МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

## Факультет информационных технологий и робототехники

Кафедра программного обеспечения информационных систем и технологий

**Отчет по лабораторной работе № 10**

по дисциплине:” Системное программирование”

на тему: “РАБОТА С КАНАЛАМИ”

Выполнил**:** студент группы 10701321

Грудинский К. А.

Принял**:** Кондратёнок Е.В.

Минск 2023

# Лабораторная работа № 11. Клиент-серверное сокетное соединение в сети.

# Цель работы: Изучить механизм сокетного сетевого взаимодействия в LINUX

# Задание 1

Задание 1 – Потоковые сокеты

Напишите программы с клиент-серверной архитектурой по следующим требованиям:

1. Обменяться числами. Сервер возводит получаемое от клиента число в

квадрат и возвращает ответ клиенту.

2. Создать два клиента на один и тот же сокет. Используют один и тот же

сокет. Первый клиент получает результат счета от 1 до 10, второй – результат счета от 100 до 110 (подумайте, как распознать, какой клиент

обратился к серверу).

Примечание. Если количество студентов команды, выполняющих задание больше двух, то

число процессов-клиентов должно соответствовать числу студентов команды.

Если клиентов больше двух, то в этом случае процессы должны дополнительно выводить

фамилии студентов, выполнивших задания. Третий клиент получает результат счета от 200

до 210. Каждому последующему клиенту сообщение увеличивается на 100.

3. Создать два клиента на один и тот же сервер. Сокеты должны быть разными (именованное соединение в каждом случае должно быть различным). Первый клиент получает результат счета от 1 до 10, второй – от

100 до 110.

**Решение**

**1.**

**сервер**

#include <stdio.h>

#include <sys/socket.h>

#include <sys/types.h>

#include <string.h>

#include <sys/un.h>

#include <stdlib.h>

#include <math.h> // Добавлен заголовочный файл для использования функции pow()

#define QUEUE\_LENGTH 10

#define BUF\_LEN 256

#define SOCK\_NAME "mysocket"

int main(void) {

int sock, client\_sock;

struct sockaddr\_un saddr;

char \*buf;

int count;

sock = socket(PF\_UNIX, SOCK\_STREAM, 0);

if (sock == -1) {

fprintf(stderr, "socket() error\n");

return 1;

}

buf = (char \*)malloc(BUF\_LEN);

if (buf == NULL) {

fprintf(stderr, "malloc() error\n");

return 1;

}

saddr.sun\_family = AF\_UNIX;

strcpy(saddr.sun\_path, SOCK\_NAME);

if (bind(sock, (struct sockaddr \*)&saddr, SUN\_LEN(&saddr)) == -1) {

fprintf(stderr, "bind() error\n");

return 1;

}

if (listen(sock, QUEUE\_LENGTH) == -1) {

fprintf(stderr, "listen() error\n");

return 0;

}

while (1) {

client\_sock = accept(sock, NULL, NULL);

if (client\_sock == -1) {

fprintf(stderr, "accept() error\n");

return 1;

}

if ((count = read(client\_sock, buf, BUF\_LEN - 1)) == -1) {

fprintf(stderr, "read() error\n");

return 1;

}

buf[count] = '\0';

// Преобразование строки в число, возведение в квадрат и обратное преобразование в строку

double number = atof(buf);

double square = pow(number, 2);

snprintf(buf, BUF\_LEN, "%f", square);

if (write(client\_sock, buf, strlen(buf)) == -1) {

fprintf(stderr, "write() error\n");

return 1;

}

close(client\_sock);

if (!strcmp(buf, "exit"))

break;

}

free(buf);

close(sock);

unlink(SOCK\_NAME);

return 0;

}

**клиент**

#include <stdio.h>

#include <sys/socket.h>

#include <sys/un.h>

#include <unistd.h>

#include <string.h>

#include <stdlib.h> // Добавлен заголовочный файл для использования функции atof()

#define SOCK\_NAME "mysocket"

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

int sock;

struct sockaddr\_un addr;

if (argc < 2) {

fprintf(stderr, "Too few arguments\n");

return 1;

}

sock = socket(PF\_UNIX, SOCK\_STREAM, 0);

if (sock == -1) {

fprintf(stderr, "socket() error\n");

return 1;

}

addr.sun\_family = AF\_UNIX;

strcpy(addr.sun\_path, SOCK\_NAME);

if (connect(sock, (struct sockaddr \*)&addr, SUN\_LEN(&addr)) == -1) {

fprintf(stderr, "connect() error\n");

return 1;

}

if (write(sock, argv[1], strlen(argv[1])) == -1) {

fprintf(stderr, "write() error\n");

return 1;

}

// Чтение ответа от сервера

char response[256];

if (read(sock, response, sizeof(response) - 1) == -1) {

fprintf(stderr, "read() error\n");

return 1;

}

response[sizeof(response) - 1] = '\0';

printf("Server response: %s\n", response);

close(sock);

return 0;

}



Рисунок 1 – задание 1.1

**1.2**

**сервер**

#include <stdio.h>

#include <sys/socket.h>

#include <sys/types.h>

#include <string.h>

#include <sys/un.h>

#include <stdlib.h>

#include <unistd.h>

#define QUEUE\_LENGTH 10

#define BUF\_LEN 256

#define SOCK\_NAME "mysocket"

void send\_numbers(int client\_sock, int start, int end) {

char buf[BUF\_LEN];

for (int i = start; i <= end; ++i) {

snprintf(buf, BUF\_LEN, "%d", i);

if (write(client\_sock, buf, strlen(buf)) == -1) {

fprintf(stderr, "write() error\n");

exit(1);

}

sleep(1); // Задержка между отправкой чисел (для демонстрации)

}

}

int main(void) {

int sock, client\_sock;

struct sockaddr\_un saddr;

char \*buf;

sock = socket(PF\_UNIX, SOCK\_STREAM, 0);

if (sock == -1) {

fprintf(stderr, "socket() error\n");

return 1;

}

buf = (char \*)malloc(BUF\_LEN);

if (buf == NULL) {

fprintf(stderr, "malloc() error\n");

return 1;

}

saddr.sun\_family = AF\_UNIX;

strcpy(saddr.sun\_path, SOCK\_NAME);

if (bind(sock, (struct sockaddr \*)&saddr, SUN\_LEN(&saddr)) == -1) {

fprintf(stderr, "bind() error\n");

return 1;

}

if (listen(sock, QUEUE\_LENGTH) == -1) {

fprintf(stderr, "listen() error\n");

return 0;

}

while (1) {

client\_sock = accept(sock, NULL, NULL);

if (client\_sock == -1) {

fprintf(stderr, "accept() error\n");

return 1;

}

// Определение клиента по дескриптору сокета

int client\_id;

if (client\_sock % 2 == 0) {

client\_id = 1; // Четные дескрипторы для первого клиента

} else {

client\_id = 2; // Нечетные дескрипторы для второго клиента

}

// Отправка чисел соответствующему клиенту

if (client\_id == 1) {

send\_numbers(client\_sock, 1, 10);

} else {

send\_numbers(client\_sock, 101, 110);

}

close(client\_sock);

// Если получен запрос с "exit", завершаем сервер

if (!strcmp(buf, "exit"))

break;

}

free(buf);

close(sock);

unlink(SOCK\_NAME);

return 0;

}

**клиент**

#include <stdio.h>

#include <sys/socket.h>

#include <sys/types.h>

#include <string.h>

#include <sys/un.h>

#include <stdlib.h>

#include <unistd.h>

#define QUEUE\_LENGTH 10

#define BUF\_LEN 256

#define SOCK\_NAME "mysocket"

int main(void)

{

int sock, client\_sock;

struct sockaddr\_un saddr;

char \*buf;

int count;

sock = socket(PF\_UNIX, SOCK\_STREAM, 0);

if (sock == -1)

{

fprintf(stderr, "socket() error\n");

return 1;

}

buf = (char \*)malloc(BUF\_LEN);

if (buf == NULL)

{

fprintf(stderr, "malloc() error\n");

return 1;

}

saddr.sun\_family = AF\_UNIX;

strcpy(saddr.sun\_path, SOCK\_NAME);

if (bind(sock, (struct sockaddr \*)&saddr, SUN\_LEN(&saddr)) == -1)

{

fprintf(stderr, "bind() error\n");

return 1;

}

if (listen(sock, QUEUE\_LENGTH) == -1)

{

fprintf(stderr, "listen() error\n");

return 1;

}

while (1)

{

client\_sock = accept(sock, NULL, NULL);

if (client\_sock == -1)

{

fprintf(stderr, "accept() error\n");

return 1;

}

if ((count = read(client\_sock, buf, BUF\_LEN - 1)) == -1)

{

fprintf(stderr, "read() error\n");

return 1;

}

buf[count] = '\0';

if (!strcmp(buf, "1"))

{

for (int i = 1; i <= 10; ++i)

{

snprintf(buf, BUF\_LEN, "%d", i);

write(client\_sock, buf, strlen(buf));

sleep(1); // Optional: add a delay between numbers

}

}

else if (!strcmp(buf, "2"))

{

for (int i = 101; i <= 110; ++i)

{

snprintf(buf, BUF\_LEN, "%d", i);

write(client\_sock, buf, strlen(buf));

sleep(1); // Optional: add a delay between numbers

}

}

close(client\_sock);

if (!strcmp(buf, "exit"))

break;

}

free(buf);

close(sock);

unlink(SOCK\_NAME);

return 0;

}

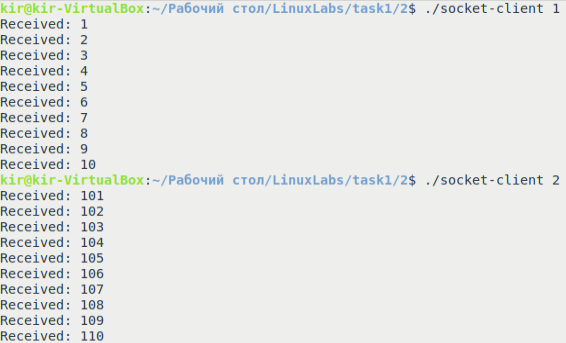


Рисунок 2 – задание 1.2

**1.3**

**сервер**

#include <stdio.h>

#include <sys/socket.h>

#include <sys/types.h>

#include <string.h>

#include <sys/un.h>

#include <stdlib.h>

#define QUEUE\_LENGTH 10

#define BUF\_LEN 256

#define SOCK\_NAME1 "mysocket1"

#define SOCK\_NAME2 "mysocket2"

int main(void)

{

int sock1, sock2, client\_sock;

struct sockaddr\_un saddr1, saddr2;

char \*buf;

int count;

// Создание первого сокета

sock1 = socket(PF\_UNIX, SOCK\_STREAM, 0);

if (sock1 == -1)

{

fprintf(stderr, "socket() error\n");

return 1;

}

// Создание второго сокета

sock2 = socket(PF\_UNIX, SOCK\_STREAM, 0);

if (sock2 == -1)

{

fprintf(stderr, "socket() error\n");

return 1;

}

buf = (char \*)malloc(BUF\_LEN);

if (buf == NULL)

{

fprintf(stderr, "malloc() error\n");

return 1;

}

// Настройка адреса для первого сокета

saddr1.sun\_family = AF\_UNIX;

strcpy(saddr1.sun\_path, SOCK\_NAME1);

// Настройка адреса для второго сокета

saddr2.sun\_family = AF\_UNIX;

strcpy(saddr2.sun\_path, SOCK\_NAME2);

// Привязка первого сокета

if (bind(sock1, (struct sockaddr \*)&saddr1, SUN\_LEN(&saddr1)) == -1)

{

fprintf(stderr, "bind() error\n");

return 1;

}

// Привязка второго сокета

if (bind(sock2, (struct sockaddr \*)&saddr2, SUN\_LEN(&saddr2)) == -1)

{

fprintf(stderr, "bind() error\n");

return 1;

}

// Начало прослушивания для первого сокета

if (listen(sock1, QUEUE\_LENGTH) == -1)

{

fprintf(stderr, "listen() error\n");

return 1;

}

// Начало прослушивания для второго сокета

if (listen(sock2, QUEUE\_LENGTH) == -1)

{

fprintf(stderr, "listen() error\n");

return 1;

}

while (1)

{

// Принятие соединения для первого сокета

client\_sock = accept(sock1, NULL, NULL);

if (client\_sock == -1)

{

fprintf(stderr, "accept() error\n");

return 1;

}

// Посылка чисел от 1 до 10 клиенту

for (int i = 1; i <= 10; ++i)

{

snprintf(buf, BUF\_LEN, "%d", i);

write(client\_sock, buf, strlen(buf));

sleep(1); // Пауза между числами

}

close(client\_sock);

free(buf);

// Принятие соединения для второго сокета

client\_sock = accept(sock2, NULL, NULL);

if (client\_sock == -1)

{

fprintf(stderr, "accept() error\n");

return 1;

}

// Посылка чисел от 101 до 110 клиенту

for (int i = 101; i <= 110; ++i)

{

snprintf(buf, BUF\_LEN, "%d", i);

write(client\_sock, buf, strlen(buf));

sleep(1); // Пауза между числами

}

close(client\_sock);

free(buf);

break; // Выход из бесконечного цикла после обслуживания обоих клиентов

}

close(sock1);

close(sock2);

unlink(SOCK\_NAME1);

unlink(SOCK\_NAME2);

return 0;

}

**клиент**

#include <stdio.h>

#include <sys/socket.h>

#include <sys/un.h>

#include <unistd.h>

#include <string.h>

#define SOCK\_NAME1 "mysocket1"

#define SOCK\_NAME2 "mysocket2"

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

{

int sock;

struct sockaddr\_un addr;

if (argc < 2)

{

fprintf(stderr, "Too few arguments\n");

return 1;

}

// Выбор сокета в зависимости от порта

const char \*socket\_name = (strcmp(argv[1], "1") == 0) ? SOCK\_NAME1 : SOCK\_NAME2;

sock = socket(PF\_UNIX, SOCK\_STREAM, 0);

if (sock == -1)

{

fprintf(stderr, "socket() error\n");

return 1;

}

addr.sun\_family = AF\_UNIX;

strcpy(addr.sun\_path, socket\_name);

if (connect(sock, (struct sockaddr \*)&addr, SUN\_LEN(&addr)) == -1)

{

fprintf(stderr, "connect() error\n");

return 1;

}

// Чтение данных от сервера

char buffer[256];

ssize\_t bytesRead;

while ((bytesRead = read(sock, buffer, sizeof(buffer) - 1)) > 0)

{

buffer[bytesRead] = '\0';

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

}

close(sock);

return 0;

}

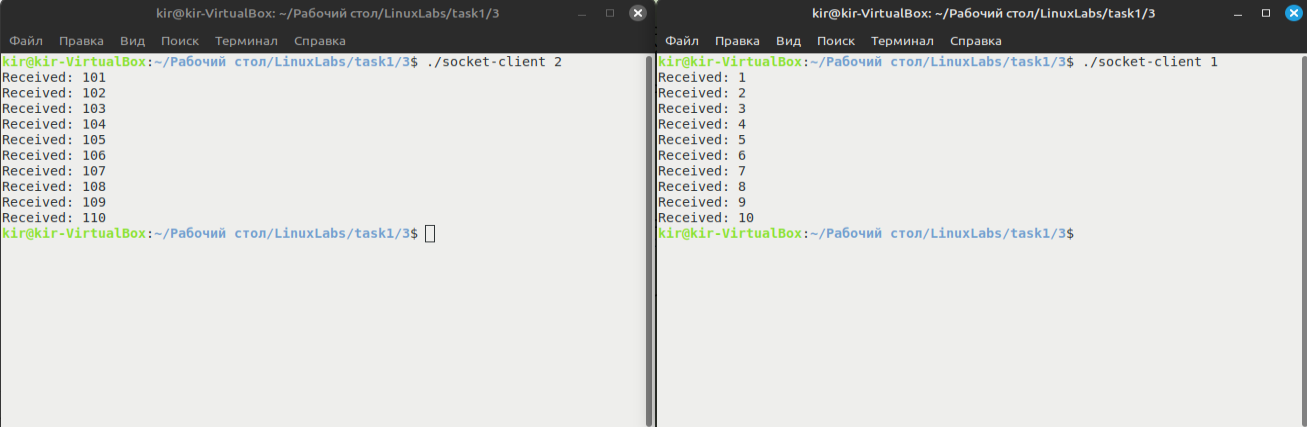


Рисунок 3 – задание 1.3

**Задание 2.**

Напишите программу, которая читает главную страницу из какого-нибудь

Web-сервера. Запуск программы должен производиться с консоли.

Варианты, простые числа – подключиться к странице сайта БНТУ, ФИТР; четные

числа – подключиться к странице сайта БНТУ, расписание занятий

3-го курса; кратные трем – подключиться к странице сайта БНТУ, абитуриент, поступление.

**Решение**

**Задание 2.1**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <netdb.h>

#include <unistd.h>

#include <openssl/ssl.h>

#include <openssl/err.h>

#define BUF\_LEN 4096

#define HTTPS\_PORT 443

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

int sock, count;

char \*buf;

struct hostent \*host;

struct sockaddr\_in addr;

SSL\_CTX \*ctx;

SSL \*ssl;

if (argc < 2) {

fprintf(stderr, "Too few arguments\n");

return 1;

}

// Extract hostname from the URL

char \*url = argv[1] + strlen("https://");

char \*slash = strchr(url, '/');

if (slash != NULL) {

\*slash = '\0'; // Null-terminate the hostname

}

buf = (char \*)malloc(BUF\_LEN);

if (buf == NULL) {

fprintf(stderr, "malloc() error\n");

return 1;

}

// Initialize OpenSSL

SSL\_library\_init();

ctx = SSL\_CTX\_new(SSLv23\_client\_method());

if (ctx == NULL) {

fprintf(stderr, "SSL\_CTX\_new() error\n");

return 1;

}

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

if (sock == -1) {

fprintf(stderr, "socket() error\n");

return 1;

}

addr.sin\_family = AF\_INET;

host = gethostbyname(url);

if (host == NULL) {

fprintf(stderr, "Unknown server\n");

return 1;

}

memcpy(&addr.sin\_addr, host->h\_addr\_list[0], host->h\_length);

addr.sin\_port = htons(HTTPS\_PORT);

if (connect(sock, (struct sockaddr\*)&addr, sizeof(addr)) == -1) {

fprintf(stderr, "connect() error\n");

return 1;

}

// Set up SSL connection

ssl = SSL\_new(ctx);

SSL\_set\_fd(ssl, sock);

if (SSL\_connect(ssl) == -1) {

fprintf(stderr, "SSL\_connect() error\n");

return 1;

}

// Send HTTPS request

char request[BUF\_LEN];

snprintf(request, BUF\_LEN, "GET / HTTP/1.1\r\nHost: %s\r\nConnection: close\r\n\r\n", url);

SSL\_write(ssl, request, strlen(request));

// Receive and print the response

while ((count = SSL\_read(ssl, buf, BUF\_LEN)) > 0) {

write(1, buf, count);

}

// Clean up

SSL\_free(ssl);

close(sock);

free(buf);

SSL\_CTX\_free(ctx);

return 0;

}

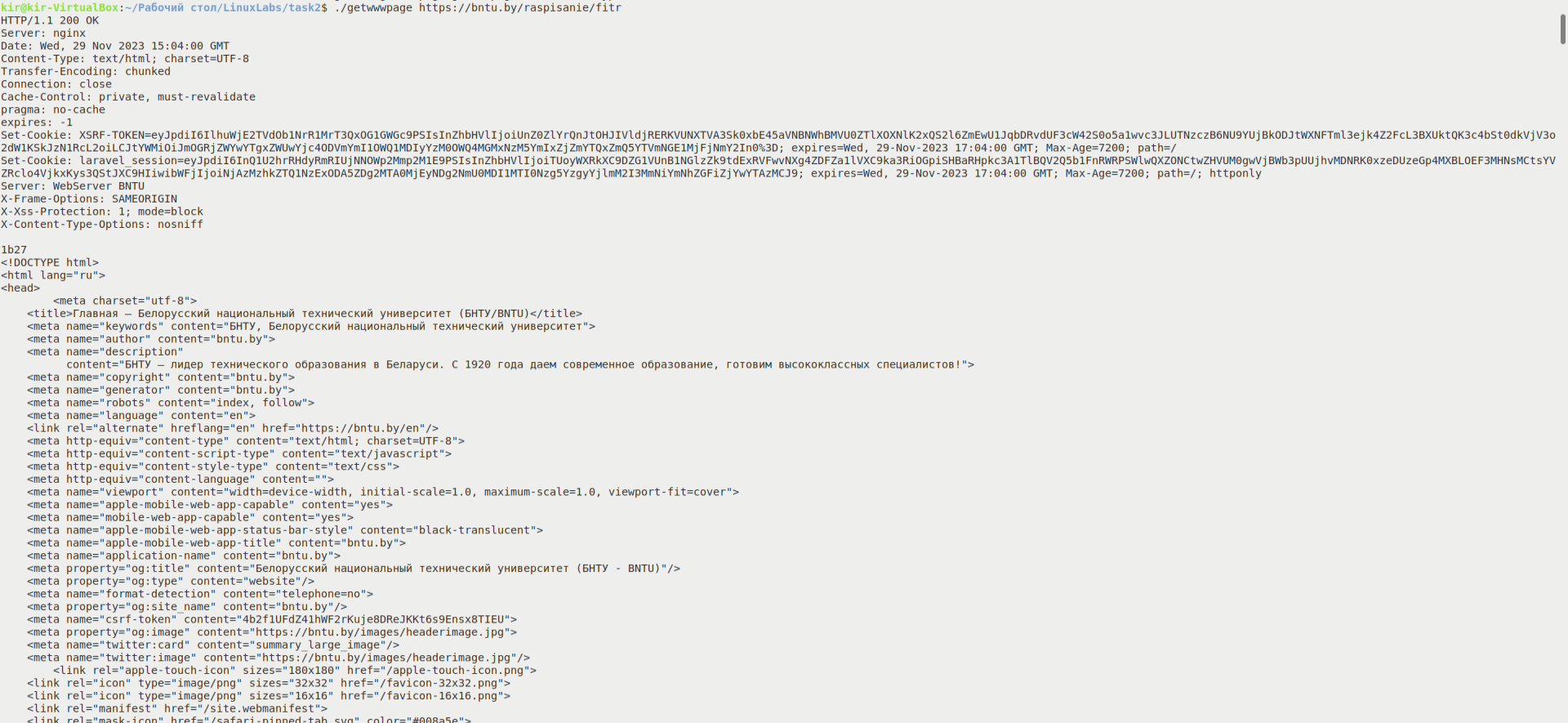


Рисунок 4 – задание 2

**Задание 3**

Напишите программу с клиент-серверной архитектурой для передачи данных

через дейтаграммные сокеты. Количество клиентов должно соответствовать

количеству членов вашей команды, но не менее двух. Семейства протоколов

и способы взаимодействия сокетов-клиентов должны отличаться.

Запустите программы в разных терминалах и определите какие программы клиенты получили данные. Объясните полученный результат.

**Решение**

**сервер**

*#include <stdio.h>*

*#include <sys/socket.h>*

*#include <sys/types.h>*

*#include <string.h>*

*#include <sys/un.h>*

*#include <stdlib.h>*

*#include <arpa/inet.h>*

*#define QUEUE\_LENGTH 10*

*#define BUF\_LEN 256*

*#define SOCK\_NAME "mysocket"*

*int main(void) {*

*int sock;*

*int count;*

*struct sockaddr\_un saddr\_un;*

*struct sockaddr\_in saddr\_in;*

*char \*buf;*

*sock = socket(PF\_UNIX, SOCK\_DGRAM, 0);*

*if (sock == -1) {*

*fprintf(stderr, "socket() error\n");*

*return 1;*

*}*

*buf = (char \*)malloc(BUF\_LEN);*

*if (buf == NULL) {*

*fprintf(stderr, "malloc() error\n");*

*return 1;*

*}*

*saddr\_un.sun\_family = AF\_UNIX;*

*strcpy(saddr\_un.sun\_path, SOCK\_NAME);*

*saddr\_in.sin\_family = AF\_INET;*

*saddr\_in.sin\_addr.s\_addr = htonl(INADDR\_ANY);*

*saddr\_in.sin\_port = htons(12345);*

*if (bind(sock, (struct sockaddr \*)&saddr\_un, SUN\_LEN(&saddr\_un)) == -1) {*

*fprintf(stderr, "bind() error\n");*

*return 1;*

*}*

*while (1) {*

*if ((count = recvfrom(sock, buf, BUF\_LEN - 1, 0, NULL, NULL)) == -1) {*

*fprintf(stderr, "recvfrom() error\n");*

*return 1;*

*}*

*buf[count] = '\0';*

*printf(">> %s\n", buf);*

*if (!strcmp(buf, "exit"))*

*break;*

*}*

*free(buf);*

*close(sock);*

*unlink(SOCK\_NAME);*

*return 0;*

*}*

**клиент 1**

#include <stdio.h>

#include <sys/socket.h>

#include <sys/un.h>

#include <unistd.h>

#include <string.h>

#define SOCK\_NAME "mysocket"

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

int sock;

struct sockaddr\_un addr;

if (argc < 2) {

fprintf(stderr, "Too few arguments\n");

return 1;

}

sock = socket(PF\_UNIX, SOCK\_DGRAM, 0);

if (sock == -1) {

fprintf(stderr, "socket() error\n");

return 1;

}

addr.sun\_family = AF\_UNIX;

strcpy(addr.sun\_path, SOCK\_NAME);

if (sendto(sock, argv[1], strlen(argv[1]), 0, (struct sockaddr \*)&addr, SUN\_LEN(&addr)) == -1) {

fprintf(stderr, "sendto() error\n");

return 1;

}

close(sock);

return 0;

}

**клиент 2**

#include <stdio.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <unistd.h>

#include <string.h>

#define SOCK\_NAME "mysocket"

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

int sock;

struct sockaddr\_in addr;

if (argc < 2) {

fprintf(stderr, "Too few arguments\n");

return 1;

}

sock = socket(PF\_INET, SOCK\_DGRAM, 0);

if (sock == -1) {

fprintf(stderr, "socket() error\n");

return 1;

}

addr.sin\_family = AF\_INET;

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

addr.sin\_port = htons(12345);

if (sendto(sock, argv[1], strlen(argv[1]), 0, (struct sockaddr \*)&addr, sizeof(addr)) == -1) {

fprintf(stderr, "sendto() error\n");

return 1;

}

close(sock);

return 0;

}





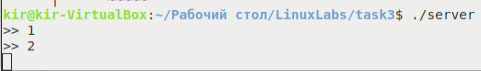


Рисунок 6 – задание 3

Объяснение результата: сервер принимает сообщения от обоих клиентов и выводит их на экран. Каждый клиент использует свой собственный семейство протоколов, но сервер способен обрабатывать их оба.

Вывод*:* В ходе выполнения лабораторной работы был освоена работа с сокетами.

Контрольные вопросы

2. Как переводится слово «socket» с английского на русском язык?

socket – розетка, гнездо, разъем.