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 В. П. Бычко

Преподаватель Н. Ю. Гриценко

Минск 2024

СОДЕРЖАНИЕ

[1 Постановка задачи 3](file:///E:\Downloads\Telegram%20Desktop\лабораторная%20работа%205%20(2).docx#_Toc146631498)

[2 Краткие теоретические сведения 4](file:///E:\Downloads\Telegram%20Desktop\лабораторная%20работа%205%20(2).docx#_Toc146631499)

[3 Результаты выполнения лабораторной работы](file:///E:\Downloads\Telegram%20Desktop\лабораторная%20работа%205%20(2).docx#_Toc146631500) 8

[Выводы](file:///E:\Downloads\Telegram%20Desktop\лабораторная%20работа%205%20(2).docx#_Toc146631501) 9

[Список использованных источников](file:///E:\Downloads\Telegram%20Desktop\лабораторная%20работа%205%20(2).docx#_Toc146631502) 10

[Приложение А (обязательное) Листинг кода](file:///E:\Downloads\Telegram%20Desktop\лабораторная%20работа%205%20(2).docx#_Toc146631503) 11

## 1 ПОСТАНОВКА ЗАДАЧИ

Целью выполнения данной лабораторной работы является создание программы, реализующей упрощенный чат для нескольких пользователей с использованием сетевых сокетов

## 2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Стек протоколов TCP/IP (Transmission Control Protocol/Internet Protocol, протокол управления передачей/протокол интернета) – сетевая модель, описывающая процесс передачи цифровых данных. Она названа по двум главным протоколам, по этой модели построена глобальная сеть интернет. Чтобы обеспечить возможность передачи данных из одной сети в другую, был разработан стек интернет-протоколов, известный как TCP/IP.

Данный стек состоит из следующих уровней: прикладного, транспортного, межсетевого и канального.

Канальный уровень (link layer). Предназначение канального уровня – дать описание тому, как происходит обмен информацией на уровне сетевых устройств, определить, как информация будет передаваться от одного устройства к другому. Информация здесь кодируется, делится на пакеты и отправляется по нужному каналу, т.е. среде передачи.

Этот уровень также вычисляет максимальное расстояние, на которое пакеты возможно передать, частоту сигнала, задержку ответа и т.д. Все это – физические свойства среды передачи информации.

Межсетевой уровень (internet layer). Глобальная сеть интернет состоит из множества локальных сетей, взаимодействующих между собой. Межсетевой уровень используется, чтобы описать обеспечение такого взаимодействия.

Межсетевое взаимодействие – это основной принцип построения интернета. Локальные сети по всему миру объединены в глобальную, а передачу данных между этими сетями осуществляют магистральные и пограничные маршрутизаторы.

Транспортный уровень (transport layer). Протоколы транспортного уровня –TCP и UDP, они занимаются доставкой информации.

TCP (протокол управления передачей) – надежный, он обеспечивает передачу информации, проверяя дошла ли она, насколько полным является объем полученной информации и т.д. TCP дает возможность двум конечным устройствам производить обмен пакетами через предварительно установленное соединение. Он предоставляет услугу для приложений, повторно запрашивает потерянную информацию, устраняет дублирующие пакеты, регулируя загруженность сети. TCP гарантирует получение и сборку информации у адресата в правильном порядке.

UDP (протокол пользовательских датаграмм) – ненадежный, он занимается передачей автономных датаграмм. UDP не гарантирует, что всех датаграммы дойдут до получателя. Датаграммы уже содержат всю необходимую информацию, чтобы дойти до получателя, но они все равно могут быть потеряны или доставлены в порядке отличном от порядка при отправлении.

UDP обычно не используется, если требуется надежная передача информации. Использовать UDP имеет смысл там, где потеря части информации не будет критичной для приложения, например, в видеоиграх или потоковой передаче видео. UDP необходим, когда делать повторный запрос сложно или неоправданно по каким-то причинам.

Прикладной уровень (application layer). В модели TCP/IP отсутствуют дополнительные промежуточные уровни (представления и сеансовый) в отличие от OSI. Функции форматирования и представления данных делегированы библиотекам и программным интерфейсам приложений (API) – своего рода базам знаний, содержащим сведения о том, как приложения взаимодействуют между собой. Когда службы или приложения обращаются к библиотеке или API, те в ответ предоставляют набор действий, необходимых для выполнения задачи и полную инструкцию, каким образом эти действия нужно выполнять.

Протоколы прикладного уровня действуют для большинства приложений, они предоставляют услуги пользователю или обмениваются данными с «коллегами» с нижних уровней по уже установленным соединениям. Здесь для большинства приложений созданы свои протоколы. Например, браузеры используют HTTP для передачи гипертекста по сети, почтовые клиенты – SMTP для передачи почты, FTP-клиенты – протокол FTP для передачи файлов, службы DHCP – протокол назначения IP-адресов DHCP и так далее.

Приложения прикладного уровня, общаются также с предыдущим, транспортным, но они видят его протоколы как «черные ящики». Для приема-передачи информации они могут работать, например, с TCP или UDP, но понимают только конечный адрес в виде IP и порта, а не принцип их работы.

Как говорилось ранее, IP-адрес присваивается каждому конечному устройству протоколом межсетевого уровня. Но обмен данными происходит не между конечными устройствами, а между приложениями, установленными на них. Чтобы получить доступ к тому или иному сетевому приложению недостаточно только IP-адреса, поэтому для идентификации приложений применяют также порты. Комбинация IP-адреса и порта называется сокетом, или гнездом (socket). [1]

Cокеты обеспечивают двухстороннюю связь типа ``точка-точка'' между двумя процессами. Они являются основными компонентами межсистемной и межпроцессной связи. Каждый сокет представляет собой конечную точку связи, с которой может быть совмещено некоторое имя. Он имеет определенный тип, и один процесс или несколько, связанных с ним процессов.

Сокеты находятся в областях связи (доменах). Домен сокета – это абстракция, которая определяет структуру адресации и набор протоколов. Сокеты могут соединяться только с сокетами в том же домене. Сокеты могут использоваться для установки связи между процессами на отдельной системе подобно другим формам IPC.

Класс сокетов UNIX обеспечивает их адресное пространство для отдельной вычислительной системы. Сокеты области UNIX называются именами файлов UNIX. Сокеты также можно использовать, чтобы организовать связь между процессами на различных системах. Адресное пространство сокетов между связанными системами называют доменом Интернета. Коммуникации домена Интернета используют стек протоколов TCP/IP.

Типы сокетов определяют особенности связи, доступные приложению. Процессы взаимодействуют только через сокеты одного и того же типа. Основные типы сокетов:

Поточный – обеспечивает двухсторонний, последовательный, надежный, и недублированный поток данных без определенных границ. Тип сокета – SOCK\_STREAM, в домене Интернета он использует протокол TCP.

Датаграммный – поддерживает двухсторонний поток сообщений. Приложение, использующее такие сокеты, может получать сообщения в порядке, отличном от последовательности, в которой эти сообщения посылались. Тип сокета – SOCK\_DGRAM, в домене Интернета он использует протокол UDP.

Сокет последовательных пакетов – обеспечивает двухсторонний, последовательный, надежный обмен датаграммами фиксированной максимальной длины. Тип сокета – SOCK\_SEQPACKET. Для этого типа сокета не существует специального протокола.

Простой сокет – обеспечивает доступ к основным протоколам связи. [2]

Вызов socket создаёт сокет, connect соединяет сокет с удалённым сокетным адресом, bind привязывает сокет к локальному адресу, listen сообщает сокету, что должны приниматься новые соединения, а accept используется для получения нового сокета для нового входящего соединения. Вызовы send, sendto и sendmsg отправляют данные в сокет, а recv, recvfrom и recvmsg принимают данные из сокета.

Каждый сокетный домен имеет свой формат сокетных адресов, выраженный в отдельной адресной структуре. Каждая из этих структур начинается с целочисленного поля «семейства» (с типом sa\_family\_t), в котором указывается тип адресной структуры. Это позволяет различным системным вызовам (например, connect, bind, accept, getsockname, getpeername), которые являются общими для всех сокетов, определить домен конкретного сокетного адреса.

Для передачи сокетного адреса любого типа через программный интерфейс сокетов служит тип struct sockaddr. Целью данного типа является приведение типов сокетных адресов определённого домена к «общему» типу, что позволяет избежать предупреждений компилятора о несовпадении типов в вызовах API сокетов. [3]

## 3 РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ

В ходе выполнения лабораторной работы была создана программа, реализующая упрощенный чат для нескольких пользователей с использованием сетевых сокетов. Результат работы этой программы представлен на рисунке ниже.

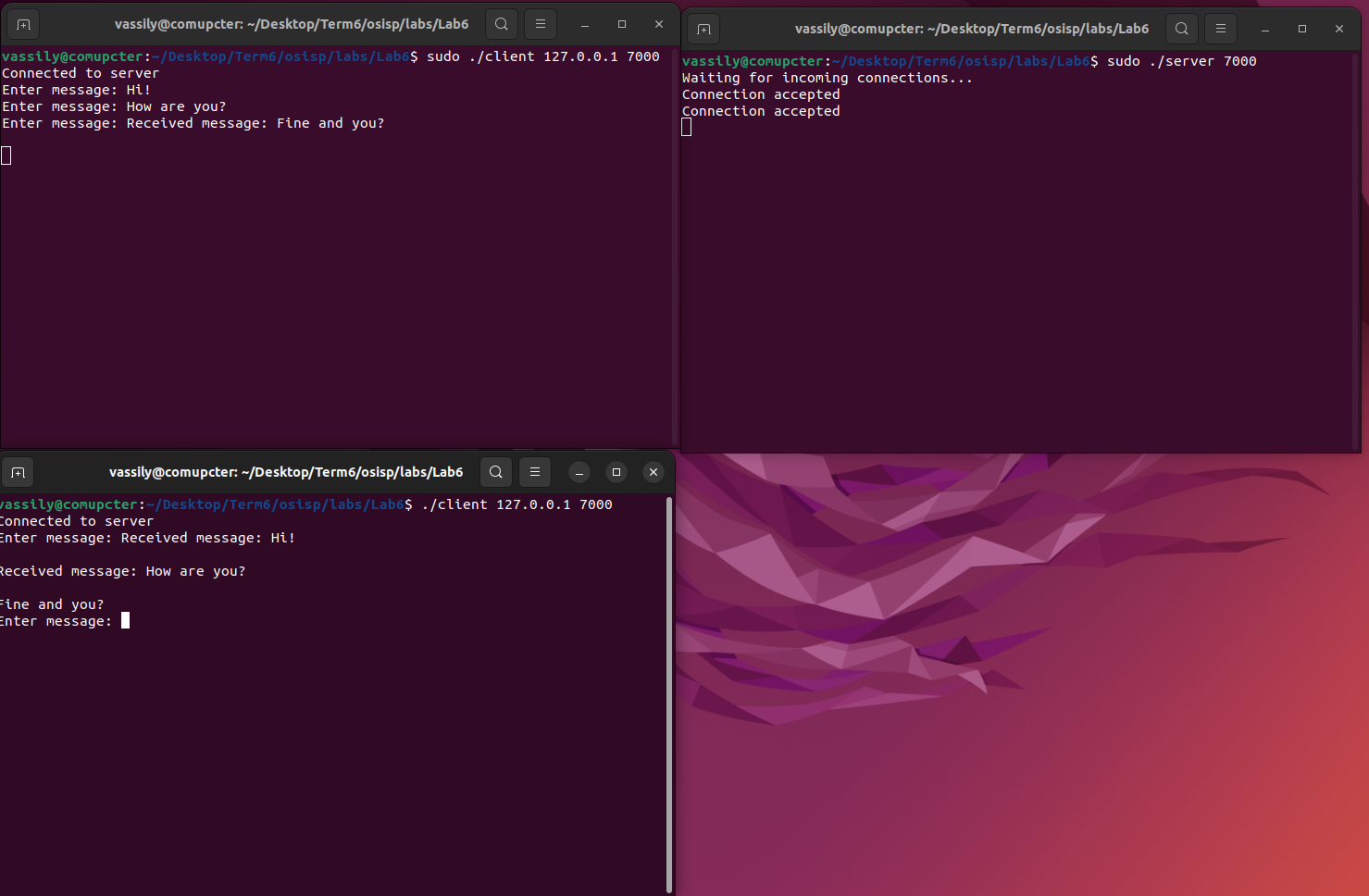


Рисунок 3.1 – Результат работы программы

## ВЫВОДЫ

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

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Руководство по стеку протоколов TCP/IP [Электронный ресурс]. – Режим доступа: https://selectel.ru/blog/tcp-ip-for-beginners/. – Дата доступа: 07.04.2024.

[2] Общие сведения о сокетах [Электронный ресурс]. – Режим доступа: https://www.opennet.ru/docs/RUS/linux\_parallel/node77.html. – Дата доступа: 07.04.2024.

[3] Интерфейс сокетов Linux [Электронный ресурс]. – Режим доступа: https://ru.manpages.org/socket/7. – Дата доступа: 07.04.2024

## ПРИЛОЖЕНИЕ А

## (обязательное)

## Листинг кода

**server.c**

#include <stdio.h>

#include <string.h>

#include <sys/socket.h>

#include <arpa/inet.h>

#include <unistd.h>

#include <pthread.h>

#define MAX\_CLIENTS 100

#define BUF\_SIZE 2000

#define NAME\_SIZE 20

struct client {

int id;

char name[NAME\_SIZE];

int sock;

};

struct client clients[MAX\_CLIENTS];

int client\_count = 0;

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

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

char buf[BUF\_SIZE];

int len;

while((len = recv(clients[id].sock, buf, BUF\_SIZE, 0)) > 0) {

buf[len] = '\0';

for(int i=0; i<client\_count; i++) {

if(i != id) {

send(clients[i].sock, buf, strlen(buf), 0);

}

}

}

close(clients[id].sock);

clients[id].sock = -1;

return NULL;

}

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

int server\_sock, client\_sock, c;

struct sockaddr\_in server, client;

pthread\_t client\_thread;

int \*thread\_id;

if(argc != 2) {

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

return 1;

}

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

if(server\_sock == -1) {

printf("Could not create socket");

return 1;

}

server.sin\_family = AF\_INET;

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

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

if(bind(server\_sock, (struct sockaddr \*)&server, sizeof(server)) < 0) {

printf("Bind failed");

return 1;

}

listen(server\_sock, 3);

printf("Waiting for incoming connections...\n");

c = sizeof(struct sockaddr\_in);

while((client\_sock = accept(server\_sock, (struct sockaddr \*)&client, (socklen\_t\*)&c))) {

printf("Connection accepted\n");

clients[client\_count].sock = client\_sock;

clients[client\_count].id = client\_count;

sprintf(clients[client\_count].name, "Client %d", client\_count);

thread\_id = malloc(sizeof(int));

\*thread\_id = client\_count;

pthread\_create(&client\_thread, NULL, handle\_client, (void \*)thread\_id);

client\_count++;

}

if(client\_sock < 0) {

printf("Accept failed");

return 1;

}

return 0;

}

**client.c**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <pthread.h>

#define MAX\_MESSAGE\_LENGTH 256

void \*receive\_message(void \*socket\_desc)

{

int sock = \*(int\*)socket\_desc;

char message[MAX\_MESSAGE\_LENGTH];

int read\_size;

while((read\_size = recv(sock, message, MAX\_MESSAGE\_LENGTH, 0)) > 0)

{

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

printf("Received message: %s\n", message);

}

if(read\_size == 0)

{

puts("Server disconnected");

fflush(stdout);

}

else if(read\_size == -1)

{

perror("recv failed");

}

free(socket\_desc);

return 0;

}

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

{

int sock;

struct sockaddr\_in server;

char message[MAX\_MESSAGE\_LENGTH];

if(argc < 3)

{

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

exit(1);

}

//Create socket

if((sock = socket(AF\_INET, SOCK\_STREAM, 0)) == -1)

{

perror("Could not create socket");

exit(1);

}

server.sin\_addr.s\_addr = inet\_addr(argv[1]);

server.sin\_family = AF\_INET;

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

//Connect to remote server

if(connect(sock, (struct sockaddr \*)&server, sizeof(server)) < 0)

{

perror("Connect failed");

exit(1);

}

puts("Connected to server");

pthread\_t receive\_thread;

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

\*new\_sock = sock;

if(pthread\_create(&receive\_thread, NULL, receive\_message, (void \*)new\_sock) < 0)

{

perror("Could not create receive thread");

exit(1);

}

while(1)

{

printf("Enter message: ");

fgets(message, MAX\_MESSAGE\_LENGTH, stdin);

if(send(sock, message, strlen(message), 0) < 0)

{

perror("Send failed");

exit(1);

}

}

close(sock);

return 0;

}