МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

## Факультет информационных технологий и робототехники

Кафедра программного обеспечения информационных систем и технологий

**Отчёт по лабораторной работе № 32(13)**

по дисциплине: ”Системное программирование”

на тему: «***Удаленное***

***межпроцессное***

***взаимодействие. Сокеты»***

Вариант -

Выполнил**:** студент группы 10701222 Зухта К. М.

Медведский Е. В.

Принял**:** пр. Давыденко Н.В.

# Лабораторная работа № 32.

## Задание 1 – Потоковые сокеты

Напишите программы с клиент-серверной архитектурой по следующим тре-

бованиям:

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 – ре-

зультат счета от 100 до 110 (подумайте, как распознать, какой клиент

обратился к серверу).

**Листинг кода**

### **server.c**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <string.h>**

**#include <unistd.h>**

**#include <arpa/inet.h>**

**#include <pthread.h>**

**#define PORT 9999**

**#define BUFFER\_SIZE 1024**

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

**int client\_socket = \*(int \*)arg;**

**free(arg);**

**char buffer[BUFFER\_SIZE];**

**while (1) {**

**// Чтение запроса от клиента**

**int bytes\_read = recv(client\_socket, buffer, sizeof(buffer) - 1, 0);**

**if (bytes\_read <= 0) {**

**break; // Если чтение не удалось, выходим**

**}**

**buffer[bytes\_read] = '\0'; // Завершаем строку**

**// Определяем ответ в зависимости от клиента**

**int client\_id = atoi(buffer);**

**int start, end;**

**if (client\_id == 1) {**

**start = 1; end = 10;**

**} else if (client\_id == 2) {**

**start = 100; end = 110;**

**} else if (client\_id == 3) {**

**start = 200; end = 210;**

**} else {**

**snprintf(buffer, sizeof(buffer), "Неизвестный клиент: %d", client\_id);**

**send(client\_socket, buffer, strlen(buffer), 0);**

**continue;**

**}**

**// Формируем ответ**

**snprintf(buffer, sizeof(buffer), "Счет от %d до %d: ", start, end);**

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

**char num[10];**

**sprintf(num, "%d ", i);**

**strcat(buffer, num);**

**}**

**send(client\_socket, buffer, strlen(buffer), 0);**

**}**

**close(client\_socket);**

**return NULL;**

**}**

**int main() {**

**int server\_socket, \*client\_socket;**

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

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

**// Создаем сокет**

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

**if (server\_socket < 0) {**

**perror("Ошибка при создании сокета");**

**exit(EXIT\_FAILURE);**

**}**

**// Настраиваем адрес**

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

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

**server\_addr.sin\_port = htons(PORT);**

**// Привязываем сокет к адресу**

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

**perror("Ошибка привязки сокета");**

**close(server\_socket);**

**exit(EXIT\_FAILURE);**

**}**

**// Слушаем входящие соединения**

**listen(server\_socket, 5);**

**printf("Сервер запущен на порту %d\n", PORT);**

**while (1) {**

**client\_socket = malloc(sizeof(int));**

**\*client\_socket = accept(server\_socket, (struct sockaddr \*)&client\_addr, &client\_addr\_len);**

**if (\*client\_socket < 0) {**

**perror("Ошибка при принятии соединения");**

**free(client\_socket);**

**continue;**

**}**

**printf("Клиент подключен.\n");**

**pthread\_t tid;**

**pthread\_create(&tid, NULL, handle\_client, client\_socket);**

**pthread\_detach(tid); // Автоматически освобождаем поток**

**}**

**close(server\_socket);**

**return 0;**

**}**

### **client.c**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <string.h>**

**#include <unistd.h>**

**#include <arpa/inet.h>**

**#define PORT 9999**

**#define BUFFER\_SIZE 1024**

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

**if (argc != 2) {**

**fprintf(stderr, "Использование: %s <client\_id>\n", argv[0]);**

**exit(EXIT\_FAILURE);**

**}**

**int client\_id = atoi(argv[1]);**

**int client\_socket;**

**struct sockaddr\_in server\_addr;**

**char buffer[BUFFER\_SIZE];**

**// Создаем сокет**

**client\_socket = socket(AF\_INET, SOCK\_STREAM, 0);**

**if (client\_socket < 0) {**

**perror("Ошибка при создании сокета");**

**exit(EXIT\_FAILURE);**

**}**

**// Настраиваем адрес**

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

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

**server\_addr.sin\_port = htons(PORT);**

**// Подключаемся к серверу**

**if (connect(client\_socket, (struct sockaddr \*)&server\_addr, sizeof(server\_addr)) < 0) {**

**perror("Ошибка подключения к серверу");**

**close(client\_socket);**

**exit(EXIT\_FAILURE);**

**}**

**// Отправляем идентификатор клиента**

**snprintf(buffer, sizeof(buffer), "%d", client\_id);**

**send(client\_socket, buffer, strlen(buffer), 0);**

**// Получаем ответ от сервера**

**int bytes\_received = recv(client\_socket, buffer, sizeof(buffer) - 1, 0);**

**if (bytes\_received > 0) {**

**buffer[bytes\_received] = '\0'; // Завершаем строку**

**printf("Ответ от сервера: %s\n", buffer);**

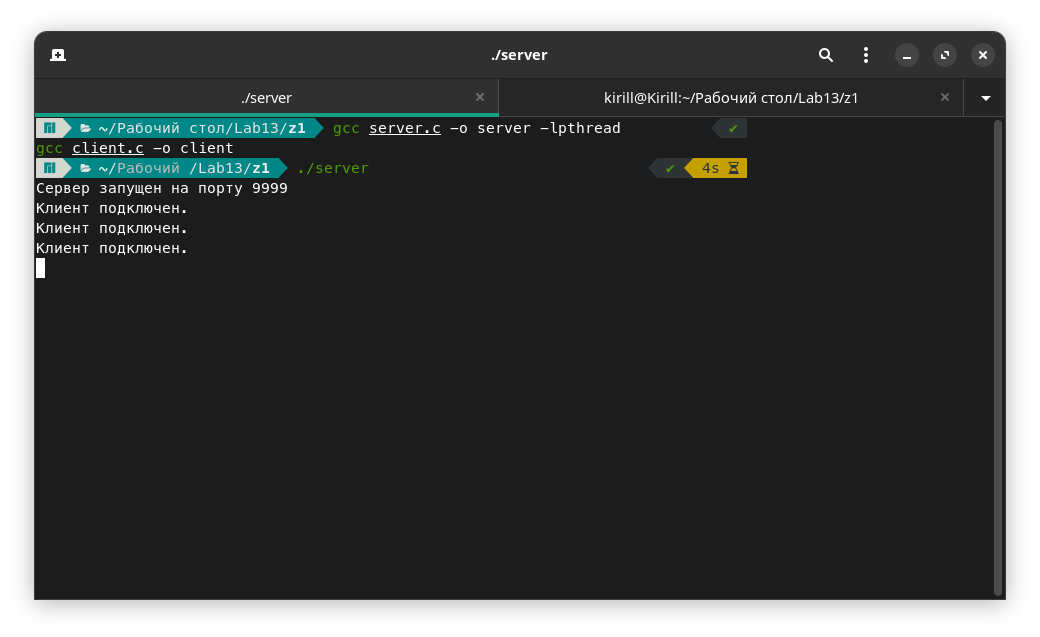
**}**

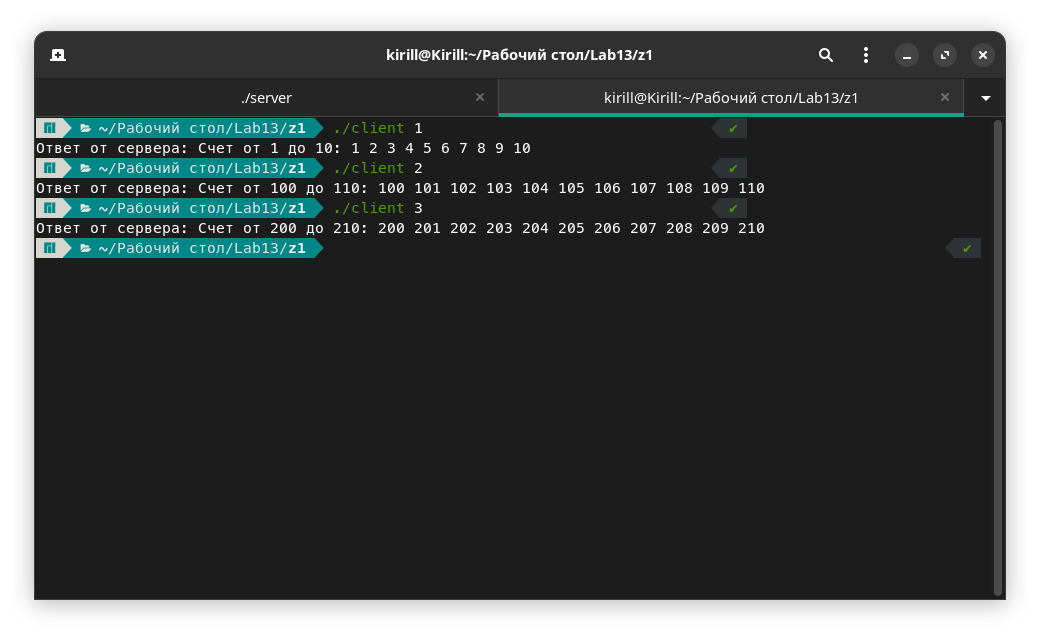
**close(client\_socket);**

**return 0;**

**}**

**Скриншоты**





**Сервер может обрабатывать нескольких клиентов одновременно с помощью потоков.**

* Каждый клиент отправляет свой идентификатор, и сервер отвечает соответствующим результатом.
* Вы можете добавить больше клиентов, просто запуская ./client <id>, где <id> — это номер клиента.

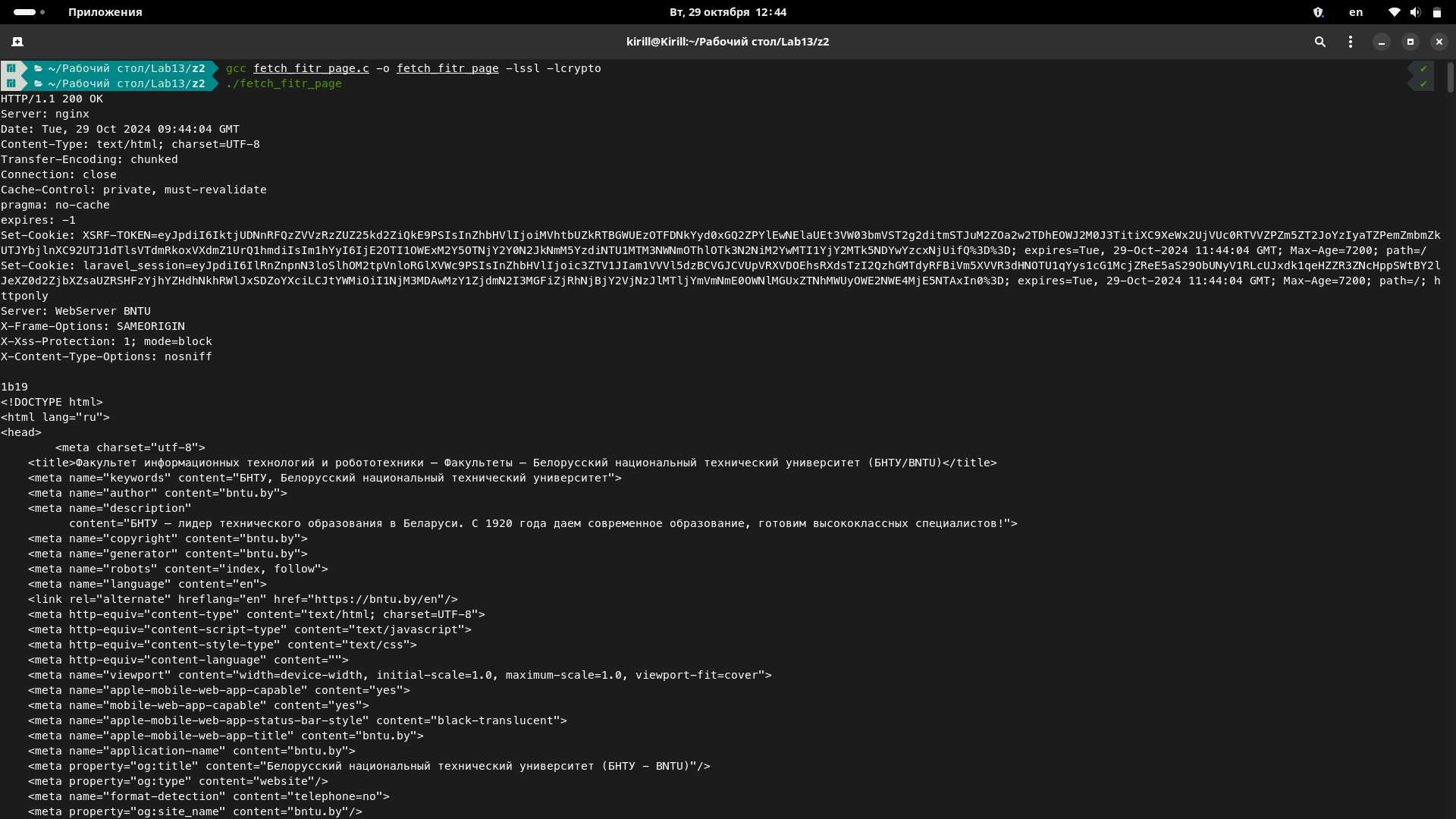
Задание 2 – Интернет сокеты

Напишите программу, которая читает главную страницу из какого-нибудь

Web-сервера. Запуск программы должен производиться с консоли.

Листинг

[#include](tg://search_hashtag?hashtag=include) <stdio.h>  
[#include](tg://search_hashtag?hashtag=include) <stdlib.h>  
[#include](tg://search_hashtag?hashtag=include) <string.h>  
[#include](tg://search_hashtag?hashtag=include) <unistd.h>  
[#include](tg://search_hashtag?hashtag=include) <arpa/inet.h>  
[#include](tg://search_hashtag?hashtag=include) <netdb.h>  
[#include](tg://search_hashtag?hashtag=include) <openssl/ssl.h>  
[#include](tg://search_hashtag?hashtag=include) <openssl/err.h>  
  
[#define](tg://search_hashtag?hashtag=define) BUFFER\_SIZE 4096  
  
void error(const char \*msg) {  
perror(msg);  
exit(1);  
}  
  
int main() {  
SSL\_CTX \*ctx;  
SSL \*ssl;  
int sockfd;  
struct sockaddr\_in server\_addr;  
struct hostent \*host;  
char \*hostname = "[bntu.by](https://bntu.by/)";  
char \*path = "/faculties/fitr";  
char buffer[BUFFER\_SIZE];  
  
// Инициализация OpenSSL  
SSL\_library\_init();  
OpenSSL\_add\_all\_algorithms();  
SSL\_load\_error\_strings();  
ctx = SSL\_CTX\_new(TLS\_client\_method());  
if (ctx == NULL) {  
error("Ошибка при создании SSL\_CTX");  
}  
  
// Получение IP-адреса хоста  
host = gethostbyname(hostname);  
if (host == NULL) {  
error("Ошибка при получении IP-адреса");  
}  
  
// Создание сокета  
sockfd = socket(AF\_INET, SOCK\_STREAM, 0);  
if (sockfd < 0) {  
error("Ошибка при создании сокета");  
}  
  
// Настройка адреса сервера  
memset(&server\_addr, 0, sizeof(server\_addr));  
server\_addr.sin\_family = AF\_INET;  
server\_addr.sin\_port = htons(443); // Порт HTTPS  
memcpy(&server\_addr.sin\_addr.s\_addr, host->h\_addr, host->h\_length);  
  
// Подключение к серверу  
if (connect(sockfd, (struct sockaddr \*)&server\_addr, sizeof(server\_addr)) < 0) {  
error("Ошибка при подключении");  
}  
  
// Создание SSL-соединения  
ssl = SSL\_new(ctx);  
SSL\_set\_fd(ssl, sockfd);  
if (SSL\_connect(ssl) <= 0) {  
error("Ошибка при SSL\_connect");  
}  
  
// Формирование HTTP-запроса  
sprintf(buffer, "GET %s HTTP/1.1\r\nHost: %s\r\nConnection: close\r\n\r\n", path, hostname);  
  
// Отправка запроса  
if (SSL\_write(ssl, buffer, strlen(buffer)) <= 0) {  
error("Ошибка при отправке запроса");  
}  
  
// Получение ответа  
while (1) {  
memset(buffer, 0, BUFFER\_SIZE);  
int bytes\_received = SSL\_read(ssl, buffer, BUFFER\_SIZE - 1);  
if (bytes\_received <= 0) {  
break; // Выход из цикла при завершении  
}  
printf("%s", buffer);  
}  
  
// Завершение работы с SSL  
SSL\_shutdown(ssl);  
SSL\_free(ssl);  
close(sockfd);  
SSL\_CTX\_free(ctx);  
return 0;  
}

Скриншоты

Задание 3 – Дейтаграммные сокеты

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.

Листинг

### server.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#define SERVER\_PORT 8080

#define BUFFER\_SIZE 1024

int main() {

int server\_socket;

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

socklen\_t client\_addr\_len;

char buffer[BUFFER\_SIZE];

// Создаем UDP-сокет

server\_socket = socket(AF\_INET, SOCK\_DGRAM, 0);

if (server\_socket == -1) {

perror("Ошибка создания сокета");

exit(EXIT\_FAILURE);

}

// Инициализируем адрес сервера

memset(&server\_addr, 0, sizeof(server\_addr));

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

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

server\_addr.sin\_port = htons(SERVER\_PORT);

// Привязываем сокет к адресу сервера

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

perror("Ошибка привязки сокета");

close(server\_socket);

exit(EXIT\_FAILURE);

}

printf("Сервер запущен, ожидание данных от клиентов...\n");

while (1) {

// Принимаем данные от клиента

client\_addr\_len = sizeof(client\_addr);

int bytes\_received = recvfrom(server\_socket, buffer, BUFFER\_SIZE, 0,

(struct sockaddr \*)&client\_addr, &client\_addr\_len);

if (bytes\_received == -1) {

perror("Ошибка при получении данных");

continue;

}

// Выводим полученные данные

buffer[bytes\_received] = '\0';

printf("Получены данные от клиента [%s:%d]: %s\n",

inet\_ntoa(client\_addr.sin\_addr), ntohs(client\_addr.sin\_port), buffer);

// Отправляем ответ клиенту

int bytes\_sent = sendto(server\_socket, buffer, bytes\_received, 0,

(struct sockaddr \*)&client\_addr, client\_addr\_len);

if (bytes\_sent == -1) {

perror("Ошибка при отправке данных");

} else {

printf("Ответ отправлен клиенту [%s:%d]\n",

inet\_ntoa(client\_addr.sin\_addr), ntohs(client\_addr.sin\_port));

}

}

close(server\_socket);

return 0;

}

### client1.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#define SERVER\_PORT 8080

#define SERVER\_IP "127.0.0.1"

#define BUFFER\_SIZE 1024

int main() {

int client\_socket;

struct sockaddr\_in server\_addr;

char buffer[BUFFER\_SIZE];

// Создаем UDP-сокет

client\_socket = socket(AF\_INET, SOCK\_DGRAM, 0);

if (client\_socket == -1) {

perror("Ошибка создания сокета");

exit(EXIT\_FAILURE);

}

// Инициализируем адрес сервера

memset(&server\_addr, 0, sizeof(server\_addr));

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

server\_addr.sin\_addr.s\_addr = inet\_addr(SERVER\_IP);

server\_addr.sin\_port = htons(SERVER\_PORT);

// Отправляем данные серверу

strcpy(buffer, "Привет, сервер!");

int bytes\_sent = sendto(client\_socket, buffer, strlen(buffer), 0,

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

if (bytes\_sent == -1) {

perror("Ошибка при отправке данных");

close(client\_socket);

exit(EXIT\_FAILURE);

}

printf("Данные отправлены серверу.\n");

// Получаем ответ от сервера

int bytes\_received = recvfrom(client\_socket, buffer, BUFFER\_SIZE, 0, NULL, NULL);

if (bytes\_received == -1) {

perror("Ошибка при получении данных");

close(client\_socket);

exit(EXIT\_FAILURE);

}

buffer[bytes\_received] = '\0';

printf("Получен ответ от сервера: %s\n", buffer);

close(client\_socket);

return 0;

}

### client2.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#define SERVER\_PORT 8080

#define SERVER\_IP "::1" // Используем IPv6 localhost

#define BUFFER\_SIZE 1024

int main() {

int client\_socket;

struct sockaddr\_in6 server\_addr;

char buffer[BUFFER\_SIZE];

// Создаем UDP-сокет (IPv6)

client\_socket = socket(AF\_INET6, SOCK\_DGRAM, 0);

if (client\_socket == -1) {

perror("Ошибка создания сокета");

exit(EXIT\_FAILURE);

}

// Инициализируем адрес сервера (IPv6)

memset(&server\_addr, 0, sizeof(server\_addr));

server\_addr.sin6\_family = AF\_INET6;

inet\_pton(AF\_INET6, SERVER\_IP, &server\_addr.sin6\_addr); // Использование IPv6-адреса

server\_addr.sin6\_port = htons(SERVER\_PORT);

// Отправляем данные серверу

strcpy(buffer, "Привет, сервер! (IPv6)");

int bytes\_sent = sendto(client\_socket, buffer, strlen(buffer), 0,

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

if (bytes\_sent == -1) {

perror("Ошибка при отправке данных");

close(client\_socket);

exit(EXIT\_FAILURE);

}

printf("Данные отправлены серверу.\n");

// Получаем ответ от сервера

int bytes\_received = recvfrom(client\_socket, buffer, BUFFER\_SIZE, 0, NULL, NULL);

if (bytes\_received == -1) {

perror("Ошибка при получении данных");

close(client\_socket);

exit(EXIT\_FAILURE);

}

buffer[bytes\_received] = '\0';

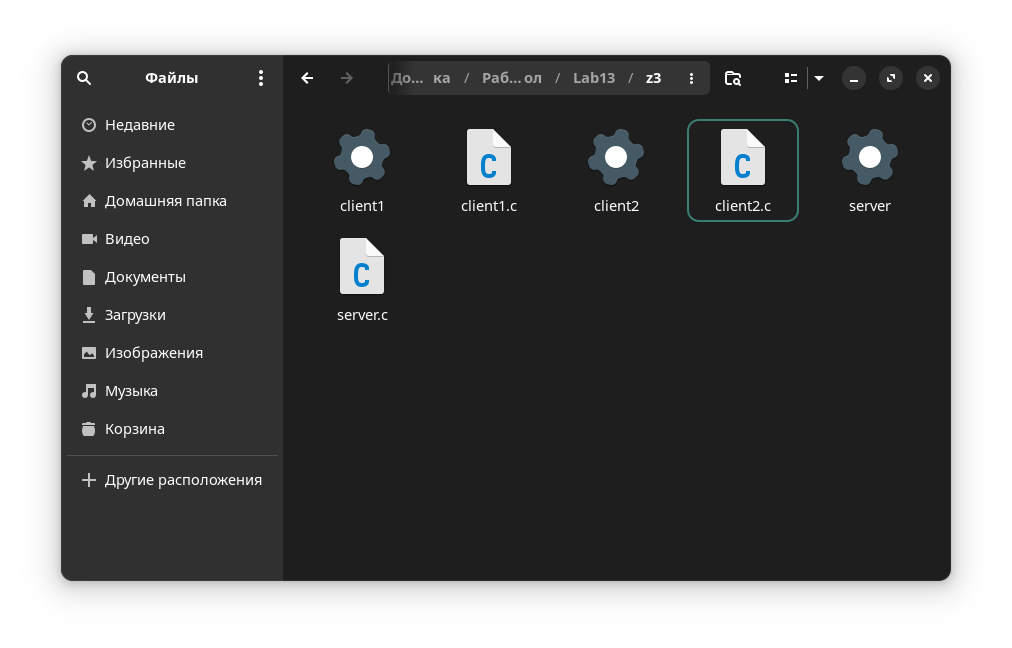
printf("Получен ответ от сервера: %s\n", buffer);

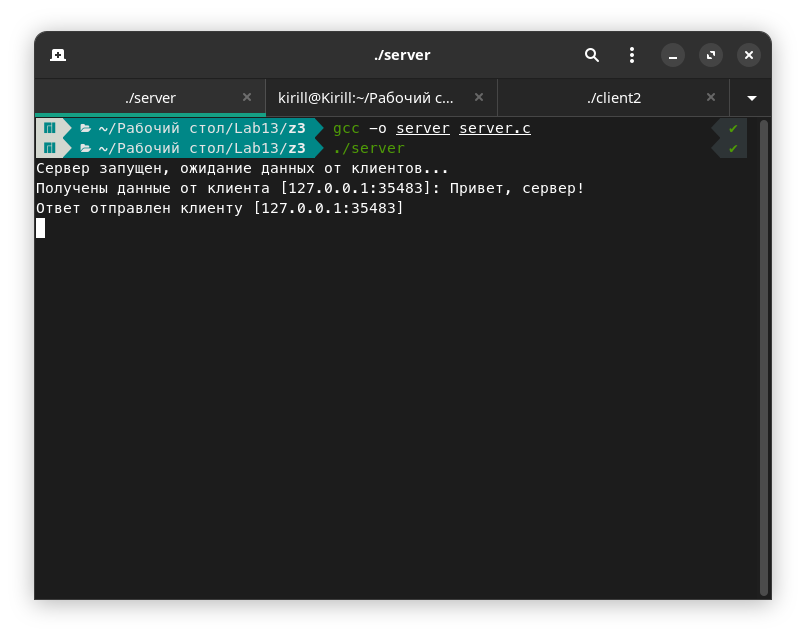
close(client\_socket);

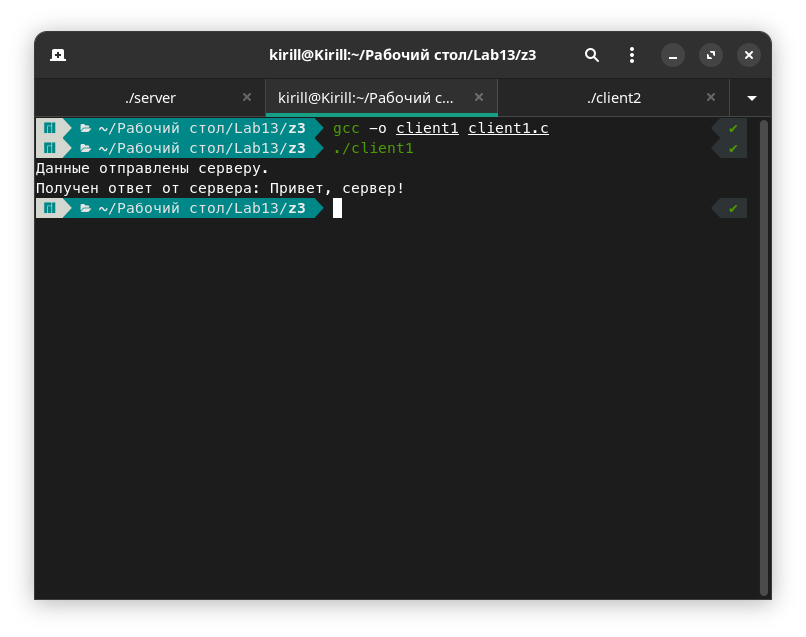
return 0;

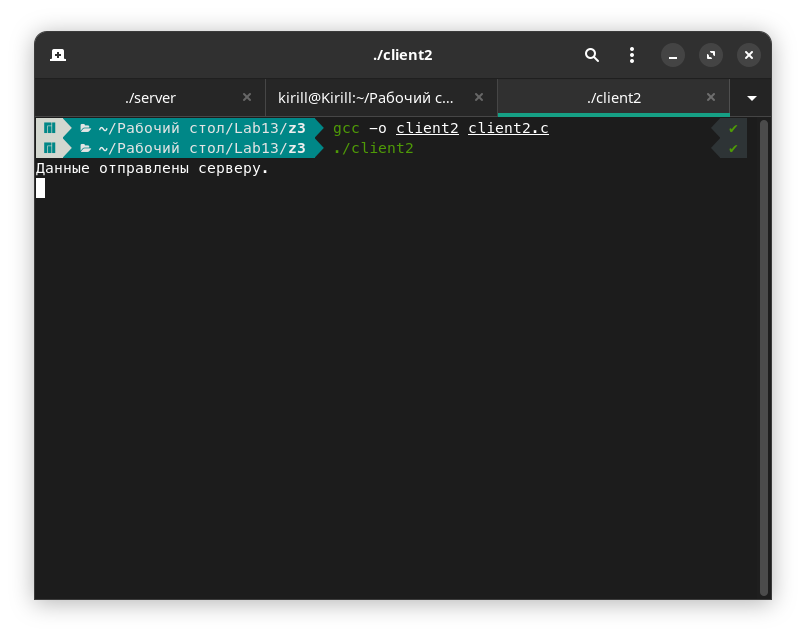
}

Скриншоты









Объяснение:

1. Сервер использует AF\_INET (IPv4) семейство протоколов для создания UDP-сокета и прослушивает порт 8080.
2. Клиент client1.c использует AF\_INET (IPv4) семейство протоколов для создания UDP-сокета и отправляет сообщение на сервер.
3. Клиент client2.c использует AF\_INET6 (IPv6) семейство протоколов для создания UDP-сокета и отправляет сообщение на сервер.

При запуске этих программ в разных терминалах:

* Сервер будет принимать данные от обоих клиентов и выводить их в консоль.
* Клиент client1.c (IPv4) получит ответ от сервера.
* Клиент client2.c (IPv6) получит ответ от сервера.

Вопросы:

Зухта

Что такое потоковые и дейтаграммные сокеты, для чего они используются, кратко и понятно

## Потоковые и дейтаграммные сокеты:  
  
Потоковые сокеты (TCP):  
  
• Как: Обеспечивают надежную, упорядоченную передачу данных между двумя приложениями.  
• Аналогия: Как разговор по телефону - данные передаются последовательно, с гарантией доставки и в правильном порядке.  
• Для чего:  
\* Веб-серверы: Для передачи веб-страниц и файлов.  
\* Файловые серверы: Для передачи файлов.  
\* Чат-приложения: Для передачи сообщений в реальном времени.  
  
Дейтаграммные сокеты (UDP):  
  
• Как: Передача данных в виде отдельных пакетов, без гарантий доставки и порядка.  
• Аналогия: Как отправка писем - письма могут приходить не в том порядке, или некоторые могут вообще не дойти.  
• Для чего:  
\* Игры: Для отправки игровых данных (позиция, действия), где потеря небольшого количества пакетов не критична.  
\* Видеоконференции: Для передачи аудио и видео в реальном времени.  
\* DNS-запросы: Для быстрой передачи небольших запросов.  
  
Вкратце:  
  
• TCP: Надежно, последовательно, но медленнее.  
• UDP: Быстрее, менее надежно, потеря пакетов возможна.  
  
Выбор типа сокета зависит от требований к надежности и производительности вашей программы.

Медведский

Какие системные вызовы обеспечивают основные операции с сокетами

Основные системные вызовы для работы с сокетами:  
  
• socket(): Создает новый сокет, задавая его тип (TCP/UDP), семейство протоколов (IPv4/IPv6) и протокол (TCP/UDP).   
• bind(): Привязывает сокет к определенному адресу и порту.  
• listen(): Ставит сокет в режим ожидания входящих соединений.   
• connect(): Устанавливает соединение с удаленным сервером.   
• accept(): Принимает входящее соединение от клиента.   
• send() и recv(): Отправляет и получает данные через сокет.   
• sendto() и recvfrom(): Отправляет и получает данные по конкретному адресу (используется для UDP-сокетов).  
• close(): Закрывает сокет, разрывая соединение.   
• getsockopt() и setsockopt(): Извлекают и устанавливают опции сокета (например, таймауты, буферизацию).  
• fcntl(): Управляет флагами файлового дескриптора сокета (например, блокировка).  
• select() и poll(): Обеспечивают множественное ожидание событий в сокетах.   
  
Используя эти вызовы, вы можете реализовать различные сетевые приложения, такие как серверы, клиенты, чаты и т.д.