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 М. Ю. Пухов

Преподаватель С. И. Сиротко

Минск 2024

СОДЕРЖАНИЕ

[1 Цель работы 3](#_Toc146752068)

[2 Теоретические сведения 4](#_Toc146752069)

[3 Результат выполнения 5](#_Toc146752070)

[Заключение 6](#_Toc146752071)

[Список использованных источников 7](#_Toc146752072)

[Приложение А (обязательное) Листинг кода 8](#_Toc146752073)

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.

1. **ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Сетевое программирование предполагает разработку приложений, компоненты которых взаимодействуют между собой по сети, обычно с использованием модели клиент-сервер.

Для организации сетевого взаимодействия в системах Unix используется сокет - конечная точка двусторонней связи между процессами. Сокет ассоциируется с портом и IP-адресом узла.

Для работы с сокетами в Linux применяются системные вызовы:

* socket() - создание сокета
* bind() - связывание сокета с IP-адресом и портом
* listen() - прослушивание входящих подключений к сокету
* accept() - прием входящего подключения
* connect() - инициация подключения к удаленному сокету
* send()/recv() - отпрака и получение данных по сокету

В данной работе реализован упрощенный чат посредством TCP-сокетов.

Сервер организует прослушивание порта, принимает подключения клиентов, регистрирует их в системе, ретранслирует сообщения между ними.

Клиент подключается к серверу, отправляет и принимает сообщения в режиме реального времени.

1. **РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ**

Было создано приложение для клиент-серверного подключения и пересылки сообщений. Клиент показан ниже (рисунок 1).

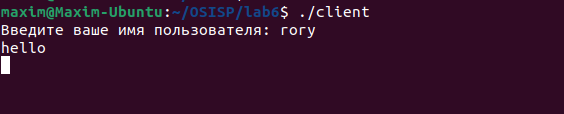


Рисунок 1 – Клиентская часть

Серверная часть, которая ожидает подключения от клиента и принимает сообщения, показа ниже (рисунок 2).

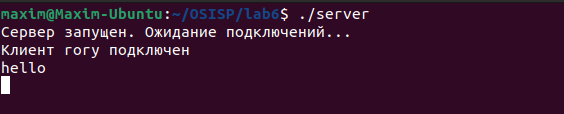


Рисунок 2 – Серверная часть

# ЗАКЛЮЧЕНИЕ

В ходе выполнения данной работы было реализовано клиент-серверное взаимодействие для организации упрощенного сетевого чата.

Были изучены базовые сетевые возможности ОС Linux, в частности интерфейс сокетов и основные системные вызовы для работы с ними.

На стороне сервера реализовано создание сокета, привязка его к порту, прослушивание входящих подключений с использованием вызовов socket(), bind(), listen(). Также была организована обработка клиентов в отдельных потоках, передача сообщений между ними.

Клиент инициирует подключение к серверу с помощью connect(), отправляет и получает данные вызовами send() и recv() соответственно. Полученная реализация демонстрирует базовое клиент-серверное взаимодействие с применением сокетов в ОС Linux.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

# Server-Client implementation in C [Электронный ресурс]. – Режим доступа: https://www.geeksforgeeks.org/tcp-server-client-implementation-in-c/.

# C Programming Blogs [Электронный ресурс]. – Режим доступа: https://www.learn-c.org/

# ПРИЛОЖЕНИЕ А (обязательное) Листинг кода

Листинг 1 – 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;

// Обработка сообщений от клиента

while ((read\_size = recv(client->client\_socket, buffer, BUFFER\_SIZE, 0)) > 0) {

buffer[read\_size] = '\0';

// Если сообщение начинается с '@', считаем, что это приватное сообщение

if (buffer[0] == '@') {

char \*recipient\_username = strtok(buffer, " ");

char \*message = strtok(NULL, "");

recipient\_username++;

// Поиск клиента с указанным именем и отправка ему приватного сообщения

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 {

// Отправка сообщения всем клиентам, кроме отправителя

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);

}

}

// Удаление отключенного клиента из списка и завершение потока

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;

// Создание сокета для сервера

server\_socket = socket(AF\_INET, SOCK\_STREAM, 0);

if (server\_socket == -1) {

perror("Ошибка создания сокета");

exit(1);

}

// Настройка адреса сервера

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

server\_addr.sin\_port = htons(12345);

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

// Привязка сокета к адресу сервера

if (bind(server\_socket, (struct sockaddr \*)&server\_addr, sizeof(server\_addr)) == -1) {

perror("Ошибка связывания сокета с адресом");

exit(1);

}

// Прослушивание сокета для подключений

if (listen(server\_socket, MAX\_CLIENTS) == -1) {

perror("Ошибка прослушивания сокета");

exit(1);

}

printf("Сервер запущен. Ожидание подключений...\n");

while (1) {

addr\_size = sizeof(client\_addr);

// Принятие подключения от клиента

client\_socket = accept(server\_socket, (struct sockaddr \*)&client\_addr, &addr\_size);

if (client\_socket == -1) {

perror("Ошибка принятия соединения");

continue;

}

char username[20];

// Получение имени пользователя от клиента

if (recv(client\_socket, username, sizeof(username), 0) <= 0) {

perror("Ошибка получения имени пользователя");

close(client\_socket);

continue;

}

// Создание структуры Client для нового клиента

Client \*client = (Client \*)malloc(sizeof(Client));

client->client\_socket = client\_socket;

strncpy(client->username, username, sizeof(client->username));

pthread\_mutex\_lock(&mutex);

// Проверка уникальности имени пользователя и добавление клиента в список

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] = "Отказано в подключении. Имя не уникально\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("Клиент %s подключен\n", client->username);

}

} else {

printf(

"Превышено максимальное количество клиентов. Клиент %s отклонен\n",

client->username);

char reject\_msg[100] = "Отказано в подключении. Слишком много клиентов\n";

send(client->client\_socket, reject\_msg, strlen(reject\_msg), 0);

free(client);

close(client\_socket);

continue;

}

pthread\_mutex\_unlock(&mutex);

// Создание отдельного потока для обработки сообщений от клиента

if (pthread\_create(&threads[num\_clients - 1], NULL, handle\_client,

(void \*)client) != 0) {

perror("Ошибка создания потока");

free(client);

close(client\_socket);

continue;

}

}

close(server\_socket);

return 0;

}

Листинг 1 – 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*

*// Функция для приема сообщений от сервера*

*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;*

*// Создаем сокет*

*server\_socket = socket(AF\_INET, SOCK\_STREAM, 0);*

*if (server\_socket == -1) {*

*perror("Ошибка создания сокета");*

*exit(1);*

*}*

*// Настраиваем адрес сервера*

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

*server\_addr.sin\_port = htons(12345);*

*server\_addr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");*

*// Устанавливаем соединение с сервером*

*if (connect(server\_socket, (struct sockaddr \*)&server\_addr,*

*sizeof(server\_addr)) == -1) {*

*perror("Ошибка соединения с сервером");*

*exit(1);*

*}*

*// Получаем имя пользователя от пользователя*

*char username[20];*

*printf("Введите ваше имя пользователя: ");*

*fgets(username, sizeof(username), stdin);*

*username[strcspn(username, "\n")] = '\0';*

*// Отправляем имя пользователя серверу*

*if (send(server\_socket, username, sizeof(username), 0) == -1) {*

*perror("Ошибка отправки имени пользователя");*

*close(server\_socket);*

*exit(1);*

*}*

*// Создаем поток для приема сообщений от сервера*

*pthread\_t thread;*

*if (pthread\_create(&thread, NULL, receive\_messages,*

*(void \*)&server\_socket) != 0) {*

*perror("Ошибка создания потока");*

*close(server\_socket);*

*exit(1);*

*}*

*// Отправка сообщений серверу*

*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("Ошибка отправки сообщения");*

*break;*

*}*

*}*

*close(server\_socket);*

*return 0;*

*}*