Міністерство освіти і науки України

Національний університет

«Львівська політехніка»

Кафедра електронних обчислювальних машин

**КУРСОВИЙ ПРОЄКТ**

з дисципліни “Системне програмне забезпечення”

на тему: “ Розробка програми, що автоматично виконує дії на комп'ютері, на який надано доступ віддалено ”

Студента 3-го курсу групи КІ-306

123 «Комп’ютерна інженерія»

Примак Б.В.

Керівник

Олексів М. В.

Національна шкала: \_\_\_\_\_\_\_

Кількість балів: \_\_\_\_\_\_\_

Оцінка ECTS: \_\_\_\_\_\_\_

Члени комісії: \_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (прізвище та ініціали)

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

(підпис) (прізвище та ініціали)

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

(підпис) (прізвище та ініціали)

Львів 2024

# Завдання на курсовий проект

Основним завданням для цього курсового проекту є розробка програми для автоматизації процесів віддаленого керування комп'ютером. Розробка програми, що автоматично виконує дії на комп'ютері, на який надано доступ віддалено. А саме віддалений запуск програм зі списку наданого сервером.

## Завдання проекту

1. Аналіз існуючих варіантів рішення питання
2. Розробка архітектури програми
3. Реалізація функціональних можливостей
   1. Поповнення на сервері списку програм для запуску
   2. Надсилання даного списку клієнту
   3. Підключення клієнта до сервера
   4. Перегляд списку програм на клієнті
   5. Створення списку програм для запуску на клієнті
   6. Очищення списку програм для запуску на клієнті
   7. Запуск вибраних програм сервером
   8. Зупинка всіх програм запущених до того
4. Провести тестування та відлагодження, усунути знайдені проблеми
5. Створити чітку та детальну документацію про проект

## Очікувані результати:

Результатом розробки має стати дві програми: серверна та клієнтська. Серверна програма має надавати доступ для поповнення списку та вдало запускати програми надіслані клієнтом. Клієнтська програма має бути зручною для користувача, надавати можливість комфортно створювати список для запуску та надсилати його серверу.

# Анотація

У цій курсовій роботі розглядається створення програмного інструменту для автоматизації віддаленого керування комп'ютером. Основною метою проекту є розробка програми, яка дозволяє автоматично виконувати дії на комп'ютері з віддаленим доступом, зокрема запускати програми зі списку, наданого сервером.

Проект охоплює кілька важливих етапів, включаючи аналіз існуючих рішень, розробку архітектури системи, реалізацію функціональних можливостей, тестування та відлагодження. У ході роботи описано методи та інструменти, необхідні для створення програмного забезпечення, яке забезпечує віддалене керування комп'ютерами.

Система складається з двох основних частин: серверної та клієнтської програм. Серверна програма дозволяє додавати до списку програми для запуску, відправляти цей список клієнтам, а також забезпечує запуск і зупинку програм на основі запитів клієнтів. Клієнтська програма дозволяє користувачам переглядати список програм, створювати нові списки для запуску, надсилати їх на сервер та очищувати список програм. Інтерфейс користувача розроблений таким чином, щоб забезпечити простоту та зручність використання для користувачів.

Результати тестування показали, що розроблене програмне забезпечення ефективно виконує поставлені завдання, надаючи користувачам можливість надійно та зручно керувати комп'ютерами віддалено. Ця система буде корисною для системних адміністраторів, IT-фахівців та інших користувачів, які потребують надійного інструменту для автоматизації та віддаленого запуску додатків.

Зміст

[Завдання на курсовий проект 2](#_Toc168262791)

[Завдання проекту 2](#_Toc168262792)

[Очікувані результати: 2](#_Toc168262793)

[Анотація 3](#_Toc168262794)

[Вступ 5](#_Toc168262795)

[1. Аналітичний огляд 6](#_Toc168262796)

[1.1 Аналіз методів для віддаленого керування комп’ютером 6](#_Toc168262797)

[1.2 Використання бібліотек та фреймворків 8](#_Toc168262798)

[1.3 Цільова аудиторія 10](#_Toc168262799)

[Висновок 11](#_Toc168262800)

[2. Проектування програми 12](#_Toc168262801)

[2.1 Вибір засобів розробки 12](#_Toc168262802)

[2.2 Високорівневий огляд програми 13](#_Toc168262803)

[2.3 Архітектура програми 13](#_Toc168262804)

[2.4 Функціональні можливості 14](#_Toc168262805)

[Висновок 16](#_Toc168262806)

[3. Реалізація 17](#_Toc168262807)

[3.1 Високорівневий опис реалізації 17](#_Toc168262808)

[3.2 Реалізація дизайну 18](#_Toc168262809)

[3.3 Опис класів 23](#_Toc168262810)

[3.4 Експорт 30](#_Toc168262811)

[Висновок 31](#_Toc168262812)

[4. Тестування 32](#_Toc168262813)

[4.1 Методика тестування 32](#_Toc168262814)

[4.2 Тестування роботи 32](#_Toc168262815)

[Висновок 38](#_Toc168262816)

[Висновок 39](#_Toc168262817)

[Список використаних джерел 40](#_Toc168262818)

[Додаток А – Вихідний код програми сервера 41](#_Toc168262819)

[Додаток Б – Вихідний код програми клієнта 53](#_Toc168262820)

# Вступ

У сучасному світі, де цифрові технології постійно розвиваються, віддалене керування комп'ютерами стає все більш актуальним і необхідним. Це пов'язано з розширенням меж робочих місць, збільшенням кількості мобільних працівників, необхідністю оперативного вирішення технічних проблем і забезпеченням безперебійної роботи IT-інфраструктури. Віддалене керування комп'ютерами дозволяє користувачам ефективно взаємодіяти з комп'ютерними системами, не залежачи від їхнього фізичного місцезнаходження.

Основною метою цього курсового проекту є розробка програмного забезпечення, яке автоматизує процеси віддаленого керування комп'ютером. Зокрема, проект зосереджується на створенні програми, яка дозволяє запускати додатки на віддаленому комп'ютері за допомогою списку, що надається сервером. Це дозволить підвищити ефективність роботи користувачів та зменшити час, необхідний для виконання рутинних завдань.

У рамках цього проекту буде проведено аналіз існуючих рішень для віддаленого керування комп'ютерами з метою визначення їхніх переваг і недоліків. Реалізація функціональних можливостей програмного забезпечення включатиме розробку серверної та клієнтської частин, що забезпечують зручний і надійний інтерфейс для користувачів. Тестування та відлагодження програмного забезпечення є важливими етапами проекту, які дозволять виявити та усунути можливі помилки, забезпечуючи стабільну роботу системи.

Таким чином, розробка програмного забезпечення для автоматизації віддаленого керування комп'ютером є актуальним завданням, що сприятиме підвищенню продуктивності та ефективності роботи користувачів.

# Аналітичний огляд

Віддалене керування комп'ютером - це технологія, що дозволяє користувачам здійснювати доступ і керувати комп'ютером або сервером з будь-якого місця через інтернет. Це забезпечує можливість виконувати завдання, запускати програми, і вирішувати технічні проблеми без фізичної присутності біля пристрою, що значно підвищує зручність і продуктивність роботи.

## 1.1 Аналіз методів для віддаленого керування комп’ютером

Для реалізації віддаленого керування комп'ютером і запуску програм можна використовувати різні методи. Кожен з них має свої переваги та недоліки. Ось основні підходи:

1. Сокети (Sockets)

Сокети дозволяють встановити зв'язок між сервером і клієнтом через мережу, зазвичай використовуючи протокол TCP/IP. Цей метод дозволяє передавати команди від клієнта до сервера для запуску програм.

Переваги:

- Гнучкість і низький рівень доступу до мережевих функцій.

- Висока швидкість передачі даних.

Недоліки:

- Складність реалізації і відладження.

- Потрібні додаткові заходи безпеки для захисту даних, що передаються.

2. HTTP API

HTTP API дозволяє створити веб-сервер, який приймає HTTP-запити для запуску програм. Це можна реалізувати за допомогою бібліотек C++, таких як Boost.Beast або cpp-httplib.

Переваги:

- Простота реалізації та використання.

- Можливість використання існуючих бібліотек і інструментів для роботи з HTTP.

Недоліки:

- Порівняно низька швидкість передачі даних через накладні витрати HTTP.

- Потрібні заходи для забезпечення безпеки, такі як аутентифікація та шифрування.

3. SSH (Secure Shell)

SSH дозволяє виконувати команди на віддаленому комп'ютері за допомогою захищеного з'єднання. Це можна реалізувати за допомогою бібліотеки libssh.

Переваги:

- Високий рівень безпеки завдяки шифруванню.

- Широке використання в адміністративних задачах.

Недоліки:

- Складність налаштування та використання.

- Залежність від зовнішніх бібліотек.

4. RPC (Remote Procedure Call)

RPC дозволяє виконувати функції на віддаленому сервері як локальні виклики. Це можна реалізувати за допомогою бібліотеки gRPC.

Переваги:

- Зручність у використанні, близьке до локальних викликів функцій.

- Підтримка різних мов програмування і платформ.

Недоліки:

- Складність налаштування інфраструктури.

- Додаткові накладні витрати на забезпечення безпеки та продуктивності.

Я обрав використання сокетів для мого проєкту через їх високу швидкість передачі даних і гнучкість. Сокети забезпечують низький рівень доступу до мережевих функцій, що є ключовим для ефективної взаємодії між клієнтом і сервером. Хоча реалізація та відлагодження можуть бути складнішими, ці труднощі виправдані перевагами. Інші методи, такі як HTTP API, SSH та RPC, також мають свої переваги, але їх вивчення та дослідження потребують більше часу, що не є оптимальним в рамках обмеженого часу на реалізацію цього проєкту.

## 1.2 Використання бібліотек та фреймворків

1. Qt

Qt – це кросплатформений фреймворк для створення графічних інтерфейсів користувача та додатків. Він забезпечує багатий набір бібліотек для роботи з графікою, мережами, базами даних та іншим.

Переваги:

- Кросплатформеність;

- Широкий функціонал;

- Зручний дизайн

2. Qt Creator

Qt Creator – це інтегроване середовище розробки (IDE) для роботи з фреймворком Qt. Воно надає інструменти для написання, відлагодження та тестування коду.

Переваги:

- Інтеграція з Qt;

- Зручність;

- Кросплатформеність.

3. QTcpSocket

QTcpSocket – клас бібліотеки Qt для роботи з TCP-з'єднаннями. Дозволяє створювати клієнтські додатки, які можуть підключатися до сервера, передавати та отримувати дані.

Переваги:

- Простота використання;

- Інтеграція з Qt, підтримка сигналів і слотів;

- Асинхронна модель, ефективна обробка мережевих подій.

4. Windows API

Windows API – це набір функцій, макросів, констант та структур, що дозволяє взаємодіяти безпосередньо з операційною системою Windows. Використовується для виконання низькорівневих операцій, роботи з вікнами, обробки подій тощо.

Переваги:

- Низькорівневі функції, які забезпечують прямий доступ до ОС;

- Широкий спектр завдань;

- Висока продуктивність.

Аналіз відомих програм для віддаленого керування ПК

У світі існує кілька програм для віддаленого керування комп'ютером, кожна з яких має свої переваги та недоліки.

Один із найпопулярніших варіантів - TeamViewer. Він відомий своєю кросплатформеністю та простотою використання, але має певні обмеження безпеки та вартість.

Друга альтернатива - AnyDesk, відзначається високою продуктивністю та безпекою, але може бути обмежена в безкоштовній версії.

Також варто звернути увагу на Microsoft Remote Desktop, інтеграція з Windows якого спрощує роботу, але його використання може бути складним для користувачів інших платформ.

Chrome Remote Desktop - це безкоштовний інструмент з кросплатформеністю, але він може мати обмежений функціонал та залежить від браузера Chrome.

Зацікавить VNC Connect, що відзначається гнучкістю та безпекою, але його продуктивність може бути меншою, а вартість - вищою. Оберіть програму з урахуванням своїх потреб та переваг.

## 1.3 Цільова аудиторія

Цільова аудиторія для програми, включає у себе не лише системних адміністраторів та технічних спеціалістів, але й звичайних користувачів, які мають потребу у зручному та ефективному управлінні своїми комп'ютерами. Системні адміністратори та технічні спеціалісти використовують такі програми для керування великими мережами комп'ютерів або серверів, що дозволяє їм віддалено запускати різноманітні програми та процеси.

Звичайні користувачі також відчувають потребу у такій програмі, оскільки вона дозволяє їм зекономити час і зручно керувати робочими процесами. Наприклад, вони можуть використовувати цей інструмент для віддаленого запуску програми на своєму комп'ютері з віддаленої локації, щоб підготувати систему до роботи або запустити процес, який може зайняти тривалий час, поки вони не будуть фізично доступні до комп'ютера.

## Висновок

Вибір сокетів для реалізації віддаленого керування комп'ютером у цьому проєкті обґрунтований їх високою швидкістю передачі даних і гнучкістю, що є критично важливим для ефективної взаємодії між клієнтом і сервером. Хоча інші методи, такі як HTTP API, SSH та RPC, також мають свої переваги, їх дослідження потребує більше часу, що не відповідає обмеженим термінам проєкту. Використання бібліотек і фреймворків, таких як Qt, Qt Creator, QTcpSocket і Windows API, дозволяє створити потужний, зручний та продуктивний інтерфейс для користувачів. Аналіз відомих програм для віддаленого керування комп'ютером, таких як TeamViewer, AnyDesk, Microsoft Remote Desktop, Chrome Remote Desktop та VNC Connect, допомагає зрозуміти їх переваги та недоліки, що сприяє вибору найкращого підходу для розробки.

# Проектування програми

## 2.1 Вибір засобів розробки

Вибір засобів розробки для даного проекту є стратегічно важливим етапом, оскільки від цього залежить подальший успіх програми.

Мова програмування С++ була обрана з огляду на її високу продуктивність, низький рівень абстракції та широкі можливості для оптимізації та розширення функціоналу. С++ є потужним інструментом для розробки ефективних та швидких програм, що особливо важливо для проектів, які працюють з інтенсивними обчисленнями або вимагають високої продуктивності.

Qt Creator, як інтегроване середовище розробки, було вибрано через його потужність та зручність. Qt Creator надає широкий набір інструментів для створення крос-платформених програм з графічним інтерфейсом, що дозволяє ефективно розробляти програми для різних операційних систем. Крім того, він ідеально поєднується з мовою С++, що полегшує процес розробки та забезпечує більш гнучкий підхід до реалізації функціоналу.

Windows API було обрано для забезпечення взаємодії програми з операційною системою Windows на найнижчому рівні. Це дає можливість отримати доступ до різноманітних системних функцій та можливостей, таких як керування процесами, робота з файловою системою та введенням/виведенням. Використання Windows API дозволяє отримати повний контроль над програмою та максимально використовувати можливості операційної системи для досягнення поставлених цілей.

## 2.2 Високорівневий огляд програми

Програма є інструментом для віддаленого керування комп'ютером, який дозволяє користувачам віддалено запускати програми на віддаленому комп'ютері через мережу. Програма має зручний графічний інтерфейс, що дозволяє користувачам легко взаємодіяти з нею, створювати список програм для запуску та надсилати його серверу. Основні функції включають поповнення списку програм на сервері, надсилання списку клієнту, перегляд та редагування списку програм на клієнті, а також запуск обраних програм на сервері.

## 2.3 Архітектура програми

Проект має дві програми, тому і описано дві архітектури. Компоненти взаємодіють між собою через внутрішню логіку програми.

Сервер:

Інтерфейс адміністратора (UI). Інтерфейс розроблений з використанням QT creator, завдяки чому всі модулі прив’язані до елементів інтерфейсу можуть взаємодіяти між собою. Функціонально реалізовано відображення списку можливих програм для запуску, з їх розташуванням на комп’ютері, поля для вводу, шляху до програми та її ім’я, та кнопка додавання цієї програми в список.

Модуль зв’язку. Розроблений з використанням класу бібліотеки QT -QTcpSocket. Забезпечує з’єднання з клієнтом та безпечне пересилання даних. Серед функцій: запуск сервера, перевірка вхідного приєднання, перевірка стану з’єднання з клієнтом, надсилання та отримання даних.

Модуль роботи з процесами. Реалізований через Win API з використанням структур бібліотеки Qt. Забезпечує роботу з списком програм на пристрої, зберігає його на пристрої та зчитує при запуску. Запуск програм, список яких прийшов з клієнта, зупинка інших процесів запущених до того.

Клієнт:

Інтерфейс користувача (UI). Інтерфейс розроблений з використанням QT creator, завдяки чому всі модулі прив’язані до елементів інтерфейсу можуть взаємодіяти між собою. Функціонально реалізовано: відображення списку можливих програм для запуску у вигляді випадаючого списку, з можливістю додавання вибраної програми в список, список відображаючий поточний список на запуск, кнопка очищення списку та кнопка запуску програм, індикатор з’єднання та кнопка підключення.

Модуль зв’язку. Так як і на сервері розроблений з використанням класу бібліотеки QT - QTcpSocket. Забезпечує з’єднання з сервером та безпечне пересилання даних. Серед функцій: запуск сокета клієнта, спроба підключення до сервера, перевірка стану з’єднання з клієнтом, надсилання та отримання даних.

Модуль роботи з списком. При розробці використано бібліотеку QT. Керує відображенням списку на екран, перевіркою дублікатів в списку та очищенням списку.

## 2.4 Функціональні можливості

Поповнення на сервері списку програм для запуску. Функція дозволяє поповняти список з інтерфейсу сервера. Вимоги: переформатування шляху, для можливості подальшого запуску функцією запуску процесу, збереження назви та шляху у файли, відображення нового списку на екран.

Надсилання даного списку клієнту Функція реалізована для передавання списку TCP сокетом. Вимоги: Об’єднання списку імен програм в один пакет, та надсилання його на клієнт кожні 5 секунд.

Підключення клієнта до сервера. Пробує підключитися до сервера та налагодити пересилання даних. Вимоги: По натисканню кнопки почати підключення до сервера, якщо воно вдале, налагодити зв'язок та змінити індикатора на зелений.

Перегляд списку програм на клієнті. Реалізовує відображення списку у випадаючому списку. Вимоги: Зчитування рядку з буфера сокета, форматування його в список та відображення його у випадаючому списку.

Створення списку програм для запуску на клієнті. Функція дозволяє поповняти список запуску вибраною програмою. Вимоги: Зчитування назви вибраної програми з списку, перевірка чи ця програма вже не була додана, додавання назви в список, якщо ще не додана, відображення оновленого списку на екран.

Очищення списку програм для запуску на клієнті. Проста функція, яка очищує список. Вимоги: очищення списку для надсилання, очищення списку, що відображається.

Запуск вибраних програм сервером. Функція, яка при отримані списку на запуск, запускає програми зі списку по їхньому шляху. Вимоги: форматування рядка в список, запуск функції на зупинку попередніх процесів, знаходження в списку шляхів потрібний, через який буде запущено новий процес.

Зупинка всіх програм запущених до того. При отримані команди зупиняє всі процеси, які були запущені сервером до того. Вимоги: зчитування списку процесів що запущені та їх зупинка.

## Висновок

Вибір засобів розробки для даного проєкту був здійснений на основі їх відповідності вимогам продуктивності, гнучкості та зручності. Мова програмування C++ обрана через її високу продуктивність та можливості для оптимізації, що є важливими для інтенсивних обчислювальних задач. Qt Framework та Qt Creator забезпечують потужний інструментарій для створення кросплатформених програм з графічним інтерфейсом, що полегшує розробку та тестування коду. Використання Windows API дозволяє здійснювати низькорівневі операції, такі як керування процесами та робота з файловою системою, забезпечуючи повний контроль над програмою та максимальну продуктивність.

# 3. Реалізація

## 3.1 Високорівневий опис реалізації

Проект спрямований на створення серверної та консольної програм, які дозволяють віддалено запускати програми на комп'ютері. Реалізація проекту передбачає використання декількох ключових технологій та компонентів, кожен з яких відіграє важливу роль у забезпеченні функціональності та ефективності програми.

Технології та інструменти

1. Мова програмування C++:

- Основна мова, яка використовується для розробки програми, завдяки своїй високій продуктивності та широким можливостям для оптимізації.

2. Qt Framework:

- Використовується для створення графічного інтерфейсу користувача (GUI) та роботи з мережею. Qt надає потужні інструменти для розробки крос-платформених програм з привабливим та зручним інтерфейсом.

3. Qt Creator:

- Інтегроване середовище розробки (IDE), яке використовується для написання, налагодження та тестування коду. Qt Creator забезпечує зручний інтерфейс для роботи з проектом та підтримує всі необхідні інструменти для розробки на C++.

4. Windows API:

- Набір функцій та інтерфейсів, що використовуються для взаємодії з операційною системою Windows. Windows API дозволяє програмі здійснювати низькорівневі операції, такі як керування процесами та взаємодія з файловою системою.

Обрані засоби розробки забезпечують ефективну та надійну реалізацію серверної програми для віддаленого запуску додатків. Мова програмування C++ пропонує високу продуктивність і можливості для оптимізації. Qt Framework надає потужні інструменти для створення графічного інтерфейсу користувача та забезпечує крос-платформеність. Qt Creator полегшує процес написання, налагодження та тестування коду, підвищуючи продуктивність розробників. Використання Windows API дозволяє здійснювати низькорівневі операції, такі як керування процесами та взаємодія з файловою системою, що необхідно для повної функціональності серверної програми. Ці інструменти створюють потужну основу для реалізації проекту.

## 3.2 Реалізація дизайну

Qt Creator забезпечує високу зручність роботи над проектом завдяки своєму інтуїтивному інтерфейсу, який інтегрує всі необхідні інструменти для розробки програм на C++. Ця інтегроване середовище розробки (IDE) підтримує функції написання, налагодження та тестування коду, що дозволяє розробникам ефективно керувати проектами. Крім того, Qt Creator забезпечує легкий доступ до бібліотек Qt, що спрощує створення графічних інтерфейсів і роботи з мережею, роблячи процес розробки швидким і зручним.

Основні елементи інтерфейсу сервера:

* Основне вікно. Містить елементи керування списком та таблицю, що відображає список програм та шлях до їх запуску.

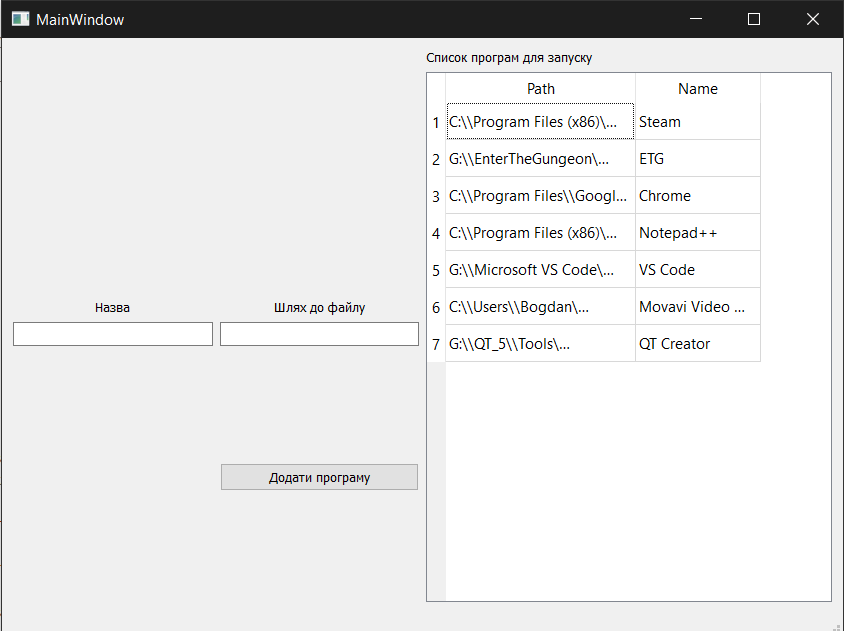


Рис. 3.2.1 Вікно програми сервера

* Поля вводу. Два поля вводу, для внесення шляху запуску програми та запису ім’ені, під яким буде збережено цей шлях.



Рис. 3.2.2 Поля вводу

* Кнопка збереження. Зберігає шлях та ім’я, внесені в поля, до файлів з списками.

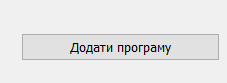


Рис. 3.2.3 Кнопка збереження

* Відображення списку. Елемент, який відображає список шляхів для запуску та назв, під якими вони збережені.

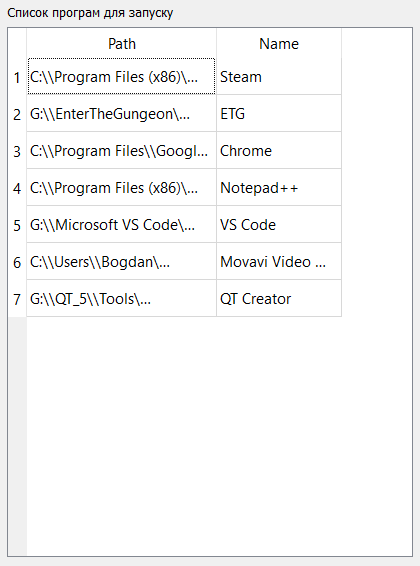


Рис. 3.2.4 Відображення списку

Основні елементи інтерфейсу клієнта:

* Основне вікно. Містить елементи керування списком надсилання, кнопку підключення, індикатор підключення, кнопку надсилання списку на сервер та таблицю, що відображає список програм для надсилання на сервер.

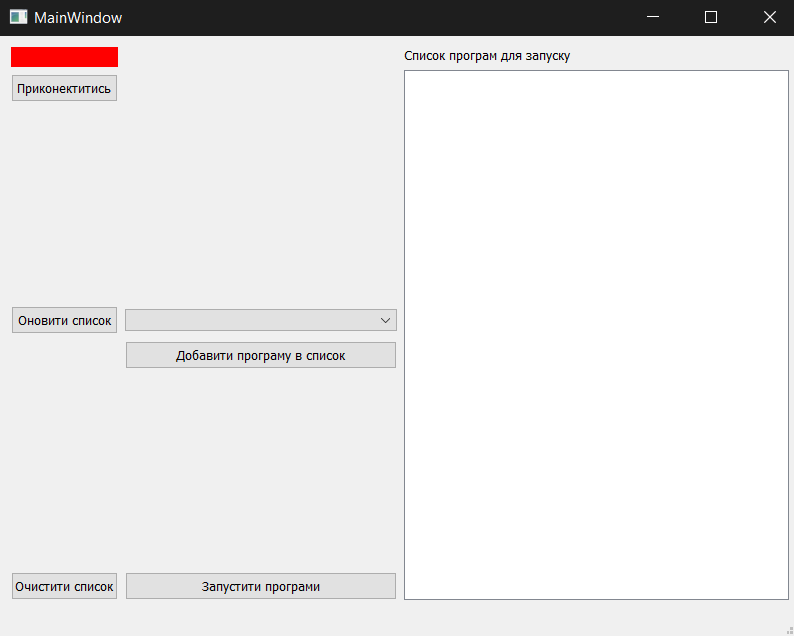


Рис. 3.2.5 Вікно програми клієнта

* Випадаючий список. Відображає список програм, які можна добавити в список запуску, дозволяє вибрати програму, яку можна буде потім добавити в список на запуск.



Рис. 3.2.6 Випадаючий список

* Кнопка додавання програми в список. Додає в список на надсилання ім’я програми, яка має бути запущена, та оновлює список відображення.



Рис. 3.2.7 Кнопка додавання програми в список

* Кнопка підключення. При натисканні, пробує з’єднатись з сервером та налаштувати надсилання даних (при вдалій спробі, змінює колір індикатора на зелений).

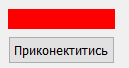


Рис. 3.2.8 Кнопка підключення

* Індикатор підключення. Відображає стан підключення (зелений – з’єднання вдале, червоний – не з’єднано).

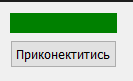


Рис. 3.2.9 Індикатор підключення

* Кнопка очищення списку. Очищує список для надсилання та оновлює таблицю.



Рис. 3.2.10 Кнопка очищення

* Кнопка надсилання списку. Форматує список для надсилання в рядок та надсилає його на сервер, після чого очищує список та таблицю.



Рис. 3.2.11 Кнопка надсилання

* Відображення списку. Елемент, який відображає назви програм, які будуть запущені.

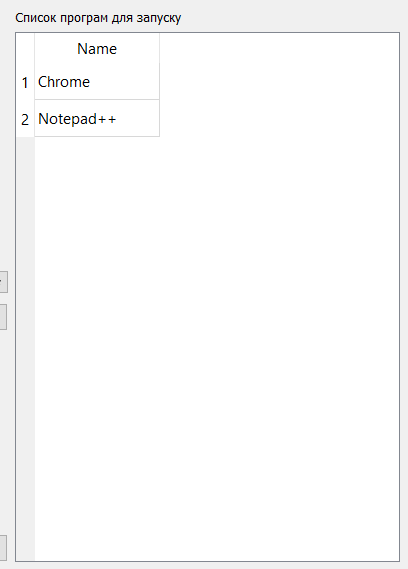


Рис. 3.2.12 Відображення списку

## 3.3 Опис класів

Сервер:

MainWindow

Клас MainWindow є головним вікном програми та успадковує клас QMainWindow з бібліотеки Qt. Він відповідає за взаємодію користувача з графічним інтерфейсом, управління серверним з'єднанням та виконанням програм. Нижче наведено детальний опис полів та методів класу.

Публічні методи

* MainWindow(QWidget \*parent = nullptr): Конструктор класу, який ініціалізує головне вікно. Приймає вказівник на батьківський об'єкт, за замовчуванням - nullptr.
* ~MainWindow(): Деструктор класу, який звільняє ресурси, що використовуються головним вікном.

Публічні поля

* cConnection server: Об'єкт для управління серверним з'єднанням.
* QTimer timerToSent: Таймер для керування періодичною відправкою даних.
* QVector<QString> listOfNames: Вектор для зберігання імен.
* QVector<QString> listOfWays: Вектор для зберігання шляхів.
* QStringList splitList: Список для зберігання розділених даних.
* std::vector<PROCESS\_INFORMATION> processes: Вектор для зберігання інформації про запущені процеси.

Приватні слоти

* void startSending(): Слот для початку відправки даних.
* void stopTimer(): Слот для зупинки таймера.
* void setInfo(): Слот для встановлення інформації.
* void updateList(): Слот для оновлення списку.
* void on\_bAdd\_clicked(): Слот, який викликається при натисканні кнопки "Add".
* bool saveFilePath(const QString& filePath, const QString& fileName): Метод для збереження шляху до файлу та його імені.
* bool readFilePaths(): Метод для зчитування шляхів до файлів.
* bool deleteFilePath(const QString& targetPath, const QString& fileName): Метод для видалення шляху до файлу та його імені.
* void startPrograms(const QStringList& programs): Метод для запуску програм.
* void stopPrograms(): Метод для зупинки програм.

Приватні поля

* \*Ui::MainWindow ui: Вказівник на інтерфейс користувача, згенерований дизайнером форм Qt.

TCPConnection

Клас cConnection відповідає за встановлення, підтримку та керування TCP-з'єднанням як сервер, успадковуючи функціонал від QTcpServer з бібліотеки Qt. Він забезпечує обробку вхідних з'єднань, управління клієнтами, передачу даних між сервером і клієнтами, а також перевірку стану з'єднань.

Приватні поля

* QVector<QTcpSocket> vSockets\*: Вектор для зберігання вказівників на активні сокети клієнтів.
* QVector<QString> vSocketsName: Вектор для зберігання імен клієнтів, пов'язаних із сокетами.
* QByteArray dataBuf: Буфер для зберігання даних, що передаються.
* QVector<qintptr> sockIndexToSend: Вектор для зберігання індексів сокетів, до яких потрібно відправити дані.

Публічні поля

* QTcpSocket socket\*: Вказівник на поточний сокет, ініціалізований як nullptr.
* QString strDate: Змінна для зберігання данихи як рядка.
* struct socketsState: Структура для зберігання стану сокетів:
* QVector<qintptr> socketIndex: Вектор для зберігання індексів сокетів.
* QVector<QString> socketName: Вектор для зберігання імен сокетів.
* QVector<bool> socketState: Вектор для зберігання стану сокетів (підключений або відключений).

Публічні методи

* cConnection(): Конструктор класу, який ініціалізує об'єкт cConnection.
* ~cConnection(): Деструктор класу, який звільняє ресурси, що використовуються cConnection.
* bool startServer(): Метод для запуску сервера.
* void incomingConnection(qintptr sockDesc): Метод для обробки вхідних з'єднань, успадкований від QTcpServer.
* bool conClient(): Метод для підключення клієнта.
* QString sockReady(): Метод для перевірки готовності сокета.
* bool sockDisc(): Метод для відключення сокета.
* bool conClToHost(): Метод для підключення клієнта до хоста.
* bool SendToClient(const QString &msg): Метод для відправки повідомлення конкретному клієнту.
* bool SendToAllClients(const QString &msg): Метод для відправки повідомлення всім клієнтам.
* bool isSocketConnected(QTcpSocket \*socket): Метод для перевірки, чи підключений сокет.
* bool isClientConnected(QTcpSocket \*socket): Метод для перевірки, чи підключений клієнт.
* void sendStringToHost(const QString &msg): Метод для відправки рядка на хост.
* bool checkState(const QString &sockName, QTcpSocket \*socket): Метод для перевірки стану сокета.
* void checkDelList(const int index): Метод для перевірки та видалення сокета зі списку за індексом.

Клієнт:

Клас MainWindow є головним вікном програми та успадковує клас QMainWindow з бібліотеки Qt. Він відповідає за взаємодію користувача з графічним інтерфейсом, керування клієнтським TCP-з'єднанням, таймером для періодичних подій, а також управлінням списком програм для запуску.

Публічні методи

* MainWindow(QWidget \*parent = nullptr): Конструктор класу, який ініціалізує головне вікно. Приймає вказівник на батьківський об'єкт, за замовчуванням nullptr.
* ~MainWindow(): Деструктор класу, який звільняє ресурси, що використовуються головним вікном.

Публічні поля

* cConnection clientSock: Об'єкт класу cConnection, який відповідає за TCP-з'єднання та керування клієнтським сокетом.
* QTimer timer: Таймер для керування періодичними подіями.
* QStringList splitList: Список для зберігання розділених даних.
* QStringList listToStart: Список програм для запуску.

Приватні слоти

* void startTimer(): Слот для запуску таймера.
* void stopTimer(): Слот для зупинки таймера.
* bool setInfo(): Слот для встановлення інформації, яка може бути відправлена або збережена.
* void on\_bConnect\_clicked(): Слот, який викликається при натисканні кнопки "Connect", для підключення до сервера.
* void on\_bStart\_clicked(): Слот, який викликається при натисканні кнопки "Start", для запуску обраних програм.
* void updateList(): Слот для оновлення списку програм на екрані.
* void onComboBoxShowPopup(): Слот, який викликається при відкритті випадаючого списку, для його оновлення або заповнення.
* void on\_bUpdBox\_clicked(): Слот, який викликається при натисканні кнопки "Update Box", для оновлення вмісту випадаючого списку.
* void on\_bAddAppToList\_clicked(): Слот, який викликається при натисканні кнопки "Add App To List", для додавання нової програми до списку.
* void on\_bClear\_clicked(): Слот, який викликається при натисканні кнопки, для очищення списку.

TCPConnection

Клас cConnection відповідає за встановлення, підтримку та керування TCP-з'єднанням як сервер, успадковуючи функціонал від QTcpServer з бібліотеки Qt. Він забезпечує обробку вхідних з'єднань, управління клієнтами, передачу даних між сервером і клієнтами, а також перевірку стану з'єднань.

Приватні поля

* QVector<QTcpSocket> vSockets\*: Вектор для зберігання вказівників на активні сокети клієнтів.
* QVector<QString> vSocketsName: Вектор для зберігання імен клієнтів, пов'язаних із сокетами.
* QByteArray dataBuf: Буфер для зберігання даних, що передаються.
* QVector<qintptr> sockIndexToSend: Вектор для зберігання індексів сокетів, до яких потрібно відправити дані.

Публічні поля

* QTcpSocket socket\*: Вказівник на поточний сокет, ініціалізований як nullptr.
* QString strDate: Змінна для зберігання данихи як рядка.
* struct socketsState: Структура для зберігання стану сокетів:
* QVector<qintptr> socketIndex: Вектор для зберігання індексів сокетів.
* QVector<QString> socketName: Вектор для зберігання імен сокетів.
* QVector<bool> socketState: Вектор для зберігання стану сокетів (підключений або відключений).

Публічні методи

* cConnection(): Конструктор класу, який ініціалізує об'єкт cConnection.
* ~cConnection(): Деструктор класу, який звільняє ресурси, що використовуються cConnection.
* bool startServer(): Метод для запуску сервера.
* void incomingConnection(qintptr sockDesc): Метод для обробки вхідних з'єднань, успадкований від QTcpServer.
* bool conClient(): Метод для підключення клієнта.
* QString sockReady(): Метод для перевірки готовності сокета.
* bool sockDisc(): Метод для відключення сокета.
* bool conClToHost(): Метод для підключення клієнта до хоста.
* bool SendToClient(const QString &msg): Метод для відправки повідомлення конкретному клієнту.
* bool SendToAllClients(const QString &msg): Метод для відправки повідомлення всім клієнтам.
* bool isSocketConnected(QTcpSocket \*socket): Метод для перевірки, чи підключений сокет.
* bool isClientConnected(QTcpSocket \*socket): Метод для перевірки, чи підключений клієнт.
* void sendStringToHost(const QString &msg): Метод для відправки рядка на хост.
* bool checkState(const QString &sockName, QTcpSocket \*socket): Метод для перевірки стану сокета.
* void checkDelList(const int index): Метод для перевірки та видалення сокета зі списку за індексом.

## 3.4 Експорт

Експорт проекту в формат .exe дозволяє кінцевим користувачам запускати програму без необхідності встановлювати додаткові інструменти або середовище розробки. Це значно полегшує розповсюдження та встановлення програми на інших комп'ютерах, а також забезпечує захист коду від зворотної інженерії. Виконуваний файл .exe може містити всі необхідні бібліотеки та ресурси, що робить програму самодостатньою і зручною для використання, надаючи їй професійний вигляд і підвищуючи довіру користувачів.

## Висновок

Проект спрямований на створення серверної та консольної програм, які дозволяють віддалено запускати програми на комп'ютері, використовуючи ключові технології та компоненти для забезпечення функціональності та ефективності. Основною мовою програмування є C++ завдяки її високій продуктивності та можливостям для оптимізації. Qt Framework використовується для створення графічного інтерфейсу та роботи з мережею, надаючи потужні інструменти для розробки крос-платформених програм.

Обрані засоби розробки забезпечують ефективну та надійну реалізацію серверної програми для віддаленого запуску додатків. C++ пропонує високу продуктивність, Qt Framework надає інструменти для створення зручного графічного інтерфейсу та крос-платформеність, а Qt Creator полегшує розробку. Використання Windows API дозволяє здійснювати необхідні низькорівневі операції для повної функціональності серверної програми. Ці інструменти створюють потужну основу для реалізації проекту, забезпечуючи зручність, продуктивність та надійність віддаленого керування програмами.

# 4. Тестування

## 4.1 Методика тестування

Ми обираємо ручне тестування для нашого проекту з урахуванням його складного функціоналу та особливостей взаємодії з апаратним забезпеченням та зовнішніми сервісами. Це дозволить нам відтворювати реальні умови використання та виявляти непередбачені проблеми, що можуть виникнути в реальному середовищі. Окрім того, на ранніх етапах розробки цей підхід буде ефективним, оскільки функціонал програми змінюється швидко і ми можемо швидко реагувати на зміни без необхідності в розробці та підтримці автоматизованих тестів.

## 4.2 Тестування роботи

Спробуймо додати програму в список.

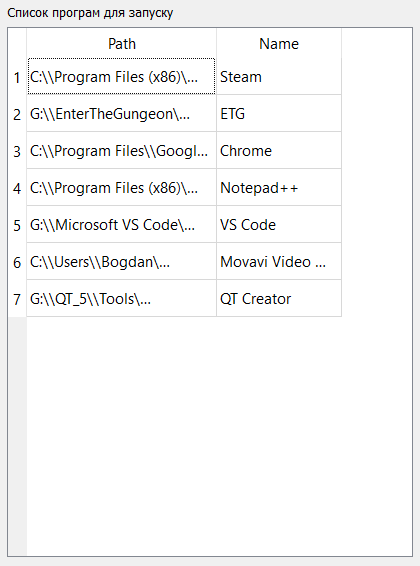


Рис. 4.2.1 Список на сервері до додавання

Вносимо ім’я, під яким буде внесено програму та шлях до цієї програми. Натискаємо кнопку «Додати програму».

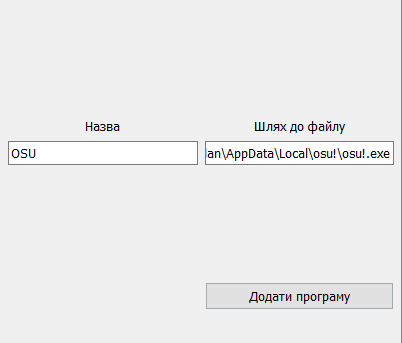


Рис. 4.2.2 Внесені дані

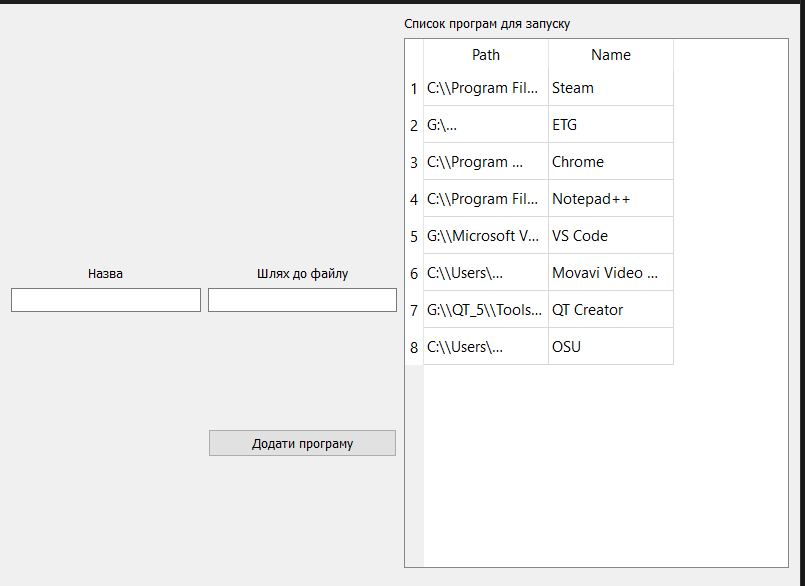


Рис. 4.2.3 Список на сервері після додавання

Тепер спробуємо приєднатись до сервера. Колір індикатора має змінитися на зелений.

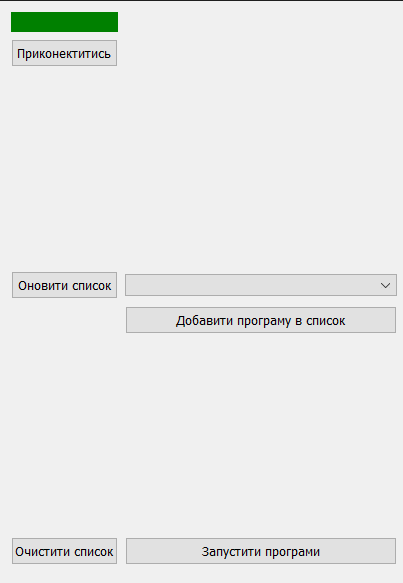


Рис. 4.2.4 Стан підключення

Як ми бачимо, підключення пройшло успішно. Тепер ми можемо оновити випадаючий список, натиснувши на кнопку «Оновити список».

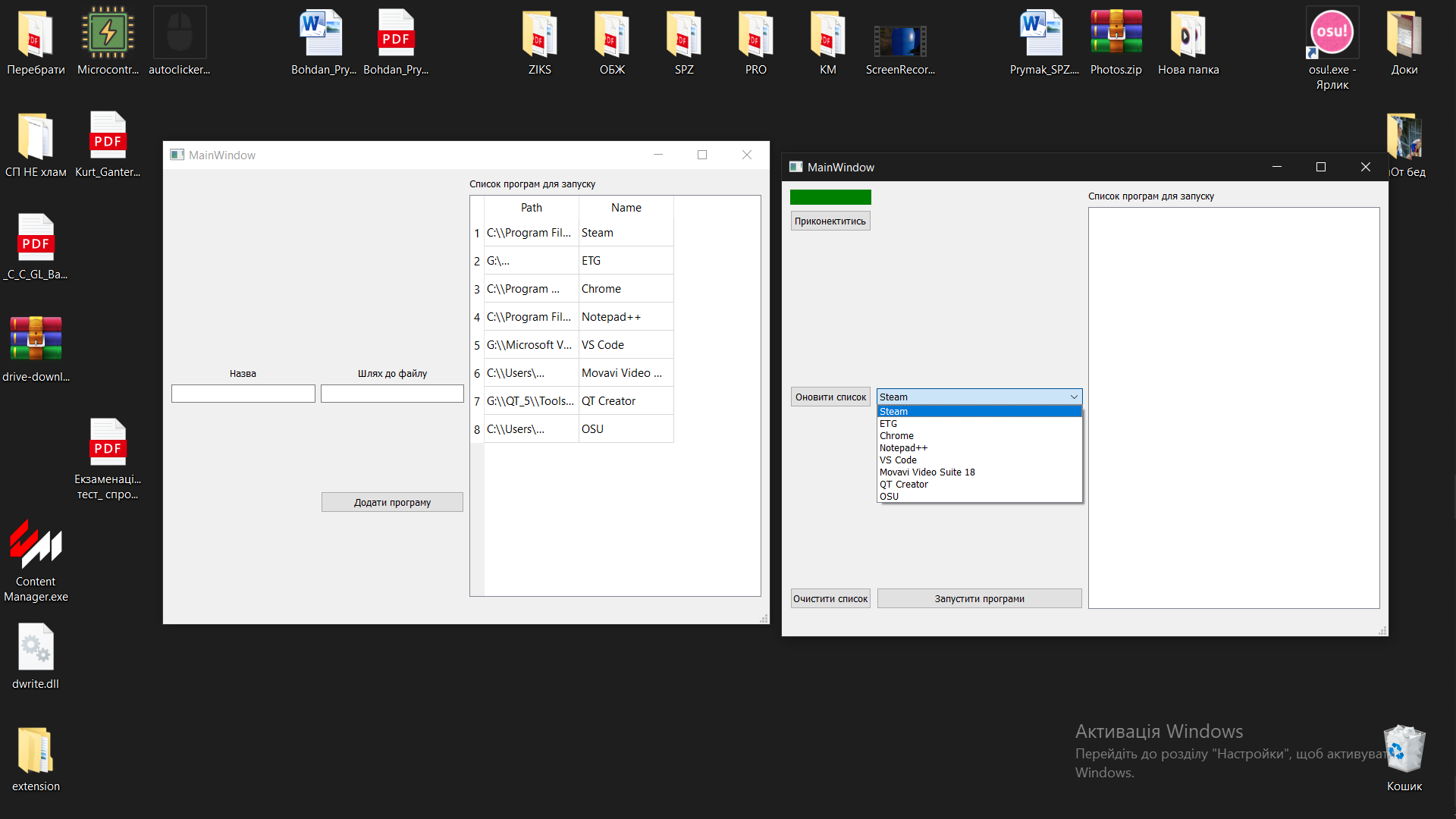


Рис. 4.2.5 Випадаючий список

На рисунку зображено вдале оновлення списку. Тепер можна вибрати програму, яка буде додана в список.

Вибираємо Notepad++ та натискаємо кнопку «Добавити програму в список».

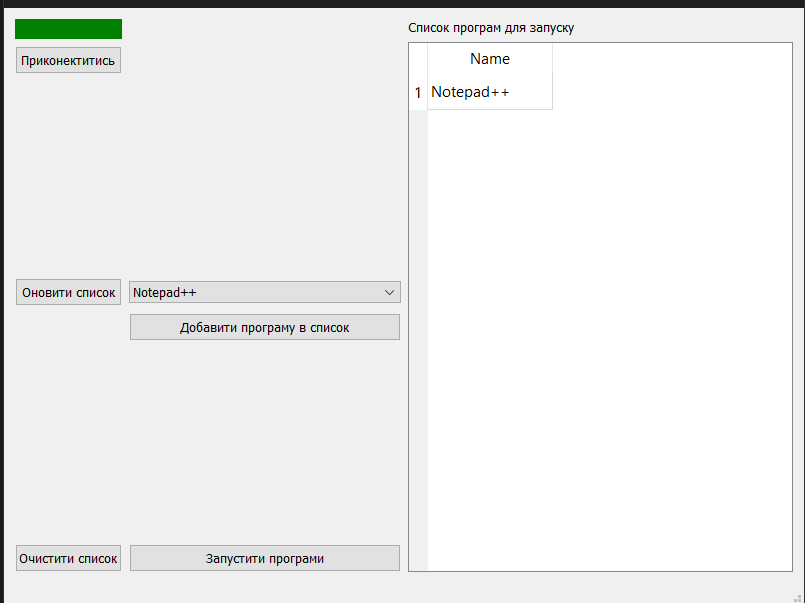


Рис. 4.2.6 Список після додавання

Як ми бачимо, програма була успішно додана в список. Так само добавимо до запуску Chrome.

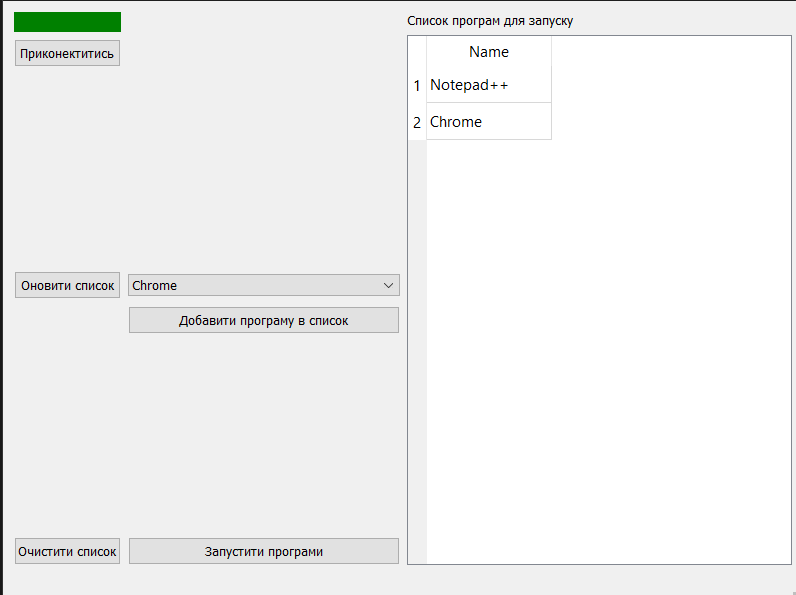


Рис. 4.2.7 Список після додавання

Тепер можна надіслати список на сервер, для подальшого запуску. На рисунку 4.2.8 зображено список програм, які запущені до отримання списку

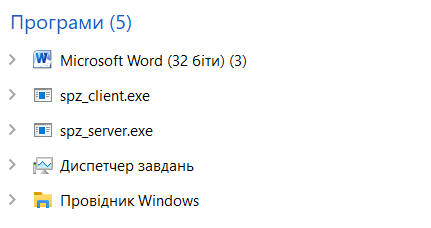


Рис. 4.2.8 Список запущених програм до отримання списку

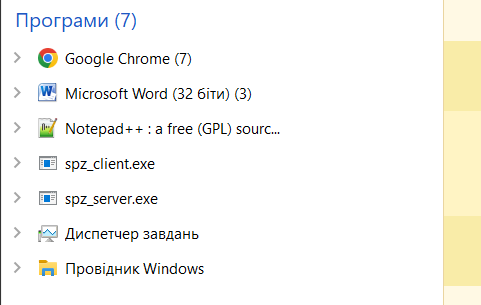


Рис. 4.2.9 Список запущених програм після отримання списку

На рисунку 4.2.9 видно, що на сервері програми були вдало запущені. Спробуємо запустити наступний список (рис. 4.2.10).

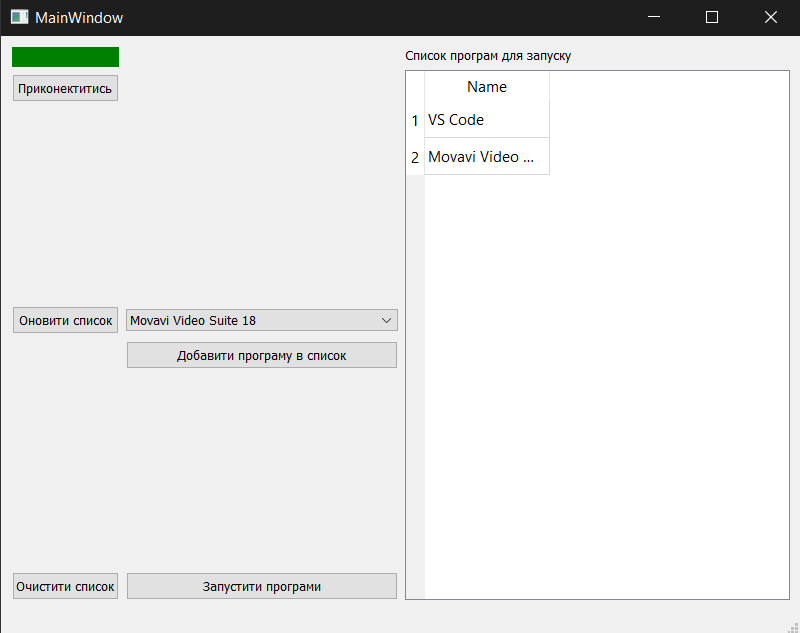


Рис. 4.2.10 Список надсилання

На наступному рисунку (4.2.11), чудово видно, що програми, які були запущені до цього, припинили роботу, а програми з нового списку запущені.

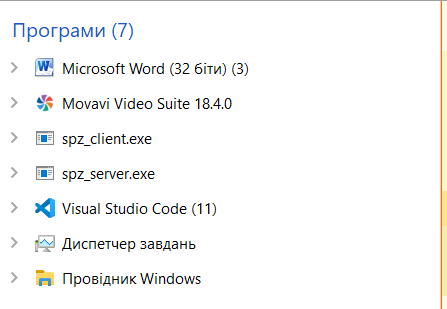


Рис. 4.2.9 Список запущених програм після отримання другого списку

## Висновок

Ручне тестування програми виявилося ефективним методом перевірки правильності її роботи. Шляхом вручного введення різноманітних вхідних даних та виконання різних сценаріїв взаємодії з програмою було підтверджено її стабільність і відповідність функціональним вимогам. Виявлення та виправлення помилок під час ручного тестування сприяло покращенню якості програми та забезпечило її надійність у реальних умовах використання.

# Висновок

В результаті виконання курсового проекту було розроблено серверну та консольну програми, спрямовані на віддалене керування комп'ютером шляхом запуску програм на відстані через мережу.

У процесі розробки курсового проекту, було використано комплекс технологій, які істотно вплинули на успішність реалізації програми. Перш за все, вибір мови програмування С++ був обумовлений не лише її високою продуктивністю, а й можливістю здійснення глибокої оптимізації та розширення функціоналу. Крім того, використання Qt Framework та Qt Creator забезпечило зручну розробку графічного інтерфейсу користувача.

Щодо інтерфейсу програми, важливим аспектом була зручність взаємодії з користувачем. Елементи керування були спроектовані таким чином, щоб вони були інтуїтивно зрозумілими та легко доступними, що сприяло зручності використання програми для користувачів будь-якого рівня навичок. Крім того тестування допомогло виявити та виправити потенційні проблеми, що могли виникнути в різних сценаріях використання програми, забезпечивши високу надійність та коректність роботи програмного забезпечення

# Список використаних джерел

1. <https://doc.qt.io/qt-5/>
2. <https://stackoverflow.com/>
3. <https://learn.microsoft.com/uk-ua/previous-versions/office/office-365-api/api/version-2.0/task-rest-operations>

# Додаток А – Вихідний код програми сервера

mainwindow.h

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QTimer>

#include <QStandardItemModel>

#include <QFile>

#include <QTextStream>

#include <QString>

#include <QVector>

#include <windows.h>

#include "TCPConnection.h"

QT\_BEGIN\_NAMESPACE

namespace Ui { class MainWindow; }

QT\_END\_NAMESPACE

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

MainWindow(QWidget \*parent = nullptr);

~MainWindow();

cConnection server;

QTimer timerToSent;

QVector<QString> listOfNames;

QVector<QString> listOfWays;

QStringList splitList;

std::vector<PROCESS\_INFORMATION> processes;

private slots:

void startSending();

void stopTimer();

void setInfo();

void updateList();

void on\_bAdd\_clicked();

bool saveFilePath(const QString& filePath, const QString& fileName);

bool readFilePaths();

bool deleteFilePath(const QString& targetPath, const QString& fileName);

void startPrograms(const QStringList& programs);

void stopPrograms();

private:

Ui::MainWindow \*ui;

};

#endif // MAINWINDOW\_H

TCPConnection.h

#ifndef TCPCONNECTION\_H

#define TCPCONNECTION\_H

#include <QTcpServer>

#include <QTcpSocket>

#include <QVector>

#include <QDataStream>

#include <QTimer>

#define PORT\_TO\_CONNECT 5555

#define ADDR\_TO\_CONNECT "127.0.0.1"

#define THREE\_SEC 3000

#define FIVE\_SEC 5000

#define ONE\_MIN 60000

class **cConnection**: public QTcpServer

{

Q\_OBJECT

private:

QVector <QTcpSocket\*> vSockets;

QVector <QString> vSocketsName;

QByteArray dataBuf;

QVector <qintptr> sockIndexToSend;

public:

**cConnection**();

~***cConnection***();

QTcpSocket\* socket = nullptr;

QString strDate;

struct **socketsState**

{

QVector <qintptr> socketIndex;

QVector <QString> socketName;

QVector <bool> socketState;

}sockState;

bool **startServer**();

void ***incomingConnection***(qintptr sockDesc);

bool **conClient**();

QString **sockReady**();

bool **sockDisc**();

bool **conClToHost**();

bool **SendToClient**(const QString &msg);

bool **SendToAllClients**(const QString &msg);

bool **isSocketConnected**(QTcpSocket \*socket);

bool **isClientConnected**(QTcpSocket \*socket);

void **sendStringToHost**(const QString &msg);

bool **checkState**(const QString &sockName, QTcpSocket \*socket);

void **checkDelList**(const int index);

};

#endif // TCPCONNECTION\_H

main.cpp

#include "mainwindow.h"

#include <QApplication>

int main(int argc, char \*argv[])

{

QApplication a(argc, argv);

MainWindow w;

w.show();

return a.exec();

}

mainwindow.cpp

#include "mainwindow.h"

#include "ui\_mainwindow.h"

MainWindow::MainWindow(QWidget \*parent)

: QMainWindow(parent)

, ui(new Ui::MainWindow)

{

readFilePaths();

server.startServer();

startSending();

ui->setupUi(this);

updateList();

}

MainWindow::~MainWindow()

{

delete ui;

}

void MainWindow::on\_bAdd\_clicked()

{

//перевірити чи не пусті поля

if(!ui->lineWay->text().isEmpty() && !ui->lineName->text().isEmpty())

{

saveFilePath(ui->lineWay->text() ,ui->lineName->text());

readFilePaths();

updateList();

ui->lineWay->clear();

ui->lineName->clear();

}

}

void MainWindow::setInfo()

{

splitList.clear();

if(server.strDate != nullptr )

{

splitList = server.strDate.split("|");

QStringList listStartPrograms;

for(int i=0;i<splitList.size();i++)

{

for(int j = 0; j < listOfNames.size(); j++)

{

if(splitList[i]==listOfNames[j])

{

listStartPrograms.push\_back(listOfWays[j]);

break;

}

}

}

startPrograms(listStartPrograms);

server.strDate = nullptr;

}

QString strToSend = listOfNames[0];

for (int i = 1; i < listOfNames.size(); i++)

{

strToSend = strToSend + "|" + listOfNames[i];

}

qDebug()<<strToSend;

server.SendToAllClients(strToSend);

}

void MainWindow::startSending()

{

QObject::connect(&timerToSent, &QTimer::timeout,this, &MainWindow::setInfo);

timerToSent.start(FIVE\_SEC);

}

void MainWindow::stopTimer()

{

timerToSent.stop();

}

//==================================================

bool MainWindow::saveFilePath(const QString& filePath, const QString& fileName) {

QFile fileWay("listWay.txt");

QFile fileNames("listName.txt");

if (!fileWay.open(QIODevice::Append | QIODevice::Text)) {

return false;

}

if (!fileNames.open(QIODevice::Append | QIODevice::Text)) {

return false;

}

QTextStream out(&fileWay);

QString newPath = filePath;

newPath.replace("\\","\\\\");

out << newPath << "\n";

fileWay.close();

QTextStream out2(&fileNames);

out2 << fileName << "\n";

fileNames.close();

return true;

}

bool MainWindow::readFilePaths() {

//QVector<QString> filePaths;

listOfWays.clear();

listOfNames.clear();

QFile fileWay("listWay.txt");

QFile fileNames("listName.txt");

if (!fileWay.open(QIODevice::ReadOnly | QIODevice::Text)) {

return false;

}

if (!fileNames.open(QIODevice::ReadOnly | QIODevice::Text)) {

return false;

}

QTextStream in(&fileWay);

while (!in.atEnd()) {

QString line = in.readLine();

listOfWays.append(line);

}

QTextStream in2(&fileNames);

while (!in2.atEnd()) {

QString line = in2.readLine();

listOfNames.append(line);

}

fileWay.close();

fileNames.close();

return true;

}

bool MainWindow::deleteFilePath(const QString& targetPath, const QString& fileName) {

QVector<QString> filePaths;

QFile file("listWay.txt");

qDebug()<<fileName;

if (!file.open(QIODevice::ReadOnly | QIODevice::Text)) {

return false;

}

QTextStream in(&file);

while (!in.atEnd()) {

QString line = in.readLine();

if (line != targetPath) {

filePaths.append(line);

}

}

file.close();

if (!file.open(QIODevice::WriteOnly | QIODevice::Text | QIODevice::Truncate)) {

return false;

}

QTextStream out(&file);

for (const QString& path : filePaths) {

out << path <<"\n";

}

file.close();

return true;

}

//===================================================

void MainWindow::updateList()

{

QStandardItemModel\* model = new QStandardItemModel();

int rows = listOfNames.size();

int columns = 2;

model->setRowCount(rows);

model->setColumnCount(columns);

model->setHorizontalHeaderItem(0,new QStandardItem("Path"));

model->setHorizontalHeaderItem(1,new QStandardItem("Name"));

for (int row = 0; row < rows; ++row) {

if (row < listOfWays.size())

{

QStandardItem\* item = new QStandardItem(listOfWays[row]);

item->setEditable(false);

model->setItem(row, 0, item);

}

else

{

QStandardItem\* item = new QStandardItem("");

item->setEditable(false);

model->setItem(row, 0, item);

}

if (row < listOfNames.size())

{

model->setItem(row, 1, new QStandardItem(listOfNames[row]));

}

else

{

QStandardItem\* item = new QStandardItem("");

item->setEditable(false);

model->setItem(row, 1, item);

}

}

ui->tList->setModel(model);

}

void MainWindow::startPrograms(const QStringList& programs)

{

stopPrograms();

for (const auto& program : programs) {

STARTUPINFOW si;

PROCESS\_INFORMATION pi;

ZeroMemory(&si, sizeof(si));

si.cb = sizeof(si);

ZeroMemory(&pi, sizeof(pi));

std::wstring wProgram = program.toStdWString();

if (!CreateProcessW(NULL,

&wProgram[0],

NULL,

NULL,

FALSE,

0,

NULL,

NULL,

&si,

&pi)

) {

qCritical() << "CreateProcessW failed for" << program << ". Error:" << GetLastError();

} else {

processes.push\_back(pi);

}

}

}

void MainWindow::stopPrograms()

{

for (auto& pi : processes) {

if (pi.hProcess != NULL) {

TerminateProcess(pi.hProcess, 0);

CloseHandle(pi.hProcess);

}

if (pi.hThread != NULL) {

CloseHandle(pi.hThread);

}

}

processes.clear();

}

TCPConnection.cpp

#include "TCPConnection.h"

cConnection::**cConnection**(){}

cConnection::~***cConnection***(){}

bool cConnection::**startServer**()

{

if(this->listen(QHostAddress::Any, PORT\_TO\_CONNECT))

{

qDebug()<<"Server started";

return true;

}

else

{

qDebug()<<"Error listening";

return false;

}

}

void cConnection::***incomingConnection***(qintptr sockDesc)

{

socket = new QTcpSocket(this);

if(socket -> *setSocketDescriptor*(sockDesc))

{

connect(socket, &QTcpSocket::readyRead, this, &cConnection::sockReady);

connect(socket, &QTcpSocket::disconnected, socket, &QTcpSocket::deleteLater);

vSockets.push\_back(socket);

sockIndexToSend.push\_back(sockDesc);

socket->*waitForReadyRead*(500);

QString sockName = strDate;

if (checkState(sockName, *socket*))

{

sockState.socketIndex.push\_back(sockDesc);

sockState.socketName.push\_back(sockName);

sockState.socketState.push\_back(true);

qDebug()<<"connected socket"<< sockState.socketIndex.last()<<" " << sockState.socketName.last()<<" "<<sockState.socketState.last();

}

}

}

bool cConnection::**conClient**()

{

try

{

socket = new QTcpSocket(this);

connect(socket, &QTcpSocket::readyRead, this, &cConnection::sockReady);

connect(socket, &QTcpSocket::disconnected, socket, &QTcpSocket::deleteLater);

qDebug()<<"connecting to host";

return true;

}

catch(const std::exception& e)

{

qDebug()<<"not connected";

return false;

}

}

QString cConnection::**sockReady**()

{

socket = (QTcpSocket\*)sender();

QDataStream in(*socket*);

in.setVersion(QDataStream::Qt\_5\_12);

if(in.status()==QDataStream::Ok)

{

strDate.clear();

in>>strDate;

qDebug()<<"msg readed"<<strDate;

return strDate;

}

qDebug()<<"msg not readed";

return 0;

}

bool cConnection::**sockDisc**()

{

try

{

socket->deleteLater();

qDebug()<<"Socket deleted";

socket = nullptr;

return true;

}

catch (const std::exception& e)

{

qDebug()<<"Socket not deleted";

return false;

}

}

bool cConnection::**conClToHost**()

{

socket->*connectToHost*(ADDR\_TO\_CONNECT, PORT\_TO\_CONNECT);

if (socket->*waitForConnected*(500))

{

qDebug()<<"connected to host";

return true;

}

else

{

qDebug()<<"not connected to host";

return false;

}

}

void cConnection::**sendStringToHost**(const QString &msg) {

if (socket != nullptr && socket->state() == QAbstractSocket::ConnectedState) {

dataBuf.clear();

QDataStream outData(*&dataBuf*, QIODevice::WriteOnly);

outData.setVersion(QDataStream::Qt\_5\_12);

outData<<msg;

socket->write(dataBuf);

if (socket->*waitForBytesWritten*(THREE\_SEC))

{

qDebug() << "Message sent to host";

} else {

qDebug() << "Failed to send" << socket->errorString();

}

} else {

qDebug() << "Socket is not connected";

}

}

bool cConnection::**SendToClient**(const QString &msg)

{

if(!vSockets.empty() && !socket->disconnect())

{

dataBuf.clear();

QDataStream outData(*&dataBuf*, QIODevice::WriteOnly);

outData.setVersion(QDataStream::Qt\_5\_12);

outData<<msg;

socket->write(dataBuf);

qDebug()<<"sended to "<<socket->*socketDescriptor*();

return true;

}

else

{

qDebug()<<"not sended to "<<socket->*socketDescriptor*();

return false;

}

}

bool cConnection::**SendToAllClients**(const QString &msg)

{

if(!vSockets.empty())

{

dataBuf.clear();

QDataStream outData(*&dataBuf*, QIODevice::WriteOnly);

outData.setVersion(QDataStream::Qt\_5\_12);

outData<<msg;

int vSocketsSize = vSockets.size();

for(int i = 0;i< vSocketsSize; i++)

{

if(!vSockets.empty())

{

if (isSocketConnected(*vSockets[i]*))

{

vSockets[i]->write(dataBuf);

qDebug()<<"sended to"<<vSockets[i];

}

else

{

vSockets.removeOne(vSockets[i]);

qDebug()<<"Removed socket"<< sockIndexToSend[i];

for (int j = 0; j < sockState.socketIndex.size();j++)

{

if(sockState.socketIndex[j] == sockIndexToSend[i])

{

sockState.socketState[j] = false;

qDebug()<<"checking for 1 min";

int q = sockState.socketIndex[j];

checkDelList(q);

}

}

sockIndexToSend.removeOne(sockIndexToSend[i]);

if (vSocketsSize > 0)

vSocketsSize--;

else

break;

}

}

else

break;

}

return true;

}

else

{

qDebug()<<"not sended, no connected clients";

return false;

}

}

void cConnection::**checkDelList**(const int index)

{

qDebug()<<"the index being checked"<<index;

QTimer::singleShot(ONE\_MIN, this, [this, index]() {

for (int ind=0; ind < sockState.socketIndex.size(); ind++)

{

if(sockState.socketIndex[ind] == index)

{

qDebug()<<"Go to remove socketIndex "<<sockState.socketIndex[ind];

qDebug()<<"Go to remove sockIndexToSend "<<index;

if(sockState.socketState[ind] == false)

{

sockState.socketIndex.remove(ind);

sockState.socketName.remove(ind);

sockState.socketState.remove(ind);

qDebug()<<index<<" deleted";

break;

}

else

qDebug()<<index<<"not deleted";

}

}

});

}

bool cConnection::**isSocketConnected**(QTcpSocket \*socket)

{

if (socket != nullptr)

{

return socket->state() == QAbstractSocket::ConnectedState;

}

return false;

}

bool cConnection::**isClientConnected**(QTcpSocket \*socket)

{

if (socket != nullptr) {

return socket->isOpen() && socket->state() == QAbstractSocket::ConnectedState;

}

return false;

}

bool cConnection::**checkState**(const QString &sockName, QTcpSocket \*socket)

{

for(int i=0; i<this->sockState.socketName.size();i++)

{

qDebug()<<this->sockState.socketName[i];

if (this->sockState.socketName[i] == sockName && this->sockState.socketState[i] == 1)

{

qDebug()<<"Socket is not connected name: "<<sockName;

sockDisc();

qDebug()<<"Removed in check vSockets" << vSockets.last();

socket->disconnect();

vSockets.removeLast();

sockIndexToSend.removeLast();

return false;

}

else if(this->sockState.socketName[i] == sockName && this->sockState.socketState[i] == 0)

{

qDebug()<<"Socket is reconnected"<<socket->*socketDescriptor*();

this->sockState.socketState[i] = true;

this->sockState.socketIndex[i] = socket->*socketDescriptor*();

return false;

}

}

qDebug()<<"connection is ok";

return true;

}

# Додаток Б – Вихідний код програми клієнта

mainwindow.h

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QInputDialog>

#include <QMessageBox>

#include <QTimer>

#include <QStandardItemModel>

#include <QTextStream>

#include <QString>

#include <QVector>

#include <QWidget>

#include <QVBoxLayout>

#include <QComboBox>

#include "TCPConnection.h"

QT\_BEGIN\_NAMESPACE

namespace **Ui** { class **MainWindow**; }

QT\_END\_NAMESPACE

class **MainWindow** : public QMainWindow

{

Q\_OBJECT

public:

**MainWindow**(QWidget \*parent = nullptr);

~***MainWindow***();

cConnection clientSock;

QTimer timer;

QStringList splitList;

QStringList listToStart;

private slots:

void **startTimer**();

void **stopTimer**();

bool **setInfo**();

void **on\_bConnect\_clicked**();

void **on\_bStart\_clicked**();

void **updateList**();

void **onComboBoxShowPopup**();

void **on\_bUpdBox\_clicked**();

void **on\_bAddAppToList\_clicked**();

void **on\_bClear\_clicked**();

private:

Ui::MainWindow \*ui;

};

#endif // MAINWINDOW\_H

TCPConnection.h

#ifndef TCPCONNECTION\_H

#define TCPCONNECTION\_H

#include <QTcpServer>

#include <QTcpSocket>

#include <QVector>

#include <QDataStream>

#include <QTimer>

#define PORT\_TO\_CONNECT 5555

#define ADDR\_TO\_CONNECT "127.0.0.1"

#define THREE\_SEC 3000

#define FIVE\_SEC 5000

#define ONE\_MIN 60000

class **cConnection**: public QTcpServer

{

Q\_OBJECT

private:

QVector <QTcpSocket\*> vSockets;

QVector <QString> vSocketsName;

QByteArray dataBuf;

QVector <qintptr> sockIndexToSend;

public:

**cConnection**();

~***cConnection***();

QTcpSocket\* socket = nullptr;

QString strDate;

struct **socketsState**

{

QVector <qintptr> socketIndex;

QVector <QString> socketName;

QVector <bool> socketState;

}sockState;

bool **startServer**();

void ***incomingConnection***(qintptr sockDesc);

bool **conClient**();

QString **sockReady**();

bool **sockDisc**();

bool **conClToHost**();

bool **SendToClient**(const QString &msg);

bool **SendToAllClients**(const QString &msg);

bool **isSocketConnected**(QTcpSocket \*socket);

bool **isClientConnected**(QTcpSocket \*socket);

void **sendStringToHost**(const QString &msg);

bool **checkState**(const QString &sockName, QTcpSocket \*socket);

void **checkDelList**(const int index);

};

#endif // TCPCONNECTION\_H

main.cpp

#include "mainwindow.h"

#include <QApplication>

int main(int argc, char \*argv[])

{

QApplication a(*argc*, *argv*);

MainWindow w;

w.show();

return a.exec();

}

mainwindow.cpp

#include "mainwindow.h"

#include "TCPConnection.h"

#include "ui\_mainwindow.h"

MainWindow::**MainWindow**(QWidget \*parent)

: QMainWindow(*parent*)

, ui(new Ui::MainWindow)

{

ui->setupUi(this);

}

MainWindow::~***MainWindow***()

{

delete ui;

}

void MainWindow::**onComboBoxShowPopup**()

{

ui->comboBox->addItems(splitList);

}

void MainWindow::**on\_bConnect\_clicked**()

{

if (!clientSock.isSocketConnected(*clientSock.socket*))

{

clientSock.conClient();

if (clientSock.conClToHost() == true)

{

QMessageBox \*msgBox = new QMessageBox();

msgBox->setText("wait a couple of seconds, if not connected, try to again");

msgBox->show();

//clientSock.sendStringToHost(clientName);

if(!clientSock.socket->*waitForDisconnected*(5000))

{

ui->lConnectionState->setStyleSheet("QLabel { background-color : green; }");

startTimer();

}

else clientSock.sockDisc();

}

}

}

void MainWindow::**on\_bStart\_clicked**()

{

QString strToSend;

if (!listToStart.empty())

{

strToSend = listToStart[0];

for (int i=1; i < listToStart.size(); i++)

strToSend = strToSend + "|" + listToStart[i];

clientSock.sendStringToHost(strToSend);

listToStart.clear();

updateList();

}

}

void MainWindow::**startTimer**()

{

QObject::connect(&timer, &QTimer::timeout,this, &MainWindow::setInfo);

timer.start(FIVE\_SEC);

}

void MainWindow::**stopTimer**()

{

timer.stop();

}

bool MainWindow::**setInfo**()

{

if(clientSock.strDate != nullptr )

{

if(clientSock.isClientConnected(*clientSock.socket*))

{

//ui->comboBox->clear();

splitList.clear();

splitList = clientSock.strDate.split("|");

//updateList();

return true;

}

else

{

stopTimer();

ui->lConnectionState->setStyleSheet("QLabel { background-color : red; }");

return false;

}

}

else

{

qDebug()<<"Empty strDate";

return false;

}

return false;

}

void MainWindow::**updateList**()

{

QStandardItemModel\* model = new QStandardItemModel();

int rows = listToStart.size();

int columns = 1;

model->setRowCount(rows);

model->setColumnCount(columns);

model->setHorizontalHeaderItem(0,new QStandardItem*(*"Name"*)*);

for (int row = 0; row < rows; ++row) {

if (row < listToStart.size())

{

QStandardItem\* item = new QStandardItem(listToStart[row]);

item->setEditable(false);

model->setItem(row, 0, *item*);

}

else

{

QStandardItem\* item = new QStandardItem("");

item->setEditable(false);

model->setItem(row, 0, *item*);

}

}

ui->tList->*setModel*(*model*);

}

void MainWindow::**on\_bUpdBox\_clicked**()

{

ui->comboBox->clear();

ui->comboBox->addItems(splitList);

}

void MainWindow::**on\_bAddAppToList\_clicked**()

{

bool p = false;

for(int i = 0; i<listToStart.size();i++)

if (listToStart[i]==ui->comboBox->currentText())

p = true;

if (p == false)

{

listToStart.push\_back(ui->comboBox->currentText());

updateList();

}

}

void MainWindow::**on\_bClear\_clicked**()

{

listToStart.clear();

updateList();

}

TCPConnection.cpp

#include "TCPConnection.h"

cConnection::**cConnection**(){}

cConnection::~***cConnection***(){}

bool cConnection::**startServer**()

{

if(this->listen(QHostAddress::Any, PORT\_TO\_CONNECT))

{

qDebug()<<"Server started";

return true;

}

else

{

qDebug()<<"Error listening";

return false;

}

}

void cConnection::***incomingConnection***(qintptr sockDesc)

{

socket = new QTcpSocket(this);

if(socket -> *setSocketDescriptor*(sockDesc))

{

connect(socket, &QTcpSocket::readyRead, this, &cConnection::sockReady);

connect(socket, &QTcpSocket::disconnected, socket, &QTcpSocket::deleteLater);

vSockets.push\_back(socket);

sockIndexToSend.push\_back(sockDesc);

socket->*waitForReadyRead*(500);

QString sockName = strDate;

if (checkState(sockName, *socket*))

{

sockState.socketIndex.push\_back(sockDesc);

sockState.socketName.push\_back(sockName);

sockState.socketState.push\_back(true);

qDebug()<<"connected socket"<< sockState.socketIndex.last()<<" " << sockState.socketName.last()<<" "<<sockState.socketState.last();

}

}

}

bool cConnection::**conClient**()

{

try

{

socket = new QTcpSocket(this);

connect(socket, &QTcpSocket::readyRead, this, &cConnection::sockReady);

connect(socket, &QTcpSocket::disconnected, socket, &QTcpSocket::deleteLater);

qDebug()<<"connecting to host";

return true;

}

catch(const std::exception& e)

{

qDebug()<<"not connected";

return false;

}

}

QString cConnection::**sockReady**()

{

socket = (QTcpSocket\*)sender();

QDataStream in(*socket*);

in.setVersion(QDataStream::Qt\_5\_12);

if(in.status()==QDataStream::Ok)

{

strDate.clear();

in>>strDate;

qDebug()<<"msg readed"<<strDate;

return strDate;

}

qDebug()<<"msg not readed";

return 0;

}

bool cConnection::**sockDisc**()

{

try

{

socket->deleteLater();

qDebug()<<"Socket deleted";

socket = nullptr;

return true;

}

catch (const std::exception& e)

{

qDebug()<<"Socket not deleted";

return false;

}

}

bool cConnection::**conClToHost**()

{

socket->*connectToHost*(ADDR\_TO\_CONNECT, PORT\_TO\_CONNECT);

if (socket->*waitForConnected*(500))

{

qDebug()<<"connected to host";

return true;

}

else

{

qDebug()<<"not connected to host";

return false;

}

}

void cConnection::**sendStringToHost**(const QString &msg) {

if (socket != nullptr && socket->state() == QAbstractSocket::ConnectedState) {

dataBuf.clear();

QDataStream outData(*&dataBuf*, QIODevice::WriteOnly);

outData.setVersion(QDataStream::Qt\_5\_12);

outData<<msg;

socket->write(dataBuf);

if (socket->*waitForBytesWritten*(THREE\_SEC))

{

qDebug() << "Message sent to host";

} else {

qDebug() << "Failed to send" << socket->errorString();

}

} else {

qDebug() << "Socket is not connected";

}

}

bool cConnection::**SendToClient**(const QString &msg)

{

if(!vSockets.empty() && !socket->disconnect())

{

dataBuf.clear();

QDataStream outData(*&dataBuf*, QIODevice::WriteOnly);

outData.setVersion(QDataStream::Qt\_5\_12);

outData<<msg;

socket->write(dataBuf);

qDebug()<<"sended to "<<socket->*socketDescriptor*();

return true;

}

else

{

qDebug()<<"not sended to "<<socket->*socketDescriptor*();

return false;

}

}

bool cConnection::**SendToAllClients**(const QString &msg)

{

if(!vSockets.empty())

{

dataBuf.clear();

QDataStream outData(*&dataBuf*, QIODevice::WriteOnly);

outData.setVersion(QDataStream::Qt\_5\_12);

outData<<msg;

int vSocketsSize = vSockets.size();

for(int i = 0;i< vSocketsSize; i++)

{

if(!vSockets.empty())

{

if (isSocketConnected(*vSockets[i]*))

{

vSockets[i]->write(dataBuf);

qDebug()<<"sended to"<<vSockets[i];

}

else

{

vSockets.removeOne(vSockets[i]);

qDebug()<<"Removed socket"<< sockIndexToSend[i];

for (int j = 0; j < sockState.socketIndex.size();j++)

{

if(sockState.socketIndex[j] == sockIndexToSend[i])

{

sockState.socketState[j] = false;

qDebug()<<"checking for 1 min";

int q = sockState.socketIndex[j];

checkDelList(q);

}

}

sockIndexToSend.removeOne(sockIndexToSend[i]);

if (vSocketsSize > 0)

vSocketsSize--;

else

break;

}

}

else

break;

}

return true;

}

else

{

qDebug()<<"not sended, no connected clients";

return false;

}

}

void cConnection::**checkDelList**(const int index)

{

qDebug()<<"the index being checked"<<index;

QTimer::singleShot(ONE\_MIN, this, [this, index]() {

for (int ind=0; ind < sockState.socketIndex.size(); ind++)

{

if(sockState.socketIndex[ind] == index)

{

qDebug()<<"Go to remove socketIndex "<<sockState.socketIndex[ind];

qDebug()<<"Go to remove sockIndexToSend "<<index;

if(sockState.socketState[ind] == false)

{

sockState.socketIndex.remove(ind);

sockState.socketName.remove(ind);

sockState.socketState.remove(ind);

qDebug()<<index<<" deleted";

break;

}

else

qDebug()<<index<<"not deleted";

}

}

});

}

bool cConnection::**isSocketConnected**(QTcpSocket \*socket)

{

if (socket != nullptr)

{

return socket->state() == QAbstractSocket::ConnectedState;

}

return false;

}

bool cConnection::**isClientConnected**(QTcpSocket \*socket)

{

if (socket != nullptr) {

return socket->isOpen() && socket->state() == QAbstractSocket::ConnectedState;

}

return false;

}

bool cConnection::**checkState**(const QString &sockName, QTcpSocket \*socket)

{

for(int i=0; i<this->sockState.socketName.size();i++)

{

qDebug()<<this->sockState.socketName[i];

if (this->sockState.socketName[i] == sockName && this->sockState.socketState[i] == 1)

{

qDebug()<<"Socket is not connected name: "<<sockName;

sockDisc();

qDebug()<<"Removed in check vSockets" << vSockets.last();

socket->disconnect();

vSockets.removeLast();

sockIndexToSend.removeLast();

return false;

}

else if(this->sockState.socketName[i] == sockName && this->sockState.socketState[i] == 0)

{

qDebug()<<"Socket is reconnected"<<socket->*socketDescriptor*();

this->sockState.socketState[i] = true;

this->sockState.socketIndex[i] = socket->*socketDescriptor*();

return false;

}

}

qDebug()<<"connection is ok";

return true;

}