Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Системное программирование (СП)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту на тему

Программное средство «Удалённый рабочий стол»

БГУИР КП I–40 01 01 020 ПЗ

Выполнил

студент гр. 251003 Панкратьев Е.С.

Проверил: Деменковец Д.В.

Минск 2024

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

УТВЕРЖДАЮ

Заведующий кафедрой ПОИТ

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

Лапицкая Н.В. 2024г.

ЗАДАНИЕ

по курсовому проектированию

Студенту Панкратьеву Егору Сергеевичу\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Тема работы Программное средство «Удалённый рабочий стол»\_\_\_\_\_\_\_

2. Срок сдачи законченной работы *16.12.2024г.*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

3. Исходные данные к работе *Исходные данные к работе Среда программирования Visual Studio; язык программирования C++; интерфейс Windows API; библиотека CryptoPP для шифрования данных; библиотека OpenCV для работы с изображениями (интерполяция, кодирование)*

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

*Введение*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*1 Анализ предметной области \_*\_\_\_\_\_\_\_\_\_\_\_*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*\_\_\_\_

*2 Проектирование программного средства* \_\_\_\_\_\_*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*3 Разработка программного средства средства*

*4 Тестирование и проверка работоспособности программного средства \_\_\_*

*5 Руководство по использованию программного средства\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Заключение \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Список использованных источников\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Приложения* \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

5. Перечень графического материала (с точным обозначением обязательных чертежей и графиков)

*Схема алгоритма в формате А1* \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

6. Консультант по курсовой работе *Деменковец Д.В.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

7.Дата выдачи задания *16.09.2024г.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

Календарный график работы над проектом на весь период проектирования (с обозначением сроков выполнения и процентом от общего объёма работы):

раздел 1 к 23.09.2024 – 15 % готовности работы;

раздел 2 к 14.10.2024 – 30 % готовности работы;

разделы 3,4 к 04.11.2024 – 60 % готовности работы;

раздел 5 к 18.11.2024 – 90 % готовности работы;

оформление пояснительной записки и графического материала к 25.11.2024. – 100 % готовности работы.

Защита курсового проекта с 23 ноября по 16 декабря 2024 г.––––––––––

РУКОВОДИТЕЛЬ *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Деменковец Д.В.*

*(подпись)*

Задание принял к исполнению *\_\_\_\_\_\_\_\_\_Панкратьев Е.С. 16.09.2024г.*

*(дата и подпись студента)*

**СОДЕРЖАНИЕ**

[Введение 6](#_Toc184901042)

[1 Анализ Предметной области 7](#_Toc184901043)

[1.1 Анализ существующих аналогов 7](#_Toc184901044)

[1.1.1 Программное средство «AnyDesk» 7](#_Toc184901045)

[1.1.2 Программное средство «TeamViewer» 8](#_Toc184901046)

[1.2 Постановка задачи 9](#_Toc184901047)

[1.2.1 Описание предметной области 9](#_Toc184901048)

[1.2.2 Информационная база задачи 9](#_Toc184901049)

[1.2.3 Функциональное назначение 10](#_Toc184901050)

[2 проектирование программного средства 12](#_Toc184901051)

[2.1 Структура программы 12](#_Toc184901052)

[2.1.1 Клиент для получения удаленного доступа 12](#_Toc184901053)

[2.1.2 Клиент для предоставления удаленного доступа 14](#_Toc184901054)

[3 Разработка программного средства 16](#_Toc184901055)

[3.1 Описание применения системного программировния 16](#_Toc184901056)

[3.1.1 Совместное использование WinAPI и многопоточности 16](#_Toc184901057)

[3.2 Описание применения элементов системного программирования 21](#_Toc184901058)

[3.2.1 WinAPI 21](#_Toc184901059)

[3.2.1 Работа с шрифтами 23](#_Toc184901060)

[3.2.1 Работа с реестром 24](#_Toc184901061)

[3.2.2 Многопоточное программирование 26](#_Toc184901062)

[3.3 Схема алгоритмов решения задач по ГОСТ 19.701-90 28](#_Toc184901063)

[3.3.1 Схема алгоритма exchangeKeys 28](#_Toc184901064)

[3.3.2 Схема алгоритма splitIntoChunks 29](#_Toc184901065)

[3.3.3 Схема алгоритма captureScreen 30](#_Toc184901066)

[3.3.4 Схема алгоритма acceptSingleConnetion 31](#_Toc184901067)

[4 Тестирование и проверка работоспособности программного средства 32](#_Toc184901068)

[4.1 Запуск программного средства 32](#_Toc184901069)

[4.1.1 Тест 1 32](#_Toc184901070)

[4.1.2 Тест 2 32](#_Toc184901071)

[4.2 Проверка изменения настроек 33](#_Toc184901072)

[4.2.1 Тест 1 33](#_Toc184901073)

[4.2.2 Тест 2 33](#_Toc184901074)

[4.2.3 Тест 3 34](#_Toc184901075)

[4.2.4 Тест 4 34](#_Toc184901076)

[4.2.5 Тест 5 34](#_Toc184901077)

[4.2.6 Тест 6 35](#_Toc184901078)

[4.2.7 Тест 7 35](#_Toc184901079)

[4.2.8 Тест 8 35](#_Toc184901080)

[4.2.9 Тест 9 36](#_Toc184901081)

[4.2.10 Тест 10 36](#_Toc184901082)

[4.2.11 Тест 11 36](#_Toc184901083)

[4.3 Сохранение настроек в реестре 37](#_Toc184901084)

[4.3.1 Тест 1 37](#_Toc184901085)

[4.3.2 Тест 2 37](#_Toc184901086)

[4.4 Демонстрация экрана 38](#_Toc184901087)

[4.4.1 Тест 1 38](#_Toc184901088)

[4.4.2 Тест 2 38](#_Toc184901089)

[4.5 Отправка событий 39](#_Toc184901090)

[4.5.1 Тест 1 39](#_Toc184901091)

[4.5.2 Тест 2 40](#_Toc184901092)

[5 Руководство по запуску и использованию программного средства 41](#_Toc184901093)

[5.1 Запуск программы 41](#_Toc184901094)

[5.1.1 Запуск программы для предоставления удаленного доступа 41](#_Toc184901095)

[5.1.2 Запуск программы для получения удаленного доступа 42](#_Toc184901096)

[5.2 Использование 42](#_Toc184901097)

[5.2.1 Использование программы для предоставления удаленного доступа 42](#_Toc184901098)

[5.2.2 Использование программы для получения удаленного доступа 43](#_Toc184901099)

[Заключение 45](#_Toc184901100)

[Список использованной литературы 46](#_Toc184901101)

[Приложение А 47](#_Toc184901102)

[Приложение Б 50](#_Toc184901103)

[Приложение В 52](#_Toc184901104)

[Приложение Г 54](#_Toc184901105)

[Приложение Д 55](#_Toc184901106)

[Приложение Е 56](#_Toc184901107)

[Приложение Ж 57](#_Toc184901108)

[Приложение З 58](#_Toc184901109)

[Приложение И 63](#_Toc184901110)

[Приложение К 67](#_Toc184901111)

[Приложение Л 69](#_Toc184901112)

[Приложение М 71](#_Toc184901113)

[Приложение Н 72](#_Toc184901114)

[Приложение О 74](#_Toc184901115)

[Приложение П 76](#_Toc184901116)

[Приложение Р 79](#_Toc184901117)

[Приложение С 81](#_Toc184901118)

[Приложение Т 83](#_Toc184901119)

[Приложение У 84](#_Toc184901120)

[Приложение Ф 86](#_Toc184901121)

[Приложение Х 88](#_Toc184901122)

[Приложение Ц 94](#_Toc184901123)

[Приложение Ч 99](#_Toc184901124)

Введение

В современном мире технологии удалённого доступа и обмена данными играют важную роль в обеспечении эффективной работы и взаимодействия. Удалённый доступ к рабочему столу становится неотъемлемой частью как в корпоративной, так и в личной среде, предоставляя возможность оперативного управления компьютерами на расстоянии.

Целью данного курсового проекта является разработка программного средства для реализации функции удалённого рабочего стола с использованием протокола UDP для передачи изображений экрана. Разрабатываемое приложение предназначено для передачи и отображения содержимого экрана одного устройства на другое в режиме реального времени, обеспечивая минимальные задержки и высокую производительность.

Основные задачи, решаемые в рамках проекта, включают:

* разработку алгоритмов захвата изображения экрана и его преобразования в формат, пригодный для передачи по сети;
* фрагментацию и сборку данных для эффективной передачи через UDP-протокол;
* обеспечение надёжного отображения передаваемых данных на принимающей стороне;
* реализацию пользовательского интерфейса для настройки параметров приложения и управления процессом подключения.
* передачу событий мыши и клавиатуры с удалённого устройства, используя протокол TCP для обеспечения гарантированной доставки данных;
* реализацию шифрования передаваемых данных для обеспечения их конфиденциальности.

Разработка программы осуществляется с использованием языка C++ и WinAPI, что обеспечивает гибкость и производительность при реализации системных функций и работы с сетью. Выбор протокола UDP обоснован необходимостью минимизации задержек при передаче данных, что критично для приложений реального времени, а использование TCP для передачи событий клавиатуры и мыши обусловлено сравнительно небольшим объёмом таких данных и потребностью в их надёжной доставке. Реализация механизмов шифрования направлена на защиту передаваемой информации от перехвата и несанкционированного доступа.

Таким образом, разработка программы удалённого рабочего стола является актуальной задачей, которая направлена на повышение удобства и эффективности удалённого управления компьютерами. Полученные в рамках проекта результаты могут найти применение в различных областях, таких как информационная безопасность, техническая поддержка и удалённое администрирование.

# Анализ Предметной области

## Анализ существующих аналогов

### Программное средство «AnyDesk»

Программное средство AnyDesk является одним из самых популярных решений для удалённого доступа к рабочему столу. Оно предоставляет пользователям возможность управлять компьютерами через сеть, обеспечивая высокую скорость работы и безопасность.

Основные функции программного средства «AnyDesk»:

* доступ к рабочему столу удалённого устройства в режиме реального времени;
* передача файлов между устройствами;
* возможность записи сессий;
* шифрование данных для обеспечения безопасности.

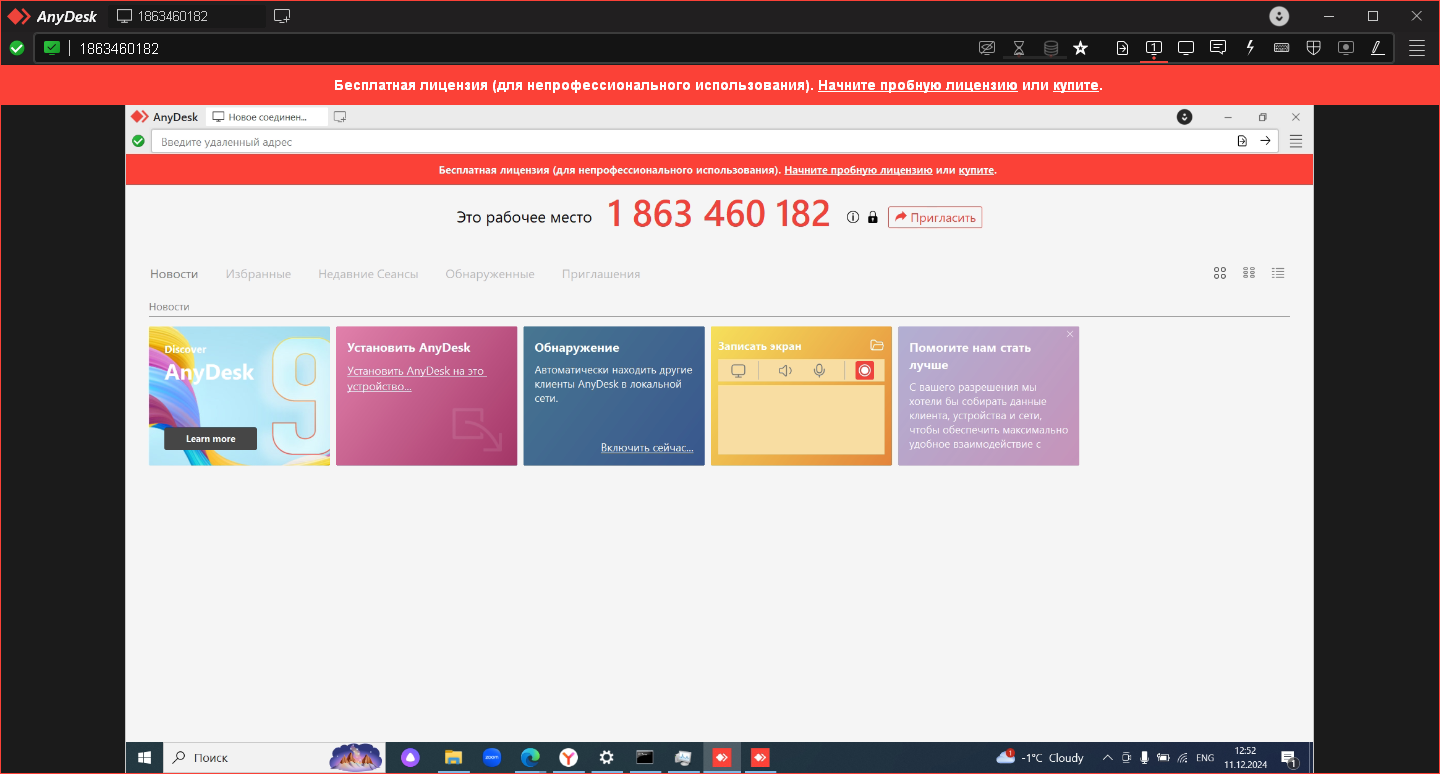


Рисунок 1.1.1 – Программное средство AnyDesk

К преимуществам программного средства «AnyDesk» можно отнести:

* высокая производительность;
* простота использования благодаря интуитивно понятному интерфейсу;
* поддержка множества операционных систем, включая Windows, macOS, Linux, iOS и Android.

К недостаткам данного программного средства можно отнести:  
– ограниченная функциональность в бесплатной версии.

### Программное средство «TeamViewer»

Программное средство TeamViewer — это ещё одно популярное решение для удалённого управления компьютерами. Оно широко используется как в корпоративной среде, так и для личных нужд.

Основные функции программного средства «TeamViewer»:

* управление удалёнными устройствами через сеть;
* передача файлов и папок между компьютерами;
* видеоконференции и онлайн-совещания;
* мониторинг и диагностика удалённых систем;
* поддержка многофакторной аутентификации и шифрования.

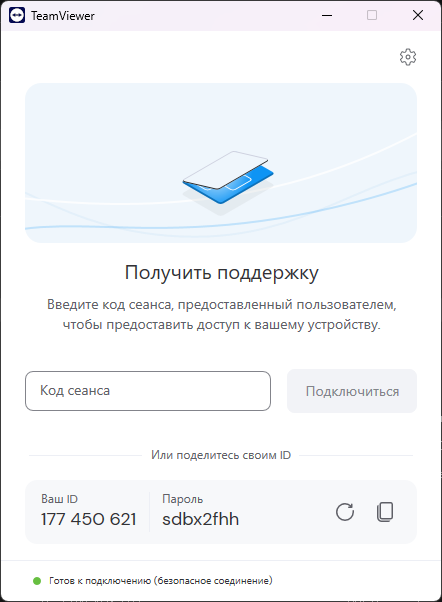


Рисунок 1.1.2 – Программное средство «TeamViewer»

К преимуществам программного средства «TeamViewer» можно отнести:

* широкий спектр возможностей, включая функционал для совместной работы;
* высокая степень безопасности за счёт шифрования и дополнительных уровней защиты;

К недостаткам данного программного средства можно отнести:

* высокая стоимость лицензии для коммерческого использования.

## Постановка задачи

### Описание предметной области

В рамках данного курсового проекта планируется разработать программное средство для удалённого доступа к рабочему столу, которое будет обеспечивать передачу изображения экрана, событий мыши и клавиатуры между устройствами через сеть.

В процессе разработки должны быть реализованы следующие базовые функции программы:

– захват изображения экрана отправляющего устройства;

– передача захваченных данных с фрагментацией через протокол UDP;

– сборка данных на принимающей стороне и вывод изображения в режиме реального времени;

– передача событий клавиатуры и мыши через протокол TCP для обеспечения гарантированной доставки данных;

– сжатие передаваемых данных для оптимизации сетевого трафика;

– шифрование данных для обеспечения конфиденциальности передаваемой информации;

– сохранение настроек соединения в системном реестре для удобства последующего использования;

– создание интуитивно понятного интерфейса для управления настройками подключения и передачи данных.

Для реализации данного программного средства будут использоваться язык программирования C++ и библиотека WinAPI, что обеспечит высокую производительность и гибкость при работе с низкоуровневыми функциями операционной системы. Использование многопоточности позволит одновременно обрабатывать передачу данных и взаимодействие с пользователем, минимизируя задержки.

### Информационная база задачи

Программное средство основывается на современных подходах к реализации удалённого доступа. Основным принципом работы является взаимодействие отправляющей и принимающей сторон через сеть. Передача изображения экрана осуществляется с использованием протокола UDP, что минимизирует задержки, а передача событий клавиатуры и мыши — через протокол TCP для обеспечения гарантированной доставки данных.

Для реализации интерфейса программы используется WinAPI, что позволяет создавать производительные графические приложения с глубокой интеграцией в среду Windows. Настройки соединения, включая параметры отправителя и приёмника, сохраняются в системном реестре, что обеспечивает удобство и гибкость настройки программы для повторного использования.

Сжатие данных будет осуществляться с использованием библиотек, оптимизированных для работы с бинарными потоками. Шифрование данных обеспечивается алгоритмами, соответствующими современным стандартам информационной безопасности, что делает систему устойчивой к перехвату данных.

Среда разработки Visual Studio будет использоваться для реализации проекта, благодаря её мощным средствам отладки, поддержки многопоточности и интеграции с внешними библиотеками. Используемые технологии и инструменты позволят создать стабильное и удобное в использовании программное средство для удалённого доступа.

### Функциональное назначение

Функции, выполняемые программным средством:

* захват изображения экрана отправляющего устройства в режиме реального времени;
* фрагментация изображения для передачи через протокол UDP;
* передача событий клавиатуры и мыши через протокол TCP;
* сборка фрагментов изображения на принимающей стороне и вывод на экран;
* масштабирование изображения для соответствия размерам окна;
* поддержка динамической настройки частоты кадров и разрешения;
* настройка параметров соединения с сохранением в системном реестре;
* сжатие передаваемых данных для уменьшения сетевого трафика;
* шифрование данных для повышения безопасности;
* обработка потери пакетов при передаче изображения через UDP;
* графический интерфейс для настройки и управления соединением.

Как итог, программное средство будет обладать достаточным функционалом (таблица 1) для обеспечения стабильной и производительной передачи изображения и событий ввода между устройствами через сеть.

Таблица 1.2.3.1 – Функциональные требования

|  |  |
| --- | --- |
| Идентификатор | Требования |
| ФТ-1 | Возможность захвата изображения экрана отправляющего устройства. |
| ФТ-2 | Возможность передачи изображения с фрагментацией через протокол UDP. |
| Продолжение таблицы 1.2.3.1 | |
| ФТ-3 | Возможность передачи событий клавиатуры и мыши через протокол TCP. |
| ФТ-4 | Возможность сборки переданных данных и вывода изображения на экран принимающего устройства. |
| ФТ-5 | Возможность масштабирования изображения для соответствия размерам окна. |
| ФТ-6 | Возможность настройки параметров соединения (частота кадров, разрешение, задержка). |
| ФТ-7 | Возможность сохранения настроек соединения в системном реестре. |
| ФТ-8 | Возможность сжатия передаваемых данных для уменьшения объёма трафика. |
| ФТ-9 | Возможность шифрования данных для обеспечения конфиденциальности. |
| ФТ-10 | Обработка потерь пакетов при передаче данных через UDP. |
| ФТ-11 | Наличие графического интерфейса для управления настройками соединения. |

# проектирование программного средства

## Структура программы

Программа состоит из двух компонентов: клиент для получения удаленного доступа и клиент для предоставления удаленного доступа.

### Клиент для получения удаленного доступа

Диалоговое окно настроек подключения.

**Назначение:**

* позволяет пользователю указать параметры подключения для устано ления связи с удаленным сервером.

Элементы интерфейса:

* поле для ввода IP-адреса сервера — для указания адреса удаленной машины;
* поле для ввода порта сервера — для указания порта, на который будут приходить UDP пакеты экрана удаленной машины;
* поле для ввода UDP-порта — для передачи скриншотов через UDP;
* поле для ввода частоты кадров — задает частоту обновления изображения;
* поле для ввода максимальной задержки — ограничивает допустимую задержку при получении данных;
* кнопки OK и Cancel — для запуска программы или выхода из неё.

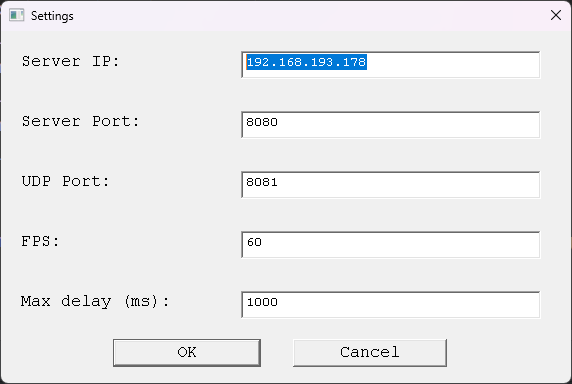


Рисунок 2.1.1.1 – Диалоговое окно настроек подключения

Главное окно.

**Назначение:**

* основное окно программы, которое отображает удаленный рабочий стол и предоставляет функции управления подключением.

Элементы интерфейса:

* окно визуализации удаленного экрана:
  + основная область окна отображает получаемое изображение с удаленного компьютера;
  + изображение масштабируется под размеры окна с учетом пропорций.

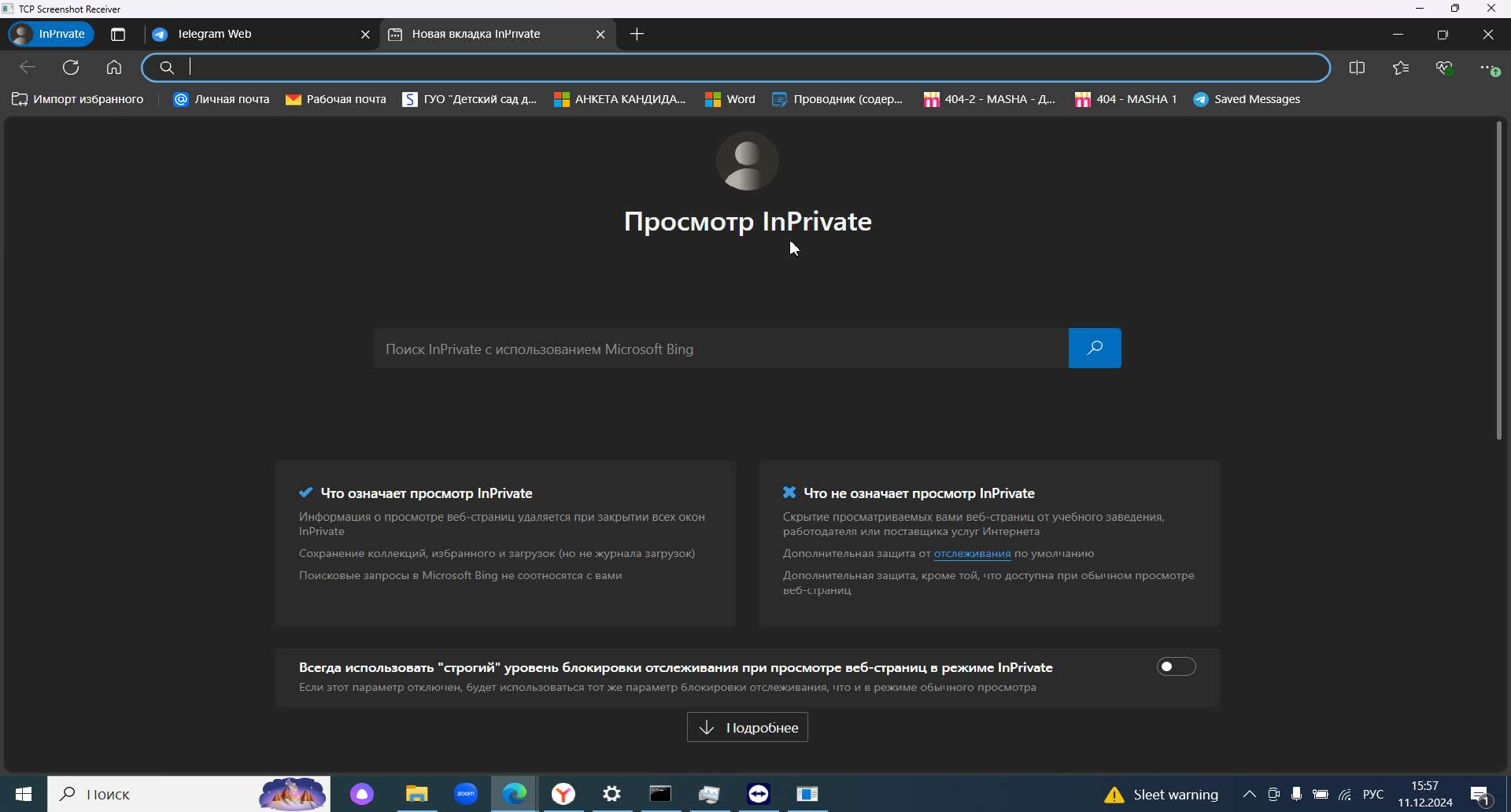


Рисунок 2.1.1.2 – Главное окно

Горячие клавиши:

* Fn + F1 — вызывает окно настроек для изменения параметров подключения.
* Fn + F12 — переключает режим отображения между оконным и полноэкранным.

Окно изменения параметров отображения.

**Назначение:**

* позволяет пользователю изменить параметры отображения во время работы программы.

Элементы интерфейса:

* поле для ввода частоты кадров — задает частоту обновления изображения;
* поле для ввода максимальной задержки — ограничивает допустимую задержку при получении данных;
* кнопки Apply и Exit — для применения параметров и выхода из приложения.

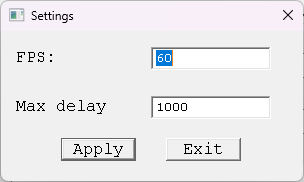


Рисунок 2.1.1.3 — Окно изменения параметров отображения

### Клиент для предоставления удаленного доступа

Диалоговое окно настроек подключения

Назначение:

* предоставляет пользователю настроить параметры подключения для предоставления удаленного доступа.

Элементы интерфейса:

* поле для ввода IP-адреса сервера — для указания адреса удаленной машины;
* поле для ввода порта сервера — для указания порта, на который будут приходить UDP пакеты экрана удаленной машины;
* поле для ввода частоты кадров — задает частоту обновления изображения;
* поле для ввода максимальной задержки— ограничивает максимальную задержку при получении данных;
* поле для ввода качества изображения — задает качество изображения при передаче;
* поля для ввода разрешения экрана (ширина и высота) — задает разрешение экрана, которое будет передаваться;
* поле для выбора формата изображения (jpg/png) — позволяет выбрать метод кодировки изображения для сжатия;
* кнопки OK и Cancel — для запуска программы или выхода из неё.



Рисунок 2.2.1.1 — Диалоговое окно настроек подключения

Главное окно.

Назначение:

* предоставляет пользователю настроить параметра захвата и передачи изображения экрана.

Элементы интерфейса:

* поле для ввода частоты кадров — задает частоту обновления изображения;
* поле для ввода максимальной задержки— ограничивает максимальную задержку при получении данных;
* поле для ввода качества изображения — задает качество изображения при передаче;
* поле для выбора формата изображения (jpg/png) — позволяет выбрать метод кодировки изображения для сжатия;
* кнопки Apply и Exit — для применения параметров и выхода из приложения.

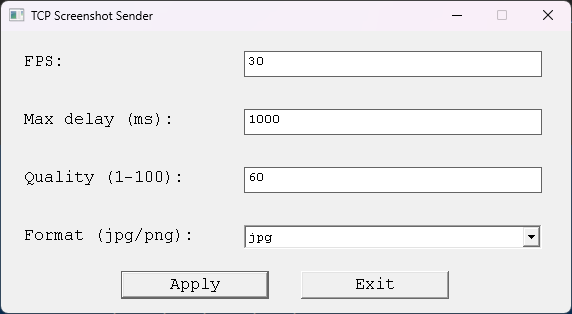


Рисунок 2.2.1.2 — Главное окно.

# Разработка программного средства

## Описание применения системного программировния

### Совместное использование WinAPI и многопоточности

В данной реализации используется WinAPI для взаимодействия с операционной системой, включая захват экрана, отрисовку удалённого экрана, создание окон и обработку сетевых запросов. Для передачи изображений используется UDP-протокол, а для отправки и получения событий мыши и клавиатуры — TCP-протокол.

Для эффективной работы в многозадачной среде применяются механизмы многопоточности, что позволяет параллельно обрабатывать пользовательский интерфейс, захват или отрисовку экрана, а также отправку или получение UDP-пакетов.

**Для создания интерфейса клиента используется WinAPI, который позволяет динамично добавлять элементы управления, такие как метки, текстовые поля и кнопки.** Пример реализации этого процесса в коде представлен ниже:

void MainForm::createControls(const HWND hwnd) {

const ControlCreator controlCreator(hwnd, \_fonts);

int currentY = 20;

controlCreator.createLabel(currentY, L"FPS:");

\_hEditFps = controlCreator.createEditControl(currentY, std::to\_wstring(\_config.fps));

currentY += ControlCreator::LABEL\_HEIGHT + ControlCreator::Y\_OFFSET;

controlCreator.createLabel(currentY, L"Max delay (ms):");

\_hEditMaxDelay = controlCreator.createEditControl(currentY, std::to\_wstring(\_config.maxDelayMs));

currentY += ControlCreator::LABEL\_HEIGHT + ControlCreator::Y\_OFFSET;

controlCreator.createLabel(currentY, L"Quality (1-100):");

\_hEditQuality = controlCreator.createEditControl(currentY, std::to\_wstring(\_config.imageConfig.quality));

currentY += ControlCreator::LABEL\_HEIGHT + ControlCreator::Y\_OFFSET;

controlCreator.createLabel(currentY, L"Format (jpg/png):");

\_hComboBoxFormat = controlCreator.createComboBox(currentY, {L"jpg", L"webp"}, static\_cast<int>(\_config.imageConfig.ext));

currentY += ControlCreator::CONTROL\_HEIGHT + ControlCreator::BUTTON\_Y\_OFFSET;

controlCreator.createDefButton(currentY, L"Apply", BTN\_APPLY\_ID);

controlCreator.createButton(currentY, L"Exit", BTN\_EXIT\_ID);

}

HWND ControlCreator::createLabel(const int y, const std::wstring& text) const {

const HWND hLabel = CreateWindow(L"STATIC", text.c\_str(), WS\_VISIBLE | WS\_CHILD, X\_OFFSET\_LABEL, y, LABEL\_WIDTH, LABEL\_HEIGHT, \_hwnd, nullptr, nullptr, nullptr);

SendMessage(hLabel, WM\_SETFONT, reinterpret\_cast<WPARAM>(\_fonts.hLabelFont), TRUE);

return hLabel;

}

HWND ControlCreator::createEditControl( const int y, const std::wstring& text) const {

const HWND hEdit = CreateWindow(L"EDIT", text.c\_str(), WS\_VISIBLE | WS\_CHILD | WS\_BORDER | ES\_NUMBER, X\_OFFSET\_CONTROL, y, CONTROL\_WIDTH, CONTROL\_HEIGHT, \_hwnd, nullptr, nullptr, nullptr);

SendMessage(hEdit, WM\_SETFONT, reinterpret\_cast<WPARAM>(\_fonts.hControlFont), TRUE);

return hEdit;

}

HWND ControlCreator::createComboBox(const int y, const std::vector<std::wstring>& items, const int selectedIndex) const {

const HWND hComboBox = CreateWindow(L"COMBOBOX", nullptr, WS\_VISIBLE | WS\_CHILD | CBS\_DROPDOWNLIST, X\_OFFSET\_CONTROL, y, CONTROL\_WIDTH, COMBO\_BOX\_HEIGHT, \_hwnd, nullptr, nullptr, nullptr);

for (const auto& item : items) {

SendMessage(hComboBox, CB\_ADDSTRING, 0, reinterpret\_cast<LPARAM>(item.c\_str()));

}

SendMessage(hComboBox, CB\_SETCURSEL, static\_cast<WPARAM>(selectedIndex), 0);

SendMessage(hComboBox, WM\_SETFONT, reinterpret\_cast<WPARAM>(\_fonts.hControlFont), TRUE);

return hComboBox;

}

HWND ControlCreator::createDefButton(const int y, const std::wstring& text, const int buttonId) const {

const HWND hButton = CreateWindow(L"BUTTON", text.c\_str(), WS\_VISIBLE | WS\_CHILD | BS\_DEFPUSHBUTTON, X\_OFFSET\_DEF\_BUTTON, y, BUTTON\_WIDTH, BUTTON\_HEIGHT, \_hwnd, reinterpret\_cast<HMENU>(buttonId), nullptr, nullptr);

SendMessage(hButton, WM\_SETFONT, reinterpret\_cast<WPARAM>(\_fonts.hLabelFont), TRUE);

return hButton;

}

HWND ControlCreator::createButton(const int y, const std::wstring& text, const int buttonId) const {

const HWND hButton = CreateWindow(L"BUTTON", text.c\_str(), WS\_VISIBLE | WS\_CHILD, X\_OFFSET\_BUTTON, y, BUTTON\_WIDTH, BUTTON\_HEIGHT, \_hwnd, reinterpret\_cast<HMENU>(buttonId), nullptr, nullptr);

SendMessage(hButton, WM\_SETFONT, reinterpret\_cast<WPARAM>(\_fonts.hLabelFont), TRUE);

return hButton;

}

Примером многопоточности является использование рабочего потока для обработки и отправки данных через UDP. Поток управляется методами для запуска, выполнения основного цикла обработки и корректного завершения, обеспечивая асинхронную отправку данных из буфера. Ниже представлен код, демонстрирующий этот подход:

void UDPSenderWorker::start() {

if (!\_running) {

\_running = true;

\_eventThread = std::thread(&ThreadWorker::eventLoop, this);

}

}

void UDPSenderWorker::eventLoop() {

while (\_running) {

if (const auto buffer = \_buffer.dequeue()) {

\_sender.send(\*buffer);

}

}

}

void UDPSenderWorker::stopRunning() {

if (\_running) {

\_running = false;

}

}

void UDPSenderWorker::waitForThread() {

if (\_eventThread.joinable()) {

\_eventThread.join();

}

}

Для захвата экрана используется последовательность операций с использованием WinAPI, что позволяет получить изображение рабочего стола в виде массива байт. Пример кода, реализующего этот процесс, приведен ниже:

std::vector<byte> ScreenCapture::capture() {

\_bitmap.copyFrom(\_dc.getHDC());

return \_bitmap.getDIBits();

}

void Bitmap::copyFrom(const HDC src) const {

BitBlt(\_hMemoryDc, 0, 0, \_size.cx, \_size.cy, src, 0, 0, SRCCOPY);

}

std::vector<byte> Bitmap::getDIBits() {

std::vector<byte> buffer(\_bi.biSizeImage);

GetDIBits(\_hMemoryDc, \_hMemoryBitmap, 0, \_size.cy,

buffer.data(), reinterpret\_cast<BITMAPINFO\*>(&\_bi), DIB\_RGB\_COLORS);

return buffer;

}

**Для отображения захваченного экрана используется WinAPI, которое позволяет отобразить изображение из массива байт на экране.** Код, который реализует вывод изображения, представлен ниже:

void ScreenRender::render(const std::vector<byte>& screen) {

\_bitmap.setDIBits(screen);

\_bitmap.copyTo(\_dc.getHDC());

}

void Bitmap::setDIBits(const std::vector<byte>& data) const {

SetDIBits(\_hMemoryDc, \_hMemoryBitmap, 0, \_size.cy,

data.data(), reinterpret\_cast<const BITMAPINFO\*>(&\_bi), DIB\_RGB\_COLORS);

}

void Bitmap::copyTo(const HDC dest) const {

BitBlt(dest, 0, 0, \_size.cx, \_size.cy, \_hMemoryDc, 0, 0, SRCCOPY);

}

Пример кода ниже демонстрирует работу с сокетами для реализации сетевого общения. Класс Socket инкапсулирует все основные операции с сокетами, такие как создание, привязка, прослушивание, принятие соединений и обмен данными с использованием различных методов передачи

Socket::Socket(const SOCKET socket): \_sock(socket) {

SocketErrorChecker::checkSocket(\_sock);

}

Socket::Socket(const int type, const int protocol): \_sock(socket(AF\_INET, type, protocol)) {

SocketErrorChecker::checkSocket(\_sock);

}

void Socket::bindSocket(const u\_short port) const {

const sockaddr\_in serverAddr = SockaddrUtils::createAddrAnyInterface(port);

const int result = bind(\_sock, reinterpret\_cast<const sockaddr\*>(&serverAddr), sizeof(serverAddr));

SocketErrorChecker::checkBindError(result);

}

void Socket::listenOnSocket(const int backlog) const {

const int result = listen(\_sock, backlog);

SocketErrorChecker::checkListenError(result);

}

Socket Socket::acceptConnection(sockaddr\_in\* clientAddr, int\* clientAddrSize) const {

const SOCKET clientSock = accept(\_sock, reinterpret\_cast<sockaddr\*>(clientAddr), clientAddrSize);

SocketErrorChecker::checkAcceptError(clientSock);

return Socket(clientSock);

}

void Socket::connectToServer(const sockaddr\_in& serverAddr) const {

const int result = connect(\_sock, reinterpret\_cast<const sockaddr\*>(&serverAddr), sizeof(serverAddr));

SocketErrorChecker::checkConnectError(result);

}

int Socket::sendSocket(const std::vector<byte>& buffer) const {

const int result = send(\_sock, reinterpret\_cast<const char\*>(buffer.data()), buffer.size(), 0);

return SocketErrorChecker::checkSend(result);

}

int Socket::sendToSocket(const std::vector<byte>& buffer, const sockaddr\_in& destAddr) const {

const int result = sendto(\_sock, reinterpret\_cast<const char\*>(buffer.data()), buffer.size(), 0,

reinterpret\_cast<const sockaddr\*>(&destAddr), sizeof(destAddr));

return SocketErrorChecker::checkSend(result);

}

int Socket::recvSocket(std::vector<byte>& buffer) const {

const int len = recv(\_sock, reinterpret\_cast<char\*>(buffer.data()), buffer.size(), 0);

return SocketErrorChecker::checkReceive(len);

}

int Socket::recvFromSocket(std::vector<byte>& buffer, sockaddr\_in\* senderAddr, int\* senderAddrSize) const {

const int len = recvfrom(\_sock, reinterpret\_cast<char\*>(buffer.data()), buffer.size(), 0,

reinterpret\_cast<sockaddr\*>(senderAddr), senderAddrSize);

return SocketErrorChecker::checkReceive(len);

}

## Описание применения элементов системного программирования

### WinAPI

WinAPI (Windows API) представляет собой набор функций, предоставляемых операционной системой Windows, для взаимодействия с системными ресурсами, такими как окна, графика, устройства ввода и другие. В данной реализации WinAPI используется для создания оконного интерфейса приложения, который позволяет пользователю взаимодействовать с функционалом программы.

Пример реализации работы с оконным интерфейсом в проекте представлен в классе MainForm. Этот класс отвечает за регистрацию оконного класса, создание окна и обработку сообщений.

void MainForm::registerClass(const HINSTANCE

hInstance){

\_hInstance = hInstance;

WNDCLASSEX wc = {};

wc.cbSize = sizeof(WNDCLASSEX);

wc.lpfnWndProc = windowProc;

wc.hInstance = hInstance;

wc.lpszClassName = WINDOWS\_CLASS\_NAME;

wc.hbrBackground = reinterpret\_cast<HBRUSH>(COLOR\_WINDOW);

RegisterClassEx(&wc);

}

HWND MainForm::createHwnd() {

return CreateWindowEx(0, WINDOWS\_CLASS\_NAME, L"TCP Screenshot Sender",

WS\_OVERLAPPED | WS\_CAPTION | WS\_SYSMENU | WS\_MINIMIZEBOX,

(GetSystemMetrics(SM\_CXSCREEN) - 586) / 2, (GetSystemMetrics(SM\_CYSCREEN) - 321) / 2,

586, 321, nullptr, nullptr, \_hInstance, this);

}

void MainForm::show() const {

ShowWindow(\_hwnd, SW\_SHOW);

\_sender.run();

MSG msg;

while (GetMessage(&msg, nullptr, 0, 0)) {

TranslateMessage(&msg);

DispatchMessage(&msg);

}

}

LRESULT MainForm::windowProc(const HWND hwnd, const UINT uMsg, const WPARAM wParam, const LPARAM lParam) {

switch (uMsg) {

case WM\_CREATE: {

const CREATESTRUCT\* pCreateStruct = reinterpret\_cast<CREATESTRUCT\*>(lParam);

MainForm\* mainForm = static\_cast<MainForm\*>(pCreateStruct->lpCreateParams);

SetWindowLongPtr(hwnd, GWLP\_USERDATA, reinterpret\_cast<LONG\_PTR>(mainForm));

mainForm->createControls(hwnd);

break;

}

case WM\_COMMAND: {

switch (LOWORD(wParam)) {

case BTN\_APPLY\_ID: {

MainForm\* mainForm = reinterpret\_cast<MainForm\*>(GetWindowLongPtr(hwnd, GWLP\_USERDATA));

mainForm->updateConfig();

MessageBox(hwnd, L"Settings applied successfully!", L"Info", MB\_OK | MB\_ICONINFORMATION);

break;

}

case BTN\_EXIT\_ID:

PostMessage(hwnd, WM\_CLOSE, 0, 0);

break;

}

break;

}

case WM\_DESTROY: {

const MainForm\* mainForm = reinterpret\_cast<MainForm\*>(GetWindowLongPtr(hwnd, GWLP\_USERDATA));

mainForm->\_sender.stop();

PostQuitMessage(0);

break;

}

}

return DefWindowProc(hwnd, uMsg, wParam, lParam);

}

### Работа с шрифтами

Шрифты в WinAPI используются для управления отображением текста в графическом интерфейсе пользователя. Они позволяют задавать параметры, такие как размер, стиль и семейство шрифта, для обеспечения читаемости и эстетичности пользовательского интерфейса. В данном проекте шрифты применяются для выделения текстовых элементов, таких как метки и элементы управления.

В проекте для управления шрифтами используется структура Fonts. Она предоставляет удобный интерфейс для создания шрифтов, которые могут быть применены к различным элементам интерфейса.

struct Fonts {

HFONT hLabelFont;

HFONT hControlFont;

explicit Fonts(const int labelFontSize = 20, const int controlFontSize = 16, const LPCWSTR fontName = L"Courier New");

~Fonts();

};

Fonts::Fonts(const int labelFontSize, const int controlFontSize, const LPCWSTR fontName):

hLabelFont(CreateFont(labelFontSize, 0, 0, 0,

FW\_NORMAL, FALSE, FALSE, FALSE, DEFAULT\_CHARSET,

OUT\_DEFAULT\_PRECIS, CLIP\_DEFAULT\_PRECIS, DEFAULT\_QUALITY,

FF\_SWISS, fontName)),

hControlFont(CreateFont(controlFontSize, 0, 0, 0,

FW\_NORMAL, FALSE, FALSE, FALSE, DEFAULT\_CHARSET,

OUT\_DEFAULT\_PRECIS, CLIP\_DEFAULT\_PRECIS, DEFAULT\_QUALITY,

FF\_SWISS, fontName)) { }

Fonts::~Fonts() {

DeleteObject(hLabelFont);

DeleteObject(hControlFont);

}

### Работа с реестром

Реестр Windows предоставляет централизованное хранилище для хранения настроек и параметров программ, таких как конфигурации пользователя, состояния интерфейса и предпочтения. Использование реестра позволяет сохранять настройки между сеансами работы программы, обеспечивая удобство для пользователей.

В данном проекте взаимодействие с реестром реализовано с помощью класса RegistrySettings. Он предоставляет методы для сохранения и загрузки конфигураций с использованием функций WinAPI.

void RegistrySettings::saveSettingsToRegistry(const SenderConfig& config) {

HKEY hKey;

if (RegCreateKeyEx(HKEY\_CURRENT\_USER, REGISTRY\_PATH, 0, nullptr, 0, KEY\_WRITE, nullptr, &hKey, nullptr) == ERROR\_SUCCESS) {

setRegistryValue(hKey, KEY\_PORT, config.tcpServerPort);

setRegistryValue(hKey, KEY\_FPS, static\_cast<DWORD>(config.fps));

setRegistryValue(hKey, KEY\_MAX\_DELAY\_MS, static\_cast<DWORD>(config.maxDelayMs));

setRegistryValue(hKey, KEY\_QUALITY, static\_cast<DWORD>(config.imageConfig.quality));

setRegistryValue(hKey, KEY\_WIDTH, static\_cast<DWORD>(config.targetSize.cx));

setRegistryValue(hKey, KEY\_HEIGHT, static\_cast<DWORD>(config.targetSize.cy));

setRegistryValue(hKey, KEY\_FORMAT, static\_cast<DWORD>(config.imageConfig.ext));

RegCloseKey(hKey);

}

}

void RegistrySettings::loadSettingsFromRegistry(SenderConfig& config) {

HKEY hKey;

if (RegOpenKeyEx(HKEY\_CURRENT\_USER, REGISTRY\_PATH, 0, KEY\_READ, &hKey) == ERROR\_SUCCESS) {

DWORD value;

if (getRegistryValue(hKey, KEY\_PORT, value))

config.tcpServerPort = static\_cast<u\_short>(value);

if (getRegistryValue(hKey, KEY\_FPS, value))

config.fps = static\_cast<int>(value);

if (getRegistryValue(hKey, KEY\_MAX\_DELAY\_MS, value))

config.maxDelayMs = static\_cast<int>(value);

if (getRegistryValue(hKey, KEY\_FORMAT, value))

config.imageConfig.ext = static\_cast<ImageFormat>(value);

if (getRegistryValue(hKey, KEY\_QUALITY, value))

config.imageConfig.quality = static\_cast<int>(value);

if (getRegistryValue(hKey, KEY\_WIDTH, value))

config.targetSize.cx = static\_cast<int>(value);

if (getRegistryValue(hKey, KEY\_HEIGHT, value))

config.targetSize.cy = static\_cast<int>(value);

RegCloseKey(hKey);

} else {

config = DEFAULT\_SENDER\_CONFIG;

}

}

bool RegistrySettings::setRegistryValue(const HKEY hKey, const LPCWSTR key, const DWORD value) {

return RegSetValueEx(hKey, key, 0, REG\_DWORD, reinterpret\_cast<const BYTE\*>(&value), sizeof(value)) == ERROR\_SUCCESS;

}

bool RegistrySettings::getRegistryValue(const HKEY hKey, const LPCWSTR key, DWORD& value) {

DWORD dataSize = sizeof(value);

return RegQueryValueEx(hKey, key, nullptr, nullptr, reinterpret\_cast<BYTE\*>(&value), &dataSize) == ERROR\_SUCCESS;

}

### Многопоточное программирование

Многопоточность в данной реализации используется для повышения производительности и обеспечения плавной работы системы в условиях многозадачности. Основная цель применения многопоточности — разделение задач на независимые потоки, чтобы избежать блокировки пользовательского интерфейса и минимизировать задержки при выполнении ресурсоёмких операций.

В приложении для предоставления удаленного доступа основной поток управляет пользовательским интерфейсом, обеспечивая его отзывчивость. Захват экрана выполняется в отдельном потоке, что позволяет непрерывно получать кадры и передавать их без задержек. Отправка данных по UDP также происходит в отдельном потоке, что обеспечивает постоянную передачу данных без блокировки других операций.

В приложении для получения удаленного доступа основной поток управляет интерфейсом, поддерживая его стабильную работу. Прием данных по UDP осуществляется в одном потоке, что позволяет эффективно обрабатывать поступающие данные, а обновление экрана для отображения полученной информации происходит в другом потоке, обеспечивая плавность и своевременность отображения данных.

## Схема алгоритмов решения задач по ГОСТ 19.701-90

### Схема алгоритма exchangeKeys



Рисунок 3.3.1.1 – Схема алгоритма exchangeKeys

### Схема алгоритма splitIntoChunks



Рисунок 3.3.2.1 – Схема алгоритма splitIntoChunks

### Схема алгоритма captureScreen



Рисунок 3.3.3.1 – Схема алгоритма captureScreen

### Схема алгоритма acceptSingleConnetion



Рисунок 3.3.4.1 – Схема алгоритма acceptSingleConnetion

# Тестирование и проверка работоспособности программного средства

## Запуск программного средства

### Тест 1

Таблица 4.1.1 – Тест 1

|  |  |
| --- | --- |
| Тестовая ситуация | Проверка корректного открытия окна при запуске программного средства для предоставления удаленного доступа |
| Исходный набор данных | Запуск программы |
| Ожидаемый результат | Корректное открытие окна |
| Полученный результат |  |

### Тест 2

Таблица 4.1.2 – Тест 2

|  |  |
| --- | --- |
| Тестовая ситуация | Проверка корректного открытия окна при запуске программного средства для получения удаленного доступа |
| Исходный набор данных | Запуск программы |
| Ожидаемый результат | Корректное открытие окна |
|  |  |

Продолжение таблицы 4.1.2

|  |  |
| --- | --- |
| Полученный результат |  |

## Проверка изменения настроек

### Тест 1

Таблица 4.2.1 – Тест 1

|  |  |
| --- | --- |
| Тестовая ситуация | Проверка корректности работы поля ввода порта в настройках программного обеспечения для предоставления удаленного доступа |
| Исходный набор данных | Попытка ввода нечисловых символов в поле для ввода порта |
| Ожидаемый результат | Поле ввода порта игнорирует нечисловые символы |
| Полученный результат | Поле для ввода порта корректно работает, не позволяя вводить буквы и специальные символы |

### Тест 2

Таблица 4.2.2 – Тест 2

|  |  |
| --- | --- |
| Тестовая ситуация | Проверка корректности работы поля ввода частоты кадров в настройках программного обеспечения для предоставления удаленного доступа |
| Исходный набор данных | Попытка ввода нечисловых символов в поле для ввода частоты кадров |
| Ожидаемый результат | Поле ввода частоты кадров игнорирует нечисловые символы |
| Полученный результат | Поле для ввода частоты кадров корректно работает, не позволяя вводить буквы и специальные символы |

### Тест 3

Таблица 4.2.3 – Тест 3

|  |  |
| --- | --- |
| Тестовая ситуация | Проверка корректности работы поля ввода максимальной задержки в настройках программного обеспечения для предоставления удаленного доступа |
| Исходный набор данных | Попытка ввода нечисловых символов в поле для ввода максимальной задержки |
| Ожидаемый результат | Поле ввода максимальной задержки игнорирует нечисловые символы |
| Полученный результат | Поле для ввода максимальной задержки корректно работает, не позволяя вводить буквы и специальные символы |

### Тест 4

Таблица 4.2.4 – Тест 4

|  |  |
| --- | --- |
| Тестовая ситуация | Проверка корректности работы поля ввода разрешения экрана в настройках программного обеспечения для предоставления удаленного доступа |
| Исходный набор данных | Попытка ввода нечисловых символов в поле для ввода разрешения экрана |
| Ожидаемый результат | Поле ввода разрешения экрана игнорирует нечисловые символы |
| Полученный результат | Поле для ввода разрешения экрана корректно работает, не позволяя вводить буквы и специальные символы |

### Тест 5

Таблица 4.2.5 – Тест 5

|  |  |
| --- | --- |
| Тестовая ситуация | Проверка корректности работы поля ввода разрешения экрана в настройках программного обеспечения для предоставления удаленного доступа |
| Исходный набор данных | Список доступных кодировок изображения (jpg и webp) |
| Ожидаемый результат | При изменении выбора индекс переключается на новый элемент, отображается правильная кодировка. |
| Полученный результат | Индекс корректно переключается при изменении выбора, выбранная кодировка отображается в поле. |

### Тест 6

Таблица 4.2.6 – Тест 6

|  |  |
| --- | --- |
| Тестовая ситуация | Проверка корректности работы поля ввода для порта сервера в настройках программного обеспечения для получения удаленного доступа |
| Исходный набор данных | Попытка ввода нечисловых символов в поле для ввода порта сервера |
| Ожидаемый результат | Поле ввода порта сервера игнорирует нечисловые символы |
| Полученный результат | Поле для ввода порта сервера корректно работает, не позволяя вводить буквы и специальные символы |

### Тест 7

Таблица 4.2.7 – Тест 7

|  |  |
| --- | --- |
| Тестовая ситуация | Проверка корректности работы поля ввода для UDP порта в настройках программного обеспечения для получения удаленного доступа |
| Исходный набор данных | Попытка ввода нечисловых символов в поле для ввода UDP порта |
| Ожидаемый результат | Поле ввода UDP порта игнорирует нечисловые символы |
| Полученный результат | Поле для ввода UDP порта корректно работает, не позволяя вводить буквы и специальные символы |

### Тест 8

Таблица 4.2.8 – Тест 8

|  |  |
| --- | --- |
| Тестовая ситуация | Проверка корректности работы поля ввода частоты кадров в настройках программного обеспечения для получения удаленного доступа |
| Исходный набор данных | Попытка ввода нечисловых символов в поле для ввода частоты кадров |
| Ожидаемый результат | Поле ввода частоты кадров игнорирует нечисловые символы |
| Полученный результат | Поле для ввода частоты кадров корректно работает, не позволяя вводить буквы и специальные символы |

### Тест 9

Таблица 4.2.9 – Тест 9

|  |  |
| --- | --- |
| Тестовая ситуация | Проверка корректности работы поля ввода максимальной задержки в настройках программного обеспечения для получения удаленного доступа |
| Исходный набор данных | Попытка ввода нечисловых символов в поле для ввода максимальной задержки |
| Ожидаемый результат | Поле ввода максимальной задержки игнорирует нечисловые символы |
| Полученный результат | Поле для ввода максимальной задержки корректно работает, не позволяя вводить буквы и специальные символы |

### Тест 10

Таблица 4.2.10 – Тест 10

|  |  |
| --- | --- |
| Тестовая ситуация | Проверка корректности работы программного обеспечения для получения удаленного доступа при некорректных данных |
| Исходный набор данных | Попытка ввода некорректных данных |
| Ожидаемый результат | Сообщение об ошибке |
| Полученный результат |  |

### Тест 11

Таблица 4.2.11 – Тест 11

|  |  |
| --- | --- |
| Тестовая ситуация | Проверка корректности работы программного обеспечения для предоставления удаленного доступа при некорректных данных |
| Исходный набор данных | Попытка ввода некорректных данных |
| Ожидаемый результат | Сообщение об ошибке |
| Полученный результат |  |

## Сохранение настроек в реестре

### Тест 1

Таблица 4.3.1 – Тест 1

|  |  |
| --- | --- |
| Тестовая ситуация | Проверка корректности работы программного обеспечения для получения удаленного доступа при сохранении настроек в реестре |
| Исходный набор данных | Порт сервера = 8080; Разрешение экрана = 1920x1080; Формат = 0; Качество = 60; Максимальная задержка = 1000; Частота кадров = 30 |
| Ожидаемый результат | Настройки сохранены в реестре |
| Полученный результат |  |

### Тест 2

Таблица 4.3.2 – Тест 2

|  |  |
| --- | --- |
| Тестовая ситуация | Проверка корректности работы программного обеспечения для предоставления удаленного доступа при сохранении настроек в реестре |
| Исходный набор данных | Адрес сервера = 0x0100007f; Порт сервера = 8080; UDP порт = 8081; Максимальная задержка = 1000; Частота кадров = 30 |
| Ожидаемый результат | Настройки сохранены в реестре |
| Полученный результат |  |

## Демонстрация экрана

### Тест 1

Таблица 4.4.1 – Тест 1

|  |  |
| --- | --- |
| Тестовая ситуация | Проверка корректности работы программного обеспечения для получения удаленного доступа при получении экрана |
| Исходный набор данных | Порт сервера = 8080; Разрешение экрана = 1920x1080; Формат = 0; Качество = 60; Максимальная задержка = 1000; Частота кадров = 30 |
| Ожидаемый результат | Показ экрана |
| Полученный результат |  |

### Тест 2

Таблица 4.4.2 – Тест 2

|  |  |
| --- | --- |
| Тестовая ситуация | Проверка корректности работы программного обеспечения для предоставления удаленного доступа при передаче экрана |
| Исходный набор данных | Адрес сервера = 0x0100007f; Порт сервера = 8080; UDP порт = 8081; Максимальная задержка = 1000; Частота кадров = 30 |
| Ожидаемый результат | Экран передается |
| Полученный результат |  |

## Отправка событий

### Тест 1

Таблица 4.4.1 – Тест 1

|  |  |
| --- | --- |
| Тестовая ситуация | Проверка корректности работы программного обеспечения для предоставления удаленного доступа при получении события нажатия левой кнопки мыши |
| Исходный набор данных | Нажатие левой кнопки мыши в программного обеспечении для получения удаленного доступа |
| Ожидаемый результат | Нажатие левой кнопки мыши |

Продолжение таблицы 4.4.1

|  |  |
| --- | --- |
| Полученный результат |  |

### Тест 2

Таблица 4.4.2 – Тест 2

|  |  |
| --- | --- |
| Тестовая ситуация | Проверка корректности работы программного обеспечения для предоставления удаленного доступа при получении события нажатия клавиши Alt |
| Исходный набор данных | Нажатие клавиши Alt в программного обеспечении для получения удаленного доступа |
| Ожидаемый результат | Нажатие клавиши Alt |
| Полученный результат |  |

# Руководство по запуску и использованию программного средства

## Запуск программы

### Запуск программы для предоставления удаленного доступа

При запуске программы открывается главное окно настроек соединения (рисунок 5.1.1). Оно содержит поля для ввода и настройки параметров подключения:

* TCP Server Port – порт для обмена управляющими командами (события мыши и клавиатуры);
* FPS – частота кадров для передачи изображения экрана;
* Max delay (ms) – максимальная задержка для обработки пакетов;
* Quality (1-100) – качество передаваемого изображения (чем выше значение, тем выше качество и объем данных);
* Screen Width и Screen Height – ширина и высота экрана для передачи, соответствующие разрешению удаленного экрана;
* Format (jpg/png) – выбор формата сжатия изображения (JPEG или PNG).

После ввода параметров пользователь может нажать кнопку OK, чтобы сохранить настройки и запустить сервер для предоставления удаленного доступа. Если необходимо выйти, достаточно нажать кнопку Cancel.

Программа готова к использованию сразу после установки параметров. Сервер начинает слушать входящие подключения, ожидая подключения клиента для просмотра экрана.

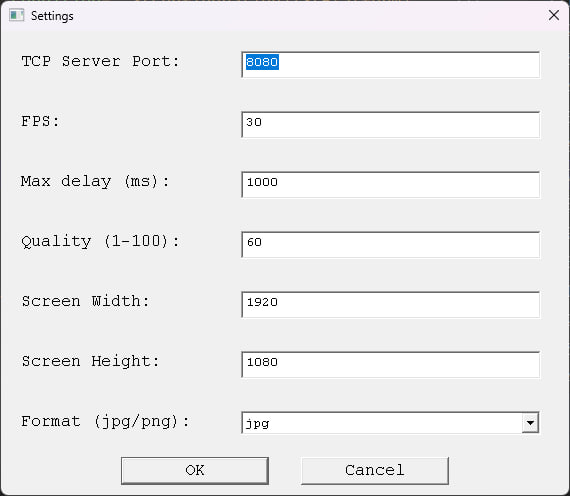


рисунок 5.1.1 — Диалоговое окно настроек соединения

### Запуск программы для получения удаленного доступа

При запуске программы открывается окно настроек подключения (рисунок 5.1.2). Пользователь может настроить параметры для подключения к удаленному компьютеру:

* Server IP – IP-адрес сервера, предоставляющего удаленный доступ;
* Server Port – порт TCP для соединения с сервером (используется для передачи управляющих событий, таких как действия мыши и клавиатуры);
* UDP Port – порт UDP для получения изображений экрана;
* FPS – частота кадров, задающая плавность отображаемого изображения;
* Max delay (ms) – максимальная допустимая задержка для передачи данных.

После ввода необходимых параметров пользователь может нажать OK для сохранения настроек и начала подключения к серверу. Если необходимо выйти, достаточно нажать Cancel.

Программа начнет попытку установить соединение с указанным сервером, после чего пользователь сможет просматривать экран удаленного компьютера в реальном времени и управлять им с помощью мыши и клавиатуры.

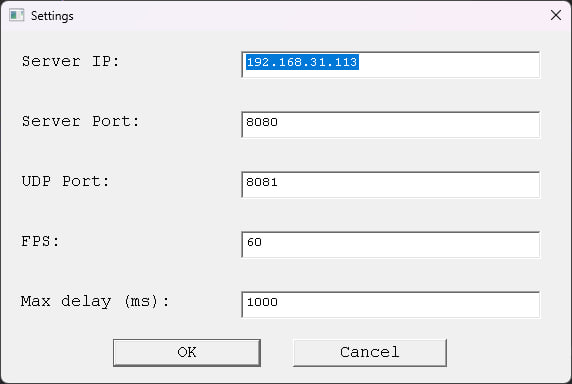


рисунок 5.1.2 — Диалоговое окно настроек подключения

## Использование

### Использование программы для предоставления удаленного доступа

После запуска программы и подтверждения настроек с помощью кнопки OK (рисунок 5.1.1), создается сервер, который ожидает подключения клиента.

Как только клиент подключится, программа начнет передачу изображения экрана. В процессе работы пользователь, предоставляющий доступ, может изменить настройки передачи изображения, открыв соответствующее окно настроек (рисунок 5.2.1).

В настройках доступны следующие параметры:

* FPS – частота кадров для отображения изображения;
* Max delay (ms) – максимальная задержка передачи данных;
* Quality (1-100) – качество изображения, передаваемого по сети;
* Format (jpg/png) – выбор формата изображения.

Для применения изменений настроек необходимо нажать Apply, после чего параметры будут применены без прерывания работы программы. Кнопка Exit завершает работу программы и отключает сервер.

Таким образом, программа позволяет гибко настраивать параметры передачи изображения, обеспечивая удобство использования при различных сетевых условиях.

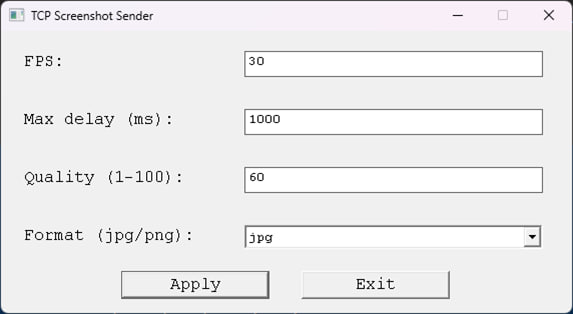


рисунок 5.2.1 – Окно программы

### Использование программы для получения удаленного доступа

После успешного соединения с сервером программа отображает экран собеседника (рисунок 5.2.2) в режиме реального времени. Интерфейс программы автоматически адаптируется для удобства просмотра изображения, передаваемого сервером.

Для управления параметрами подключения и режимами отображения доступны следующие горячие клавиши:

* Fn + F1 – вызывает окно настроек, где пользователь может изменить параметры подключения, такие как частота кадров и максимальная задержка.
* Fn + F12 – переключает режим отображения между оконным и полноэкранным, что позволяет пользователю выбрать наиболее удобный способ просмотра экрана собеседника.

Программа обеспечивает стабильное соединение и удобство использования, предоставляя пользователю необходимые инструменты для управления подключением и настройки параметров отображения.

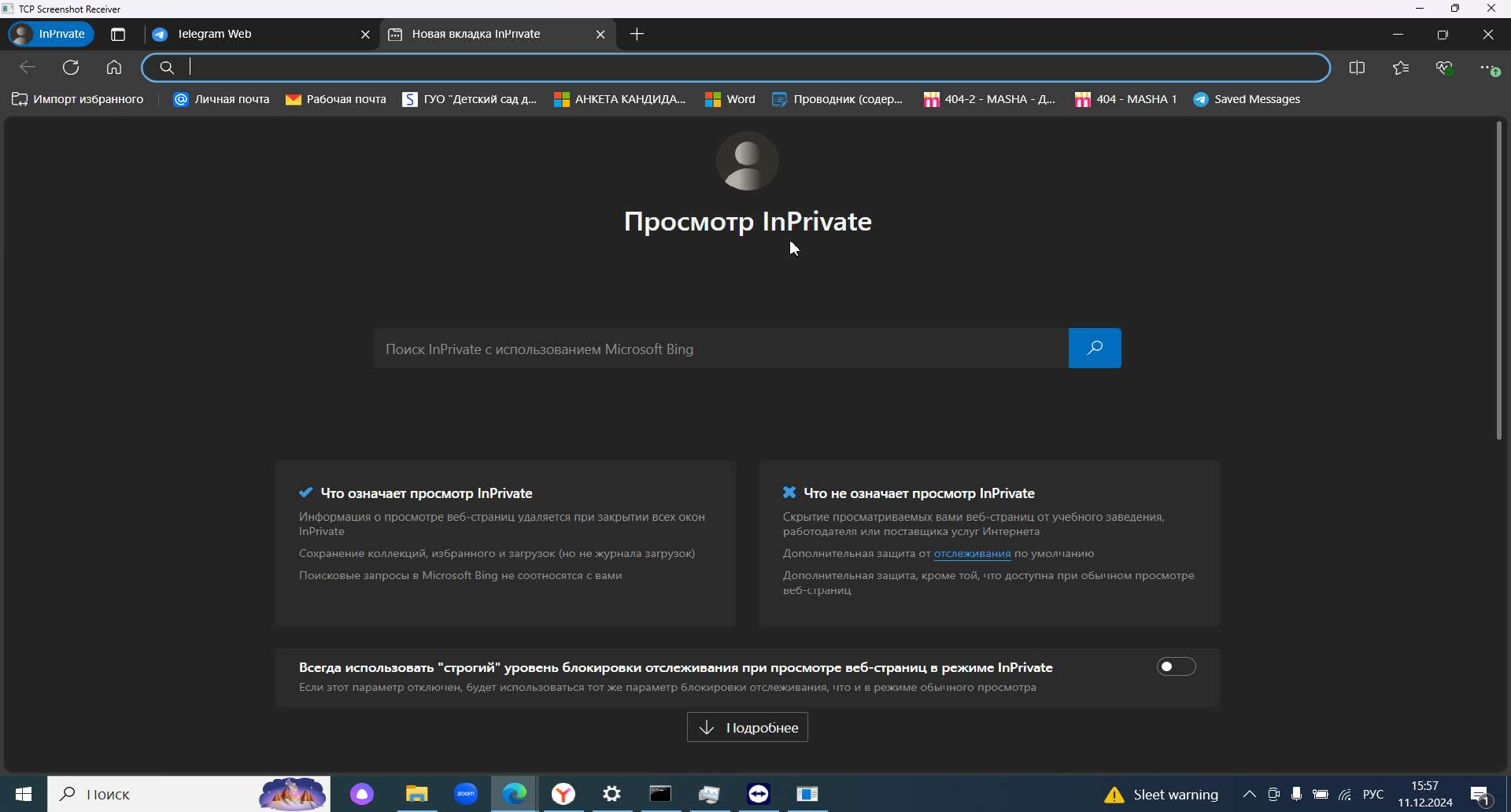


Рисунок 5.2.2 – Главное окно

# Заключение

По итогам работы над курсовым проектом было разработано программное средство для передачи и просмотра удаленного экрана с использованием протокола UDP. Основной упор был сделан на использование возможностей WinAPI и сторонних библиотек, таких как OpenCV и CryptoPP, а также на проработку эффективных алгоритмов передачи данных, учитывающих особенности протокола UDP.

Программа прошла этапы тестирования, продемонстрировав стабильную работу. Были обработаны исключительные ситуации. Все функции, включая захват экрана, передачу фрагментов, сборку изображения и его отображение, работают без сбоев.

Программное средство включает следующие ключевые возможности:

* два клиента: один для предоставления доступа к своему экрану, другой для просмотра удаленного рабочего стола;
* гибкие настройки подключения, включая выбор IP-адреса, портов TCP и UDP, частоты кадров, а также области захвата экрана;
* поддержка передачи данных через UDP с механизмом восстановления порядка и обработки потерянных пакетов;
* передача событий мыши и клавиатуры по TCP, что обеспечивает взаимодействие с удаленным рабочим столом;
* высококачественное масштабирование с использованием HALFTONE для отображения экрана;
* горячие клавиши для переключения между оконным и полноэкранным режимами;
* сохранение пользовательских настроек для быстрого доступа при последующем использовании программы.

Данный проект отличается продуманной структурой, удобным интерфейсом и стабильной работой. Среди его преимуществ можно отметить минималистичный дизайн, высокую производительность передачи данных, поддержку широкого диапазона разрешений и настройку параметров отображения.

Разработанное программное средство может быть полезно как для демонстрации рабочего процесса, так и для удаленной технической поддержки, предоставляя пользователям эффективный инструмент для работы в реальном времени с возможностью управления удаленным рабочим столом.

Список использованной литературы

[1] Рихтер Д. Создание эффективных WIN32-приложений с учетом специфики 64-разрядной версии Windows. – М.: Microsoft Press, 2001. – 800 с.: ил.

[2] Руссинович М., Соломон Д. Внутреннее устройство Microsoft Windows. – 5-е изд. – М.: Microsoft Press, 2009. – 1248 с.: ил.

[3] Dabak P., Phadke S., Borate M. Undocumented Windows NT. – 1st ed. – New Riders, 1999. – 608 p.: ил.

[4] Nebbett G. Windows NT, 2000 Native API Reference. – M.: Sams Publishing, 2000. – 480 p.: ил.

[5] Корнеев В., Киселев А. Современные микропроцессоры. – 4-е изд. – М.: Горячая Линия - Телеком, 2020. – 784 с.: ил.

Приложение А

(обязательное)

Исходный код программы (модуль Bitmap.hpp)

class Bitmap {

friend class BitmapFactory;

HDC \_hMemoryDc;

HBITMAP \_hMemoryBitmap;

HGDIOBJ \_oldBitmap;

SIZE \_size;

BITMAPINFOHEADER \_bi;

Bitmap(const HDC hScreenDc, const HBITMAP hMemoryBitmap, const SIZE& size, const BITMAPINFOHEADER& bi);

public:

void copyFrom(const HDC src) const;

void copyTo(const HDC dest) const;

std::vector<byte> getDIBits();

void setDIBits(const std::vector<byte>& data) const;

void drawIcon(const Icon& icon) const;

const SIZE& getSize() const;

~Bitmap();

Bitmap(Bitmap&&) noexcept;

Bitmap& operator=(Bitmap&&) noexcept;

Bitmap(const Bitmap&) = delete;

Bitmap& operator=(const Bitmap&) = delete;

private:

void releaseResources() const noexcept;

void resetResources() noexcept;

};

Bitmap::Bitmap(const HDC hScreenDc, const HBITMAP hMemoryBitmap, const SIZE& size, const BITMAPINFOHEADER& bi):

\_hMemoryDc(CreateCompatibleDC(hScreenDc)), \_hMemoryBitmap(hMemoryBitmap),

\_oldBitmap(SelectObject(\_hMemoryDc, \_hMemoryBitmap)), \_size(size), \_bi(bi) { }

void Bitmap::copyFrom(const HDC src) const {

BitBlt(\_hMemoryDc, 0, 0, \_size.cx, \_size.cy, src, 0, 0, SRCCOPY);

}

void Bitmap::copyTo(const HDC dest) const {

BitBlt(dest, 0, 0, \_size.cx, \_size.cy, \_hMemoryDc, 0, 0, SRCCOPY);

}

std::vector<byte> Bitmap::getDIBits() {

std::vector<byte> buffer(\_bi.biSizeImage);

GetDIBits(\_hMemoryDc, \_hMemoryBitmap, 0, \_size.cy,

buffer.data(), reinterpret\_cast<BITMAPINFO\*>(&\_bi), DIB\_RGB\_COLORS);

return buffer;

}

void Bitmap::setDIBits(const std::vector<byte>& data) const {

SetDIBits(\_hMemoryDc, \_hMemoryBitmap, 0, \_size.cy,

data.data(), reinterpret\_cast<const BITMAPINFO\*>(&\_bi), DIB\_RGB\_COLORS);

}

void Bitmap::drawIcon(const Icon& icon) const {

DrawIcon(\_hMemoryDc, icon.point.x, icon.point.y, icon.hIcon);

}

const SIZE& Bitmap::getSize() const {

return \_size;

}

Bitmap::~Bitmap() {

releaseResources();

}

Bitmap::Bitmap(Bitmap&& other) noexcept:

\_hMemoryDc(other.\_hMemoryDc),

\_hMemoryBitmap(other.\_hMemoryBitmap),

\_oldBitmap(other.\_oldBitmap),

\_size(other.\_size),

\_bi(other.\_bi) {

other.resetResources();

}

Bitmap& Bitmap::operator=(Bitmap&& other) noexcept {

if (this != &other) {

releaseResources();

\_hMemoryDc = other.\_hMemoryDc;

\_hMemoryBitmap = other.\_hMemoryBitmap;

\_oldBitmap = other.\_oldBitmap;

\_size = other.\_size;

\_bi = other.\_bi;

other.resetResources();

}

return \*this;

}

void Bitmap::releaseResources() const noexcept {

if (\_hMemoryDc) {

SelectObject(\_hMemoryDc, \_oldBitmap);

DeleteDC(\_hMemoryDc);

}

DeleteObject(\_hMemoryBitmap);

}

void Bitmap::resetResources() noexcept {

\_hMemoryDc = nullptr;

\_hMemoryBitmap = nullptr;

\_oldBitmap = nullptr;

}

Приложение Б

(обязательное)

Исходный код программы (модуль BitmapFactory.hpp)

class BitmapFactory {

static constexpr DWORD BYTES\_PER\_PIXEL = 4;

public:

BitmapFactory() = delete;

static Bitmap createDDBitmap(const HDC hScreenDc, const SIZE& size);

static Bitmap createDIBitmap(const HDC hScreenDc, const SIZE& size);

static DWORD calcBitmapSize(const SIZE& size);

private:

static BITMAPINFOHEADER createBitmapInfoHeader(const SIZE& size);

};

Bitmap BitmapFactory::createDDBitmap(const HDC hScreenDc, const SIZE& size) {

const BITMAPINFOHEADER bi = createBitmapInfoHeader(size);

const HBITMAP hBitmap = CreateCompatibleBitmap(hScreenDc, size.cx, size.cy);

return {hScreenDc, hBitmap, size, bi};

}

Bitmap BitmapFactory::createDIBitmap(const HDC hScreenDc, const SIZE& size) {

const BITMAPINFOHEADER bi = createBitmapInfoHeader(size);

const HBITMAP hBitmap = CreateDIBitmap(hScreenDc, &bi, CBM\_INIT, nullptr,

reinterpret\_cast<const BITMAPINFO\*>(&bi), DIB\_RGB\_COLORS);

return {hScreenDc, hBitmap, size, bi};

}

BITMAPINFOHEADER BitmapFactory::createBitmapInfoHeader(const SIZE& size) {

BITMAPINFOHEADER bi = {};

bi.biSize = sizeof(BITMAPINFOHEADER);

bi.biWidth = size.cx;

bi.biHeight = -size.cy;

bi.biPlanes = 1;

bi.biBitCount = BYTES\_PER\_PIXEL \* 8;

bi.biCompression = BI\_RGB;

bi.biSizeImage = calcBitmapSize(size);

return bi;

}

DWORD BitmapFactory::calcBitmapSize(const SIZE& size) {

return BYTES\_PER\_PIXEL \* size.cx \* size.cy;

}

Приложение В

(обязательное)

Исходный код программы (модуль CursorCapture.hpp)

class CursorCapture {

std::map<HICON, POINT> \_cachedHotspots;

public:

std::optional<Icon> capture();

private:

static CURSORINFO getCursorInfo();

static Icon createCursorIcon(const CURSORINFO& cursorInfo, const POINT& hotspot);

POINT\* getHotspotFromCache(const HICON hIcon);

static std::optional<POINT> getHotspotFromIconInfo(const HICON hIcon);

static void releaseIconInfoResources(const ICONINFO& iconInfo) noexcept;

};

std::optional<Icon> CursorCapture::capture() {

const CURSORINFO cursorInfo = getCursorInfo();

if (cursorInfo.flags == CURSOR\_SHOWING) {

if (const POINT\* hotspot = getHotspotFromCache(cursorInfo.hCursor)) {

return createCursorIcon(cursorInfo, \*hotspot);

}

}

return std::nullopt;

}

CURSORINFO CursorCapture::getCursorInfo() {

CURSORINFO cursorInfo;

cursorInfo.cbSize = sizeof(CURSORINFO);

if (!GetCursorInfo(&cursorInfo)) {

cursorInfo.flags = 0;

}

return cursorInfo;

}

Icon CursorCapture::createCursorIcon(const CURSORINFO& cursorInfo, const POINT& hotspot) {

return {

cursorInfo.hCursor,

{

cursorInfo.ptScreenPos.x - hotspot.x,

cursorInfo.ptScreenPos.y - hotspot.y

}

};

}

POINT\* CursorCapture::getHotspotFromCache(const HICON hIcon) {

const auto it = \_cachedHotspots.find(hIcon);

if (it != \_cachedHotspots.end()) {

return &it->second;

}

if (const auto hotspot = getHotspotFromIconInfo(hIcon)) {

return &\_cachedHotspots.emplace(hIcon, \*hotspot).first->second;

}

return nullptr;

}

std::optional<POINT> CursorCapture::getHotspotFromIconInfo(const HICON hIcon) {

ICONINFO iconInfo;

if (GetIconInfo(hIcon, &iconInfo)) {

releaseIconInfoResources(iconInfo);

return POINT{static\_cast<LONG>(iconInfo.xHotspot), static\_cast<LONG>(iconInfo.yHotspot)};

}

return std::nullopt;

}

void CursorCapture::releaseIconInfoResources(const ICONINFO& iconInfo) noexcept {

DeleteObject(iconInfo.hbmMask);

DeleteObject(iconInfo.hbmColor);

}

Приложение Г

(обязательное)

Исходный код программы (модуль ScreenCapture.hpp)

class ScreenCapture {

DC \_dc;

SIZE \_desktopSize;

SIZE \_targetSize;

Bitmap \_bitmap;

CursorCapture \_cursorCapture;

public:

explicit ScreenCapture(const SIZE& targetSize);

std::vector<byte> capture();

};

ScreenCapture::ScreenCapture(const SIZE& targetSize): \_dc(nullptr),

\_desktopSize(ScreenUtils::getScreenSize()),

\_targetSize(targetSize),

\_bitmap(BitmapFactory::createDDBitmap(\_dc.getHDC(), \_desktopSize)) { }

std::vector<byte> ScreenCapture::capture() {

\_bitmap.copyFrom(\_dc.getHDC());

if (const auto cursor = \_cursorCapture.capture()) {

\_bitmap.drawIcon(\*cursor);

}

return \_desktopSize == \_targetSize

? \_bitmap.getDIBits()

: ImageUtils::scaleImageBuffer(\_bitmap.getDIBits(), \_desktopSize, \_targetSize);

}

Приложение Д

(обязательное)

Исходный код программы (модуль ScreenRender.hpp)

class ScreenRender {

DC \_dc;

SIZE \_appSize;

SIZE \_originalSize;

Bitmap \_bitmap;

std::mutex \_appSizeMutex;

public:

ScreenRender(const HWND hWnd, const SIZE& appSize, const SIZE& originalSize);

void render(const std::vector<byte>& screen);

void updateAppSize(const SIZE& appSize);

};

ScreenRender::ScreenRender(const HWND hWnd, const SIZE& appSize, const SIZE& originalSize):

\_dc(hWnd), \_appSize(appSize), \_originalSize(originalSize),

\_bitmap(BitmapFactory::createDDBitmap(\_dc.getHDC(), \_appSize)) { }

void ScreenRender::render(const std::vector<byte>& screen) {

std::lock\_guard<std::mutex> lock(\_appSizeMutex);

\_bitmap.setDIBits(\_originalSize == \_appSize

? screen

: ImageUtils::scaleImageBuffer(screen, \_originalSize, \_appSize)

);

\_bitmap.copyTo(\_dc.getHDC());

}

void ScreenRender::updateAppSize(const SIZE& appSize) {

std::lock\_guard<std::mutex> lock(\_appSizeMutex);

\_appSize = appSize;

\_bitmap = BitmapFactory::createDDBitmap(\_dc.getHDC(), \_appSize);

}

Приложение Е

(обязательное)

Исходный код программы (модуль DC.hpp)

class DC {

HWND \_hWnd;

HDC \_hDc;

public:

explicit DC(const HWND hWnd);

HDC getHDC() const;

~DC();

DC(DC&&) = delete;

DC& operator=(DC&&) = delete;

DC(const DC&) = delete;

DC& operator=(const DC&) = delete;

};

DC::DC(const HWND hWnd): \_hWnd(hWnd), \_hDc(GetDC(hWnd)) { }

HDC DC::getHDC() const { return \_hDc; }

DC::~DC() { ReleaseDC(\_hWnd, \_hDc); }

Приложение Ж

(обязательное)

Исходный код программы (модуль QueueSizeMonitor.hpp)

template <typename T>

class QueueSizeMonitor {

ThreadSafeQueue<T>& \_queue;

std::atomic<size\_t> \_maxFrames;

public:

QueueSizeMonitor(ThreadSafeQueue<T>& queue, const int fps, const int maxDelayMs);

void setMaxFrames(const int fps, const int maxDelayMs);

void maintainQueueSize();

~QueueSizeMonitor() = default;

QueueSizeMonitor(QueueSizeMonitor&&) = delete;

QueueSizeMonitor& operator=(QueueSizeMonitor&&) = delete;

QueueSizeMonitor(const QueueSizeMonitor&) = delete;

QueueSizeMonitor& operator=(const QueueSizeMonitor&) = delete;

};

template <typename T>

QueueSizeMonitor<T>::QueueSizeMonitor(ThreadSafeQueue<T>& queue, const int fps, const int maxDelayMs):

\_queue(queue) {

setMaxFrames(fps, maxDelayMs);

}

template <typename T>

void QueueSizeMonitor<T>::setMaxFrames(const int fps, const int maxDelayMs) {

\_maxFrames = static\_cast<size\_t>(FrameUtils::calcFrames(fps, maxDelayMs));

}

template <typename T>

void QueueSizeMonitor<T>::maintainQueueSize() {

const size\_t size = \_queue.getSize();

if (size > \_maxFrames) {

const size\_t excess = size - \_maxFrames;

\_queue.trimQueue(excess);

}

}

Приложение З

(обязательное)

Исходный код программы (модуль Socket.hpp)

class Socket {

SOCKET \_sock;

explicit Socket(const SOCKET socket);

public:

Socket(const int type, const int protocol);

void bindSocket(const u\_short port) const;

void listenOnSocket(const int backlog) const;

Socket acceptConnection(sockaddr\_in\* clientAddr = nullptr, int\* clientAddrSize = nullptr) const;

void connectToServer(const sockaddr\_in& serverAddr) const;

int sendSocket(const std::vector<byte>& buffer) const;

int sendToSocket(const std::vector<byte>& buffer, const sockaddr\_in& destAddr) const;

int recvSocket(std::vector<byte>& buffer) const;

int recvFromSocket(std::vector<byte>& buffer,

sockaddr\_in\* senderAddr = nullptr, int\* senderAddrSize = nullptr) const;

void setReceiveTimeout(const DWORD milliseconds) const;

void setSendTimeout(const DWORD milliseconds) const;

void setSendBufferSize(const DWORD bufferSize) const;

void setReceiveBufferSize(const DWORD bufferSize) const;

sockaddr\_in getPeerAddress() const;

void shutdownSocket(const int how = SD\_BOTH) const;

~Socket();

Socket(Socket&&) noexcept;

Socket& operator=(Socket&&) noexcept;

Socket(const Socket&) = delete;

Socket& operator=(const Socket&) = delete;

private:

void releaseResources() const noexcept;

void resetResources() noexcept;

int setSockOpt(const DWORD value, const int option) const;

};

Socket::Socket(const SOCKET socket): \_sock(socket) {

SocketErrorChecker::checkSocket(\_sock);

}

Socket::Socket(const int type, const int protocol): \_sock(socket(AF\_INET, type, protocol)) {

SocketErrorChecker::checkSocket(\_sock);

}

void Socket::bindSocket(const u\_short port) const {

const sockaddr\_in serverAddr = SockaddrUtils::createAddrAnyInterface(port);

const int result = bind(\_sock, reinterpret\_cast<const sockaddr\*>(&serverAddr), sizeof(serverAddr));

SocketErrorChecker::checkBindError(result);

}

void Socket::listenOnSocket(const int backlog) const {

const int result = listen(\_sock, backlog);

SocketErrorChecker::checkListenError(result);

}

Socket Socket::acceptConnection(sockaddr\_in\* clientAddr, int\* clientAddrSize) const {

const SOCKET clientSock = accept(\_sock, reinterpret\_cast<sockaddr\*>(clientAddr), clientAddrSize);

SocketErrorChecker::checkAcceptError(clientSock);

return Socket(clientSock);

}

void Socket::connectToServer(const sockaddr\_in& serverAddr) const {

const int result = connect(\_sock, reinterpret\_cast<const sockaddr\*>(&serverAddr), sizeof(serverAddr));

SocketErrorChecker::checkConnectError(result);

}

int Socket::sendSocket(const std::vector<byte>& buffer) const {

const int result = send(\_sock, reinterpret\_cast<const char\*>(buffer.data()), buffer.size(), 0);

return SocketErrorChecker::checkSend(result);

}

int Socket::sendToSocket(const std::vector<byte>& buffer, const sockaddr\_in& destAddr) const {

const int result = sendto(\_sock, reinterpret\_cast<const char\*>(buffer.data()), buffer.size(), 0,

reinterpret\_cast<const sockaddr\*>(&destAddr), sizeof(destAddr));

return SocketErrorChecker::checkSend(result);

}

int Socket::recvSocket(std::vector<byte>& buffer) const {

const int len = recv(\_sock, reinterpret\_cast<char\*>(buffer.data()), buffer.size(), 0);

return SocketErrorChecker::checkReceive(len);

}

int Socket::recvFromSocket(std::vector<byte>& buffer, sockaddr\_in\* senderAddr, int\* senderAddrSize) const {

const int len = recvfrom(\_sock, reinterpret\_cast<char\*>(buffer.data()), buffer.size(), 0,

reinterpret\_cast<sockaddr\*>(senderAddr), senderAddrSize);

return SocketErrorChecker::checkReceive(len);

}

void Socket::setReceiveTimeout(const DWORD milliseconds) const {

const int result = setSockOpt(milliseconds, SO\_RCVTIMEO);

SocketErrorChecker::checkReceiveTimeoutError(result);

}

void Socket::setSendTimeout(const DWORD milliseconds) const {

const int result = setSockOpt(milliseconds, SO\_SNDTIMEO);

SocketErrorChecker::checkSendTimeoutError(result);

}

void Socket::setSendBufferSize(const DWORD bufferSize) const {

const int result = setSockOpt(bufferSize, SO\_SNDBUF);

SocketErrorChecker::checkSetSendBufferError(result);

}

void Socket::setReceiveBufferSize(const DWORD bufferSize) const {

const int result = setSockOpt(bufferSize, SO\_RCVBUF);

SocketErrorChecker::checkSetReceiveBufferError(result);

}

sockaddr\_in Socket::getPeerAddress() const {

sockaddr\_in peerAddr;

int peerAddrSize = sizeof(peerAddr);

const int result = getpeername(\_sock, reinterpret\_cast<sockaddr\*>(&peerAddr), &peerAddrSize);

SocketErrorChecker::checkGetPeerAddress(result);

return peerAddr;

}

void Socket::shutdownSocket(const int how) const {

const int result = shutdown(\_sock, how);

SocketErrorChecker::checkShutdownSocket(result);

}

int Socket::setSockOpt(const DWORD value, const int option) const {

return setsockopt(\_sock, SOL\_SOCKET, option,

reinterpret\_cast<const char\*>(&value), sizeof(value));

}

Socket::~Socket() {

releaseResources();

}

Socket::Socket(Socket&& other) noexcept : \_sock(other.\_sock) {

other.resetResources();

}

Socket& Socket::operator=(Socket&& other) noexcept {

if (this != &other) {

releaseResources();

\_sock = other.\_sock;

other.resetResources();

}

return \*this;

}

void Socket::releaseResources() const noexcept {

closesocket(\_sock);

}

void Socket::resetResources() noexcept {

\_sock = INVALID\_SOCKET;

}

Приложение И

(обязательное)

Исходный код программы (модуль SocketErrorChecker.hpp)

class SocketErrorChecker {

public:

SocketErrorChecker() = delete;

static constexpr int SOCKET\_TIMEOUT = -2;

static void checkSocket(const SOCKET sock);

static void checkBindError(const int result);

static void checkInetPton(const int result);

static void checkListenError(const int result);

static void checkAcceptError(const SOCKET clientSock);

static void checkConnectError(const int result);

static int checkSend(const int result);

static int checkReceive(const int len);

static void checkReceiveTimeoutError(const int result);

static void checkSendTimeoutError(const int result);

static void checkSetSendBufferError(const int result);

static void checkSetReceiveBufferError(const int result);

static void checkGetPeerAddress(const int result);

static void checkShutdownSocket(const int result);

private:

static void checkSetSockOptError(const int result, const std::string& option);

[[noreturn]] static void throwWSAError(const std::string& message);

[[noreturn]] static void throwWSAError(const std::string& message, const int errorCode);

};

void SocketErrorChecker::checkSocket(const SOCKET sock) {

if (sock == INVALID\_SOCKET) {

throwWSAError("Failed to create socket");

}

}

void SocketErrorChecker::checkBindError(const int result) {

if (result == SOCKET\_ERROR) {

throwWSAError("Failed to bind socket");

}

}

void SocketErrorChecker::checkInetPton(const int result) {

if (result <= 0) {

throwWSAError("Invalid IP address format");

}

}

void SocketErrorChecker::checkListenError(const int result) {

if (result == SOCKET\_ERROR) {

throwWSAError("Failed to listen on socket");

}

}

void SocketErrorChecker::checkAcceptError(const SOCKET clientSock) {

if (clientSock == INVALID\_SOCKET) {

throwWSAError("Failed to accept client connection");

}

}

void SocketErrorChecker::checkConnectError(const int result) {

if (result == SOCKET\_ERROR) {

throwWSAError("Failed to connect to server");

}

}

int SocketErrorChecker::checkSend(const int result) {

if (result == SOCKET\_ERROR) {

const int errorCode = WSAGetLastError();

if (errorCode == WSAETIMEDOUT) {

return SOCKET\_TIMEOUT;

}

throwWSAError("Failed to send data", errorCode);

}

return result;

}

int SocketErrorChecker::checkReceive(const int len) {

if (len == SOCKET\_ERROR) {

const int errorCode = WSAGetLastError();

if (errorCode == WSAETIMEDOUT) {

return SOCKET\_TIMEOUT;

}

throwWSAError("Failed to receive data or connection closed", errorCode);

}

return len;

}

void SocketErrorChecker::checkReceiveTimeoutError(const int result) {

checkSetSockOptError(result, "receive timeout");

}

void SocketErrorChecker::checkSendTimeoutError(const int result) {

checkSetSockOptError(result, "send timeout");

}

void SocketErrorChecker::checkSetSendBufferError(const int result) {

checkSetSockOptError(result, "send buffer size");

}

void SocketErrorChecker::checkSetReceiveBufferError(const int result) {

checkSetSockOptError(result, "receive buffer size");

}

void SocketErrorChecker::checkSetSockOptError(const int result, const std::string& option) {

if (result == SOCKET\_ERROR) {

throwWSAError("Failed to set socket option: " + option);

}

}

void SocketErrorChecker::checkGetPeerAddress(const int result) {

if (result == SOCKET\_ERROR) {

throwWSAError("Failed to get peer address");

}

}

void SocketErrorChecker::checkShutdownSocket(const int result) {

if (result == SOCKET\_ERROR) {

throwWSAError("Failed to shutdown socket");

}

}

void SocketErrorChecker::throwWSAError(const std::string& message) {

const int errorCode = WSAGetLastError();

throwWSAError(message, errorCode);

}

void SocketErrorChecker::throwWSAError(const std::string& message, const int errorCode) {

throw std::runtime\_error(message + ": " + std::to\_string(errorCode));

}

Приложение К

(обязательное)

Исходный код программы (модуль TCPConnection.hpp)

class TCPConnection {

friend class TCPServer;

Socket \_socket;

explicit TCPConnection(Socket&& socket);

public:

explicit TCPConnection(const sockaddr\_in& addr);

virtual int sendData(const std::vector<byte>& buffer);

virtual std::vector<byte> receiveData(int& bufferSize);

void setReceiveTimeout(const DWORD milliseconds) const;

void setSendTimeout(const DWORD milliseconds) const;

void setSendBufferSize(const DWORD bufferSize) const;

void setReceiveBufferSize(const DWORD bufferSize) const;

sockaddr\_in getPeerAddress() const;

void shutdownSocket(const int how = SD\_BOTH) const;

virtual ~TCPConnection() = default;

TCPConnection(TCPConnection&&) noexcept = default;

TCPConnection& operator=(TCPConnection&&) noexcept = default;

TCPConnection(const TCPConnection&) = delete;

TCPConnection& operator=(const TCPConnection&) = delete;

};

TCPConnection::TCPConnection(Socket&& socket): \_socket(std::move(socket)) {}

TCPConnection::TCPConnection(const sockaddr\_in& addr)

: \_socket(SOCK\_STREAM, IPPROTO\_TCP){

\_socket.connectToServer(addr);

}

int TCPConnection::sendData(const std::vector<byte>& buffer) {

return \_socket.sendSocket(buffer);

}

std::vector<byte> TCPConnection::receiveData(int& bufferSize) {

std::vector<byte> buffer(bufferSize);

bufferSize = \_socket.recvSocket(buffer);

return buffer;

}

void TCPConnection::setReceiveTimeout(const DWORD milliseconds) const {

\_socket.setReceiveTimeout(milliseconds);

}

void TCPConnection::setSendTimeout(const DWORD milliseconds) const {

\_socket.setSendTimeout(milliseconds);

}

void TCPConnection::setSendBufferSize(const DWORD bufferSize) const {

\_socket.setSendBufferSize(bufferSize);

}

void TCPConnection::setReceiveBufferSize(const DWORD bufferSize) const {

\_socket.setReceiveBufferSize(bufferSize);

}

sockaddr\_in TCPConnection::getPeerAddress() const {

return \_socket.getPeerAddress();

}

void TCPConnection::shutdownSocket(const int how) const {

return \_socket.shutdownSocket(how);

}

Приложение Л

(обязательное)

Исходный код программы (модуль SecureTCPConnection.hpp)

class SecureTCPConnection final : public TCPConnection {

AESEncryptor \_encryptor;

AESDecryptor \_decryptor;

public:

SecureTCPConnection(const std::vector<byte>& key, TCPConnection &&connection);

int sendData(const std::vector<byte>& buffer) override;

std::vector<byte> receiveData(int& bufferSize) override;

private:

static void adjustBufferSizeForMetadata(int& bufferSize);

static bool isBufferSizeSufficient(const int bufferSize);

std::vector<byte> createDecryptedData(const std::vector<byte>& buffer, int& bufferSize);

static std::vector<byte> createEmptyData(int& bufferSize);

};

SecureTCPConnection::SecureTCPConnection(const std::vector<byte>& key, TCPConnection&& connection)

: TCPConnection(std::move(connection)), \_encryptor(key), \_decryptor(key) { }

int SecureTCPConnection::sendData(const std::vector<byte>& buffer) {

return TCPConnection::sendData(\_encryptor.encrypt(buffer));

}

std::vector<byte> SecureTCPConnection::receiveData(int& bufferSize) {

adjustBufferSizeForMetadata(bufferSize);

const std::vector<byte> buffer = TCPConnection::receiveData(bufferSize);

if (isBufferSizeSufficient(bufferSize)) {

return createDecryptedData(buffer, bufferSize);

}

return createEmptyData(bufferSize);

}

void SecureTCPConnection::adjustBufferSizeForMetadata(int& bufferSize) {

bufferSize += AESToolkit::METADATA\_SIZE;

}

bool SecureTCPConnection::isBufferSizeSufficient(const int bufferSize) {

return bufferSize >= AESToolkit::METADATA\_SIZE;

}

std::vector<byte> SecureTCPConnection::createDecryptedData(const std::vector<byte>& buffer, int& bufferSize) {

bufferSize -= AESToolkit::METADATA\_SIZE;

return \_decryptor.decrypt(buffer);

}

std::vector<byte> SecureTCPConnection::createEmptyData(int& bufferSize) {

bufferSize = 0;

return {};

}

Приложение М

(обязательное)

Исходный код программы (модуль TCPServer.hpp)

class TCPServer {

Socket \_listenSocket;

public:

explicit TCPServer(const u\_short port);

TCPConnection acceptConnection(sockaddr\_in\* clientAddr = nullptr, int\* clientAddrSize = nullptr) const;

~TCPServer() = default;

TCPServer(TCPServer&&) noexcept = default;

TCPServer& operator=(TCPServer&&) noexcept = default;

TCPServer(const TCPServer&) = delete;

TCPServer& operator=(const TCPServer&) = delete;

};

TCPServer::TCPServer(const u\_short port)

: \_listenSocket(SOCK\_STREAM, IPPROTO\_TCP) {

\_listenSocket.bindSocket(port);

\_listenSocket.listenOnSocket(SOMAXCONN);

}

TCPConnection TCPServer::acceptConnection(sockaddr\_in\* clientAddr, int\* clientAddrSize) const {

return TCPConnection(\_listenSocket.acceptConnection(clientAddr, clientAddrSize));

}

Приложение Н

(обязательное)

Исходный код программы (модуль UDPSender.hpp)

class UDPSender {

std::unique\_ptr<DataFragmenter> \_dataFragmenter;

Socket \_socket {SOCK\_DGRAM, IPPROTO\_UDP};

sockaddr\_in \_addr;

public:

UDPSender(const sockaddr\_in& addr, std::unique\_ptr<DataFragmenter> dataFragmenter,

const DWORD sendBufferSize = MemoryUnits::MEGABYTE, const DWORD sendTimeoutMs = 100);

void send(const std::vector<byte>& data) const;

DataFragmenter& getDataFragmenter() const;

private:

void sendFragments(const std::vector<std::vector<byte>>& fragments) const;

void sendFragment(const std::vector<byte>& fragment) const;

};

UDPSender::UDPSender(const sockaddr\_in& addr, std::unique\_ptr<DataFragmenter> dataFragmenter,

const DWORD sendBufferSize, const DWORD sendTimeoutMs):

\_dataFragmenter(std::move(dataFragmenter)), \_addr(addr) {

\_socket.setSendBufferSize(sendBufferSize);

\_socket.setSendTimeout(sendTimeoutMs);

}

void UDPSender::send(const std::vector<byte>& data) const {

const std::vector<std::vector<byte>> fragments = \_dataFragmenter->createFragments(data);

sendFragments(fragments);

}

DataFragmenter& UDPSender::getDataFragmenter() const {

return \*\_dataFragmenter;

}

void UDPSender::sendFragments(const std::vector<std::vector<byte>>& fragments) const {

for (const auto& fragment : fragments) {

sendFragment(fragment);

}

}

void UDPSender::sendFragment(const std::vector<byte>& fragment) const {

\_socket.sendToSocket(fragment, \_addr);

}

Приложение О

(обязательное)

Исходный код программы (модуль UDPReceiver.hpp)

class UDPReceiver {

std::unique\_ptr<DataReassembler> \_dataReassembler;

Socket \_socket {SOCK\_DGRAM, IPPROTO\_UDP};

FragmentCollector \_fragmentCollector;

public:

UDPReceiver(const u\_short port, std::unique\_ptr<DataReassembler> dataReassembler,

const DWORD receiveBufferSize = 8\*MemoryUnits::MEGABYTE, const DWORD receiveTimeoutMs = 100);

std::vector<byte> receive();

DataReassembler& getDataReassembler() const;

private:

std::vector<Fragment> receiveFragments();

std::optional<Fragment> receiveFragment() const;

int receiveData(std::vector<byte>& data) const;

static bool isBytesReceivedValid(const int bytesReceived);

};

UDPReceiver::UDPReceiver(const u\_short port, std::unique\_ptr<DataReassembler> dataReassembler,

const DWORD receiveBufferSize, const DWORD receiveTimeoutMs):

\_dataReassembler(std::move(dataReassembler)),

\_fragmentCollector(\_dataReassembler->getFragmentDescriptor().getPayloadSize()) {

\_socket.bindSocket(port);

\_socket.setReceiveBufferSize(receiveBufferSize);

\_socket.setReceiveTimeout(receiveTimeoutMs);

}

std::vector<byte> UDPReceiver::receive() {

std::vector<Fragment> fragments(receiveFragments());

return \_dataReassembler->reassembleData(fragments);

}

DataReassembler& UDPReceiver::getDataReassembler() const {

return \*\_dataReassembler;

}

std::vector<Fragment> UDPReceiver::receiveFragments() {

if (\_fragmentCollector.collectPreviousFragment()) {

return \_fragmentCollector.extractFragments();

}

std::optional<Fragment> optionalFragment;

do {

optionalFragment = receiveFragment();

} while(\_fragmentCollector.collectFragment(optionalFragment));

return \_fragmentCollector.extractFragments();

}

std::optional<Fragment> UDPReceiver::receiveFragment() const {

Fragment fragment;

const int bytesReceived = receiveData(fragment.payload);

if (isBytesReceivedValid(bytesReceived)) {

UDPToolkit::populateFragment(fragment, bytesReceived);

return std::move(fragment);

}

return std::nullopt;

}

int UDPReceiver::receiveData(std::vector<byte>& data) const {

data.resize(\_dataReassembler->getFragmentDescriptor().getSize());

const int len = \_socket.recvFromSocket(data);

return len;

}

bool UDPReceiver::isBytesReceivedValid(const int bytesReceived) {

return bytesReceived >= static\_cast<int>(UDPToolkit::METADATA\_SIZE);

}

Приложение П

(обязательное)

Исходный код программы (модуль FragmentCollector.hpp)

class FragmentCollector {

FragmentBuffer \_fragmentBuffer;

FragmentStateTracker \_stateTracker;

std::optional<Fragment> \_prevFragment;

public:

explicit FragmentCollector(const size\_t fragmentPayloadSize);

bool collectPreviousFragment();

bool collectFragment(std::optional<Fragment>& optionalFragment);

std::vector<Fragment> extractFragments();

private:

bool processFragment(Fragment& fragment);

bool processUninitializedStateTracker(Fragment& fragment);

bool processOldFragment() const;

bool processNewFragment(Fragment& fragment);

bool processCurrentFragment(Fragment& fragment);

bool isAllFragmentsReceived() const;

};

FragmentCollector::FragmentCollector(const size\_t fragmentPayloadSize): \_stateTracker(fragmentPayloadSize) { }

bool FragmentCollector::collectPreviousFragment() {

if (\_prevFragment) {

const bool isSingleFragmentPacket = processUninitializedStateTracker(\*\_prevFragment);

\_prevFragment.reset();

return isSingleFragmentPacket;

}

return false;

}

// ReSharper disable once CppParameterMayBeConstPtrOrRef

bool FragmentCollector::collectFragment(std::optional<Fragment>& optionalFragment) {

return optionalFragment.has\_value() && !processFragment(\*optionalFragment);

}

bool FragmentCollector::processFragment(Fragment& fragment) {

if (!\_stateTracker.isInitialized()) {

return processUninitializedStateTracker(fragment);

}

if (\_stateTracker.isOldFragment(fragment)) {

return processOldFragment();

}

if (\_stateTracker.isNewFragment(fragment)) {

return processNewFragment(fragment);

}

return processCurrentFragment(fragment);

}

bool FragmentCollector::processUninitializedStateTracker(Fragment& fragment) {

if (\_stateTracker.initialize(fragment)) {

\_fragmentBuffer.reserve(\_stateTracker.getTotalFragments());

\_fragmentBuffer.insertFragment(std::move(fragment));

return isAllFragmentsReceived();

}

return false;

}

bool FragmentCollector::processOldFragment() const {

return \_stateTracker.hasExceededOldFragmentCountInRow();

}

bool FragmentCollector::processNewFragment(Fragment& fragment) {

\_prevFragment = std::move(fragment);

return true;

}

bool FragmentCollector::processCurrentFragment(Fragment& fragment) {

\_fragmentBuffer.insertFragment(std::move(fragment));

return isAllFragmentsReceived();

}

bool FragmentCollector::isAllFragmentsReceived() const {

return \_stateTracker.getTotalFragments() == \_fragmentBuffer.getSize();

}

std::vector<Fragment> FragmentCollector::extractFragments() {

\_stateTracker.reset();

return \_fragmentBuffer.extractFragments();

}

Приложение Р

(обязательное)

Исходный код программы (модуль FragmentStateTracker.hpp)

class FragmentStateTracker {

static constexpr size\_t MAX\_OLD\_FRAGMENTS\_IN\_ROW = 10;

size\_t \_fragmentPayloadSize;

bool \_isInitialized = false;

size\_t \_oldFragmentCountInRow = 0;

size\_t \_prevId = 0;

size\_t \_currentId = 0;

size\_t \_totalFragments = 0;

public:

explicit FragmentStateTracker(const size\_t fragmentPayloadSize);

void reset();

bool initialize(const Fragment& fragment);

bool isOldFragment(const Fragment& fragment);

bool hasExceededOldFragmentCountInRow() const;

bool isNewFragment(const Fragment& fragment) const;

bool isInitialized() const;

size\_t getTotalFragments() const;

};

FragmentStateTracker::FragmentStateTracker(const size\_t fragmentPayloadSize): \_fragmentPayloadSize(fragmentPayloadSize) { }

void FragmentStateTracker::reset() {

\_isInitialized = false;

\_oldFragmentCountInRow = 0;

}

bool FragmentStateTracker::initialize(const Fragment& fragment) {

if (\_prevId < fragment.fragmentId) {

\_isInitialized = true;

\_prevId = \_currentId;

\_currentId = fragment.fragmentId;

\_totalFragments = fragment.totalFragments;

return true;

}

return false;

}

bool FragmentStateTracker::isOldFragment(const Fragment& fragment) {

if (\_currentId > fragment.fragmentId) {

++\_oldFragmentCountInRow;

return true;

}

\_oldFragmentCountInRow = 0;

return false;

}

bool FragmentStateTracker::hasExceededOldFragmentCountInRow() const {

return \_oldFragmentCountInRow >= MAX\_OLD\_FRAGMENTS\_IN\_ROW;

}

bool FragmentStateTracker::isNewFragment(const Fragment& fragment) const {

return \_currentId < fragment.fragmentId;

}

bool FragmentStateTracker::isInitialized() const { return \_isInitialized; }

size\_t FragmentStateTracker::getTotalFragments() const { return \_totalFragments; }

Приложение С

(обязательное)

Исходный код программы (модуль ThreadSafeQueue.hpp)

template<typename T>

class ThreadSafeQueue {

std::queue<std::unique\_ptr<T>> \_queue;

std::mutex \_mutex;

std::condition\_variable \_cond;

public:

void enqueue(std::unique\_ptr<T> item);

std::unique\_ptr<T> dequeue();

void trimQueue(const size\_t n);

size\_t getSize();

bool isEmpty();

void notifyAll();

private:

std::unique\_ptr<T> extractItem();

};

template <typename T>

void ThreadSafeQueue<T>::enqueue(std::unique\_ptr<T> item) {

std::lock\_guard<std::mutex> lock(\_mutex);

\_queue.push(std::move(item));

\_cond.notify\_one();

}

template <typename T>

std::unique\_ptr<T> ThreadSafeQueue<T>::dequeue() {

std::unique\_lock<std::mutex> lock(\_mutex);

\_cond.wait(lock, [this] { return !\_queue.empty(); });

return extractItem();

}

template<typename T>

void ThreadSafeQueue<T>::trimQueue(const size\_t n) {

std::lock\_guard<std::mutex> lock(\_mutex);

for (size\_t i = 0; i < n && !\_queue.empty(); ++i) {

\_queue.pop();

}

}

template <typename T>

size\_t ThreadSafeQueue<T>::getSize() {

std::lock\_guard<std::mutex> lock(\_mutex);

return \_queue.size();

}

template <typename T>

bool ThreadSafeQueue<T>::isEmpty() {

std::lock\_guard<std::mutex> lock(\_mutex);

return \_queue.empty();

}

template <typename T>

void ThreadSafeQueue<T>::notifyAll() {

std::lock\_guard<std::mutex> lock(\_mutex);

\_cond.notify\_all();

}

template <typename T>

std::unique\_ptr<T> ThreadSafeQueue<T>::extractItem() {

if (\_queue.empty()) {

return nullptr;

}

auto item = std::move(\_queue.front());

\_queue.pop();

return item;

}

Приложение Т

(обязательное)

Исходный код программы (модуль ThreadWorker.hpp)

class ThreadWorker {

std::thread \_eventThread;

protected:

std::atomic<bool> \_running = false;

ThreadWorker() = default;

public:

void start();

void stopRunning();

void waitForThread();

virtual ~ThreadWorker();

ThreadWorker(ThreadWorker&&) = delete;

ThreadWorker& operator=(ThreadWorker&&) = delete;

ThreadWorker(const ThreadWorker&) = delete;

ThreadWorker& operator=(const ThreadWorker&) = delete;

protected:

virtual void eventLoop() = 0;

};

void ThreadWorker::start() {

if (!\_running) {

\_running = true;

\_eventThread = std::thread(&ThreadWorker::eventLoop, this);

}

}

void ThreadWorker::stopRunning() {

if (\_running) {

\_running = false;

}

}

void ThreadWorker::waitForThread() {

if (\_eventThread.joinable()) {

\_eventThread.join();

}

}

ThreadWorker::~ThreadWorker() {

stopRunning();

waitForThread();

}

Приложение У

(обязательное)

Исходный код программы (модуль AESDecryptor.hpp)

class AESDecryptor final : public AESKeyController {

using AESCtrDecryption = CryptoPP::CTR\_Mode<CryptoPP::AES>::Decryption;

AESCtrDecryption \_decryptor;

public:

explicit AESDecryptor(const std::vector<byte>& key);

AESDecryptor() = default;

std::vector<byte> decrypt(const std::vector<byte>& ciphertext);

std::vector<byte> decrypt(const byte\* ciphertext, const size\_t ciphertextSize);

private:

static void validateCiphertextSize(const size\_t ciphertextSize);

static CryptoPP::SecByteBlock extractIV(const byte\* ciphertext);

void performDecryption(const byte\* ciphertext, std::vector<byte>& plaintext);

};

AESDecryptor::AESDecryptor(const std::vector<byte>& key): AESKeyController(key) { }

std::vector<byte> AESDecryptor::decrypt(const std::vector<byte>& ciphertext) {

return decrypt(ciphertext.data(), ciphertext.size());

}

std::vector<byte> AESDecryptor::decrypt(const byte\* ciphertext, const size\_t ciphertextSize) {

validateCiphertextSize(ciphertextSize);

std::vector<byte> plaintext(ciphertextSize - AESToolkit::METADATA\_SIZE);

const CryptoPP::SecByteBlock iv = extractIV(ciphertext);

setKeyWithIV(\_decryptor, iv);

performDecryption(ciphertext, plaintext);

return plaintext;

}

void AESDecryptor::validateCiphertextSize(const size\_t ciphertextSize) {

if (ciphertextSize < AESToolkit::METADATA\_SIZE) {

throw std::invalid\_argument("Invalid ciphertext");

}

}

CryptoPP::SecByteBlock AESDecryptor::extractIV(const byte\* ciphertext) {

CryptoPP::SecByteBlock iv(AESToolkit::METADATA\_SIZE);

std::memcpy(iv.data(), ciphertext, iv.size());

return iv;

}

void AESDecryptor::performDecryption(const byte\* ciphertext, std::vector<byte>& plaintext) {

\_decryptor.ProcessData(plaintext.data(), &ciphertext[AESToolkit::METADATA\_SIZE], plaintext.size());

}

Приложение Ф

(обязательное)

Исходный код программы (модуль AESEncryptor.hpp)

class AESEncryptor final : public AESKeyController {

using AESCtrEncryption = CryptoPP::CTR\_Mode<CryptoPP::AES>::Encryption;

AESCtrEncryption \_encryptor;

CryptoPP::AutoSeededRandomPool \_rng;

public:

explicit AESEncryptor(const std::vector<byte>& key);

AESEncryptor() = default;

std::vector<byte> encrypt(const std::vector<byte>& plaintext);

std::vector<byte> encrypt(const byte\* plaintext, const size\_t plaintextSize);

private:

CryptoPP::SecByteBlock generateIV();

static void appendIVToCiphertext(std::vector<byte>& ciphertext, const CryptoPP::SecByteBlock& iv);

void performEncryption(std::vector<byte>& ciphertext, const byte\* plaintext, const size\_t plaintextSize);

};

AESEncryptor::AESEncryptor(const std::vector<byte>& key): AESKeyController(key) { }

std::vector<byte> AESEncryptor::encrypt(const std::vector<byte>& plaintext) {

return encrypt(plaintext.data(), plaintext.size());

}

std::vector<byte> AESEncryptor::encrypt(const byte\* plaintext, const size\_t plaintextSize) {

const CryptoPP::SecByteBlock iv = generateIV();

std::vector<byte> ciphertext(plaintextSize + AESToolkit::METADATA\_SIZE);

setKeyWithIV(\_encryptor, iv);

appendIVToCiphertext(ciphertext, iv);

performEncryption(ciphertext, plaintext, plaintextSize);

return ciphertext;

}

CryptoPP::SecByteBlock AESEncryptor::generateIV() {

CryptoPP::SecByteBlock iv(AESToolkit::METADATA\_SIZE);

\_rng.GenerateBlock(iv, iv.size());

return iv;

}

void AESEncryptor::appendIVToCiphertext(std::vector<byte>& ciphertext, const CryptoPP::SecByteBlock& iv) {

std::memcpy(ciphertext.data(), iv.data(), iv.size());

}

void AESEncryptor::performEncryption(std::vector<byte>& ciphertext, const byte\* plaintext, const size\_t plaintextSize) {

\_encryptor.ProcessData(&ciphertext[AESToolkit::METADATA\_SIZE], plaintext, plaintextSize);

}

Приложение Х

(обязательное)

Исходный код программы (модуль ReceiverMainForm.hpp)

class ReceiverMainForm {

static constexpr auto WINDOWS\_CLASS\_NAME = L"ScreenshotReceiverClass";

static constexpr int ID\_MENU = 1;

static constexpr int ID\_TOGGLE\_FULLSCREEN = 2;

static ACCEL \_accelTableEntries[];

static HINSTANCE \_hInstance;

const Fonts& \_fonts;

HWND \_hwnd;

ReceiverConfig \_config{};

Receiver \_receiver{};

SimpleConfigDialogForm \_configDialog;

HACCEL \_hAccel;

WINDOWPLACEMENT \_wpc;

LONG \_hwndStyle;

LONG \_hwndStyleEx;

bool \_isFullscreen;

public:

static void registerClass(const HINSTANCE hInstance);

explicit ReceiverMainForm(const Fonts &fonts);

void set(const ReceiverConfig &config, Receiver&& receiver);

void show() const;

HWND getHwnd() const;

~ReceiverMainForm();

ReceiverMainForm(ReceiverMainForm&&) = delete;

ReceiverMainForm& operator=(ReceiverMainForm&&) = delete;

ReceiverMainForm(const ReceiverMainForm&) = delete;

ReceiverMainForm& operator=(const ReceiverMainForm&) = delete;

private:

HWND createHwnd();

void onConnectionClosed() const;

static LRESULT CALLBACK windowProc(const HWND hwnd, const UINT uMsg, const WPARAM wParam, const LPARAM lParam);

LRESULT handleInput(const HWND hwnd, const UINT uMsg, const WPARAM wParam, const LPARAM lParam) const;

void toggleFullscreen();

};

ACCEL ReceiverMainForm::\_accelTableEntries[] = {

{FALT | FVIRTKEY, VK\_F1, ID\_MENU},

{FALT | FVIRTKEY, VK\_F11, ID\_TOGGLE\_FULLSCREEN},

};

HINSTANCE ReceiverMainForm::\_hInstance = nullptr;

void ReceiverMainForm::registerClass(const HINSTANCE hInstance) {

\_hInstance = hInstance;

WNDCLASSEX wc = {};

wc.cbSize = sizeof(WNDCLASSEX);

wc.lpfnWndProc = windowProc;

wc.hInstance = hInstance;

wc.lpszClassName = WINDOWS\_CLASS\_NAME;

RegisterClassEx(&wc);

}

ReceiverMainForm::ReceiverMainForm(const Fonts& fonts):

\_fonts(fonts), \_hwnd(createHwnd()), \_configDialog(\_hInstance, \_fonts),

\_hAccel(CreateAcceleratorTable(\_accelTableEntries, ARRAYSIZE(\_accelTableEntries))){ }

HWND ReceiverMainForm::createHwnd() {

return CreateWindowEx(0, WINDOWS\_CLASS\_NAME, L"TCP Screenshot Receiver",

WS\_OVERLAPPEDWINDOW, CW\_USEDEFAULT, CW\_USEDEFAULT,

GetSystemMetrics(SM\_CXSCREEN), GetSystemMetrics(SM\_CYSCREEN),

nullptr, nullptr, \_hInstance, this);

}

void ReceiverMainForm::set(const ReceiverConfig& config, Receiver&& receiver) {

\_config = config;

\_receiver = std::move(receiver);

\_receiver.setDisconnectCallback([this]{ onConnectionClosed(); });

\_configDialog.set(\_config, \_receiver);

}

void ReceiverMainForm::onConnectionClosed() const {

MessageBox(\_hwnd, L"Connection closed.", L"Info", MB\_OK | MB\_ICONINFORMATION);

PostMessage(\_hwnd, WM\_CLOSE, 0, 0);

}

void ReceiverMainForm::show() const {

ShowWindow(\_hwnd, SW\_SHOW);

\_receiver.run();

MSG msg;

while (GetMessage(&msg, nullptr, 0, 0)) {

if (!TranslateAccelerator(\_hwnd, \_hAccel, &msg)) {

TranslateMessage(&msg);

DispatchMessage(&msg);

}

}

}

HWND ReceiverMainForm::getHwnd() const { return \_hwnd; }

LRESULT ReceiverMainForm::windowProc(const HWND hwnd, const UINT uMsg, const WPARAM wParam, const LPARAM lParam) {

switch (uMsg) {

case WM\_CREATE: {

const CREATESTRUCT\* pCreateStruct = reinterpret\_cast<CREATESTRUCT\*>(lParam);

ReceiverMainForm\* mainForm = static\_cast<ReceiverMainForm\*>(pCreateStruct->lpCreateParams);

SetWindowLongPtr(hwnd, GWLP\_USERDATA, reinterpret\_cast<LONG\_PTR>(mainForm));

break;

}

case WM\_COMMAND: {

switch (LOWORD(wParam)) {

case ID\_MENU: {

ReceiverMainForm\* mainForm = reinterpret\_cast<ReceiverMainForm\*>(GetWindowLongPtr(hwnd, GWLP\_USERDATA));

if (!mainForm->\_configDialog.show(hwnd)) {

PostMessage(hwnd, WM\_CLOSE, 0, 0);

}

break;

}

case ID\_TOGGLE\_FULLSCREEN: {

ReceiverMainForm\* mainForm = reinterpret\_cast<ReceiverMainForm\*>(GetWindowLongPtr(hwnd, GWLP\_USERDATA));

mainForm->toggleFullscreen();

break;

}

}

break;

}

case WM\_DESTROY: {

const ReceiverMainForm\* mainForm = reinterpret\_cast<ReceiverMainForm\*>(GetWindowLongPtr(hwnd, GWLP\_USERDATA));

mainForm->\_receiver.stop();

PostQuitMessage(0);

break;

}

default: {

const ReceiverMainForm\* mainForm = reinterpret\_cast<ReceiverMainForm\*>(GetWindowLongPtr(hwnd, GWLP\_USERDATA));

if (mainForm != nullptr) {

return mainForm->handleInput(hwnd, uMsg, wParam, lParam);

}

break;

}

}

return DefWindowProc(hwnd, uMsg, wParam, lParam);

}

LRESULT ReceiverMainForm::handleInput(const HWND hwnd, const UINT uMsg, const WPARAM wParam, const LPARAM lParam) const {

if (hwnd == \_hwnd) {

switch (uMsg) {

case WM\_SIZE: {

if (wParam != SIZE\_MINIMIZED) {

const ReceiverMainForm\* mainForm = reinterpret\_cast<ReceiverMainForm\*>(GetWindowLongPtr(hwnd, GWLP\_USERDATA));

mainForm->\_receiver.updateAppSize({LOWORD(lParam), HIWORD(lParam)});

}

break;

}

case WM\_MOUSEMOVE:

case WM\_LBUTTONDOWN:

case WM\_LBUTTONUP:

case WM\_RBUTTONDOWN:

case WM\_RBUTTONUP:

case WM\_MBUTTONDOWN:

case WM\_MBUTTONUP:

case WM\_MOUSEWHEEL:

case WM\_MOUSEHWHEEL:

case WM\_KEYDOWN:

case WM\_KEYUP:

case WM\_SYSKEYDOWN:

case WM\_SYSKEYUP: {

const ReceiverMainForm\* mainForm = reinterpret\_cast<ReceiverMainForm\*>(GetWindowLongPtr(hwnd, GWLP\_USERDATA));

mainForm->\_receiver.sendEvent(uMsg, wParam, lParam);

return 0;

}

}

}

return DefWindowProc(hwnd, uMsg, wParam, lParam);

}

void ReceiverMainForm::toggleFullscreen() {

if (\_isFullscreen) {

SetWindowLong(\_hwnd, GWL\_STYLE, \_hwndStyle);

SetWindowLong(\_hwnd, GWL\_EXSTYLE, \_hwndStyleEx);

SetWindowPlacement(\_hwnd, &\_wpc);

ShowWindow(\_hwnd, SW\_SHOWNORMAL);

} else {

GetWindowPlacement(\_hwnd, &\_wpc);

if (\_hwndStyle == 0) \_hwndStyle = GetWindowLong(\_hwnd, GWL\_STYLE);

if (\_hwndStyleEx == 0) \_hwndStyleEx = GetWindowLong(\_hwnd, GWL\_EXSTYLE);

const LONG newHwndStyle = \_hwndStyle & ~WS\_BORDER & ~WS\_DLGFRAME & ~WS\_THICKFRAME;

const LONG newHwndStyleEx = \_hwndStyleEx & ~WS\_EX\_WINDOWEDGE;

SetWindowLong(\_hwnd, GWL\_STYLE, newHwndStyle | WS\_POPUP);

SetWindowLong(\_hwnd, GWL\_EXSTYLE, newHwndStyleEx | WS\_EX\_TOPMOST);

ShowWindow(\_hwnd, SW\_SHOWMAXIMIZED);

}

\_isFullscreen = !\_isFullscreen;

}

ReceiverMainForm::~ReceiverMainForm() {

DestroyAcceleratorTable(\_hAccel);

}

Приложение Ц

(обязательное)

Исходный код программы (модуль SenderMainForm.hpp)

class SenderMainForm {

static constexpr auto WINDOWS\_CLASS\_NAME = L"ScreenshotSenderClass";

static constexpr int BTN\_APPLY\_ID = 1;

static constexpr int BTN\_EXIT\_ID = 2;

static HINSTANCE \_hInstance;

HWND \_hEditFps = nullptr;

HWND \_hEditMaxDelay = nullptr;

HWND \_hEditQuality = nullptr;

HWND \_hComboBoxFormat = nullptr;

const Fonts &\_fonts;

SenderConfig \_config;

HWND \_hwnd;

Sender \_sender;

public:

static void registerClass(const HINSTANCE hInstance);

SenderMainForm(const Fonts &fonts, const SenderConfig &config);

void show() const;

~SenderMainForm() = default;

SenderMainForm(SenderMainForm&&) = delete;

SenderMainForm& operator=(SenderMainForm&&) = delete;

SenderMainForm(const SenderMainForm&) = delete;

SenderMainForm& operator=(const SenderMainForm&) = delete;

private:

HWND createHwnd();

void onConnectionClosed() const;

static LRESULT CALLBACK windowProc(const HWND hwnd, const UINT uMsg, const WPARAM wParam, const LPARAM lParam);

void createControls(const HWND hwnd);

void updateConfig();

};

HINSTANCE SenderMainForm::\_hInstance = nullptr;

void SenderMainForm::registerClass(const HINSTANCE hInstance) {

\_hInstance = hInstance;

WNDCLASSEX wc = {};

wc.cbSize = sizeof(WNDCLASSEX);

wc.lpfnWndProc = windowProc;

wc.hInstance = hInstance;

wc.lpszClassName = WINDOWS\_CLASS\_NAME;

wc.hbrBackground = reinterpret\_cast<HBRUSH>(COLOR\_WINDOW);

RegisterClassEx(&wc);

}

SenderMainForm::SenderMainForm(const Fonts& fonts, const SenderConfig& config):

\_fonts(fonts),

\_config(config),

\_hwnd(createHwnd()),

\_sender(\_config) {

\_sender.setDisconnectCallback([this]{ onConnectionClosed(); });

}

void SenderMainForm::onConnectionClosed() const {

MessageBox(\_hwnd, L"Connection closed.", L"Info", MB\_OK | MB\_ICONINFORMATION);

PostMessage(\_hwnd, WM\_CLOSE, 0, 0);

}

HWND SenderMainForm::createHwnd() {

return CreateWindowEx(0, WINDOWS\_CLASS\_NAME, L"TCP Screenshot Sender",

WS\_OVERLAPPED | WS\_CAPTION | WS\_SYSMENU | WS\_MINIMIZEBOX,

(GetSystemMetrics(SM\_CXSCREEN) - 586) / 2, (GetSystemMetrics(SM\_CYSCREEN) - 321) / 2,

586, 321, nullptr, nullptr, \_hInstance, this);

}

void SenderMainForm::show() const {

ShowWindow(\_hwnd, SW\_SHOW);

\_sender.run();

MSG msg;

while (GetMessage(&msg, nullptr, 0, 0)) {

TranslateMessage(&msg);

DispatchMessage(&msg);

}

}

LRESULT SenderMainForm::windowProc(const HWND hwnd, const UINT uMsg, const WPARAM wParam, const LPARAM lParam) {

switch (uMsg) {

case WM\_CREATE: {

const CREATESTRUCT\* pCreateStruct = reinterpret\_cast<CREATESTRUCT\*>(lParam);

SenderMainForm\* mainForm = static\_cast<SenderMainForm\*>(pCreateStruct->lpCreateParams);

SetWindowLongPtr(hwnd, GWLP\_USERDATA, reinterpret\_cast<LONG\_PTR>(mainForm));

mainForm->createControls(hwnd);

break;

}

case WM\_COMMAND: {

switch (LOWORD(wParam)) {

case BTN\_APPLY\_ID: {

SenderMainForm\* mainForm = reinterpret\_cast<SenderMainForm\*>(GetWindowLongPtr(hwnd, GWLP\_USERDATA));

mainForm->updateConfig();

break;

}

case BTN\_EXIT\_ID:

PostMessage(hwnd, WM\_CLOSE, 0, 0);

break;

}

break;

}

case WM\_DESTROY: {

const SenderMainForm\* mainForm = reinterpret\_cast<SenderMainForm\*>(GetWindowLongPtr(hwnd, GWLP\_USERDATA));

mainForm->\_sender.stop();

PostQuitMessage(0);

break;

}

}

return DefWindowProc(hwnd, uMsg, wParam, lParam);

}

void SenderMainForm::createControls(const HWND hwnd) {

const ControlCreator controlCreator(hwnd, \_fonts);

int currentY = 20;

controlCreator.createLabel(currentY, L"FPS:");

\_hEditFps = controlCreator.createEditControl(currentY, std::to\_wstring(\_config.fps));

currentY += ControlCreator::LABEL\_HEIGHT + ControlCreator::Y\_OFFSET;

controlCreator.createLabel(currentY, L"Max delay (ms):");

\_hEditMaxDelay = controlCreator.createEditControl(currentY, std::to\_wstring(\_config.maxDelayMs));

currentY += ControlCreator::LABEL\_HEIGHT + ControlCreator::Y\_OFFSET;

controlCreator.createLabel(currentY, L"Quality (1-100):");

\_hEditQuality = controlCreator.createEditControl(currentY, std::to\_wstring(\_config.imageConfig.quality));

currentY += ControlCreator::LABEL\_HEIGHT + ControlCreator::Y\_OFFSET;

controlCreator.createLabel(currentY, L"Format (jpg/png):");

\_hComboBoxFormat = controlCreator.createComboBox(currentY, {L"jpg", L"webp"}, static\_cast<int>(\_config.imageConfig.ext));

currentY += ControlCreator::CONTROL\_HEIGHT + ControlCreator::BUTTON\_Y\_OFFSET;

controlCreator.createDefButton(currentY, L"Apply", BTN\_APPLY\_ID);

controlCreator.createButton(currentY, L"Exit", BTN\_EXIT\_ID);

}

void SenderMainForm::updateConfig() {

static constexpr size\_t BUFFER\_SIZE = 256;

wchar\_t buffer[BUFFER\_SIZE];

int tempValue;

std::wstring accumulatedErrors;

GetWindowText(\_hEditFps, buffer, BUFFER\_SIZE);

if (!safeStoi(buffer, tempValue)) {

accumulatedErrors += L"Invalid FPS value.\n";

} else {

\_config.fps = tempValue;

}

GetWindowText(\_hEditMaxDelay, buffer, BUFFER\_SIZE);

if (!safeStoi(buffer, tempValue)) {

accumulatedErrors += L"Invalid max delay value.\n";

} else {

\_config.maxDelayMs = tempValue;

}

GetWindowText(\_hEditQuality, buffer, BUFFER\_SIZE);

if (!safeStoi(buffer, tempValue) || tempValue < 1 || tempValue > 100) {

accumulatedErrors += L"Invalid quality value.\n";

} else {

\_config.imageConfig.quality = tempValue;

}

int formatIndex = SendMessage(\_hComboBoxFormat, CB\_GETCURSEL, 0, 0);

\_config.imageConfig.ext = static\_cast<ImageFormat>(formatIndex);

if (!accumulatedErrors.empty()) {

MessageBox(\_hwnd, std::wstring(accumulatedErrors.begin(), accumulatedErrors.end()).c\_str(), L"Error", MB\_ICONERROR);

return;

}

\_sender.updateFPSAndMaxDelay(\_config.fps, \_config.maxDelayMs);

\_sender.updateImageConfig(\_config.imageConfig);

MessageBox(\_hwnd, L"Settings applied successfully!", L"Info", MB\_OK | MB\_ICONINFORMATION);

}

Приложение Ч

(обязательное)

Исходный код программы (модуль DHKeyExchanger.hpp)

class DHKeyExchanger {

DHHelper \_dhHelper;

CryptoPP::SecByteBlock \_privateKey;

CryptoPP::SecByteBlock \_publicKey;

TCPConnection& \_tcpConnection;

public:

explicit DHKeyExchanger(TCPConnection& tcpConnection);

void generateAndSendGroupParameters();

void receiveGroupParameters();

CryptoPP::Integer exchangeKeys();

private:

void generateRandomGroupParameters();

void sendGroupParameters() const;

void generateKeys();

void sendPublicKey();

CryptoPP::Integer receivePublicKey() const;

};

DHKeyExchanger::DHKeyExchanger(TCPConnection& tcpConnection): \_tcpConnection(tcpConnection) {}

void DHKeyExchanger::generateAndSendGroupParameters() {

generateRandomGroupParameters();

sendGroupParameters();

}

void DHKeyExchanger::generateRandomGroupParameters() {

\_dhHelper.generateRandomGroupParameters();

}

void DHKeyExchanger::sendGroupParameters() const {

TCPKeyExchangeUtils::sendDHParameters(\_tcpConnection, \_dhHelper.getP(), \_dhHelper.getG());

}

void DHKeyExchanger::receiveGroupParameters() {

const auto [p, g] = TCPKeyExchangeUtils::receiveDHParameters(\_tcpConnection);

\_dhHelper.initializeGroupParameters(p, g);

}

CryptoPP::Integer DHKeyExchanger::exchangeKeys() {

generateKeys();

sendPublicKey();

const CryptoPP::Integer serverPublicKey = receivePublicKey();

return \_dhHelper.calcSharedSecret(\_privateKey, serverPublicKey);

}

void DHKeyExchanger::generateKeys() {

\_privateKey = \_dhHelper.generatePrivateKey();

\_publicKey = \_dhHelper.generatePublicKey(\_privateKey);

}

void DHKeyExchanger::sendPublicKey() {

TCPKeyExchangeUtils::sendPublicKey(\_tcpConnection, {\_publicKey.data(), \_publicKey.size()});

}

CryptoPP::Integer DHKeyExchanger::receivePublicKey() const {

return TCPKeyExchangeUtils::receivePublicKey(\_tcpConnection);

}

ВЕДОМОСТЬ ДОКУМЕНТОВ

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Обозначение | | | | | Наименование | | | | Дополнительные сведения | |
|  | | | | | Текстовые документы | | | |  | |
|  | | | | |  | | | |  | |
| БГУИР КП 1–40 01 01 020 ПЗ | | | | | Пояснительная записка | | | | 100 с. | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | | Графические документы | | | |  | |
|  | | | | |  | | | |  | |
| ГУИР.251003-020 СА | | | | | Метод sendScreen. Схема алгоритма | | | | Формат А1 | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  |  |  |  |  | БГУИР КП I- 40 01 01 020 ПЗ | | | | | |
|  |  |  |  |  |
|  |  |  |  |  |
| Изм | Лист | № докум. | Подп. | Дата | Программное средство «Удалённый рабочий стол»  Ведомость курсового  Проекта | Литера | | | Лист | Листов |
| Разраб. | | Панкратьев Е.С. |  |  | Т |  |  | 100 | 100 |
| Провер. | | Деменковец Д.В. |  |  | Кафедра ПОИТ  гр. 251003 | | | | |