Министерство образования Республики Беларусь
Учреждение образования «Белорусский государственный
университетинформатики и радиоэлектроники»

Факультет компьютерных систем и сетей
Кафедра электронных вычислительных машин
Дисциплина: Операционные системы и системное программирование

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту на тему

PEER-TO-PEER СЕТЕВОЙ ЧАТ С ТЕКСТОВЫМ ИНТЕРФЕЙСОМ NCURSES

БГУИР КП 1-40 02 01 218 ПЗ

Студент: Максимчик Е. В.

Руководитель: старший преподаватель
каф. ЭВМ Поденок Л. П.

Минск 2023

Учреждение образования
«Белорусский государственный университет
информатикии радиоэлектроники»

Факультет компьютерных систем и сетей

УТВЕРЖДАЮ
Заведующий кафедрой ЭВМ
_____Б.В. Никульшин
(подпись)
«___»_____2023 г.

ЗАДАНИЕ
по курсовому проектированию

Студенту *Максимчику Егору Валерьевичу*

1. Тема проекта: *Peer-to-peer сетевой чат с текстовым интерфейсом ncurses*

2. Срок сдачи студентом законченного проекта *15.05.2023*.

3. Исходные данные к проекту *Язык программирования – C*

4. Содержание расчетно-пояснительной записки (перечень вопросов, которые подлежат разработке)

Введение. 1. Обзор литературы. 2. Системное проектирование. 3. Функциональное проектирование. 4. Разработка программных модулей. 5. Программа и методика испытаний. 6. Руководство пользователя. Заключение. Список использованных источников

5. Перечень графического материала (с точным обозначением обязательных чертежей и графиков) *1. Схема структурная. 2. Диаграмма классов*

6. Дата выдачи задания *23.02.2023.*

7. Календарный график работы над проектом на весь период проектирования (с обозначением сроков выполнения и трудоемкости отдельных этапов):

разделы 1,2 к 01.03.23 – 20 %;
разделы 3,4 к 01.04.23 – 30 %;
разделы 5,6 к 01.05.23 – 30 %;
оформление пояснительной записки и графического материала к 15.05.23 г. 20 %Защита курсового проекта с 25.05.23 по 30.05.23г.

РУКОВОДИТЕЛЬ _____ *Л. П. Поденок*
(подпись)

Задание принял к исполнению _____ *Е. В.Максимчик*
(подпись)

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Курсовой проект на тему P2P-Chat с использованием «ncurses» представляет собой разработку чат-приложения на основе протокола «Peer-to-Peer» (P2P) с использованием библиотеки «ncurses» для создания интерфейса командной строки. P2P-чат позволяет пользователям обмениваться сообщениями напрямую, без необходимости централизованного сервера.

Цель данного проекта заключается в разработке удобного и эффективного способа общения между пользователями, где каждый участник является как клиентом, так и сервером одновременно. Это позволяет достичь децентрализованной архитектуры, где сообщения могут передаваться напрямую между участниками, минуя централизованный сервер.

Для реализации такой системы чата мы используем библиотеку ncurses, которая предоставляет функциональность для создания интерфейсов командной строки с возможностью манипуляции окнами, рисования на экране и обработки пользовательского ввода. «Ncurses» обеспечивает удобство использования и возможность создания интерактивного и интуитивно понятного пользовательского интерфейса для P2P-чата.

В ходе разработки курсового проекта мы планируем создать клиентскую и серверную части приложения. Клиентская часть будет обрабатывать пользовательский ввод, отправлять и принимать сообщения от других участников P2P-сети, а также отображать их на экране с использованием библиотеки «ncurses». Серверная часть будет отвечать за установление связи между клиентами, пересылку сообщений и поддержку P2P-архитектуры.

Ожидаемый результат проекта - полноценное P2P-чат-приложение, способное обеспечить обмен сообщениями между участниками P2P-сети с использованием интерфейса командной строки, реализованного с помощью библиотеки «ncurses». Пользователи смогут подключаться друг к другу, обмениваться текстовыми сообщениями и управлять своим присутствием в сети.

Разработка данного проекта предоставит возможность изучить и применить основные принципы работы P2P-сетей, научиться использовать библиотеку «ncurses» для создания интерфейсов командной строки и развить навыки программирования, такие как сетевое взаимодействие, обработка пользовательского ввода и многопоточность.

Кроме того, разработка P2P-чата с использованием ncurses позволит изучить и реализовать некоторые важные функциональности, включая:

1. Установление соединения между клиентами. Клиентская часть приложения должна иметь возможность обнаруживать и подключаться к другим клиентам в P2P-сети. Это может включать в себя обмен информацией о доступных клиентах и установление надежных соединений для передачи сообщений.
2. Обмен сообщениями. Когда соединение установлено, клиенты должны иметь возможность отправлять и получать сообщения друг от друга.

Это может включать в себя механизмы управления потоком сообщений, обработку ошибок и обеспечение доставки сообщений к адресату.

3. Интерфейс пользователя. С использованием библиотеки ncurses, мы сможем создать удобный и интуитивно понятный интерфейс командной строки для чата. Это может включать в себя отображение списка доступных участников, истории сообщений, ввода и отправки сообщений, а также другие функциональные элементы для повышения удобства использования.

4. Безопасность и конфиденциальность. При разработке P2P-чата важно обеспечить безопасность и конфиденциальность передаваемых сообщений. Мы можем рассмотреть возможности шифрования и аутентификации для защиты приватности пользователей и предотвращения несанкционированного доступа к данным.

В результате успешного завершения проекта мы получим функциональное P2P-чат-приложение, которое позволит пользователям обмениваться сообщениями в режиме реального времени, используя децентрализованную архитектуру. Это предоставит пользователю возможность общения без необходимости полагаться на централизованный сервер и будет способствовать более надежной и гибкой коммуникации.

В дальнейшем, проект может быть доработан и расширен для включения дополнительных функциональностей, таких как передача файлов, групповые чаты и расширенные возможности аутентификации и шифрования.

1 ОБЗОР ЛИТЕРАТУРЫ

1.1 Анализ существующих аналогов

Тема курсового проекта была выбрана для углубленного изучения языка программирования C, а также получения опыта и знаний в проектировании пользовательских приложений, знакомства с работой компьютерной сети

«peer-to-peer» основанной на равноправии участников. Для создания корректно работающего приложения необходимо иметь представление о существующих аналогах, изучить их преимущества и недостатки, выделить основные концепции для соблюдения при проектировании собственного приложения.

1.1.1 Telegram

Telegram – кроссплатформенный мессенджер с функцией VoIP (англ. Voice over Internet Protocol), предоставляющий возможности пользователям для обмена текстовыми, голосовыми и видео сообщениями, фотографиями и файлами различных форматов. Имеется необходимый функционал для проведения видео и аудио-звонков, создание конференций и многопользовательских чатов и каналов.

Для связи «Telegram» использует «peer-to-peer» соединение, что позволяет обмениваться информацией намного быстрее. Наиболее ценноепреимущество в «peer-to-peer» соединении – это экономия интернет-трафика и улучшенное качество связи. На рисунке 1.1 представление графический интерфейс «Telegram».

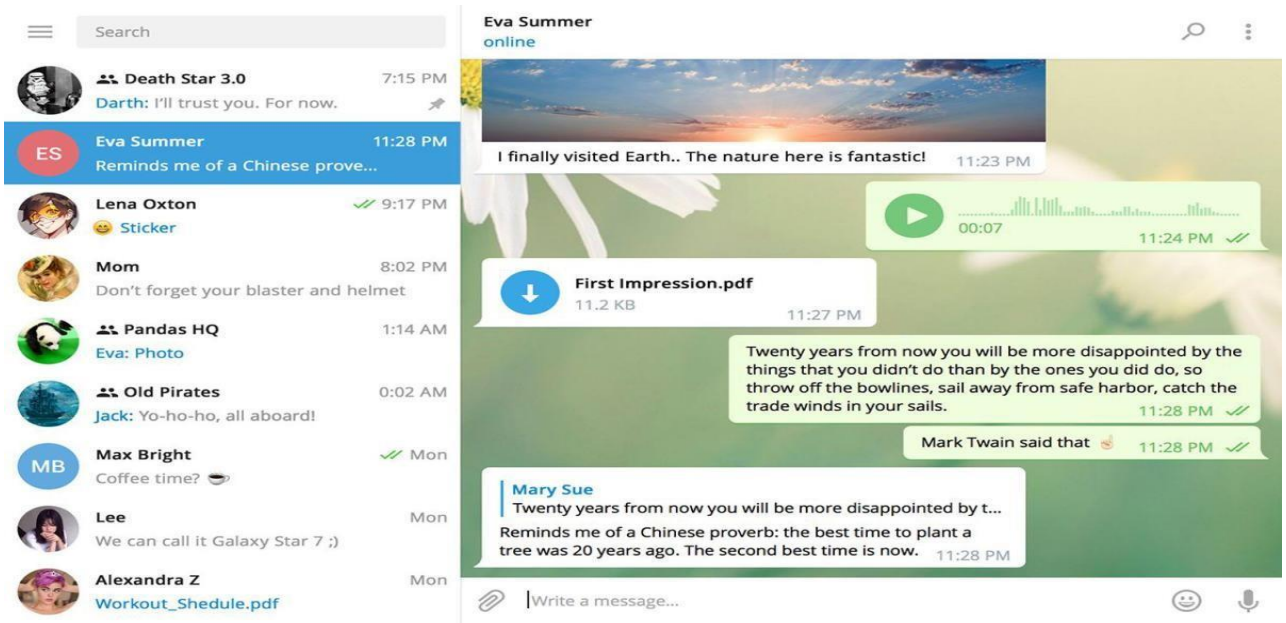


Рисунок 1.1 – Изображение интерфейса «Telegram»

Приложение «Telegram» было разработано компанией «Telegram Messenger» и выпущено в открытый доступ 14 августа 2013г. Для реализации данной программы использовались языки программирования «C++» и «Java», графический интерфейс реализован с помощью фреймворка Qt. «Telegram» является кроссплатформенным приложением с поддержкой для всех популярных операционных систем (ОС) и активно обновляется по сегодняшний день.

1.1.2 StrongDC++

«StrongDC++» - клиентское приложение для обмена файлами в пиринговых «peer-to-peer» сетях «Direct Connect» (основанных на принципе «расшаривания» ресурсов), базирующийся на исходном коде «DC++». Приложение поддерживает обмен данных по протоколам NMDC (NeoModus Direct Connect) и ADC (Advanced Direct Connect), созданных для пиринговых сетей. Графический интерфейс программы «StrongDC++» представлен на рисунке 1.2.

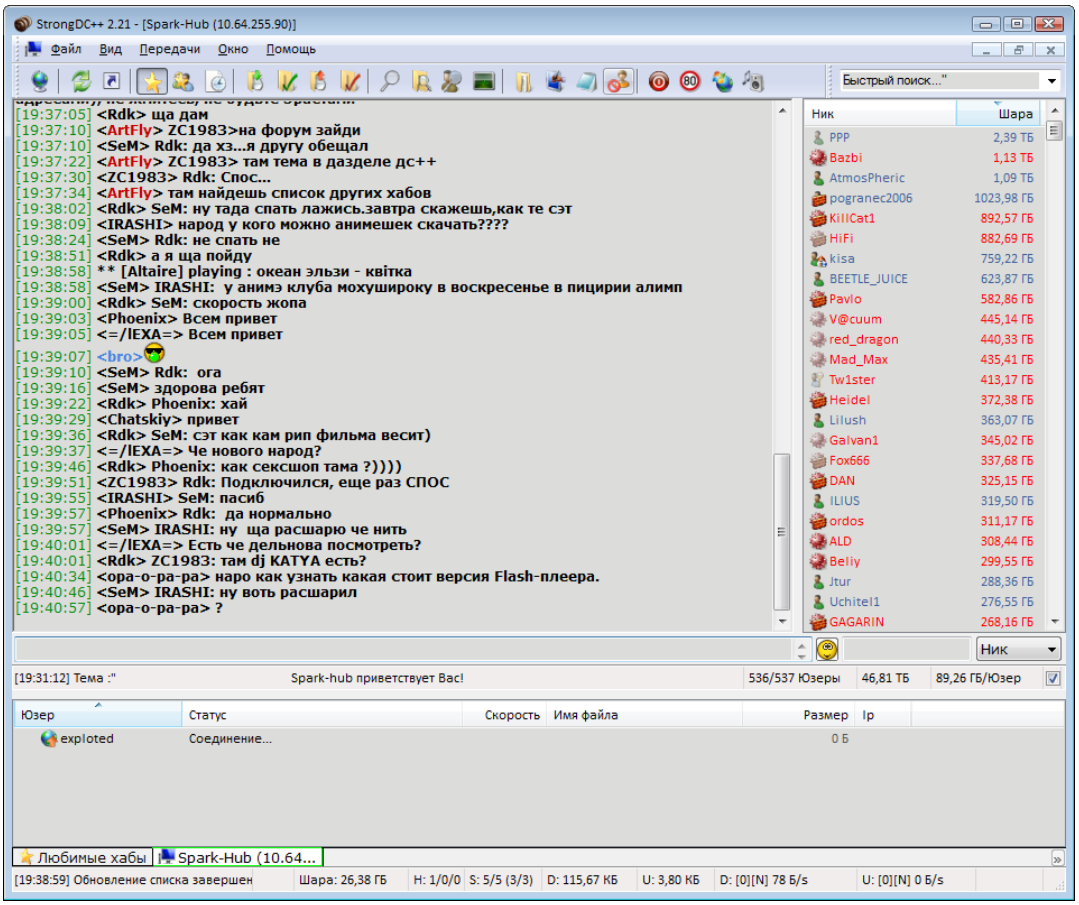


Рисунок 1.2 – Изображение интерфейса «StrongDC++»

Программа была разработана компанией «BigMuscle», написана на языке программирования «C++» для операционных систем «Windows XP/7/Vista». На данный момент приложение не поддерживается.

1.1.3 BitMessage

«BitMessage» (также известный как «BitMessage» P2P-система обмена сообщениями, разработанная для обеспечения безопасной и приватной коммуникации между пользователями. Он основан на идеях шифрования, децентрализации и анонимности. «BitMessage» также предлагает дополнительные функции, такие как подпись сообщений, возможность создания адресной книги контактов и групповые чаты. Кроме того, «BitMessage» является открытым исходным кодом, что позволяет разработчикам изучать и улучшать его безопасность и функциональность.

Однако, стоит отметить, что «BitMessage» может быть несколько сложным для новых пользователей из-за своей сложной архитектуры и требования к выполнению Proof-of-Work. Кроме того, скорость доставки сообщений в Bitmessage может быть ниже, чем в централизованных системах, из-за необходимости распределения сообщений через сеть узлов.

В целом, «BitMessage» представляет собой интересную P2P-систему обмена сообщениями, которая обеспечивает безопасность, приватность и анонимность. Графический интерфейс программы «BitMessage» представлен на рисунке 1.3.

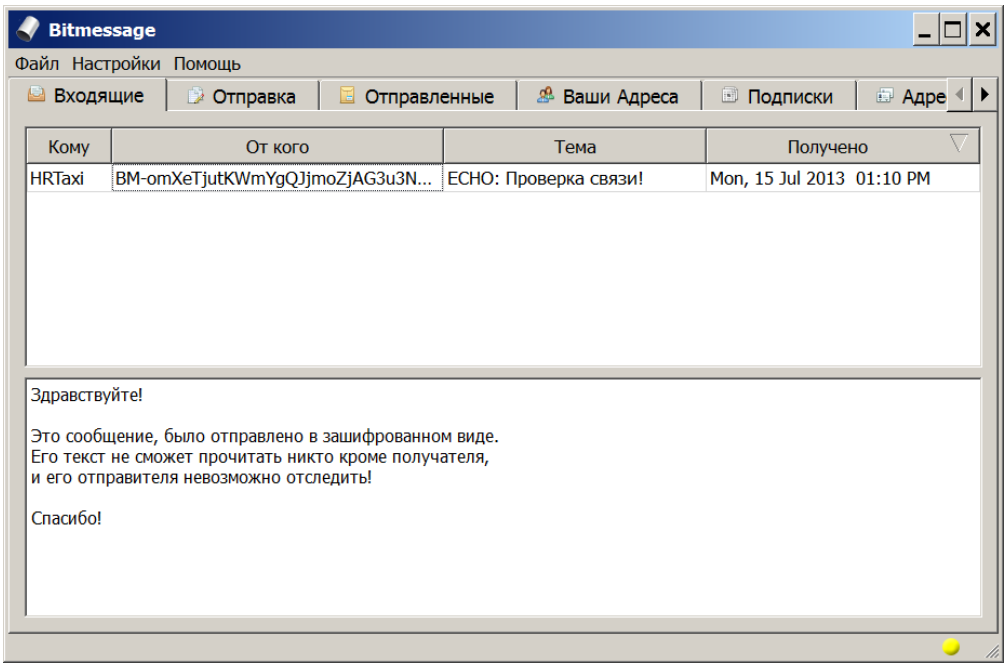


Рисунок 1.3 – Изображение интерфейса «BitMessage»

1.2 Постановка задачи

После рассмотрения аналогичных приложений можно сделать вывод, что все они обладают различным спектром функций. Можно выделить несколько ключевых особенностей, которые будут реализованы в программе данного курсового проекта:

- Использование библиотеки ncurses для создания интерактивного пользовательского интерфейса в консоли. Интерфейс должен предоставлять удобные элементы управления, такие как окна для отображения сообщений и ввода текста, списки пользователей, и другие элементы, позволяющие пользователям удобно взаимодействовать с чатом.;
- В приложении должна быть реализована функция хранения отправленных сообщений;
- Пользователи должны иметь возможность отправлять текстовые сообщения другим участникам чата. Сообщения должны быть отображены в соответствующих окнах, разделенных по пользователям или группам. Важно обеспечить доставку сообщений в режиме реального времени, чтобы пользователи могли немедленно видеть и отвечать на новые сообщения;
- Реализация приложения на основе протокола P2P (peer-to-peer). Каждый участник должен иметь возможность подключаться непосредственно к другим участникам чата без необходимости центрального сервера. Это позволит обмениваться сообщениями напрямую между пользователями, повышая приватность и уменьшая нагрузку на сеть.;

В качестве средств для выполнения курсового проекта был выбран язык программирования «C» 11-ой версии, по причине наивысшей производительности в сравнении с другими языками. Для реализации пользовательского интерфейса была выбрана библиотека «ncurses», написанная на языках «C» и «Ada», предназначенная для управления вводом-выводом на терминал, а также позволяющая задавать экранные координаты (взнакоместах). Для разработки и тестирования приложения использовалась операционная система Linux, дистрибутив - Garuda Linux.

Данный список средств позволяет реализовать все поставленные задачи для приложения, разрабатываемого в курсовом проекте.

2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ

После определения требований к функционалу разрабатываемого приложения его следует разбить на функциональные блоки. Такой подход упростит понимание проекта, позволит устранить проблемы в архитектуре, обеспечит гибкость и масштабируемость программного продукта в будущем путем добавления новых блоков.

2.1 Модуль пользовательского интерфейса

Модуль пользовательского интерфейса предназначен для взаимодействия программы с пользователем, основываясь на интуитивно понятном текстовом интерфейсе.

Для разрабатываемого сетевого приложения была выбрана библиотека «ncurses» для создания текстового интерфейса в терминале для системы «Linux». Данный пользовательский интерфейс представляет собой набор нескольких прямоугольных блоков в терминале, в которых содержится вся необходимая информация для пользователя, список клиентов, информация о пользователе, блок с сообщениями из чата, место для ввода сообщений.

2.2 Модуль работы с приложением

Модуль работы с приложением необходим для взаимодействия пользователя и программы, находится на уровень ниже графического интерфейса и является неотъемлемой частью приложения, дополняющий основной функционал, например:

- Просмотр подключенных пользователей;
- Просмотр отправленных сообщений в чате;
- Отправка сообщений другим пользователям;
- Просмотр собственной информации.

2.3 Модуль настройки приложения

Модуль настройки приложения позволяет пользователю подключится к необходимому серверу благодаря вводу необходимого IP адреса и порта сервера в виде аргументов командной строки. После чего приложение выбирает, необходимо создать новый сервер-чат или подключится к уже существующему.

2.4 Модуль чтения и записи данных

Данный модуль должен отвечать за считывание информации от пользователя при оправке сообщений, её записи в используемую область памяти для дальнейшей работы, а также считывание и запись всей другой необходимой информации по мере работы с приложением.

2.5 Модуль преобразования данных

Модуль преобразования данных необходим для преобразования данных, считываемых от пользователя или из приложения для корректной работы с ними.

2.6 Модуль сети

Модуль сети предназначен для подключения пользователей друг к другу и необходим для их взаимодействия в реальном времени.

Данный модуль должен отвечать за передачу сообщений по технологии peer-to-peer между пользователями и отслеживать активность клиентов в приложении.

3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

В приложении все необходимые методы разбиты на логические структуры и распределены по отдельным файлам. Ниже представлен функционал и структура разрабатываемого приложения.

3.1 Socket

В данном файле описываются необходимые методы для работы приложения в сети используя сокеты с помощью существующей библиотеки «socket.h».

Функции:

- `int create_socket` – функция необходима для создания сокета, возвращаемое число – дескриптор сокета;
- `void close_socket` – функция необходима для закрытия сокета;
- `void bind_address` – функция необходима для назначения локального адреса сокету;
- `void send_udp` – функция для отправки сообщений;
- `int socket_read` – блокирующая функция для получения данных пришедших на сокет, возвращает количество байт;
- `void set_nonblock_flag` – установка неблокирующего флага для дескриптора, для продолжения работы;
- `int is_equal_address` – проверка двух адресов на эквивалентность;
- `void create_adress` – создание структуры адреса для работы в сокете.

3.2 Packet

В данном файле определяются пакеты – блоки данных, передаваемые по сети в пакетном режиме, которые будут использоваться в приложении чата. Так же будут определены функции для работы с этими пакетами.

- `PACKET_CONNECT_REQUES - '0'` – пакет отправляется одному из клиентов для подключения к сети. В ответ клиент должен отправить пакет `PACKET_CONNECT_ACCEPT`.
- `PACKET_CONNECT_ACCEPT - '1'` – пакет отправляется в ответ на `PACKET_CONNECT_REQUEST`;
- `PACKET_PING '3'` – пакет отправляется для проверки активности клиента.
- `PACKET_TIMEOUT '4'` – пакет отправляется клиенту, если он долго не отвечал на запросы, при этом клиент убирается из массива клиентов, если пришёл данный пакет, необходимо отправить повторный

PACKET_CONNECT_REQUEST;

- PACKET_REQUEST_USERS ‘5’ – пакет отправляется клиенту для получения всех подключенных клиентов, в ответ приходит список всех подключенных клиентов;
- PACKET_LIST_USERS ‘6’ – пакет со списком всех подключенных клиентов;
- PACKET_SEND_MESSAGE ‘7’ – пакет отправляется всем клиентам, содержит сообщение.

Функции:

- int get_packet_id – получаем id пакета;
- int create_simple_packet – функция для простых однобайтовых пакетов;
- int create_connect_request_packet – функция для пакета с запросом на подключение;
- int create_connect_accept_packet – функция для пакета с подтверждением на подключение;
- int create_message_packet – создание пакета с сообщением для клиентов;
- int create_list_users_packet – создание пакета с списком клиентов.

3.3 Interface

Данный файл содержит определение функций для создания графического интерфейса в терминале с помощью утилиты «ncurses».

- void interface_init – функция для инициализации всех блоков интерфейса;
- void interface_close – функция для завершения работы графического интерфейса;
- void update_client_box – функция для обновления блока со списком клиентов, если были подключены новые клиенты;
- void update_info_box – функция для обновления блока информации о клиенте;
- void add_message – функция для добавления сообщения в блок сообщений;
- void read_input – функция для чтения вводимых символов из блока ввода текстовой информации.

3.4 Utils

Данные файлы содержат функции необходимые для получения аргументов командной строки в надлежащем виде для работы приложения

- `parse_connect_address` – функция необходимая для получения массива символов содержащих адрес подключения клиента к серверу;
- `parse_source_port` – функция необходимая для получения порта на котором будет открыт сокет клиента;
- `parse_name` – функция необходимая для получения имени сервера/клиента которое будет использоваться в графическом интерфейсе для идентификации разных пользователей.

3.5 Clients

В файлах «Clients» описана структура, содержащая основную информацию о клиентах и объявлен статический массив структур, размером `MAX_CLIENTS`. Так же описаны функции реализующий функционал для работы приложения с клиентами.

Функции:
- `add_client` – функция добавляющая информацию о новом клиенте в массив клиентов;
- `get_client` – функция возвращающая структуру содержащую информацию о клиенте;
- `is_exist` – проверяет наличие клиента с заданным адресом;
- `remove_client` – удаляет информацию о клиенте из программы.

3.6 Config

В данном заголовочном файле содержатся определение основных констант, используемых в ходе работы приложения.

- `TICK_PER_SECOND` – стандартное значение 10, для регулировки задержки в работе приложения;
- `SEND_PING_PAUSE` – стандартное значение 10, для регулировки времени на ответ от клиента;
- `PING_SKIP_TO_TIMEOUT` – стандартное значение 10, количество пакетов которое даётся клиенту на ответ для сервера;
- `BUFLEN` – стандартное значение 512, описывает размер буфера для работы с массивами символов;
- `MAX_CLIENTS` – стандартное значение 20, описывает максимальное количество клиентов для одновременного подключения к серверу;

- `DEFAULT_PORT` – стандартное значение 8888, порт на котором открывается сокет сервера;
- `MAX_NAME_LENGTH` – стандартное значение 13, размер буфера для хранения имени клиента.

3.7 Chat

В данных файлах расположены функции для выхода из разработанного приложения, отправки пакета и подключения клиентов.

- `void escape` – функция для выхода из приложения при возникновении ошибок с выводом сообщения для пользователей;
- `void connect_to_client` – функция для подключения к серверу;
- `void send_packet` – функция для отправки пакетов.

4 РАЗРАБОТКА ПРОГРАМНЫХ МОДУЛЕЙ

4.1 Функция `bind_address`

Шаг 1. Начало.

Шаг 2. Заполняем структуру `sockaddr_in`.

Шаг 3. Инициализируем поле `sin_family` значение `AF_INET`

Шаг 4. Инициализируем поле `sin_port` значением введённого пользователем порта, используя функцию `htonl` для перевода порядка байт из формата хоста в формат сети.

Шаг 5. Инициализируем поле `sin_addr.s_addr` используя функцию `htonl` в паре с константой `IN_ADDRANY`, отвечающая за все адреса локального хоста (0.0.0.0).

Шаг 6. Вызов функции `bind` для связывания сокета с заданной структурой и сохранение результата в переменную `result`.

Шаг 7. Если result равен -1, то не удалось связать адрес и структуру, вызывается функция `escape()` с сообщением об ошибке.

Шаг 8. Конец.

4.2 Функция `read_input_box`

Шаг 1. Начало

Шаг 2. Инициализация переменной `symbol` с нулевым значением.

Шаг 3. Вход в цикл `while`, который будет выполняться до тех пор, пока `symbol` = `wgetch(box_input)` не будет равно ERR (ошибка) — это означает, что символ считывания не был получен или произошла ошибка.

Шаг 4. Внутри цикла происходит проверка значения `symbol` на различные события:

- Если `symbol` равен символу новой строки (\n), то выполняется очистка содержимого `box_input` и возвращается значение 1 для указания наличия ввода.
- Если `symbol` равен символу `"Backspace"`, то происходит проверка, что `size` больше 0. Если `true`, то очищается символ в `box_input` и значение в `buf` уменьшается на 1.
- Если `size` меньше 99 (предел длины ввода), то символ `symbol` добавляется в `buf` и значение `size` увеличивается на 1.

Шаг 5. Если цикл завершается (нет больше символов для считывания или произошла ошибка), то содержимое `buf` выводится на `box_input`.

Шаг 6. Функция возвращает 0, чтобы указать отсутствие ввода.

Шаг 7. Конец

4.3 Основной алгоритм работы приложения

Шаг 1. Начало

Шаг 2. Инициализация переменной address_size с размером local_address.

Шаг 3. Вход в цикл while, который будет выполняться до тех пор, пока socket_read не вернет значение -1.

Шаг 4. Чтение данных из сокета с помощью socket_read, передавая в качестве аргументов sockfd, указатель на буфер buf_read, указатель на структуру buf_address и указатель на переменную address_size.

Шаг 5. Проверка, если адрес в buf_address совпадает с local_address, то продолжить выполнение цикла с помощью оператора continue.

Шаг 6. Получение packet_id из данных, хранящихся в buf_read, с помощью функции get_packet_id.

Шаг 7. Получение указателя на клиента (struct Client) с помощью функции get_client, передавая buf_address.

Шаг 8. Если клиент равен NULL и packet_id не равен PACKET_CONNECT_REQUES и не равен PACKET_CONNECT_ACCEPT, то продолжить выполнение цикла с помощью оператора continue.

Шаг 9. Если клиент не равен NULL, то проверить, если packet_id не равен PACKET_PING, то создать простой пакет с PACKET_PING с помощью функции create_simple_packet, записать его в buf_send, и отправить его с помощью send_udp.

Шаг 10. Добавление завершающего нуля ('\0') к buf_read.

Шаг 11. Получение строки с IP-адресом из buf_address с помощью inet_ntoa и сохранение в buf_ip.

Шаг 12. Получение порта из buf_address с помощью ntohs и сохранение в buf_port.

Шаг 13. Выполнение операций в зависимости от значения packet_id, используя оператор switch.

Шаг 14. Для каждого случая (case) PACKET_CONNECT_REQUES, PACKET_CONNECT_ACCEPT, PACKET_PING, PACKET_TIMEOUT, PACKET_SEND_MESSAGE, PACKET_REQUEST_USERS, PACKET_LIST_USERS, выполняются соответствующие операции.

Шаг 15. Повторение цикла while с шага 3, если условие выполнено.

Шаг 16. Конец.

4.4 Алгоритм функции connect_to_client

Шаг 1. Начало

Шаг 2. Объявляем массив символов buf размером 100 и переменную buf_size, которую будем использовать для хранения размера пакета.

Шаг 3. Начинаем бесконечный цикл while(1).

Шаг 4. Создаем UDP-пакет для запроса на соединение с помощью функции create_connect_request_packet, передавая в качестве аргументов имя пользователя name и указатель на массив buf. Размер пакета сохраняем в переменной buf_size.

Шаг 5. Отправляем UDP-пакет на адрес addr, используя функцию send_udp, передавая в качестве аргументов дескриптор сокета sockfd, адрес addr, массив buf и его размер buf_size.

Шаг 6. Спим на 2 секунды, чтобы дать серверу время на обработку запроса.

Шаг 7. Создаем структуру sockaddr_in buf_address, которую будем использовать для хранения адреса, с которого пришел ответ от сервера. Инициализируем переменную address_size размером sizeof(struct sockaddr_in).

Шаг 8. Запускаем вложенный цикл while, который будет читать входящие данные из сокета sockfd с помощью функции socket_read, передавая в качестве аргументов дескриптор сокета sockfd, указатель на массив buf, указатель на структуру sockaddr_in buf_address и указатель на переменную address_size. Функция возвращает размер прочитанных данных в переменную buf_size.

Шаг 9. Если размер данных равен -1, значит произошла ошибка чтения из сокета, и мы переходим к следующей итерации вложенного цикла, чтобы продолжить попытки чтения.

Шаг 10. Если размер данных не равен -1, то мы проверяем тип пакета с помощью функции get_packet_id, передавая в качестве аргумента указатель на массив buf. Если тип пакета PACKET_CONNECT_ACCEPT, и адрес, с которого пришел ответ, совпадает с адресом, на который мы отправляли запрос на соединение, то:

Шаг 11. Извлекаем имя пользователя из пакета и сохраняем его в массив buf_name, используя функцию strcpy.

Шаг 12. Добавляем клиента с адресом buf_address и именем buf_name с помощью функции add_client.

Шаг 13. Обновляем список клиентов в окне приложения с помощью функции update_client_box.

Шаг 14. Создаем строку с сообщением о том, что мы успешно подключились к клиенту с именем buf_name, используя функцию sprintf, и сохраняем ее в массив buf.

Шаг 15. Добавляем сообщение в окно приложения с помощью функции `add_message`.

Шаг 16. Создаем UDP-пакет для запроса списка пользователей с помощью функции `create_simple_packet`, передавая в качестве аргументов тип пакета `PACKET_REQUEST_USERS` и указатель на массив `buf`. Размер пакета сохраняем в перемененную.

Шаг 17. Отправляем UDP-пакет на адрес `addr`, используя функцию `send_udp`, передавая в качестве аргументов дескриптор сокета `sockfd`, адрес `addr`, массив `buf` и его размер `buf_size`.

Шаг 18. завершаем функцию с помощью оператора `return`.

Шаг 19. Если тип пакета не равен `PACKET_CONNECT_ACCEPT` или адрес не совпадает, мы переходим к следующей итерации вложенного цикла, чтобы продолжить чтение данных из сокета.

Шаг 20. После завершения вложенного цикла `while` мы возвращаемся к началу внешнего цикла и повторяем процесс запроса на соединение через UDP.

Шаг 21. Конец

5 РУКВОДСТВО ПОЛЬЗОВАТЕЛЯ

5.1 Сборка и компиляция программного обеспечения

Данное приложение является консольной утилитой и запускается в терминале UNIX подобных операционных систем. Для компиляции и сборки приложения используется «cmake» - кроссплатформенное программное обеспечение для сборки программного обеспечения из исходного кода. Сценарий сборки описан в файле CMakeLists.txt. Для начала сборки необходимо ввести команду «cmake .» в терминал. В результате выполнения команды, идёт проверка на наличие установленной библиотеки «ncurses» и остальных ошибок, при отсутствии ошибок происходит создание «Makefile» для сборки приложения. После создания Makefile`а необходимо ввести команду «make» для сборки приложения.

5.2 Запуск приложение в режиме хостинга

Для запуска экземпляра-хостинга необходимо ввести следующую команду в терминал, где <hosting name> - имя пользователя, которое будет показано для остальных пользователей во время работы приложения и обмена сообщениями.

```
$./C_P2P_Chat -name <hosting name>                    (5.1)
```

После выполнения команды изменяется размер окна терминала и запускается приложение, ожидающее подключения пользователей. Пример запуска приложения приведён на рисунках 5.3

Рисунок 5.3 – окно после запуска экземпляра-хостинга

5.3 Запуск приложения в режиме пользователя

Для запуска экземпляра-пользователя необходимо ввести команду (5.2) в терминал, где <client name> - имя пользователя. <connect ip> - IP экземпляра-хостинга, <connect port> - порт экземпляра-хостинга, на которых был запущен экземпляр-хостинг. Для экземпляра-хостинга используется 8888 порт. <client port> - порт, который будет использоваться пользователем для обмена информацией с хостингом.

```
$./C_P2P_Chat -name <client name> -connect <connect ip>
<connect port> -port <client port>
```

После выполнения команды изменяется размер окна терминала и запускается приложение в режиме ожидания подключения к хостингу. При успешном подключении к хостингу выводится соответствующее сообщение в окно с сообщениями. Экземпляр-хостинг так же получает информационное сообщение о подключении нового пользователя. Пример запуска приведен на рисунке 5.4.

Рисунок 5.4 – окно после запуска экземпляра-пользователя

6 ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ

Раздел "Программа и методика испытаний" представляет собой важную составляющую курсовой работы, поскольку он описывает план и процедуру, которые будут использоваться для проверки работоспособности и эффективности разработанной программы или системы. В данном разделе будет представлена информация о целях, задачах, методике, плане и ожидаемых результатах испытаний. Кроме того, будут также рассмотрены возможные риски и ограничения, которые могут повлиять на проведение испытаний.

6.1 Цель испытаний

Целью проведения испытаний является проверка работоспособности и эффективности разработанной программы в соответствии с заданными требованиями. Главной целью является убедиться, что программа работает корректно, выполняет необходимые функции и отвечает требованиям, установленным в начальном проекте.

6.2 Задачи испытаний

В рамках испытаний будут выполнены следующие задачи:

1. Проверка функциональности: Оценка работы модуля сети и обмена сообщений с целью обнаружения и исправления ошибок и несоответствий требованиям. В данной задаче будет уделено внимание проверке правильности обмена сообщений между пользователями.
2. Проверка надёжности: Определение устойчивости к ошибкам и исключительным ситуациям. Будет проведено тестирование на корректность подключения, а также проверка обработки ошибок.

6.3 Испытания

Испытания проводились на операционной системе Linux, с использованием дистрибутива Garuda Linux, на 64-ёх разрядном персональном компьютере внутри домашней локальной сети. Перед тестированием было подготовлено окружение, установлены необходимые для работы приложения пакеты и библиотеки, такие как «ncurses», «cmake», «make», «gcc». Также был собран и скомпилирован проект для запуска приложения.

Для проверки функциональности использовались два экземпляра приложения, хостинга и пользователя. Было запущено приложение от лица хостинга, после чего присоединение клиента и отправлено сообщение от лица его лица. Результат работы продемонстрирован на рисунках 6.1 и 6.2.

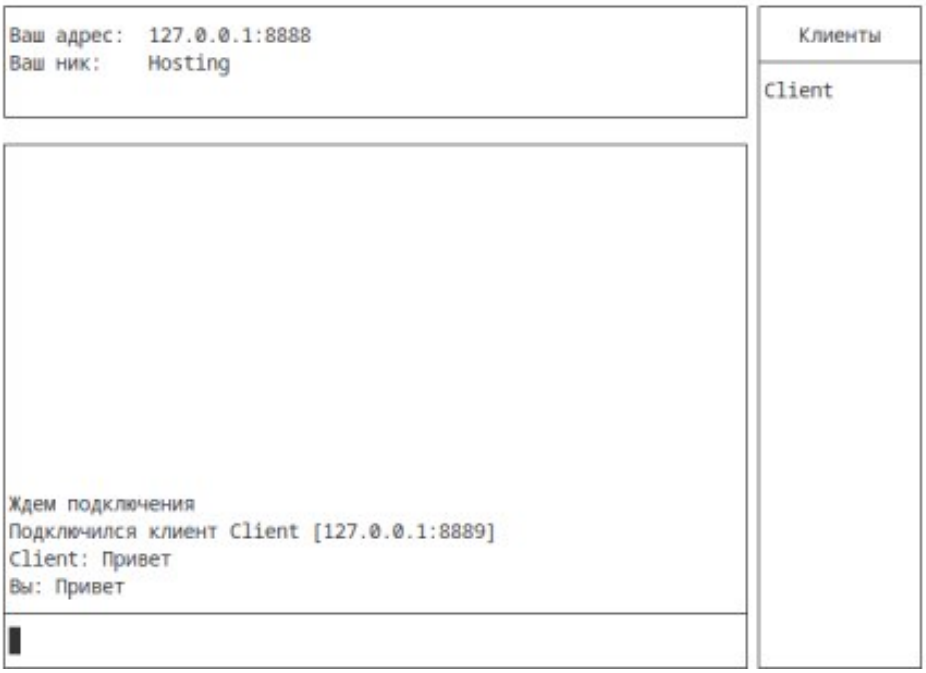


Рисунок 6.1 – демонстрация работы приложения

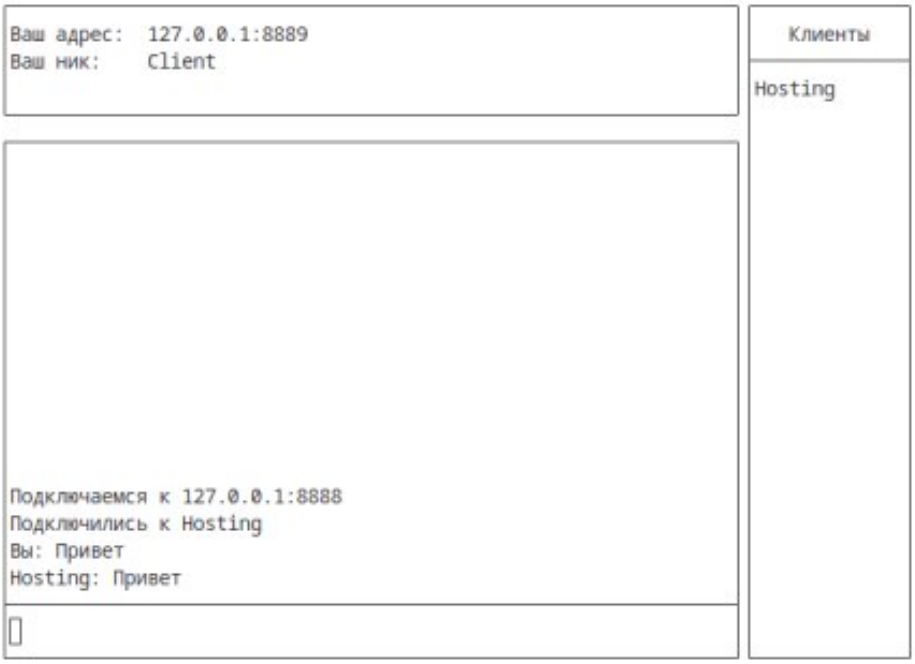


Рисунок 6.2 – демонстрация работы приложения

Можно сделать вывод что функционал обмена сообщений и работа приложения в целом выполнена корректно.

ЗАКЛЮЧЕНИЕ

В результате работы над данным курсовым проектом было разработано работоспособное приложение со своим набором функций и графическим интерфейсом. Данный курсовой проект был разработан в соответствии с поставленными задачами, весь функционал был реализован в полном объеме.

Для создания программного продукта была подробно исследована технология peer-to-peer и принцип работы сетевого чата. В ходе разработки были углублены знания языка программирования C, а также получен опыт работы с библиотекой «ncurses», используемой для разработки текстовых интерфейсов в терминале.

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.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

[1] . Брайан “Beej Jorgensen” Холл [Электронный ресурс] / Сетевое программирование от Биджа Использование Интернет Сокетов. Режимдоступа: https://beej.us/guide/bgnet/html/split/ Дата доступа: 15.03.2022

[2] . Visual Studio Code - Электронные данные - Режим доступа: https://code.visualstudio.com/docs - Дата доступа: 15.05.2023.

[3] . GCC Wiki [Электронный ресурс] - Электронные данные - Режим доступа: https://gcc.gnu.org/wiki - Дата доступа: 15.05.2023.

[4] . NCURSES Programming HOWTO [Электронный ресурс] - Электронные данные - Режим доступа: https://tldp.org/HOWTO/NCURSES-Programming-HOWTO/ - Дата доступа: 15.05.2023.

[5] . GCC make [Электронный ресурс] - Электронные данные - Режим доступа: https://www.gnu.org/software/make/manual/make.html - Дата доступа: 15.05.2023.

[6] . The Linux Programming Interface: A Linux and UNIX System Programming Handbook / Michael Kerrisk – No Starch Press, 2010.

ПРИЛОЖЕНИЕ А

(обязательное)

Листинг кода

```
### main.c

#include "Chat.h"
#include "Utils/Utils.h"

int main(int argc, char *argv[]) {
    int connect_port = DEFAULT_PORT;
    int source_port = DEFAULT_PORT;
    char* connect_ip = NULL;
    char name[MAX_NAME_LENGTH] = "";

    parse_connect_adress(argc, argv, &connect_ip, &source_port);
    parse_source_port(argc, argv, &source_port);
    parse_name(argc, argv, (char *) &name);

    if (strcmp((char *) &name, "") == 0) {
        escape("Необходимо ввести имя: -name <имя>");
    }

    fflush(stdin);
    interface_init();

    struct sockaddr_in local_address;
    struct sockaddr_in buf_address;

    char buf_read[BUFLEN] = {0};
    char buf_send[BUFLEN] = {0};
    char buf_name[MAX_NAME_LENGTH] = {0};

    int buf_read_size = 0;
    int buf_send_size = 0;

    int sockfd = create_socket();

    bind_address(sockfd, &local_address, source_port);

    char* source_ip = inet_ntoa(local_address.sin_addr);

    update_info_box((char *) &name, source_ip, source_port);
    add_client(&buf_address, (char *) &name);

    sen_nonblock_flag(sockfd);
    sen_nonblock_flag(0);

    if (connect_ip != NULL) {
        create_adress(connect_ip, connect_port, &buf_address);

            sprintf((char *) &buf_send, "Подключаемся к %s:%d", connect_ip,
connect_port);
        add_message((char *) &buf_send);

        connect_to_client(sockfd, &buf_address, (char *) &name);
    } else {
        add_message("Ждем подключения");
    }

    int timeToSendPing = SEND_PING_PAUSE;
    while (1) {
        unsigned int address_size = sizeof(local_address);
            while ((buf_read_size = socket_read(sockfd, (char *) &buf_read,
```

```
&buf_address, &address_size)) != -1) {
                    if (is_equal_address(&local_address, &buf_address)) {
                        continue;
                    }
                    int packet_id = get_packet_id((char *) &buf_read);

                    struct Client* client = get_client(&buf_address);

                     if (client == NULL && packet_id != PACKET_CONNECT_REQUES && packet_id
!= PACKET_CONNECT_ACCEPT) {
                        continue;
                    }

                    if (client != NULL) {
                        if (packet_id != PACKET_PING) {
                            create_simple_packet(PACKET_PING, (char *) &buf_send);
                            send_udp(sockfd, &buf_address, (char *) &buf_send, 1);
                        }
                    }

                    buf_read[buf_read_size] = '\0';
                    char* buf_ip = inet_ntoa(buf_address.sin_addr);
                    int buf_port = ntohs(buf_address.sin_port);
                    switch (packet_id) {
                        case PACKET_CONNECT_REQUES:
                            if (!is_exist(&buf_address)) {
                                strcpy((char *) &buf_name, buf_read + 1);
                                add_client(&buf_address, (char *) &buf_name);
                                if (strcmp(buf_name, name) == 0) {
                                    break;
                                }
                                update_client_box();
                               sprintf((char *) &buf_send, "Подключился клиент %s [%s:%d]",
buf_name, buf_ip, buf_port);
                                add_message((char *) &buf_send);
                            }
                           buf_send_size = create_connect_accept_packet((char *) &buf_send,
(char *) &name);
                                    send_udp(sockfd,  &buf_address,  (char  *)  &buf_send,
buf_send_size);
                            break;
                        case PACKET_CONNECT_ACCEPT:
                            if (!is_exist(&buf_address)) {
                                strcpy((char *) &buf_name, buf_read + 1);
                                add_client(&buf_address, (char *) &buf_name);
                                update_client_box();

                               sprintf((char *) &buf_send, "Подключились к %s", buf_name);
                                add_message((char *) &buf_send);
                            }
                            break;
                        case PACKET_PING:
                            client->isActive = PING_SKIP_TO_TIMEOUT;
                            break;
                        case PACKET_TIMEOUT:
                            connect_to_client(sockfd, &buf_address, (char *) &name);
                            break;
                        case PACKET_SEND_MESSAGE:
                            get_name(client, (char *) &buf_name);
                            if (strcmp(buf_name, name) == 0) {
```

```
                            break;
                        }
                        sprintf((char *) &buf_send, "%s: %s", buf_name, buf_read + 1);
                        add_message(buf_send);
                        break;
                    case PACKET_REQUEST_USERS:
                        buf_send_size = create_list_users_packet((char *) &buf_send);
                                send_udp(sockfd, &buf_address, (char *) &buf_send,
buf_send_size);
                        break;
                    case PACKET_LIST_USERS:
                      buf_send_size = create_connect_request_packet((char *) &buf_send,
(char *) &name);
                        int count =  buf_read[1];
                        for (int i = 0; i < count; i++) {
                                 memcpy(&(buf_address), buf_read + 2, sizeof(struct
sockaddr_in));
                                 send_udp(sockfd, &buf_address, (char *) &buf_send,
buf_send_size);
                        }
                        break;
                }
            }
            static int size_input = 0;
            static char buf_input[100] = {0};
            while (read_input_box((char *) buf_input, &size_input) == 1) {
                if (strcmp(buf_input, "/quit") == 0) {
                    close_socket(sockfd);
                    interface_close();
                    return 0;
                }
                sprintf((char *) &buf_send, "Вы: %s", buf_input);
                add_message((char *) &buf_send);
                     create_message_packet((char *) &buf_send, (char *) &buf_input,
size_input);
                send_packet(sockfd, (char *) &buf_send, size_input + 1);
                memset(buf_input, 0, 100);
                size_input = 0;
            }

        }
        close_socket(sockfd);
        interface_close();
        return 0;
    }


    ### Clients.h
    #ifndef C_P2P_CHAT_CLIENTS_H
    #define C_P2P_CHAT_CLIENTS_H

    #include "Chat.h"

    struct Client {
        int isActive;
        char name[MAX_NAME_LENGTH];
        struct sockaddr_in address;
    };

    extern struct Client clients[MAX_CLIENTS];
```

```
void add_client(const struct sockaddr_in* addr, const char* name);

struct Client* get_client(const struct sockaddr_in* addr);

int is_exist(const struct sockaddr_in* addr);

void remove_client(struct Client* client);

void get_name(const struct Client* client, char* name);

#endif //C_P2P_CHAT_CLIENTS_H
```

```
### Clients.c

#include "Clients.h"

struct Client clients[MAX_CLIENTS] = {0};

void add_client(const struct sockaddr_in* addr, const char* name) {
    for (int i = 0; i < MAX_CLIENTS; i++) {
        if (clients[i].isActive <= 0) {
            memcpy(&(clients[i].address), addr, sizeof(struct sockaddr_in));
            strcpy((char *) &(clients[i].name), name);
            clients[i].isActive = PING_SKIP_TO_TIMEOUT;
            update_client_box();
            return;
        }
    }
}

struct Client* get_client(const struct sockaddr_in* addr) {
    for     (int i = 0; i < MAX_CLIENTS; i++) {
        if (clients[i].isActive > 0) {
            // Сравниваем ip
            if (is_equal_address(addr, &(clients[i].address))) {
                return &(clients[i]);
            }
        }
    }
    return NULL;
}

int is_exist(const struct sockaddr_in* addr) {
    return get_client(addr) != NULL;
}

void remove_client(struct Client* client) {
    client->isActive = 0;
    update_client_box();
}

void get_name(const struct Client* client, char* name) {
    strcpy(name, (char *) &(client->name));
}

### Chat.h
#ifndef C_P2P_CHAT_CHAT_H
#define C_P2P_CHAT_CHAT_H

#include <curses.h>
#include <locale.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <arpa/inet.h>
#include <sys/ioctl.h>

#include "Config.h"
#include "Clients.h"
#include "Network/Packet.h"
```

```
#include "Network/Socket.h"
#include "Utils/Interface.h"

void escape(const char* error);
void connect_to_client(int sockfd, const struct sockaddr_in* addr, const char*
name);
void send_packet(int sockfd, const char* buf, int buf_size);

#endif //C_P2P_CHAT_CHAT_H


#### Chat.c
#include "Chat.h"

void escape(const char* error) {
    printf("Error!!!\n");
    printf("%s\n", error);
    exit(EXIT_FAILURE);
}

void connect_to_client(int sockfd, const struct sockaddr_in* addr, const char* name)
{
    char buf[100];
    int buf_size = 0;

    while (1) {
        buf_size = create_connect_request_packet((char *) &buf, name);
        send_udp(sockfd, addr, buf, buf_size);
        sleep(2);

        struct sockaddr_in buf_address = {0};
        unsigned int address_size = sizeof(struct sockaddr_in);
           while ((buf_size = socket_read(sockfd, (char *) &buf, &buf_address,
&address_size)) != -1) {
              buf[buf_size] = '\0';
              int packet_id = get_packet_id((char *) &buf);
                 if (packet_id == PACKET_CONNECT_ACCEPT && is_equal_address(addr,
&buf_address)) {

                  char buf_name[MAX_NAME_LENGTH * 2];
                  strcpy((char *) &buf_name, buf + 1);
                  add_client(&buf_address, (char *) &buf_name);
                  update_client_box();

                  sprintf((char *) &buf, "Подключились к %s", buf_name);
                  add_message((char *) &buf);

                     buf_size = create_simple_packet(PACKET_REQUEST_USERS, (char *)
&buf);
                  send_udp(sockfd, addr, buf, buf_size);
                  return;
              }
        }
    }
}

void send_packet(int sockfd, const char* buf, int buf_size) {
    for (int i = 0; i < MAX_CLIENTS; i++) {
        if (clients[i].isActive > 0) {
            send_udp(sockfd, &(clients[i].address), buf, buf_size);
```

```
            }
        }
    }
```

Utils.h

```
    #ifndef C_P2P_CHAT_UTILS_H
    #define C_P2P_CHAT_UTILS_H

    #include "../Chat.h"

    void parse_connect_adress(int argc, char *argv[], char** ip, int* port);
    void parse_source_port(int argc, char *argv[], int* port);
    void parse_name(int argc, char *argv[], char* name);

    #endif //C_P2P_CHAT_UTILS_H
```

Utils.c

```
    #include "Utils.h"

    void parse_connect_adress(int argc, char *argv[], char** ip, int* port) {
        for (int i = 0; i < argc; i++) {
            if (strcmp(argv[i], "-connect") == 0 && (i + 2) <= argc) {
                *ip = argv[i + 1];
                *port = atoi(argv[i + 2]);

                if (*port == 0) {
                    escape("Неправильно указан порт подключения\n");
                }
                return;
            }
        }
    }

    void parse_source_port(int argc, char *argv[], int* port) {
        for (int i = 0; i < argc; i++) {
            if (strcmp(argv[i], "-port") == 0 && (i + 1) <= argc) {
                *port = atoi(argv[i + 1]);

                if (*port == 0) {
                    escape("Неправильно указан локальный порт\n");
                }
                return;
            }
        }
    }

    void parse_name(int argc, char *argv[], char* name) {
        for (int i = 0; i < argc; i++) {
            if (strcmp(argv[i], "-name") == 0 && (i + 1) <= argc) {
                strcpy(name, argv[i + 1]);
                return;
            }
        }
    }
```

Interface.h

```
    #ifndef C_P2P_CHAT_INTERFACE_H
```

```
#define C_P2P_CHAT_INTERFACE_H

#include "../Chat.h"

void interface_init();
void interface_close();
void update_client_box();
void update_info_box(const char* name, const char* ip, int port);
void add_message(const char* msg);
int read_input_box(char* buf, int* size);

#endif //C_P2P_CHAT_INTERFACE_H


### Interface.c
#include "Interface.h"

static WINDOW* box_info = NULL;
static WINDOW* box_client = NULL;
static WINDOW* box_messages = NULL;
static WINDOW* box_input = NULL;

static char messages[16][126] = {{0}};

static void init_info_box() {
    box_info = newwin(5, 65, 0, 0);
    box(box_info, 0, 0);
    wrefresh(box_info);
}

static void init_client_box() {
    box_client = newwin(25, 15, 0, 65);
    box(box_client, 0, 0);
    mvwprintw(box_client, 1, 1, "   Клиенты   ");
    mvwprintw(box_client, 2, 0, "├──────────────┤");
    wrefresh(box_client);
}

static void init_message_box() {
    box_messages = newwin(17, 65, 5, 0);
    box(box_messages, 0, 0);
                            mvwprintw(box_messages,             16,             0,
"│                                                             │");

    wrefresh(box_messages);
}

static void init_inpit_box() {
    box_input = newwin(3, 65, 22, 0);
    box(box_input, 0, 0);
                                mvwprintw(box_input,             0,             0,
"├─────────────────────────────────────────────────────────────┤");;

    wrefresh(box_input);
}

void update_client_box() {
    wclear(box_client);
    box(box_client, 0, 0);
    mvwprintw(box_client, 1, 1, "   Клиенты   ");
```

```
    mvwprintw(box_client, 2, 0, "├──────────────┤");
    int position = 3;
    for (int i = 0; i < MAX_CLIENTS; i++) {
        if (clients[i].isActive > 0) {
            mvwprintw(box_client, position, 1, clients[i].name);
            position++;
        }
    }

    wrefresh(box_client);
}

static void update_message_box() {
    wclear(box_messages);
    box(box_messages, 0, 0);
                            mvwprintw(box_messages,            16,            0,
"│                                                          │");
    for (int i = 0; i < 16; i++) {
        mvwprintw(box_messages, i + 1, 1, messages[i]);
    }

    wrefresh(box_messages);
}

void add_message(const char* msg) {
    for (int i = 1; i < 16; i++) {
        memset((char *) &messages[i - 1], ' ', sizeof(char) * 18);
        strcpy((char *) &messages[i - 1], (char *) &messages[i]);
    }
    strcpy((char *) &messages[15], msg);
    update_message_box();
}

void update_info_box(const char* name, const char* ip, int port) {
    wclear(box_info);
    box(box_info, 0, 0);
    mvwprintw(box_info, 2, 1, "  Ваш ник: ");
    mvwprintw(box_info, 2, 13, name);

    wrefresh(box_info);
}

void interface_init() {
    setlocale(LC_ALL,"");
    printf("\e[8;25;80;t");

    initscr();

    init_info_box();
    init_message_box();
    init_client_box();
    init_inpit_box();

    keypad(box_input, TRUE);
    echo();
    cbreak();      // disable line-buffering
    wtimeout(box_input, 1000 / TICK_PER_SECOND);  // wait 100 milliseconds for input
}

int read_input_box(char* buf, int* size) {
```

```
            int symbol = 0;
            while ((symbol = wgetch(box_input)) != ERR) {
                if (symbol == '\n') {
                    for (int i = 0; i < *size; i++) {
                        mvwprintw(box_input, 1, i + 1, " ");
                    }
                    return 1;
                }else if (symbol == KEY_BACKSPACE) {
                    if (*size > 0) {
                        mvwprintw(box_input, 1, (*size), " ");
                        buf[--(*size)] = 0;
                    }
                }else if (*size < 99) {
                    buf[(*size)++] = (char) symbol;
                }
            }
            mvwprintw(box_input, 1, 1, (char *)buf);
            return 0;
        }

        void interface_close() {
            delwin(box_info);
            delwin(box_client);
            delwin(box_client);
            delwin(box_messages);

            endwin();
        }

        ### Socket.h
        #ifndef C_P2P_CHAT_SOCKET_H
        #define C_P2P_CHAT_SOCKET_H

        #include "../Chat.h"

        int create_socket();

        void close_socket(int sockfd);

        void bind_address(int sockfd, struct sockaddr_in *addr, int port);

        void send_udp(int sockfd, const struct sockaddr_in *addr, const char *buf, int
buf_size);

        int socket_read(int sockfd, char *buf, struct sockaddr_in* addr, unsigned int
*addr_len);

        void sen_nonblock_flag(int descriptor);

        int is_equal_address(const struct sockaddr_in* first, const struct sockaddr_in*
second);

        void create_adress(const char* ip, int port, struct sockaddr_in* addr);

        #endif //C_P2P_CHAT_SOCKET_H


        ### Socket.c

        #include "Socket.h"
```

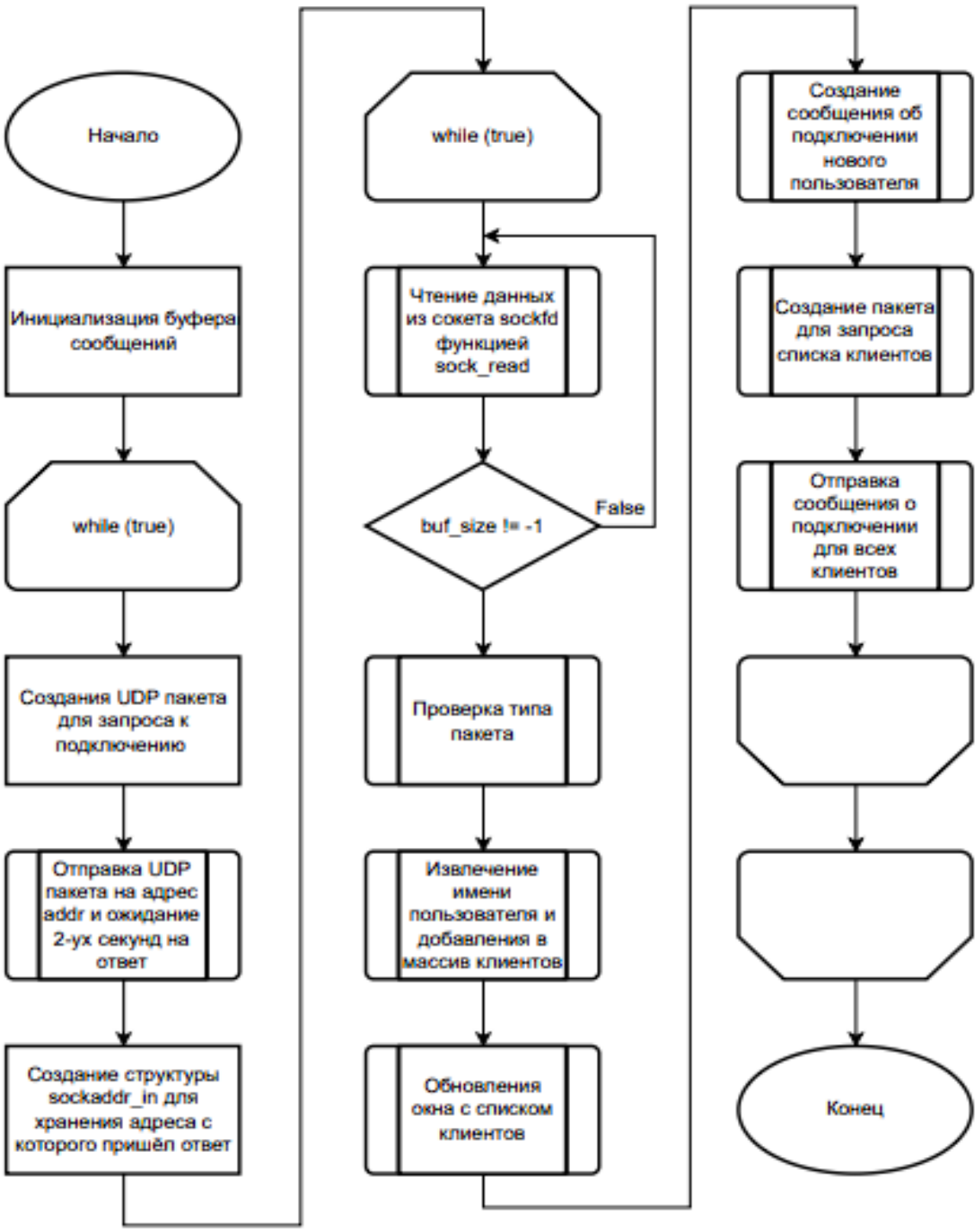
```
int create_socket() {
    int sockfd = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
    if (sockfd == -1) escape("Can't create socket");
    return sockfd;
}

void close_socket(int sockfd) {
    close(sockfd);
}

void bind_address(int sockfd, struct sockaddr_in *addr, int port) {
    memset((char *) addr, 0, sizeof(*addr));
    addr->sin_family = AF_INET;
    addr->sin_port = htons((unsigned short) port); // To network byte order
    addr->sin_addr.s_addr = htonl(INADDR_ANY);

    int result = bind(sockfd, (struct sockaddr *) addr, sizeof(*addr));
    if (result == -1) escape("Can't bind address");
}

void send_udp(int sockfd, const struct sockaddr_in *addr, const char *buf, int
buf_size) {
        sendto(sockfd, buf, (size_t) buf_size, 0, (struct sockaddr*) addr,
sizeof(*addr));
}

int socket_read(int sockfd, char *buf, struct sockaddr_in* addr, unsigned int
*addr_len) {
    int recv_len = (int) recvfrom(sockfd, buf, BUFLEN, 0, (struct sockaddr*) addr,
addr_len);
    return recv_len;
}

void sen_nonblock_flag(int descriptor) {
    int flags = fcntl(descriptor, F_GETFL);
    flags |= O_NONBLOCK;
    fcntl(descriptor, F_SETFL, flags);
}

int is_equal_address(const struct sockaddr_in* first, const struct sockaddr_in*
second) {
    return  (first->sin_addr.s_addr == second->sin_addr.s_addr) &&
            (first->sin_port == second->sin_port);
}

void create_adress(const char* ip, int port, struct sockaddr_in* addr) {
    addr->sin_family = AF_INET;
    addr->sin_addr.s_addr = inet_addr(ip);
    addr->sin_port = htons(port);
}

### Packet.h
#ifndef C_P2P_CHAT_PACKET_H
#define C_P2P_CHAT_PACKET_H

#include "../Chat.h"

#define PACKET_CONNECT_REQUES '0'
#define PACKET_CONNECT_ACCEPT '1'
```

```
#define PACKET_PING '3'
#define PACKET_TIMEOUT '4'
#define PACKET_REQUEST_USERS '5'
#define PACKET_LIST_USERS '6'
#define PACKET_SEND_MESSAGE '7'

int get_packet_id(const char* data);
int create_simple_packet(char type, char* buf);
int create_connect_request_packet(char* buf, const char* name);
int create_connect_accept_packet(char* buf, const char* name);
int create_message_packet(char* buf_send, char* buf_input, int len_msg);
int create_list_users_packet(char* buf);

#endif //C_P2P_CHAT_PACKET_H

### Packet.c
#include "Packet.h"

int get_packet_id(const char* buf) {
    return buf[0];
}

int create_simple_packet(char type, char* buf) {
    buf[0] = type;
    return 1;
}

int create_connect_request_packet(char* buf, const char* name) {
    buf[0] = PACKET_CONNECT_REQUES;
    strcpy(buf + 1, name);
    return 1 + (int) strlen(name);
}

int create_connect_accept_packet(char* buf, const char* name) {
    buf[0] = PACKET_CONNECT_ACCEPT;
    strcpy(buf + 1, name);
    return 1 + (int) strlen(name);
}

int create_message_packet(char* buf_send, char* buf_input, int len_msg) {
    buf_send[0] = PACKET_SEND_MESSAGE;
    strcpy(buf_send + 1, buf_input);
    return 1 + len_msg;
}

int create_list_users_packet(char* buf) {
    buf[0] = PACKET_LIST_USERS;
    buf[1] = 0; // Кол-во клиентов

    int pos = 2;
    // Для всех клиентов
    for (int i = 0; i < MAX_CLIENTS; i++) {
        if (clients[i].isActive > 0) {
            buf[1]++;
            memcpy(buf + pos, &(clients->address), sizeof(struct sockaddr_in));
            pos += sizeof(struct sockaddr_in);
        }
    }
    return pos;
```

ПРИЛОЖЕНИЕ Б

(обязательное)

Блок-схема алгоритма присоединения к клиенту



Изм	Лист	№ докум.	Подп.	Дата
Разраб.		Максимчик		
Пров.		Поденок		

ГУИР.400201.216 ПД.1

Блок-схема алгоритма присоединения к клиенту

Лит.	Масса	Масштаб
У		1:1
Лист		Листов 1

ЭВМ, гр. 150502

Формат А4

ПРИЛОЖЕНИЕ В

(обязательное)

Ведомость документов

Обозначение	Наименование	Примечание
	Графические документы	
ГУИР 400201.216 ПД1	Блок-схема алгоритма присоединения	А4
	пользователя	
	Текстовые документы	
БГУИР КР 1-400201.216 ПЗ	Peer-to-Peer сетевой чат с текстовым	28 с.
	интерфейсом ncurses	
	Пояснительная записка	
	Листинг кода	20 с.

Изм.	Лист	№ докум.	Подп.	Дата	ГУИР 400201.216 Д1		
Разраб.		Максимчик			P2P сетевой чат с текствоым интерфейсом ncurses Ведомость документов	Лит.	Лист	Листов
Пров.		Поденок						
						ЭВМ, гр. 150502		