

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. 253503  
Тимошевич К.С.

Проверил: ассистент кафедры  
информатики Гриценко Н.Ю.

Минск 2024

## СОДЕРЖАНИЕ

1 Постановка задачи.....	3
2 Описание работы программы.....	4
2.1 Обработка сообщений на стороне клиента.....	4
2.2 Отправка сообщений на стороне клиента.....	4
2.3 Обработка сообщений на стороне сервера.....	4
3 Ход выполнения программы.....	5
3.1 Примеры выполнения задания.....	5
Вывод.....	7
Список использованных источников.....	8
Приложение А (справочное) Исходный код (к пункту 2.1).....	9

# 1 ПОСТАНОВКА ЗАДАЧИ

Задачей данной лабораторной работы является создание модели взаимодействия между процессами в сетевой среде. Цель — реализовать упрощенный чат для нескольких пользователей с использованием сокетов, демонстрирующий интерактивное взаимодействие через протоколы TCP или UDP. Основная архитектура предполагает централизованный подход, где выделенный сервер управляет соединениями клиентов, хранением сообщений и маршрутизацией передаваемых данных.

Модель предусматривает централизованную серверную часть, принимающую соединения от клиентов, временно хранящую сообщения и передающую их адресно конкретному получателю, а также клиентскую часть, позволяющую пользователям устанавливать соединение с сервером, вводить и отправлять сообщения, а также принимать входящие сообщения.

В процессе реализации будет разработана структура сообщений, включающая минимум два элемента: адрес получателя и текст сообщения.

## 2 ОПИСАНИЕ РАБОТЫ ПРОГРАММЫ

В данном разделе описаны основные функции программы, использованные для реализации сетевого чата, а также методы организации взаимодействия между сервером и клиентами.

### 2.1 Обработка сообщений на стороне клиента

Функция *receive\_messages* отвечает за прием сообщений от сервера. Клиент создает поток для постоянного прослушивания сообщений, поступающих на его сокет [1]. При получении сообщения функция выводит его на экран. Если соединение с сервером потеряно, программа уведомляет об этом пользователя и завершает прослушивание. Данная функция помогает обеспечить прием сообщений, не прерывая основной поток клиента.

### 2.2 Отправка сообщений на стороне клиента

Основной поток клиента обеспечивает ввод и отправку сообщений. Пользователь указывает имя получателя и текст сообщения, которые объединяются в строку формата имя отправителя | имя получателя | текст сообщения [2]. Сообщение отправляется на сервер, который отвечает за его маршрутизацию. Таким образом, клиентская программа обеспечивает прямую отправку сообщений и их отображение, а также поддерживает базовую структуру сообщения, включающую отправителя, получателя и содержимое.

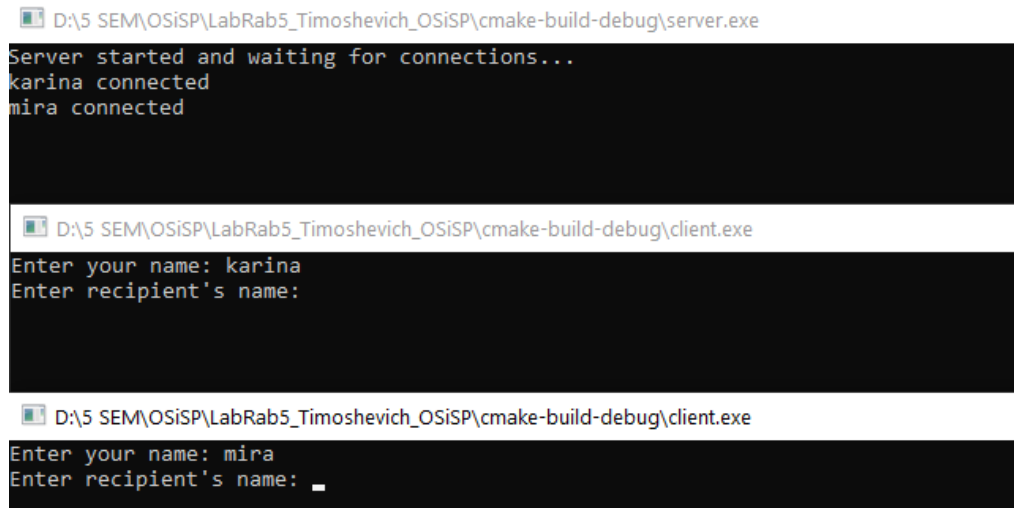
### 2.3 Обработка сообщений на стороне сервера

Функция *handle\_client* отвечает за обработку сообщений от подключенных клиентов. При подключении нового клиента сервер принимает его имя и добавляет в список активных пользователей. Затем сервер начинает прослушивание сообщений от клиента. При получении сообщения сервер извлекает из него имя получателя и пересылает текст адресату. Если получатель не подключен, сообщение сохраняется в очереди для доставки при следующем подключении клиента. Это гарантирует доставку сообщения адресату даже при временном отсутствии соединения, улучшая надежность программы [3].

## 3 ХОД ВЫПОЛНЕНИЯ ПРОГРАММЫ

### 3.1 Примеры выполнения задания

На рисунке 3.1 показан момент подключения двух клиентов к серверу. Каждый клиент вводит свое имя, после чего сервер принимает соединение и отображает сообщение о том, что к чату присоединились два пользователя.



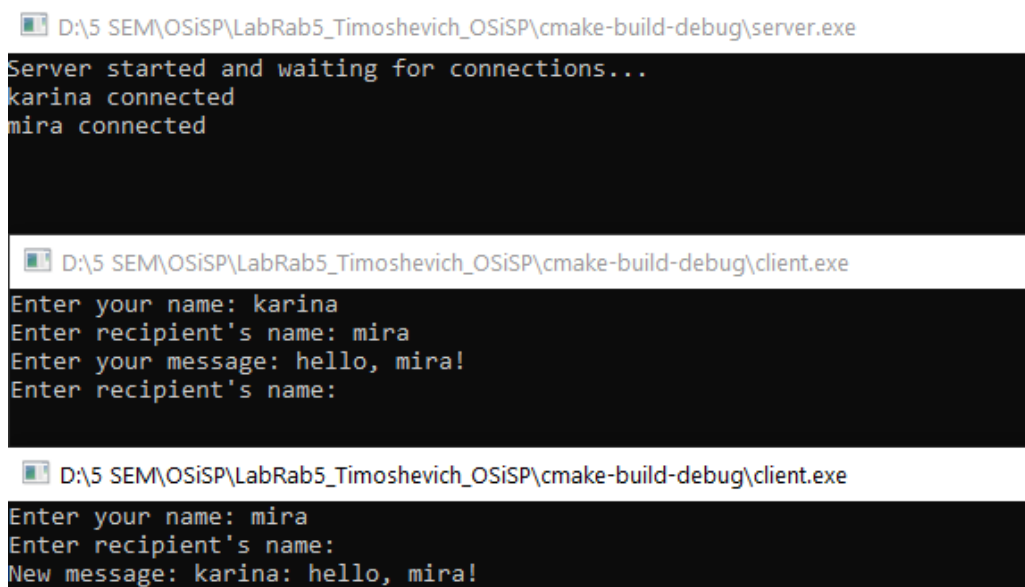
```
D:\5 SEM\OSiSP\LabRab5_Timoshevich_OSiSP\cmake-build-debug\server.exe
Server started and waiting for connections...
karina connected
mira connected

D:\5 SEM\OSiSP\LabRab5_Timoshevich_OSiSP\cmake-build-debug\client.exe
Enter your name: karina
Enter recipient's name:

D:\5 SEM\OSiSP\LabRab5_Timoshevich_OSiSP\cmake-build-debug\client.exe
Enter your name: mira
Enter recipient's name: _
```

Рисунок 3.1 - Подключение двух клиентов к серверу

На рисунке 3.2 один из клиентов вводит сообщение и указывает имя получателя. Сервер обрабатывает это сообщение и перенаправляет его указанному клиенту, у которого отображается новое сообщение от отправителя.




```
D:\5 SEM\OSiSP\LabRab5_Timoshevich_OSiSP\cmake-build-debug\server.exe
Server started and waiting for connections...
karina connected
mira connected

D:\5 SEM\OSiSP\LabRab5_Timoshevich_OSiSP\cmake-build-debug\client.exe
Enter your name: karina
Enter recipient's name: mira
Enter your message: hello, mira!
Enter recipient's name:

D:\5 SEM\OSiSP\LabRab5_Timoshevich_OSiSP\cmake-build-debug\client.exe
Enter your name: mira
Enter recipient's name:
New message: karina: hello, mira!
```

Рисунок 3.2 - Отправка и получение сообщения между клиентами


На рисунке 3.3 показан экран клиента, который получил сообщение от другого пользователя. Отправитель и текст сообщения отображаются у получателя, подтверждая успешную доставку данных..

 D:\5 SEM\OSiSP\LabRab5\_Timoshevich\_OSiSP\cmake-build-debug\client.exe

```
Enter your name: karina
Enter recipient's name: mira
Enter your message: hello, mira!
Enter recipient's name:
New message: mira: hi, my friend
```

Рисунок 3.3 - Получение сообщения от другого клиента

На рисунке 3.4 отображено состояние сервера после того, как оба клиента отключились от чата. В окне сервера выводятся сообщения о том, что клиенты покинули чат, и указывается информация о завершении их сессий. Это подтверждает, что сервер корректно отслеживает состояние подключений пользователей.

 D:\5 SEM\OSiSP\LabRab5\_Timoshevich\_OSiSP\cmake-build-debug\server.exe

```
Server started and waiting for connections...
karina connected
mira connected
mira disconnected
karina disconnected
```

Рисунок 3.4 - Отключение клиентов на сервере

## ВЫВОД

В ходе выполнения лабораторной работы была реализована модель простого сетевого взаимодействия по типу чата, где клиенты общаются через централизованный сервер с использованием *TCP*-сокетов. Реализация программы включала как клиентскую, так и серверную части, где сервер отвечает за маршрутизацию сообщений между пользователями и управление списком подключений, а клиенты за отправку и получение сообщений.

Система продемонстрировала корректное функционирование всех основных компонентов. Сообщения успешно передавались от одного клиента к другому по указанию адресата, а сервер поддерживал активные подключения и фиксировал отключения клиентов. Было разработано несколько примеров взаимодействия: от момента подключения клиентов и отправки сообщений до корректного завершения сессий при отключении.

Тестирование подтвердило, что программа способна обрабатывать запросы на передачу данных, обеспечивая передачу сообщений в режиме реального времени. Данная работа показала на практике принципы сетевого программирования и работу с сокетами в *Windows*, а также организацию взаимодействия в клиент-серверной архитектуре.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Функция `recv` (`winsock2.h`) [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/ru-ru/windows/win32/api/winsock2/nf-winsock2-recv>.  
– Дата доступа: 10.11.2024.

[2] Функция `send` (`winsock2.h`) [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/ru-ru/windows/win32/api/winsock2/nf-winsock2-send>.  
– Дата доступа: 10.11.2024.

[3] функция `listen` (`winsock2.h`) [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/ru-ru/windows/win32/api/winsock2/nf-winsock2-listen>.  
– Дата доступа: 10.11.2024.



## ПРИЛОЖЕНИЕ А

### (справочное)

### Исходный код

```
#include <iostream>
#include <winsock2.h>
#include <map>
#include <thread>
#include <string>
#include <sstream>
#include <queue>
#pragma comment(lib, "ws2_32.lib")

#define PORT 12345
#define BUFFER_SIZE 1024

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

void send_to_client(const std::string& message, const std::string&
receiver_name) {
    bool found = false;
    for (const auto& client : clients) {
        if (client.second == receiver_name) {
            send(client.first, message.c_str(), message.size(), 0);
            found = true;
            break;
        }
    }

    if (!found) {
        message_queue[receiver_name].push(message);
    }
}

void handle_client(SOCKET client_socket) {
    char buffer[BUFFER_SIZE];
    std::string client_name;
    int recv_size = recv(client_socket, buffer, BUFFER_SIZE, 0);
    if (recv_size > 0) {
        buffer[recv_size] = '\0';
        client_name = buffer;
        clients[client_socket] = client_name;
        std::cout << client_name << " connected\n";
        while (!message_queue[client_name].empty()) {
            send(client_socket, message_queue[client_name].front().c_str(),
message_queue[client_name].front().size(), 0);
            message_queue[client_name].pop();
        }
    }

    while (true) {
        int recv_size = recv(client_socket, buffer, BUFFER_SIZE, 0);
        if (recv_size <= 0) {
            std::cout << client_name << " disconnected\n";
            closesocket(client_socket);
            clients.erase(client_socket);
            break;
        }
        buffer[recv_size] = '\0';
    }
}
```

```

        std::stringstream ss(buffer);
        std::string sender, receiver, message;
        std::getline(ss, sender, '|');
        std::getline(ss, receiver, '|');
        std::getline(ss, message);
        std::string full_message = sender + ": " + message;
        send_to_client(full_message, receiver);
    }
}

int main() {
    WSADATA wsaData;
    WSASStartup(MAKEWORD(2, 2), &wsaData);
    SOCKET server_socket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    sockaddr_in server_addr;
    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = INADDR_ANY;
    server_addr.sin_port = htons(PORT);
    bind(server_socket, (sockaddr*)&server_addr, sizeof(server_addr));
    listen(server_socket, SOMAXCONN);

    std::cout << "Server started and waiting for connections...\n";

    while (true) {
        SOCKET client_socket = accept(server_socket, nullptr, nullptr);
        std::thread client_thread(handle_client, client_socket);
        client_thread.detach();
    }

    closesocket(server_socket);
    WSACleanup();
    return 0;
}

#include <iostream>
#include <winsock2.h>
#include <thread>
#include <string>
#pragma comment(lib, "ws2_32.lib")

#define PORT 12345
#define BUFFER_SIZE 1024

SOCKET client_socket;

void receive_messages() {
    char buffer[BUFFER_SIZE];
    while (true) {
        int recv_size = recv(client_socket, buffer, BUFFER_SIZE, 0);
        if (recv_size <= 0) {
            std::cout << "Connection lost with the server\n";
            break;
        }
        buffer[recv_size] = '\0';
        std::cout << "\nNew message: " << buffer << "\n";
    }
}

int main() {
    WSADATA wsaData;
    WSASStartup(MAKEWORD(2, 2), &wsaData);
    client_socket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);

```

```

sockaddr_in server_addr;
server_addr.sin_family = AF_INET;
server_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
server_addr.sin_port = htons(PORT);

if (connect(client_socket, (sockaddr*)&server_addr, sizeof(server_addr)) <
0) {
    std::cerr << "Failed to connect to the server.\n";
    return 1;
}

std::string name;
std::cout << "Enter your name: ";
std::getline(std::cin, name);
send(client_socket, name.c_str(), name.size(), 0);
std::thread receiver(receive_messages);
receiver.detach();

while (true) {
    std::string receiver_name, message;
    std::cout << "Enter recipient's name: ";
    std::getline(std::cin, receiver_name);
    std::cout << "Enter your message: ";
    std::getline(std::cin, message);
    std::string full_message = name + "|" + receiver_name + "|" + message;
    send(client_socket, full_message.c_str(), full_message.size(), 0);
}

closesocket(client_socket);
WSACleanup();
return 0;
}

```