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

ОТЧЁТ

к лабораторной работе №6

на тему

**ЭЛЕМЕНТЫ СЕТЕВОГО ПРОГРАММИРОВАНИЯ**

|  |
| --- |
| Выполнил: студент гр. 253503  Кудош А.С. |
| Проверил: ассистент кафедры информатики Гриценко Н.Ю. |

Минск 2025

**СОДЕРЖАНИЕ**

[1 Постановка задачи и программа для её решения 3](#_Toc193113658)

[1.1 Постановка задачи 3](#_Toc193113659)

[1.2 Демонстрация работы программы 3](#_Toc193113660)

[Заключение 4](#_Toc193113661)

[Список использованных источников 5](#_Toc193113662)

[Приложение А (справочное) Исходный код 6](#_Toc193113663)

**1 ПОСТАНОВКА ЗАДАЧИ И ПРОГРАММА ДЛЯ ЕЁ РЕШЕНИЯ**

**1.1 Постановка задачи**

Постановка задачи для лабораторной работы заключается в разработке упрощенного чата для нескольких пользователей с использованием сетевых сокетов и транспортного протокола *TCP* (или *UDP*) [1]. Цель работы – практическое освоение основ построения и функционирования сетей, а также программного интерфейса сокетов в *Unix*-системах.

Серверная часть чата должна включать создание сокета для приема соединений от клиентов, прием и временное хранение сообщений, а также передачу сообщений адресно одному или нескольким клиентам. Сервер также должен поддерживать список актуальных клиентов для обеспечения корректной доставки сообщений. Клиентская часть должна обеспечивать обнаружение сервера и установление с ним соединения, а также позволять пользователю вводить сообщения, которые передаются на сервер или напрямую соответствующему клиенту. Клиенты должны принимать и отображать сообщения от других пользователей.

Структура передаваемых сообщений должна включать адрес отправителя и текст сообщения. Также необходимо определить порядок обмена сообщениями между клиентом и сервером, включая формат сообщения и правила взаимодействия. Для тестирования требуется обеспечить взаимодействие нескольких потоков и продемонстрировать работоспособность чата, включая обмен сообщениями между как минимум тремя клиентами. В работе рекомендуется использовать язык программирования C/C++, а также систему сборки make для управления проектом.

**1.2 Демонстрация работы программы**

Программный код предоставлен в приложении А.

Для реализации поставленной задачи были созданы две программы (сервер и клиент), для начала общения через чат нужно запустить сервер, затем открыть клиент и ввести *username*, после этого возможно общение через реализованный чат (рисунок 1.1).



Рисунок 1.1 – Реализованный чат

**ЗАКЛЮЧЕНИЕ**

В ходе выполнения лабораторной работы была успешно реализована программа для интерактивного взаимодействия пользователей в виде чата, использующего сетевые сокеты и протокол *TCP*. Созданный чат позволяет нескольким пользователям одновременно подключаться к серверу, отправлять и получать сообщения, как в общем чате, так и в виде личных сообщений [2].

Проект включает две основные части: серверную и клиентскую. Серверная часть отвечает за управление подключениями клиентов, обработку входящих сообщений и их распространение среди участников чата. Клиентская часть позволяет пользователю вводить сообщения и получать их от других участников. Использование потоков для обработки каждого клиента обеспечивает параллельное взаимодействие и стабильность работы приложения.

В процессе работы над проектом были изучены ключевые аспекты сетевого программирования, такие как работа с сокетами, управление потоками и синхронизация данных между ними. Также была разработана структура сообщений, что позволило реализовать функционал личных сообщений между клиентами.

Выводы по лабораторной работе подтверждают, что освоение основ сетевого программирования является важным шагом для дальнейшего изучения более сложных сетевых технологий и протоколов. Реализованный чат стал практическим примером применения теоретических знаний и навыков, полученных в ходе курса.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] Транспортный уровень: TCP И UDP [Электронный ресурс]. – Режим доступа: https://habr.com/ru/articles/878534/.

[2] Клиент-серверный чат, используя сокеты Qt/C++ [Электронный ресурс]. – Режим доступа: https://habr.com/ru/articles/131585/.

ПРИЛОЖЕНИЕ А  
(справочное)  
Исходный код

Листинг А.1 – Сервер

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#include <pthread.h>

#define MAX\_CLIENTS 10

#define BUFFER\_SIZE 1024

typedef struct {

int sockfd;

char username[32];

} Client;

Client clients[MAX\_CLIENTS];

int num\_clients = 0;

pthread\_mutex\_t clients\_mutex = PTHREAD\_MUTEX\_INITIALIZER;

void send\_to\_all(char \*message, int sender\_sock) {

pthread\_mutex\_lock(&clients\_mutex);

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

if (clients[i].sockfd != sender\_sock) {

send(clients[i].sockfd, message, strlen(message), 0);

}

}

pthread\_mutex\_unlock(&clients\_mutex);

}

void send\_to\_user(char \*username, char \*message, int sender\_sock) {

pthread\_mutex\_lock(&clients\_mutex);

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

if (strcmp(clients[i].username, username) == 0) {

send(clients[i].sockfd, message, strlen(message), 0);

break;

}

}

pthread\_mutex\_unlock(&clients\_mutex);

}

void \*handle\_client(void \*arg) {

int sockfd = \*((int \*)arg);

free(arg);

char buffer[BUFFER\_SIZE];

char username[32];

int read\_size;

if ((read\_size = recv(sockfd, username, sizeof(username), 0)) > 0) {

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

pthread\_mutex\_lock(&clients\_mutex);

int username\_taken = 0;

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

if (strcmp(clients[i].username, username) == 0) {

username\_taken = 1;

break;

}

}

if (username\_taken) {

send(sockfd, "Username taken. Disconnecting...", 32, 0);

close(sockfd);

pthread\_mutex\_unlock(&clients\_mutex);

pthread\_exit(NULL);

} else {

clients[num\_clients].sockfd = sockfd;

strcpy(clients[num\_clients].username, username);

num\_clients++;

pthread\_mutex\_unlock(&clients\_mutex);

sprintf(buffer, "%s joined the chat!", username);

send\_to\_all(buffer, sockfd);

}

while ((read\_size = recv(sockfd, buffer, sizeof(buffer), 0)) > 0) {

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

if (strncmp(buffer, "/", 1) == 0) {

char \*recipient = strtok(buffer + 1, " ");

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

if (recipient && message) {

sprintf(buffer, "[PM from %s] %s", username, message);

send\_to\_user(recipient, buffer, sockfd);

}

} else {

char formatted\_buffer[BUFFER\_SIZE \* 2];

snprintf(formatted\_buffer, sizeof(formatted\_buffer), "%s: %s", username, buffer);

send\_to\_all(formatted\_buffer, sockfd);

}

}

}

pthread\_mutex\_lock(&clients\_mutex);

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

if (clients[i].sockfd == sockfd) {

clients[i] = clients[num\_clients - 1];

num\_clients--;

break;

}

}

pthread\_mutex\_unlock(&clients\_mutex);

close(sockfd);

pthread\_exit(NULL);

}

int main(int argc, char \*argv[]) {

if (argc != 2) {

printf("Usage: %s <port>\n", argv[0]);

exit(EXIT\_FAILURE);

}

int server\_sock, client\_sock;

struct sockaddr\_in server\_addr, client\_addr;

socklen\_t addr\_len = sizeof(client\_addr);

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

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

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

server\_addr.sin\_port = htons(atoi(argv[1]));

bind(server\_sock, (struct sockaddr \*)&server\_addr, sizeof(server\_addr));

listen(server\_sock, 5);

printf("Server listening on port %s...\n", argv[1]);

while ((client\_sock = accept(server\_sock, (struct sockaddr \*)&client\_addr, &addr\_len))) {

printf("New connection from %s:%d\n", inet\_ntoa(client\_addr.sin\_addr), ntohs(client\_addr.sin\_port));

int \*new\_sock = malloc(sizeof(int));

\*new\_sock = client\_sock;

pthread\_t thread\_id;

pthread\_create(&thread\_id, NULL, handle\_client, (void \*)new\_sock);

}

close(server\_sock);

return 0;

}

Листинг А.2 – Клиент

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#include <pthread.h>

#define BUFFER\_SIZE 1024

int sockfd;

char username[32];

void \*receive\_handler(void \*arg) {

char buffer[BUFFER\_SIZE];

while (1) {

int read\_size = recv(sockfd, buffer, sizeof(buffer), 0);

if (read\_size > 0) {

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

printf("%s\n", buffer);

}

}

}

int main(int argc, char \*argv[]) {

if (argc != 3) {

printf("Usage: %s <server IP> <port>\n", argv[0]);

exit(EXIT\_FAILURE);

}

printf("Enter username: ");

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

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

struct sockaddr\_in server\_addr;

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

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

server\_addr.sin\_port = htons(atoi(argv[2]));

inet\_pton(AF\_INET, argv[1], &server\_addr.sin\_addr);

connect(sockfd, (struct sockaddr \*)&server\_addr, sizeof(server\_addr));

send(sockfd, username, strlen(username) + 1, 0);

pthread\_t recv\_thread;

pthread\_create(&recv\_thread, NULL, receive\_handler, NULL);

char buffer[BUFFER\_SIZE];

while (fgets(buffer, sizeof(buffer), stdin) != NULL) {

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

if (strncmp(buffer, "/quit", 5) == 0) {

break;

}

send(sockfd, buffer, strlen(buffer), 0);

}

close(sockfd);

return 0;

}

Листинг А.3 – Makefile

all: server client

server: server.c

gcc -pthread -o server server.c

client: client.c

gcc -pthread -o client client.c

clean:

rm -f server client