

Министерство образования Республики Беларусь

Учреждение образования

«Белорусский государственный университет  
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей  
Кафедра электронных вычислительных машин

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

на тему

ТСР СЕРВЕР

БГУИР КР 1-40 02 01 311 ПЗ

Студент:

Кириллов В. И.

Руководитель:

Глоба А. А.

Минск 2022



## Оглавление

ВВЕДЕНИЕ.....	4
1. ОБЗОР ЛИТЕРАТУРЫ .....	5
2. СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ .....	8
2.1. Блок инициализации сервера.....	8
2.2. Блок встроенных команд.....	8
2.3. Модуль сервера .....	8
2.4. Блок инициализации клиента .....	8
2.5. Блок передачи команд .....	9
2.6. Модуль клиента.....	9
3. ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ .....	10
3.1. Блок инициализации сервера.....	10
3.2. Блок встроенных команд.....	10
3.3. Модуль сервера .....	10
3.4. Блок инициализации клиента .....	12
3.5. Блок передачи команд .....	12
3.6. Модуль клиента.....	12
4. РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ.....	14
4.1. Функция <code>configure_address</code> .....	14
4.2. Функция <code>create_socket</code> .....	14
4.3. Функция <code>get_string</code> .....	15
4.4. Функция <code>send_to_server</code> .....	15
4.5. Функция <code>print_dir</code> .....	15
4.6. Функция <code>handle_server</code> .....	16
5. ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ .....	17
6. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ.....	20
ЗАКЛЮЧЕНИЕ .....	21
ЛИТЕРАТУРА.....	22
ПРИЛОЖЕНИЕ А .....	23
ПРИЛОЖЕНИЕ Б.....	24
ПРИЛОЖЕНИЕ В .....	25

## **ВВЕДЕНИЕ**

Тема курсового проекта была выбрана в первую очередь для углубления знаний по языку С и в области программирования UNIX-подобных систем, а также получения знаний в проектировании серверных многопоточных приложений.

В современном мире серверы играют огромную роль в хранении, передаче данных, взаимодействии между пользователями и вычислениях. Серверы используются для хранения файлов и веб-сайтов пользователей Интернета, ответа на запросы и выдачи запрашиваемой информации, обработки и выполнения скриптов на веб-сайтах, работы с базой данных и большим количеством пользователей.

Находящиеся на сервере данные не требуется хранить на компьютерах клиента, так как они могут быть получены по запросу к серверу, не требуется самому производить вычисления, так как можно обратиться к серверу, на котором будут производиться необходимые вычисления, после чего ответ придет к сделавшему запрос клиенту. Благодаря этому серверы получили такое широкое распространение.

Серверы производятся как правило крупными фирмами, такими как IBM, NEC, Hewlett-Packard, Intel. Оболочку составляет операционная система для серверов, чаще всего это UNIX или операционные системы, созданные на ее базе – FreeBSD, Linux, AIX, IRIX и др. Реже встречаются серверные модификации Microsoft Windows Server, Solaris, SunOS и Mac Apple. Подобная популярность UNIX-подобных операционных систем объясняется их высокой надежностью и меньшими ресурсозатратами по сравнению с другими системами. Ввиду этого разработка программы ведется на операционной системе Linux.

Взаимодействие между сервером и пользователями (клиентами) определяет набор правил (протокол передачи данных). Существует множество различных протоколов, таких как UDP, FTP, HTTP, IP и другие. В данной курсовой работе используется протокол TCP, который обеспечивает высокую надежность в передаче данных, так как перед обменом данными устанавливает соединение клиента с сервером.

# 1. ОБЗОР ЛИТЕРАТУРЫ

## 1.1 Информация об серверах и их видах

Сервер — программный компонент вычислительной системы, выполняющий сервисные (обслуживающие) функции по запросу клиента, предоставляя ему доступ к определённым ресурсам или услугам.

Клиент — это аппаратный или программный компонент вычислительной системы, посылающий запросы серверу.

Важной частью взаимодействия между сервером и клиентом является протокол связи.

Протокол связи — набор определённых правил или соглашений интерфейса логического уровня, который определяет обмен данными между различными программами. Эти правила задают единообразный способ передачи сообщений и обработки ошибок.

Понятия сервер, клиент и закреплённые за ними роли образуют программную концепцию «клиент-сервер».

Характеристика «клиент-сервер» описывает отношения взаимодействующих программ в приложении. Серверный компонент предоставляет функцию или услугу одному или нескольким клиентам, которые инициируют запросы на такие услуги. Серверы классифицируются по предоставляемым ими услугам. Например, веб-сервер обслуживает веб-страницы, а файловый сервер обслуживает компьютерные файлы. Общий ресурс может быть любой из программного обеспечения и электронных компонентов компьютера — сервера, от программ и данных в процессорах и запоминающих устройств. Совместное использование ресурсов сервера представляет собой услугу. В данной программе в качестве средства взаимодействия между клиентами и сервером используются сокеты.[1][3]

Transmission Control Protocol (TCP, протокол управления передачей) — один из основных протоколов передачи данных.[2]

Механизм TCP предоставляет поток данных с предварительной установкой соединения, осуществляет повторный запрос данных в случае потери данных и устраняет дублирование при получении двух копий одного пакета, гарантируя тем самым целостность передаваемых данных и уведомление отправителя о результатах передачи.

Протокол TCP «заточен» под соединение. Иными словами, прежде, чем начать обмен данными, данному протоколу требуется установить соединение между двумя хостами. Данный протокол имеет высокую надёжность, поскольку позволяет не терять данные при передаче, запрашивает подтверждения о получении от принимающей стороны и в случае необходимости отправляет данные повторно. При этом отправляемые пакеты данных сохраняют порядок отправки, то есть можно сказать, что передача данных упорядочена. Минусом данного протокола является относительно

низкая скорость передачи данных, за счет того, что выполнение надежной и упорядоченной передачи занимает больше времени, чем в альтернативном протоколе UDP.

Клиенты и серверы обмениваются сообщениями в шаблоне запрос-ответ. Клиент отправляет запрос, а сервер возвращает ответ. Этот обмен сообщениями является примером межпроцессного взаимодействия. Для взаимодействия компьютеры должны иметь общий язык, и они должны следовать правилам, чтобы и клиент, и сервер знали, чего ожидать. Язык и правила общения определены в протоколе связи. Все протоколы клиент-серверной модели работают на уровне приложений. Протокол прикладного уровня определяет основные шаблоны диалога. Чтобы ещё больше формализовать обмен данными, сервер может реализовать интерфейс прикладного программирования (API). API — это уровень абстракции для доступа к сервису. Ограничивая связь определённым форматом контента, он облегчает синтаксический анализ. Абстрагируя доступ, он облегчает межплатформенный обмен данными.

Сервер может получать запросы от множества различных клиентов за короткий период времени. Компьютер может выполнять только ограниченное количество задач в любой момент и полагается на систему планирования для определения приоритетов входящих запросов от клиентов для их удовлетворения. Чтобы предотвратить злоупотребления и максимизировать доступность серверное программное обеспечение может ограничивать доступность для клиентов. Атаки типа «отказ в обслуживании» используют обязанности сервера обрабатывать запросы, такие атаки действуют путем перегрузки сервера чрезмерной частотой запросов. Шифрование следует применять, если между клиентом и сервером должна передаваться конфиденциальная информация.

## **1.2 Анализ существующих аналогов**

Наиболее похожим, в отличие от прочих аналогов, на протокол TCP является протокол UDP.

Протокол UDP (User Datagram Protocol) более простой вариант протокола обмена данными, так как для передачи ему не обязательно устанавливать соединение между отправителем и получателем. Информация в таком случае передается без предварительной проверки готовности принимающей стороны. Это делает протокол менее надежным, так как при передаче некоторые фрагменты данных могут теряться. Кроме того, упорядоченность данных не соблюдается — возможен непоследовательный прием данных получателем, что увеличивает скорость передачи, но небезопасно с точки зрения получения данных, что является более важной составляющей сервера.

### 1.3 Постановка задачи

В данной работе будут применяться следующие алгоритмы работы сервера:

1. Алгоритм обработки подключений к серверу включает в себя логику и обработку ошибок в случае неудачных подключений клиентов к серверу, в случае отсутствия клиентов сервер ожидает подключения.
2. Алгоритм обмена данными с сервером включает в себя возможность ввода различных команд пользователем, которые будут отправляться серверу и исполняться либо сервером, либо клиентом.

После рассмотрения поставленных задач были выбраны несколько ключевых возможностей, которые будут выполнены в рамках данного курсового проекта:

- Программа должна иметь понятный пользовательский интерфейс с возможностью просмотра списка команд.
- Сервер должен поддерживать множество одновременных подключений.
- В программе должна быть предусмотрена возможность вызова некоторых функций.
- Сервер должен работать до пользовательского прерывания и отслеживать подключения клиентов, сообщать об ошибках подключения, а также обрабатывать пользовательские команды.

## **2. СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ**

Сначала следует поставить конкретные функциональные требования разрабатываемой программе, разбить утилиту на модули и функциональные блоки. Данный подход в большинстве упростит понимание проектирования, сможет помочь устранить проблемы в архитектуре и обеспечить гибкость каждого из модулей. После того, как аналоги рассмотрены и проведён краткий экскурс в основные понятия, можно поставить конкретные цели разрабатываемому программному обеспечению.

Структурно программа будет состоять из 2 модулей: сервера и клиента, а также нескольких вспомогательных блоков. Перечень блоков и описание главных модулей приведены ниже. Структурная схема приведена в приложении «А».

### **2.1. Блок инициализации сервера**

Данный блок отвечает за начальную настройку сервера и представлен в виде функций `create_socket()`, `configure_address()`, `bind()` и `listen()`. Здесь происходит инициализация сокета, создание массива клиентов, а также буферы для приема команд клиентов, связывание и прослушивание сокета.

### **2.2. Блок встроенных команд**

Данный блок содержит обработчики пользовательских запросов в виде набора функций и состоит из функций: `print_help()`, `print_dir()`, `print_full_path()`, `close_socket()`. Блок служит для взаимодействия клиентов с сервером и для выполнения различных задач сервером. Здесь проводятся основные действия: операции вывода пути расположения сервера или клиента, вывод содержимого текущей папки сервера или клиента, вывод информации о файле, выбранном пользователем. Данный модуль можно разбить на более мелкие блоки ответственности.

### **2.3. Модуль сервера**

Данный модуль отвечает за строение сервера и включает в себя блоки инициализации, подключения и отключения, а также обработки запросов клиентов, в результате чего образуется полноценный модуль сервера с полным набором вышеописанных функций.

### **2.4. Блок инициализации клиента**

Данный блок отвечает за начальную настройку клиента и представлен в виде функции `configure_address()` и `connect()`, которые будут вызваны единожды в начале программы. Здесь происходит инициализация сокета, создание массива для отправки команд клиента, а также связывание сокета



с сервером по переданным параметрам.

## **2.5. Блок передачи команд**

Данный блок отвечает за ввод пользовательских команд в консоли клиента и за передачу введенной команды серверу. Для ввода команды используется функция `get_string()`, для передачи её серверу `send_to_server()`.

## **2.6. Модуль клиента**

Данный модуль отвечает за строение клиента и включает в себя блоки инициализации, подключения и отключения от сервера, а также обработки запросов клиентов, в результате чего образуется полноценный модуль клиента с полным набором вышеописанных функций.

### 3. ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

В данной главе будут рассмотрены функции и структуры, используемые для реализации сервера и клиента, описанное при помощи разбиения функциональных частей программы на модули и блоки. Программа содержит два основных модуля: сервера и клиента и включает в себя несколько блоков.

#### 3.1. Блок инициализации сервера

`void configure_address(struct sockaddr_in *address, int port, char *ip)` инициализирует экземпляр структуры `address` значениями `AF_INET` для поля `sin_family` (семейство адресов), `port` для поля `sin_port`, а также конвертирует переданный `ip` в набор байт, после чего присваивает его полю `sin_addr.s_addr`.

`int create_socket()` создает уникальный сокет и если создание прошло неуспешно, сообщает об ошибке, после чего программа завершается.

`int check(ssize_t exp, const char *msg)` – проверочная функция, служащая для выполнения действия, связанного с настройкой сокетов, а также проверки результата на успешность. В случае успешного действия функция возвращает значение `exp`, иначе выводит сообщение `msg` в поток `stderr` и завершает программу с кодом 1.

`void close_socket(int socket_fd)` обертка функции `close()`, проверяющая успешность операции закрытия сокета `socket_fd` с использованием вышеописанной функции `check()`. Если закрытие прошло неуспешно, сообщает об ошибке, после чего программа завершается с кодом 1.

`void pipe_handler(int)` функция-обработчик сигнала `SIGPIPE`. Служит для корректного завершения работы сервера и клиента.

#### 3.2. Блок встроенных команд

`void print_help()` выводит на экран список доступных команд.

`void print_dir()` выводит на экран полный список находящихся в папке файлов и папок для введенного пользователем каталога. Функция запрашивает пользовательский ввод, и в случае корректно введенного пути каталога выводит его содержимое. Локальный параметр `char *path` является путем каталога (относительным или абсолютным).

`void print_full_path()` выводит на экран абсолютный путь текущей папки для сервера либо клиента (в зависимости от команды).

`void get_file_info()` выводит на экран все параметры запрошенного функцией файла: время создания, изменения, размер в байтах, ID пользователя и группы пользователей, и значение индексного дескриптора `inode`, содержащего метаданные файла помимо его данных и имени.

#### 3.3. Модуль сервера

В данном модуле находятся глобальные переменные с основными параметрами сервера, а также основная функция обработки запросов клиентов.

`void cancel_server_threads(int amount)` функция для завершения всех действующих потоков клиентов, которое определяется переменной `amount`.

`void * handle_server(void *arg)` основная функция сервера, служащая для обработки подключений клиента к серверу, а также выполняющая инструкции, полученных от клиента. Указатель на данную функцию передается каждому новому клиенту из массива `client_threads[]`.

`PORT 5566` порт для сервера.

`LOCALHOST "127.0.0.1"` IP-адрес для сервера (по умолчанию локальный адрес машины).

`int server_sock` глобальная переменная, содержащая сокет (дескриптор) сервера. Требуется для подключения клиентов.

`int client_sockets[MAX_CLIENTS]` глобальный массив для хранения сокетов клиентов. Требуется для одновременного взаимодействия нескольких клиентов с сервером. Количество одновременных подключений определяется константой `MAX_CLIENTS`, по умолчанию равной 4.

`pthread_t[MAX_CLIENTS]` глобальный массив для хранения потоков подключаемых клиентов. Обеспечивает многопоточность сервера.

`bool create_new_thread` глобальная переменная, выполняющая функцию индикатора создания нового потока. Начальное значение `false`.

`bool thread_ended` переменная для определения завершения потока клиента.

`struct sockaddr_in server_address, client_address` описывают сокеты для работы с протоколами IP. Значение поля `sin_family` всегда равно `AF_INET`. Поле `sin_port` содержит номер порта, который намерен занять процесс. Поле `sin_addr` содержит IP адрес, к которому будет привязан сокет.

`socklen_t address_size` используется для хранения размера переменной `client_address`.

`MESSAGE_SIZE 256` константа, используемая для буфера приема команд от пользователя.

`FILENAME_SIZE 256` константа, используемая для максимальной длины файла.

`char buffer[MESSAGE_SIZE]` буфер для команд, считанных с консоли пользователем, и которые будут использоваться для передачи команд серверу.

`COLOR_RED "\033[1;31;40m"` используется для создания красного цвета текста в консоли.

`COLOR_NO "\033[1;0m"` используется для снятия выделения красным цветом от константы `COLOR_RED`.

`LS_COMMAND_CLIENT "ls\n"` команда вызова функции `print_dir co`

стороны клиента.

LS\_COMMAND\_SERVER “!ls\n” команда вызова функции print\_dir со стороны сервера.

PWD\_COMMAND\_CLIENT “pwd\n” команда вызова функции print\_full\_path со стороны клиента.

PWD\_COMMAND\_SERVER “!pwd\n” команда вызова функции print\_full\_path со стороны сервера.

HELP\_COMMAND “help\n” команда вызова функции print\_help со стороны клиента.

EXIT\_COMMAND\_SERVER “exit\n” команда отключения клиента от сервера.

EXIT\_COMMAND\_ALL “!exit\n” команда отключения текущего клиента и завершения работы сервера.

SOCKET\_ERROR -1 код ошибки всех используемых функций взаимодействия между клиентами и сервером.

### **3.4. Блок инициализации клиента**

void configure\_address(struct sockaddr\_in \*address, int port, char \*ip) инициализирует экземпляр структуры address значениями AF\_INET для поля sin\_family (семейство адресов), port для поля sin\_port, а также конвертирует переданный ip в набор байт, после чего присваивает его полю sin\_addr.s\_addr.

int create\_socket() создает уникальный сокет и если создание прошло неуспешно, сообщает об ошибке, после чего программа завершается.

### **3.5. Блок передачи команд**

void send\_to\_server(int sock, char \*buffer) функция для отправки данных из переменной buffer серверу с дескриптором sock. При вызове запрашивает пользовательский ввод, после чего посылает его серверу. В случае ошибок программа завершается с кодом 1.

void get\_string(char \*string, int len) считывает в переданную переменную string количество символов, равное len, после чего удаляет все лишние символы до символа-разделителя ‘\n’. В результате остается нужная заполненная строка string и очищенный буфер потока stdin, что предотвращает некорректный ввод в дальнейшем.

### **3.6. Модуль клиента**

В данном модуле находятся глобальные переменные с основными параметрами клиента, а также основная функция для ввода команд пользователем и отправки их серверу.

`struct sockaddr_in address` описывает сокет для работы с протоколами IP. Значение поля `sin_family` всегда равно `AF_INET`. Поле `sin_port` содержит номер порта, который намерен занять процесс. Поле `sin_addr` содержит IP адрес, к которому будет привязан сокет.

`char buffer[MESSAGE_SIZE]` используется для приема и передачи команд/сообщений серверу с фиксированным размером, равным константе `MESSAGE_SIZE`.

## 4. РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

Основополагающими функциями данной программы являются функции подключения клиентов к серверу (`configure_address`, `create_socket`), ввод команд и отправка их серверу (`send_to_server`), а также функция взаимодействия с несколькими клиентами одновременно (`handle_server`).

### 4.1. Функция `configure_address`

Данная функция принимает в качестве аргумента указатель на структуру `sockaddr_in`, целочисленную переменную, хранящую порт, а также указатель на массив `char`, хранящий IP-адрес сервера. Структура `sockaddr_in` имеет следующие поля, которые требуется инициализировать каждым из вышеописанных значений:

- 1) `sin_port` номер порта, которому присваивается значение переменной `port`;
- 2) `sin_family` семейство адресов, которому присваивается значение `AF_INET`.
- 3) `sin_addr.s_addr` интернет-адрес, которому присваивается значение, полученное в результате работы функции `inet_addr()` с переданным ей значением `ip`. Данная функция преобразует строковое представление IP в набор байт.

Таким образом, данная функция используется для первичной инициализации как сервера, так и клиентов.

### 4.2. Функция `create_socket`

Данная функция используется для создания уникального сокета и является оберткой функции `socket()` из библиотеки `socket.h`. Данная функция принимает в качестве аргументов целочисленную переменную домена, тип сокета и значение протокола.

В функции используются следующие параметры для функции `socket`:

- 1) `AF_INET` в качестве домена;
- 2) `SOCK_STREAM` в качестве типа сокета, который является достаточно надежным и последовательным;
- 3) `0` в качестве идентификатора протокола используется стандартный протокол.

После вызова функции `socket()` происходит проверка на успешность создания сокета. Если функция вернула значение, равное `-1`, то произошел сбой при создании сокета, поэтому выводится сообщение о соответствующей ошибке, после чего происходит аварийное завершение программы. В противном случае возвращается значение созданного сокета.

### 4.3. Функция `get_string`

Данная функция в качестве параметров принимает указатель на массив `char *string` и целочисленную переменную длины `len`. Получение ввода пользователя осуществляется при помощи функции `fgets()` из стандартной библиотеки `stdio.h`. Данная функция в качестве параметра принимает указатель типа `char*` со строкой, целочисленную переменную длины строки, а также поток, из которого требуется считать данные и поместить ее в строку. Результатом работы функции является новая строка заданной длины, в конце которой расположен нуль-терминатор. После получения строки происходит очистка буфера потока `stdin` при помощи цикла `while` и функции `getchar()`.

Таким образом, функция `get_string` обеспечивает надежный и безопасный ввод строки с заданной длиной с клавиатуры, что особенно важно для серверов, ведь они не должны хранить никаких лишних данных.

### 4.4. Функция `send_to_server`

Данная функция является основополагающей во взаимодействии пользователя клиента с сервером, так как обеспечивает обмен клиента с сервером, отвечая за передачу тому команд.

Функция принимает в качестве аргументов целочисленное значение сокета-приемника `sock`, а также буфер `buffer` типа `char`. Затем производится очищение переданного буфера при помощи функции `bzero()`, после чего делает запрос на ввод команды пользователем. После этого вызывается вышеописанная функция `get_string` для заполнения буфера, а затем при помощи функции `send()` из библиотеки `socket.h` содержимое буфера отсылается сокету

Сама функция `send()` принимает в качестве параметров:

- 1) `int sock_fd` значение сокета-получателя;
- 2) `const void* buf` указатель на буфер данных, которые требуется отправить;
- 3) `size_t n` количество байт, которое требуется отправить из буфера;
- 4) `int flags` поле бит с опциями отправки.

При успешном завершении работы функции возвращается число 0, поэтому в функции `send_to_server` предусмотрена проверка, которая в случае сбоя функции `send` оповестит об этом пользователя и экстренно завершит программу. Таким образом, пересылка данных также является достаточно надежной, чтобы не опасаться утери данных.

### 4.5. Функция `print_dir`

Данная функция является пользователем аналогом команды “ls” Unix систем и выводит список файлов, находящихся по переданному в качестве параметра пути `char *path`.

Если переданный в качестве аргумента путь к папке равен пустой строке (то есть не указан), то программа работает для текущей папки. Для этого используется функция `getcwd`, определенная в стандарте POSIX, которая позволяет получить абсолютный путь текущей папки.

Далее создается указатель на структуру `DIR *dir`, и с помощью функции `opendir` открывает папку в виде потока, и в качестве аргумента принимает путь к папке.

В случае если открытие папки прошло неуспешно в консоль выводится сообщение о возникшей ошибке, после чего функция завершает выполнение. В противном случае происходит последовательное чтение содержимого папки при помощи функции `readdir`, которая в качестве единственного аргумента принимает указатель `DIR *dir`, и результат работы присваивает предварительно созданной переменной типа `struct dirent *entry`. Для работы функции `print_dir` используется лишь два поля данной структуры: `d_name` и `d_type`. Поле `d_name` используется для хранения имени считанного содержимого, и представляет собой массив типа `char*`. Поле `d_type` представляет собой тип считанного содержимого: файл, папка, ссылка и т.д.

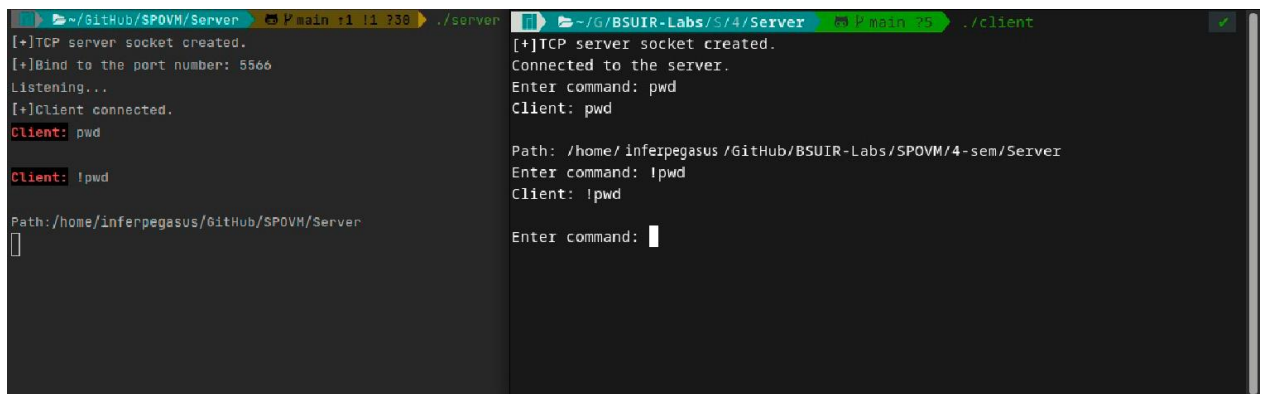
После считывания проверяется тип считанного содержимого, и если оно является корневой или родительской папкой, то совершается следующая итерация цикла. В противном случае имя считанного содержимого выводится в консоль.

#### **4.6. Функция `handle_server`**

Данная функция служит для обработки нескольких одновременных подключений клиентов к серверу. В качестве параметра ей передается `void *arg`, который внутри функции преобразуется к типу `int`, который служит номером текущего клиента. Внутри функции находятся функции для создания сокета и подключения к серверу, после чего ожидается ввод команд пользователем для передачи их непосредственно серверу. Так как функция используется в совокупности с массивом потоков, обеспечивается обработка нескольких клиентов, максимальное количество которых определяется константой `MAX_CLIENTS` и по умолчанию равно 4.



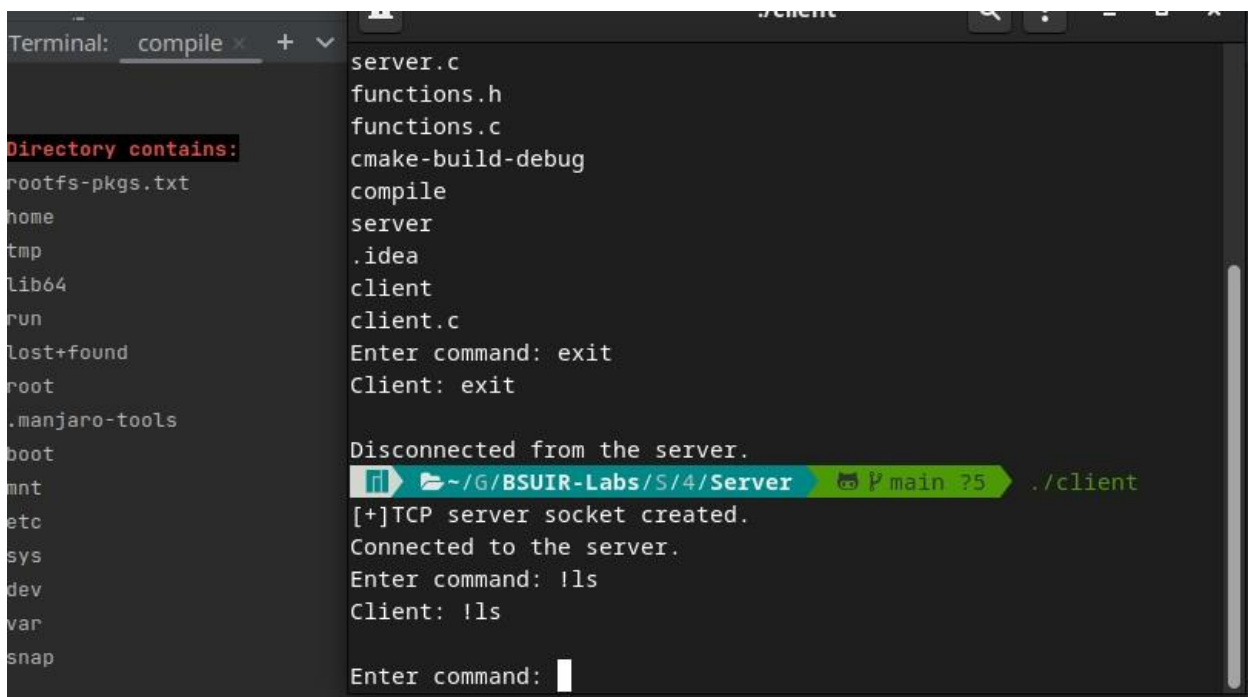
## 5. ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ



```
[+]TCP server socket created.
[+]Bind to the port number: 5566
Listening...
[+]Client connected.
Client: pwd
Client: !pwd
Path: /home/inferpegasus/GitHub/SPOVM/Server

[+]TCP server socket created.
Connected to the server.
Enter command: pwd
Client: pwd
Path: /home/inferpegasus/GitHub/BSUIR-Labs/SPOVM/4-sem/Server
Enter command: !pwd
Client: !pwd
Path: /home/inferpegasus/GitHub/BSUIR-Labs/SPOVM/4-sem/Server
Enter command:
```

Рисунок 5.1. Результат работы pwd и !pwd

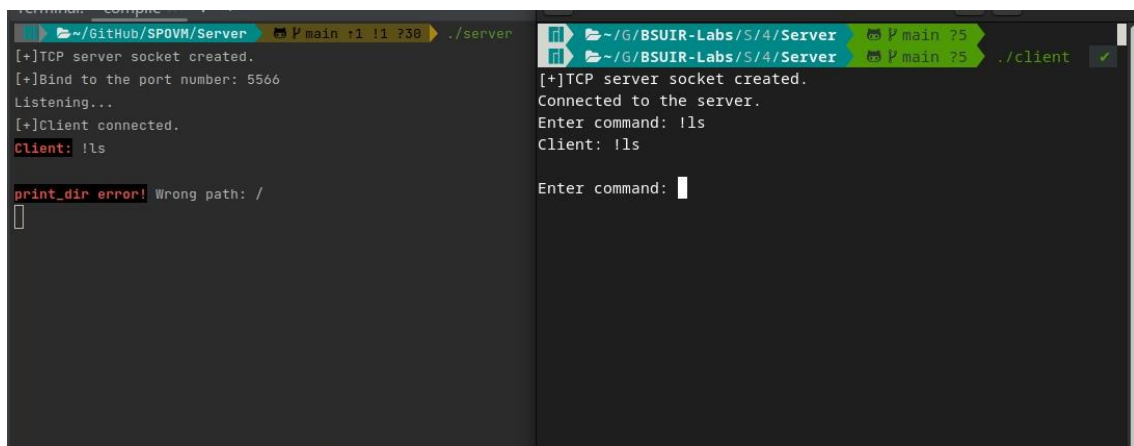


```
server.c
functions.h
functions.c
cmake-build-debug
compile
server
client
client.c
Enter command: exit
Client: exit

Disconnected from the server.

[+]TCP server socket created.
Connected to the server.
Enter command: !ls
Client: !ls
Enter command:
```

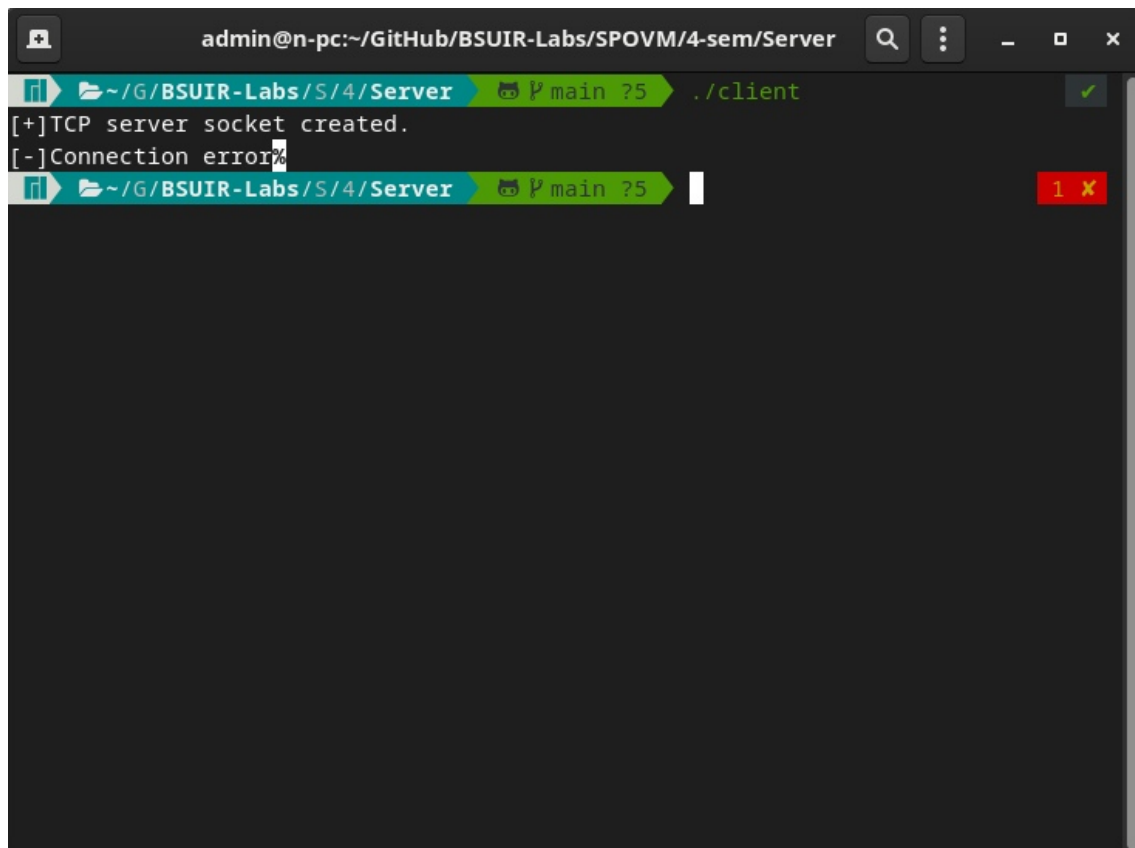
Рисунок 5.2. Результат работы !ls



```
[+]TCP server socket created.
[+]Bind to the port number: 5566
Listening...
[+]Client connected.
Client: !ls
print_dir error! Wrong path: /

[+]TCP server socket created.
Connected to the server.
Enter command: !ls
Client: !ls
Enter command:
```

Рисунок 5.3. Вывод ошибки от неправильного переданного пути команде !ls



The screenshot shows a terminal window with a dark background. The title bar at the top reads "admin@n-pc:~/GitHub/BSUIR-Labs/SPOVM/4-sem/Server". The terminal content shows a prompt at the directory "~/G/BSUIR-Labs/S/4/Server" with the command `./client` entered. The output consists of two lines: "[+]TCP server socket created." and "[-]Connection error". The prompt for the second line is partially obscured by the text "Connection error". To the right of the terminal window, there is a red status bar with the number "1" and a red 'X' icon.

Рисунок 5.4. Ошибка при подключении к выключенному серверу

```
./.client
[+]TCP server socket created.
Connected to the server.
Enter command: help
Client: help

Commands:
help - print list of commands
ls - print files in client dir
!ls - print files in server dir
pwd - print full client path
!pwd - print full server path
exit - close client

Enter command: 
```

Рисунок 5.5. Вывод списка команд для пользователя

```
tmux
Client: pwd
Client: !exit
[+]Server closed.
~/G/Server/cmake-build-debug main +2 14 ?
~/G/Server/cmake-build-debug main +2 14 ?
20 > ./server 1m 51s
[+]TCP server socket created.
[+]Bind to the port number: 5566
Listening...
[+]Client connected.
Client: file
Enter filename: client.c
Getting stats error!: No such file or directory
Client: file
Enter filename: ../client.c
Inode number: 1201141
User ID of owner: 1000
Group ID of owner: 1000
Total file size: 1358 bytes
Last status change: Sat May 21 20:42:41 2022
Last file access: Thu May 12 18:44:28 2022
Last file modification: Sat May 21 20:42:41 2022
Client: !exit
[+]Server closed.
~/G/Server/cmake-build-debug main !4 718
[0] 0:zsh*

Unknown command!
Enter command: pwd
Client: pwd
Path: /home/fistingasus/GitHub/Server/cmake-build-debug
Enter command: !pwd
Client: !pwd
Enter command: exit
Client: exit
Disconnected from the server.
~/G/Server/cmake-build-debug main +2 14
~/G/Server/cmake-build-debug main +2 14
720 > ./client 1m 54s
[+]TCP server socket created.
Connected to the server.
Enter command: file
Client: file
Enter command: file
Client: file
Enter command: !exit
Client: !exit
[+]Client closed.
~/G/Server/cmake-build-debug main +2 14 720 > 1m 45s

Enter command: ls
Client: ls
Empty path provided! Showing info in '/home/fistingasus/GitHub/Server/cmake-build-debug':
Directory contains:
.ninja_deps
CMakeFiles
Testing
build.ninja
cmake_install.cmake
CMakeCache.txt
.cmake
.ninja_log
server
client
Enter command: pwd
Client: pwd
Path: /home/fistingasus/GitHub/Server/cmake-build-debug
Enter command: !exit
Client: !exit
[+]Client closed.
~/G/Server/cmake-build-debug main +2 14 720 > 1m 47s

Enter command: ls
Client: ls
Empty path provided! Showing info in '/home/fistingasus/GitHub/Server/cmake-build-debug':
Directory contains:
.ninja_deps
CMakeFiles
Testing
build.ninja
cmake_install.cmake
CMakeCache.txt
.cmake
.ninja_log
server
client
Enter command: pwd
Client: pwd
Path: /home/fistingasus/GitHub/Server/cmake-build-debug
Enter command: exit
Client: exit
Disconnected from the server.
~/G/Server/cmake-build-debug main +2 14 720 > 1m 47s
```

Рисунок 5.6. Работа сервера с несколькими клиентами

## 6. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Программное обеспечение тестировалась на ОС Manjaro Linux и нет гарантий, что она компилируется и работает на macOS или FreeBSD, возможно проблемы возникнут и на других дистрибутивах GNU/Linux.

Для того чтобы скомпилировать сервер и клиент, требуется запустить файл `compile`. Сделать это можно посредством команды `bin/bash compile`. Для этого потребуется наличие утилиты `bash`.

Для запуска сервера необходимо запустить файл с именем `server`. Для этого требуется ввести `./server` в терминале. Для запуска клиента требуется создать новое окно терминала и прописать в нем `./client`. Для работы клиента и сервера требуется запускать терминалы в папках, где хранятся сервер и клиент.

Программное обеспечение поддерживает следующий функционал:

- 1) Вывод содержимого папки сервера;
- 2) Вывод содержимого папки клиента;
- 3) Вывод абсолютного пути сервера;
- 4) Вывод абсолютного пути клиента;
- 5) Вывод информации о любом файле, путь к которому указывает пользователь;
- 6) Вывод списка команд, доступных пользователю;
- 7) Поддержание сервера в активном режиме даже без подключенных пользователей.

## ЗАКЛЮЧЕНИЕ

В данном разделе будут проведены итоги разработки программного обеспечения.

Плюсы данного приложения:

- Простой, понятный и нетребовательный к ресурсам компьютера пользовательский интерфейс;
- Наличие многопоточности сервера, что позволяет взаимодействовать с сервером нескольким клиентам.
- Возможность выбора команды для исполнения сервером или клиентом;
- Достаточно большой потенциал для расширения функциональности;
- Стабильная работа из-за надежного протокола TCP.

В результате выполнения курсового проекта были изучены: проектирование серверных приложений, основы протоколов передачи данных, работы с потоками, а также были углублены знания языка программирования C.[5]

Работа была разделена на такие этапы, как анализ существующих аналогов, литературных источников, постановка требований к проектируемому программному продукту, системное и функциональное проектирование, конструирование программного продукта, разработка программных модулей и тестирование проекта. После последовательного выполнения вышеперечисленных этапов разработки было получено исправно работающее приложение.

В дальнейшем будет доработан функционал приложения, добавлена возможность производить GET и POST запросы, а также добавлены новые функции клиента и сервера.

## ЛИТЕРАТУРА

- [1]. What is a socket? – IBM Documentation [Электронный ресурс]. Режим доступа: <https://www.ibm.com/docs/en/zos/2.1.0?topic=services-what-is-socket>
- [2]. What is a TCP protocol? – IBM Documentation [Электронный ресурс]. Режим доступа: <https://www.ibm.com/docs/en/zos-basic-skills?topic=layertransmission-control-protocol-tcp>
- [3]. The Client/Server Model – IBM Documentation [Электронный ресурс]. Режим доступа: <https://www.ibm.com/docs/en/zos/2.1.0?topic=applications-clientserver-model>
- [4]. Multithreaded server recommendations – IBM Documentation [Электронный ресурс]. Режим доступа: <https://www.ibm.com/docs/en/zos/2.1.0?topic=applications-clientserver-model>
- [5]. Сеть: Отличный гайд по сетевому программированию – Brian «Beej Jorgensen» Hall [Электронный ресурс]. Режим доступа: [https://masandilov.ru/network/guide\\_to\\_network\\_programming](https://masandilov.ru/network/guide_to_network_programming)

**ПРИЛОЖЕНИЕ А**  
(обязательное)

Структурная схема приложения

## **ПРИЛОЖЕНИЕ Б**

**(обязательное)**

Блок-схема алгоритма подключений к серверу



## **ПРИЛОЖЕНИЕ В**

Репозиторий с исходным кодом

<https://github.com/InfernumPegasus/TCP-Server>