

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**

Студент  
Преподаватель

О. Л. Дайнович  
Н. Ю. Гриценко

Минск 2024

## СОДЕРЖАНИЕ

1 Цель работы .....	3
2 Теоретические сведения .....	4
3 Полученные результаты .....	6
Заключение .....	9
Список использованных источников .....	10
Приложение А (обязательное) Листинг кода .....	11

# 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 (программы) в соответствии с вариантом задания. Спланировать и обеспечить тестирование (демонстрацию) выполнения – для нескольких взаимодействующих потоков это может быть существенно более сложно и трудоемко.

Желательно продолжать использовать make (и сценарии makefile) для управления обработкой проекта.

Упрощенный чат для нескольких пользователей с использованием сетевых сокетов.

Транспортные протоколы: TCP или UDP.

Архитектура: централизованная (выделенный процесс-сервер и процессы-клиенты) или децентрализованная (процессы-клиенты с «серверными» функциями).

Сервер: создание сокета для приема соединений или отдельных сообщений; прием и временное хранение сообщений; передача сообщений адресно одному или нескольким клиентам; поддержание списка актуальных клиентов.

Клиент: обнаружение сервера и соединение с ним; ввод пользовательских сообщений и передача их серверу либо напрямую соответствующему клиенту; прием и отображение сообщений.

## 2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

TCP, или Transmission Control Protocol используется для транспортировки сообщений между устройствами в сети.

В сети файлы не передаются целиком, а дробятся и передаются в виде относительно небольших сообщений. Далее они передаются другому устройству — получателю, где повторно собираются в файл.

Например, человек хочет скачать картинку. Сервер обрабатывает запрос и высылает в ответ требуемое изображение. Ему, в свою очередь, необходим путь или канал, по которому он будет передавать информацию. Поэтому сервер обращается к сетевому сокету для установки требуемого соединения и отправки картинки. Сервер дробит данные, инкапсулирует их в блоки, которые передаются на уровень TCP получателя при помощи IP-протокола. Далее получатель подтверждает факт передачи.

У протокола TCP есть несколько особенностей:

- Система нумерации сегментов. TCP отслеживает передаваемые и принимаемые сегменты, присваивая номера каждому из них. Байтам данных, которые должны быть переданы, присваивается определенный номер байта, в то время как сегментам присваиваются порядковые номера.

- Управление потоком. Функция ограничивает скорость, с которой отправитель передает данные. Это делается для обеспечения надежности доставки, в том числе чтобы компьютер не генерировал пакетов больше, чем может принять другое устройство. Если говорить простым языком, то получатель постоянно сообщает отправителю о том, какой объем данных может быть получен.

- Контроль ошибок. Функция реализуется для повышения надежности путем проверки байтов на целостность.

- Контроль перегрузки сети. Протокол TCP учитывает уровень перегрузки в сети, определяемый объемом данных, отправленных узлом.

Если нам очень важна скорость передачи, а вот потеря пакетов не так критична (как, например, в голосовом или видеотрафике), то лучше использовать UDP, или User Datagram Protocol. В отличие от TCP он обеспечивает передачу данных без получения подтверждения от пользователя. Проще говоря, просто отправляет пакеты и не ждет ничего в ответ. Из-за этого достигается высокая скорость в ущерб надежности.

Чаще всего UDP применяется в чувствительных ко времени службах, где потерять пакеты лучше, чем ждать. Звонки в Skype или Google Meet, стриминг видео, онлайн-трансляции используют этот протокол из-за того, что они

чувствительны ко времени и рассчитаны на определенный уровень потерь. Вся голосовая связь через интернет работает по протоколу UDP. Также UDP очень часто используется в онлайн-играх. Аналогичная история с DNS-серверами, поскольку они должны быть быстрыми и эффективными. [1]

Коммуникация в режиме реального времени и практически мгновенная передача данных являются обязательными стандартами современного интернета. Чтобы удовлетворить эти стандарты, в 2011 году появился протокол связи WebSocket, который позволяет сайтам отправлять и получать данные без задержки. С помощью веб-сокетов можно создавать многопользовательские игры, мессенджеры, а также сервисы для совместной работы.

WebSocket — это технология, которая позволяет клиенту установить двухстороннюю («дуплексную») связь с сервером. Сразу поясним: клиент — это приложение на компьютере или смартфоне пользователя, а сервер — это удаленный компьютер, на котором хранится веб-сайт и связанные с ним данные.

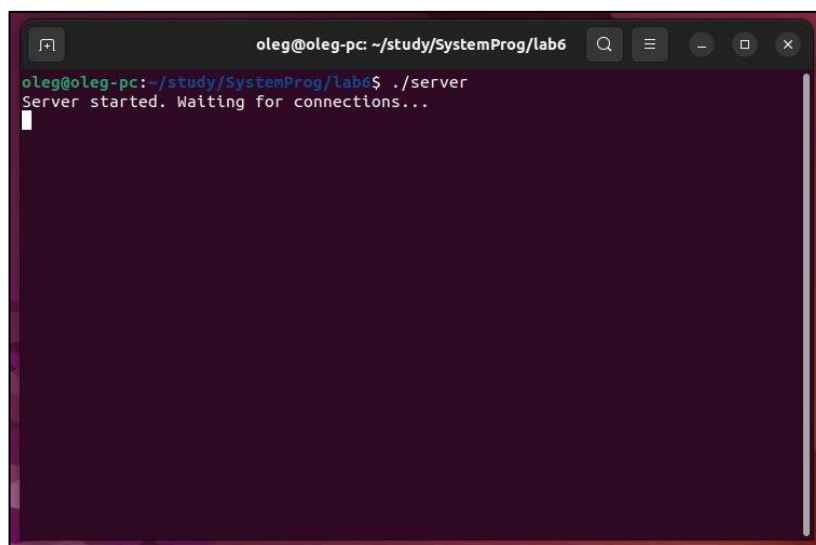
Ключевое слово в этом определении — двусторонний: с помощью веб-сокетов клиент и сервер могут инициировать связь друг с другом, а также могут отправлять сообщения одновременно. Почему это так важно? Чтобы в полной мере оценить возможности WebSocket, сделаем шаг назад и рассмотрим несколько самых распространенных способов, с помощью которых компьютеры могут получать данные с сервера. [2]

### 3 ПОЛУЧЕННЫЕ РЕЗУЛЬТАТЫ

В результате выполнения лабораторной работы была разработана программа, реализующая упрощенный чат для нескольких пользователей с использованием сетевых сокетов.

Основной код написан на языке C. Bash скрипт собирает программу, состоящую из файлов client.c и server.c, с помощью файла Makefile.

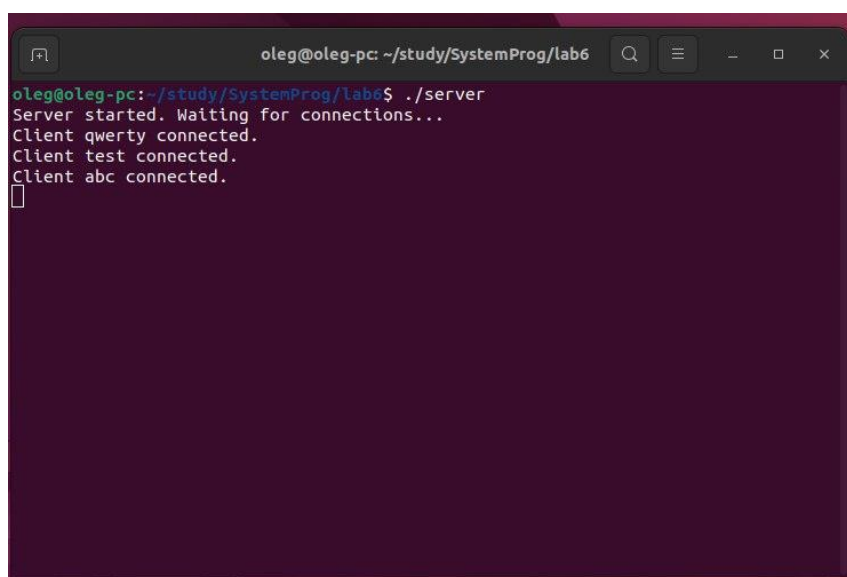
Сначала запускается сервер, который находится в режиме ожидания подключений от клиентов (рисунок 1).



```
oleg@oleg-pc: ~/study/SystemProg/lab6
oleg@oleg-pc:~/study/SystemProg/lab6$ ./server
Server started. Waiting for connections...
```

Рисунок 1 – Запуск программы сервера

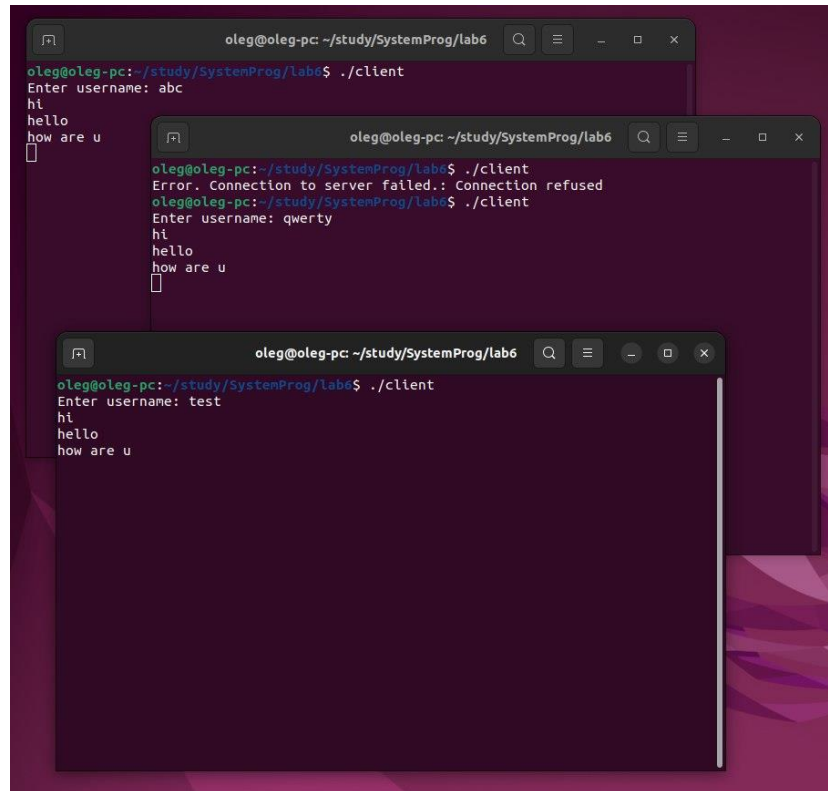
При подключении клиентов к серверу в консоли программы сервера выводится соответствующее сообщение (рисунок 2).



```
oleg@oleg-pc: ~/study/SystemProg/lab6
oleg@oleg-pc:~/study/SystemProg/lab6$ ./server
Server started. Waiting for connections...
Client qwerty connected.
Client test connected.
Client abc connected.
```

Рисунок 2 – Подключение клиентов к серверу

После подключения к серверу, клиенты могут обмениваться сообщениями, которые выводятся в общем чате всем пользователям (рисунок 3).



The image shows three terminal windows stacked vertically, all with the title bar 'oleg@oleg-pc: ~/study/SystemProg/lab6'. The top window shows a client connecting with username 'abc' and sending messages 'hi', 'hello', and 'how are u'. The middle window shows a client connecting with username 'qwerty', receiving an error message 'Error. Connection to server failed.: Connection refused', and then successfully connecting with the same username and sending the same three messages. The bottom window shows a client connecting with username 'test' and sending the same three messages.

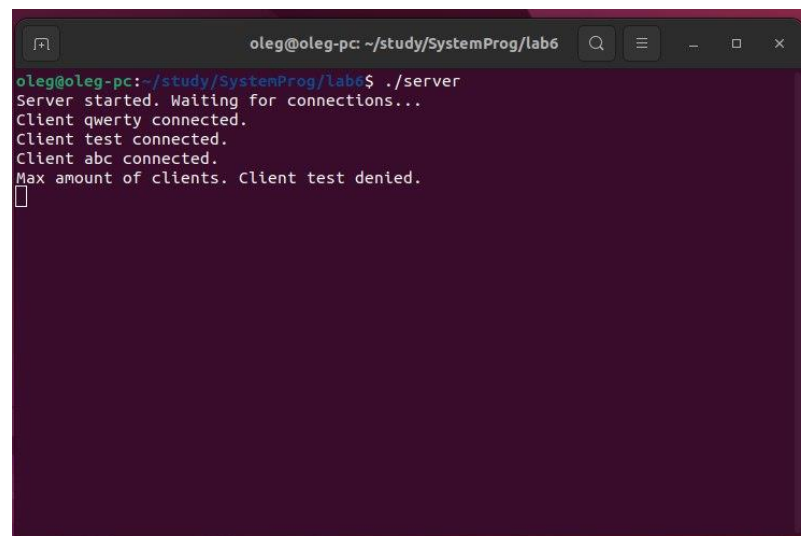
```
oleg@oleg-pc: ~/study/SystemProg/lab6$ ./client
Enter username: abc
hi
hello
how are u
[]

oleg@oleg-pc: ~/study/SystemProg/lab6$ ./client
Error. Connection to server failed.: Connection refused
oleg@oleg-pc: ~/study/SystemProg/lab6$ ./client
Enter username: qwerty
hi
hello
how are u
[]

oleg@oleg-pc: ~/study/SystemProg/lab6$ ./client
Enter username: test
hi
hello
how are u
```

Рисунок 3 – Вывод сообщений в чате

На сервере стоит ограничение числа пользователей. При попытке подключиться к переполненному серверу выводится соответствующее сообщение (рисунок 4).

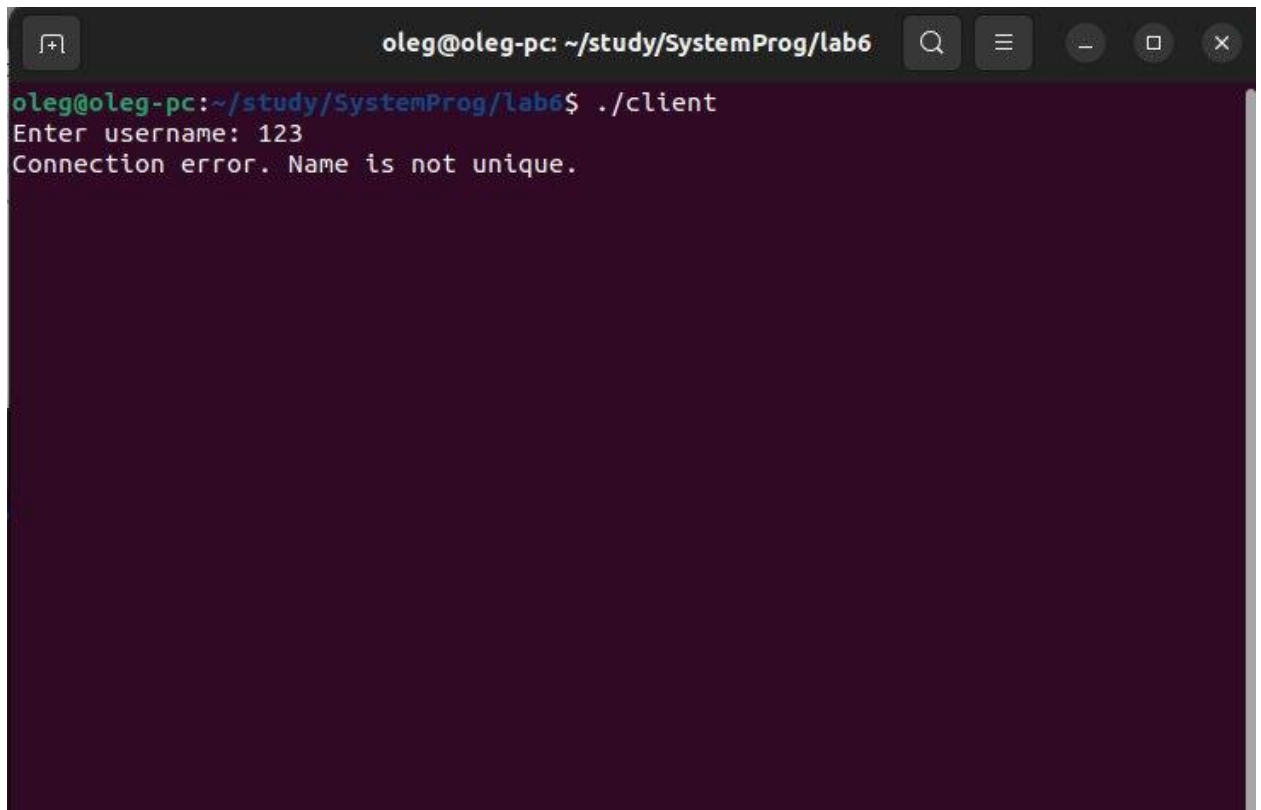


The image shows a terminal window with the title bar 'oleg@oleg-pc: ~/study/SystemProg/lab6'. It shows the output of running './server'. The server logs show three clients connecting: 'qwerty', 'test', and 'abc'. After the third connection, it displays the message 'Max amount of clients. Client test denied.'.

```
oleg@oleg-pc: ~/study/SystemProg/lab6$ ./server
Server started. Waiting for connections...
Client qwerty connected.
Client test connected.
Client abc connected.
Max amount of clients. Client test denied.
[]
```

Рисунок 4 – Вывод сообщения о переполненности сервера

Помимо регулирования количества пользователей на сервере, программой предусмотрена проверка уникальности имен пользователей. При попытке подключения нового пользователя с уже существующим именем, в консоль выводится соответствующее сообщение (рисунок 5).

A terminal window with a dark background and light text. The title bar at the top reads 'oleg@oleg-pc: ~/study/SystemProg/lab6'. The terminal content shows a command prompt 'oleg@oleg-pc:~/study/SystemProg/lab6\$' followed by the command './client'. Below this, the prompt 'Enter username: 123' is shown, and the output is 'Connection error. Name is not unique.'.

```
oleg@oleg-pc: ~/study/SystemProg/lab6
oleg@oleg-pc:~/study/SystemProg/lab6$ ./client
Enter username: 123
Connection error. Name is not unique.
```

Рисунок 5 – Вывод сообщения о некорректном имени пользователя



## **ЗАКЛЮЧЕНИЕ**

В ходе данной лабораторной работы были усвоены практические основы построения и функционирования сетей, стеков протоколов, программных интерфейсов, изучены сетевая подсистема и программный интерфейс сокетов в Unix-системах. Разобрано практическое проектирование, реализация и отладка программ, взаимодействующих через сеть TCP/IP.

А также был разработан программный продукт, реализующий упрощенный чат с возможностью подключения нескольких пользователей с централизованной архитектурой в виде сервера в качестве центрального узла.

## **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] Отличия TCP- и UDP-протоколов [Электронный ресурс]. – Режим доступа: <https://selectel.ru/blog/tcp-vs-udp/> – Дата доступа: 07.04.2024.

[2] Что такое веб-сокеты и как они вообще работают [Электронный ресурс]. – Режим доступа <https://ru.hexlet.io/blog/posts/что-такое-websocket-i-kak-oni-voobsche-rabotayut> – Дата доступа: 07.04.2024.

# ПРИЛОЖЕНИЕ А

## (обязательное)

### Листинг кода

#### Листинг 1 – Файл lab6.sh

```
#!/usr/bin/bash
make
./build/main
```

#### Листинг 2 – Файл makefile

```
CC = gcc
CFLAGS = -Wall -Wextra -pthread
all: server client
server: server.o
    $(CC) $(CFLAGS) -o server server.o
client: client.o
    $(CC) $(CFLAGS) -o client client.o
server.o: server.c
    $(CC) $(CFLAGS) -c server.c
client.o: client.c
    $(CC) $(CFLAGS) -c client.c
clean:
    rm -f server client *.o
```

#### Листинг 3 – Файл client.c

```
#include <arpa/inet.h>
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <unistd.h>
#define BUFFER_SIZE 1024
void *receive_messages(void *arg) {
    int server_socket = *(int *)arg;
    char buffer[BUFFER_SIZE];
    int read_size;
    while ((read_size = recv(server_socket, buffer, BUFFER_SIZE, 0)) > 0) {
        buffer[read_size] = '\0';
        printf("%s\n", buffer);
    }
    pthread_exit(NULL);
}
int main() {
    int server_socket;
    struct sockaddr_in server_addr;
    server_socket = socket(AF_INET, SOCK_STREAM, 0);
    if (server_socket == -1) {
        perror("Socket creation error.");
        exit(1);
    }
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(12345);
```

```

server_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
if (connect(server_socket, (struct sockaddr *)&server_addr,
    sizeof(server_addr)) == -1) {
    perror("Error. Connection to server failed.");
    exit(1);
}
char username[20];
printf("Enter username: ");
fgets(username, sizeof(username), stdin);
username[strcspn(username, "\n")] = '\0';
if (send(server_socket, username, sizeof(username), 0) == -1) {
    perror("Error. Failed to send username to server");
    close(server_socket);
    exit(1);
}
pthread_t thread;
if (pthread_create(&thread, NULL, receive_messages,
    (void *)&server_socket) != 0) {
    perror("Thread creation error.");
    close(server_socket);
    exit(1);
}
char buffer[BUFFER_SIZE];
while (1) {
    fgets(buffer, sizeof(buffer), stdin);
    buffer[strcspn(buffer, "\n")] = '\0';
    if (send(server_socket, buffer, strlen(buffer), 0) == -1) {
        perror("Message sending error.");
        break;
    }
}
close(server_socket);
return 0;
}

```

#### Листинг 4 – Файл server.c

```

#include <arpa/inet.h>
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <unistd.h>
#define MAX_CLIENTS 3
#define BUFFER_SIZE 1024
typedef struct {
    int client_socket;
    char username[20];
} Client;
Client clients[MAX_CLIENTS];
pthread_t threads[MAX_CLIENTS];
size_t num_clients = 0;
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
void *handle_client(void *arg) {
    Client *client = (Client *)arg;
    char buffer[BUFFER_SIZE];
    int read_size;
    while ((read_size = recv(client->client_socket, buffer, BUFFER_SIZE, 0))
>
        0) {

```

```

        buffer[read_size] = '\0';
        if (buffer[0] == '@') {
            char *recipient_username = strtok(buffer, " ");
            char *message = strtok(NULL, "");
            recipient_username++;
            pthread_mutex_lock(&mutex);
            for (size_t i = 0; i < num_clients; i++) {
                if (strcmp(clients[i].username, recipient_username) == 0) {
                    send(clients[i].client_socket, message, strlen(message),
0);
                        break;
                    }
                }
            pthread_mutex_unlock(&mutex);
        } else {
            pthread_mutex_lock(&mutex);
            for (size_t i = 0; i < num_clients; i++) {
                if (clients[i].client_socket != client->client_socket) {
                    send(clients[i].client_socket, buffer, strlen(buffer),
0);
                }
            }
            pthread_mutex_unlock(&mutex);
        }
    }
    pthread_mutex_lock(&mutex);
    for (size_t i = 0; i < num_clients; i++) {
        if (clients[i].client_socket == client->client_socket) {
            clients[i] = clients[num_clients - 1];
            break;
        }
    }
    num_clients--;
    pthread_mutex_unlock(&mutex);
    close(client->client_socket);
    free(client);
    pthread_exit(NULL);
}

int main() {
    int server_socket, client_socket;
    struct sockaddr_in server_addr, client_addr;
    socklen_t addr_size;
    server_socket = socket(AF_INET, SOCK_STREAM, 0);
    if (server_socket == -1) {
        perror("Socket creation error.");
        exit(1);
    }
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(12345);
    server_addr.sin_addr.s_addr = INADDR_ANY;
    if (bind(server_socket, (struct sockaddr *)&server_addr,
        sizeof(server_addr)) == -1) {
        perror("Error. Connection socket to adress failed.");
        exit(1);
    }
    if (listen(server_socket, MAX_CLIENTS) == -1) {
        perror("Error in socket listening.");
        exit(1);
    }
    printf("Server started. Waiting for connections...\n");
    while (1) {
        addr_size = sizeof(client_addr);

```

```

        client_socket =
            accept(server_socket, (struct sockaddr *)&client_addr,
&addr_size);
        if (client_socket == -1) {
            perror("Error in connection acceptance.");
            continue;
        }
        char username[20];
        if (recv(client_socket, username, sizeof(username), 0) <= 0) {
            perror("Error. Failed to get username.");
            close(client_socket);
            continue;
        }
        Client *client = (Client *)malloc(sizeof(Client));
        client->client_socket = client_socket;
        strncpy(client->username, username, sizeof(client->username));
        pthread_mutex_lock(&mutex);
        if (num_clients < MAX_CLIENTS) {
            int is_name_unique = 1;
            for (size_t i = 0; i < num_clients; i++) {
                if (strcmp(clients[i].username, client->username) == 0) {
                    char reject_msg[100] = "Connection error. Name is not
unique.\n";
                    send(client->client_socket, reject_msg,
strlen(reject_msg),
                        0);
                    close(client_socket);
                    is_name_unique = 0;
                    break;
                }
            }
            if (is_name_unique) {
                clients[num_clients] = *client;
                num_clients++;
                printf("Client %s connected.\n", client->username);
            }
        } else {
            printf(
                "Max amount of clients. Client %s "
                "denied.\n",
                client->username);
            char reject_msg[100] = "Connection refused. Server is full.\n";
            send(client->client_socket, reject_msg, strlen(reject_msg), 0);
            free(client);
            close(client_socket);
            continue;
        }
        pthread_mutex_unlock(&mutex);
        if (pthread_create(&threads[num_clients - 1], NULL, handle_client,
            (void *)client) != 0) {
            perror("Thread creation error.");
            free(client);
            close(client_socket);
            continue;
        }
    }
    close(server_socket);
    return 0;
}

```