

Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: «Операционные среды и системное программирование»

*К защите допустить*

И.о. заведующего кафедрой  
информатики

\_\_\_\_\_ С. И. Сиротко

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему

**УТИЛИТА ДЛЯ АВТОМАТИЧЕСКОЙ НАСТРОЙКИ СЕТЕВЫХ  
ПАРАМЕТРОВ (IP-АДРЕСА, МАСКА ПОДСЕТИ, ШЛЮЗЫ) НА  
ОСНОВЕ АНАЛИЗА СЕТЕВОЙ ИНФРАСТРУКТУРЫ**

БГУИР КП 1-40 04 01 01 001

Студент

Руководитель

Нормконтроль

К.А. Тимофеев

Н.Ю. Гриценко

Н.Ю. Гриценко

Минск 2024

## СОДЕРЖАНИЕ

Введение.....	5
1 Настраиваемые параметры сетевой инфраструктуры.....	6
2 Платформа программного обеспечения.....	14
3 Теоретическое обоснование разработки программного продукта.....	144
4 Проектирование функциональных возможностей программы .....	14
5 Выполнение программы в разных сетевых средах.....	214
Заключение .....	14
Список использованных источников .....	25
Приложение А (обязательное) Листинг исходного кода .....	27

## ВВЕДЕНИЕ

В настоящее время сетевые технологии играют ключевую роль в работе организаций и предприятий, обеспечивая связность и эффективность бизнес-процессов. Современные сетевые технологии позволяют быстро и без потерь передавать колоссальные объемы данных, обеспечивая развитие современного интернета.

Не меньшее значение сетевые технологии имеют в сфере бизнеса. Позволяя быстро передавать многочисленные пакеты данных, они позволяют значительно повысить скорость управления на больших предприятиях.

Однако для обеспечения стабильной работы сетей необходимо правильно настраивать сетевые параметры, такие как IP-адреса, маска подсети и шлюзы. Неэффективное использование доступных IP-адресов может привести к конфликтам, перегрузке сети и снижению производительности.

Для оптимизации использования ресурсов и предотвращения конфликтов необходимо регулярно анализировать сетевую инфраструктуру и настраивать сетевые параметры на основе полученных данных. Такой подход позволит оптимально использовать доступные IP-адреса.

Но при регулярном изменении количества устройств в сети регулярный процесс анализа сетевой инфраструктуры будет занимать неоправданно большое количество времени. Использование автоматической утилиты для настройки сетевых параметров не только повысит эффективность работы сети, но и значительно сократит время, затрачиваемое IT-специалистами на рутинные задачи по настройке сетевых параметров. В итоге это позволит сосредоточиться на более важных задачах и повысить общую производительность и надежность сети.

Дополнительным преимуществом использования автоматической утилиты для настройки сетевых параметров является уменьшение вероятности человеческих ошибок при конфигурации сети. Автоматизация процесса настройки позволяет исключить случайные ошибки, которые могут возникнуть при ручной настройке сетевых параметров.

В целом, автоматизация настройки сетевых параметров при помощи специализированных утилит позволяет значительно повысить эффективность работы сети, обеспечить стабильность и надежность ее функционирования, а также уменьшить трудозатраты на администрирование сети.

# 1 НАСТРАИВАЕМЫЕ ПАРАМЕТРЫ СЕТЕВОЙ ИНФРАСТРУКТУРЫ

Сетевая инфраструктура представляет собой комплекс элементов, необходимых для обеспечения надежной и эффективной работы сети. Важными компонентами сетевой инфраструктуры являются IP-адреса, подсети, шлюзы, DNS-сервера, DHCP-серверы, VLANы, маршруты, системы безопасности (включая файерволы и VPN), прокси-серверы, NAT (Network Address Translation) и системы мониторинга и управления сетью. Каждый из этих элементов играет свою роль в обеспечении стабильной работы сети и обеспечении безопасности передаваемых данных. Управление и настройка всех этих компонентов сетевой инфраструктуры являются ключевыми задачами сетевых администраторов для обеспечения эффективной работы организации.

**IP-адрес и маска подсети:** IP-адрес является уникальным идентификатором устройства в сети, который используется для маршрутизации данных. Каждому устройству в сети присваивается уникальный IP-адрес, который позволяет маршрутизаторам определять путь передачи данных. IP-адреса используются для идентификации устройств в сети, обеспечивая возможность обмена данными между ними.

Стандартный IP-адрес использует 32-битную адресацию, которая обычно записывается как четыре числа от 0 до 255, разделенные точками, например, 192.168.1.1. Всего возможно около 4 миллиардов уникальных IPv4-адресов.

Маска подсети определяет, какая часть IP-адреса относится к адресу сети, а какая - к адресу устройства в этой сети. Маска подсети состоит из последовательности битов, которые определяют длину сетевой части и длину адреса устройства в сети. Для простоты записи маску принято записывать в виде десятичного числа, обозначающего количество бит в маске, которые начиная с первого бита маски имеют значение 1. [1]

Проведя операцию конъюнкции над битами маски и IP-адреса можно получить адрес сети устройства. Оставшиеся биты обозначают адрес хоста в подсети. Этот принцип изображен на рисунке 1.1.

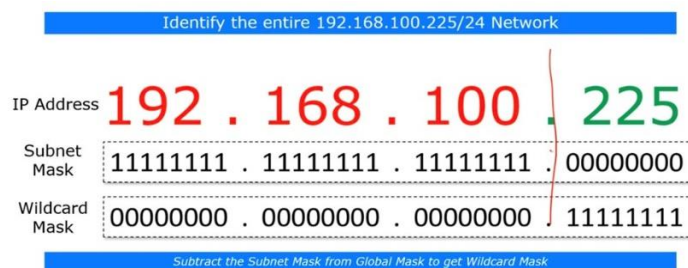


Рисунок 1.1 – Принцип работы маски подсети

На данный момент существует два стандарта IP-адресов: IPv4 и IPv6. Для расширения возможного адресного пространства в IPv6 расширили размер адреса до 128 бит, что позволяет иметь огромное количество уникальных адресов. Исходя из этого запись адреса в IPv6 представляется в виде шестнадцатеричных чисел, разделенных точкам двоеточиями. [2]

Примеры записи IPv6-адреса приведены на рисунке 1.2.



Рисунок 1.2 – Пример IPv6-адреса

Однако из-за сложностей перехода и настройки IPv6 основным стандартом все еще считается IPv4.

В IPv4 существуют четыре основных класса IP-адресов:

- класс А (от 1.0.0.0 до 126.255.255.255) обычно используется для крупных организаций;
- класс В (от 128.0.0.0 до 191.255.255.255) используется для средних организаций;
- класс С (от 192.0.0.0 до 223.255.255.255) используется для малых организаций и домашних сетей;
- класс D (от 224.0.0.0 до 239.255.255.255) зарезервирован для многоадресных групп. [3]

**Сетевые шлюзы:** сетевой шлюз (или просто шлюз) - это устройство или программное обеспечение, которое обеспечивает связь между различными

сетями, позволяя устройствам в одной сети общаться с устройствами в другой сети. Шлюзы имеют множество функций:

- маршрутизация: шлюз принимает пакеты данных от устройств в одной сети и передает их в другую сеть, выбирая оптимальный путь для доставки;
- перевод адресов (NAT): шлюз может изменять IP-адрес отправителя или получателя в пакетах данных, обеспечивая прозрачное взаимодействие между локальной сетью и интернетом;
- фильтрация трафика: шлюз может блокировать или разрешать определенные типы трафика на основе правил безопасности;
- проксирование: шлюз может действовать как посредник между устройствами и серверами, управляя запросами и ответами для повышения производительности сети. [4]

Сетевыми шлюзами могут быть маршрутизаторы или специальные прокси-серверы. Так же существует особый тип шлюзов под названием Firewall. Это специальные сервера, обеспечивающие защиту сети от несанкционированного доступа. [5]

Шлюзы играют важную роль в обеспечении связности и безопасности сетей, поэтому их правильная настройка и управление являются ключевыми задачами сетевых администраторов.

**DHCP:** DHCP (Dynamic Host Configuration Protocol) - это протокол, который позволяет устройствам в компьютерной сети автоматически получать IP-адрес, шлюз по умолчанию, DNS-серверы и другие сетевые настройки от DHCP сервера. [6]

DHCP-сервер предоставляет автоматическую настройку сетевых параметров для устройств в сети, что облегчает администрирование сети и уменьшает вероятность ошибок при конфигурации устройств.

Когда устройство подключается к сети, оно отправляет запрос на получение IP-адреса DHCP серверу. DHCP сервер выделяет свободный IP-адрес из своего пула адресов и предоставляет его устройству на определенное время. Общий формат работы DHCP сервера по выдаче адресов продемонстрирован на рисунке 1.3.

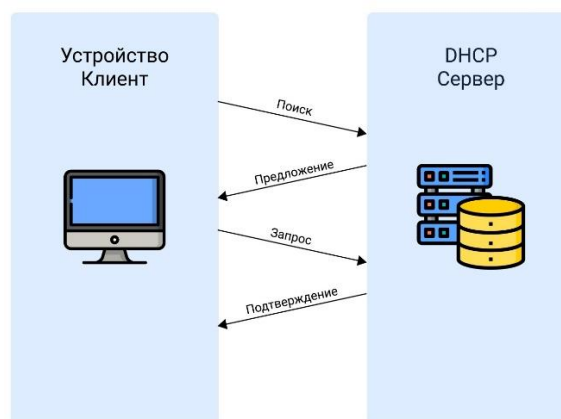


Рисунок 1.3. – Схема выдачи IP-адресов DHCP-сервером

DHCP-сервер имеет определенный диапазон IP-адресов, из которого он выделяет адреса клиентам. Этот диапазон должен быть настроен таким образом, чтобы не пересекаться с уже используемыми статическими адресами в сети. Администратор сети может зарезервировать определенные IP-адреса для конкретных устройств (например, для серверов или принтеров), чтобы избежать конфликтов адресов.

DHCP-сервер назначает каждому устройству IP-адрес на определенное время, называемое «временем аренды». По истечении этого времени устройство должно обновить свой IP-адрес у DHCP-сервера.

Помимо IP-адреса, DHCP-сервер также может предоставлять другие параметры сети, такие как шлюз по умолчанию, DNS-серверы, маску подсети т.д. [7]

## 2 ПЛАТФОРМА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Платформа программного обеспечения — это среда, в которой работают приложения. Она включает в себя операционную систему, которая управляет аппаратными ресурсами компьютера, а также программные инструменты и библиотеки, которые используются для разработки приложений. Для разработки данного проекта была выбрана ОС Ubuntu, являющаяся членом семейства операционных систем Linux. В качестве языка для написания программы был выбран язык общего назначения C++.

**ОС Ubuntu и семейство Linux:** Linux — это семейство Unix-подобных операционных систем, основанных на ядре Linux. Они включают в себя различные утилиты и программы проекта GNU. Ядро Linux является монолитным и состоит из многих компонентов:

- Linux framebuffer USB core — подсистема для поддержки USB-устройств и контроллеров шины USB;
- Evdev (англ. event device) — компонент для работы с устройствами ввода (клавиатурой, джойстиком, мышью);
- Kvm (англ. kernel-based virtual machine);
- DRM (англ. direct rendering manager);
- Cgroups (англ. control groups);
- Dm (англ. device mapper);
- SELinux (англ. security-enhanced Linux) и многие другие. [8]

Ядро Linux обладает мощными возможностями для взаимодействия с сетью и предоставляет программные интерфейсы для реализации различных сетевых функций. Ядро Linux поддерживает широкий спектр сетевых протоколов, таких как TCP/IP, UDP, ICMP, HTTP, FTP и другие, что позволяет разработчикам создавать сетевые приложения, работающие поверх этих протоколов. Также ядро предоставляет возможности управления сетевыми устройствами такими как сетевые карты, мосты, VLAN и т. д. Это позволяет настраивать и контролировать сетевые соединения на уровне ядра.

В ядре Linux есть API для работы с сокетами, что позволяет приложениям устанавливать и управлять сетевыми соединениями. Кроме того, ядро предоставляет интерфейсы для работы с сетевыми устройствами и настройки сетевых параметров.

В ядре Linux встроены механизмы безопасности для защиты сетевых соединений и данных. Это включает в себя механизмы аутентификации, шифрования данных, контроль доступа и другие функции безопасности.





для широкого спектра задач, от программирования микроконтроллеров до разработки сложных систем. На рисунке 2.2 изображен список встроенных в язык C++ библиотек. [12]

Header File	Library	man Pages
<b>C++ STL</b>		
string	STL strings type	std::string
sstream	stringstream, for writing to strings as if they are streams	std::stringstream
iostream	C++ standard stream library	std::ios, std::iostream
memory	C++ memory-related routines	std::bad_alloc, std::auto_ptr
<b>C Standard Library</b>		
cstring, string.h	Functions for C char* strings	string, strcpy, strcmp
cstdlib, stdlib.h	C Standard Library	random, srand, getenv, setenv
cstdio, stdio.h	Standard input/output	stdin, stdout, printf, scanf
cassert	Assert macros	assert

Рисунок 2.2 – Список стандартных библиотек

Синтаксис C++ унаследован от языка C, что обеспечивает совместимость с ним. Однако, стоит отметить, что C++ не является в строгом смысле надмножеством C. Одной из ключевых особенностей C++ является поддержка объектно-ориентированного программирования, что позволяет создавать модульные и повторно используемые программы.

Наконец, C++ имеет активное сообщество разработчиков и множество ресурсов для изучения, что делает его одним из наиболее мощных и гибких языков программирования.

Для взаимодействия с сетями в каждой операционной системе предусмотрен свой набор программных интерфейсов. Для дистрибутивов Linux предусмотрен заголовочный файл <sys/socket.h>. Он позволяет устанавливать API сокетов для работы с сетью.

В операционной системе Linux существуют различные программные интерфейсы для настройки IP-адресов, шлюзов и масок подсети. Один из наиболее распространенных методов - это использование системного вызова `ioctl`. Этот метод позволяет получить информацию о сетевом интерфейсе, например, его IP-адрес. Однако, для более сложных задач, таких как настройка IP-адресов, шлюзов и масок подсети, можно использовать инструмент командной строки `nmcli`.

`nmcli` - это мощный инструмент, который позволяет управлять сетевыми настройками прямо из командной строки. Можно использовать `nmcli` для настройки статического IP-адреса, шлюза и DNS-сервера. Это делается с

помощью одной простой команды, которую можно вызвать из программы на C++, используя функцию `system`.

**Visual Studio Code:** в качестве среды разработки был выбран текстовый редактор Visual Studio Code от компании Microsoft. Редактор доступен для Windows, Linux и macOS. Он предлагает множество функций, таких как подсветка синтаксиса, автодополнение кода с помощью IntelliSense, интеграция с Git, инструменты профилирования и маркетплейс с более чем 9000 расширений. Visual Studio Code поддерживает различные языки программирования, фреймворки и инструменты, что делает его идеальным для кроссплатформенной разработки. Установка VS Code проста и возможна на всех популярных операционных системах. Несмотря на свою легкость и быстрдействие, VS Code обладает всем необходимым функционалом и подходит для работы даже на не очень мощных компьютерах. Простота освоения делает его доступным для начинающих разработчиков, в то время как его мощные функции и гибкость делают его полезным инструментом для опытных программистов. [13]

Интерфейс программы приведен на рисунке 2.3.

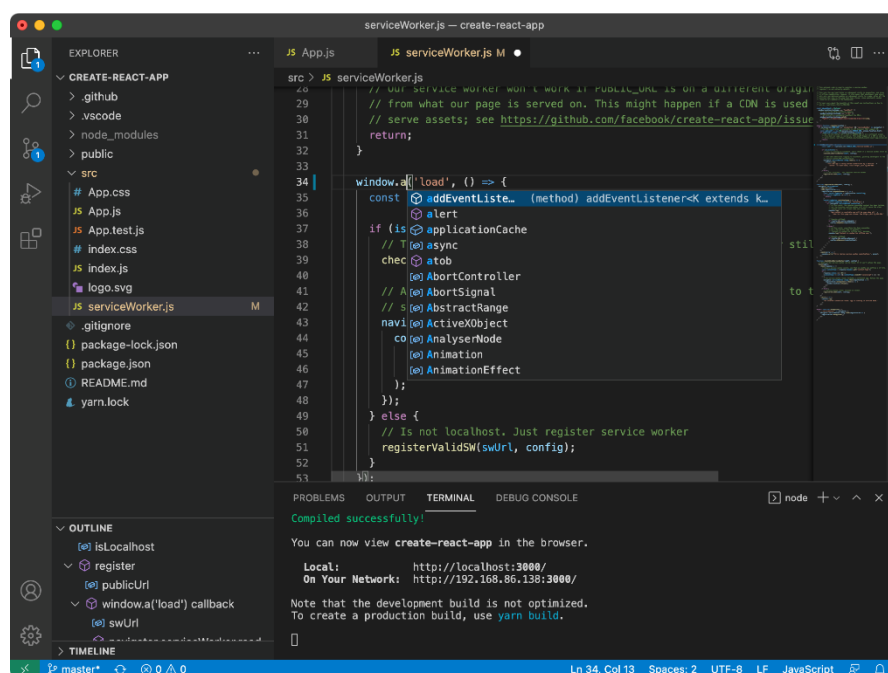


Рисунок 2.3 – Интерфейс редактора Visual Studio Code

Выбранная платформа программного обеспечения использует популярные и проверенные временем решения.

### 3 ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ ПРОГРАММНОГО ПРОДУКТА

**Существующие виды назначения IP-адресов:** Существует несколько различных видов назначения IP-адресов в компьютерных сетях. Они могут быть разделены на две основные категории: статическое и динамическое назначение IP-адресов.

Первый вариант это статическое назначение IP-адресов. Оно может быть двух видов:

- вручную: администратор сети назначает IP-адрес каждому устройству вручную. Этот метод требует постоянного вмешательства администратора и может быть неэффективным в больших сетях;

- автоматически: администратор настраивает сервер DHCP (Протокол динамической конфигурации хоста), который автоматически назначает IP-адрес каждому устройству в сети. Этот метод более эффективен, чем ручное назначение IP-адресов, особенно в больших сетях.

Второй вид это динамическое назначение IP-адресов. Их так же существуют два вида:

- DHCP (Протокол динамической конфигурации хоста): DHCP-сервер автоматически назначает IP-адрес устройствам в сети при их подключении. Каждое устройство получает временный IP-адрес из пула доступных адресов. Этот метод упрощает процесс управления IP-адресами и позволяет эффективно использовать доступные адреса.

- APIPA (Автоматическое назначение частных IP-адресов): если DHCP сервер недоступен, устройство может назначить себе временный IP-адрес из диапазона 169.254.0.1 - 169.254.255.254. Это позволяет устройствам в локальной сети общаться друг с другом, даже если нет доступного DHCP сервера. [14]

**Выбор маски подсети:** выбор маски подсети для IP-адреса зависит от нескольких факторов.

Первым фактором является тип сети. Для локальных сетей выбирается маска /24, а для крупных сетей или для сетей провайдеров выбираются маски /16 или /8.

Вторым фактором является количество устройств в настраиваемой сети. Количество битов, которые маска оставляет для обозначения номера устройства в сети, должно вмещать в себя количество устройств в сети или превышать.

Так же стоит учитывать наличие подсетей. [15]

**Особые IP-адреса:** перед присвоением адреса стоит проверить, не является ли данный адрес одним из зарезервированных. Если первый октет сети начинается со 127, такой адрес считается адресом машины – источника пакета. В этом случае пакет не выходит в сеть, а возвращается на компьютер-отправитель. Такие адреса называются loopback (петля, замыкание на себя) и используются для проверки функционирования стека TCP/IP.

Если все биты IP-адреса равны нулю, адрес обозначает узел отправитель и используется в некоторых сообщениях ICMP.

Если все биты IP-адреса сети равны 0, а все биты IP-адреса хоста равны 1, то адрес называется ограниченным широковещательным (Limited Broadcast). Пакеты, направленные по такому адресу, рассылаются всем узлам той подсети, в которой находится отправитель пакета. К данному же типу адреса можно относить и IP-адрес, в котором 32 разряда заполнены 1, так как пакет с таким адресом не будет «выпущен» за пределы сети отправителем устройством, связующим сети, например коммутатором или маршрутизатором.

Если все биты IP-адреса хоста равны 1, а биты IP-адреса сети не равны 0, т. е. однозначно идентифицируют адрес подсети, то адрес называется широковещательным (Broadcast); пакеты, имеющие широковещательный адрес, доставляются всем узлам подсети назначения. [16]

**Частные и публичные IP-адреса:** Поскольку каждый узел сети Интернет должен обладать уникальным IP-адресом, то, безусловно, важной является задача координации распределения адресов отдельным сетям и узлам. Таковую координирующую роль выполняет интернет-корпорация по распределению имен и адресов (the Internet Corporation for Assigned Names and Numbers, ICANN).

Служба распределения номеров IANA (Internet Assigned Numbers Authority) зарезервировала для частных сетей следующие три блока адресов:

- 10.0.0.0–10.255.255.255 (1 сеть класса A);
- 172.16.0.0–172.31.255.255 (16 сетей класса B);
- 192.168.0.0–192.168.255.255 (256 сетей класса C).

Любая организация может использовать IP-адреса из этих блоков без согласования с ICANA или Internet-регистраторами. В результате эти адреса используются во множестве организаций. Таким образом, уникальность адресов сохраняется только в масштабе одной или нескольких организаций, которые согласованно используют общий блок адресов. В такой сети каждая рабочая станция может обмениваться информацией с любой другой рабочей станцией частной сети. [17]

Перед распределением адресов из частного и публичного блоков следует определить, какие из рабочих станций сети должны иметь связь с внешними системами на сетевом уровне. Для таких рабочих станций следует использовать публичные адреса, остальным же – можно присваивать адреса из частных блоков, это не мешает им взаимодействовать со всеми рабочими станциями частной сети организации, независимо от того, какие адреса используются (частные или публичные). Однако прямой доступ во внешние сети для рабочих станций с адресами из частного блока невозможен. Для организации их доступа во внешние шлюзы придется использовать прокси-серверы. Перемещение рабочей станции из частной сети в публичную (и обратно) связано со сменой IP-адреса, соответствующих записей DNS и изменением конфигурационных файлов на других рабочих станциях, которые их идентифицируют по IP-адресам. Поскольку частные адреса не имеют глобального значения, маршрутная информация о частных сетях не должна выходить за пределы этих сетей, а пакеты с частными адресами отправителей или получателей не должны передаваться через межсетевые каналы. Предполагается, что маршрутизаторы в публичных сетях (особенно маршрутизаторы провайдеров Internet) будут отбрасывать маршрутную информацию из частных сетей. Если маршрутизатор публичной сети получает такую информацию, ее отбрасывание не должно трактоваться как ошибка протокола маршрутизации.

**Шлюзы:** Шлюз является наиболее сложной ретрансляционной системой, обеспечивающей взаимодействие сетей с различными наборами протоколов всех семи уровней. В свою очередь, наборы протоколов могут опираться на различные типы физических средств соединения. На рисунке 3.1 изображена структура работы шлюза.



Рисунок 3.1 – Структура работы шлюза

При соединении информационных сетей часть уровней может иметь одни и те же протоколы. В этом случае сети соединяются не при помощи шлюза, а на основе более простых ретрансляционных систем, именуемых маршрутизаторами и мостами.

В качестве шлюза обычно используется выделенный компьютер, на котором запущено программное обеспечение шлюза и производятся преобразования, позволяющие взаимодействовать нескольким системам в сети. Другой функцией шлюзов является преобразование протоколов. При получении сообщения IPX/SPX для клиента TCP/IP шлюз преобразует сообщения в протокол TCP/IP. [18]

Таким образом, для правильной настройки сетевых параметров необходимо знать, является сеть локальной или глобальной. Также нужно знать количество устройств в сети, и их назначение в общем устройстве сети.

## 4 ПРОЕКТИРОВАНИЕ ФУНКЦИОНАЛЬНЫХ ВОЗМОЖНОСТЕЙ ПРОГРАММЫ

**Получение информации о сетевой инфраструктуре:** функция `get_network_info` открывает процесс для выполнения команды `ip route show`, которая выводит информацию о маршрутах IP. Результат команды сохраняется в строку и возвращается из функции. Это позволяет пользователю получить обзор текущей сетевой конфигурации устройства.

**Получение сетевых параметров:** для получения сетевых параметров используется функция `system` ядра Linux. Она позволяет выполнять из код программ различные вызовы встроенных в ОС утилит, которые в обычном случае вызываются пользователем персонального компьютера через терминал. Это значительно расширяет возможности написания прикладных программ, позволяя вызывать в них скрипты `bash` или другие программы и использовать их результаты.

Для удобства код вызова системных через функцию `system` обернут в функцию `execute_command`. В коде функции также присутствует код получения выведенных командой данных из специального файла.

Получение текущих сетевых параметров осуществлено при помощи вызовов функции `execute_command`.

Для получения адресов в текущей сети в семействе ОС Linux существует множество различных утилит, однако они не всегда надежны. Поэтому в разрабатываемой программе текущие IP-адреса анализируются при помощи команды `ping`. Она проверяет доступ к указанному в параметре IP-адресу. Так как команда позволяет указать время, в течение которого нужно ожидать ответ, то ее данные в среднем случае более достоверны чем данные других утилит.

Для получения текущего адреса и маски сети используется команда `ip`. Она предназначена для работы с IP-адресами.

**Автоматическое определение параметров:** параметры сетевой конфигурации вычисляются исходя из текущего количества хостов в сети и их адресов. Зная текущее количество хостов, можно вычислить оптимальную маску.

Первый по порядку хост в сети, запустивший данную утилиту, становится вторым хостом в сети и запускает на ограниченное время прослушку TCP-соединений по заранее определенному порту. В этот момент данный хост становится своего рода DHCP-сервером, призванным синхронизировать выдачу новых адресов остальным хостам.



Однако при массовом одновременном запуске программы может возникнуть коллизия, когда несколько хостов пытаются занять второй адрес, или когда второй адрес еще не занят, и, следовательно, сервера с данными по адресам еще нет. Для этого программа при отсутствии хоста со вторым адресом становится в режим получения UDP-пакетов. Если в течение заранее определенного количества времени не было получено пакетов, то программа сама начинает рассылку UDP-пакетов, обозначая то, что компьютер, с которого она работает, намеревается занять второй адрес. Таким образом хосты, запустившие программу позже, не будут занимать второй адрес и роль сервера по выдаче адресов ляжет на хост, на котором программа была запущена раньше всего. При возникновении ситуации, когда два хоста почти одновременно перешли в режим рассылки, программа при рассылке UDP-пакетов не прекращает получать пакеты. При наличии подобной коллизии второй адрес займет хост с наименьшим текущим адресом.

Такой алгоритм позволяет динамически изменять параметры исходя из количества подключенных к сети устройств и типа их IP-адресов. При добавлении нового устройства в сеть достаточно запустить утилиту на всех нужных устройствах, и новые IP-адреса будут выданы.

**Обработка ошибок:** в коде предусмотрены проверка на успешность открытия процесса для выполнения команды. Ошибки могут возникнуть на стороне системы, поэтому результат каждого вызова системного API проверяется на ожидаемое поведение. При ошибках, программа прекратит выполнение и выведет сообщение об ошибке. Системное API ядра Linux при ошибках возвращает отрицательные значения, а описание ошибки хранит в системной переменной. Этот общий интерфейс выявления ошибок в вызовах позволяет однотипно обрабатывать различные ошибки.

Код программы представлен в приложении А.

**Возможные улучшения:** утилиту можно доработать. Возможны как расширение текущих возможностей, так и повышение удобства использования программы.

Вместо IP-адреса действующего шлюза можно передавать в параметры адрес сервера, отправив TCP запрос на который можно узнать адрес шлюза устройства с данным MAC-адресом. Также на сервере можно хранить файл настройки в установленном формате, который позволял бы однозначно установить шлюзы всех устройств, использующих эту утилиту. Таким образом, системный администратор может назначить шлюзы, потом запустив программу и получив файл конфигурации автоматически распределить все устройства по шлюзам и выдать исходя из их расположения в сети IP-адреса.

В таком случае, устройство, хранящее файл конфигурации, должно иметь постоянный в рамках данной сети IP-адрес, который можно будет указать при вызове программы.

Также вместо ручного вызова программы ее можно сделать фоновым процессом, который при определенных условиях будет автоматически проверять параметры сетевой конфигурации раз в определенное количество времени. Таким образом при добавлении новых устройств в сеть не будет необходимости в ручном запуске программы.

Совместив два вышеописанных подхода, можно получить приложение, работающее в фоновом режиме с файлом конфигурации сети. При изменении файла программа будет динамически переназначать свои сетевые параметры, отчего задачей сетевого администратора станет только составление файла конфигурации сети на конфигурационном сервере.

Таким образом, основная инфраструктура программы готова, и открыта к дальнейшим расширениям. Текущий алгоритм работы программы не мешает добавлению новых настраиваемых параметров, участвующих в настройке сетевых параметров.

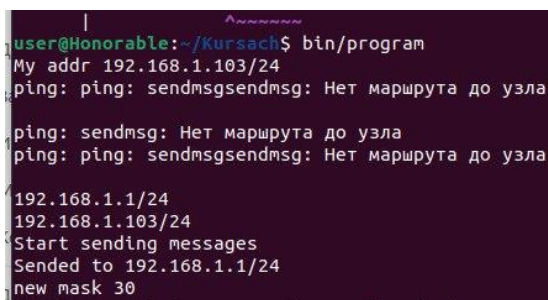
## 5 ВЫПОЛНЕНИЕ ПРОГРАММЫ В РАЗЛИЧНЫХ СЕТЕВЫХ СРЕДАХ

**Вычисление маски:** маска подсети является одним из ключевых аспектов при настройке сетей, поскольку она определяет, какие узлы в сети будут видны друг другу и какие будут скрыты. Важность выбора правильной маски подсети заключается в том, что она позволяет эффективно использовать доступные IP-адреса в сети, обеспечивает безопасность и управляемость сети, а также повышает производительность.

Алгоритм выбора маски подсети зависит от конкретных требований и характеристик сети. Одним из наиболее распространенных алгоритмов является метод "разделения на подсети" (subnetting), который позволяет разделить сеть на более мелкие подсети для улучшения управляемости и безопасности. Для выбора подходящей маски подсети необходимо учитывать количество узлов в сети, требования к безопасности, географическое расположение узлов и другие факторы.

Кроме того, при выборе маски подсети важно учитывать возможность будущего масштабирования сети, чтобы избежать необходимости перестраивать всю инфраструктуру в случае расширения сети. Поэтому рекомендуется выбирать маску подсети с запасом IP-адресов для будущего роста.

Так как разрабатываемая утилита позволяет массово настраивать сетевые параметры, не имеет смысла делать маску больше необходимой, ведь в любой момент можно заново перенастроить сетевые параметры, еще раз запустив программу. При вычислении минимально возможной маски в программе учитывается 2 зарезервированных адреса: адреса сети и широковещательного адреса. Получение маски для различных количеств активных хостов в сети продемонстрировано на рисунках 4.1 и 4.2.



```
user@Honorable:~/kursach$ bin/program
My addr 192.168.1.103/24
ping: ping: sendmsgsendmsg: Нет маршрута до узла

ping: sendmsg: Нет маршрута до узла
ping: ping: sendmsgsendmsg: Нет маршрута до узла

192.168.1.1/24
192.168.1.103/24
Start sending messages
Sended to 192.168.1.1/24
new mask 30
```

Рисунок 4.1 – Маска при двух активных хостах

```

My addr 192.168.1.103/24
ping: sendmsg: Нет маршрута до узла
192.168.1.1/24
192.168.1.103/24
192.168.1.106/24
Start sending messages
Sended to 192.168.1.1/24
Sended to 192.168.1.106/24
new mask 29

```

Рисунок 4.2 – Маска при трех активных хостах

Как видно на рисунках выше, маска подсети выбирается минимально возможной для вмещения активных хостов.

**Вычисление IP-адреса:** правильный выбор IP-адресов в локальной сети имеет огромное значение для обеспечения ее эффективной работы, безопасности и управляемости. IP-адреса являются уникальными идентификаторами устройств в сети, поэтому неправильный выбор адресов может привести к конфликтам, перегрузке сети и другим проблемам. Поэтому необходимо уделить должное внимание процессу выбора IP-адресов и использовать соответствующие алгоритмы для их получения.

Один из наиболее распространенных алгоритмов получения IP-адресов в локальной сети – это использование протокола DHCP (Dynamic Host Configuration Protocol). DHCP позволяет автоматически назначать IP-адреса устройствам в сети, что упрощает процесс управления адресами и предотвращает конфликты. DHCP также позволяет настраивать различные параметры сети, такие как шлюз по умолчанию, DNS-серверы и другие.

Еще одним алгоритмом получения IP-адресов является статическое назначение адресов. При этом администратор сети вручную назначает каждому устройству уникальный IP-адрес. Этот метод подходит для небольших сетей или в случаях, когда требуется точный контроль над IP-адресами.

Программа предоставляет возможность автоматически настроить IP-адрес, так что по своему функционалу программа походит на DHCP-сервер, за исключением того, что адреса назначаются статически и меняются программой.

На рисунках 4.3 и 4.4 показано преобразование различных адресов для различных конфигураций сети.

```

user@Honorable:~/Kursach$ bin/program
My addr 192.168.1.103/24
ping: ping: sendmsgsendmsgping: ping: ping: sendr
: Нет маршрута до узла: Нет маршрута до узла: Нет
: Нет маршрута до узла

192.168.1.1/24
192.168.1.103/24
192.168.1.106/24
Start sending messages
Sended to 192.168.1.1/24
Sended to 192.168.1.106/24
MyNew addr 192.168.1.2/29

```

Рисунок 4.3 – Новый адрес для 103-его хоста

```

My addr 192.168.1.106/24
192.168.1.1/24
192.168.1.103/24
192.168.1.106/24
Start sending messages
Sended to 192.168.1.1/24
Sended to 192.168.1.106/24
MyNew addr 192.168.1.3/29

```

Рисунок 4.4 – Новый адрес для 106-го хоста

Как видно на рисунках, программа выдает номера по возрастанию от 2 до 254 хостам, которые были в сети в момент первого запуска программы.

Таким образом, разработанная программа вычисляет и назначает сетевые параметры согласно заданному алгоритму.

## ВЫВОДЫ

В ходе выполнения курсового проекта по созданию утилиты для настройки сетевых параметров была разработана программа, настраивающая сетевые параметры исходя из текущей конфигурации сети. При разработке данной утилиты были учтены основные принципы сетевого взаимодействия, что позволило создать удобный инструмент сетевого администрирования.

Использование ОС семейства Linux позволило использовать эффективные встроенные утилиты для получения и настройки сетевых параметров. Выбранный для разработки данной утилиты язык программирования C++ позволил использовать хорошо задокументированное и отработанное годами API ядра Linux благодаря совместимостью с языком C. Используемый текстовый редактор Visual Studio Code позволил эффективно разработать программу, помогая при написании кода подсветкой ошибок и подсказками с автозаполнением кода.

Созданная в данном проекте отвечает разработанным требованиям и корректно настраивает сетевые параметры, анализируя текущее состояние сети.

Как видно из полученных результатов, программе для эффективной работы необходимо чтобы все устройства, на которых она будет запускаться, были активны при первом ее запуске. Для публичных сетей, рассчитанных на постоянный приток и отток хостов, такое условие практически не выполнимо. Для работы программы приходилось бы на каждом новом устройстве запускать программу, а при таком сценарии, возникающем при большом потоке устройств, утилита не предоставляет никаких преимуществ и становится только обузой в работе по настройке сети.

Лучше всего программа подойдет для запуска в корпоративных сетях персональных компьютеров. В таком случае новые устройства будут появляться относительно редко, а старые будут редко удаляться из сети. Так же в таких сетях в момент запуска программы все хосты в сети могут быть активны, что позволит корректнее настроить адреса и маску.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Что такое IP-адрес и маска подсети и зачем они нужны [Электронный ресурс]. – Режим доступа: <https://skillbox.ru/media/code/что-такое-ipadres-i-mask-a-podseti-i-zachem-oni-nuzhny/>. – Дата доступа 02.03.2024.

[2] Что такое и зачем нужен IPV6? Разбор [Электронный ресурс]. – Режим доступа: <https://habr.com/ru/companies/droider/articles/56877>. – Дата доступа 02.03.2024.

[3] Всё об IP адресах и о том, как с ними работать [Электронный ресурс]. – Режим доступа: <https://habr.com/ru/articles/350878/>. – Дата доступа 03.03.2024.

[4] Сетевой шлюз: что это и как работает? [Электронный ресурс]. – Режим доступа: <https://vasexperts.ru/resources/glossary/setevoj-shlyuz/>. – Дата доступа 03.03.2024.

[5] Глава 10 [Электронный ресурс]. – Режим доступа: <https://seticom.narod.ru/net/sportaknet/Chapter10.html/>. – Дата доступа 03.03.2024.

[6] DHCP-протокол: что это такое и как он работает [Электронный ресурс]. – Режим доступа: <https://selectel.ru/blog/dhcp-protocol/>. – Дата доступа 03.03.2024.

[7] Что такое DHCP сервер и как его настроить [Электронный ресурс]. – Режим доступа: <https://servergate.ru/articles/что-такое-dhcp-server-i-kak-ego-nastroit/>. – Дата доступа 03.03.2024.

[8] Что же такое Linux, кто пользуется этой операционной системой, основные дистрибутивы Линукс | Часто задаваемые вопросы [Электронный ресурс]. – Режим доступа: <https://ipkey.com.ua/faq/924-linux.html>. – Дата доступа 05.03.2024.

[9] Сетевая подсистема в ОС [Электронный ресурс]. – Режим доступа: <https://habr.com/ru/companies/otus/articles/540582/>. – Дата доступа 05.03.2024.

[10] Что такое Ubuntu и зачем её устанавливать [Электронный ресурс]. – Режим доступа: <https://thecode.media/ubuntu-linux/>. – Дата доступа 05.03.2024.

[11] Язык программирования C++ - История создания, плюсы и минусы [Электронный ресурс]. – Режим доступа: <https://verity.by/news/yazyk-programmirovaniya-c-chast-1-istoriya-sozdaniya-plyusy-i-minusy/>. – Дата доступа 10.03.2024.

[12] Прикладное программирование [Электронный ресурс]. – Режим доступа: [http://aco.ifmo.ru/el\\_books/applied\\_programming/lectures/part2-1.html](http://aco.ifmo.ru/el_books/applied_programming/lectures/part2-1.html) – Дата доступа 01.04.2024.

[13] Что такое Visual Studio Code и как им пользоваться? [Электронный ресурс]. – Режим доступа: <https://skyeng.ru/magazine/chto-takoe-visual-studio-code/>. – Дата доступа 01.04.2024.

[14] APIPA vs DHCP IP [Электронный ресурс]. – Режим доступа: <https://www.linkedin.com/pulse/apipa-vs-dhcp-ip-prabhudas-borkar-smp2f/>. – Дата доступа 03.04.2024.

[15] Пример расчета количества хостов и подсетей на основе IP-адреса и маски [Электронный ресурс]. – Режим доступа: <https://help.keenetic.com/hc/ru/articles/213965829-Пример-расчета-количества-хостов-и-подсетей-на-основе-IP-адреса-и-маски>. – Дата доступа 03.04.2024.

[16] Особые ip-адреса [Электронный ресурс]. – Режим доступа: <https://studfile.net/preview/1579425/page:16/>. – Дата доступа 07.04.2024.

[17] Адрес ip сети, классы сетей, диапазоны ip адресов локальной сети - Помощь, офис класса А [Электронный ресурс]. – Режим доступа: <http://www.novelway.ru/inform-tehnologii/ip-seti/>. – Дата доступа 07.04.2024.

[18] Трудности перевода: уязвимости шлюзов промышленных протоколов [Электронный ресурс]. – Режим доступа: <https://habr.com/ru/companies/trendmicro/articles/519440/>. – Дата доступа 07.04.2024.



# ПРИЛОЖЕНИЕ А

## (обязательное)

### Листинг исходного кода

#### Листинг 1 – Программный код разработанного приложения

```
#include <iostream>
#include <cstdlib>
#include <pthread.h>
#include <semaphore.h>
#include <vector>
#include <cmath>

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <string.h>
#include <stdio.h>
#include <errno.h>
#include <unistd.h>
#include <stdlib.h>
#include <chrono>

#define LISTEN_PORT 40000
#define SEND_PORT 40001
#define WAIT_TIME 2000
#define WAIT_TIME_DISTR 300
#define BUFF_SIZE sizeof(Message)
#define SERVER_EXIST_TIME 2

int Mask = 24;

class IpAddr
{
public:
    short mask;
    unsigned int addr;

    IpAddr(){}
    IpAddr(unsigned int addr, short mask)
    {
        this->addr = addr;
        this->mask = mask;
    }

    IpAddr NetAddr() const
    {
        int m = 32 - mask;

        return IpAddr((addr >> m) << m, mask);
    }

    IpAddr(const IpAddr& other)
    {
        addr = other.addr;
        mask = other.mask;
    }

    IpAddr(const std::string str, short mask)
    {
        std::string copy = str;
        addr = 0;
        this->mask = mask;
        for(int i = 0; i < 4; ++i)
        {
            int pos = copy.find_first_of('.');
            unsigned long oct = std::stoi(copy.substr(0, pos));
            // std::cout << "Oct " << i << " : " << oct << "\n";
            addr |= oct << (24 - i * 8);
            // std::cout << "In addr " << ((addr << i * 8) >> 24 ) << "\n";
            copy = copy.substr(pos + 1);
        }
    }
}
```

```

// from 0 to 3
unsigned int oct(int n) const
{
    return ((addr << n * 8) >> 24);
}

std::string ToString() const
{
    std::string res = "";
    for(int i = 0; i < 4; ++i)
    {
        res += std::to_string(oct(i));
        res += '.';
    }

    return res.substr(0, res.size() - 1) + "/" + std::to_string(mask);
}

std::string ToStringWithoutMask() const
{
    std::string res = "";
    for(int i = 0; i < 4; ++i)
    {
        res += std::to_string(oct(i));
        res += '.';
    }

    return res.substr(0, res.size() - 1);
}

unsigned long host_number()
{
    return (addr << mask) >> mask;
}

void set_host(unsigned long host)
{
    int m = 32 - mask;

    addr = ((addr >> m) << m) | (host << m) >> m;
}

};

std::string get_network_info() {
    FILE* pipe = popen("ip route show", "r");
    if (!pipe) return "Error";

    char buffer[128];
    std::string result = "";
    while (!feof(pipe)) {
        if (fgets(buffer, 128, pipe) != NULL)
            result += buffer;
    }

    pclose(pipe);
    return result;
}

std::string execute_command(std::string command)
{
    FILE* pipe = popen(command.c_str(), "r");
    if (!pipe) return "Error";

    char buffer[128];
    std::string result = "";
    while (!feof(pipe)) {
        if (fgets(buffer, 128, pipe) != NULL)
            result += buffer;
    }

    pclose(pipe);
    return result;
}

struct TaskArg

```

```

{
    sem_t* sem;
    std::vector<IpAddr>* v;
    short mask;
    IpAddr addr;
};

void* task(void* taskArg)
{
    TaskArg ta = *(TaskArg*)taskArg;

    std::string res = execute_command("ping -w 6 -c 3 " + ta.addr.ToStringWithoutMask() + " >
/dev/null && echo " + ta.addr.ToStringWithoutMask());

    if(res.length() != 0)
    {
        res = res.substr(0, res.length() - 1);
        IpAddr addr = IpAddr(res, ta.mask);
        sem_wait(ta.sem);
        ta.v->push_back(addr);
        sem_post(ta.sem);
    }

    delete taskArg;

    return NULL;
}

std::vector<IpAddr> pingall(IpAddr myAddr, short mask)
{
    sem_t* sem = new sem_t();
    if(sem_init(sem, 0, 1))
    {
        perror("Sem init error");
    }

    std::vector<IpAddr> v;
    pthread_t tids[254];
    for (int i = 1; i < 255; ++i)
    {
        TaskArg* ta = new TaskArg();
        ta->addr = IpAddr(myAddr);
        ta->addr.set_host(i);
        ta->sem = sem;
        ta->v = &v;
        ta->mask = mask;
        pthread_create(tids + i - 1, NULL, task, ta);
    }

    for (int i = 0; i < 254; ++i)
    {
        pthread_join(tids[i], NULL);
    }

    sem_close(sem);
    sem_destroy(sem);
    delete sem;

    return v;
}

// void configure_network(std::string ip_address, std::string netmask, std::string gateway) {
//     system(("ifconfig eth0 " + ip_address + " netmask " + netmask).c_str());
//     system(("route add default gw " + gateway).c_str());
// }

std::vector<std::string> new_addrs(const std::vector<std::string>& old)
{
    std::cout << "size " << old.size() << "\n";
    float bits = ceil(log(old.size() + 2) / log(2));
    std::cout << "bits " << bits << "\n";
    bits = bits + 0.5;
    int int_bits = bits;
    int mask = 32 - int_bits;
    std::cout << "new mask " << mask << std::endl;

    std::string base = "192.168.1.";
    std::vector<std::string> newAddrs;

```

```

    for (int i = 0; i < old.size(); ++i)
    {
        newAddrs.push_back(base + std::to_string(i + 1));
        std::cout << "old: " << old[i] << ", new " << newAddrs[i] << "\n";
    }

    return newAddrs;
}

int host_number(std::string addr, std::string base)
{
    return std::stoi(addr.substr(base.size()));
}

void sort_addrs(std::vector<IpAddr>& addrs)
{
    for(int i = 0; i < addrs.size(); ++i)
    {
        for(int j = 0; j < addrs.size() - 1 - i; ++j)
        {
            if(addrs[j].addr > addrs[j+1].addr)
            {
                IpAddr tmp = addrs[j];
                addrs[j] = addrs[j+1];
                addrs[j+1] = tmp;
            }
        }
    }
}

sockaddr_in create_sockaddr(int port, std::string ip = "")
{
    sockaddr_in addr;
    bzero(&addr, sizeof(addr));
    addr.sin_family = AF_INET;
    addr.sin_port = htons(port);
    if (ip == "")
    {
        addr.sin_addr.s_addr = htonl(INADDR_ANY);
    }
    else
    {
        addr.sin_addr.s_addr = inet_addr(ip.c_str());
    }

    return addr;
}

int create_socket(int port)
{
    auto sock_addr = create_sockaddr(port);
    int descriptor;
    if((descriptor = socket(PF_INET, SOCK_DGRAM, 0)) < 0){
        std::cout << "Error in creating socket\n";
        perror(NULL); /* Печатаем сообщение об ошибке */
        exit(1);
    }

    if(bind(descriptor, (sockaddr *) &sock_addr, sizeof(sock_addr)) < 0){
        std::cout << "Error nigga\n";
        perror(NULL);
        close(descriptor);
        exit(1);
    }

    return descriptor;
}

struct DistrArgs
{
    sem_t* sem_flag;
    IpAddr myAddr;
    std::vector<IpAddr> addrs;
    bool* flag;
    sem_t* sem_console;
};

```

```

};

struct Message
{
    IpAddr sender;
};

void* DoDistr(void* void_arg)
{
    DistrArgs* args = (DistrArgs*)void_arg;

    int sender = create_socket(SEND_PORT);

    for (int i = 0; i < args->addrs.size(); ++i)
    {
        if(args->addrs[i].addr == args->myAddr.addr) continue;
        sockaddr_in receiver = create_sockaddr(LISTEN_PORT, args->addrs[i].ToStringWithoutMask());
        Message messg;
        messg.sender = args->myAddr;
        if(sendto(sender, &messg, BUFF_SIZE, 0, (struct sockaddr *) &receiver, sizeof(receiver)) <
0)
        {
            std::cout << "Error in send\n";

            perror(NULL);
            close(sender);
            exit(1);
        }
        sem_wait(args->sem_console);
        std::cout << "Sended to " << args->addrs[i].ToString() << "\n";
        sem_post(args->sem_console);
    }

    sem_wait(args->sem_flag);
    *(args->flag) = true;
    sem_post(args->sem_flag);
    close(sender);
    return NULL;
}

int createTcpSocket(int port)
{
    int descriptor;
    if((descriptor = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP))
    {
        perror(NULL);
        exit(1);
    }

    sockaddr_in addr = create_sockaddr(port);
    if(bind(descriptor, (sockaddr*)&addr, sizeof(addr)) < 0)
    {
        perror(NULL);
        exit(1);
    }

    return descriptor;
}

struct TcpMessage
{
    IpAddr addr;
};

void tcpServer()
{
    int listener = createTcpSocket(LISTEN_PORT);

    int start = time(NULL);

    while(time(NULL) < start + SERVER_EXIST_TIME)
    {
        int fd;
        if(listen(listener, 10) != 0)
        {
            perror(NULL);

```

```

        exit(1);
    }

    fd = accept(listener, NULL, NULL);
    if(fd < 0)
    {
        perror(NULL);
    }

    TcpMessage messg;
    read(fd, &messg, sizeof(messg));

    // logic
    TcpMessage response;
    response.addr = IpAddr("192.168.1.3", 24);

    write(fd, &response, sizeof(response));
}
std::cout << "Exiting server";
close(listener);
}

IpAddr ask_server(IpAddr myaddr)
{
    int sender = createTcpSocket(SEND_PORT);

    sockaddr_in serv = create_sockaddr(LISTEN_PORT);
    if(connect(sender, (sockaddr*)&serv, sizeof(serv)))
    {
        perror(NULL);
        exit(1);
    }

    TcpMessage messg;
    messg.addr = myaddr;
    write(sender, &messg, sizeof(messg));
    TcpMessage resp;

    read(sender, &resp, sizeof(resp));

    close(sender);
    return resp.addr;
}

IpAddr get_new_addr(IpAddr myaddr, std::vector<IpAddr> addrs)
{
    for (int i = 0; i < addrs.size(); ++i)
    {
        if(addrs[i].host_number() == 2)
        {
            return ask_server(myaddr);
        }
    }

    // start listen
    bool isFirst = true;
    int listener = create_socket(LISTEN_PORT);
    int wait_time = WAIT_TIME;
    timeval tv;
    tv.tv_sec = 0;
    tv.tv_usec = WAIT_TIME;
    setsockopt(listener, SOL_SOCKET, SO_RCVTIMEO, (const char*)&tv, sizeof tv);
    Message messg;

    sockaddr_in sender;
    socklen_t sender_len;
    int n = 0;

    int start = std::chrono::duration_cast<std::chrono::milliseconds >(
        std::chrono::system_clock::now().time_since_epoch()
    ).count();

    int now = std::chrono::duration_cast<std::chrono::milliseconds >(
        std::chrono::system_clock::now().time_since_epoch()
    ).count();

    while (now < start + WAIT_TIME)
    {

```

```

// std::cout << "Current time " << time(NULL) << "\n";
bool recvFlag;
if (n = recvfrom(listener, &messg, BUFF_SIZE, 0, (sockaddr*)&sender, &sender_len) < 0)
{
    if(errno != EAGAIN && errno != EWOULDBLOCK)
    {
        std::cout << "Error in recvfrom in listen stage\n";
        perror(NULL);
        exit(1);
    }

    recvFlag = false;
}
if(recvFlag)
{
    std::cout << "Received from in wait state " << messg.sender.ToString() << "\n";
    isFirst = false;
}

now = std::chrono::duration_cast<std::chrono::milliseconds >(
    std::chrono::system_clock::now().time_since_epoch()
).count();
}

if(!isFirst)
{
    sleep(2);
    return ask_server(myaddr);
}

std::vector<IpAddr> collisions;
sem_t* sem_flag = new sem_t();
if(sem_init(sem_flag, 0, 1))
{
    perror("Sem init error");
    exit(1);
}

sem_t* sem_console = new sem_t();
if(sem_init(sem_console, 0, 1))
{
    perror("Sem init error");
    sem_close(sem_flag);
    sem_destroy(sem_flag);
    exit(1);
}

bool isDoneDistr = false;

DistrArgs* args = new DistrArgs();
args->sem_flag = sem_flag;
args->flag = &isDoneDistr;
args->myAddr = myaddr;
args->addrs = addrs;
args->sem_console = sem_console;
pthread_t pid;
pid = pthread_create(&pid, NULL, DoDistr, args);

std::cout << "Start sending messages\n";

tv.tv_sec = 0;
tv.tv_usec = WAIT_TIME_DISTR;
setsockopt(listener, SOL_SOCKET, SO_RCVTIMEO, (const char*)&tv, sizeof tv);
while(true)
{
    sem_wait(sem_flag);
    if(isDoneDistr)
    {
        sem_post(sem_flag);
        break;
    }
    sem_post(sem_flag);
    bool recvFlag = true;
    if (n = recvfrom(listener, &messg, BUFF_SIZE, 0, (sockaddr*)&sender, &sender_len) < 0)
    {

```

```

        // sem_wait(sem_console);
        // std::cout << "Not received\n";
        // sem_post(sem_console);

        if(errno != EAGAIN && errno != EWOULDBLOCK)
        {
            std::cout << "Error in recvfrom in listen stage\n";
            perror(NULL);

            sem_close(sem_flag);
            sem_destroy(sem_flag);

            sem_close(sem_console);
            sem_destroy(sem_console);
            delete sem_flag;
            delete sem_console;

            exit(1);
        }

        recvFlag = false;
    }

    if (recvFlag)
    {
        sem_wait(sem_console);
        std::cout << "Received from in wait thread " << messg.sender.ToString() << "\n";
        sem_post(sem_console);
        collisions.push_back(messg.sender);
    }
}
sem_close(sem_flag);
sem_destroy(sem_flag);

sem_close(sem_console);
sem_destroy(sem_console);
delete sem_flag;
delete sem_console;

close(listener);

bool amIServer = true;
if(!collisions.empty())
{
    for(int i = 0; i < collisions.size(); ++i)
    {
        if(collisions[i].host_number() < myaddr.host_number())
        {
            amIServer = false;
        }
    }
}

if(amIServer)
{
    // fork()

    int pid = fork();
    if(pid < 0)
    {
        perror(NULL);
        exit(1);
    }
    if(pid == 0)
    {
        tcpServer();
        exit(0);
    }

    std::cout << "Child process start\n";

    IpAddr res = myaddr;

    res.set_host(0);
    float bits = ceil(log(addr.size() + 2) / log(2));
    bits = bits + 0.5;
    int int_bits = bits;
    int mask = 32 - int_bits;

```



```

        res.mask = mask;
        res.set_host(2);

        return res;
    }

    // wait til .2 shows up
    float bits = ceil(log(addrsize.size() + 2) / log(2));
    bits = bits + 0.5;
    int int_bits = bits;
    Mask = 32 - int_bits;
    return ask_server(myaddr);
}

IpAddress findMyAddr()
{
    std::string res = execute_command("ip -o -f inet addr show | awk '/scope global/ {print $4}'");

    int pos = res.find_first_of('/');
    short mask = stoi(res.substr(pos + 1));
    res = res.substr(0, pos);
    return IpAddr(res, mask);
}

int main() {
    // Анализ информации о сети и автоматическая настройка параметров
    // Здесь можно добавить логику для анализа и определения параметров ip_address, netmask,
    gateway
    IpAddr myAddr = findMyAddr();
    std::cout << "My addr " + myAddr.ToString() << "\n";
    std::vector<IpAddress> addrs = pingall(myAddr, myAddr.mask);
    sort_addrs(addrs);
    for(IpAddr addr : addrs)
    {
        std::cout << addr.ToString() << "\n";
    }

    IpAddr res = get_new_addr(myAddr, addrs);
    std::cout << "MyNew addr " << res.ToString() << "\n";

    // auto newAddrs = new_addrs(addrs);

    // Пример автоматической настройки сетевых параметров
    // configure_network("192.168.1.100", "255.255.255.0", "192.168.1.1");

    return 0;
}

```