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»

Выполнил:

студент гр. 153504

Сивый А. А.

Проверил:

Гриценко Н. Ю.

Минск 2024

СОДЕРЖАНИЕ

[1 Цель работы 3](#_Toc146631498)

[2 Краткие теоретические сведения 4](#_Toc146631499)

[3 Полученные результаты 5](#_Toc146631500)

[Выводы 6](#_Toc146631501)

[Список использованных источников 7](#_Toc146631502)

[Приложение А (обязательное) Листинг исходного кода 8](#_Toc146631503)

## 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.

## 2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Сетевое программирование в Unix-подобных системах представляет собой область, которая занимается созданием и управлением сетевыми соединениями. Это достигается с помощью набора функций и протоколов, которые предоставляются операционной системой.

Основным элементом сетевого программирования в Unix является сокет. Сокет – это абстрактный объект, который предоставляет интерфейс для отправки и получения данных через сеть. В Unix сокеты могут быть созданы для различных типов сетевых протоколов, включая TCP, UDP и Unix domain sockets.

TCP, или протокол управления передачей, является надежным протоколом, который обеспечивает упорядоченную и безошибочную доставку данных. Он используется в большинстве веб-приложений и служб.

UDP, или протокол пользовательских датаграмм, является протоколом, который не гарантирует доставку данных. Он обычно используется в приложениях, где скорость важнее надежности, таких как видео- или аудиостриминг.

Unix domain sockets – это специальный тип сокетов, который используется для обмена данными между процессами на одной и той же машине. Они обеспечивают высокую производительность и низкую задержку по сравнению с сетевыми сокетами [1].

Важным аспектом сетевого программирования в Unix является модель клиент-сервер. В этой модели сервер создает сокет и прослушивает входящие соединения. Клиент, в свою очередь, создает свой сокет и подключается к серверу для обмена данными.

Сетевое программирование на C в Unix в основном основано на использовании системных вызовов для работы с сокетами. Среди наиболее важных системных вызовов можно выделить:

1 socket() – это системный вызов, который используется для создания нового сокета.

2 bind() – это системный вызов, который связывает сокет с конкретным адресом и портом.

3 listen() – это системный вызов, использующийся сервером для прослушивания входящих соединений.

4 accept() – используется для принятия входящих соединений.

5 connect() – используется клиентом для установления соединения с сервером.

6 send(), recv(), read(), write() – используются для отправки и получения данных через сокет.

7 close() – используется для закрытия сокета.

Системные вызовы должны использоваться в правильном порядке и с правильной обработкой ошибок для корректной работы сетевого приложения.

## 3 ПОЛУЧЕННЫЕ РЕЗУЛЬТАТЫ

В ходе данной лабораторной работы был разработан программный продукт для интерактивного взаимодействия пользователей (чат), представленный на рисунке 3.1.

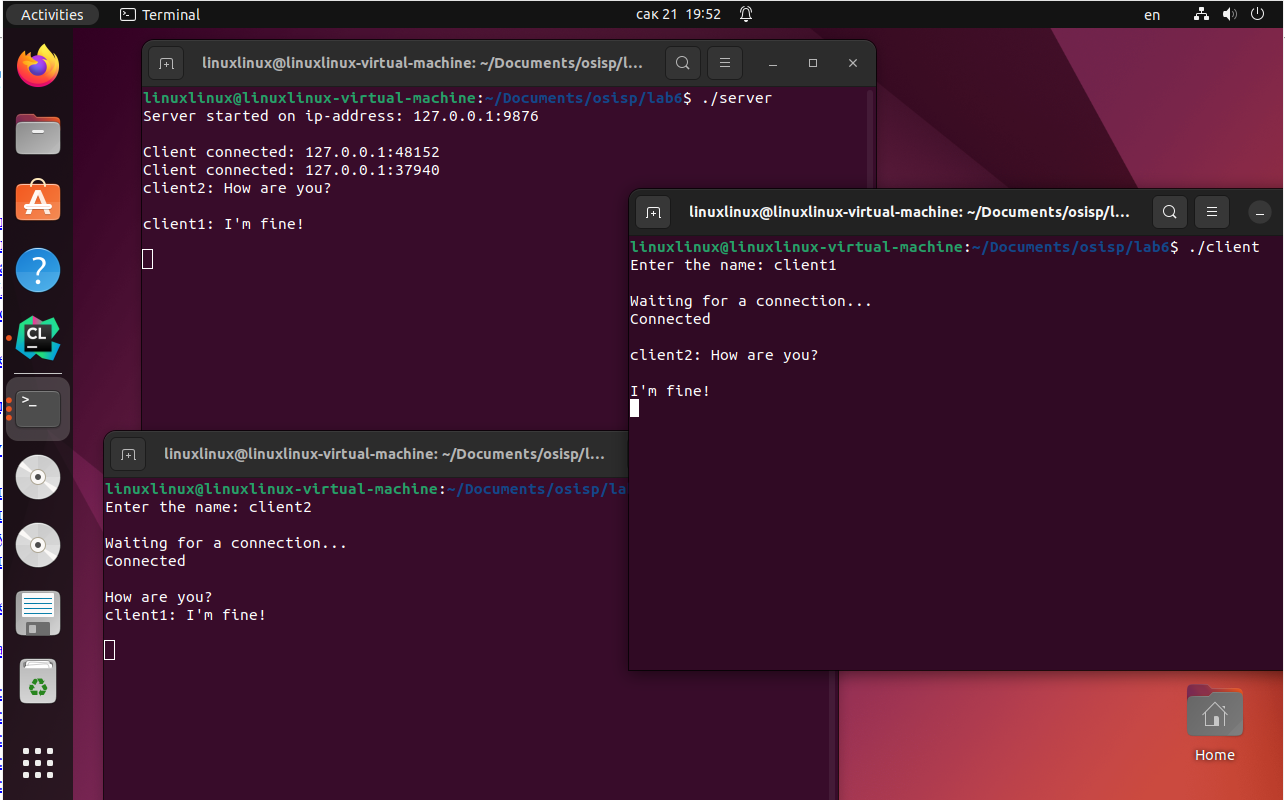


Рисунок 3.1 – Результат работы программы

Данный программный продукт использует TCP протокол для передачи информации по сети. Приложение использует централизованную архитектуру, то есть имеется выделенный процесс-сервер и процесс-клиенты. Сервер выполняет функции приема соединений, временного хранения, а также передачи сообщений клиентам. Клиент подключается к серверу, осуществляет ввод пользовательских сообщений и передает их серверу.

## ВЫВОДЫ

В ходе выполнения данной лабораторной работы было реализовано приложение для интерактивного взаимодействия пользователей (чат) на языке программирования С, используя элементы сетевого программирования. Данная лабораторная работа позволила улучшить навыки программирования на языке С, а также расширить знания в области сетевого программирования.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Сетевые средства UNIX [Электронный ресурс] – Режим доступа: [https://uneex.ru/static/AltDocs\_unix\_base\_admin/](https://uneex.ru/static/AltDocs_unix_base_admin/ch04.html) – Дата доступа: 15.03.2024

[2] Руководство по Си [Электронный ресурс] – Режим доступа: <https://metanit.com/c/tutorial/> – Дата доступа: 01.03.2024

**ПРИЛОЖЕНИЕ А**

## (обязательное)

## Листинг исходного кода

Листинг 1 – Программный код разработанного приложения

#include <stdio.h>

#include <stdlib.h>

#include <sys/socket.h>

#include <arpa/inet.h>

#include <pthread.h>

#include <string.h>

typedef struct

{

    int \*sock\_client;

    int \*count;

    int client;

} args\_t;

void \*receive\_send(void \*data)

{

    args\_t \*arguments = (args\_t\*)data;

    char message[200];

    while(1)

    {

        int size = recv(arguments->client, message, 200, 0);

        if (size > 0)

        {

            message[size] = '\0';

            printf("%s\n", message);

            for(int i = 0; i < \*arguments->count; i++)

            {

                if (arguments->sock\_client[i] != arguments->client)

                {

                    send(arguments->sock\_client[i], message, strlen(message), 0);

                }

            }

        }

        if (size == 0)

        {

            break;

        }

    }

    return NULL;

}

int main()

{

    int socket\_descriptor = socket(AF\_INET, SOCK\_STREAM, 0); // 0 is protocol value for IP

    if (socket\_descriptor == -1)

    {

        printf("Error: invalid socket\n");

        return -1;

    }

    struct sockaddr\_in server;

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

    server.sin\_port = htons(9876);

    server.sin\_family = AF\_INET;

    if (bind(socket\_descriptor, (struct sockaddr \*)&server, sizeof(server)) == -1)

    {

        printf("Error: failed connection\n");

        return -1;

    }

    listen(socket\_descriptor, 10);

    char \*server\_address = inet\_ntoa(server.sin\_addr);

    int server\_port = ntohs(server.sin\_port);

    printf("Server started on ip-address: %s:%d\n\n", server\_address, server\_port);

    int \*sock\_client = (int\*)malloc(sizeof(int));

    int count = 0;

    while (1)

    {

        struct sockaddr\_in client;

        int c = sizeof(struct sockaddr\_in);

        sock\_client[count] = accept(socket\_descriptor, (struct sockaddr \*)&client, (socklen\_t \*)&c);

        char \*client\_address = inet\_ntoa(client.sin\_addr);

        int client\_port = ntohs(client.sin\_port);

        if (sock\_client[count] != -1)

        {

            printf("Client connected: %s:%d\n", client\_address, client\_port);

            args\_t \*arguments = (args\_t\*)malloc(sizeof(args\_t));

            arguments->sock\_client = sock\_client;

            arguments->count = &count;

            arguments->client = sock\_client[count];

            pthread\_t thread;

            pthread\_create(&thread, NULL, receive\_send, arguments);

        }

        count++;

        sock\_client = (int\*)realloc(sock\_client, sizeof(int) \* (count + 1));

    }

    return 0;

}

#include <stdio.h>

#include <stdlib.h>

#include <sys/socket.h>

#include <arpa/inet.h>

#include <unistd.h>

#include <pthread.h>

#include <string.h>

typedef struct {

    int server;

} args\_t;

void\* receive\_msg(void\* data) {

    args\_t\* args = (args\_t\*)data;

    char msg[200];

    while(1) {

        int size = recv(args->server, msg, 200, 0);

        if (size > 0) {

            msg[size] = '\0';

            printf("%s\n", msg);

        }

        if (size == 0) {

            printf("Connection lost\n");

            break;

        }

    }

    return NULL;

}

int main() {

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

    if (socket\_desc == -1) {

        printf("Error: invalid socket\n");

        return -1;

    }

    struct sockaddr\_in server;

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

    server.sin\_port = htons(9876);

    server.sin\_family = AF\_INET;

    char username[11];

    printf("Enter the name: ");

    scanf("%s", username);

    printf("\nWaiting for a connection...\n");

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

        printf("Error: failed connection\n");

        return -1;

    } else {

        printf("Connected\n\n");

    }

    args\_t\* args = (args\_t\*)malloc(sizeof(args\_t));

    args->server = socket\_desc;

    pthread\_t thread;

    pthread\_create(&thread, NULL, receive\_msg, args);

    while(1) {

        char text[100];

        char msg[200];

        fgets(text, 100, stdin);

        if (strcmp(text, "/q") == 0) {

            break;

        }

        if(text[0] != '\n'){

            sprintf(msg, "%s: %s", username, text);

            send(socket\_desc, msg, strlen(msg), 0);

        }

    }

    close(socket\_desc);

    return 0;

}