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 А. В. Скворцов

Преподаватель Н. Ю. Гриценко

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. Создание чата, симулирующего интерактивное взаимодействие пользователей, используя протокол TCP.

1. **ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Практическое проектирование, реализация и отладка программ, взаимодействующих через сеть TCP/IP, включает в себя несколько ключевых аспектов. Вот теоретический обзор этих аспектов.

TCP/IP (Transmission Control Protocol/Internet Protocol) является набором протоколов, который обеспечивает связь и передачу данных в компьютерных сетях. Он состоит из нескольких протоколов на разных уровнях, включая IP (Internet Protocol), TCP (Transmission Control Protocol), UDP (User Datagram Protocol) и другие. Протокол TCP обеспечивает надежную и упорядоченную доставку данных, в то время как протокол UDP предоставляет ненадежную и неупорядоченную доставку [1].

Взаимодействие программ через сеть в TCP/IP осуществляется с помощью сокетов. Сокет представляет собой программный интерфейс, который позволяет программам создавать сетевые соединения, отправлять и принимать данные. Сокеты бывают двух типов: сокеты клиента и сокеты сервера. Клиентский сокет устанавливает соединение с сервером, а серверный сокет слушает и принимает входящие соединения от клиентов.

При проектировании программы, взаимодействующей через сеть TCP/IP, необходимо определить роли клиента и сервера, а также функциональность, которую они должны выполнять. Это включает определение протоколов обмена данными, форматов сообщений и логики обработки запросов и ответов [2].

Реализация программы включает написание кода на выбранном языке программирования, который использует сокеты для создания соединений, отправки и приема данных. Для TCP-соединений обычно используются функции, такие как socket, bind, listen, accept, connect, send и recv, для установления и управления соединениями.

Отладка программы, взаимодействующей через сеть TCP/IP, может быть сложной из-за распределенной природы приложения. Важно использовать инструменты и методы отладки, которые позволяют отслеживать передачу данных, проверять правильность протоколов и обрабатывать возможные ошибки, такие как потеря соединения или неправильные сообщения.

1. **РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ**

В результате лабораторной работы был создан программный продукт, выполняющий условия лабораторной работы, ниже на рисунке можно видеть запуск приложения сервера и подключение клиентов (рисунок 1):

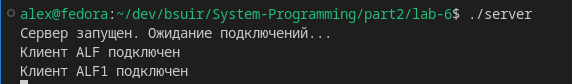


Рисунок 1 – Запуск сервера и подключение к нему клиентов

На втором и третьем рисунке можно наблюдать взаимодействие двух клиентов, которые обмениваются сообщениями (рисунок 2, 3):

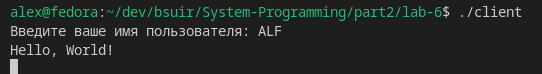


Рисунок 2 – Общение первого клиента



Рисунок 2 – Общение второго клиента

Исходя из данных рисунков легко понять, что программа работает исправно и выполняет все свои функции.

# ЗАКЛЮЧЕНИЕ

Изучены основы построения и функционирования сетей, стеков протоколов, программных интерфейсов. Изучена сетевая подсистемы и программные интерфейсы сокетов в Unix-системах. Произведено практическое проектирование, реализация и отладка программ, взаимодействующих через сеть TCP/IP. Создан чат, симулирующий интерактивное взаимодействие пользователей, используя протокол TCP.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. TCP и UDP [Электронный ресурс]. – Режим доступа: https://habr.com/ru/companies/ruvds/articles/759988/. – Дата доступа: 22.04.2024
2. Разработка клиент серверных приложений [Электронный ресурс]. – Режим доступа: https://otherreferats.allbest.ru/programming/00194309\_0.html. – Дата доступа: 22.04.2024.

# ПРИЛОЖЕНИЕ А (обязательное) Листинг кода

Листинг 1 – server.c

#include <arpa/inet.h>

#include <pthread.h>

#include <stdio.h>

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("Ошибка создания сокета");

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("Ошибка связывания сокета с адресом");

exit(1);

}

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);

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

if (client\_socket == -1) {

perror("Ошибка принятия соединения");

continue;

}

char username[20];

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

perror("Ошибка получения имени пользователя");

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] = "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);

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;

}