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 И. В. Бобко

Преподаватель Н. Ю. Гриценко

Минск 2024

СОДЕРЖАНИЕ

[1 Цель работы 3](#_Toc146752068)

[2 Теоретические сведения 4](#_Toc146752069)

[3 Результат выполнения 6](#_Toc146752070)

[Заключение 6](#_Toc146752071)

[Список использованных источников 7](#_Toc146752072)

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

1. **ЦЕЛЬ РАБОТЫ**

Целью данной работы является 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.

1. **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].

1. **РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ**

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).

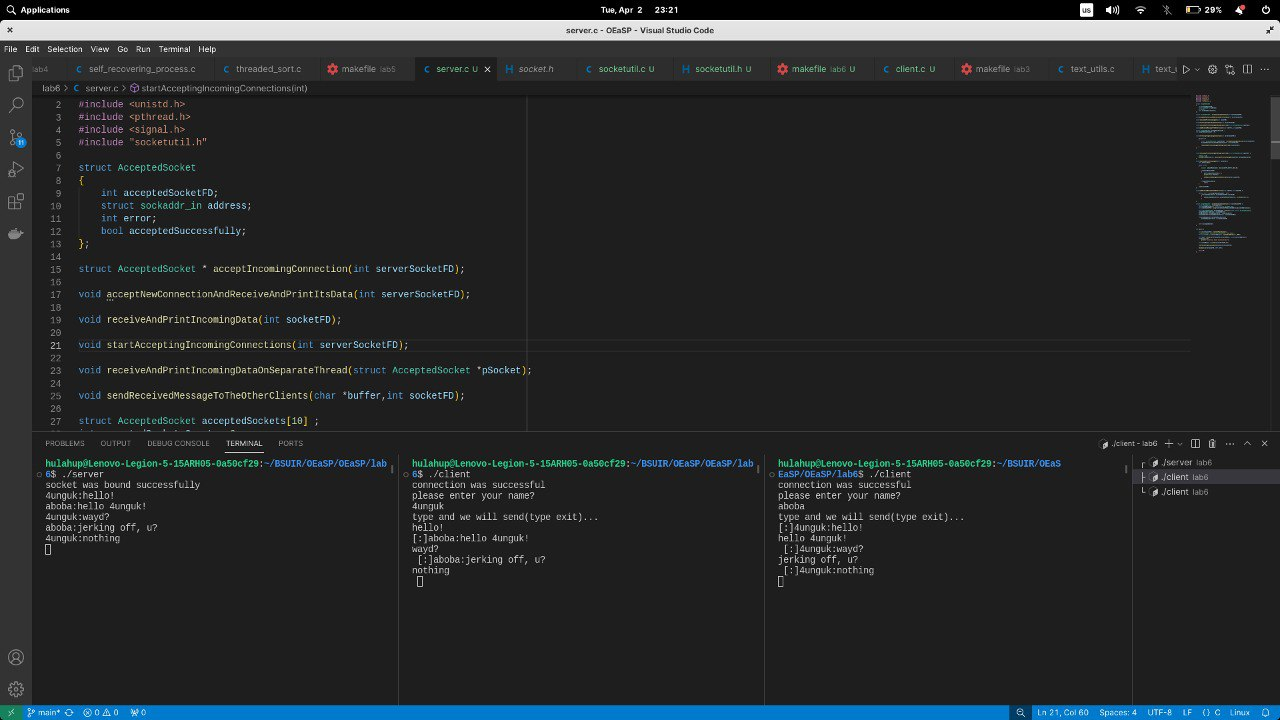


Рисунок 1 – Результаты выполнения программы

# ЗАКЛЮЧЕНИЕ

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/ – Дата доступа: 30.03.2024.
2. Сетевая модель OSI [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Клиент\_–\_сервер – Дата доступа: 30.03.2024.
3. TCP/IP [Электронный ресурс]. – Режим доступа: https://www.nic.ru/help/chto-takoe-tcpip\_11168.html – Дата доступа: 30.03.2024.

# ПРИЛОЖЕНИЕ А (обязательное) Листинг кода

Листинг 1 – Файл server.c

#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

#define PORT 12325

#define ADRESS "127.0.0.1"

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';

pthread\_mutex\_lock(&mutex);

for (size\_t i = 0; i < num\_clients; i++) {

if (clients[i].client\_socket != client->client\_socket) {

char full\_message[BUFFER\_SIZE];

snprintf(full\_message, sizeof(full\_message), "[%s]: %s", client->username, buffer);

send(clients[i].client\_socket, full\_message, strlen(full\_message), 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); // (Ipv4, TCP, TCP/IP)

if (server\_socket == -1) {

perror("Error creating socket");

exit(1);

}

server\_addr.sin\_family = AF\_INET; // Ipv4

server\_addr.sin\_port = htons(PORT); // "host to net"

server\_addr.sin\_addr.s\_addr = inet\_addr(ADRESS); // All local interfaces

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

perror("Error binding");

exit(1);

}

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

perror("Socket listening error");

exit(1);

}

printf("The server is running. 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 accepting connection");

continue;

}

char username[20];

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

perror("Error retrieving 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 refused. 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 is connected\n", client->username);

}

} else {

printf("The maximum number of clients has been exceeded. Client %s rejected\n", client->username);

char reject\_msg[100] = "Connection refused. Too many clients\n";

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

free(client);

Листинг 2 – Файл client.c

#include <pthread.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/socket.h>

#include <unistd.h>

#define BUFFER\_SIZE 1024

#define PORT 12325

#define ADRESS "127.0.0.1"

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); // (Ipv4, TCP, TCP/IP)

if (server\_socket == -1) {

perror("Error creating socket");

exit(1);

}

server\_addr.sin\_family = AF\_INET; // Ipv4

server\_addr.sin\_port = htons(PORT); // "host to net"

server\_addr.sin\_addr.s\_addr = inet\_addr(ADRESS);

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

perror("Error connecting to server");

exit(1);

}

char username[20];

printf("Enter your username: ");

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

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

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

perror("Error sending username");

close(server\_socket);

exit(1);

}

pthread\_t thread;

if (pthread\_create(&thread, NULL, receive\_messages, (void \*)&server\_socket) != 0) {

perror("Error creating thread");

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("Error sending message");

break;

}

}

close(server\_socket);

return 0;

}

Листинг 3 – Файл makefile

CC = gcc

CFLAGS = -w

OBJ = server.o client.o

EXEC = server client

all: $(EXEC)

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 $(OBJ) $(EXEC)