

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

**ОТЧЕТ**  
К лабораторной работе № 6  
на тему

**ЭЛЕМЕНТЫ СЕТЕВОГО ПРОГРАММИРОВАНИЯ**

Выполнил:  
студент гр. 153503  
Кончик Д.С.

Проверил:  
Гриценко Н.Ю.

Минск 2024

## СОДЕРЖАНИЕ

1 Цель работы .....	3
2 Теоретические сведения .....	4
3 Полученные результаты .....	5
Выводы .....	6
Список использованных источников .....	7
Приложение А (обязательное) листинг кода.....	8

## **1 ЦЕЛЬ РАБОТЫ**

Практическое освоение основ построения и функционирования сетей, стеков протоколов, программных интерфейсов. Изучение сетевой подсистемы и программного интерфейса сокетов в Unix-системах. Практическое проектирование, реализация и отладка программ, взаимодействующих через сеть TCP/IP.

Написать программу, реализующую упрощенный чат для нескольких пользователей с использованием сетевых сокетов. Сервер должен создать сокет для приема соединений или отдельных сообщений, передавать сообщения адресно одному или нескольким клиентам. Клиент должен обнаруживать сервер, отправлять пользовательские сообщения серверу.

## 2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Сетевое программирование – это область программирования, связанная с разработкой приложений, способных взаимодействовать и обмениваться данными через компьютерные сети. Эта область охватывает создание программ, работающих в распределенной среде, где различные компьютеры могут общаться между собой, отправлять и принимать данные.

Сеть представляет собой инфраструктуру, позволяющую различным устройствам обмениваться информацией. Это может быть локальная сеть (*LAN*) в пределах одного офиса или дома, глобальная сеть, такая как Интернет, или виртуальная сеть между программами на одном компьютере.

Сокет – это виртуальная конструкция из *IP*-адреса и номера порта. Сокет служит для того, чтобы было проще писать код, а программы могли передавать данные друг другу даже в пределах одного компьютера. Т.к. сокет на сервере один, а программ, которые должны подключаться, много, то сервер копирует сокеты. Когда на сервер поступает запрос на соединение с сокетом, он не устанавливает связь напрямую, а копирует этот сокет и настраивает связь через него. После копирования сервер запоминает, какая копия отвечает за какое соединение, и дальше просто обрабатывает все запросы по очереди. При этом исходный сокет остаётся нетронутым – он не используется для связи напрямую, а служит шаблоном для создания копий [1].

Протоколы представляют собой набор правил и соглашений, определяющих формат, порядок и правила взаимодействия между устройствами или программами в сети. Они служат основой для обмена данными и обеспечивают структурированный способ передачи информации между различными компьютерами, устройствами или системами. Протоколы могут работать на разных уровнях стека сетевых протоколов (например, модель *OSI* или *TCP/IP*), предоставляя структуру для обработки различных аспектов передачи данных.

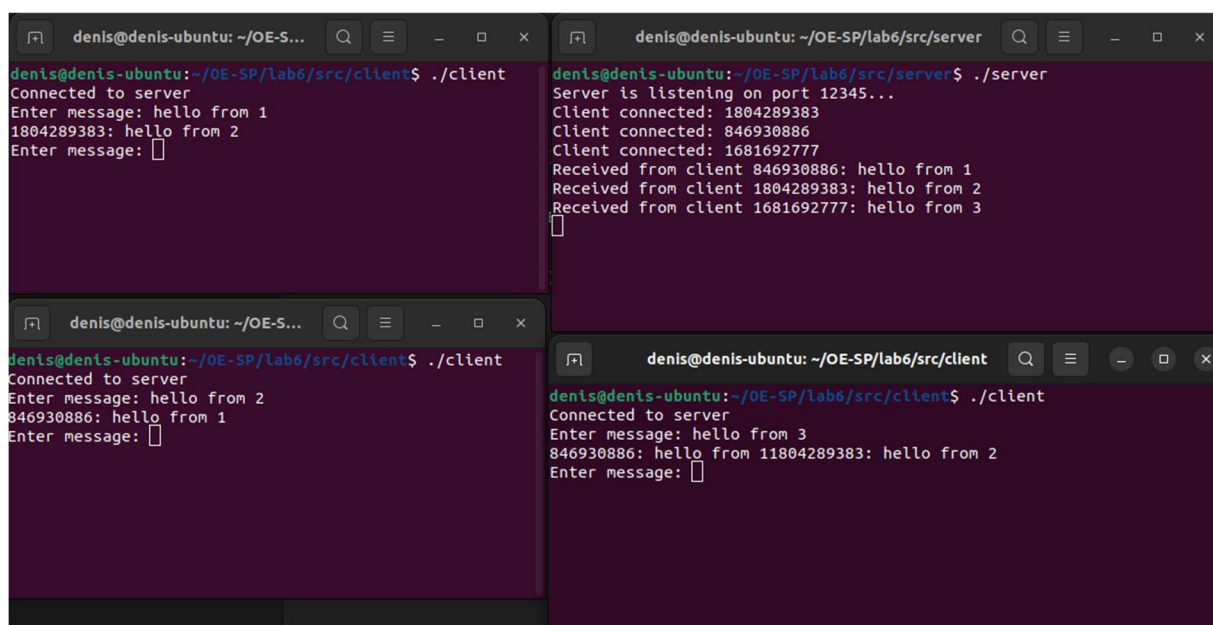
Сетевая модель *OSI* – сетевая модель стека сетевых протоколов *OSI/ISO*. Посредством данной модели различные сетевые устройства могут взаимодействовать друг с другом. Модель определяет различные уровни взаимодействия систем. Каждый уровень выполняет определённые функции при таком взаимодействии [2].

*TCP/IP* – это модель передачи цифровых данных. Протокол передачи *TCP/IP* описывает правила передачи данных, стандарты связи между компьютерами, а также содержит соглашения о маршрутизации и межсетевом взаимодействии [3].

### 3 ПОЛУЧЕННЫЕ РЕЗУЛЬТАТЫ

В результате выполнения лабораторной работы был написана программа, реализующая Упрощенный чат для нескольких пользователей с использованием сетевых сокетов.

Запускается сначала сервер, потом к нему подключаются клиенты. Клиенты могут отправлять сообщения на сервер. Сервер эти сообщения отправляет всем клиентам, за исключением того, от кого оно пришло. Сервер фиксирует подключение и отключение клиентов (рисунок 1).



```
denis@denis-ubuntu: ~/OE-S...  
denis@denis-ubuntu:~/OE-SP/lab6/src/client$ ./client  
Connected to server  
Enter message: hello from 1  
1804289383: hello from 2  
Enter message:   
denis@denis-ubuntu:~/OE-SP/lab6/src/server$ ./server  
Server is listening on port 12345...  
Client connected: 1804289383  
Client connected: 846930886  
Client connected: 1681692777  
Received from client 846930886: hello from 1  
Received from client 1804289383: hello from 2  
Received from client 1681692777: hello from 3  
  
denis@denis-ubuntu:~/OE-S...  
denis@denis-ubuntu:~/OE-SP/lab6/src/client$ ./client  
Connected to server  
Enter message: hello from 2  
846930886: hello from 1  
Enter message:   
denis@denis-ubuntu:~/OE-SP/lab6/src/client$ ./client  
Connected to server  
Enter message: hello from 3  
846930886: hello from 11804289383: hello from 2  
Enter message: 
```

Рисунок 1 – Результат работы программы

## **ВЫВОДЫ**

В результате выполнения лабораторной работы были практически освоены основы построения и функционирования сетей, стеков протоколов, программных интерфейсов. Изучены сетевые подсистемы и программные интерфейсы сокетов в Unix-системах.

Написана программа, реализующая упрощенный чат для нескольких пользователей с использованием сетевых сокетов. Сервер создает сокет для приема соединений или отдельных сообщений, передает сообщения адресно одному или нескольким клиентам. Клиент обнаруживает сервер, отправляет пользовательские сообщения серверу.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Сокет [Электронный ресурс]. – Режим доступа:  
<https://thecode.media/socket/>.

[2] Сетевая модель *OSI* [Электронный ресурс]. – Режим доступа:  
[https://ru.wikipedia.org/wiki/Клиент\\_–\\_сервер](https://ru.wikipedia.org/wiki/Клиент_–_сервер).

[3] *TCP/IP* [Электронный ресурс]. – Режим доступа:  
[https://www.nic.ru/help/что-такое-tcpip\\_11168.html](https://www.nic.ru/help/что-такое-tcpip_11168.html).

## ПРИЛОЖЕНИЕ А

### (обязательное)

### Листинг кода

#### Листинг 1 – Файл *server.c*:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <pthread.h>

#define DEFAULT_PORT 12345
#define BUFFER_SIZE 2056
#define MAX_CLIENTS 10

struct ClientInfo {
    int socket;
    char clientId[20];
};

struct ClientInfo clients[MAX_CLIENTS];
int num_clients = 0;

char* generate_client_id() {
    char* client_id = malloc(20 * sizeof(char));
    if (client_id == NULL) {
        perror("Memory allocation failed");
        exit(EXIT_FAILURE);
    }

    snprintf(client_id, 20, "%d", rand());
    return client_id;
}

void* handle_client(void* client_info_ptr) {
    struct ClientInfo* client_info = (struct ClientInfo*)client_info_ptr;
    int client_socket = client_info->socket;
    char buffer[BUFFER_SIZE];

    while (1) {
        int bytes_received = recv(client_socket, buffer, sizeof(buffer) - 1,
0);
        if (bytes_received == -1 || bytes_received == 0) {
            printf("Client %s disconnected\n", client_info->clientId);
            close(client_socket);
            return NULL;
        }

        buffer[bytes_received] = '\0';
        printf("Received from client %s: %s\n", client_info->clientId,
buffer);

        for (int i = 0; i < num_clients; ++i) {
            if (clients[i].socket != client_socket) {
                char message_with_id[BUFFER_SIZE];
                snprintf(message_with_id, sizeof(message_with_id), "%s: %s",
client_info->clientId, buffer);
            }
        }
    }
}
```



```

        send(clients[i].socket, message_with_id,
strlen(message_with_id), 0);
    }
}

}

}

int main() {
    struct sockaddr_in server_addr, client_addr;
    socklen_t client_addr_len = sizeof(client_addr);
    pthread_t client_threads[MAX_CLIENTS];

    int server_socket = socket(AF_INET, SOCK_STREAM, 0);
    if (server_socket == -1) {
        perror("Failed to create server socket");
        exit(EXIT_FAILURE);
    }

    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = INADDR_ANY;
    server_addr.sin_port = htons(DEFAULT_PORT);

    if (bind(server_socket, (struct sockaddr*)&server_addr,
sizeof(server_addr)) == -1) {
        perror("Failed to bind server socket");
        close(server_socket);
        exit(EXIT_FAILURE);
    }

    if (listen(server_socket, SOMAXCONN) == -1) {
        perror("Error in listen function");
        close(server_socket);
        exit(EXIT_FAILURE);
    }

    printf("Server is listening on port %d...\n", DEFAULT_PORT);

    while (1) {
        int client_socket = accept(server_socket, (struct
sockaddr*)&client_addr, &client_addr_len);
        if (client_socket == -1) {
            perror("Failed to accept client connection");
            continue;
        }

        if (num_clients >= MAX_CLIENTS) {
            printf("Maximum number of clients reached. Closing new
connection.\n");
            close(client_socket);
            continue;
        }

        struct ClientInfo new_client;
        new_client.socket = client_socket;
        strncpy(new_client.clientId, generate_client_id(),
sizeof(new_client.clientId));

        printf("Client connected: %s\n", new_client.clientId);
        clients[num_clients++] = new_client;
        pthread_create(&client_threads[num_clients - 1], NULL, handle_client,
(void*)&clients[num_clients - 1]);
    }
}

```

```
}  
}
```

## Листинг 2 – Файл *client.c*:

```
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <unistd.h>  
#include <arpa/inet.h>  
#include <sys/socket.h>  
#define SERVER_IP "127.0.0.1"  
#define SERVER_PORT 12345  
#define BUFFER_SIZE 1024  
  
int main() {  
    struct sockaddr_in server_addr;  
    int client_socket;  
  
    client_socket = socket(AF_INET, SOCK_STREAM, 0);  
    if (client_socket == -1) {  
        perror("Socket creation failed");  
        exit(EXIT_FAILURE);  
    }  
  
    server_addr.sin_family = AF_INET;  
    server_addr.sin_port = htons(SERVER_PORT);  
    if (inet_pton(AF_INET, SERVER_IP, &server_addr.sin_addr) != 1) {  
        perror("Invalid address");  
        close(client_socket);  
        exit(EXIT_FAILURE);  
    }  
  
    if (connect(client_socket, (struct sockaddr *)&server_addr,  
sizeof(server_addr)) == -1) {  
        perror("Connection failed");  
        close(client_socket);  
        exit(EXIT_FAILURE);  
    }  
    printf("Connected to server\n");  
    char buffer[BUFFER_SIZE];  
    while (1) {  
        printf("Enter message: ");  
        fgets(buffer, BUFFER_SIZE, stdin);  
        buffer[strcspn(buffer, "\n")] = 0;  
  
        if (send(client_socket, buffer, strlen(buffer), 0) == -1) {  
            perror("Message sending failed");  
            break;  
        }  
  
        int bytes_received = recv(client_socket, buffer, BUFFER_SIZE - 1, 0);  
        if (bytes_received <= 0) {  
            perror("Server disconnected");  
            break;  
        }  
        buffer[bytes_received] = '\0';  
        printf("%s\n", buffer);  
    }  
    close(client_socket);  
  
    return 0;  
}
```