

closesocket(client.socket);

return 0;

}

}

}

}

}

std::string get\_message\_from\_log(int aim\_message\_index) {

server.messages\_log.seekg(0, std::ios::beg);

std::string message;

for (int message\_index = 0; message\_index <= aim\_message\_index; message\_index++)

std::getline(server.messages\_log, message);

return message;

}

int process\_command\_message(const std::string& command\_message, CLIENT& client) {

std::string command;

std::istringstream command\_stream(command\_message);

command\_stream >> command;

if (command == "/register") {

command\_stream >> client.name;

std::cout << "rigister" << std::endl;

std::string chat\_message = "Client " + client.name + "(" + client.ip + ":" + client.port + ") has joined the chat";

server.messages\_log << chat\_message << std::endl;;

send\_message\_to\_all(chat\_message);

server.message\_count++;

return CLIENT\_REGISTER;

}

if (command == "/scroll\_up") {

int offset;

command\_stream >> offset;

int aim\_message\_index = server.message\_count - server.display\_message\_count - offset;

std::cout << "aim\_message\_index " << aim\_message\_index << std::endl;

if (aim\_message\_index >= 0) {

send\_message\_to\_sender("/scroll\_up " + get\_message\_from\_log(aim\_message\_index), client);

}

return CLIENT\_SCROLL\_UP;

}

if (command == "/scroll\_down") {

int offset;

command\_stream >> offset;

int aim\_message\_index = server.message\_count - offset;

if (aim\_message\_index < server.message\_count) {

send\_message\_to\_sender("/scroll\_down " + get\_message\_from\_log(aim\_message\_index), client);

}

return CLIENT\_SCROLL\_DOWN;

}

if (command == "/exit") {

std::string chat\_message = "Client " + client.name + " (" + client.ip + ":" + client.port + ") has left the chat";

server.messages\_log << chat\_message << std::endl;

send\_message\_to\_all(chat\_message, &client);

server.message\_count++;

std::cout << "Client " << client.ip << ":" << client.port << " has disconnected" << std::endl;

server.clients.erase(client.socket);

closesocket(client.socket);

return CLIENT\_EXIT;

}

send\_message\_to\_sender("Unknown command", client);

return 0;

}

int listen\_client(CLIENT& client) {

setlocale(LC\_ALL, "rus");

int ret\_val;

std::string message\_utf8;

std::wstring message\_utf16;

while (true) {

message\_utf8.clear();

message\_utf8.resize(1024);

std::cout << "Receiving data from client" << client.ip << std::endl;

ret\_val = recv(client.socket, (char\*)message\_utf8.data(), message\_utf8.size(), 0);

server.message\_rec\_mtx.lock();

if (ret\_val == SOCKET\_ERROR) {

process\_command\_message("/exit", client);

server.message\_rec\_mtx.unlock();

return 0;

}

message\_utf8.resize(ret\_val);

message\_utf16 = utf8\_to\_utf16(message\_utf8);

server\_echo(message\_utf16, client);

if (message\_utf8.data()[0] == '/') {

int ret\_val = process\_command\_message(message\_utf8, client);

if (ret\_val == CLIENT\_EXIT) {

server.message\_rec\_mtx.unlock();

return SUCCESS;

}

}

else {

message\_utf8 = client.name + ": " + message\_utf8;

server.messages\_log << message\_utf8 << std::endl;

send\_message\_to\_all(message\_utf8);

server.message\_count++;

}

server\_echo(message\_utf16, client);

server.message\_rec\_mtx.unlock();

}

}

int process\_server\_socket() {

int clients\_count = 15;

int ret\_val;

while (true) {

std::cout << "Listening to the server socket" << std::endl;

ret\_val = listen(server.socket, clients\_count);

if (ret\_val == SOCKET\_ERROR) {

std::cout << "Listening to the server socket failed with error: " << WSAGetLastError();

closesocket(server.socket);

WSACleanup();

return 0;

}

std::cout << "Accepting the client socket" << std::endl;

SOCKET client\_socket;

SOCKADDR\_IN client\_sockaddr;

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

client\_socket = accept(server.socket, (sockaddr\*)&client\_sockaddr, &client\_addr\_size);

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

std::cout << "Accepting failed with error: " << WSAGetLastError();

closesocket(server.socket);

WSACleanup();

return 0;

}

char client\_ip[INET\_ADDRSTRLEN];

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

CLIENT client(client\_socket, client\_ip, std::to\_string(ntohs(client\_sockaddr.sin\_port)));

auto [pair\_socket\_client, emplaced] = server.clients.emplace(client.socket, std::move(client));

std::thread client\_socket\_listener(listen\_client, std::ref(pair\_socket\_client->second));

client\_socket\_listener.detach();

}

}

int start\_server() {

PCSTR server\_ip = "192.168.0.104";

int port = 2011;

int clients\_count = 15;

int recv\_size = 2;

int send\_size = 1;

int ret\_val;

WSAData wsa\_data;

if (WSAStartup(MAKEWORD(2, 2, ), &wsa\_data) != NO\_ERROR) {

std::cout << "WSAStartup failed" << std::endl;

return 0;

}

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

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

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

std::cout << "Creating the server socket" << std::endl;

server.socket= socket(AF\_INET, SOCK\_STREAM, IPPROTO\_IP);

if (server.socket == INVALID\_SOCKET) {

std::cout << "Socket creation failed with error: " << WSAGetLastError();

WSACleanup();

return 0;

}

std::cout << "Binding the server socket" << std::endl;

int option = 1;

setsockopt(server.socket, SOL\_SOCKET, SO\_REUSEADDR, (char\*)&option, sizeof(option));

ret\_val = bind(server.socket, (sockaddr\*)&server.sockaddr, sizeof(server.sockaddr));

if (ret\_val == SOCKET\_ERROR) {

std::cout << "Binding server socket failed with error: " << WSAGetLastError();

closesocket(server.socket);

WSACleanup();

return 0;

}

process\_server\_socket();

}

int main() {

setlocale(LC\_ALL, "rus");

start\_server();

}

Клиентская часть

#include <winsock2.h>

#include <ws2tcpip.h>

#include <locale.h>

#include <string>

#include <sstream>

#include <iostream>

#include <thread>

#include <deque>

#include <queue>

#include <sstream>

#include <mutex>

#include <Windows.h>

#pragma comment(lib, "ws2\_32.lib")

enum {

ERR,

SUCCESS,

EXIT

};

struct CHAT {

CONSOLE\_SCREEN\_BUFFER\_INFO csbi;

std::deque<std::wstring> client\_messages\_buf;

int window\_scroll\_offset = 0;

SOCKET client\_socket;

std::mutex make\_chat\_window\_mutex;

};

CHAT chat;

int connect\_server() {

std::string server\_ip;

int server\_port;

std::cout << "Enter chat server IP and port space-separated" << std::endl;

std::cin >> server\_ip >> server\_port;

int ret\_val;

WSAData wsa\_data;

if (WSAStartup(MAKEWORD(2, 2), &wsa\_data) != NO\_ERROR) {

std::cout << "WSAStartup failed" << std::endl;

return ERR;

}

chat.client\_socket = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_IP);

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

std::cout << "Socket creation failed with error: " << WSAGetLastError() << "\n\n";

WSACleanup();

return ERR;

}

SOCKADDR\_IN server\_sockaddr{};

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

server\_sockaddr.sin\_port = htons(server\_port);

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

ret\_val = connect(chat.client\_socket, (sockaddr\*)&server\_sockaddr, sizeof(server\_sockaddr));

if (ret\_val == SOCKET\_ERROR) {

std::cout << "Connecting failed with error: " << WSAGetLastError() << "\n\n";

closesocket(chat.client\_socket);

WSACleanup();

return ERR;

}

return SUCCESS;

}

int send\_message(const std::string& message) {

int ret\_val;

ret\_val = send(chat.client\_socket, message.data(), message.size(), 0);

if (ret\_val == SOCKET\_ERROR) {

std::cout << "Sending failed with error: " << WSAGetLastError() << "\n\n";

closesocket(chat.client\_socket);

WSACleanup();

return ERR;

}

return SUCCESS;

}

int register\_client() {

std::string name;

std::cout << "Enter your name" << std::endl;

std::cin >> name;

std::cin.ignore();

return send\_message("/register " + name);

}

std::string utf16\_to\_utf8(const std::wstring& utf16\_string) {

int size\_needed = WideCharToMultiByte(CP\_UTF8, 0, utf16\_string.c\_str(), -1, nullptr, 0, nullptr, nullptr);

std::string utf8\_string(size\_needed - 1, 0);

WideCharToMultiByte(CP\_UTF8, 0, utf16\_string.c\_str(), -1, &utf8\_string[0], size\_needed, nullptr, nullptr);

return utf8\_string;

}

std::wstring utf8\_to\_utf16(const std::string& utf8\_string) {

int size\_needed = MultiByteToWideChar(CP\_UTF8, 0, utf8\_string.c\_str(), -1, nullptr, 0);

std::wstring utf16\_string(size\_needed - 1, 0);

MultiByteToWideChar(CP\_UTF8, 0, utf8\_string.c\_str(), -1, &utf16\_string[0], size\_needed);

return utf16\_string;

}

int proccess\_client\_command(const std::string& client\_command\_message) {

std::string command = client\_command\_message.substr(0, client\_command\_message.find\_first\_of(' ', 0));

if (command == "/exit") {

send\_message(client\_command\_message);

return EXIT;

}

return SUCCESS;

}

int proccess\_client\_message(const std::wstring& client\_message) {

std::string sending\_message = utf16\_to\_utf8(client\_message);

if (sending\_message.data()[0] == '/')

return proccess\_client\_command(sending\_message);

else

return send\_message(sending\_message);

}

int client\_input() {

HANDLE console = GetStdHandle(STD\_INPUT\_HANDLE);

INPUT\_RECORD inputRecord;

DWORD events;

std::wstring input\_message;

std::string sending\_message;

DWORD chars\_written;

std::cout << "> ";

while (true) {

ReadConsoleInput(console, &inputRecord, 1, &events);

if (inputRecord.EventType == KEY\_EVENT && inputRecord.Event.KeyEvent.bKeyDown) {

WORD input\_virtial\_key\_code = inputRecord.Event.KeyEvent.wVirtualKeyCode;

//CHAR input\_character = inputRecord.Event.KeyEvent.uChar.AsciiChar;

WCHAR input\_character = inputRecord.Event.KeyEvent.uChar.UnicodeChar;

switch (input\_virtial\_key\_code) {

case(VK\_RETURN):

if (!input\_message.empty()) {

std::cout << "\n";

if (proccess\_client\_message(input\_message) == EXIT)

return EXIT;

input\_message.clear();

}

break;

case(VK\_BACK):

if (!input\_message.empty()) {

input\_message.pop\_back();

std::cout << "\b \b";

}

break;

case(VK\_UP):

chat.window\_scroll\_offset++;

send\_message("/scroll\_up " + std::to\_string(chat.window\_scroll\_offset));

break;

case(VK\_DOWN):

if (chat.window\_scroll\_offset != 0) {

chat.window\_scroll\_offset--;

send\_message("/scroll\_down " + std::to\_string(chat.window\_scroll\_offset));

}

break;

default:

if (input\_character >= 0x20 && input\_character != 0x7F) {

input\_message += input\_character;

std::wcout << input\_character;

}

}

}

}

}

int make\_chat\_window() {

chat.make\_chat\_window\_mutex.lock();

system("cls");

GetConsoleScreenBufferInfo(GetStdHandle(STD\_OUTPUT\_HANDLE), &chat.csbi);

for (int message\_index = 0; message\_index < chat.client\_messages\_buf.size(); message\_index++) {

std::wcout << chat.client\_messages\_buf.at(message\_index) << std::endl;

}

SetConsoleCursorPosition(GetStdHandle(STD\_OUTPUT\_HANDLE), { 0,SHORT(chat.csbi.srWindow.Bottom-1 )});

std::cout << std::string(120, ' ');

std::cout << "> ";

chat.make\_chat\_window\_mutex.unlock();

return 0;

}

int proccess\_server\_command(const std::string& server\_command\_message) {

std::string server\_command = server\_command\_message.substr(0, server\_command\_message.find\_first\_of(' ', 0));

if (server\_command == "/scroll\_up") {

chat.client\_messages\_buf.pop\_back();

std::wstring client\_scroll\_message = utf8\_to\_utf16(server\_command\_message.substr(

server\_command\_message.find\_first\_of(' ', 0) + 1));

chat.client\_messages\_buf.push\_front(client\_scroll\_message);

make\_chat\_window();

return SUCCESS;

}

if (server\_command == "/scroll\_down") {

chat.client\_messages\_buf.pop\_front();

std::wstring client\_scroll\_message = utf8\_to\_utf16(server\_command\_message.substr(

server\_command\_message.find\_first\_of(' ', 0) + 1));

chat.client\_messages\_buf.push\_back(client\_scroll\_message);

make\_chat\_window();

return SUCCESS;

}

}

int receiving\_server\_messages() {

std::string server\_message;

std::wstring client\_message;

int ret\_val;

while (true) {

server\_message.clear();

server\_message.resize(1024);

ret\_val = recv(chat.client\_socket, (char\*)server\_message.data(), server\_message.size(), 0);

if (ret\_val == SOCKET\_ERROR) {

std::cout << "Receiving failed with error: " << WSAGetLastError();

closesocket(chat.client\_socket);

WSACleanup();

return ERR;

}

server\_message.resize(ret\_val);

client\_message = utf8\_to\_utf16(server\_message);

if (server\_message.data()[0] == '/') {

proccess\_server\_command(server\_message);

}

else {

client\_message = utf8\_to\_utf16(server\_message);

if (chat.client\_messages\_buf.size() == 20)

chat.client\_messages\_buf.pop\_front();

chat.client\_messages\_buf.push\_back(client\_message);

if (chat.window\_scroll\_offset == 0)

make\_chat\_window();

}

}

}

int client() {

if (connect\_server() == SUCCESS and register\_client() == SUCCESS) {

std::thread receiving\_messages\_thread(receiving\_server\_messages);

receiving\_messages\_thread.detach();

client\_input();

return SUCCESS;

}

else

return ERR;

}

int main() {

SetConsoleOutputCP(CP\_UTF8);

SetConsoleCP(CP\_UTF8);

setlocale(LC\_ALL, "");

client();

}