Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Дисциплина: Операционные системы и системное программирование

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту
на тему
УТИЛИТА УДАЛЕННОГО УПРАВЛЕНИЯ КОМПЬЮТЕРОМ

БГУИР КП 1–40 02 01 01 209 ПЗ

Студент: А. И. Дроздов

Руководитель: Д. Н. Басак

МИНСК 2024

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В последние годы в связи с переходом всего мира IT на удаленную работу стал очень актуален вопрос звонков по сети и доступа к своему рабочему месту. Многие компании работают полностью удаленно, даже не имея офиса. Это привело к росту популярности таких сервисов для видеоконференций, как Zoom, Google Meet, Microsoft Teams и другие. Для максимальной производительности, приложения для удаленного контроля и видеозвонков обычно пишут на языке C++.

Возможность удаленного подключения к рабочему месту особенно важна в окружениях, где компьютеры настроены на работу с конфиденциальными данными в частной сети. Если дать возможность подключатся к ней любым компьютерам, которые могут находится где угодно вместе с их владельцами, то могут возникнуть утечки данных.

C++ - это мощный [1] и универсальный язык программирования, который широко используется при разработке программных приложений, системного программного обеспечения, драйверов устройств и встроенного микропрограммного обеспечения.

С тех пор этот язык эволюционировал и стал одним из самых популярных и широко используемых языков программирования в мире. Его популярность обусловлена его эффективностью, гибкостью и широким спектром применений, для которых он может быть использован. C++ известен своей высокой производительностью, поскольку позволяет выполнять низкоуровневые манипуляции с оборудованием и памятью, что делает его подходящим для разработки ресурсоемких приложений.

Для отправки видео и аудио по сети обычно используют протокол UDP либо RTP, так как он помогает отправлять постоянные потоки данных эффективно, безопасно и с минимальными потерями качества.

Также существует протокол SRTP – протокол RTP, который выполняется по защищенному соединению. А протокол RTCP добавляет возможность контролировать трафик и качество его получения. Он отправляет дополнительные метаданные о качестве и количестве полученных пакетов.

Захват и применение действий с клавиатурой и мышью – задача нетривиальная и зависит от целевой операционной системы. Целевая операционная система – Linux, так как он наиболее часто используется разработчиками. Почти все графические окружения Linux работают с помощью X.org window server. В последние годы внедряется и новый протокол Wayland, но пока что он поддерживает гораздо меньше программ и драйверов.

1 ПОСТАНОВКА ЗАДАЧИ

Исследовать принцип работы протокола RTP, UDP и реализацию приложений для удаленного доступа. Исследовать возможности захвата и применения действий с клавиатурой и мышью на X window server. Реализовать протокол взаимодействия и графический интерфейс пользователя с возможностью настраивать параметры соединения.

Программа для удаленного взаимодействия должна содержать классы, отражающие основной функционал: запись видео, запись аудио, вывод аудио и видео, захват событий ввода с хост-системы, применение событий на управляемой системе.

Для обеспечения максимальной производительности и для того, чтобы избежать задержки при использовании графического интерфейса, все взаимодействие между клиентом и сервером должно выполняться в отдельных потоках. Обычно для каждой операции создается отдельный поток, так как нельзя, например, одновременно слушать входящие сообщения на порту и отправлять сообщения на другой сервер последовательно.

Необходимо добавить возможность отключать микрофон и проводить настройку параметров соединения. Обычно как минимум необходимо регулировать число кадров в секунду (как часто сохранять и отправлять изображения экрана), качество изображения (для экономии трафика изображения должны сжиматься в JPG), а также размер одного пакета данных.

Размер пакета данных должен быть кратным 2 (для выравнивания), а максимальный теоретический размер пакета ограничен 65535 байтами. На практике все зависит от значения MTU соединения – оно отличается у Ethernet и Wi-Fi сетей, а при использовании VPN или иных средств ещё меньше данных возможно поместить в один фрейм данных. Необходимо выбрать подходящие значения по умолчанию.

При отключении клиента необходимо в разумный промежуток времени это проверить и вернуться к главному экрану.

Качество передаваемых данных должно сохраняться при разных условиях соединения и даже на расстоянии. Необходимо поддерживать возможность подключиться сам к себе (ip 127.0.0.1)

На главном экране должен быть выбор, является ли текущая система хост-системой, или управляемой системой. Если система основная, то она не должна делать запись экрана и захватывать события ввода. Если система управляемая, то она должна отправлять изображения на хост-систему и применять события ввода. Обе системы должны одновременно и записывать аудио, и воспроизводить.

2 ОБЗОР ЛИТЕРАТУРЫ

2.1 Анализ существующих аналогов

В настоящее время существует огромное множество приложений для удаленного доступа. Все началось с протокола RDP, которое использовалось для удаленного доступа к рабочему столу в Windows. Оно иногда использовалось технической поддержкой Microsoft, а также некоторыми сервисами удаленных серверов. На рисунке 2.1 можно наблюдать окно подключения к удаленному рабочему столу. Со стороны сервера нужно было запустить нужную службу.

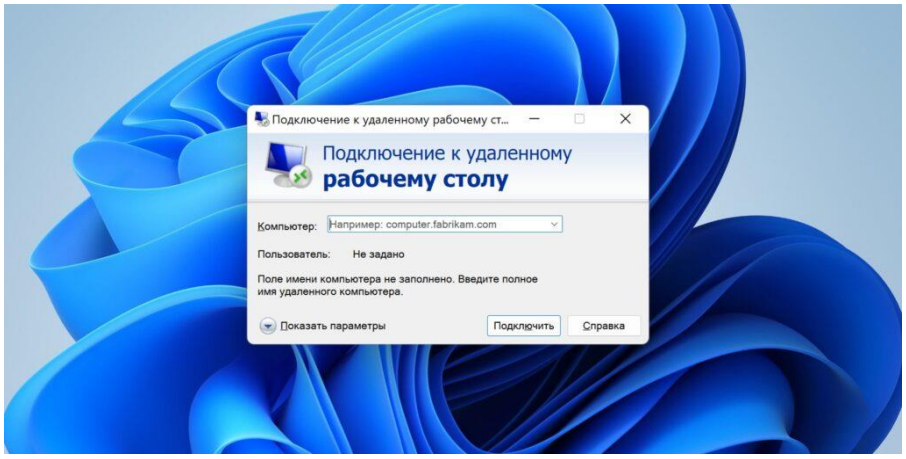

Рисунок 2.1 – Удаленный рабочий стол Windows

Apple внедрила аналогичную технологию в MacOS. Из особенностей, есть возможность подключаться к устройствам по их Apple ID.

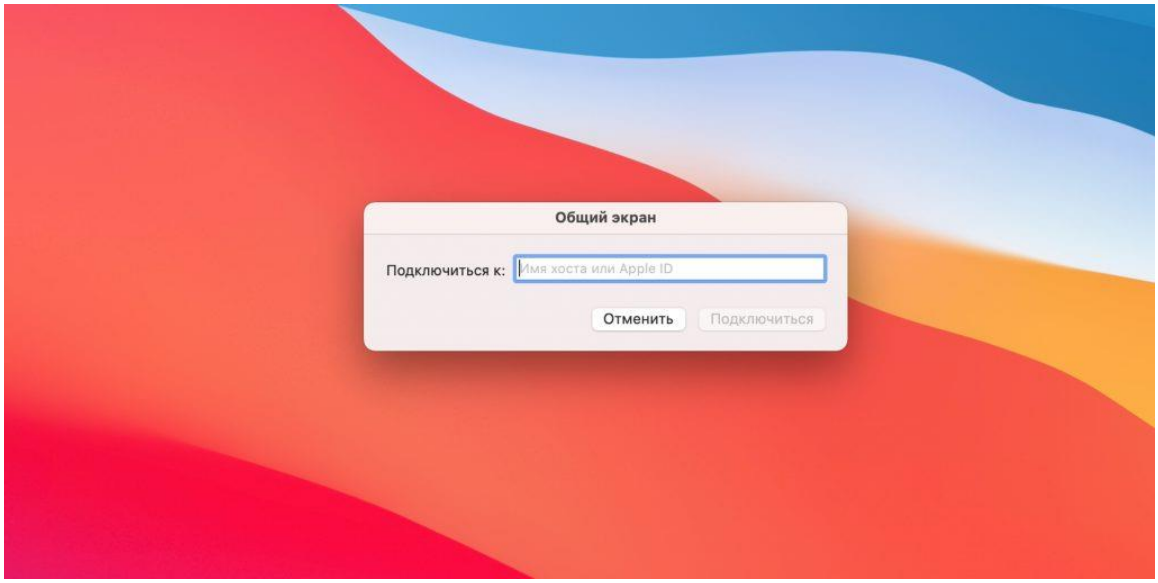

Рисунок 2.2 – Общий экран Apple

С развитием технологий, устаревшая и небезопасная технология RDP стала заменяться более совершенным протоколом VNC.

В настоящее время протокол VNC используется на всех облачных провайдерах для консоли аварийного доступа (emergency shell). Иногда даже нет никакого оконного менеджера. Таким образом VNC может использоваться как замена протоколу SSH, если конфигурация протокола SSH нарушена.

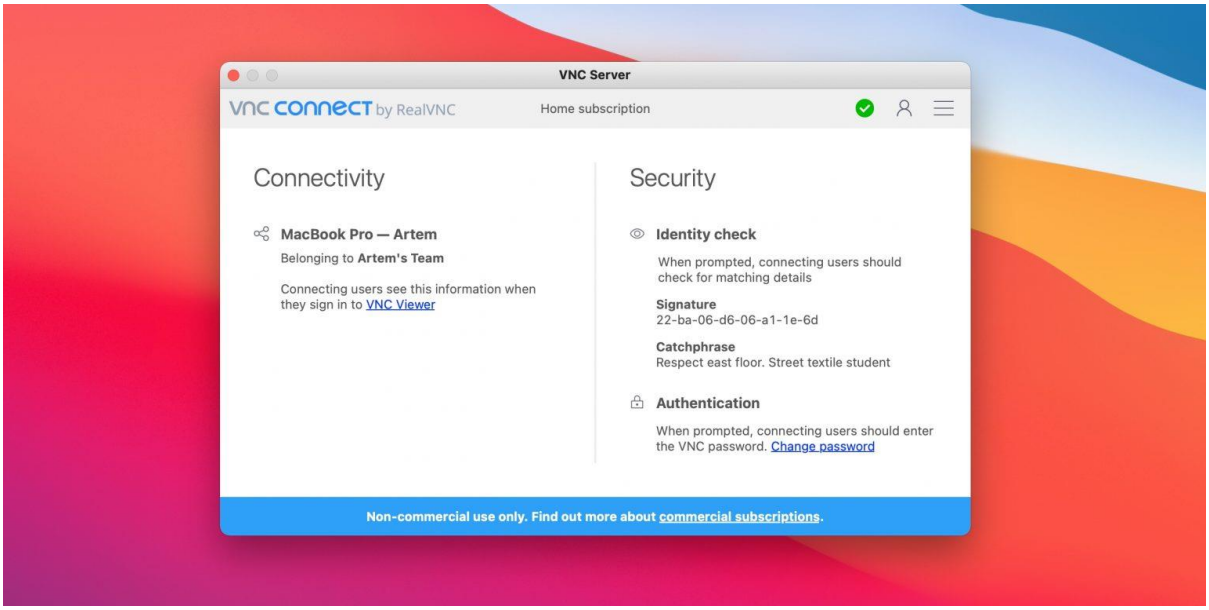

Рисунок 2.3 – RealVNC клиент

Клиенты VNC есть под все операционные системы. На рисунке 2.3 показан клиент RealVNC. Также существует клиент TigerVNC и другие.

VNC (Virtual Network Computing) - это протокол для удаленного управления рабочим столом с графическим интерфейсом. Он позволяет пользователю управлять удаленным компьютером или сервером, как если бы он находился непосредственно перед ним. VNC работает по архитектуре клиент-сервер: на удаленной системе запускается VNC-сервер, который захватывает изображение экрана и передает его на VNC-клиент, установленный на локальной машине. Клиент отображает полученное изображение на своем экране в режиме реального времени.

Кроме трансляции экрана, VNC обеспечивает двустороннюю передачу ввода с клавиатуры и мыши. Клавиатурный ввод и движения мыши, выполненные на локальной системе, передаются на VNC-сервер и интерпретируются так, как если бы они были выполнены непосредственно на удаленной системе. Эта особенность позволяет полностью управлять удаленным компьютером или сервером, взаимодействуя с его графическим интерфейсом.

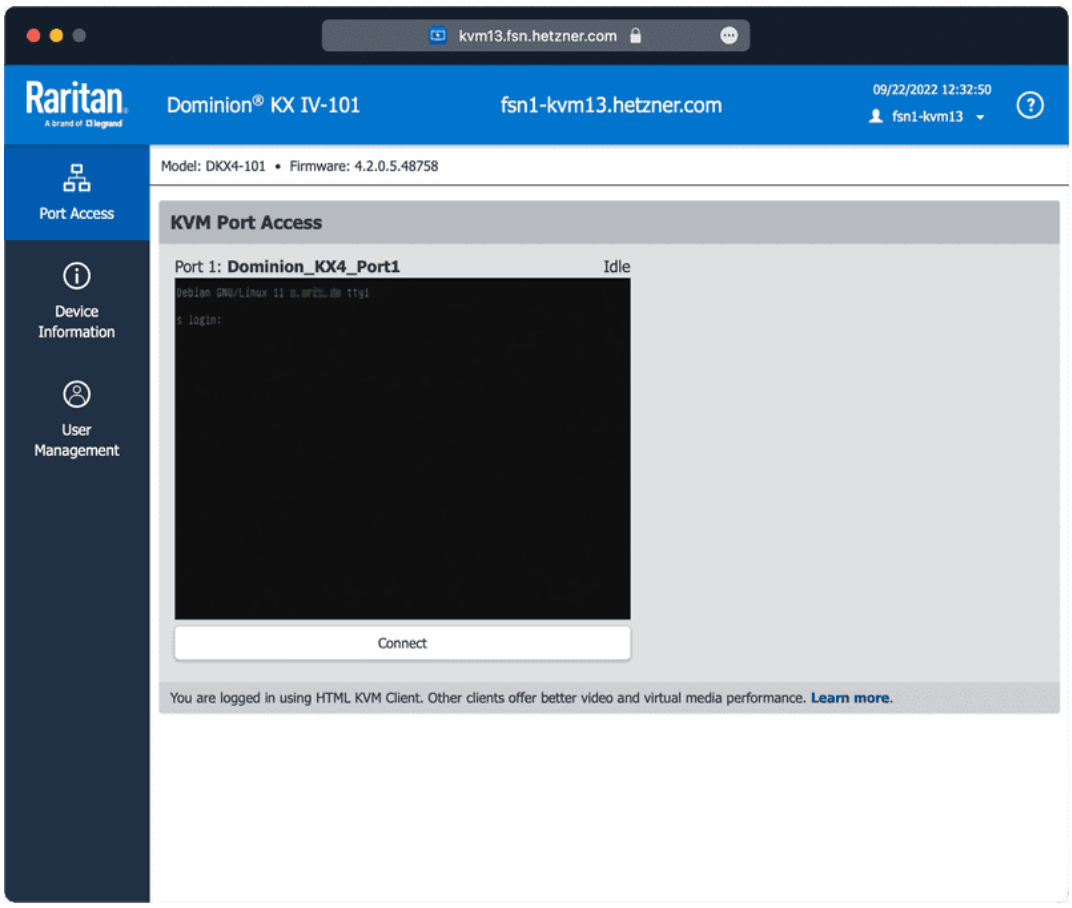

Рисунок 2.3 – Hetzner emergency shell

На рисунке 2.3 показана консоль аварийного доступа одного из крупнейших облачных провайдеров Hetzner. Она используется лишь в экстренных случаях, когда нет возможности подключится к серверу по SSH или другими способами.

Но существуют и более простые для обывателя программы удаленного доступа, о которых слышал каждый. Эти программы не требуют глубоких технических знаний и легко устанавливаются даже неопытными пользователями для удаленного доступа к компьютерам.

Одной из наиболее известных и популярных таких программ является TeamViewer. Она предоставляет удобный и интуитивно понятный интерфейс для установки полноценного удаленного подключения между компьютерами.

Особенностью TeamViewer является то, что для подключения необходимо ввести лишь код доступа. То есть это уже не полностью P2P архитектура, а клиент-серверная архитектура, где все сессии изначально проходят через центральные серверы TeamViewer. Благодаря такому подходу, TeamViewer способен обходить ограничения брандмауэров и NAT, что упрощает процесс подключения, особенно для неопытных пользователей. За счет централизованной архитектуры можно легко подключаться даже через сложные сетевые настройки и фаерволы.

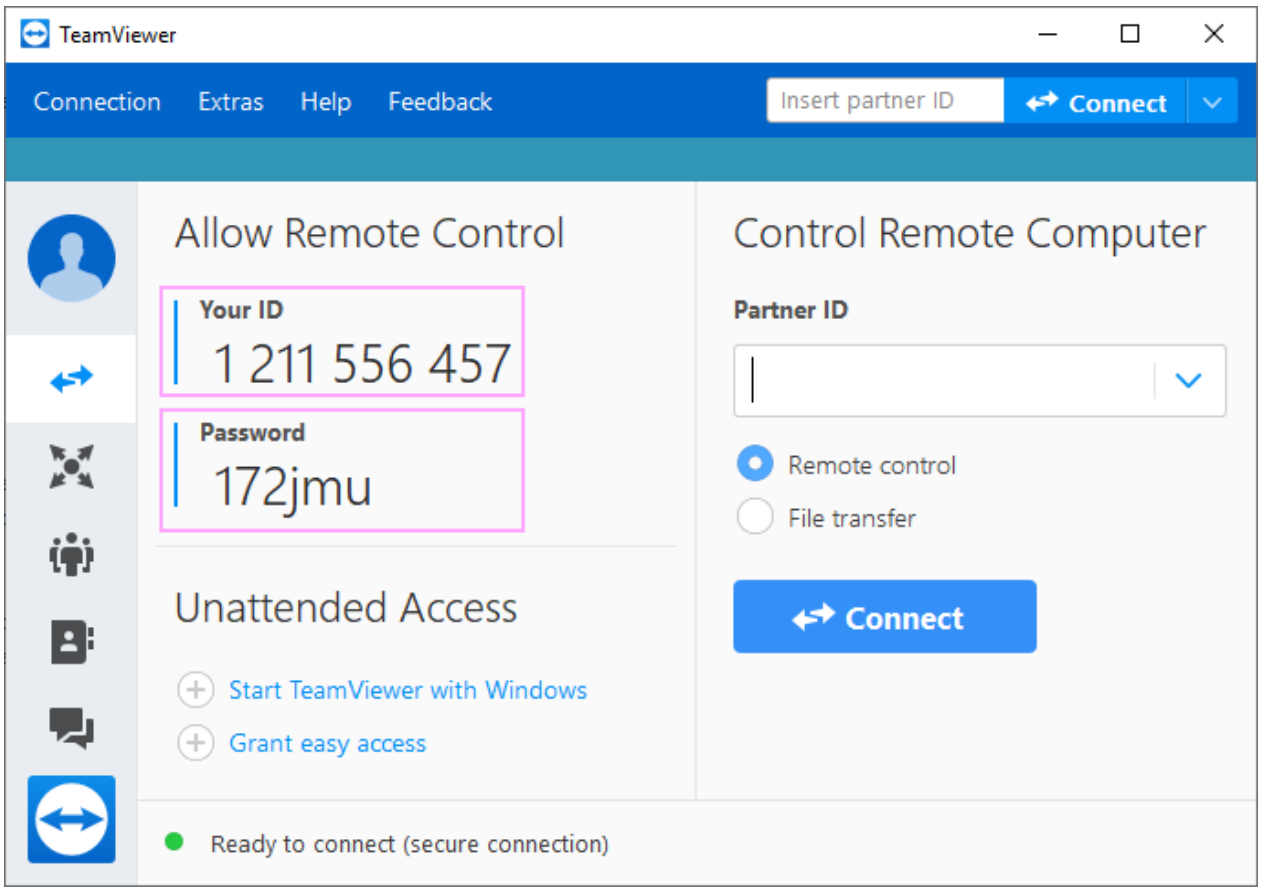

Рисунок 2.4 – TeamViewer

Интерфейс TeamViewer максимально прост (как показан на рисунке 2.4), и в то же время функционален. Он активно используется компаниями, такими как Yandex и Norton Rose Fulbright.

Ещё одной популярной альтернативой является AnyDesk, который представляет еще более минималистичный и легковесный интерфейс. Одним из ключевых плюсов AnyDesk является возможность запуска портативной версии приложения, которая не требует установки и может работать прямо с внешнего носителя, обеспечивая удобство для мобильных пользователей.

Как и TeamViewer, AnyDesk использует клиент-серверную архитектуру и систему кодов доступа для упрощения процесса подключения через NAT и фаерволы. Он также предоставляет шифрование трафика и различные функции, такие как передача файлов, удаленный доступ к мобильным устройствам и даже инструменты для удаленной техподдержки.

Благодаря своей простоте в использовании и минималистичному дизайну, AnyDesk завоевал популярность среди домашних пользователей и небольших предприятий, которым требуется быстрый и легкий способ для удаленного доступа.

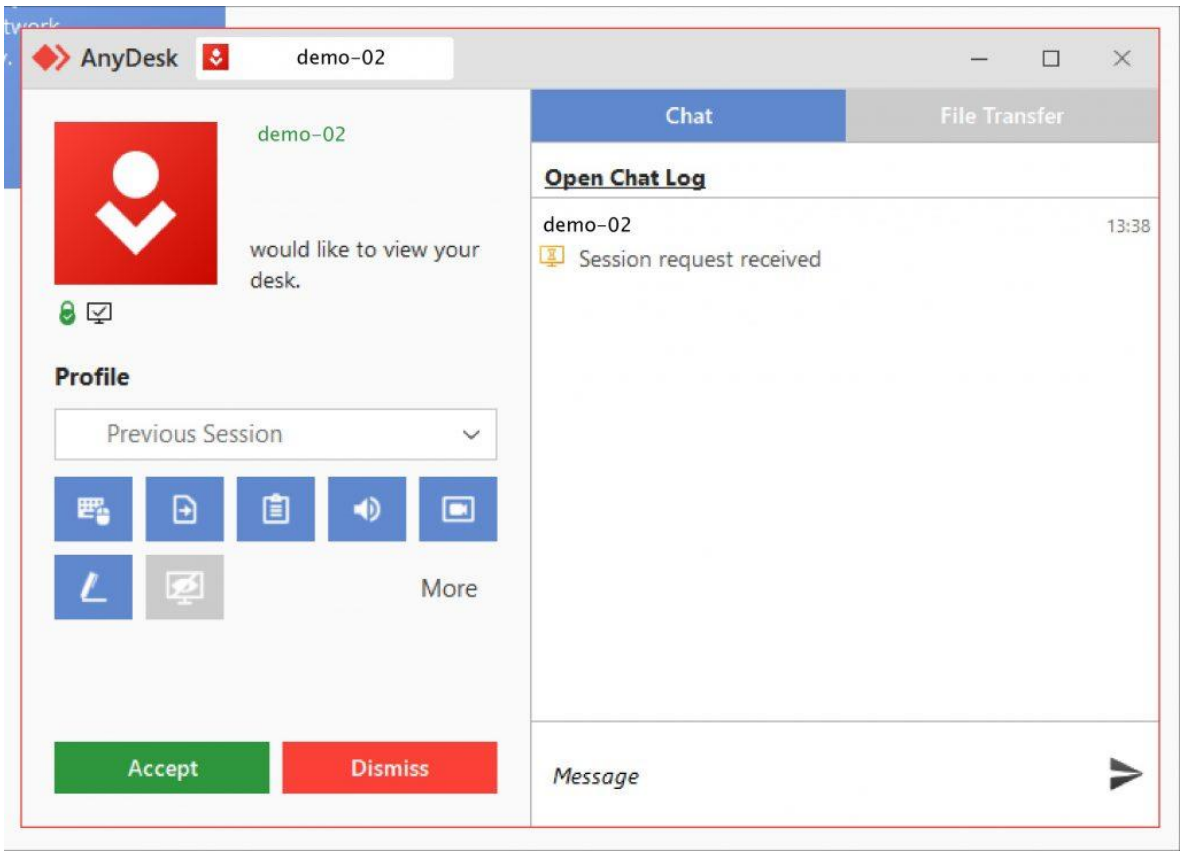

Рисунок 2.5 – AnyDesk

На рисунке 2.5 можно наблюдать интерфейс AnyDesk.

2.2 Обзор программирования сокетов Linux

2.2.1 Сокеты – концепция

Сокеты [2] представляют собой универсальный программный интерфейс, впервые реализованный в операционной системе Berkeley UNIX, но позже распространившийся практически на все вариации Unix, включая Linux. Хотя реализации могут немного отличаться, основной набор функций остается неизменным. Изначально сокеты использовались в программах на C/C++, но сейчас большинство языков программирования, таких как Perl, Java и другие, предоставляют средства для работы с ними.

Сокеты – это мощный и гибкий механизм, который позволяет организовать взаимодействие между процессами как на одном компьютере, так и в локальной сети или через Интернет. Это открывает широчайшие возможности для создания распределенных приложений различной сложности. Более того, благодаря сокетам можно взаимодействовать с программами, работающими под управлением других операционных систем. Например, Windows Sockets, основанный на socket API, обеспечивает совместимость с Unix–системами.

Сокет (socket) – это конечная точка сетевых коммуникаций, своего рода "портал", через который можно отправлять и получать байты данных во внешний мир. Приложение просто записывает данные в сокет, а их дальнейшая буферизация, отправка, транспортировка и обработка осуществляются используемым стеком протоколов и сетевым оборудованием. Чтение данных из сокета происходит аналогичным образом. В программе сокет идентифицируется дескриптором – переменной типа int. Программа получает дескриптор от операционной системы при создании сокета и затем передает его функциям socket API для указания сокета, над которым нужно выполнить определенное действие.

Сокеты являются фундаментальной частью многих важных технологий и протоколов, таких как HTTP, FTP, SSH и многих других. Они обеспечивают базовый механизм для передачи данных между различными приложениями и системами, что делает их незаменимым инструментом для разработки современных сетевых и распределенных приложений.

2.2.2 Атрибуты сокета

Каждый сокет в Linux характеризуется тремя основными атрибутами: домен, тип и протокол. Эти атрибуты задаются при создании сокета с помощью функции socket() и остаются неизменными на протяжении всего его жизненного цикла.

Домен сокета определяет пространство адресов и множество используемых протоколов. Наиболее часто используются следующие домены:

- AF_UNIX – домен Unix, который использует файловую систему ввода/вывода Unix для межпроцессного взаимодействия на одном компьютере.
- AF_INET – домен Internet, который позволяет работать в любой IP–сети, включая Интернет.

Тип сокета определяет способ передачи данных и режим работы соединения. Наиболее распространенные типы сокетов:

- SOCK_STREAM – потоковый тип, обеспечивающий установку соединения и надежный канал передачи данных с гарантированной доставкой и упорядочиванием пакетов.
- SOCK_DGRAM – датаграммный тип, при котором данные передаются в виде отдельных сообщений (датаграмм) без установки соединения, что обеспечивает ненадежный обмен данными без гарантий доставки и упорядочивания.
- SOCK_RAW – низкоуровневые "сырые" сокеты, которые предоставляют прямой доступ к протоколам нижнего уровня, таким как IP или ICMP.

Атрибут протокол определяет конкретный протокол передачи данных, который будет использоваться сокетом. Часто этот атрибут однозначно определяется комбинацией домена и типа сокета. Например, для потоковых

сокетов в домене Internet (AF_INET, SOCK_STREAM) используется протокол TCP, а для датаграммных сокетов (AF_INET, SOCK_DGRAM) – протокол UDP.

Правильный выбор атрибутов сокета имеет решающее значение для обеспечения требуемого режима работы и характеристик сетевого взаимодействия, таких как надежность, упорядоченность, скорость передачи данных и т.д. Поэтому разработчики должны тщательно продумывать выбор этих атрибутов в зависимости от конкретных требований к приложению.

Кроме того, следует отметить, что поддержка сокетов в Linux реализована на уровне ядра операционной системы, что обеспечивает высокую производительность и эффективность работы с сетевыми соединениями. Благодаря этому, сокеты широко используются в различных сетевых приложениях и сервисах, таких как веб–серверы, почтовые клиенты, файловые серверы и многих других.

2.2.3 Адреса

В Linux, перед тем как передавать данные через сокет, необходимо установить соединение с определенным адресом в выбранном домене, что называется "именованием сокета". Вид адреса, который используется, зависит от домена, в котором работает сокет.

В случае домена Unix (AF_UNIX) адресом является имя файла в файловой системе, через который осуществляется обмен данными между процессами. В домене Internet (AF_INET) адрес состоит из комбинации IP–адреса и 16–битного номера порта, которые определяют конкретный узел в сети и сокет на этом узле соответственно.

Для связывания сокета с определенным адресом используется функция bind(). Она принимает в качестве параметров дескриптор сокета, указатель на структуру sockaddr, содержащую адрес, а также длину этой структуры. Структура sockaddr является обобщенной структурой, используемой для представления адресов в различных доменах. В ней присутствует поле sa_family, которое содержит идентификатор домена, используемого для данного сокета.

Для удобства работы с различными доменами существуют альтернативные структуры, такие как sockaddr_in для домена Internet. Эти структуры содержат дополнительные поля, специфичные для каждого домена, например, поля sin_addr и sin_port для IP–адреса и номера порта соответственно в случае sockaddr_in. При передаче указателя на эти альтернативные структуры в функцию bind(), они приводятся к типу sockaddr, чтобы обеспечить совместимость с обобщенным интерфейсом.

После вызова bind() сокет становится "привязанным" к указанному адресу и готов к приему или отправке данных, в зависимости от его типа. Для потоковых сокетов (SOCK_STREAM) требуется дополнительный шаг –

установка соединения с удаленным сокетом с помощью функции connect() для клиентов или listen()/accept() для серверов.

Важно отметить, что привязка сокета к адресу является необязательным шагом для некоторых типов сокетов, таких как SOCK_DGRAM (датаграммные сокеты). В этом случае адрес может быть указан при каждой отправке данных с помощью функции sendto().

Правильное именование и привязка сокетов к адресам имеют решающее значение для обеспечения корректного сетевого взаимодействия между приложениями. Поэтому разработчики должны тщательно продумывать выбор адресов и способ их использования в зависимости от конкретных требований к приложению и сетевой архитектуре.

2.2.4 Процесс установки соединения на серверной стороне

В Linux для успешного установления сервером соединения с клиентами необходимо пройти четыре обязательных этапа. Первоначально создается сокет с помощью функции socket(), и ему присваивается локальный адрес через функцию bind(). При наличии нескольких сетевых интерфейсов на сервере можно принимать соединения только с определенного интерфейса, указав его IP–адрес в качестве адреса при вызове bind(), либо с любого интерфейса, используя специальную константу INADDR_ANY. Порт, на котором будет работать сервер, также можно задать конкретным номером или автоматически (указав 0, что позволит системе выбрать свободный порт).

Следующим шагом является активация прослушивания входящих запросов на соединение с помощью функции listen(). Она переводит сокет в режим ожидания клиентских запросов и формирует очередь для хранения этих запросов. Размер очереди задается параметром backlog, при переполнении которой последующие запросы на соединение будут игнорироваться.

Затем сервер вызывает функцию accept() для получения нового сокета, связанного с клиентом из очереди ожидающих соединений. Функция accept() возвращает дескриптор этого нового сокета, который будет использоваться для обмена данными с конкретным клиентом, а исходный слушающий сокет остается открытым для приема других входящих соединений. Адрес клиента, с которым установлено соединение, можно получить опционально через дополнительный параметр функции accept().

Несмотря на то, что клиент и сервер могут иметь один и тот же общий IP–адрес, новый сокет, созданный функцией accept(), уникально идентифицируется парой адресов взаимодействующих хостов в протоколе TCP (комбинацией IP–адресов и номеров портов клиента и сервера).

После выполнения этих четырех шагов сервер готов к приему данных от клиента через созданный сокет. Аналогичным образом клиент также должен создать сокет, установить соединение с сервером с помощью функции connect() и затем использовать полученный сокет для отправки и приема данных.

Важно отметить, что для обеспечения высокой производительности и эффективности работы сервера, рекомендуется использовать асинхронный или многопоточный подход при обработке множества одновременных соединений. Это позволит избежать блокировки сервера и обеспечить своевременное обслуживание всех клиентов.

2.2.5 Процесс установки соединения на клиентской стороне

В Linux для установления соединения клиент использует функцию connect(), которая принимает в качестве параметров дескриптор сокета и адрес/порт сервера, с которым клиент хочет установить связь. Обычно нет необходимости явно привязывать клиентский сокет к локальному адресу, поскольку функция connect() автоматически назначает свободный порт для клиента во время установки соединения. Однако в некоторых случаях может возникнуть необходимость явно указать локальный порт, используя функцию bind(), перед вызовом функции connect(). Это может быть полезно, например, при использовании протокола rlogind или других специализированных протоколов.

После создания сокета с помощью функции socket(), клиент может опционально вызвать bind(), чтобы привязать сокет к определенному локальному адресу и порту. Затем клиент вызывает connect(), передавая в качестве аргументов дескриптор сокета и адрес/порт удаленного сервера, к которому нужно подключиться. Функция connect() пытается установить соединение с указанным удаленным узлом и возвращает ошибку, если это не удается.

Если клиент не привязывает явно локальный адрес с помощью bind(), то операционная система автоматически выбирает для него свободный локальный порт при установке соединения через connect(). Это наиболее распространенный и предпочтительный сценарий, поскольку он упрощает разработку клиентского кода и снижает вероятность конфликтов с другими приложениями, использующими сокеты.

Тем не менее, в некоторых специфических ситуациях, таких как работа с определенными протоколами или приложениями, требующими фиксированных номеров портов для клиентов, может потребоваться явная привязка клиентского сокета к локальному адресу и порту перед вызовом connect(). Например, протокол rlogind, используемый для удаленного входа в систему, требует, чтобы клиенты использовали зарезервированные привилегированные порты (номера портов меньше 1024).

В целом, явная привязка клиентского сокета к локальному адресу и порту с помощью bind() перед установкой соединения с помощью connect() является опциональной и необходимой только в специфических случаях. В большинстве ситуаций автоматическое назначение локального порта операционной системой во время connect() является предпочтительным

вариантом, обеспечивающим более простую и гибкую реализацию клиентского кода.

2.2.6 Обмен данными между хостами

После успешной установки соединения между хостами начинается обмен данными. Для отправки данных используется функция send(), а для приема данных – функция recv(). Хотя теоретически можно использовать стандартные POSIX–функции read() и write() для чтения и записи в сокеты, это не рекомендуется, поскольку они могут быть не полностью совместимы на разных платформах и не поддерживать весь необходимый функционал, предоставляемый функциями send() и recv().

Функция send() принимает следующие параметры: дескриптор сокета sockfd, через который будут отправляться данные, указатель msg на буфер с данными для передачи, длину len этого буфера в байтах и необязательные флаги flags для управления дополнительными опциями передачи. Два распространенных флага: MSG_OOB для срочной передачи данных вне основного потока и MSG_DONTROUTE для запрета маршрутизации данных (если поддерживается). Функция send() возвращает количество успешно переданных байтов, которое может быть меньше длины буфера в случае, если не все данные были отправлены сразу.

Функция recv() имеет схожий интерфейс: sockfd – дескриптор сокета, из которого будут считываться данные, buf – буфер для принятых данных, len – размер буфера, flags – флаги управления. Флаг MSG_OOB позволяет получать срочные данные, передаваемые с помощью MSG_OOB при отправке, а флаг MSG_PEEK – просматривать данные, не извлекая их из системного буфера приема. В случае разрыва удаленной стороной соединения recv() возвращает 0, что сигнализирует о закрытии соединения.

Важно отметить, что функции send() и recv() не гарантируют передачу или прием всех данных за один вызов. Поэтому при программировании сетевых приложений необходимо использовать циклы для отправки или приема всех данных по частям, пока не будет передана или получена вся требуемая порция данных. Кроме того, необходимо обрабатывать возможные ошибки и исключительные ситуации, такие как разрыв соединения, переполнение буферов и т.д.

Для повышения производительности и эффективности передачи данных рекомендуется использовать буферизацию и асинхронный режим работы с сокетами, а также оптимизировать размеры буферов в соответствии с требованиями приложения и сетевой инфраструктуры.

Помимо функций send() и recv(), в программировании сокетов также могут использоваться другие функции, такие как sendto() и recvfrom() для работы с датаграммными сокетами (SOCK_DGRAM), а также вспомогательные функции для управления опциями сокетов, таймаутами и другими параметрами.

В целом, грамотное использование функций send() и recv(), а также других функций программирования сокетов, является ключевым аспектом разработки надежных и высокопроизводительных сетевых приложений в Linux.

2.2.7 Закрытие сокета

По завершении обмена данными через сокет необходимо корректно его закрыть с помощью функции close(). Эта функция разрывает соединение и освобождает все системные ресурсы, связанные с данным сокетом. Однако в некоторых случаях может возникнуть необходимость в более тонком контроле за процессом закрытия соединения. Для этих целей в Linux используется функция shutdown(), которая позволяет частично запретить передачу данных в одном или обоих направлениях, не разрывая само соединение полностью.

Функция shutdown() принимает в качестве параметров дескриптор сокета, подлежащего частичному закрытию, и значение параметра how, определяющего направление, в котором будет запрещена передача данных. Возможные значения параметра how: 0 – прием данных, 1 – отправка данных, 2 – оба направления.

После вызова shutdown() с параметром 2, эффективно запрещающим любой обмен данными через сокет, необходимо вызвать функцию close() для полного освобождения системных ресурсов, связанных с этим сокетом.

Использование функции shutdown() может быть полезным в различных сценариях работы с сокетами. Например, при реализации протоколов, требующих особой процедуры завершения соединения, или при необходимости асинхронно завершить отправку или прием данных, не дожидаясь завершения операции в противоположном направлении.

Важно понимать, что даже после вызова shutdown() с любым значением параметра how, сокет остается открытым и продолжает занимать системные ресурсы до тех пор, пока не будет вызвана функция close(). Поэтому в сетевых приложениях крайне важно корректно закрывать сокеты после завершения их использования, чтобы избежать утечек ресурсов и связанных с этим проблем производительности и стабильности.

Кроме функций close() и shutdown(), в Linux также доступны другие функции для управления состоянием сокетов, такие как fcntl() для установки неблокирующего режима работы, setsockopt() и getsockopt() для настройки различных опций сокетов и т.д. Эти функции позволяют более тонко настраивать поведение сокетов в соответствии с требованиями конкретного приложения.

Грамотное использование функций для закрытия и управления состоянием сокетов является критически важным аспектом разработки надежных и корректно работающих сетевых приложений в Linux. Пренебрежение этими аспектами может привести к утечкам ресурсов, нестабильности и другим проблемам в работе приложений.

2.2.8 Обработка ошибок

При работе с сокетами в Linux крайне важно учитывать возможность возникновения ошибок на любом этапе и корректно их обрабатывать. Практически все рассмотренные функции для работы с сокетами, такие как bind(), listen(), accept(), connect(), send(), recv(), shutdown(), close(), при успешном выполнении возвращают 0 или дескриптор сокета, а при возникновении ошибки возвращают значение –1 и устанавливают соответствующий код ошибки в глобальную переменную errno согласно определениям в заголовочном файле errno.h. Регулярная проверка кода возврата и значения errno позволяет своевременно реагировать на возникающие ошибки и отказы.

Обработка ошибок при работе с сокетами имеет решающее значение для обеспечения надежности и устойчивости сетевых приложений. Реакция на возникшую ошибку может варьироваться в зависимости от сложности и типа ошибки. В некоторых случаях может быть целесообразно пытаться выполнить операцию, приведшую к ошибке, повторно после устранения причины ошибки. В других случаях может потребоваться завершить выполнение программы с соответствующим кодом ошибки и выводом диагностической информации.

Например, при возникновении ошибки при создании сокета или привязке его к серверному адресу разумной реакцией может быть попытка создать сокет заново после устранения причины ошибки, такой как нехватка системных ресурсов или конфликт с другим приложением, использующим тот же порт. В то же время, ошибка передачи байт между клиентом и сервером может иметь совсем иную природу, такую как разрыв соединения, ошибка сети или проблемы с буферизацией данных. В этом случае решением может быть как повторная отправка данных, так и экстренный выход из программы с соответствующим кодом ошибки и уведомлением администратора или пользователя.

Кроме того, важно учитывать, что некоторые ошибки могут быть временными или зависеть от состояния системы. В таких случаях может быть полезно использовать механизмы повторных попыток с экспоненциальным наращиванием интервалов между попытками, чтобы избежать перегрузки системы и обеспечить плавное восстановление работоспособности приложения после устранения причины ошибки.

Грамотная обработка ошибок также включает в себя ведение подробных журналов ошибок, содержащих информацию о типе ошибки, стеке вызовов, значениях параметров и других диагностических данных, которые могут быть полезны для выявления и устранения причин ошибок.

Таким образом, корректная обработка ошибок при работе с сокетами в Linux является критически важным аспектом разработки надежных и устойчивых сетевых приложений. Пренебрежение этим аспектом может

привести к нестабильности, сбоям и потере данных, что недопустимо для большинства современных приложений, работающих в сетевой среде.

2.3 Обзор методов и алгоритмов решения поставленной задачи

Приложение для удаленного управления является оконным приложением на Qt [3] (так же, как и, например, Anydesk), что позволяет ему быть кроссплатформенным. Для упрощения концепта, приложению не нужен централизованный сервер – каждый клиент является одновременно и клиентом, и сервером. Приложение работает в локальной сети (теоретически можно подключится и к удаленному серверу, но это нетривиально показать и могут быть потери пакетов).

Для передачи изображений был использован протокол RTP, аудио передается по UDP, а события ввода – по протоколу TCP. Сам по себе протокол RTP не гарантирует упорядоченности полученных данных и качества, но помогает в удобном формате отправлять изображения. Так как пакеты могут приходить в произвольном порядке, протокол был расширен – каждый пакет данных содержит текущий номер фрейма (1 фрейм = 1 скриншот экрана), и текущий номер sequence (так как каждый фрейм может быть потенциально большим, он разбивается на чанки и отправляется по частям, восстановить изображение можно лишь в правильном порядке). Для аудио был выбран протокол UDP вместо TCP для улучшения качества звука, т.к. звук — это непрерывный поток байт, а не что-то, имеющее четкую структуру. Для реконструкции изображения и получения скриншотов экрана (для получения новых фреймов) была использована библиотека OpenCV [4]. Интерфейсы были разработаны с помощью Qt Designer. Для событий ввода был выбран протокол TCP для гарантий упорядоченности событий. Если этого не сделать, можно, например, сначала сделать клик мышью, а лишь потом переместить ее на нужную позицию.

В современных приложениях обычно используется множество протоколов – сначала на самом нижнем уровне есть physical и data link layer – либо LAN, либо WAN, которое предоставляет доступ к глобальному интернету. Здесь и далее описывается модель OSI. Наблюдать стек, используемый в подобных приложениях можно на рисунке 2.6.

Далее идет network layer – IP (internet protocol) – используется для адресации и локации компьютеров в сети.

Самое главное происходит на transport layer – базово всегда используется протокол UDP, так как он быстрее и лучше для приложений с непрерывными потоками данных.

Но использовать сырой протокол UDP обычно нецелесообразно. Были созданы особые протоколы, направленные именно на отправку медиа.

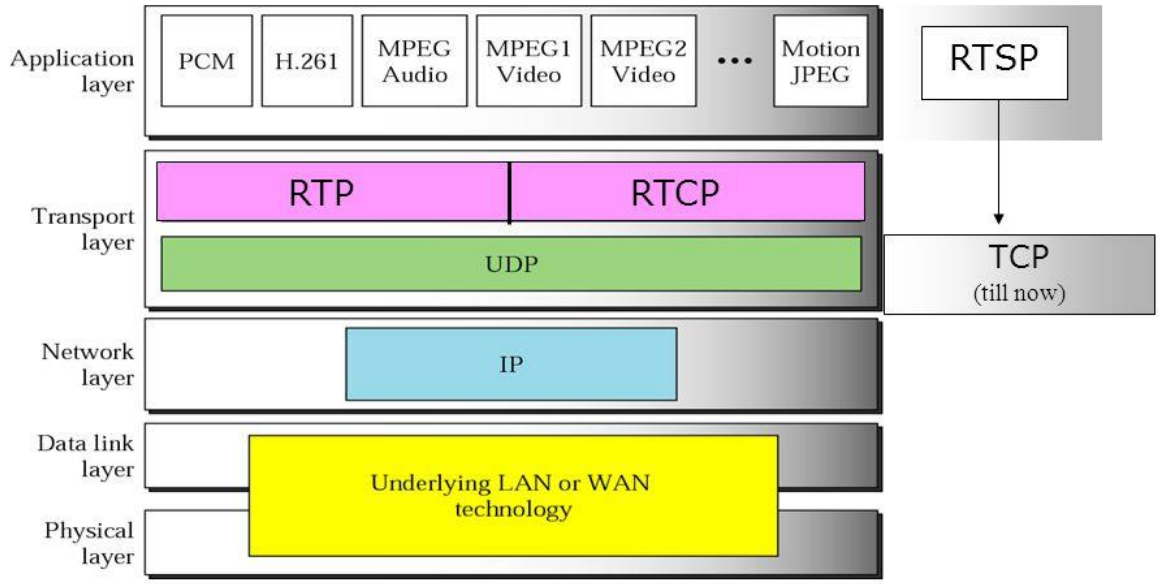

Рисунок 2.6 – стек, используемый в подобных приложениях

Протокол RTP задает формат данных для отправки медиа. Существует множество форматов данных, в данной работе использовался формат Generic для произвольных данных. Для лучшей производительности в реальных проектах используются протокол H264 и подобные. Они работают за счет небольшой нагрузки на процессор в момент закодирования фрейма, зато по сети они отправляются гораздо более эффективно.

Сами по себе фреймы не отправляются в нужном порядке. Для контроля качества используется RTCP – расширение протокола RTP. В данной работе используется свой протокол взаимодействия.

На application layer уже добавляется дополнительный функционал, например шумоподавление.

Для шумоподавления часто используются различные методы цифровой обработки сигналов и изображений. Например, часто применяют быстрое преобразование Фурье для приведения к ряду Фурье, а далее гораздо проще анализировать диапазоны частот.

Также можно реализовывать шумоподавление с помощью наложения специально сгенерированного звука. Но часто просто можно избежать рекурсивного воспроизведения собственного микрофона через кольцевой буфер.

3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

В данном разделе описываются входные и выходные данные программы, диаграмма классов, а также приводится описание используемых классов и их методов.

3.1 Структура входных и выходных данных

Таблица 3.1 – файл с настройками соединения config.ini.

FPS	Размер одного чанка данных	Качество изображения
60	60416	90

Пример для таблицы 3.1:
[General]
fps=60
pack_size=60416
quality=90

3.2 Разработка диаграммы классов

Диаграмма классов для курсового проекта приведена в **Приложении А**.

3.3 Описание классов

3.3.1 Класс менеджера сессии

Класс `SessionManager` представляет текущую сессию и хранит все состояние для возможности завершения сессии и ее повторного запуска.

Поля класса:

- `UDPPlayer *player` – проигрыватель аудио по UDP.
- `ScreenRecorder *recorder` – запись экрана текущего устройства и отправка другому клиенту.
- `MyThread *listen_thread` – получение изображений от клиента, реконструкция и отрисовка на окне (отдельный поток).
- `MainWindow *window` – окно звонка. После завершения удаляется и создается новым на каждый звонок.
- `StartWindow &start_window` – окно подключения к клиенту, которое отображается до или после звонка.
- `SettingsWindow *settings_window` – окно настроек, создается лишь на время его вызова через кнопку.
- `GrabberSender *grabber` – захватчик событий ввода, который как клиент отправляет ивенты на сервер.

- `X11InputApplicant *inputApplicant` – применитель событий ввода, который работает как сервер и потребляет ивенты.
- `std::string ConnectServer` – сервер, к которому необходимо подключится.
- `bool is_control` –флаг, который показывает, является ли текущий сервер хост-сервером.

Методы:

- `SessionManager(StartWindow &start_window)` – принимает объект начального окна, и настраивает обработку окна настроек.
- `connectButtonClicked()` – вызывается Qt при нажатии кнопки соединения, он обновляет поля `ConnectServer` и `is_control`.
- `start()` – начинает сессию – загружает настройки из файла, запускает запись экрана (для управляемого сервера) и звука, запускает фоновые потоки получения и воспроизведения картинки (для хост-сервера) и звука. Также создает потоки захвата событий ввода (для хост-сервера) и применения событий (для управляемого сервера).
- `stop()` – ждет завершения потоков воспроизведение и очищает память (вызывается по кнопке End).

Класс `SessionManager` позволяет совершать неограниченное число соединений во время сессии нашей программы.

3.3.2 Класс записи экрана

Класс `ScreenRecorder` является потоком (`QThread`), который `FPS` раз в секунду делает снимки экрана и отправляет по сети (то есть передает видео с экрана)

Поля класса:

- `char* server` – сервер, на который отправлять видео.
- `int pack_size` – размер одного пакета.
- `int frame_interval` – промежуток, который необходимо ждать до отправки следующего фрейма.
- `int quality` – качество передаваемого изображения.

Методы:

- `ScreenRecorder(char* server, int pack_size, int frame_interval, int quality)` – конструктор инициализирует поля класса.
- `run()` – работает внутри другого потока и вызывается Qt. Он получает текущее окно, рисует курсор на скриншоте, преобразует `QPixMap` в `cv::Mat` и сжимает изображение для отправки, разбивает его на чанки и отправляет по сети. Метод работает до остановки (при окончании звонка) с промежутком `frame_interval`

Класс `ScreenRecorder` работает до тех пор, пока не запрошена остановка потока. С помощью сигнала Qt о завершении программы устанавливается специальный флаг, что позволяет потоку завершить работу

без неожиданной остановки. Поток выполняет свою последнюю итерацию цикла и выходит из него и из функции `run()`. Тем временем сигнал после метода `requestInterruption()` вызывает метод `wait()`, что позволяет дождаться завершения потока.

Это является стандартным подходом при работе с потоками, другим методом являлось бы завести очередь задач и помещать задачи в очередь, а поток бы обрабатывал их в порядке добавления.

3.3.3 Класс захвата входных событий

Класс `X11Grabber` реализует обобщенный интерфейс для парсинга событий мыши и клавиатуры для X window server. Он реализован используя `std::thread` а не `QThread`, так как он может использоваться в любых программах. Для отправки сообщений используются сокеты Linux (`sys/socket.h`), так как `QtNetwork` не всегда доставляет события корректно. В бесконечном цикле, используя `popen2()`, вызывается команда `xinput test-xi2 --root`, которая выводит информацию о любом событии ввода, производимом в системе. Извлекаются лишь нужные ивенты и отправляются в `callback`.

Поля класса:

- `std::function<void(const Event&)> callback` – функций, которая должна обрабатывать каждый новый ивент.
- `std::thread thread` – поток, в котором запущен цикл парсинга.
- `std::atomic<bool> running` – атомарная переменная, которая позволяет реализовать нормальное завершение работы.

Методы:

- `X11Grabber()` – конструирует объект, поток ещё не запущен.
- `start()` – запуск потока парсинга.
- `stop()` – остановка потока парсинга используя атомарную переменную.
- `set_callback(std::function<void(const Event&)> callback)` – установка обработчика новых ивентов.
- `loop()` – сам цикл парсинга.

Для отправки сообщений используются сокеты Linux (`sys/socket.h`), так как `QtNetwork` не всегда доставляет события корректно.

Класс `GrabberSender` является надстройкой над `X11Grabber`, реализуя отправку ивентов по сети.

Поля класса:

- `std::string host` – сервер, куда отправлять события.
- `struct timespec ts` – последнее время, когда было отправлено событие MouseMove.
- `struct sockaddr_in server_address` – сервер, куда отправлять события, подготовленный для работы с функциями socket().

- `int socket` – открытый сокет к серверу.

Методы:

- `GrabberSender()` – конструирует объект, поток ещё не запущен.
- `set_host(std::string host)` – установка адреса сервера, куда отправлять события.
- `start()` – подключение к серверу и запуск потока парсинга.
- `stop()` – остановка потока парсинга используя атомарную переменную и отключение от сервера.

3.3.4 Класс применения входных событий

Класс `X11InputApplicant` работает как сервер и применяет события к управляемой системе.

Поля класса:

- `int socket` – сокет, слушающий события от клиентов.
- `std::thread thread` – поток, в котором запущен сервер.
- `std::atomic<bool> running` – атомарный флаг, позволяющий реализовать нормальное завершение.

Методы:

- `X11InputApplicant()` – конструирует объект, поток ещё не запущен.
- `listen_loop()` – функция, которая запускает сервер и ждет клиентов.
- `start()` – запуск сервера.
- `stop()` – остановка сервера.
- `consume(const Event &event)` – применение очередного входного ивента.

Класс `X11InputApplicant` применяет ивенты используя `xdotool` – данная утилита позволяет контролировать мышь и клавиатуру системы. Она использует методы из `Xlib` и `XTest`.

3.3.5 Структуры данных фреймов

Структуры `FrameData` и `FrameChunk` помогают реконструировать изображения после получения по сети, вне зависимости от порядка присланных пакетов. `Event` это одно входное событие системы.

Структура `FrameData` имеет следующие поля:

- `int frame_num` – текущий номер фрейма (для отображение картинок в нужном порядке)
- `int buffer_size` – размер буфера, который нужно выделить под хранение одного изображения

Конструкторы `FrameData()` и `FrameData(int frame_num, int buffer_size)` инициализируют поля структуры значениями по умолчанию и заданными значениями соответственно.

Структура `FrameChunk` имеет следующие поля:

- `int seq` – номер чанка (чанки от 0 до N, объединенные по возрастанию seq дадут изображение)
- `int size` – размер чанка (все чанки равного размера кроме последнего)
- `uint8_t *data` – данные

Методы:

- `FrameChunk()` и `FrameChunk(int seq, int size, uint8_t *data)` – конструкторы инициализируют поля структуры значениями по умолчанию и заданными значениями соответственно.
- `FrameChunk(const FrameChunk &other)` – конструктор копирования позволяет избежать ошибочного освобождения памяти при передачи чанков в функцию
- `~FrameChunk()` – декструктор освобождает данные
- `operator<` – сравнивает два чанка по sequence (так как нам необходимо восстанавливать изображение по возрастанию)

Структура `Event` имеет следующие поля:

- `Mode mode` – режим ивента (KeyDown, KeyUp, MouseDown, MouseUp, MouseMove)
- `int code` – код действия (например, левая или правая кнопка мыши)
- `int x` – позиция x мыши
- `int y` – позиция y мыши

Свободные функции:

- `bool write(int socket, Event req)` – отправка ивента по сокету. Возвращает true, если запись прошла успешно.
- `bool read(int socket, Event *buf)` – запись ивента по сокету. Возвращает true, если чтение прошло успешно.

3.3.6 Класс воспроизведения звука

Класс `UDPPlayer` получает аудио по UDP и воспроизводит в системный вывод звука

Он содержит следующие поля:

- `QAudioOutput *output` – системный вывод звука;
- `QUdpSocket *socket` – подключение по UDP для получения звука;
- `QIODevice *device` – абстракция Qt для соединения QAudioOutput и QUdpSocket.

Методы:

- `UDPPlayer()` – конструктор инициализирует аудиовыход и подключает функцию обработки данных.
- `playData()` – читает UDP датаграму и записывает на аудио устройство

3.3.7 Класс обработки изображений

Класс `MyThread` получает фреймы по чанкам и отображает фреймы на экран

Методы:

- `run()` – работает в другом потоке. Он получает данные и конструирует изображение из чанков.
- `terminateThread()` – запрашивает остановку потока и ждет, пока поток сам себя завершит.

3.3.8 Классы главных оконных интерфейсов

Класс `MainWindow` содержит окно звонка

Поля класса:

- `QPixmap mainimg` – текущее изображение, полученное от клиента
- `QAudioInput *audio_input` – ввод с микрофона пользователя
- `QUdpSocket *audio_socket` – сокет для отправки аудио по UDP
- `bool mic_enabled` – показывает, включен ли микрофон

Методы:

- `MainWindow(QWidget *parent=nullptr)` – конструктор соединяет нажатие кнопки микрофона с методом toggleMic
- `init_audio_input(char *server)` – подключается к серверу и подсоединяет ввод с микрофона к отправке аудио по UDP.
- `start_audio()` – начинает запись с микрофона.
- `stop_audio()` – приостанавливает запись с микрофона. Используется для кнопки включения и выключения микрофона.
- `deinit_audio_input()` – останавливает запись и очищает память и исходящие соединения. Используется при завершении звонка
- `processImage(const QImage &img)` – устанавливает новое изображение, полученное по сети
- `toggleMic()` – меняет иконку кнопки и обновляет состояние записи с микрофона

Класс `SettingsWindow` содержит окно для редактирования настроек в файле настроек config.ini

Методы:

- `SettingsWindow(QWidget *parent=nullptr)` – конструктор загружает настройки из файла и устанавливает начальные значения
- `saveSettings()` – сохраняет новые введенные значения в файл

4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

4.1 Разработка алгоритмов

void MyThread::run() – функция предназначена для получения и отображения потока изображений на экран.

Алгоритм по шагам:

1. Начало.
2. Запускаем слушатель пакетов на определенном порту.
3. Пока не запрошена остановка потока, переходим к шагу 4, иначе к шагу 14.
4. Получаем входящий фрейм (с таймаутом ожидания).
5. Извлекаем параметры из фрейма: текущий фрейм, номер чанка и данные.
6. Сохраняем новый чанк в std::map чанков для текущего фрейма.
7. Если есть чанк с номером 0, перейти к пункту 8. Иначе к пункту 9.
8. Выделяем буфер размера, извлеченного из нулевого чанка, и удаляем нулевой чанк.
9. Если буфер для текущего фрейма выделен, то переходим к пункту 10. Иначе к пункту 13
10. Проходим по массиву чанков для текущего фрейма и копируем во временный буфер.
11. Если размер временного буфера совпадает с ожидаемым, то переходим к пункту 12. Иначе к пункту 13.
12. Создаем изображение из буфера, меняем его под размер окна и отправляем на обработку главному окну.
13. Очищаем полученный фрейм и переходим к шагу 3
14. Конец.

void X11Grabber::loop() – функция предназначена для захвата входных событий и вызова callback.

Алгоритм по шагам:

1. Начало.
2. Запускаем команду `xinput test-xi2 --root` в подпроцессе . Если все прошло успешно, переходим на шаг 4. Иначе переходим на шаг 3.
3. Выводим сообщение об ошибке и выходим из функции.
4. Устанавливаем текущий режим ивента в пустой ивент (std::nullopt).
5. Пока не запрошена остановка потока и вывод команды не закончился, переходим к шагу 6, иначе к шагу 15.
6. Выделяем буфер размера BUFFER_SIZE.
7. Читаем максимум BUFFER_SIZE байт в буфер из вывода команды
8. Разбиваем буфер по символу перехода на новую строку (\n)
9. Проходимся по каждой строке вывода

10. Если текущий режим пуст, переходим к шагу 11, иначе к шагу 12

11. Если текущая строка начинается с одного из поддерживаемых типов ивентов, то устанавливаем соответствующий режим и переходим к шагу 9, иначе к шагу 12

12. Создаем пустой ивент

13. В зависимости от текущего режима, вызываем parse_keyboard или parse_click, обновляя значение режима.

14. Если ивент не пустой, вызываем callback, передавая ивент. Переходим к шагу 9.

15. Завершаем процесс xinput через сигнал SIGTERM и закрываем pipe.

16. Конец.

MyThread::run() является основной частью кода изменения изображения. Так как количество данных, что можно передать за раз ограничено протоколом UDP, мы сжимаем изображение и отправляем его по частям. Для обеспечения надежности доставки пакетов полученные чанки сортируются по своему номеру.

X11Grabber::loop() использует платформо-зависимую команду xinput, а далее используя функции парсинга строк получает нужные данные. Важно не забыть остановить процесс xinput по завершении программы, так как он работает бесконечно, а простой вызов pclose блокируется до завершения подпроцесса.

При разработке алгоритмов были использованы библиотеки OpenCV, uvgrtp, xinput и Qt.

Данные алгоритмы являются основными во всем приложении. В будущем их можно оформить как интерфейсы и использовать для реализации любых других функций.

При разработке функций часто бывает полезно сначала продумать общий алгоритм действий, составить схему алгоритма, а потом реализация становится очень простой.

При разработке алгоритма по шагам и схемы алгоритма важно не включать детали реализации, которые никак не помогают понять суть алгоритма. Например, какие-то специфические особенности разных операционных систем, языков программирования и не только вряд ли будут указаны на схеме алгоритма. Схему алгоритма можно описать простыми словами, иногда с указанием сторонних функций.

В данном случае не были включены особенности того, как получить скриншот на разных операционных системах, как получить данные по сети и другие важные детали реализации.

Структурная схема программы представлена в приложении Б.

6 РЕЗУЛЬТАТЫ РАБОТЫ

При запуске программы нас приветствует окно подключения к серверу (рисунок 6.1). Можно ввести адрес 127.0.0.1 для подключения к самому себе (очень похоже на обычную запись экрана через программу OBS). При нажатии Connect происходит переход на окно сессии. Необходимо выбрать, является текущий сервер хост-машиной, или управляемой машиной.

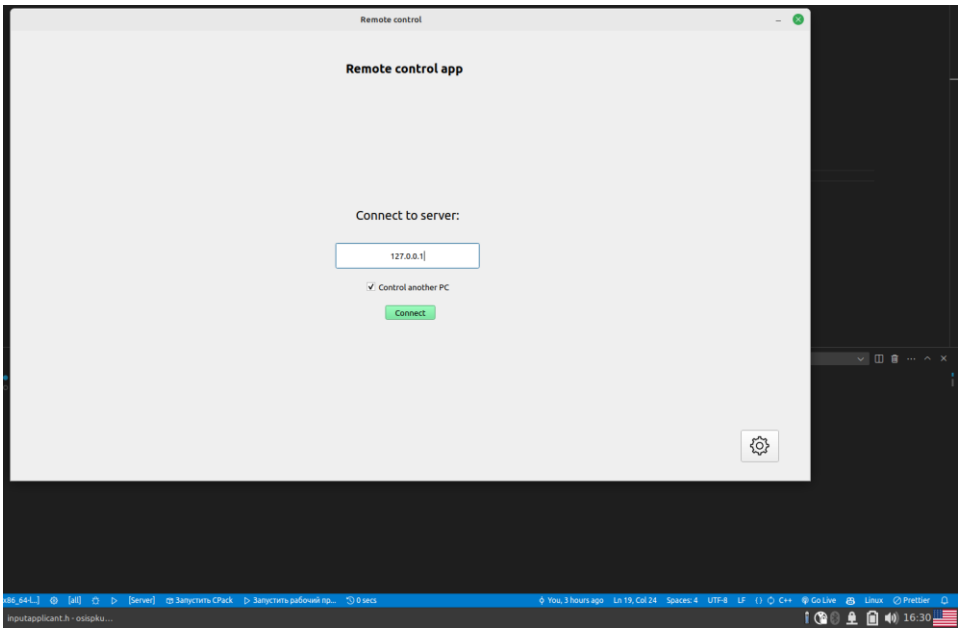

Рисунок 6.1 – Стартовое окно

Как показано на рисунке 6.2, окно настроек позволяет настроить основные параметры соединения: fps, размер пакета и качество изображения.

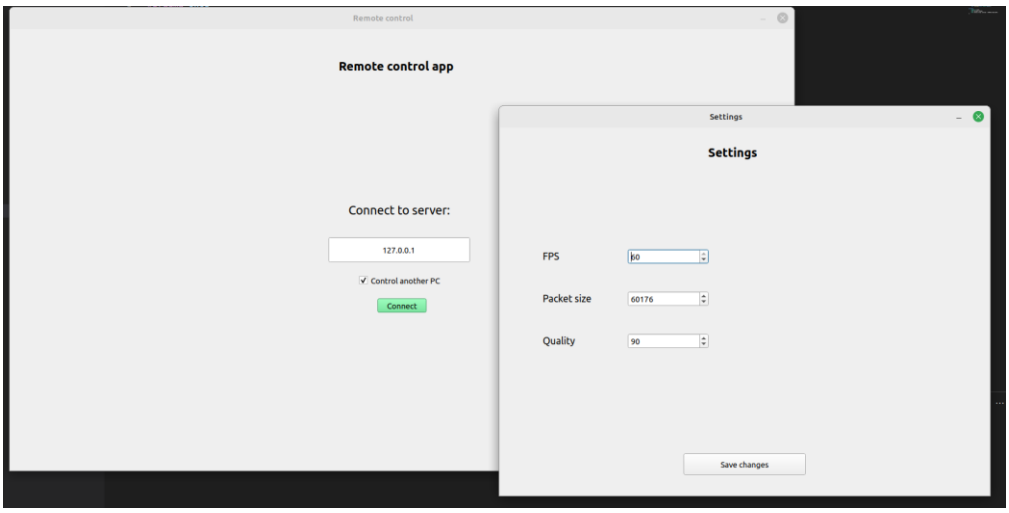

Рисунок 6.2 – Окно настроек

Настройки сохраняются в файл config.ini. Перед открытием диалога загружаются актуальные настройки. .ini файл был выбрал, так как это удобный читабельный формат и для пользователя, и для компьютера.

При подключении к серверу пользователь увидит окно, отображенное на рисунке 6.3.

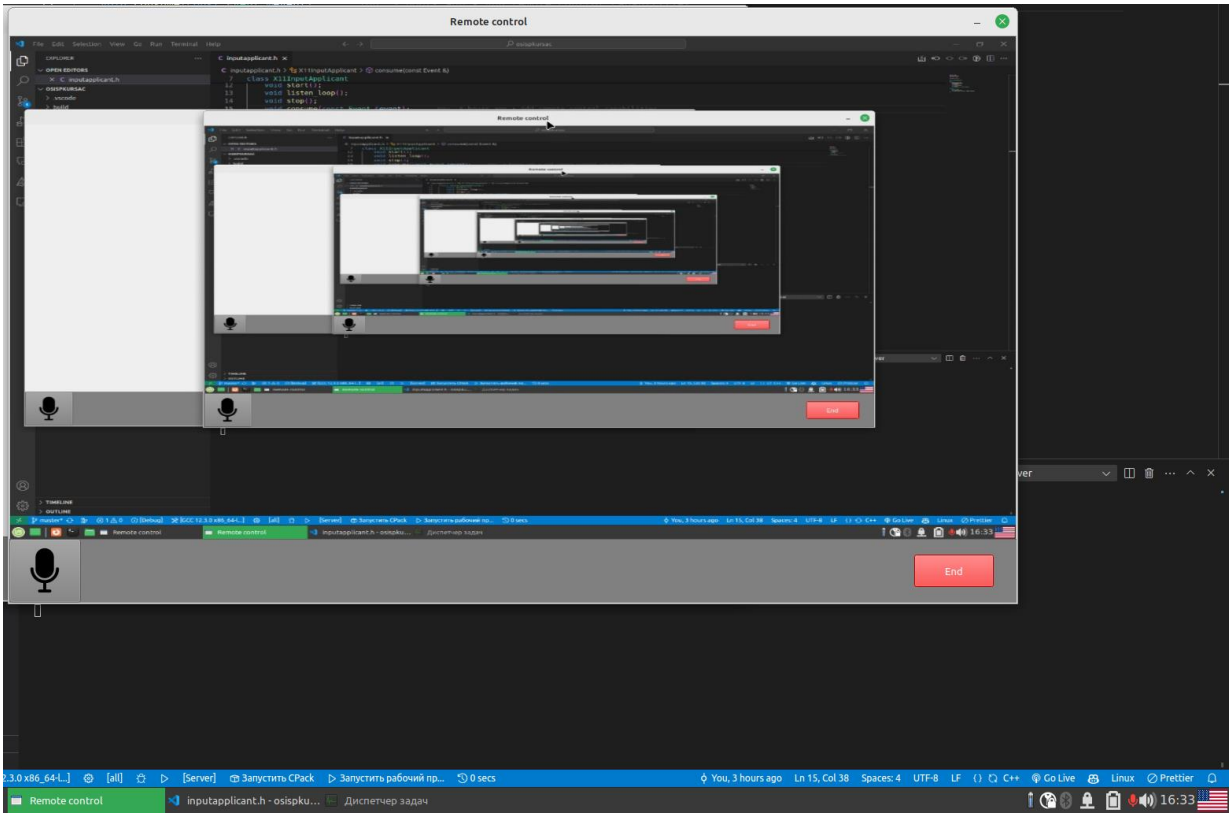

Рисунок 6.3 – Окно звонка (микрофон включен)

Интерфейс похож на интерфейс Zoom при звонке с 2 участниками: вместо сетки с видео участников, видно только видео противоположного участника. На панели управления можно завершить диалог и менять состояние микрофона. Как можно наблюдать в нижнем правом углу экрана, операционная система успешно отображает то, что происходит запись с микрофона. Так как в примере мы подключились сами к себе, можно наблюдать знаменитое рекурсивное отображение текущего экрана.

Аналогичную картинку можно было было бы наблюдать, если бы мы запустили запись в OBS. Это никак не влияет на работоспособность программы, но можно наблюдать за тем, как при движении мышки постепенно это отображается во всех окнах. Рекурсия бесконечная.

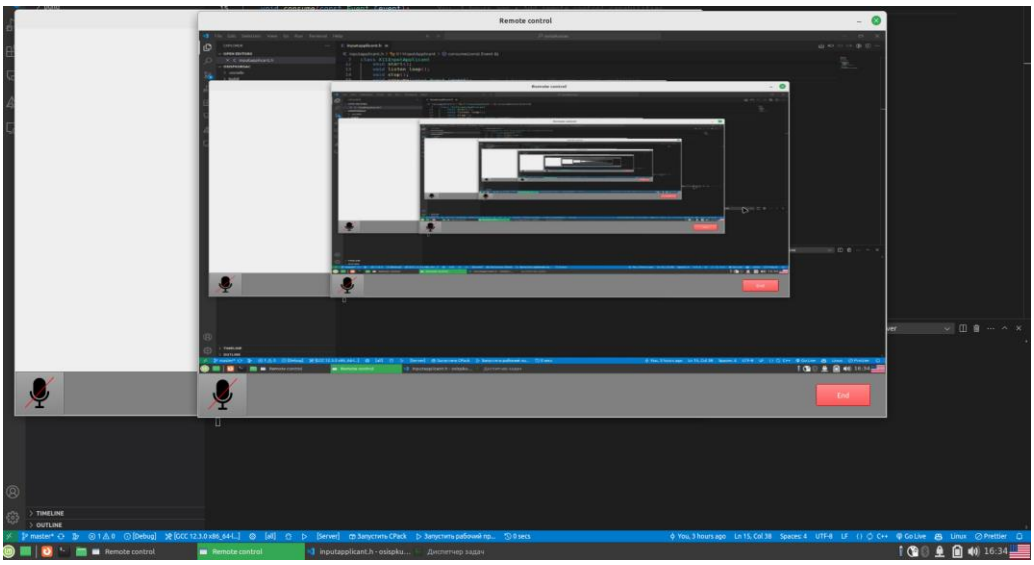

Рисунок 6.4 – Окно звонка (микрофон выключен)

Как можно наблюдать на рисунке 6.4, при выключении микрофона успешно обновляется иконка в приложении, и операционная система больше не отображает то, что какое-то приложение записывает микрофон.

Приложение было успешно протестировано на нескольких вариантах локальных сетей. Для тестирования из меню роутера узнавался ip адрес клиентов и вводился в поле ввода. Даже на расстоянии 15-20 метров между клиентами (при силе сигнала всего в 26% на одном из клиентов) с небольшими задержками можно было продолжать сессию. С экрана можно было успешно прочитать текст.

Приложение ограничено возможностями роутера по передачи пакетов. Из-за того, что абсолютно все сырые данные о фреймах должны быть переданы, происходит довольно сильная нагрузка на сеть. Это один из минусов P2P архитектуры. Но зато можно запустить сессию, не имея никаких серверов, а имея только 2 клиента, которые хотят созвонится.

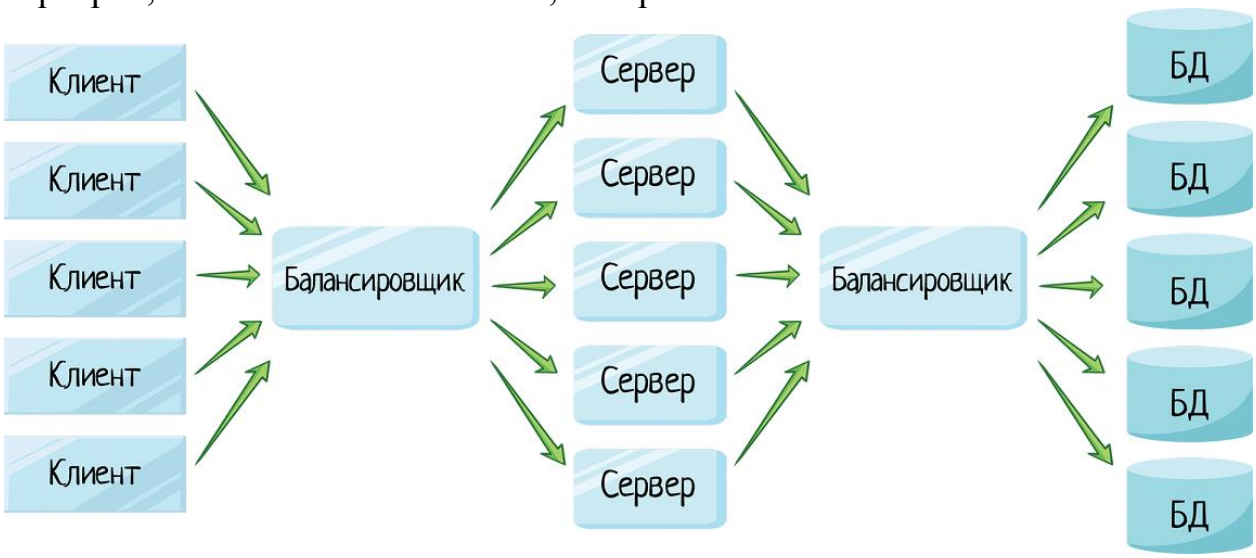

Рисунок 6.5 – Клиент-серверная архитектура

Теоретически можно было бы установить приложение на сервер как 1 клиент, а вторым клиентом было бы обычное потребительское устройство. Но на серверах нет микрофона и графического сервера в принципе. Его конечно можно установить, но обычно это не рекомендуется. Тем более что запись с микрофона в принципе невозможна, так как сервера в облаке обычно виртуальные и запущены вместе с тысячами себе подобных. Если бы это было бы возможно, то это являлось бы дырой безопасности в средствах виртуализации, которые используются на серверах (обычно QEMU или KVM)

Обычно в приложениях для удаленного доступа, которые выдерживают большую нагрузку используется архитектура такая же, как указана на рисунке 6.5.

Множество клиентов подключается к балансировщику, который обычно 1 (обычно на 1 порте может работать лишь 1 сервис, а клиентам нужен фиксированный адрес сервера для подключения). После этого в зависимости от региона пользователя, загруженности серверов и других факторов запрос направляется на нужный сервер. Каждый сервер является полной копией всех остальных по реализации.

Причем часто доступ к базе данных тоже помещен за балансировщик, и база данных имеет много реплик.

6.1 Руководство пользователя

Программа была разработана и протестирована на операционной системе Linux. На операционной системе Windows программа скомпилировалась с небольшими изменениями, однако столкнулась с проблемами при приеме входящих данных, которые, вероятно, были связаны с брандмауэром или другими политиками безопасности Windows.

Разработка программы велась в среде Visual Studio Code с использованием расширения CMake Tools. Это расширение предоставило удобный, быстрый и эффективный инструментарий для работы со сложными проектами на языке C++. На рисунке 6.6 показано, как с помощью нажатия одной кнопки запуска автоматически производится генерация зависимостей проекта, сборка необходимых файлов и линковка их в один бинарный исполняемый файл.

Важно отметить, что процесс разработки и отладки программы в Linux обладает рядом преимуществ по сравнению с Windows. Операционная система Linux предоставляет более гибкие и расширенные возможности для работы с сетевыми сокетами и низкоуровневыми системными вызовами, что упрощает разработку сетевых приложений и обеспечивает лучшую совместимость с различными протоколами и стандартами.

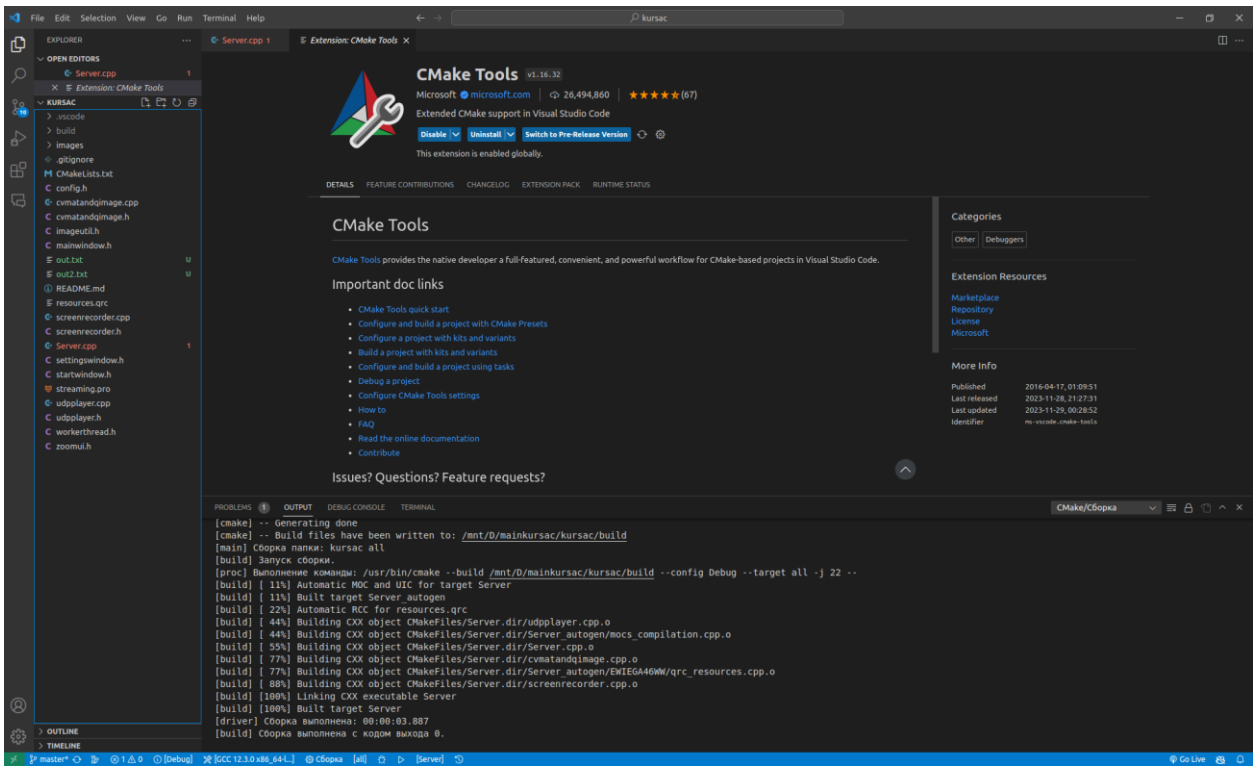

Рисунок 6.6 – Сборка в одну кнопку в VS Code

Проект можно собрать как с помощью cmake, так и qmake. Начиная с версии Qt 6 предпочтительней использовать CMake из-за его универсальности и нативной интеграции со всем функционалом Qt.

В данном проекте использовалась Qt 5 за счет большей доступности документации и самих версий Qt.

Сначала необходимо установить используемые библиотеки. Подразумевается, что Qt уже установлен с официального сайта. Здесь и далее инструкции для Ubuntu 22.04

```
sudo apt update
sudo apt install ccache libopencv-dev python3-opencv cmake git libx11-dev libxfixes-dev libgl-dev libxext-dev xdotool
git clone https://github.com/ultravideo/uvgRTP
cd uvgRTP
mkdir build && cd build
cmake -DUVGRTP_DISABLE_CRYPTO=1 ..
make
sudo make install
```

Данные команды установят OpenCV, средство сборки CMake, а также некоторые дополнительные библиотеки, которые помогают взять скриншот экрана. Эта функция платформо-зависимая, так как Qt не предоставляет возможности получить скриншот экрана (это есть в Qt 6, но курсор мыши не виден)

Библиотека uvgrtp [5] используется для упрощения работы с протоколом RTP. Поверх этого строится свой протокол взаимодействия. На рисунке 6.7 можно наблюдать успешный процесс сборки библиотеки под OS Linux.

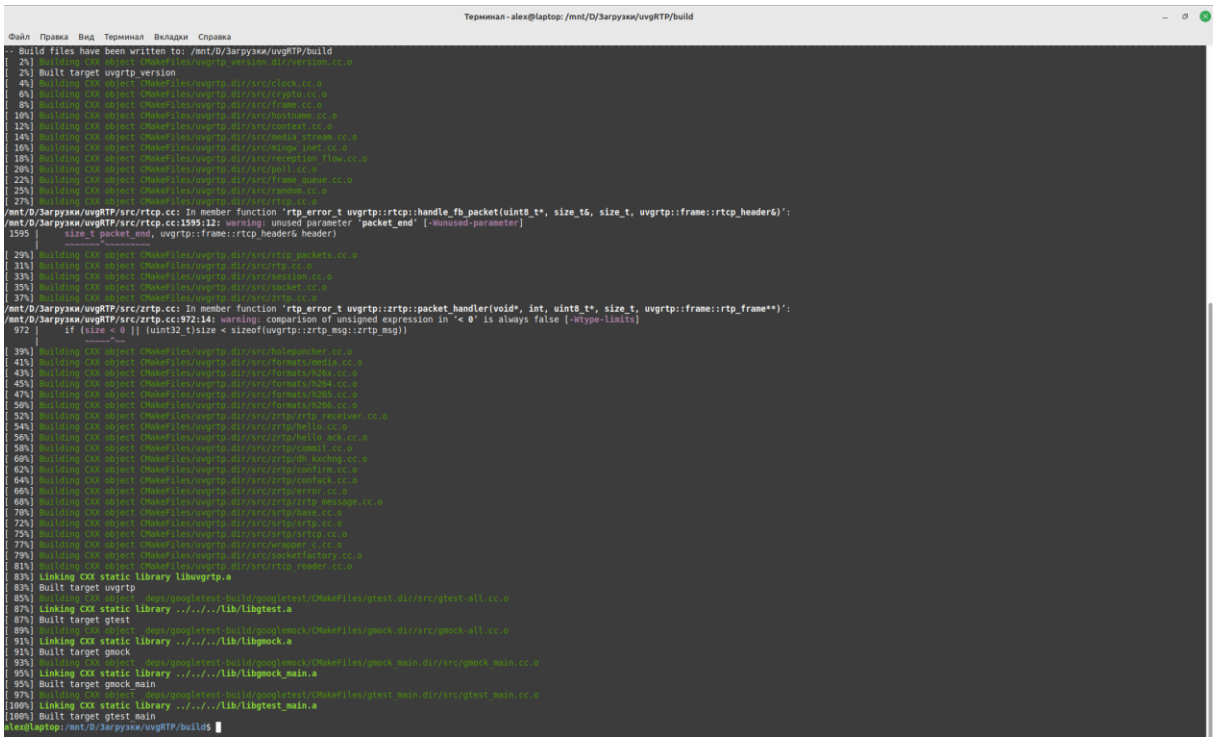

Рисунок 6.7 – сборка библиотеки uvgrtp через CMake

Далее сборка производится через команды `qmake && make`, или `cmake . && make`. Утилита `xdotool` использовалась для применения событий ввода.

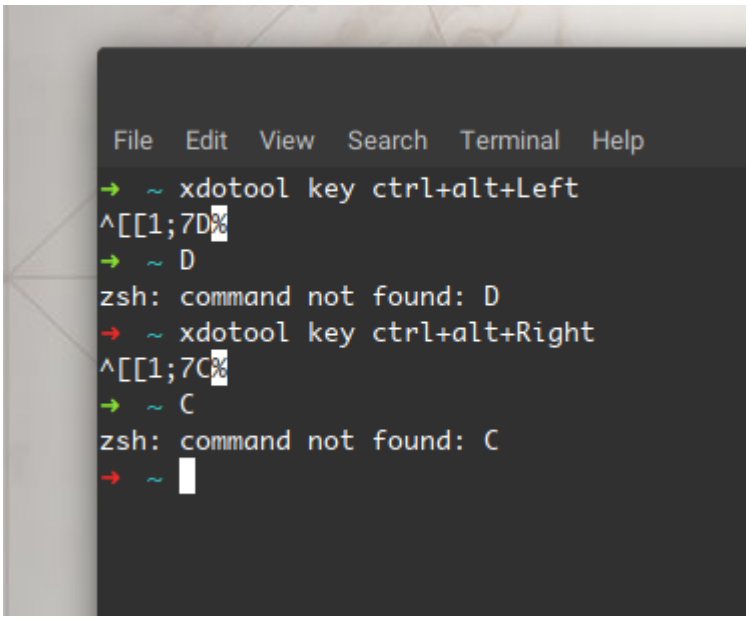

Рисунок 6.8 – утилита xdotool

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы было создано приложение для удаленного управления компьютером. Для этого использовалось Qt для графического интерфейса, OpenCV для обработки изображений и протоколы RTP, UDP и TCP.

В результате приложение предоставляет возможность соединять две 2 машины в локальной сети. Качество изображения и звука сохранялось на удовлетворительном уровне даже через несколько стен и других преград для сигнала. В будущем возможно развить приложение до возможности сохранения сессий и быстрого подключения к одному из установленных компьютеров, а также возможность управляемого компьютера не вводить ip адрес хост-компьютера.

Приложения для удаленного управления компьютером являются ключевым компонентом в процессах работы современных команд. Это позволило создавать полностью удаленные компании без офисов и расширило возможности для глобальной кооперации. За счет реализации записи микрофона теперь не требуется одновременно созваниваться через мессенджер и подключаться к компьютеру отдельно – теперь все в одном приложении.

Данная работа показала, что написать собственное приложение для удаленного доступа не так сложно, как может показаться на первый взгляд. Самыми сложными частями проекта было разбитие фреймов на чанки для обеспечения их доставки в нужном порядке (иначе данные перетирались и буфер переполнялся), а также реализовать быструю реконструкцию изображения и отправку его в пользовательский интерфейс. Для этого пришлось преобразовывать типы Qt QPixmap в типы OpenCV cv::Mat и наоборот. Ещё одной сложностью было взаимодействие Qt с TCP сокетами, когда данные иногда терялись после отправки. За счет использования сокетов Linux данную проблему получилось избежать.

Еще одной важной оптимизацией было бы улучшение отправки изображений: вместо того, чтобы FPS раз в секунду брать снимок экрана, сжимать его и отправлять и сети, можно было бы использовать готовые кодеки, такие как h264, которые были созданы для эффективного стриминга по сети.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Stroustrup, B. The C++ Programming Language/ Bjarne Stroustrup. – 1985. – 504 p.

[2] Socket documentation [Электронный ресурс]. – Электронные данные. – Режим доступа: https://linux.die.net/man/2/socket. – Дата доступа: 01.05.2024.

[3] Qt 5 Documentation [Электронный ресурс]. – Режим доступа: https://doc.qt.io/qt5 – Дата доступа: 01.05.2024.

[4] OpenCV Documentation [Электронный ресурс]. – Режим доступа: https://docs.opencv.org/4.x/. – Дата доступа: 01.05.2024.

[5] uvgRTP Documentation [Электронный ресурс]. – Режим доступа: https://github.com/ultravideo/uvgRTP. – Дата доступа: 01.05.2024.

ПРИЛОЖЕНИЕ А
(обязательное)

Диаграмма классов

ПРИЛОЖЕНИЕ Б
(обязательное)

Схема структурная

ПРИЛОЖЕНИЕ В
(обязательное)

Полный код программы

Файл config.h:

```
#ifndef CONFIG_H
#define CONFIG_H

#define FRAME_HEIGHT 720  // for transfer
#define FRAME_WIDTH 1080  // for transfer
#define FPS 60            // fps
#define PACK_SIZE 60176   // < max UDP packet size
#define ENCODE_QUALITY 90 // larger=more quality but large packet sizes

#define IMAGE_UDP_PORT 50000
#define AUDIO_UDP_PORT 55455
#define IO_TCP_PORT 50001
#define MOUSEMOVE_DELAY_NS 10000000

#define SETTINGS_FILE "config.ini"
#endif
```

Файл event.h:

```
#pragma once

#include <iostream>
#include <sys/types.h>
#include <sys/socket.h>

enum class Mode
{
    KeyDown,
    KeyUp,
    MouseDown,
    MouseUp,
    MouseMove
};

inline std::string get_mode_value(const Mode &m)
{
    switch (m)
    {
    case Mode::KeyDown:
        return "KeyDown";
    case Mode::KeyUp:
        return "KeyUp";
    case Mode::MouseDown:
        return "MouseDown";
    case Mode::MouseUp:
        return "MouseUp";
    case Mode::MouseMove:
        return "MouseMove";
    }
    return "Unknown";
}

inline std::ostream &operator<<(std::ostream &out, const Mode value)
```

```
    {
        return out << get_mode_value(value);
    }

    struct Event
    {
        Mode mode;
        int code;
        int x;
        int y;
    };

    inline std::ostream &operator<<(std::ostream &os, const Event &event)
    {
        return os << "Mode: " << event.mode << " Code: " << event.code << "
X: " << event.x << " Y: " << event.y;
    }

    inline bool write(int socket, Event req)
    {
        auto bytes_sent = ::send(socket, &req, sizeof(Event), 0);
        return bytes_sent == sizeof(Event);
    }

    inline bool read(int socket, Event *buf)
    {
        auto bytes_read = ::read(socket, buf, sizeof(Event));
        return bytes_read == sizeof(Event);
    }
```

Файл grabber.cpp:

```
    #include "utils.h"
    #include "grabber.h"
    #include "config.h"
    #include <optional>
    #include <sys/types.h>
    #include <signal.h>
    #include "popen2.h"

    std::pair<int, int> get_current_mouse_location()
    {
        auto output = exec("xdotool getmouselocation");
        auto parts = Split(output, " ");
        if (parts.size() < 2)
            return {0, 0};
        auto x = std::stoi(parts[0].substr(2));
        auto y = std::stoi(parts[1].substr(2));
        return {x, y};
    }

    std::optional<Mode>     parse_keyboard(const     std::string     &line,
std::optional<Mode> mode, std::optional<Event> &event)
    {
        if (line.starts_with("    detail: "))
        {
            auto num = std::stoi(line.substr(12));
            event.emplace(Event{mode.value(), num, 0, 0});
            return std::nullopt;
        }
```

```
        return mode;
    }

    std::optional<Mode> parse_click(const std::string &line,
std::optional<Mode> mode, std::optional<Event> &event)
    {
        if (line.starts_with("    detail: "))
        {
            auto code = 0;
            if (line.size() > 12)
                code = std::stoll(line.substr(12));
            auto coords = get_current_mouse_location();
            event.emplace(Event{mode.value(), code, coords.first,
coords.second});
            return std::nullopt;
        }
        return mode;
    }

    X11Grabber::~X11Grabber()
    {
        X11Grabber::stop();
    }

    void X11Grabber::start()
    {
        if (running.load())
            return;
        running.store(true);
        thread = std::thread(&X11Grabber::loop, this);
    }

    void X11Grabber::stop()
    {
        if (!running.load())
            return;
        running.store(false);
        thread.join();
    }

    void X11Grabber::set_callback(std::function<void(const Event &)>
callback)
    {
        this->callback = callback;
    }

    void X11Grabber::loop()
    {
        pid_t child_pid;
        auto input = popen2("xinput test-xi2 --root", "r", &child_pid);
        if (!input)
        {
            fprintf(stderr,
                    "incorrect parameters or too many files.\n");
            return;
        }
        std::optional<Mode> mode = std::nullopt;
        while (running.load() && !std::feof(input))
        {
            char buffer[BUFFER_SIZE] = {0};
```

```
        if (std::fgets(buffer, BUFFER_SIZE, input) != NULL)
        {
            std::string lines = buffer;
            for (const auto &line : Split(lines, "\n"))
            {
                if (!mode.has_value())
                {
                    if (line.starts_with("EVENT type 2"))
                        mode = Mode::KeyDown;
                    else if (line.starts_with("EVENT type 3"))
                        mode = Mode::KeyUp;
                    else if (line.starts_with("EVENT type 4"))
                        mode = Mode::MouseDown;
                    else if (line.starts_with("EVENT type 5"))
                        mode = Mode::MouseUp;
                    else if (line.starts_with("EVENT type 15"))
                        mode = Mode::MouseDown;
                    else if (line.starts_with("EVENT type 16"))
                        mode = Mode::MouseUp;
                    else if (line.starts_with("EVENT type 17"))
                        mode = Mode::MouseMove;
                    continue;
                }
                std::optional<Event> event = std::nullopt;
                switch (mode.value())
                {
                case Mode::KeyDown:
                    mode = parse_keyboard(line, Mode::KeyDown, event);
                    break;
                case Mode::KeyUp:
                    mode = parse_keyboard(line, Mode::KeyUp, event);
                    break;
                case Mode::MouseDown:
                    mode = parse_click(line, Mode::MouseDown, event);
                    break;
                case Mode::MouseUp:
                    mode = parse_click(line, Mode::MouseUp, event);
                    break;
                case Mode::MouseMove:
                    mode = parse_click(line, Mode::MouseMove, event);
                    break;
                }
                if (event.has_value())
                    callback(event.value());
            }
        }
    }
    kill(child_pid, SIGTERM);
    pclose(input);
}

void GrabberSender::set_host(std::string host)
{
    this->host = host;
    server_address.sin_family = AF_INET;
    server_address.sin_addr.s_addr = inet_addr(this->host.c_str());
    server_address.sin_port = htons(IO_TCP_PORT);
}

void GrabberSender::start()
{
```

```
        socket = ::socket(AF_INET, SOCK_STREAM, 0);
        if (socket == -1)
        {
            std::cerr << "Failed to create socket" << std::endl;
            return;
        }
        if    (::connect(socket,    (struct    sockaddr    *)&server_address,
sizeof(struct sockaddr_in)) == -1)
        {
            ::close(socket);
            std::cerr << "Failed to connect to server" << std::endl;
            return;
        }
        set_callback([this](const Event &event)
                     {
            if (event.mode == Mode::MouseMove) {
                struct timespec now;
                clock_gettime(CLOCK_MONOTONIC, &now);
                if (now.tv_sec - ts.tv_sec < 1 && now.tv_nsec - ts.tv_nsec <
MOUSEMOVE_DELAY_NS)
                    return;
                ts = now;
            }
            write(socket, event); });
        X11Grabber::start();
    }

    void GrabberSender::stop()
    {
        X11Grabber::stop();
        ::close(socket);
    }

    GrabberSender::~GrabberSender()
    {
        GrabberSender::stop();
    }
```

Файл grabber.h:

```
    #pragma once

    #include <string>
    #include <utility>
    #include <optional>
    #include <functional>
    #include <thread>
    #include <arpa/inet.h>

    #include "event.h"

    #define BUFFER_SIZE 1024 * 1024

    std::pair<int, int> get_current_mouse_location();

    std::optional<Mode>     parse_keyboard(const     std::string     &line,
std::optional<Mode> mode, std::optional<Event> &event);
```

```
std::optional<Mode> parse_click(const std::string &line,
std::optional<Mode> mode, std::optional<Event> &event);

class X11Grabber
{
public:
    X11Grabber() = default;
    virtual ~X11Grabber();
    virtual void start();
    virtual void stop();
    void set_callback(std::function<void(const Event &)> callback);

private:
    std::function<void(const Event &)> callback;
    std::thread thread;
    std::atomic<bool> running = false;
    void loop();
};

class GrabberSender : public X11Grabber
{

public:
    GrabberSender() = default;
    void set_host(std::string host);
    void start() override;
    void stop() override;

    ~GrabberSender() override;

private:
    std::string host;
    struct timespec ts = {0, 0};
    struct sockaddr_in server_address;
    int socket;
};
```

Файл inputapplicant.cpp:

```
#include "inputapplicant.h"
#include "utils.h"
#include "config.h"
#include <sys/socket.h>
#include <sys/types.h>
#include <arpa/inet.h>
#include <thread>
#include <cstring>
#include <signal.h>

static std::atomic_flag alarmed = ATOMIC_FLAG_INIT;

void handle_alarm(int)
{
    alarmed.test_and_set();
    return;
}

X11InputApplicant::X11InputApplicant()
```

```
    {
        sigemptyset(&mask);
        sigaddset(&mask, SIGALRM);
        if (pthread_sigmask(SIG_BLOCK, &mask, nullptr) < 0)
        {
            std::cerr << "Failed to block signal" << std::endl;
            return;
        }
        struct sigaction sigbreak;
        std::memset(&sigbreak, 0, sizeof sigbreak);
        sigbreak.sa_handler = &handle_alarm;
        if (sigaction(SIGALRM, &sigbreak, NULL) != 0)
        {
            std::cerr << "Failed to set signal handler" << std::endl;
            return;
        }
    }

    X11InputApplicant::~X11InputApplicant()
    {
        stop();
    }

    void X11InputApplicant::start()
    {
        if (running.test())
            return;
        running.test_and_set();
        thread = std::thread(&X11InputApplicant::listen_loop, this);
    }

    void X11InputApplicant::stop()
    {
        if (!running.test())
            return;
        running.clear();
        thread.join();
    }

    void X11InputApplicant::listen_loop()
    {
        if (pthread_sigmask(SIG_UNBLOCK, &mask, nullptr) < 0)
        {
            std::cerr << "Failed to unblock signal" << std::endl;
            return;
        }
        int listen_socket = socket(AF_INET, SOCK_STREAM, 0);
        if (listen_socket == -1)
        {
            std::cerr << "Failed to create socket" << std::endl;
            return;
        }
        struct sockaddr_in server_address;
        server_address.sin_family = AF_INET;
        server_address.sin_addr.s_addr = INADDR_ANY;
        server_address.sin_port = htons(IO_TCP_PORT);
        if   (bind(listen_socket,   (struct   sockaddr   *)&server_address,
sizeof(server_address)) == -1)
        {
            std::cerr << "Failed to bind socket" << std::endl;
```

```
            ::close(listen_socket);
            return;
        }
        if (listen(listen_socket, 1) == -1)
        {
            std::cerr << "Failed to listen on socket" << std::endl;
            ::close(listen_socket);
            return;
        }
        std::cout << "Listening for incoming connections..." << std::endl;
        while (running.test())
        {
            if (alarmed.test())
                return;
            struct sockaddr_in client_address;
            socklen_t client_addr_length = sizeof(client_address);
            alarm(5);
            int clientSocket = accept(listen_socket, (struct sockaddr
*)&client_address, &client_addr_length);
            if (alarmed.test())
                return;
            if (clientSocket == -1)
            {
                std::cerr << "Failed to accept connection" << std::endl;
                continue;
            }
            alarm(0);
            while (running.test())
            {
                std::cout << "Reading" << std::endl;
                Event event;
                if (!read(clientSocket, &event))
                    break;
                consume(event);
            }
            ::close(clientSocket);
        }
        ::close(listen_socket);
    }

    void X11InputApplicant::consume(const Event &event)
    {
        switch (event.mode)
        {
        case Mode::KeyDown:
            exec("xdotool keydown " + std::to_string(event.code));
            break;
        case Mode::KeyUp:
            exec("xdotool keyup " + std::to_string(event.code));
            break;
        case Mode::MouseDown:
            exec("xdotool mousemove " + std::to_string(event.x) + " " +
std::to_string(event.y) + " mousedown " + std::to_string(event.code));
            break;
        case Mode::MouseUp:
            exec("xdotool mousemove " + std::to_string(event.x) + " " +
std::to_string(event.y) + " mouseup " + std::to_string(event.code));
            break;
        case Mode::MouseMove:
            exec("xdotool mousemove " + std::to_string(event.x) + " " +
std::to_string(event.y));
            break;
```

```
    }
}
```

Файл inputapplicant.h:

```
#pragma once
#include <thread>
#include <atomic>

#include "event.h"

class X11InputApplicant
{
public:
    X11InputApplicant();
    ~X11InputApplicant();
    void start();
    void listen_loop();
    void stop();
    void consume(const Event &event);

private:
    int socket;
    std::thread thread;
    std::atomic<bool> running = false;
};
```

Файл mainwindow.h:

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H
#include <QMainWindow>
#include <QPixmap>
#include <QAudioInput>
#include <QUdpSocket>

#include "zoomui.h"
#include "udpplayer.h"

class MainWindow : public QMainWindow, public Ui::MainWindow
{
    Q_OBJECT
private:
    QPixmap mainimg;
    QAudioInput *audio_input;
    QUdpSocket *audio_socket;
    bool mic_enabled;

public:
    MainWindow(QWidget *parent = nullptr)
        : QMainWindow(parent), mic_enabled(true)
    {
        setupUi(this);
        connect(micButton, SIGNAL(clicked()), this, SLOT(toggleMic()));
    }
    void init_audio_input(char *server)
    {
        QAudioFormat format = getAudioFormat();
        audio_input = new QAudioInput(format);
        audio_socket = new QUdpSocket();
```

```
            audio_socket->connectToHost(server, AUDIO_UDP_PORT);
            audio_socket->waitForConnected();
            start_audio();
        }
        void start_audio()
        {
            audio_input->start(audio_socket);
        }
        void stop_audio()
        {
            audio_input->stop();
        }
        void deinit_audio_input()
        {
            stop_audio();
            audio_socket->deleteLater();
            audio_input->deleteLater();
        }

    public slots:
        void processImage(const QImage &img)
        {
            imgpix = QPixmap::fromImage(img);
            pixmap->setPixmap(imgpix);
        }
        void toggleMic()
        {
            mic_enabled = !mic_enabled;
            QImage  img(mic_enabled  ==  true  ?  ":/mic-on.png"  :  ":/mic-
off.png");
            if (mic_enabled)
                start_audio();
            else
                stop_audio();
            micButton->setIcon(QPixmap::fromImage(img));
        }
        void beforeStopAll()
        {
            emit stopAll();
        }
    signals:
        void stopAll();
    };
    #endif
```

Файл screenrecorder.cpp:

```
#include <string>
#include <vector>

#include <QApplication>
#include <QScreen>
#include <QDebug>

#include "opencv2/opencv.hpp"
#include <uvgrtp/lib.hh>

#include "screenrecorder.h"
#include "imageutil.h"
#include "cvmatandqimage.h"
#include "config.h"
```

```
using namespace cv;

void ScreenRecorder::run()
{
    QRect rect = QApplication::screens().at(0)->geometry();
    uvgrtp::context ctx;
    uvgrtp::session *sess = ctx.create_session(server);
    int flags = RCE_FRAGMENT_GENERIC | RCE_SEND_ONLY;
    uvgrtp::media_stream  *stream  =  sess->create_stream(IMAGE_UDP_PORT,
RTP_FORMAT_GENERIC, flags);
    std::vector<uint8_t> encoded;
    uint32_t frame_counter = 0;
    if (stream)
    {
        Mat image, send;
        while (!QThread::currentThread()->isInterruptionRequested())
        {
            QPixmap pixmap = imageutil::takeScreenShot(rect);
            image = QtOcv::image2Mat(pixmap.toImage());
            resize(image,  send,  Size(FRAME_WIDTH,  FRAME_HEIGHT),  0,  0,
INTER_LINEAR);
            std::vector<int> compression_params;
            compression_params.push_back(IMWRITE_JPEG_QUALITY);
            compression_params.push_back(quality);
            imencode(".jpg", send, encoded, compression_params);
            int payload_len = encoded.size();
            int current_seq = 0;
            auto    header_frame    =    std::unique_ptr<uint8_t[]>(new
uint8_t[sizeof(uint32_t) + 2 * sizeof(int)]);
            memcpy(header_frame.get(),                  &frame_counter,
sizeof(frame_counter));
            memcpy(header_frame.get()      +      sizeof(frame_counter),
&current_seq, sizeof(current_seq));
            memcpy(header_frame.get()    +    sizeof(frame_counter)    +
sizeof(current_seq), &payload_len, sizeof(payload_len));
            stream->push_frame(header_frame.get(), sizeof(uint32_t) + 2
* sizeof(int), RTP_NO_FLAGS);
            current_seq++;

std::this_thread::sleep_for(std::chrono::milliseconds(frame_interval));
            int total_pack = 1 + (payload_len - 1) / pack_size;
            for (int i = 0; i < total_pack; i++)
            {
                int  to_send  =  min<int>(pack_size,  payload_len  -  i  *
pack_size);
                auto     frame     =     std::unique_ptr<uint8_t[]>(new
uint8_t[sizeof(uint32_t) + sizeof(int) + to_send]);
                memcpy(frame.get(),                     &frame_counter,
sizeof(frame_counter));
                memcpy(frame.get()       +       sizeof(frame_counter),
&current_seq, sizeof(current_seq));
                memcpy(frame.get()     +     sizeof(frame_counter)     +
sizeof(current_seq), encoded.data() + i * pack_size, to_send);
                stream->push_frame(frame.get(),    sizeof(uint32_t)    +
sizeof(int) + to_send, RTP_NO_FLAGS);
                current_seq++;

std::this_thread::sleep_for(std::chrono::milliseconds(frame_interval));
            }
            frame_counter++;
```

```
std::this_thread::sleep_for(std::chrono::milliseconds(frame_interval));
            }
            sess->destroy_stream(stream);
        }
        if (sess)
            ctx.destroy_session(sess);
    }
```

Файл screenrecorder.h:

```
#ifndef SCREENRECORDER_H
#define SCREENRECORDER_H

#include <QThread>
#include <QMutex>

class ScreenRecorder : public QThread
{
    Q_OBJECT
private:
    char *server;
    int pack_size;
    int frame_interval;
    int quality;

public:
    ScreenRecorder(char *server, int pack_size, int frame_interval, int
quality)
            :             server(server),             pack_size(pack_size),
frame_interval(frame_interval), quality(quality) {}

protected:
    virtual void run();
public slots:
    void terminateThread()
    {
        if (isRunning())
        {
            requestInterruption();
            wait();
        }
    }
};

#endif
```

Файл Server.cpp:

```
#include <iostream>
#include <cstdlib>
#include <map>
#include <set>

#include <QApplication>
#include <QMainWindow>
#include <QThread>
#include <QMutex>
#include <QDebug>
#include <QFile>
```

```
#include <QGraphicsPixmapItem>
#include <QMessageBox>
#include <QSettings>

#include <uvgrtp/lib.hh>
#include "opencv2/opencv.hpp"
using namespace cv;

#include "udpplayer.h"
#include "screenrecorder.h"
#include "config.h"
#include "zoomui.h"
#include "cvmatandqimage.h"
#include "workerthread.h"
#include "mainwindow.h"
#include "startwindow.h"
#include "settingswindow.h"
#include "grabber.h"
#include "inputapplicant.h"

constexpr int RECEIVER_WAIT_TIME_MS = 10 * 1000;

struct FrameData
{
    int frame_num;
    int buffer_size;
    FrameData() : frame_num(-1), buffer_size(0) {}
    FrameData(int frame_num, int buffer_size) : frame_num(frame_num),
buffer_size(buffer_size) {}
};

struct FrameChunk
{
    int seq;
    int size;
    uint8_t *data;
    FrameChunk() : seq(-1), size(0), data(nullptr) {}
    FrameChunk(int seq, int size, uint8_t *data) : seq(seq), size(size),
data(data) {}
    bool operator<(const FrameChunk &other) const
    {
        return seq < other.seq;
    }
    FrameChunk(const FrameChunk &other) : seq(other.seq),
size(other.size)
    {
        data = new uint8_t[size];
        memcpy(data, other.data, size);
    }
    ~FrameChunk()
    {
        delete[] data;
    }
};

void MyThread::run()
{
    uvgrtp::context ctx;
    uvgrtp::session *sess = ctx.create_session("0.0.0.0");
    int flags = RCE_FRAGMENT_GENERIC | RCE_RECEIVE_ONLY;
```

```
        uvgrtp::media_stream *receiver = sess->create_stream(IMAGE_UDP_PORT,
RTP_FORMAT_GENERIC, flags);
        if (receiver)
        {
            std::map<uint32_t, FrameData> frames;
            std::map<uint32_t, std::set<FrameChunk>> chunks;
            while (!QThread::currentThread()->isInterruptionRequested())
            {
                uvgrtp::frame::rtp_frame      *frame      =      receiver-
>pull_frame(RECEIVER_WAIT_TIME_MS);
                if (!frame)
                    break;
                uint32_t current_frame;
                int current_seq;
                memcpy(&current_frame, frame->payload, sizeof(uint32_t));
                memcpy(&current_seq,   frame->payload   +   sizeof(uint32_t),
sizeof(int));
                size_t  real_len  =  frame->payload_len  -  sizeof(uint32_t)  -
sizeof(int);
                uint8_t *data = new uint8_t[real_len];
                memcpy(data, frame->payload + sizeof(uint32_t) + sizeof(int),
real_len);
                chunks[current_frame].insert(FrameChunk(current_seq,
real_len, data));
                if  (chunks[current_frame].begin()->seq  ==  0)  //  received
header
                {
                    int buffer_size;
                    memcpy(&buffer_size,      chunks[current_frame].begin()-
>data, sizeof(int));
                    frames[current_frame]     =     FrameData(current_frame,
buffer_size);

chunks[current_frame].erase(chunks[current_frame].begin());
                }
                if (frames.count(current_frame))
                {
                    int offset = 0;
                    int buffer_size = frames[current_frame].buffer_size;
                    uint8_t *buffer = new uint8_t[buffer_size];
                    for  (auto  it  =  chunks[current_frame].begin();  it  !=
chunks[current_frame].end(); it++)
                    {
                        memcpy(buffer + offset, it->data, it->size);
                        offset += it->size;
                    }
                    if (offset == buffer_size)
                    {
                        Mat rawData = Mat(1, buffer_size, CV_8UC1, buffer);
                        Mat cvimg = imdecode(rawData, IMREAD_COLOR);
                        if (cvimg.size().width == 0)
                        {
                            std::cerr << "decode failure!" << std::endl;
                            continue;
                        }
                        resize(cvimg,   cvimg,   Size(1278,   638),   0,   0,
INTER_LINEAR);
                        QImage image = QtOcv::mat2Image(cvimg);
                        emit signalGUI(image);
                        frames.erase(current_frame);
                        chunks.erase(current_frame);
                    }
```

```
                    delete[] buffer;
                }
                (void)uvgrtp::frame::dealloc_frame(frame);
            }
            sess->destroy_stream(receiver);
        }
        if (sess)
            ctx.destroy_session(sess);
    }

    class SessionManager
    {
    private:
        UDPPlayer *player;
        ScreenRecorder *recorder;
        MyThread *listen_thread;
        MainWindow *window;
        StartWindow &startWindow;
        SettingsWindow *settingsWindow;
        GrabberSender *grabber;
        X11InputApplicant *inputApplicant;
        std::string ConnectServer;
        bool is_control;

    public:
        SessionManager(StartWindow &startWindow) : startWindow(startWindow)
        {
            QObject::connect(startWindow.settingsButton,
&QPushButton::clicked, [&]()
                             {
                settingsWindow = new SettingsWindow();
                settingsWindow->setFixedSize(settingsWindow->width(),
settingsWindow->height());
                QObject::connect(settingsWindow->saveButton,
&QPushButton::clicked, [&]()
                                 {
                    settingsWindow->saveSettings();
                    settingsWindow->close();
                    settingsWindow->deleteLater(); });
                settingsWindow->show(); });
        }
        void start()
        {
            startWindow.hide();
            player = new UDPPlayer();
            QSettings settings(SETTINGS_FILE, QSettings::IniFormat);
            int pack_size = settings.value("pack_size", PACK_SIZE).toInt();
            int frame_interval = (1000 / settings.value("fps", FPS).toInt());
            int quality = settings.value("quality", ENCODE_QUALITY).toInt();
            if (!is_control)
            {
                inputApplicant = new X11InputApplicant();
                inputApplicant->start();
                recorder = new ScreenRecorder((char *)ConnectServer.c_str(),
pack_size, frame_interval, quality);
                recorder->start();
                QObject::connect(QApplication::instance(),
SIGNAL(aboutToQuit()), recorder, SLOT(terminateThread()));
            }
            window = new MainWindow();
            window->setFixedSize(window->width(), window->height());
```

```
            QObject::connect(window->endButton,       &QPushButton::clicked,
[&](bool)
                                {
                stop();
                startWindow.show(); });
            window->init_audio_input((char *)ConnectServer.c_str());
            window->prepareScene(is_control);
            window->show();
            if (is_control)
            {
                grabber = new GrabberSender();
                grabber->set_host(ConnectServer);
                grabber->start();
                listen_thread = new MyThread();
                QObject::connect(listen_thread,       SIGNAL(signalGUI(const
QImage &)), window, SLOT(processImage(const QImage &)));
                QObject::connect(listen_thread,  &QThread::finished,  window,
&MainWindow::beforeStopAll);
                QObject::connect(window, &MainWindow::stopAll, [&]()
                                 { stop();
                startWindow.show(); });
                listen_thread->start();
                QObject::connect(QApplication::instance(),
SIGNAL(aboutToQuit()), listen_thread, SLOT(terminateThread()));
            }
        }
        void stop()
        {
            if (is_control)
            {
                listen_thread->terminateThread();
                listen_thread->deleteLater();
                grabber->stop();
                delete grabber;
                grabber = nullptr;
            }
            else
            {
                recorder->terminateThread();
                recorder->deleteLater();
                inputApplicant->stop();
                delete inputApplicant;
                inputApplicant = nullptr;
            }
            window->deinit_audio_input();
            player->deleteLater();
            window->deleteLater();
        }
        void connectButtonClicked()
        {
            QString ip = startWindow.ipLabel->text();
            if (ip.isEmpty() || QHostAddress(ip).isNull())
            {
                QMessageBox::warning(&startWindow, "Error", "Please enter an
IP address");
                return;
            }
            ConnectServer = ip.toLocal8Bit().data();
            is_control = startWindow.controlCheckbox->isChecked();
            start();
        }
    };
```

```
int main(int argc, char **argv)
{
    QApplication app(argc, argv);
    StartWindow startWindow;
    startWindow.setFixedSize(startWindow.width(), startWindow.height());
    SessionManager manager(startWindow);
    QObject::connect(startWindow.connectButton,   &QPushButton::clicked,
[&]()
                     { manager.connectButtonClicked(); });
    startWindow.show();
    return app.exec();
}
```

Файл settingswindow.h:

```
#ifndef UI_SETTINGSWINDOW_H
#define UI_SETTINGSWINDOW_H
#include <QtCore/QVariant>
#include <QtCore/QSettings>
#include <QtWidgets/QApplication>
#include <QtWidgets/QHBoxLayout>
#include <QtWidgets/QLabel>
#include <QtWidgets/QPushButton>
#include <QtWidgets/QSpacerItem>
#include <QtWidgets/QSpinBox>
#include <QtWidgets/QVBoxLayout>
#include <QtWidgets/QWidget>
#include <QtWidgets/QLineEdit>
#include "config.h"

class CustomSpinbox : public QSpinBox
{
    Q_OBJECT
public:
    CustomSpinbox(QWidget *parent = nullptr)
        : QSpinBox(parent)
    {
        lineEdit()->setReadOnly(true);
    }
};

class Ui_SettingsWindow
{
public:
    QLabel *label;
    QWidget *verticalLayoutWidget;
    QVBoxLayout *verticalLayout;
    QHBoxLayout *horizontalLayout;
    QLabel *label_2;
    QSpinBox *fpsVal;
    QHBoxLayout *horizontalLayout_2;
    QLabel *label_3;
    CustomSpinbox *packetVal;
    QHBoxLayout *horizontalLayout_3;
    QLabel *label_4;
    QSpinBox *qualityVal;
    QWidget *horizontalLayoutWidget_4;
    QHBoxLayout *horizontalLayout_4;
    QSpacerItem *horizontalSpacer;
    QPushButton *saveButton;
```

```
        QSpacerItem *horizontalSpacer_2;

        void setupUi(QWidget *SettingsWindow)
        {
            if (SettingsWindow->objectName().isEmpty())
                SettingsWindow-
>setObjectName(QString::fromUtf8("SettingsWindow"));
            SettingsWindow->resize(800, 600);
            label = new QLabel(SettingsWindow);
            label->setObjectName(QString::fromUtf8("label"));
            label->setGeometry(QRect(330, 20, 101, 41));
            QFont font;
            font.setPointSize(15);
            font.setBold(true);
            label->setFont(font);
            label->setAlignment(Qt::AlignCenter);
            verticalLayoutWidget = new QWidget(SettingsWindow);
            verticalLayoutWidget-
>setObjectName(QString::fromUtf8("verticalLayoutWidget"));
            verticalLayoutWidget->setGeometry(QRect(70, 160, 271, 241));
            verticalLayout = new QVBoxLayout(verticalLayoutWidget);
            verticalLayout-
>setObjectName(QString::fromUtf8("verticalLayout"));
            verticalLayout->setContentsMargins(0, 0, 0, 0);
            horizontalLayout = new QHBoxLayout();
            horizontalLayout-
>setObjectName(QString::fromUtf8("horizontalLayout"));
            label_2 = new QLabel(verticalLayoutWidget);
            label_2->setObjectName(QString::fromUtf8("label_2"));
            QFont font1;
            font1.setPointSize(12);
            label_2->setFont(font1);

            horizontalLayout->addWidget(label_2);

            fpsVal = new QSpinBox(verticalLayoutWidget);
            fpsVal->setObjectName(QString::fromUtf8("fpsVal"));
            fpsVal->setMinimum(1);
            fpsVal->setMaximum(60);

            horizontalLayout->addWidget(fpsVal);

            verticalLayout->addLayout(horizontalLayout);

            horizontalLayout_2 = new QHBoxLayout();
            horizontalLayout_2-
>setObjectName(QString::fromUtf8("horizontalLayout_2"));
            label_3 = new QLabel(verticalLayoutWidget);
            label_3->setObjectName(QString::fromUtf8("label_3"));
            label_3->setFont(font1);

            horizontalLayout_2->addWidget(label_3);

            packetVal = new CustomSpinbox(verticalLayoutWidget);
            packetVal->setObjectName(QString::fromUtf8("packetVal"));
            packetVal->setMinimum(1024);
            packetVal->setMaximum(60416);
            packetVal->setSingleStep(1024);
```

```
            horizontalLayout_2->addWidget(packetVal);

            verticalLayout->addLayout(horizontalLayout_2);

            horizontalLayout_3 = new QHBoxLayout();
            horizontalLayout_3-
>setObjectName(QString::fromUtf8("horizontalLayout_3"));
            label_4 = new QLabel(verticalLayoutWidget);
            label_4->setObjectName(QString::fromUtf8("label_4"));
            label_4->setFont(font1);

            horizontalLayout_3->addWidget(label_4);

            qualityVal = new QSpinBox(verticalLayoutWidget);
            qualityVal->setObjectName(QString::fromUtf8("qualityVal"));
            qualityVal->setMinimum(1);
            qualityVal->setMaximum(100);

            horizontalLayout_3->addWidget(qualityVal);

            verticalLayout->addLayout(horizontalLayout_3);

            horizontalLayoutWidget_4 = new QWidget(SettingsWindow);
            horizontalLayoutWidget_4-
>setObjectName(QString::fromUtf8("horizontalLayoutWidget_4"));
            horizontalLayoutWidget_4->setGeometry(QRect(-1, 510, 801, 80));
            horizontalLayout_4 = new QHBoxLayout(horizontalLayoutWidget_4);
            horizontalLayout_4-
>setObjectName(QString::fromUtf8("horizontalLayout_4"));
            horizontalLayout_4->setContentsMargins(0, 0, 0, 0);
            horizontalSpacer      =      new      QSpacerItem(40,      20,
QSizePolicy::Expanding, QSizePolicy::Minimum);

            horizontalLayout_4->addItem(horizontalSpacer);

            saveButton = new QPushButton(horizontalLayoutWidget_4);
            saveButton->setObjectName(QString::fromUtf8("saveButton"));
            QSizePolicy                    sizePolicy(QSizePolicy::Minimum,
QSizePolicy::Fixed);
            sizePolicy.setHorizontalStretch(0);
            sizePolicy.setVerticalStretch(0);
            sizePolicy.setHeightForWidth(saveButton-
>sizePolicy().hasHeightForWidth());
            saveButton->setSizePolicy(sizePolicy);
            saveButton->setMinimumSize(QSize(200, 35));

            horizontalLayout_4->addWidget(saveButton);

            horizontalSpacer_2      =      new      QSpacerItem(40,      20,
QSizePolicy::Expanding, QSizePolicy::Minimum);

            horizontalLayout_4->addItem(horizontalSpacer_2);

            retranslateUi(SettingsWindow);
```

```
            QMetaObject::connectSlotsByName(SettingsWindow);
        } // setupUi

        void retranslateUi(QWidget *SettingsWindow)
        {
            SettingsWindow-
>setWindowTitle(QCoreApplication::translate("SettingsWindow",      "Settings",
nullptr));
            label->setText(QCoreApplication::translate("SettingsWindow",
"Settings", nullptr));
            label_2->setText(QCoreApplication::translate("SettingsWindow",
"FPS", nullptr));
            label_3->setText(QCoreApplication::translate("SettingsWindow",
"Packet size", nullptr));
            label_4->setText(QCoreApplication::translate("SettingsWindow",
"Quality", nullptr));
            saveButton-
>setText(QCoreApplication::translate("SettingsWindow",     "Save     changes",
nullptr));
        } // retranslateUi
    };

    class SettingsWindow : public QWidget, public Ui_SettingsWindow
    {
        Q_OBJECT

    public:
        SettingsWindow(QWidget *parent = nullptr)
            : QWidget(parent)
        {
            setupUi(this);
            QSettings settings(SETTINGS_FILE, QSettings::IniFormat);
            fpsVal->setValue(settings.value("fps", FPS).toInt());
            packetVal->setValue(settings.value("packet",
PACK_SIZE).toInt());
            qualityVal->setValue(settings.value("quality",
ENCODE_QUALITY).toInt());
        }

        void saveSettings()
        {
            QSettings settings(SETTINGS_FILE, QSettings::IniFormat);
            settings.setValue("fps", fpsVal->value());
            settings.setValue("packet", packetVal->value());
            settings.setValue("quality", qualityVal->value());
        }
    };

    #endif
```

Файл startwindow.h:

```
#ifndef STARTWINDOW_H
#define STARTWINDOW_H

#include <QtCore/QVariant>
#include <QtGui/QIcon>
#include <QtWidgets/QApplication>
#include <QtWidgets/QCheckBox>
#include <QtWidgets/QLabel>
```

```
#include <QtWidgets/QLineEdit>
#include <QtWidgets/QPushButton>
#include <QtWidgets/QWidget>

class Ui_StartWindow
{
public:
    QLabel *label;
    QLabel *label_2;
    QLineEdit *ipLabel;
    QPushButton *connectButton;
    QPushButton *settingsButton;
    QCheckBox *controlCheckbox;

    void setupUi(QWidget *StartWindow)
    {
        if (StartWindow->objectName().isEmpty())
            StartWindow-
>setObjectName(QString::fromUtf8("StartWindow"));
        StartWindow->resize(1280, 720);
        label = new QLabel(StartWindow);
        label->setObjectName(QString::fromUtf8("label"));
        label->setGeometry(QRect(510, 10, 241, 101));
        QFont font;
        font.setPointSize(15);
        font.setBold(true);
        label->setFont(font);
        label->setTextFormat(Qt::AutoText);
        label->setAlignment(Qt::AlignCenter);
        label_2 = new QLabel(StartWindow);
        label_2->setObjectName(QString::fromUtf8("label_2"));
        label_2->setGeometry(QRect(510, 270, 251, 51));
        QFont font1;
        font1.setPointSize(15);
        label_2->setFont(font1);
        label_2->setAlignment(Qt::AlignCenter);
        ipLabel = new QLineEdit(StartWindow);
        ipLabel->setObjectName(QString::fromUtf8("ipLabel"));
        ipLabel->setGeometry(QRect(520, 340, 231, 41));
        ipLabel->setAlignment(Qt::AlignCenter);
        connectButton = new QPushButton(StartWindow);
        connectButton-
>setObjectName(QString::fromUtf8("connectButton"));
        connectButton->setGeometry(QRect(600, 440, 80, 23));
        connectButton->setStyleSheet(QString::fromUtf8("background-
color: rgb(87, 227, 137);"));
        settingsButton = new QPushButton(StartWindow);
        settingsButton-
>setObjectName(QString::fromUtf8("settingsButton"));
        settingsButton->setGeometry(QRect(1170, 640, 61, 51));
        QIcon icon;
        icon.addFile(QString::fromUtf8(":/config.png"),         QSize(),
QIcon::Normal, QIcon::Off);
        settingsButton->setIcon(icon);
        settingsButton->setIconSize(QSize(32, 32));
        controlCheckbox = new QCheckBox(StartWindow);
        controlCheckbox-
>setObjectName(QString::fromUtf8("controlCheckbox"));
        controlCheckbox->setGeometry(QRect(570, 400, 141, 21));

        retranslateUi(StartWindow);
```

```
            QMetaObject::connectSlotsByName(StartWindow);
        } // setupUi

        void retranslateUi(QWidget *StartWindow)
        {
            StartWindow-
>setWindowTitle(QCoreApplication::translate("StartWindow",  "Remote  control",
nullptr));
            label->setText(QCoreApplication::translate("StartWindow",
"Remote control app", nullptr));
            label_2->setText(QCoreApplication::translate("StartWindow",
"Connect to server:", nullptr));
            ipLabel-
>setPlaceholderText(QCoreApplication::translate("StartWindow",  "IP  address",
nullptr));
            connectButton-
>setText(QCoreApplication::translate("StartWindow", "Connect", nullptr));
            settingsButton->setText(QString());
            controlCheckbox-
>setText(QCoreApplication::translate("StartWindow",   "Control   another   PC",
nullptr));
        } // retranslateUi
    };

    class StartWindow : public QWidget, public Ui_StartWindow
    {
        Q_OBJECT

    public:
        StartWindow(QWidget *parent = nullptr)
            : QWidget(parent)
        {
            setupUi(this);
        }
    };
    #endif
```

Файл udpplayer.cpp:

```
#include "udpplayer.h"

UDPPlayer::UDPPlayer(QObject *parent) : QObject(parent)
{
    socket = new QUdpSocket();
    socket->bind(AUDIO_UDP_PORT);
    QAudioFormat format = getAudioFormat();
    QAudioDeviceInfo info(QAudioDeviceInfo::defaultOutputDevice());
    if (!info.isFormatSupported(format))
        format = info.nearestFormat(format);
    output = new QAudioOutput(format);
    device = output->start();
    connect(socket, &QUdpSocket::readyRead, this, &UDPPlayer::playData);
}

void UDPPlayer::playData()
{
    while (socket->hasPendingDatagrams())
    {
        QByteArray data;
```

```
        data.resize(socket->pendingDatagramSize());
        socket->readDatagram(data.data(), data.size());
        device->write(data.data(), data.size());
    }
}

QAudioFormat getAudioFormat()
{
    QAudioFormat format;
    format.setSampleRate(8000);
    format.setChannelCount(1);
    format.setSampleSize(16);
    format.setByteOrder(QAudioFormat::LittleEndian);
    format.setSampleType(QAudioFormat::SignedInt);
    format.setCodec("audio/pcm");
    QAudioDeviceInfo info(QAudioDeviceInfo::defaultInputDevice());
    if (!info.isFormatSupported(format))
        format = info.nearestFormat(format);
    return format;
}
```

Файл udpplayer.h:

```
#ifndef UDPPLAYER_H
#define UDPPLAYER_H
#include <QObject>
#include <QtMultimedia/QAudioOutput>
#include <QtMultimedia/QAudioInput>
#include <QtMultimedia/QAudioFormat>
#include <QUdpSocket>

#include "config.h"

class UDPPlayer : public QObject
{
    Q_OBJECT
public:
    explicit UDPPlayer(QObject *parent = 0);
    ~UDPPlayer()
    {
        socket->deleteLater();
        output->deleteLater();
    }

private slots:
    void playData();

private:
    QAudioOutput *output;
    QUdpSocket *socket;
    QIODevice *device;
};

QAudioFormat getAudioFormat();

#endif
```

Файл utils.cpp:

```
#include "utils.h"
#include <array>
#include <memory>
#include <stdexcept>

std::vector<std::string> Split(const std::string &string, const
std::string &delimiter)
{
    std::vector<std::string> result;
    if (string.empty())
    {
        return result;
    }
    size_t start = 0;
    size_t end = string.find(delimiter);
    while (end != std::string::npos)
    {
        result.push_back(string.substr(start, end - start));
        start = end + delimiter.size();
        end = string.find(delimiter, start);
    }
    result.push_back(string.substr(start, end - start));
    return result;
}

std::string exec(std::string cmd)
{
    std::array<char, 128> buffer;
    std::string result;
    std::unique_ptr<FILE, decltype(&pclose)> pipe(popen(cmd.c_str(),
"r"), pclose);
    if (!pipe)
        throw std::runtime_error("popen() failed!");
    while (fgets(buffer.data(), static_cast<int>(buffer.size()),
pipe.get()) != nullptr)
        result += buffer.data();
    return result;
}
```

Файл utils.h:

```
#pragma once
#include <string>
#include <vector>

std::vector<std::string> Split(const std::string &string, const
std::string &delimiter = " ");

std::string exec(std::string cmd);
```

Файл workerthread.h:

```
#ifndef WORKERTHREAD_H
#define WORKERTHREAD_H

#include <QThread>
#include <QMutex>
#include <QImage>

class MyThread : public QThread
```

```
{
    Q_OBJECT
protected:
    virtual void run();
signals:
    void signalGUI(QImage);
public slots:
    void terminateThread()
    {
        if (isRunning())
        {
            requestInterruption();
            wait();
        }
    }
};

#endif
```

Файл zoomui.h:

```
/********************************************************************************
** Form generated from reading UI file 'zoomui.ui'
**
** Created by: Qt User Interface Compiler version 5.15.2
**
** WARNING! All changes made in this file will be lost when recompiling UI file!
********************************************************************************/

#ifndef UI_ZOOMUI_H
#define UI_ZOOMUI_H

#include <QtCore/QVariant>
#include <QtGui/QIcon>
#include <QtWidgets/QApplication>
#include <QtWidgets/QFrame>
#include <QtWidgets/QGraphicsView>
#include <QtWidgets/QMainWindow>
#include <QtWidgets/QPushButton>
#include <QtWidgets/QWidget>
#include <QtWidgets/QLabel>
#include <QtWidgets/QGraphicsPixmapItem>

QT_BEGIN_NAMESPACE

class Ui_MainWindow
{
public:
    QWidget *centralwidget;
    QGraphicsScene *graphicsScene;
    QGraphicsView *graphicsView;
    QGraphicsPixmapItem *pixmap;
    QPixmap imgpix;
    QFrame *frame;
    QLabel *controlledLabel;
    QPushButton *micButton;
    QPushButton *endButton;
```

```
          void setupUi(QMainWindow *MainWindow)
          {
              if (MainWindow->objectName().isEmpty())
                  MainWindow->setObjectName(QString::fromUtf8("MainWindow"));
              MainWindow->resize(1280, 720);
              centralwidget = new QWidget(MainWindow);
              centralwidget-
>setObjectName(QString::fromUtf8("centralwidget"));
              controlledLabel = new QLabel(centralwidget);
              controlledLabel-
>setObjectName(QString::fromUtf8("controlledLabel"));
              controlledLabel->setGeometry(QRect(510, 10, 241, 101));
              QFont font;
              font.setPointSize(15);
              font.setBold(true);
              controlledLabel->setFont(font);
              controlledLabel->setTextFormat(Qt::AutoText);
              controlledLabel->setAlignment(Qt::AlignCenter);
              graphicsScene = new QGraphicsScene;
              pixmap = new QGraphicsPixmapItem;
              graphicsScene->addItem(pixmap);
              graphicsView = new QGraphicsView(centralwidget);
              graphicsView->setObjectName(QString::fromUtf8("graphicsView"));
              graphicsView->setGeometry(QRect(0, 0, 1280, 640));
              graphicsView->setScene(graphicsScene);
              frame = new QFrame(centralwidget);
              frame->setObjectName(QString::fromUtf8("frame"));
              frame->setGeometry(QRect(0, 639, 1281, 102));
              frame->setStyleSheet(QString::fromUtf8("background-color:
gray"));
              frame->setFrameShape(QFrame::StyledPanel);
              frame->setFrameShadow(QFrame::Raised);
              micButton = new QPushButton(frame);
              micButton->setObjectName(QString::fromUtf8("pushButton"));
              micButton->setGeometry(QRect(0, 0, 91, 81));
              micButton->setAutoFillBackground(false);
              micButton->setIcon(QPixmap::fromImage(QImage(":/mic-on.png")));
              micButton->setIconSize(QSize(64, 64));
              endButton = new QPushButton(frame);
              endButton->setObjectName(QString::fromUtf8("endButton"));
              endButton->setGeometry(QRect(1150, 20, 101, 41));
              endButton->setStyleSheet(QString::fromUtf8("background-color:
red;\n"
                                                         "color: rgb(255, 255,
255);"));
              MainWindow->setCentralWidget(centralwidget);


              retranslateUi(MainWindow);


              QMetaObject::connectSlotsByName(MainWindow);
          } // setupUi


          void retranslateUi(QMainWindow *MainWindow)
          {
              MainWindow-
>setWindowTitle(QCoreApplication::translate("MainWindow",  "Remote  control",
nullptr));
              micButton->setText(QString());
```

```
            endButton->setText(QCoreApplication::translate("MainWindow",
"End", nullptr));
        } // retranslateUi
    };

    namespace Ui
    {
        class MainWindow : public Ui_MainWindow
        {
        public:
            void prepareScene(bool is_control)
            {
                if (is_control)
                {
                    controlledLabel->hide();
                    graphicsView->show();
                }
                else
                {
                    controlledLabel->show();
                    graphicsView->hide();
                }
            }
        };
    } // namespace Ui

    QT_END_NAMESPACE

    #endif // UI_ZOOMUI_H
```

ПРИЛОЖЕНИЕ Г
(обязательное)

Ведомость документов