Міністерство освіти і науки України

ДВНЗ «Донецький національний технічний університет»

Факультет комп’ютерних наук і технологій

Кафедра комп’ютерної інженерії

**Пояснювальна записка**

до курсової роботи

з дисципліни

«Програмне забезпечення мереж»

Виконав

студент групи КН-16

спеціальності 122 Комп’ютерні науки

Рашевский В.В.

Оцінка

Кількість балів: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Керівник:

доц. каф. КІ Цололо С.О.

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

Покровськ, 2020

ЗМІСТ

ВСТУП 4

1. СТРУКТУРА СИСТЕМИ 5
2. СТРУКТУРА ДАНИХ 7
3. ПРОЕКТУВАННЯ КОМПОНЕНТІВ СИСТЕМИ 11
   1. Функції та алгоритм роботи КОМ 11
   2. Функції та алгоритм роботи серверу 13
   3. Функції та алгоритм роботи PC 16
4. РЕАЛІЗАЦІЯ КОМПОНЕНТІВ СИСТЕМИ 18
   1. Модель введення-виведення 18
   2. Програмне забезпечення КОМ 19
   3. Програмне забезпечення серверу 24
   4. Програмне забезпечення PC 31

ВИСНОВКИ 35

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 36

Додаток А – Лістинг ПЗ серверу 37

Додаток Б – Лістинг ПЗ керуючих обчислювальних машин 49

Додаток В – Лістинг ПЗ робочих станцій 54

ВСТУП

Мета курсової роботи полягає в практичному опанування засобів реалізації мережної взаємодії компонентів систем, що побудовані за архітектурою «клієнтсервер». Ця архітектура, зокрема, широко використовується при побудові автоматизованих систем керування технологічними процесами (АСК ТП). У системах АСК ТП можна виділити три основні компоненти: − керуючі обчислювальні машини (КОМ), призначені для безпосереднього керування технологічним процесом; − сервер бази даних, що здійснює зберігання інформації про хід ТП; − робочі станції, що надають користувачам доступ до цієї інформації, і дозволяють контролювати хід технологічного процесу. У якості транспортного протоколу в роботі використовується протокол TCP/IP[1], для якого інтерфейс Windows Sockets надає зручні кошти організації передачі даних з використанням датаграм і каналів зв'язку між вузлами мережі. У локальних і глобальних мережах існує два принципово різні способи передачі даних. Перший з них припускає посилку пакетів даних від одного вузла іншому (або відразу декільком вузлам) без одержання підтвердження про доставку. Другий спосіб передачі даних передбачає створення каналу передачі даних між двома різними вузлами мережі. При цьому канал створюється засобами датаграмних протоколів, однак доставка пакетів у каналі є гарантованою. Пакети завжди доходять у цілісності, хоча задля цього і можуть бути використані декілька спроб. Пакети надходити до приймача в будь-якому порядку, але після отримання вони організуються в правильному порядку, відповідно му до порядку відправлення. Мінусом є те, що швидкодія при цьому виходить у середньому нижче за рахунок посилки підтверджень. Отже, завдання курсової роботи передбачає розробку програмного забезпечення для реалізації мережної взаємодії всіх компонентів АСК ТП на основі бібліотеки Windows Sockets.

1 СТРУКТУРА СИСТЕМИ

Система складається із сервера, декількох керуючих обчислювальних машин (КОМ)[2] і робочих станцій.

Керуючі обчислювальні машини збирають первинну інформацію про хід ТП, виконують попередню обробку цієї інформації й періодично передають результати обробки серверу. Для забезпечення надійності функції КОМ дублюються. Кожна група складається із двох машин і займається збором певних параметрів ТП. Усі КОМ, що належать до однієї групи, працюють незалежно і передають серверу ту саму інформацію.

Сервер ухвалює інформацію від КОМ і зберігає її в базі даних (БД), крім того, він обслуговує запити від робочих станцій і відповідає за синхронізацію часу в системі.

Робочі станції використовуються для візуального контролю ТП і аналізу інформації. Вони звертаються до сервера і отримують необхідні дані від нього. Для обміну даними все компоненти системи використовують той самий транспортний протокол. Зв'язок між контролерами і сервером встановлюється з ініціативи контролерів, між робочими станціями й сервером – з ініціативи робочих станцій. У якості операційної системи застосовується Microsoft Windows.

Отже, в курсовій роботі необхідно розробити наступне програмне забезпечення:

1. ПЗ сервера, що складається із трьох незалежних компонентів:

− перший займається прийманням інформації від контролерів і збереженням її в БД;

− другий обслуговує запити від робочих станцій, передаючи їм необхідні дані із БД;

− третій відповідає за синхронізацію часу.

2. ПЗ КОМ, що формує дані про хід технологічного процесу і періодично передає їх серверу.

3. ПЗ робочих станцій, що посилає запити серверу і одержує від нього інформацію для наступного відображення у виді, зручному для користувача. Для синхронізації часу між усіма компонентами системи, КОМ і робочі станції періодично посилають серверу запити про точний час. Сервер відповідає на ці запити, повідомляючи поточний час по своєму годиннику.

2 СТРУКТУРА ДАНИХ

Для передачі корисної та службової інформаціїміж компонентами системи були розроблені структури:

struct Message{

enum Type : unsigned short {

CONTROLLER\_HELLO,

CONTROLLER\_DATA,

CONTROLLER\_TIMEDIFF,

WORKSTATION\_HELLO,

WORKSTATION\_REQUEST,

WORKSTATION\_ANSWER

};

unsigned short type;

unsigned long time;

};

struct ControllerInfoMessage : Message{

unsigned char controllerNumber;

};

struct ControllerDataMessage : ControllerInfoMessage{

unsigned char speed1, speed2;

unsigned short temp1, temp2;

unsigned short mass;

unsigned char length;

};

struct TimeDiffMessage : Message{

short timediff;

};

struct WorkstationRequest : Message{

unsigned long from;

unsigned long to;

};

struct WorkstationAnswer : Message{

ControllerDataMessage data;

ControllerInfo::Type dataType;

bool finish;

};

Основною структурою є Message, у якій заповнюються загальні,службові дані для всіх повідомлень. Інші структури ієрархічно успадковуються від неї, та при необхідності додають необхідні для себе інформаційні поля.

У структурі КОМ заповнюються лише дані, які вони збирають та передають.

Призначення полів структур наведені в таблиці 2.1:

Таблиця 2.1 – Призначення полів структур повідомлень

|  |  |  |
| --- | --- | --- |
| **Структура** | **Поле** | **Пояснення** |
| Message | type | Тип (або код) повідомлення. Всі доступні типи вказані у перерахуванні Type:  1) CONTROLLER\_TIMEDIFF – різниця у часі між сервером та КОМ, що сервер відправляє то КОМ для синхронізації часу;  2) CONTROLLER\_DATA – дані з КОМ;  3) WORKSTATION\_HELLO – початкове повідомлення РС на сервер для її ідентифікації;  4) WORKSTATION\_ANSWER – відповідь серверу на запит РС;  5) WORKSTATION\_REQUEST – запит РС на отримання даних з серверу;  6) CONTROLLER\_HELLO – початкове повідомлення КОМ для її ідентифікації. |
| time | Час відправлення повідомлення. |
| ControllerInfoMessage | controllerNumber | Номер КОМ, що відсилає повідомлення. |
| ControllerDataMessage | speed1 | Швидкість 1, м/с |
| speed2 | Швидкість 2, м/с |
| temp1 | Температура 1, °С |
| temp2 | Температура 2, °С |
| mass1 | Вага 1, кг |
| mass2 | Вага 2, кг |
| length | Довжина, м |
| TimeDiffMessage | timediff | Різниця у часі між сервером та КОМ, що відсилається сервером. |
| WorkstationRequest | from | Початок періоду для отримання даних з РС. |
| to | Кінець періоду для отримання даних з РС |
| WorkstationAnswer | data | Структура ControllerDataMessage, що в даному випадку представляє собою один запис з даними БД. |
| dataType | Тип групи КОМ (перша чи друга). |
| finish | Флаг, чи були відправлені всі записи з БД. |

Для зберігання даних використано СУБД SQLite.

SQLite[4] підтримує динамічне тіпізірованія даних. Можливі типи значень: INTEGER, REAL, TEXT і BLOB. Так само підтримується спеціальне значення NULL.

Розміри значень типу TEXT і BLOB не обмежені нічим, крім константи SQLITE\_MAX\_LENGTH в исходниках sqlite, рівній мільярду.

Кожне значення в будь-якому полі будь-якого запису може бути будь-якого з цих типів, незалежно від типу, зазначеного при оголошенні полів таблиці. Зазначений при оголошенні поля тип зберігається для довідки в його вихідному написанні, і використовується в якості основи для вибору переваг (так зване «type affinity»: це підхід, що рідко зустрічається в інших СУБД) при виконанні неявних перетворень типів на підставі схожості цієї назви типу на що-небудь, знайоме SQLite. В цей алгоритм зашитий великий перелік практикуються в інших СУБД варіантів назв типів даних. Якщо безпечного перетворення записується значення в бажаний тип не виходить, SQLite записує значення в його початковому вигляді. Для отримання значень з бази є ряд функцій для кожного з типів, і якщо тип значення, що зберігається не відповідає запитуваній, воно теж, по можливості, перетворюється.

У базі даних було створено одну таблицю data зі структурою:

1. type (tinyint) – тип групи КОМ;
2. time (unsigned big int) – час отримання значень з датчиків на КОМ;
3. speed1 (integer) – швидкість 1, м/с;
4. speed2 (integer) – швидкість 2, м/с;
5. temp1 (integer) – температура 1, °С;
6. temp2 (integer) – температура 2, °С;
7. mass1 (integer) – вага 1, кг;
8. mass2 (integer) – вага 2, кг;
9. length (integer) – довжина, м.

Для емуляції надходження даних в програмному забезпеченні КОМз датчиківвикористовується випадкова генерація значень, однакової послідовності у межах однієї групи КОМ, але різної для різних груп.

3 ПРОЕКТУВАННЯ КОМПОНЕНТІВ СИСТЕМИ

3.1 Функції та алгоритм роботи КОМ

Програмне забезпечення КОМ виконує наступні функції:

1. З'єднання із сервером, що задається IP-адресою і портом, за допомогою протоколу TCP/IP.

2. Синхронізація КОМ і сервера, шляхом посилки останньому поточного значення часу, установленого на КОМ.

3. Зчитування даних з датчиків контролю ТП, у якості емуляторів датчиків можна використовувати файли даних, з яких надходить інформація типу 1-2 і 3-4, або використати випадкову генерацію.

4. Передача даних заданого типу на сервер із встановленим інтервалом передачі.

5. Коректне закриття з'єднання із сервером у випадку одержання від нього інформації про зупинку сервера.

6. Можливість настроювання параметрів підключення до сервера.

7. Відображення процесу передачі інформації на сервер.

При запуску програма ініціалізує бібліотеку Windows Sockets. Після введення IP-адресу та порту сервера з’являється можливість підключення. Створюється сокет, що прив’язується до цього адресу і відбувається спроба підключення. Якщо вона виявляється вдалою, програма оновлює статус про успішне підключення та дає можливість налаштувати інтервал передачі у секундах та почати передачу даних.

Відразу після підключення, КОМ відправляє на сервер повідомлення про її номер, щоб сервер знав, що по відповідному сокету підключилась саме ця КОМ. Якщо час на сервері відрізняється, то з серверу прийде повідомлення з різницею в часі для синхронізації.

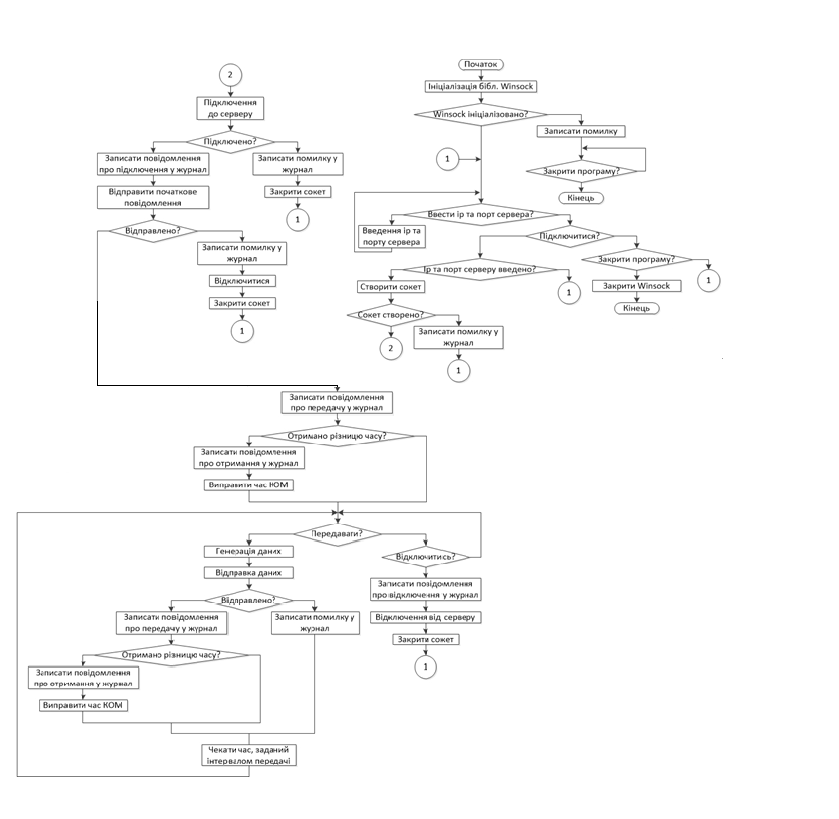


Рис. 3.1 – Граф-схема алгоритму роботи КОМ

Коли користувач почне передавати дані, запускається таймер с заданим інтервалом. У момент передачі генеруються дані для емуляції їх збору с датчиків, і відправляються на сервер. Якщо з серверу приходить повідомлення з різницею у часі, КОМ знову уточнює свій час.

Після завершення передачі є можливість почати знову або зовсім відключитися від серверу, після чого змінити адресу та/або порт сервера і знову підключитися и передавати дані, або закрити програму. При відключенні від серверу відповідний сокет закривається.При завершенні всієї програми бібліотека Windows Sockets також буде закрита.

Під час роботи програми у системний журнал, журнал прийому та передачі записуються всі повідомлення про відповідні події. У системному журналі це такі записи, як повідомлення про підключення, відключення та помилки, що сталися під час роботи якоїсь функції.

3.2 Функції та алгоритм роботи серверу

Програмне забезпечення серверу виконує наступні функції:

1. Запуск і зупинка процесу прослуховування заданого порту на предмет підключення КОМ чи РС.

2. Підтвердження підключення клієнтів, запис інформації про різницю в часі між сервером і клієнтом (для КОМ).

3. Прийом інформації від датчиків КОМ, запис найбільш свіжої інформації типу 1-2 і 3-4, запис дати останнього прийому інформації кожного типу.

4. Запис даних з буфера прийому до бази або файлів даних (розмір буфера прийому задається користувачем).

5. Прийом запитів від РС, запуск в окремому потоці процесу передачі інформації до РС, посилка ознаки завершення передачі.

6. Розсилання всім підключеним клієнтам (КОМ і РС) ознаки завершення у випадку зупинки сервера.

7. Ведення журналів усіх системних подій, а також журналів прийому і передачі даних.

8. Можливість настроювання буфера прийому, що дозволяє регулювати навантаження на сервер.

9. Відображення інформації про всіх підключених клієнтів, кількість переданих їм і отриманих від них пакетів.

10. Можливість настроювання порту прослуховування, виведення інформації про поточне значенні IP-адреси сервера.

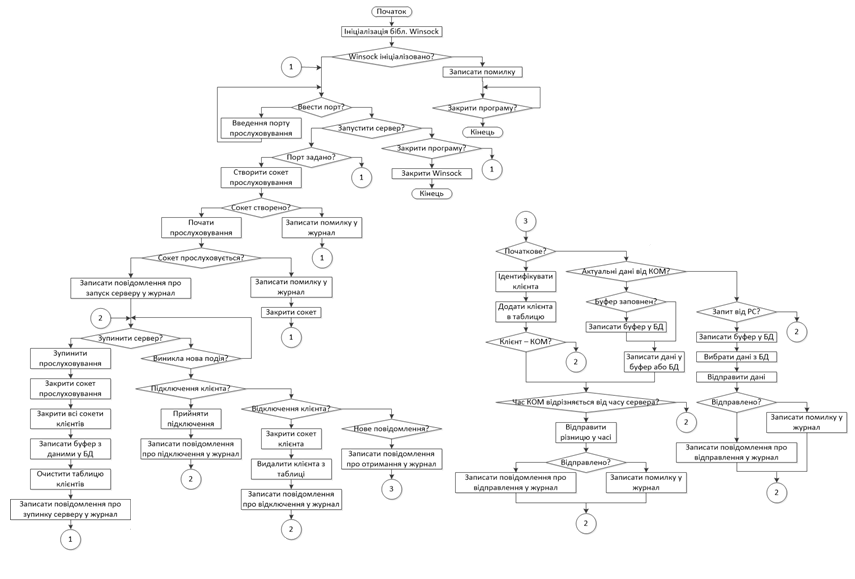


Рис. 3.2 – Граф-схема алгоритму роботи серверу

Виконання програми-серверу починається з ініціалізації бібліотеки Windows Socket[5]. Як тільки від користувача надходить команда «Запустити сервер», програма створює сокет і починає прослуховування порту, заданого користувачем. Номер порту відомий клієнтам, вони повинні використовувати його для передачі інформації. При надходженні мережного повідомлення програма контролює наявність помилок. У разі виникнення помилки, вона обробляється і користувачу видається відповідне повідомлення. Якщо ж мережне повідомлення не викликає помилки, то воно обробляється та виконуються дії відповідно до типу повідомлення, що надійшло. Якщо це FD\_ACCEPT, то виконується підтвердження підключення клієнта (КОМ чи РС), до списку підключених клієнтів додається новий запис з IP-адресою, сокетом і іншою інформацією про нового клієнта. Потім сокет, через який відбулося підтвердження підключення клієнта, налаштовується на прийом мережних повідомлень типу FD\_READ і FD\_CLOSE. Якщо отримано повідомлення типу FD\_CLOSE, то з масиву-опису підключених клієнтів видаляється вся інформація про клієнта, що надіслав це повідомлення. Якщо є дані для запису на диск, вони записуються. Сокет клієнта закривається. При надходженні повідомлення FD\_READ (і пакета даних від клієнта відповідно) насамперед аналізується поле what цього пакета і за його значенням визначаються наступні дії:

− якщо поле дорівнює «1», що означає ініціалізацію підключеного клієнта, то в цьому випадку за полем who визначається тип підключеного клієнта – для КОМ (who=1,2,3,4) виконується визначення різниці в часі між КОМ і сервером, а для РС (who=5) відбувається запуск нового потоку, у якому до РС будуть передаватися дані; − якщо поле what дорівнює 2, що означає прийом даних від КОМ, то відбувається аналіз отриманих даних на актуальність за часом. Розглянемо це питання докладніше. Справа в тім, що при наявності декількох КОМ одного типу, їхній поточний час може трохи відрізнятися один від одного. Тому час відправлення пакету, що вказується в поле time самого пакету, в момент надходження на сервер може бути недостовірним. Тобто, більш пізніше значення часу може не відповідати більш пізньому моменту часу. Тому як тільки на сервер надходить пакет даних, сервер виконує прохід за масивом інформації про підключених клієнтів. Для елементів масиву заданого типу (1-2 чи 3-4) виконується підсумовування різниці в часі «КОМ-сервер» і часу відправлення пакета з КОМ. Якщо хоча б одна сума більше, ніж останній час запису інформації (для двох типів даних це різні значення), то прийнятий пакет є актуальним і його вміст записується в буфер даних. Значення останнього часу запису оновлюється. Після цього виконується аналіз буфера прийому, якщо він заповнений, то інформація записується на диск, а буфер звільняється. Обробку FD\_READ завершено. Після аналізу повідомлень виконується перевірка на команду «Зупинити сервер». Якщо команди немає, то виконується подальше прослуховування порту, як це було описано вище. Якщо ж така команда надійшла, то всім підключеним клієнтам надсилається пакет із ознакою зупинки сервера (who=6, what=4). Це необхідно для того, щоб КОМ і РС могли коректно завершити передачу чи прийом інформації і видати відповідні повідомлення. Після цього виконується закриття усіх відкритих сокетів і видалення масиву інформації про підключених клієнтів.

3.3 Функції та алгоритм роботи РС

Перелік функцій РС:

1. З'єднання із сервером, що задається IP-адресою і портом, за допомогою протоколу TCP/IP.

2. Уведення користувачем параметрів періоду часу інформації, що запитується.

3. Посилка пакета з вказівкою періоду часу інформації, що запитується.

4. Одержання інформації від сервера і вивід її у виді таблиці (окремо для типів інформації 1-2 і 3-4).

5. Одержання ознаки завершення передачі інформації, виведення інформації про кількість отриманих структур.

6. Коректне закриття з'єднання із сервером у випадку прийому пакета з ознакою зупинки/закриття сервера.

7. Можливість налаштування параметрів підключення до сервера. Процес налаштування та запуску робочої станції, її зупинки та запису повідомлень до журналу у програмному забезпеченні РС аналогічний КОМ.

Також як і в КОМ, при підключенні РС відправляє початкове повідомлення на сервер.

Далі, можна встановити новий період даних для отримання та запустити процес передачі. Таблиця з поточними даними очищається, встановлюється статус отримання та відправляється запит на сервер.

При отриманні відповіді дані додаються у таблицю відповідного типу. Якщо ж у відповіді зазначена ознака кінця передачі, процес отримання даних завершується. Стає доступною функція змінити період отримання даних та почати процес знову або вийти з програми.

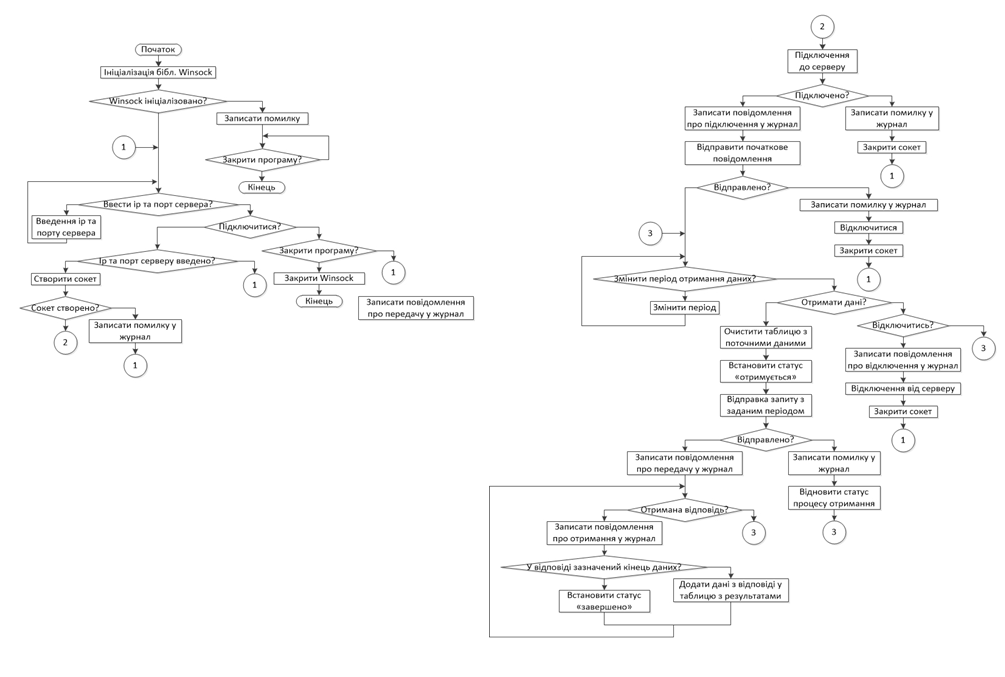


Рис. 3.3 – Граф-схема алгоритму роботи РС

4 РЕАЛІЗАЦІЯ КОМПОНЕНТІВ СИСТЕМИ

4.1 Модель введення-виведення

Згідно до завдання у роботі необхідно використовувати модель введення-виведення WSAEventSelect.

При роботі з сокетом використовується стандартний набір подій: для відправки даних (константа події FD\_WRITE), отримання даних (FD\_READ), з'єднання з іншим сокетом (FD\_CONNECT), для встановлення каналу передачі даних при вхідному запиті (FD\_ACCEPT) і для закриття сокета (FD\_CLOSE).

Далі створюється подія, яку необхідно відстежувати на даному сокеті. Для цього використовується виклик функції WSACreateEvent(). Після того, як подію створено, його потрібно зв'язати з сонетом. Це робиться функцією WSAEventSelect(). Одним із параметрів цієї функції є константа події. Необхідно зауважити, що для одного і того ж сокета неможливо створити більше одного об'єкту події. Для цього можна використовувати операцію «побітового якщо» на константах для того, щоб контролювати і оброблювати декілька подій.

Коли події задані, необхідно очікувати їх і, відповідно, обробляти. Для очікування подій можна використовувати функцію WSAWaitForMultipleEvents(). Перший параметр – це кількість подій, які потрібно очікувати. Другий параметр – це покажчик на масив подій. Третій параметр має значення BOOL, який визначає чи буде функція залишатися в сплячому режимі до тих пір поки не спрацюють всі події. Четвертий параметр визначає як довго чекати настання. П'ятий параметр вказує на те, чи потрібно отримуватиоповіщення з застереженнями.

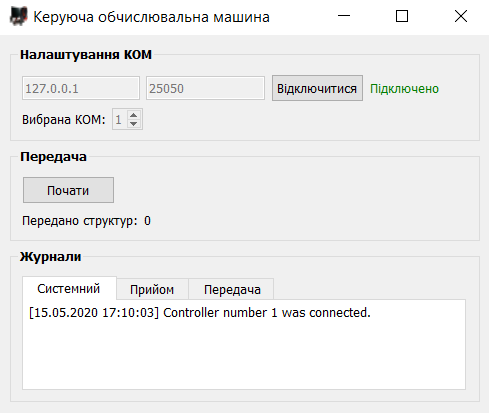
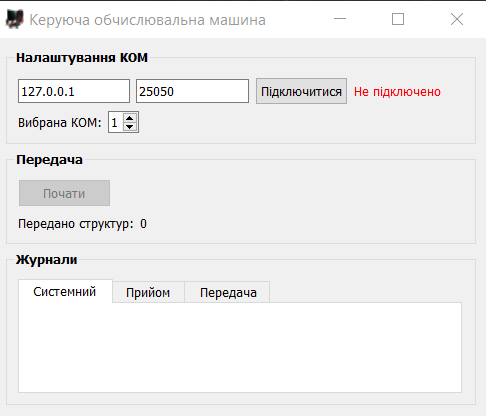
Далі необхідно отримати достовірну інформацію про те, яка подія виникла. Для цього існує функція WSAEnumNetworkEvents().У результаті отримується подія, яка виникає на одному з сокетів.

Для реалізації курсової роботи була обрана мова С++ та середовище Qt, тому що вони дають всі необхідні можливості для створення швидких та продуктивних програм з графічним інтерфейсом, що можуть використовувати бібліотеку WindowsSockets.

4.2 Програмне забезпечення КОМ

Екранні форми програмного забезпечення КОМ в різних режимах роботи наведені на рисунках нижче.

Вікно ПЗ КОМ ділиться на три частини: налаштування, передача та журнали. У розділі налаштувань знаходиться два поля для вводу IP-адреси серверу та його порту відповідно. Поряд з цими полями розміщено кнопку для підключення та текстове відображення стану підключення. Якщо КОМ підключено до серверу, на місті кнопки для підключення буде знаходитись кнопка для відключення. Під полями для вводу адреси серверу також знаходиться поле для вибору номеру КОМ – від 1 до 4 включно.



а) б)

Рис. 4.1 – Екранні форми ПЗ КОМ: а) до з’єднання з сервером; б) після встановлення з’єднання з сервером

Елементи керування у розділі для передачі становляться доступними тільки у стані підключеного КОМ до серверу. Тут знаходяться кнопка для початку передачі та кількість переданих структур даних вибраного КОМ. У стані передачі кнопка початку замінюється кнопкою зупинки та поряд з нею виводиться повідомлення про стан активної передачі.

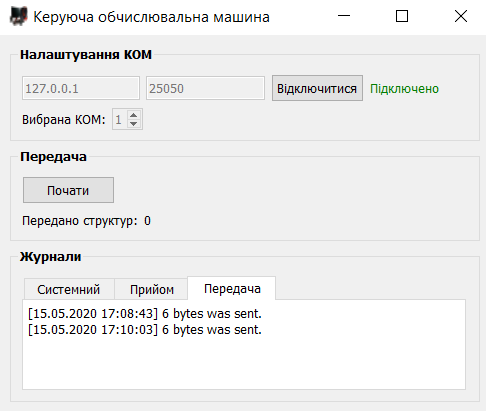


Рис. 4.2 – Екранні форми ПЗ КОМ: передача даних на сервер

У розділі журналів знаходяться три журнали: системний, журнал прийому та передачі. Вони розміщені у вигляді вкладок, при активації одної з яких, буд виведено текстове поле з повідомленнями відповідного журналу.

При з’єднанні з сервером, КОМ посилає перше повідомлення з вибраним номером КОМ. На початку передачі даних запускається таймер, що буде виконувати передачу однієї структури через заданий інтервал. Зупинка передачі зупиняє цей таймер. При втраті з’єднання з сервером, КОМ автоматично зупиняє передачу, якщо вона була активною, та відключається від серверу (закриває сокет з’єднання).

Всі налаштування програми зберігаються за допомогою INI-файлу. При кожному виході з програми у INI-файл записуються налаштування, що були виконані під час роботи. При кожному запуску програми, вони автоматично загружаються з нього.

Для реалізації ПЗ КОМ було використано парадигму ООП. Діаграма класів наведена на рисунку нижче.

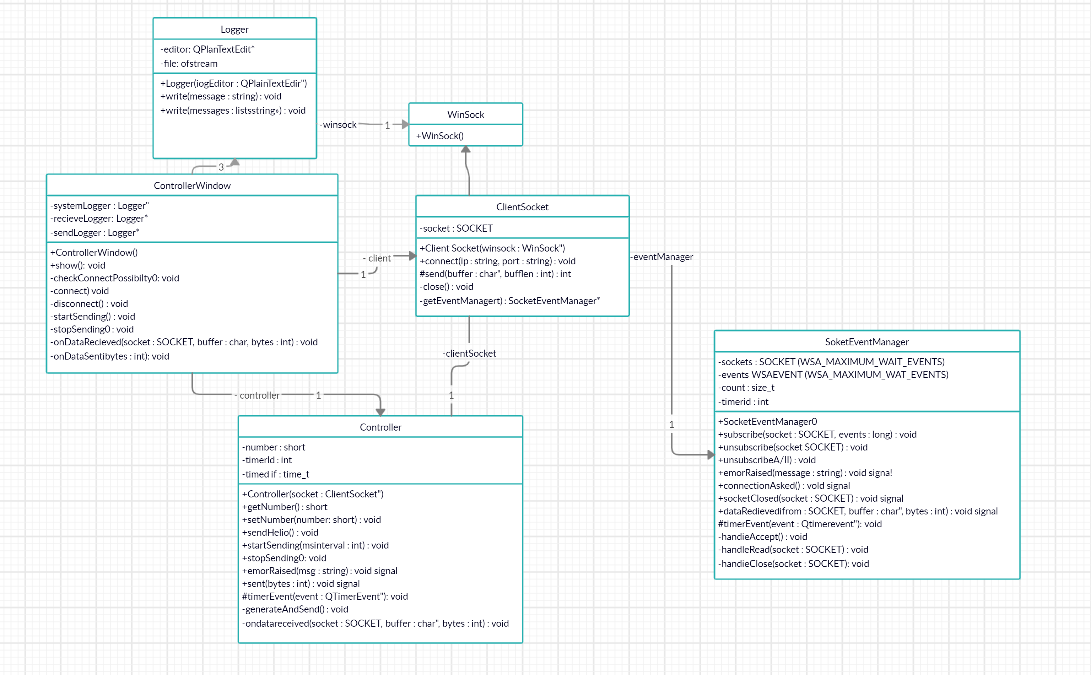


Рис. 4.3 – Діаграма класів, використаних у ПЗ КОМ

Клас WinSock у конструкторі ініціалізує бібліотеку WindowsSockets. При знищенні об’єкту цього класу бібліотека закривається. Цей клас було розроблено для того, щоб виділити загальний код, що використовується декількома класами у межах всієї роботи та щоб прямо показати, що для створення екземпляру класу сокета необхідно спочатку створити об’єкт WinSock, тому що він запитується у конструкторах класів сокетів.

Клас ClientSocketвикористовується як сокет зі сторони клієнту для встановлення зв’язку з сервером та передачі даних між ним:

1. socket – номер сокету;
2. eventManager – екземпляр SocketEventManager для контролювання подій, що виникають на сокеті;
3. ClientSocket – конструктор, запитує об’єкт WinSock;
4. connect – з’єднується з сервером;
5. send – відправляє дані на сервер;
6. close – закриває з’єднання з сервером;
7. getEventManager – повертає екземпляр класу SocketEventManager.

Клас SocketEventManager вміщає в собі реалізацію механізму вводу-виводу WSAEventSelect. Працює на основі таймеру з інтервалом у 100мс:

1. sockets – номери сокетів, на які була виконана підписка о подіях;
2. events – події сокетів, які потрібно відстежувати;
3. count – кількість сокетів/подій (завжди співпадають);
4. timerId – ідентифікатор таймеру, який запускається для перевірки на нові події, що сталися;
5. SocketEventManager – конструктор;
6. subscribe – підписує сокет на події;
7. unsubscribe – відписує сокет;
8. unsubscribeAll – відписує всі сокети;
9. errorRaised – сигнал Qt про помилку, що виникла під час роботи;
10. connectionAsked – сигнал Qt про запит на нове підключення клієнта;
11. socketClosed – сигнал Qt про закриття сокету;
12. dataRecieved – сигнал Qt про отримання даних;
13. timerEvent – таймер, що виконує перевірку на нові події;
14. handleAccept – обробляє подію про нове підключення клієнта;
15. handleRead – обробляє подію про отримання даних;
16. handleClose – обробляє подію про закриття сокету.

Клас Controllerвикористовується для реалізації основних функцій КОМ, що працюють з обробкою та передачею даних.

1. number – номер КОМ;
2. timerId – ідентифікатор таймеру, що відсилає дані з заданим інтервалом;
3. timediff – різниця у часі з сервером; використовується для синхронізації;
4. clientSocket – екземпляр ClientSocket у якості сокету клієнта;
5. Controller – конструктор, запитує екземпляр ClientSocket;
6. getNumber – повертає номер КОМ;
7. setNumber – встановлює номер КОМ;
8. sendHello – відсилає серверу початкове повідомлення;
9. startSending – запускає таймер та починає передачу;
10. stopSending – зупиняє таймер та передачу;
11. errorRaised – сигнал Qt про помилку, що виникла під час роботи;
12. sent – сигнал Qtпро відіслані дані;
13. timerEvent – таймер, що генерує та відсилає дані;
14. generateAndSend – генерує та відсилає наступну структуру даних;
15. onDataRecieved – обробляє подію, коли з серверу прийшли дані.

Клас Loggerвикористовується для відформатованого запису повідомлень у журнали:

1. editor – графічний елемент, що представляє собою графічне поле, у яке будуть записуватися повідомлення;
2. file – файл для логування повідомлень;
3. Logger – конструктор, щозапитує цей графічний елемент;
4. write – записує повідомлення; існує у двох варіантах: перший приймає строку з повідомленням, а другий список таких строк.

Клас ControllerWindowпредставляє собою саме вікно КОМ і виконує в основному функції обробки подій графічних компонентів:

1. systemLogger – екземпляр Loggerдля запису повідомлень у системний журнал;
2. recieveLogger – екземпляр Loggerдля запису повідомлень у журнал про дані, що прийшли з серверу;
3. sendLogger – екземпляр Loggerдля запису повідомлень у журнал про дані, що були відправлені;
4. winsock – екземпляр WinSockдля ініціалізації бібліотеки WindowsSocketsта передачі його іншим об’єктам, що запитують цей об’єкт;
5. client – екземпляр ClientSocketу якості сокету клієнта;
6. controller – екземпляр Controller для роботи КОМ;
7. ControllerWindow – конструктор вікна;
8. show – показує вікно;
9. checkConnectPossibility – перевіряє чи були коректно введені IP-адреса та порт серверу і у результаті вмикає чи вимикає можливість з’єднання з сервером;
10. connect – обробляє натискання на кнопку з’єднання з сервером;
11. disconnect – обробляє натискання на кнопку відключення від серверу;
12. startSending – обробляє натискання на кнопку початку передачі;
13. stopSending – обробляє натискання на кнопку закінчення передачі;
14. onDataRecieved – обробляє подію, коли на сокет прийшли дані;
15. onDataSent – обробляє подію, коли з сокету відправлено дані.

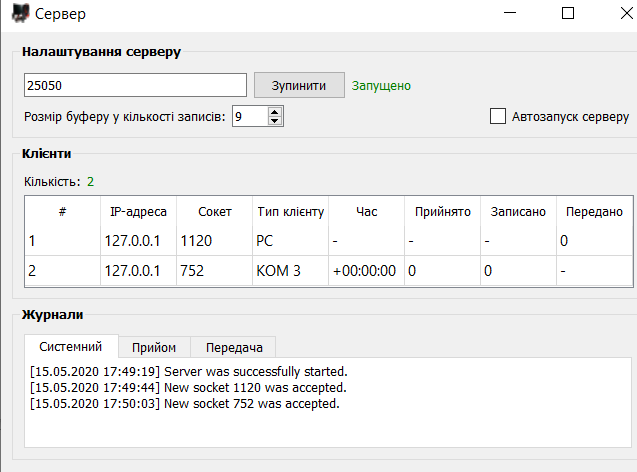
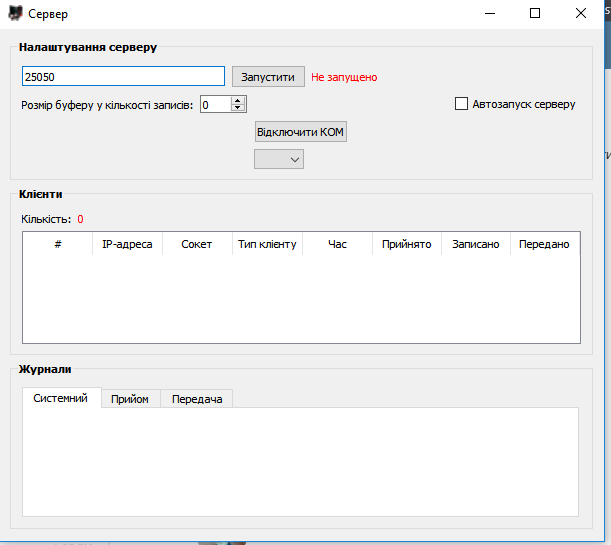
Емуляція даних від датчиків реалізовано через генерацію псевдовипадкових значень, ключем яких задано тип КОМ (перший чи другий). Таким чином на всіх КОМ одного типу будуть генеруватися однакові дані, але на КОМ різного типу вони будуть різні.

Контроль та обробка виняткових керованих ситуацій (виключення) у ПЗ КОМ оброблюється за допомогою конструкцій С++ try-catch, у частині обробки яких повідомлення про помилки записуються у системний журнал. Всі некеровані ситуації (помилки), що перевіряються на наяву, перетворюються у виключення і далі оброблюються аналогічно. Всі помилки, що виникають у таймерах, контролюються за допомогою сигналів Qt. Класи підписуються на ці сигнали і оброблюють помилки, що прийшли з них, аналогічно виключенням – записують повідомлення про них у системний журнал. При цьому для кожного журналу автоматично створюються відповідні файли, у які також записуються всі повідомлення.

4.3 Програмне забезпечення серверу

Екранні форми програмного забезпечення серверу в різних режимах роботи наведені на рисунках нижче.

Вікно серверу також, як і вікно КОМ, ділиться на частини: налаштування, клієнти та журнали. У розділі налаштувань знаходиться поле для вводу порту прослуховування, що буде використовуватись сервером. Поряд з ним розміщено кнопку для запуску серверу та статус роботи. Ці два елементи змінюються аналогічно КОМ при запуску. Нижче можна налаштувати розмір буферу прийому у кількості записів, що дозволяє регулювати навантаження на сервер. Нижче можна відєднати КОМ вибрав його зі списку і нажати кнопку відєднати .



а) б)

Рис. 4.4 – Екранні форми ПЗ серверу: а) не запущений сервер; б) запущений сервер з підключеними клієнтами

У розділі клієнтів знаходиться таблиця з підключеними в даний момент клієнтами та різна інформація про них. Загальна інформація це IP-адреса, номер сокету, тип клієнту (КОМ та її номер чи РС). Далі, інформація для КОМ включає кількість прийнятих та записаних структур даних, а інформація РС – кількість переданих. Також над таблицею знаходиться лічильник з кількістю підключених клієнтів.

Частина з журналами повністю співпадає з відповідним розділом вікна програмного забезпечення КОМ.

При запуску ПЗ серверу, ініціалізується бібліотека WindowsSockets. При запуску серверу створюється сокет і здійснюється перехід в режим прослуховування – за допомогою таймеру кожні 100мс перевіряється чи сталися якісь події на сокеті прослуховування або на сокетах клієнтах, що згодом підключаються. При відключенні серверу, він закриває всі сокети. При цьому клієнти автоматично отримують повідомлення про відключення серверу і оброблюють це відповідним чином. При відключенні клієнтів, за допомогою механізму вводу-виводу серверу також приходять повідомлення про те, що клієнт був відключений.

При запуску серверу, якщо на комп’ютері, де він працює база даних ще не створена, ПЗ серверу створює її та необхідні таблиці. Під час роботи, файл БД блокується. Коли приходять нові дані з КОМ, вони додаються у вектор коли прийшло вже 5 повідомлень тільки тоді вони додаються в БД. При вивантаженні його у базу даних, виконується SQL-запит на додавання нових записів. Коли приходить запит від РС на вибірку даних, буфер автоматично вивантажується у БД незалежно від ступеню наповненості, після чого зразу ж виконується SQL-запит на вибірку даних. Вибираються дані, що були отримані у заданий РС період.

Дані з КОМ, що є не актуальними, ігноруються і нікуди не додаються. Перевірка на те, чи є отримані дані актуальними, здійснюється за часом їх відправлення.

Зберігання налаштувань програми реалізовано аналогічно реалізації ПЗ КОМ за допомогою INI-файлів.

Для реалізації ПЗ серверу було використано парадигму ООП. Діаграма класів наведена на рисунку нижче.

Класи WinSock, SocketEventManagerта Loggerвже були описані у розділі реалізації програмного забезпечення КОМ. Структура Messageта успадковані від неї структури були розглянуті у розділі з структурами даних. Далі розглянемо інші класи, що використовуються у ПЗ серверу.

Клас ServerSocketвикористовується для реалізації роботи з сокетом зі сторони серверу:

1. listenSocket – номер сокету прослуховування;
2. clients – список номерів сокетів підключених клієнтів;
3. eventManager – екземпляр SocketEventManagerдля контролювання подій, що виникають на сокеті прослуховування та сокетах клієнтів;
4. ServerSocket – конструктор; приймає екземпляр WinSockяк параметр аналогічно класу ClientSocket;
5. listen – починає прослуховування на предмет нових підключень;
6. send – відправляє дані клієнту;
7. close – закриває сокет прослуховування та сокети всіх клієнтів;
8. acceptClient – узгоджує запит на нове підключення клієнта;
9. closeClient – закриває заданий сокет клієнта;
10. closeAllClients – закриває сокети всіх клієнтів;
11. getEventManager – повертає екземпляр SocketEventManager;
12. getClientIp – повертає IP-адресу клієнта за його сокетом;
13. bindSocket– створює сокет прослуховування та прив’язує його до заданого порту.

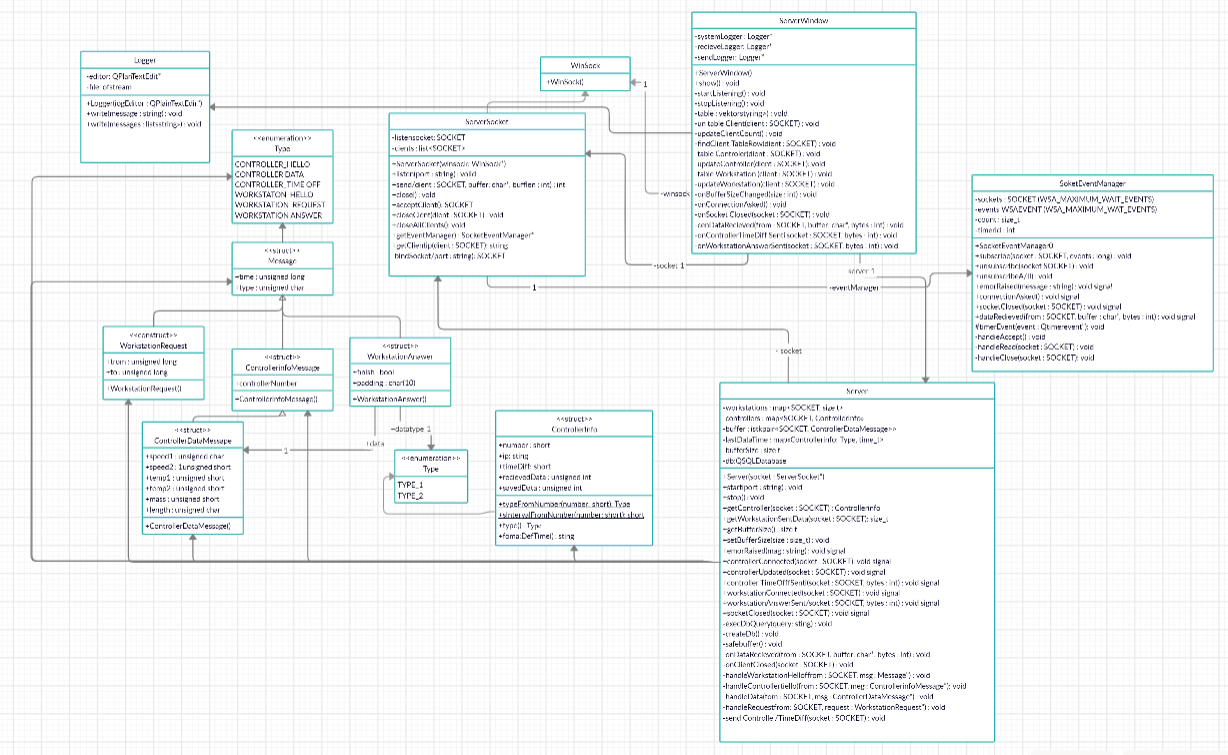


Рис. 4.5 – Діаграма класів, використаних у ПЗ серверу

Клас Serverвиконує всі основні функції серверу, що не стосуються графічних елементів вікна:

1. workstations – асоціативний масив РС; ключем елементу є номер сокету робочої станції, а значенням – кількість переданих до РС записів;
2. controllers – асоціативний масив КОМ; ключем елементу є номер сокету КОМ, а значенням – структура ControllerInfo, у якій міститься біль детальна інформація про КОМ;
3. buffer – буфер прийому зі списком повідомлень КОМ, що на даний момент ще не були записані у базу даних;
4. lastDataTime – асоціативний масив значень часу останніх повідомлень з КОМ, згрупованих по типам КОМ; ключем елементу є тип КОМ, а значенням – значення часу останнього повідомлення відповідного типу;
5. bufferSize – заданий користувачем розмір буферу прийому;
6. db – екземпляр вбудованого в Qtкласу для роботи з базами даних;
7. socket – екземпляр ServerSocket для роботи з сокетом прослуховування та його клієнтами зі сторони серверу;
8. Server – конструктор; запитує об’єкт ServerSocketдля роботи;
9. start – запускає сервер по заданому порту;
10. stop – зупиняє сервер;
11. getController – повертає інформацію про підключений КОМ у вигляді структури ControllerInfoпо його сокету;
12. getWorkstationSentData – повертає кількість відісланих записів підключеного РС по його сокету;
13. getBufferSize – повертає розмір буферу прийому;
14. setBufferSize – встановлює розмір буферу прийому;
15. errorRaised – сигнал Qt про помилку, що виникла під час роботи;
16. controllerConnected – сигнал Qtпро підключення нової КОМ;
17. controllerUpdated – сигнал Qtпро оновлення інформації підключеного КОМ;
18. controllerTimeDiffSent – сигнал Qtпро передачу КОМ її різниці у часі з сервером;
19. workstationConnected – сигнал Qtпро підключення нового РС;
20. workstationAnswerSent – сигнал Qtпро передачу відповіді РС;
21. socketClosed – сигнал Qtпро закриття сокету клієнта;
22. execDbQuery – допоміжний метод для виконання SQL-запитів у БД;
23. created – створює БД та всі необхідні таблиці у ній;
24. saveBuffer – вивантажує буфер прийому у БД;
25. onDataRecieved – оброблює подію про отримання даних;
26. onClientSocket – оброблює подію про закриття сокету клієнта;
27. handleWorkstationHello – оброблює початкове повідомлення РС;
28. handleControllerHello – оброблює початкове повідомлення КОМ;
29. handleData – оброблює прийняті дані від КОМ;
30. handleRequest – оброблює запит на вибірку даних РС;
31. sendControllerTimeDiff – відсилає КОМ її різницю у часі з сервером.

Для більш зручного способу зберігання інформації про КОМ була створена структура ControllerInfo:

1. number – номер КОМ;
2. ip – IP-адреса;
3. timeDiff – різниця у часі з сервером;
4. recieveData – кількість отриманих структур даних від КОМ;
5. savedData – кількість збережених у БД структур даних від КОМ;
6. typeFromNumber – статичний метод, що повертає тип КОМ за заданим номером;
7. sIntervalFromNumber – статичний метод, який повертає інтервал передачі дних КОМ по її номеру згідно за завданням;
8. type – метод екземпляру, що повертає тип КОМ;
9. formatDiffTime – повертає строку різниці у часі КОМ з сервером у форматованому вигляді.

Клас ServerWindowпредставляє собою вікно ПЗ серверу:

1. systemLogger – екземпляр Loggerдля запису повідомлень у системний журнал;
2. recieveLogger – екземпляр Loggerдля запису повідомлень у журнал про дані, що прийшли з серверу;
3. sendLogger – екземпляр Loggerдля запису повідомлень у журнал про дані, що були відправлені;
4. winsock – екземпляр WinSockдля ініціалізації бібліотеки WindowsSocketsта передачі його іншим об’єктам, що запитують цей об’єкт;
5. socket – екземпляр ServerSocketдля роботи з сокетами прослуховування та клієнтів;
6. server – екземпляр Serverдля роботи серверу;
7. ServerWindow – конструктор;
8. show – показує вікно серверу;
9. startListening – оброблює натискання на кнопку запуску серверу;
10. stopListening – оброблює натискання на кнопку зупинки серверу;
11. tableClient – додає список строк з інформацією клієнта у таблицю;
12. untableClient – видаляє клієнта по його сокету з таблиці;
13. updateClientCount – оновлює поточне значення кількості клієнтів;
14. findClientTableRow – знаходить номер рядку клієнта у таблиці;
15. tableController – додає підключений КОМ у таблицю;
16. updateController – оновлює інформацію про КОМ у таблиці;
17. tableWorkstation – додає підключений РС у таблицю;
18. updateWorkstation – оновлює інформацію про РС у таблиці;
19. onBufferSizeChanged – оброблює подію про зміну розміру буферу прийому;
20. onConnectionAsked – оброблює подію про запит на нове підключення клієнта;
21. onSocketClosed – оброблює подію про закриття сокету клієнта;
22. onDataRecieved – оброблює подію про отримання даних;
23. onControllerTimeDiffSent – оброблює подію про відправлення КОМ її різниці у часі з сервером;
24. onWorkstationAnswerSent – оброблює подію про відправлення відповіді РС.

Контроль та обробка виняткових ситуацій у ПЗ серверу виконується таким же чином, як і у ПЗ КОМ. Принципи реалізації журналів, де фіксуються всі події ПЗ також співпадають з ПЗ КОМ.Також у ПЗ серверу передбачена можливість роботи у фоновому режимі.

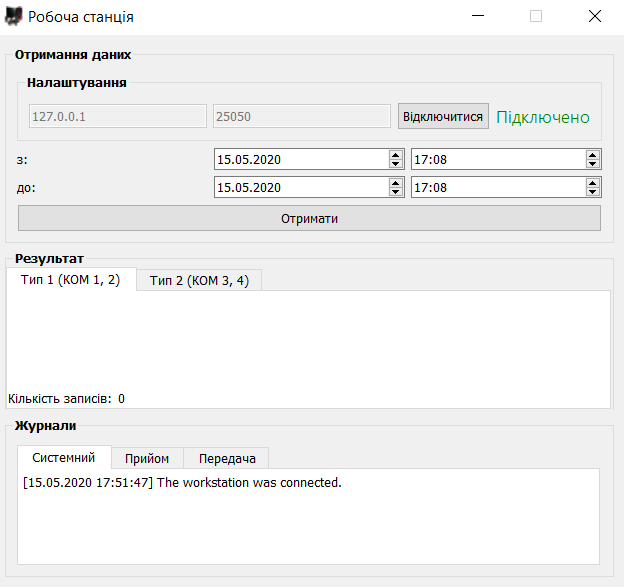
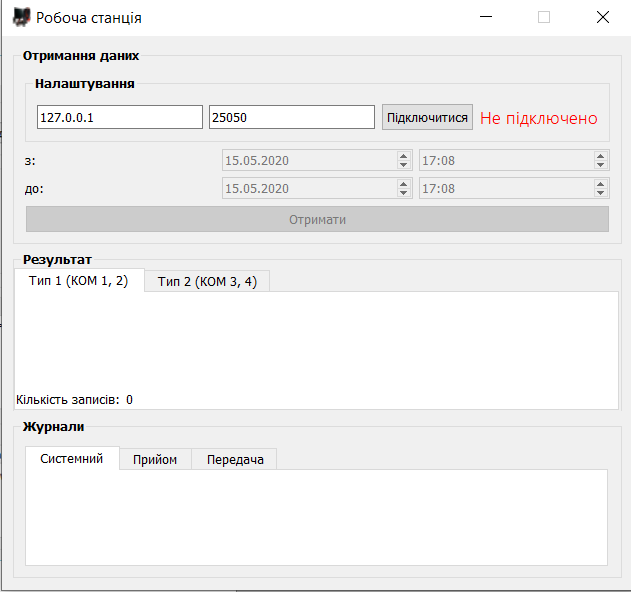
4.4 Програмне забезпечення РС

Екранні форми програмного забезпечення РС в різних режимах роботи наведені на рисунках нижче.

Вікно робочої станції ділиться на чотири розділи: налаштування, отримання даних, результат та журнали. Розділ налаштувань та журналів повністю співпадає з відповідними розділами вікна КОМ.

У розділі отримання даних знаходиться два поля для вибору відповідно дати та часу початку отримання даних і два таких же поля для кінця отримання даних. Нижче знаходиться кнопка, що запускає процес отримання даних з серверу.

У розділі результату розміщено дві таблиці на кожний тип даних. Вони розділені вкладками. В один момент може бути активною лише одна вкладка. Також під кожною таблицею знаходиться лічильник з кількістю записів у таблиці.



а) б)

Рис. 4.6 – Екранні форми ПЗ РС: а) до з’єднання з сервером; б) після встановлення з’єднання з сервером

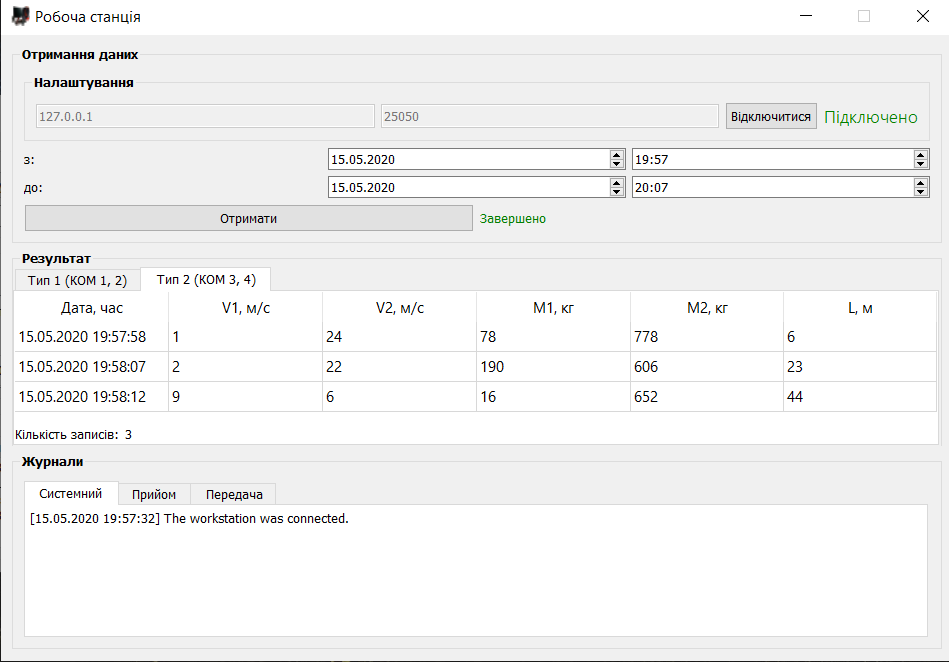


Рис. 4.7 – Екранні форми ПЗ РС: отримання даних з серверу

При з’єднанні з сервером, РС посилає перше повідомлення, у якому зазначено, що повідомлення відсилає саме робоча станція. Для отримання даних РС відсилає відповідне повідомлення серверу з періодом часу, після чого оброблює кожну нову структуру даних, що приходить з серверу як відповідь та заносить ці дані у відповідну таблицю. Якщо у відповіді зазначений кінець передачі, РС оброблює це відповідним чином. При втраті з’єднання з сервером, РС автоматично відключається від серверу (закриває сокет з’єднання).

Для реалізації ПЗ РС було використано парадигму ООП. Діаграма класів наведена на рисунку нижче.

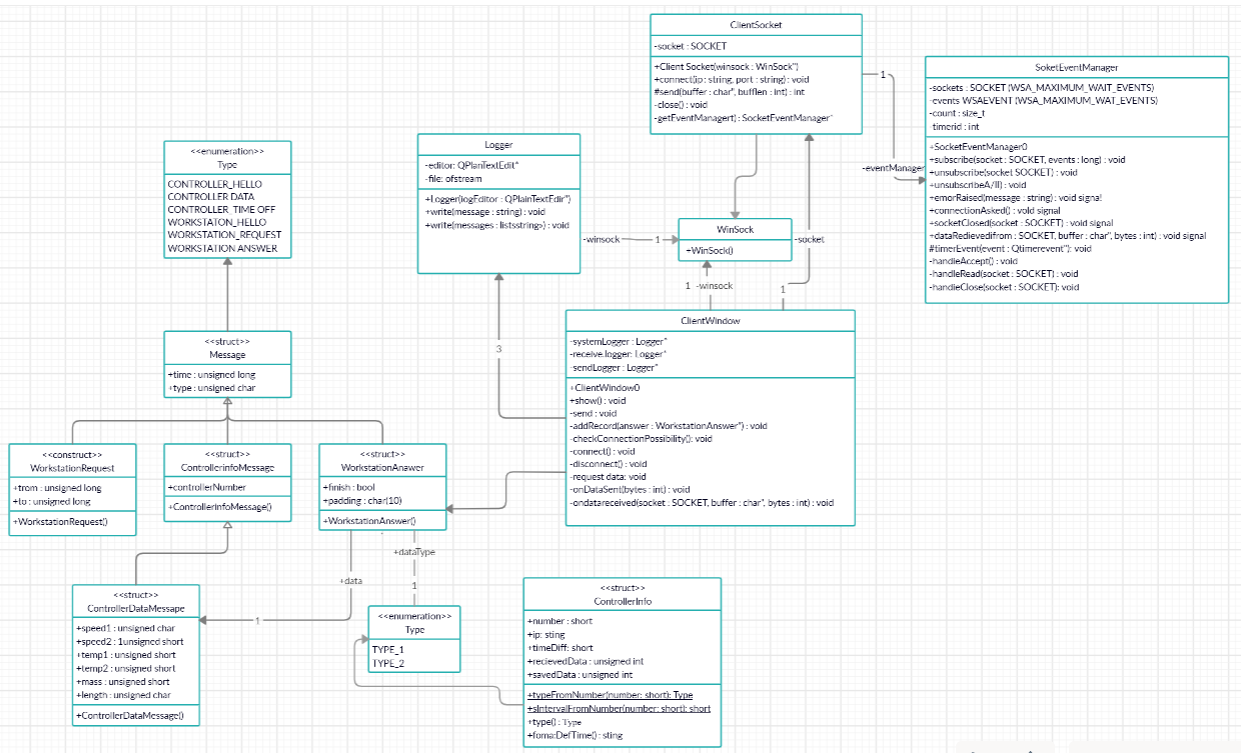


Рис. 4.8 – Діаграма класів, використаних у ПЗ РС

Всі наведені класи і структури, окрім класу ClientWindow, вже були використані і описані у попередніх розділах, тому розглянемо лише клас ClientWindow, що реалізує вікно ПЗ РС:

1. systemLogger – екземпляр Loggerдля запису повідомлень у системний журнал;
2. recieveLogger – екземпляр Loggerдля запису повідомлень у журнал про дані, що прийшли з серверу;
3. sendLogger – екземпляр Loggerдля запису повідомлень у журнал про дані, що були відправлені;
4. winsock – екземпляр WinSockдля ініціалізації бібліотеки WindowsSocketsта передачі його іншим об’єктам, що запитують цей об’єкт;
5. socket – екземпляр ClientSocketдля роботи з сокетом як клієнт;
6. ClientWindow – конструктор;
7. show – показує вікно;
8. sendHello – відсилає початкове повідомлення на сервер;
9. addRecord – додає дані з відповіді серверу у таблицю результатів;
10. checkConnectionPossibility – перевіряє коректність введених IP-адресу та порту серверу і у разі коректного вводу, вмикає доступність підключення до серверу;
11. connect – підключається до серверу;
12. disconnect – відключається від серверу;
13. requestData – відсилає запит на сервер на вибірку даних;
14. onDataSent – оброблює подію про відправлення даних;
15. onDataRecieved – оброблює подію про отримання даних (відповіді від серверу).

Методи контролю та обробки виняткових ситуацій, журнали, де фіксуються всі події ПЗ, та налаштування програми реалізовано таким же способом, як в ПЗ КОМ і серверу.

ВИСНОВКИ

У данній курсовій роботі були виконані задачі по розробці програмного забезпечення серверу, робочих станцій (РС), та керуючих обчислювальних машин (КОМ).

ПЗ серверу приймає дані від КОМ, зберігає їх у базу даних, та відсилає їх робочим станціям (РС) при запиті на вибірку даних у заданому проміжку часу. Також сервер синхронізує підключені КОМ.

ПЗ РС відправляє запити серверу на вибірку даних та відображає їх у своїх таблицях з результатами.

ПЗ КОМ збирає дані з датчиків і передає їх на сервер. Також отримує повідомлення від серверу з різницею у часі для синхронізації.

Під час роботи були отримані навички реалізації мережної взаємодії компонентів систем, що побудовані за архітектурою «клієнт-сервер»на основі протоколу TCP/IPза допомогою бібліотеки WindowsSockets.

У програми є такі перспективи розвитку та поліпшення:

– оптимізувати роботу алгоритмів;

– виконати рефакторинг коду та поліпшити архітектуру класів таким чином, щоб робота з сокетами була більш нативно зрозумілою;

– реалізувати перевірки повідомлень, що приходять на сокети, на коректність, щоб ускладнити зовнішнє втручання у систему;

– оптимізувати структури даних по пам’яті;

– уніфікувати архітектуру класів, щоб можна було без великого втручання у код додавати нові типи КОМ та інших клієнтських додатків.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Джонс, Э. Программирование в сетях Microsoft Windows. Мастер класс / Э. Джонс, Дж. Оланд ; Пер. с англ. – СПб. : Питер ; М. : Издательско-торговый дом «Русская редакция», 2002. – 608 c.

2. Снейдер, Й. Эффективное программирование TCP/IP. Библиотека программиста / Й. Снейдер. – СПб. : Питер, 2002. – 320 с. : ил.

3. Douglas Comer. Internetworking with TCP/IP / Douglas Comer. – Volume One (6th Ed.). – Pearson, 2013. – 744 p.

4. SQLite [Електронний ресурс]. – Режим доступу:<https://ru.wikipedia.org/wiki/SQLite>

5. Craig Hunt. TCP/IP Network Administration / Craig Hunt – 3nd Ed. – O'Reilly Media, 2002.

Додаток А

Лістинг ПЗ серверу

definitions.h:

#include <string>

#include <sstream>

#include <iomanip>

#unclude <winsock>

struct ControllerInfo{

enum Type : unsigned short { TYPE\_1 = 1, TYPE\_2 = 2 };

static Type typeFromNumber(short number){

return number < 3 ? TYPE\_1 : TYPE\_2;

}

static short sIntervalFromNumber(short number) {

return number < 3 ? 11 : 9;

}

SOCKET socket;

short number;

std::string ip;

short timeDiff;

unsigned recievedData;

unsigned savedData;

Type type(){

return ControllerInfo::typeFromNumber(number);

}

std::string formatDiffTime(){

std::stringstream stream;

stream << (timeDiff < 0 ? '-' : '+');

stream << std::setw(2) << std::setfill('0') << abs(timeDiff) / 3600 << ':';

stream << std::setw(2) << std::setfill('0') << (abs(timeDiff) / 60) % 60 << ':';

stream << std::setw(2) << std::setfill('0') << abs(timeDiff) % 60;

return stream.str();

}

};

#pragma pack(push, 1)

struct Message{

enum Type : unsigned char {

CONTROLLER\_HELLO,

CONTROLLER\_DATA,

CONTROLLER\_TIMEDIFF,

WORKSTATION\_HELLO,

WORKSTATION\_REQUEST,

WORKSTATION\_ANSWER

};

unsigned long time;

unsigned char type;

};

#pragma pack(pop)

struct ControllerInfoMessage : Message{

unsigned char controllerNumber;

ControllerInfoMessage() {

type = Message::CONTROLLER\_HELLO;

}

};

#pragma pack(push, 1)

struct ControllerDataMessage : ControllerInfoMessage{

unsigned char speed1, speed2;

unsigned short temp1, temp2;

unsigned short mass;

unsigned char length;

ControllerDataMessage() {

type = Message::CONTROLLER\_DATA;

}

};

#pragma pack(pop)

struct TimeDiffMessage : Message{

short timediff;

TimeDiffMessage() {

type = Message::CONTROLLER\_TIMEDIFF;

}

};

struct WorkstationRequest : Message{

unsigned long from;

unsigned long to;

WorkstationRequest() {

type = Message::WORKSTATION\_REQUEST;

}

};

#pragma pack(push, 1)

struct WorkstationAnswer : Message{

ControllerDataMessage data;

ControllerInfo::Type dataType;

bool finish;

char padding[9];

WorkstationAnswer() {

type = Message::WORKSTATION\_ANSWER;

}

};

#pragma pack(pop)

Клас SocketEventManager:

#include "SocketEventManager.h"

SocketEventManager::SocketEventManager() :

count(0)

{

timerId = startTimer(1000);

}

SocketEventManager::~SocketEventManager(){

unsubscribeAll();

killTimer(timerId);

}

void SocketEventManager::subscribe(SOCKET socket, long events) noexcept{

WSAEVENT newEvent = WSACreateEvent();

WSAEventSelect(socket, newEvent, events);

sockets[count] = socket;

this->events[count] = newEvent;

count += 1;

}

/\*\*

\* Remove subscriber from sockets and events arrays and !compress! them

\* (all elements are shifted left by one).

\*/

void SocketEventManager::unsubscribe(SOCKET socket) noexcept{

size\_t index;

for (index = 0; index < count; ++index) {

if (sockets[index] == socket) break;

}

if (index == count) return;

for (auto i = index; i < count - 1; ++i) {

sockets[i] = sockets[i + 1];

events[i] = events[i + 1];

}

count -= 1;

}

void SocketEventManager::unsubscribeAll() noexcept{

while (count != 0) unsubscribe(sockets[0]);

}

void SocketEventManager::timerEvent(QTimerEvent \*){

DWORD index = WSAWaitForMultipleEvents(count, events, false, 0, false);

if (index - WSA\_WAIT\_EVENT\_0 >= count) return;

WSANETWORKEVENTS networkEvents;

SOCKET socket = sockets[index - WSA\_WAIT\_EVENT\_0];

WSAEVENT event = events[index - WSA\_WAIT\_EVENT\_0];

WSAEnumNetworkEvents(socket, event, &networkEvents);

if (networkEvents.lNetworkEvents & FD\_ACCEPT) {

if (networkEvents.iErrorCode[FD\_ACCEPT\_BIT] != 0) {

emit errorRaised(

QString("FD\_ACCEPT failed with error %1")

.arg(networkEvents.iErrorCode[FD\_ACCEPT\_BIT])

);

return;

}

handleAccept();

}

if (networkEvents.lNetworkEvents & FD\_READ) {

if (networkEvents.iErrorCode[FD\_READ\_BIT] != 0) {

emit errorRaised(

QString("FD\_READ failed with error %1")

.arg(networkEvents.iErrorCode[FD\_READ\_BIT])

);

return;

}

handleRead(socket);

}

if (networkEvents.lNetworkEvents & FD\_CLOSE) {

if (networkEvents.iErrorCode[FD\_CLOSE\_BIT] != 0) {

emit errorRaised(

QString("Close event resulted with error %1")

.arg(networkEvents.iErrorCode[FD\_CLOSE\_BIT])

);

}

handleClose(socket);

}

}

void SocketEventManager::handleAccept() noexcept{

if (count + 1 > WSA\_MAXIMUM\_WAIT\_EVENTS) {

emit errorRaised("There are too many connections already.");

return;

}

emit connectionAsked();

}

void SocketEventManager::handleRead(SOCKET socket) noexcept{

char buffer[512];

int bytesRecieved = recv(socket, buffer, sizeof(buffer), 0);

emit dataRecieved(socket, buffer, bytesRecieved);

}

void SocketEventManager::handleClose(SOCKET socket) noexcept{

unsubscribe(socket);

closesocket(socket);

emit socketClosed(socket);

}

Клас WinSock:

#include "WinSock.h"

#include <WinSock2.h>

#include <QString>

WinSock::WinSock(){

WSADATA wsaData;

int result = WSAStartup(MAKEWORD(2,2), &wsaData);

if (result != 0) throw QString("WSAStartup failed with error: %1").arg(result);

}

WinSock::~WinSock() noexcept{

WSACleanup();

}

Клас ServerSocket:

#include "ServerSocket.h"

#include <WS2tcpip.h>

#include <QString>

using namespace std;

ServerSocket::ServerSocket(WinSock \*) noexcept :

QObject(),

listenSocket(INVALID\_SOCKET),

eventManager(new SocketEventManager)

{}

ServerSocket::~ServerSocket() noexcept{

close();

delete eventManager;

}

void ServerSocket::listen(string port){

if (listenSocket != INVALID\_SOCKET) throw QString("Socket is listening already");

listenSocket = bindSocket(port);

int result = ::listen(listenSocket, SOMAXCONN);

if (result == SOCKET\_ERROR) {

closesocket(listenSocket);

listenSocket = INVALID\_SOCKET;

throw QString("listen failed with error: %1").arg(WSAGetLastError());

}

eventManager->subscribe(listenSocket, FD\_ACCEPT);

}

int ServerSocket::send(SOCKET client, const char \*buffer, int bufferlen){

int result = ::send(client, buffer, bufferlen, 0);

if (result == SOCKET\_ERROR) {

throw QString("Send failed with error: %1").arg(WSAGetLastError());

}

return result;

}

void ServerSocket::close() noexcept{

if (listenSocket == INVALID\_SOCKET) return;

eventManager->unsubscribeAll();

closeAllClients();

shutdown(listenSocket, SD\_BOTH);

closesocket(listenSocket);

listenSocket = INVALID\_SOCKET;

}

SOCKET ServerSocket::acceptClient(){

SOCKET newClient = ::accept(listenSocket, nullptr, nullptr);

if (newClient == INVALID\_SOCKET) throw QString("There is no new client");

clients.push\_back(newClient);

eventManager->subscribe(newClient, FD\_READ | FD\_CLOSE);

return newClient;

}

void ServerSocket::closeClient(SOCKET client) noexcept{

eventManager->unsubscribe(client);

clients.remove(client);

shutdown(client, SD\_BOTH);

closesocket(client);

}

void ServerSocket::closeAllClients() noexcept{

while (!clients.empty()) closeClient(clients.front());

}

SocketEventManager\* ServerSocket::getEventManager() const{

return eventManager;

}

string ServerSocket::getClientIp(SOCKET client) const{

SOCKADDR\_IN info = {}; // For TCP/IP.

int addrsize = sizeof(info); // This is probably too.

getsockname(client, reinterpret\_cast<struct sockaddr\*>(&info), &addrsize);

char buffer[INET\_ADDRSTRLEN];

inet\_ntop(info.sin\_family, &info.sin\_addr, buffer, INET\_ADDRSTRLEN);

return buffer;

}

SOCKET ServerSocket::bindSocket(string port){

SOCKET resultSocket = INVALID\_SOCKET;

struct addrinfo \*addressInfo = nullptr;

struct addrinfo hints;

int iResult;

ZeroMemory(&hints, sizeof(hints));

hints.ai\_family = AF\_INET;

hints.ai\_socktype = SOCK\_STREAM;

hints.ai\_protocol = IPPROTO\_TCP;

hints.ai\_flags = AI\_PASSIVE;

// Resolve the server address and port

iResult = getaddrinfo(nullptr, port.c\_str(), &hints, &addressInfo);

if (iResult != 0) {

throw QString("getaddrinfo failed with error: %1").arg(WSAGetLastError());

}

// Create a SOCKET for connecting to server

resultSocket = socket(addressInfo->ai\_family,

addressInfo->ai\_socktype,

addressInfo->ai\_protocol);

if (resultSocket == INVALID\_SOCKET) {

freeaddrinfo(addressInfo);

throw QString("socket failed with error: %1").arg(WSAGetLastError());

}

iResult = ::bind(resultSocket,

addressInfo->ai\_addr,

static\_cast<int>(addressInfo->ai\_addrlen));

if (iResult == SOCKET\_ERROR) {

freeaddrinfo(addressInfo);

closesocket(resultSocket);

throw QString("bind failed with error: %1").arg(WSAGetLastError());

}

freeaddrinfo(addressInfo);

return resultSocket;

}

Клас Logger:

#include "Logger.h"

#include <QDateTime>

#include <QDir>

using namespace std;

Logger::Logger(QPlainTextEdit \*logEditor) :

QObject(),

editor(logEditor)

{

QDir dir;

if (!dir.exists("logs")) dir.mkdir("logs");

auto filename = string("logs/") + to\_string(time(nullptr))

+ "-" + logEditor->whatsThis().toStdString() + ".txt";

file.open(filename);

if (!file.is\_open())

this->write(QString("Cannot open file %1").arg(filename.c\_str()));

}

Logger::~Logger(){

if (file.is\_open()) file.close();

}

void Logger::write(const QString &string){

write(QStringList(string));

}

void Logger::write(const QStringList &messages){

QDateTime datetime = QDateTime::currentDateTime();

for (auto message : messages) {

QString timeMessage = "[" + datetime.toString("dd.MM.yyyy hh:mm:ss") + "] ";

QString output = timeMessage + message;

editor->appendPlainText(output);

file << output.toStdString().c\_str() << endl;

}

}

Клас Server:

#include "Server.h"

#include <QSettings>

using namespace std;

Server::Server(ServerSocket \*socket) :

QObject(),

socket(socket),

bufferSize(0)

{

db = QSqlDatabase::addDatabase("QSQLITE");

db.setDatabaseName("db.sqlite");

if (!db.open()) throw db.lastError().text();

if (db.tables().size() == 0) createDb();

lastDataTime[ControllerInfo::TYPE\_1] = 0;

lastDataTime[ControllerInfo::TYPE\_2] = 0;

auto eventManager = socket->getEventManager();

QObject::connect(eventManager, &SocketEventManager::dataRecieved,

this, &Server::onDataRecieved);

QObject::connect(eventManager, &SocketEventManager::socketClosed,

this, &Server::onClientClosed);

vectorMsg.reserve(5);

}

Server::~Server()

{

saveBuffer();

db.close();

}

void Server::start(string port)

{

socket->listen(port);

}

void Server::stop()

{

socket->close();

saveBuffer();

auto controllers = this->controllers;

for (auto &item : controllers) onClientClosed(item.first);

auto workstations = this->workstations;

for (auto &item : workstations) onClientClosed(item.first);

}

const ControllerInfo& Server::getController(SOCKET socket) const

{

return controllers.at(socket);

}

const ControllerInfo& Server::getControllerNumber(int number) const

{

return controllers.at(number);

}

const ControllerInfo& Server::getControllerFromNumber(int number) const

{

for(auto it = controllers.begin(); it != controllers.end(); ++it)

{

if (it->second.number == number)

return it->second;

}

}

size\_t Server::getWorkstationSentData(SOCKET socket) const

{

return workstations.at(socket);

}

size\_t Server::getBufferSize() const

{

return bufferSize;

}

void Server::setBufferSize(size\_t size) noexcept

{

if (size < bufferSize) saveBuffer();

bufferSize = size;

}

void Server::execDbQuery(const QString &query)

{

db.exec(query);

if (db.lastError().isValid()) throw QString("Query: " + query + "\n" +

"Error: " + db.lastError().text());

}

void Server::createDb()

{

execDbQuery("CREATE TABLE data ("

" type TINYINT NOT NULL,"

" time UNSIGNED BIG INT NOT NULL,"

" speed1 INTEGER DEFAULT NULL,"

" speed2 INTEGER DEFAULT NULL,"

" temp1 INTEGER DEFAULT NULL,"

" mass INTEGER DEFAULT NULL,"

" temp2 INTEGER DEFAULT NULL,"

" length INTEGER DEFAULT NULL"

");");

}

void Server::saveBuffer()

{

for (auto &item : buffer) {

if (controllers.find(item.first) != controllers.end()) {

controllers[item.first].savedData += 1;

emit controllerUpdated(item.first);

}

bool vectorHave5Msgs = false;

auto \*msg = &item.second;

auto type = ControllerInfo::typeFromNumber(msg->controllerNumber);

mapFiveLastMessages.insert( pair<unsigned short, std::vector<ControllerDataMessage>>(type,vectorMsg));

for (auto it = mapFiveLastMessages.begin(); it != mapFiveLastMessages.end(); ++it)

{

// проходимо по всім записам який хранить мап<Номер КОМу, Вектор на 5 елементів для повідомлень>

ControllerDataMessage AverageControllerMsg;

AverageControllerMsg.mass = 0;

AverageControllerMsg.temp1 = 0;

AverageControllerMsg.temp2 = 0;

AverageControllerMsg.length = 0;

AverageControllerMsg.speed1 = 0;

AverageControllerMsg.speed2 = 0;

AverageControllerMsg.time = 0;

//Перевіряемо якщо вектор повідомлень не дорівнює 5

if(isSizeVectorNotEqualNumber(it->second, 5)){

vectorMsg.insert(vectorMsg.begin(), \*msg);

} else {

// Якщо в векторі вже 5 повідомлень то видаляємо останне повідомлення

vectorMsg.pop\_back();

//Та додаемо нове повідомленя в початок

vectorMsg.insert(vectorMsg.begin(), \*msg);

}

for (auto itVectorMsgs = it->second.begin(); itVectorMsgs != it->second.end(); ++itVectorMsgs)

{

//Проходимо по останнім 5 повідомленням КОМу

// Якщо немає ще 5 повідомлень у КОМа

if(isSizeVectorNotEqualNumber(it->second, 5)){

break;

} else {

vectorHave5Msgs = true;

//Cкладуемо

AverageControllerMsg.mass += itVectorMsgs->mass;

AverageControllerMsg.temp1 += itVectorMsgs->temp1;

AverageControllerMsg.temp2 += itVectorMsgs->temp2;

AverageControllerMsg.length += itVectorMsgs->length;

AverageControllerMsg.speed1 += itVectorMsgs->speed1;

AverageControllerMsg.speed2 += itVectorMsgs->speed2;

AverageControllerMsg.time += itVectorMsgs->time;

}

}

if(vectorHave5Msgs){

//Ділимо на кількість слагаемих, так як у нас вектор на 5 елементів то ділимо на 5

AverageControllerMsg.mass /= it->second.size();

AverageControllerMsg.temp1 /= it->second.size();

AverageControllerMsg.temp2 /= it->second.size();

AverageControllerMsg.length /= it->second.size();

AverageControllerMsg.speed1 /= it->second.size();

AverageControllerMsg.speed2 /= it->second.size();

AverageControllerMsg.time /= it->second.size();

execDbQuery(QString("INSERT INTO data ("

" type, time, speed1, speed2, "

" temp1, mass, temp2, length"

") VALUES ("

" %1, %2, %3, %4,"

" %5, %6, %7, %8 "

");")

.arg(static\_cast<unsigned short>(type))

.arg(AverageControllerMsg.time)

.arg(AverageControllerMsg.speed1).arg(AverageControllerMsg.speed2)

.arg(AverageControllerMsg.temp1)

.arg(AverageControllerMsg.mass).arg(AverageControllerMsg.temp2)

.arg(AverageControllerMsg.length)

);

}

}

buffer.clear();

}

}

bool Server::isSizeVectorNotEqualNumber(std::vector<ControllerDataMessage> v, int needSize)

{

if(needSize > v.size()){

return true;

}

else{

return false;

}

}

void Server::onDataRecieved(SOCKET from, char \*buffer, int) noexcept

{

Message\* msg = reinterpret\_cast<Message\*>(buffer);

try {

switch (msg->type)

{

case Message::CONTROLLER\_HELLO:

handleControllerHello(from, reinterpret\_cast<ControllerInfoMessage\*>(msg));

break;

case Message::CONTROLLER\_DATA:

handleData(from, reinterpret\_cast<ControllerDataMessage\*>(msg));

break;

case Message::WORKSTATION\_HELLO:

handleWorkstationHello(from, msg);

break;

case Message::WORKSTATION\_REQUEST:

handleRequest(from, reinterpret\_cast<WorkstationRequest\*>(msg));

break;

default:

return;

}

} catch (const QString &msg) {

emit errorRaised(msg);

}

}

void Server::onClientClosed(SOCKET socket) noexcept

{

controllers.erase(socket);

workstations.erase(socket);

emit socketClosed(socket);

}

void Server::handleWorkstationHello(SOCKET from, const Message \*)

{

workstations[from] = 0;

emit workstationConnected(from);

}

void Server::handleControllerHello(SOCKET from, const ControllerInfoMessage \*msg)

{

ControllerInfo controller;

controller.socket = from;

controller.ip = socket->getClientIp(from);

controller.number = msg->controllerNumber;

controller.recievedData = 0;

controller.savedData = 0;

controller.timeDiff = static\_cast<short>(msg->time - time(nullptr));

controllers[from] = controller;

emit controllerConnected(from);

if (controller.timeDiff != 0) sendControllerTimeDiff(from);

}

void Server::handleData(SOCKET from, const ControllerDataMessage \*msg)

{

ControllerInfo controller = controllers[from];

controller.timeDiff = static\_cast<short>(msg->time - time(nullptr));

controller.recievedData += 1;

controllers[from] = controller;

if (controller.timeDiff != 0) sendControllerTimeDiff(from);

auto type = ControllerInfo::typeFromNumber(msg->controllerNumber);

auto sInterval = ControllerInfo::sIntervalFromNumber(msg->controllerNumber);

if (msg->time - lastDataTime[type] < sInterval) {

emit controllerUpdated(from);

return;

}

lastDataTime[type] = msg->time;

buffer.push\_back({from, \*msg});

if (buffer.size() >= bufferSize) saveBuffer();

else emit controllerUpdated(from);

}

void Server::handleRequest(SOCKET from, const WorkstationRequest \*request)

{

saveBuffer();

QSqlQuery query;

query.exec(QString("SELECT \* FROM data WHERE time BETWEEN %1 AND %2")

.arg(request->from).arg(request->to));

if (db.lastError().isValid())

throw QString("Select query error: " + db.lastError().text());

WorkstationAnswer answer;

answer.finish = false;

int bytes = 0;

while (query.next()) {

answer.dataType = static\_cast<ControllerInfo::Type>(query.value("type").toInt());

answer.data.time = static\_cast<unsigned long>(query.value("time").toULongLong());

answer.data.speed1 = static\_cast<unsigned char>(query.value("speed1").toUInt());

answer.data.speed2 = static\_cast<unsigned char>(query.value("speed2").toUInt());

answer.data.temp1 = static\_cast<unsigned short>(query.value("temp1").toUInt());

answer.data.mass = static\_cast<unsigned short>(query.value("mass").toUInt());

answer.data.temp2 = static\_cast<unsigned short>(query.value("temp2").toUInt());

answer.data.length = static\_cast<unsigned char>(query.value("length").toUInt());

answer.time = static\_cast<unsigned long>(time(nullptr));

bytes += socket->send(from, reinterpret\_cast<char\*>(&answer), sizeof(WorkstationAnswer));

workstations[from] += 1;

}

answer.finish = true;

answer.time = static\_cast<unsigned long>(time(nullptr));

bytes += socket->send(from, reinterpret\_cast<char\*>(&answer), sizeof(WorkstationAnswer));

emit workstationAnswerSent(from, bytes);

}

void Server::sendControllerTimeDiff(SOCKET socket)

{

auto controller = controllers[socket];

TimeDiffMessage message;

message.timediff = controller.timeDiff;

message.time = static\_cast<unsigned long>(time(nullptr));

int size = sizeof(TimeDiffMessage);

int bytes = this->socket->send(socket, reinterpret\_cast<char\*>(&message), size);

emit controllerTimeDiffSent(socket, bytes);

}}

Клас ServerWindow:

#include "ServerWindow.h"

#include "ui\_ServerWindow.h"

#include <WS2tcpip.h>

#include <vector>

#include <QSettings>

using namespace std;

ServerWindow::ServerWindow(QWidget \*parent) :

QWidget(parent),

ui(new Ui::ServerWindow),

winsock(nullptr),

socket(nullptr),

server(nullptr)

{

ui->setupUi(this);

ui->clientsTable->horizontalHeader()->setSectionResizeMode(QHeaderView::Stretch);

ui->clientsTable->verticalHeader()->setSectionResizeMode(QHeaderView::ResizeToContents);

ui->stopButton->hide();

ui->startedLabel->hide();

systemLogger = new Logger(ui->systemLog);

recieveLogger = new Logger(ui->recieveLog);

sendLogger = new Logger(ui->sendLog);

try {

winsock = new WinSock;

socket = new ServerSocket(winsock);

server = new Server(socket);

}

catch (const QString &msg) {

systemLogger->write(msg);

return;

}

QObject::connect(ui->portInput, &QLineEdit::textChanged, [=] (const QString &newText) {

ui->startButton->setEnabled(newText.contains(QRegExp("^[0-9]{1,5}$")));

});

QObject::connect(ui->startButton, &QPushButton::clicked,

this, &ServerWindow::startListening);

QObject::connect(ui->stopButton, &QPushButton::clicked,

this, &ServerWindow::stopListening);

QObject::connect(ui->bufferSize, SIGNAL(valueChanged(int)),

this, SLOT(onBufferSizeChanged(int)));

auto eventManager = socket->getEventManager();

QObject::connect(eventManager, SIGNAL(errorRaised(const QString &)),

systemLogger, SLOT(write(const QString &)));

QObject::connect(eventManager, &SocketEventManager::connectionAsked,

this, &ServerWindow::onConnectionAsked);

QObject::connect(eventManager, &SocketEventManager::dataRecieved,

this, &ServerWindow::onDataRecieved);

QObject::connect(server, SIGNAL(errorRaised(const QString &)),

systemLogger, SLOT(write(const QString &)));

QObject::connect(server, &Server::controllerConnected,

this, &ServerWindow::tableController);

QObject::connect(server, &Server::controllerUpdated,

this, &ServerWindow::updateController);

QObject::connect(server, &Server::controllerTimeDiffSent,

this, &ServerWindow::onControllerTimeDiffSent);

QObject::connect(server, &Server::workstationConnected,

this, &ServerWindow::tableWorkstation);

QObject::connect(server, &Server::workstationAnswerSent,

this, &ServerWindow::onWorkstationAnswerSent);

QObject::connect(server, &Server::socketClosed,

this, &ServerWindow::onSocketClosed);

QObject::connect(server, &Server::controllerConnected,

this, &ServerWindow::addClientToComboBox);

QObject::connect(ui->disconectButton, &QPushButton::clicked,

this, &ServerWindow::stopListening);

QSettings settings("settings.ini", QSettings::IniFormat);

ui->portInput->setText(settings.value("port", "").toString());

ui->bufferSize->setValue(settings.value("buffsize", 0).toInt());

ui->autostart->setChecked(settings.value("autostart", false).toBool());

if (ui->autostart->isChecked() && ui->startButton->isEnabled()) startListening();

}

ServerWindow::~ServerWindow() {

QSettings settings("settings.ini", QSettings::IniFormat);

settings.setValue("port", ui->portInput->text());

settings.setValue("buffsize", ui->bufferSize->value());

settings.setValue("autostart", ui->autostart->isChecked());

if (server) {

delete server;

server = nullptr;

}

if (socket) {

socket->close();

delete socket;

socket = nullptr;

}

delete winsock;

delete sendLogger;

delete recieveLogger;

delete systemLogger;

delete ui;

}

void ServerWindow::changeEvent(QEvent \*event) {

QWidget::changeEvent(event);

if (event->type() == QEvent::WindowStateChange)

if (isMinimized()) this->hide();

}

void ServerWindow::startListening() {

try {

server->setBufferSize(static\_cast<size\_t>(ui->bufferSize->value()));

server->start(ui->portInput->text().toStdString());

ui->portInput->setReadOnly(true);

ui->startButton->hide();

ui->stopButton->show();

ui->stoppedLabel->hide();

ui->startedLabel->show();

systemLogger->write("Server was successfully started.");

}

catch (const QString &msg) {

systemLogger->write(msg);

}

}

void ServerWindow::stopListening() {

try {

server->stop();

ui->stopButton->hide();

ui->startButton->show();

ui->startedLabel->hide();

ui->stoppedLabel->show();

ui->portInput->setReadOnly(false);

ui->startButton->setFocus();

systemLogger->write("Server was stopped.");

} catch (const QString &msg) {

systemLogger->write(msg);

}

}

void ServerWindow::tableController(SOCKET client) noexcept {

vector<QString> columns;

auto info = server->getController(client);

columns.push\_back(QString::fromStdString(info.ip));

columns.push\_back(QString("%1").arg(client));

columns.push\_back(QString("КОМ %1").arg(info.number));

columns.push\_back(QString::fromStdString(info.formatDiffTime()));

columns.push\_back(QString("%1").arg(info.recievedData));

columns.push\_back(QString("%2").arg(info.savedData));

columns.push\_back("-");

tableClient(columns);

}

void ServerWindow::updateController(SOCKET client) noexcept

{

auto info = server->getController(client);

int row = findClientTableRow(client);

ui->clientsTable->item(row, 4)->setText(QString::fromStdString(info.formatDiffTime()));

ui->clientsTable->item(row, 5)->setText(QString("%1").arg(info.recievedData));

ui->clientsTable->item(row, 6)->setText(QString("%1").arg(info.savedData));

}

void ServerWindow::tableWorkstation(SOCKET client) noexcept

{

auto sentData = server->getWorkstationSentData(client);

vector<QString> columns;

columns.push\_back(QString::fromStdString(socket->getClientIp(client)));

columns.push\_back(QString("%1").arg(client));

columns.push\_back("PC");

columns.push\_back("-");

columns.push\_back("-");

columns.push\_back("-");

columns.push\_back(QString("%1").arg(sentData));

tableClient(columns);

}

void ServerWindow::updateWorkstation(SOCKET client) noexcept{

int row = findClientTableRow(client);

size\_t sentData = server->getWorkstationSentData(client);

ui->clientsTable->item(row, 7)->setText(QString("%1").arg(sentData));

}

void ServerWindow::tableClient(const vector<QString>&columns) noexcept{

int row = ui->clientsTable->rowCount();

ui->clientsTable->insertRow(row);

ui->clientsTable->setItem(row, 0, new QTableWidgetItem(QString("%1").arg(row + 1)));

for (size\_t i = 0; i < columns.size(); ++i) {

int col = static\_cast<int>(i + 1);

ui->clientsTable->setItem(row, col, new QTableWidgetItem(columns[i]));

}

updateClientCount();

}

void ServerWindow::untableClient(SOCKET client) noexcept{

bool removed = false;

for (int row = 0; row < ui->clientsTable->rowCount(); ++row) {

if (removed) {

ui->clientsTable->item(row, 0)->setText(QString("%1").arg(row + 1));

continue;

}

if (ui->clientsTable->item(row, 2)->text().toUInt() == client) {

ui->clientsTable->removeRow(row);

removed = true;

row -= 1;

};

}

updateClientCount();

}

void ServerWindow::updateClientCount() noexcept{

int count = ui->clientsTable->rowCount();

auto color = QString("color: %1").arg(count ? "green" : "red");

ui->clientCount->setNum(count);

ui->clientCount->setStyleSheet(color);

}

int ServerWindow::findClientTableRow(SOCKET client) noexcept{

for (int i = 0; i < ui->clientsTable->rowCount(); ++i) {

if (ui->clientsTable->item(i, 2)->text().toUInt() == client) return i;

}

return -1;

}

void ServerWindow::onBufferSizeChanged(int size) noexcept{

server->setBufferSize(static\_cast<size\_t>(size));

}

void ServerWindow::onConnectionAsked() noexcept{

SOCKET newClient = socket->acceptClient();

systemLogger->write(QString("New socket %1 was accepted.").arg(newClient));

}

void ServerWindow::onSocketClosed(SOCKET socket) noexcept{

untableClient(socket);

systemLogger->write(QString("Socket %1 was closed.").arg(socket));

}

void ServerWindow::onDataRecieved(SOCKET from, char \*, int bytes) noexcept{

recieveLogger->write(

QString("%1 bytes recieved from %2 socket.").arg(bytes).arg(from));

}

void ServerWindow::onControllerTimeDiffSent(SOCKET client, int bytes) noexcept{

sendLogger->write(QString("Time difference was sent to %1 socket "

"controller in %2 bytes.").arg(client).arg(bytes));

}

void ServerWindow::onWorkstationAnswerSent(SOCKET client, int bytes) noexcept{

updateWorkstation(client);

sendLogger->write(QString("There was %1 bytes sent to workstation of %2 socket.")

.arg(bytes).arg(client));

}

//При підключенні кліента додаемо його в випадаючий список

void ServerWindow::addClientToComboBox() noexcept

{

int number = server->getControllerNumber(ui->clientsTable->rowCount()).number;

ui->clientComboBox->insertItem(number, QString::number(ui->clientsTable->rowCount()));

}

//Відключаемо КОМ

void ServerWindow::dropClient()

{

try {

int numberClient = ui->clientComboBox->itemData(ui->clientComboBox->currentIndex()).toInt();

auto client = server->getControllerFromNumber(numberClient);

server->socket->closeClient(client.socket);

systemLogger->write("Client erased successfully");

ui->clientComboBox->removeItem(numberClient);

} catch (const QString &msg) {

systemLogger->write(msg);

}

}

main.cpp:

#include "ServerWindow.h"

#include <QApplication>

#include <QSystemTrayIcon>

#include <QMenu>

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

QApplication a(argc, argv);

QIcon icon(":icon.png");

ServerWindow window;

window.setWindowIcon(icon);

QSystemTrayIcon \*trayIcon = new QSystemTrayIcon(&window);

trayIcon->setIcon(icon);

QObject::connect(trayIcon, &QSystemTrayIcon::activated,

[&] (QSystemTrayIcon::ActivationReason reason)

{

switch (reason){

case QSystemTrayIcon::Trigger:

case QSystemTrayIcon::DoubleClick:

window.showNormal();

window.activateWindow();

break;

default:

break;

}

});

trayIcon->show();

window.show();

return a.exec();

}Додаток Б

Лістинг ПЗ керуючих обчислювальних машин

Клас ClientSocket:

#include "ClientSocket.h"

#include <WS2tcpip.h>

using namespace std;

ClientSocket::ClientSocket(WinSock \*) :

QObject(),

socket(INVALID\_SOCKET),

eventManager(new SocketEventManager)

{}

ClientSocket::~ClientSocket(){

close();

delete eventManager;

}

void ClientSocket::connect(string ip, string port){

if (socket != INVALID\_SOCKET) throw QString("This socket was connected already.");

struct addrinfo \*result = nullptr, \*ptr = nullptr, hints;

int iResult;

ZeroMemory(&hints, sizeof(hints));

hints.ai\_family = AF\_UNSPEC;

hints.ai\_socktype = SOCK\_STREAM;

hints.ai\_protocol = IPPROTO\_TCP;

// Resolve the server address and port

iResult = getaddrinfo(ip.c\_str(), port.c\_str(), &hints, &result);

if (iResult != 0) {

throw QString("getaddrinfo failed with error: %1").arg(iResult);

}

// Attempt to connect to an address until one succeeds

for (ptr = result; ptr != nullptr; ptr = ptr->ai\_next) {

// Create a SOCKET for connecting to server

socket = ::socket(ptr->ai\_family, ptr->ai\_socktype, ptr->ai\_protocol);

if (socket == INVALID\_SOCKET) {

throw QString("socket failed with error: %1").arg(WSAGetLastError());

}

// Connect to server.

iResult = ::connect(socket, ptr->ai\_addr,

static\_cast<int>(ptr->ai\_addrlen));

if (iResult == SOCKET\_ERROR) {

closesocket(socket);

socket = INVALID\_SOCKET;

continue;

}

break;

}

freeaddrinfo(result);

if (socket == INVALID\_SOCKET) {

throw QString("Unable to connect to server!");

}

eventManager->subscribe(socket, FD\_READ | FD\_CLOSE);

}

int ClientSocket::send(const char \*buffer, int bufferlen){

int result = ::send(socket, buffer, bufferlen, 0);

if (result == SOCKET\_ERROR) {

throw QString("Send failed with error: %1").arg(WSAGetLastError());

}

return result;

}

void ClientSocket::close() noexcept{

if (socket == INVALID\_SOCKET) return;

eventManager->unsubscribe(socket);

shutdown(socket, SD\_BOTH);

closesocket(socket);

socket = INVALID\_SOCKET;

}

SocketEventManager\* ClientSocket::getEventManager() const{

return eventManager;

}

Клас Controller:

#include "Controller.h"

Controller::Controller(ClientSocket \*socket) noexcept :

QObject(),

socket(socket),

number(0),

timerId(-1),

timediff(0)

{

auto eventManager = socket->getEventManager();

QObject::connect(eventManager, &SocketEventManager::dataRecieved,

this, &Controller::onDataRecieved);

}

short Controller::getNumber() const{

return number;

}

void Controller::setNumber(short number) noexcept{

this->number = number;

auto type = ControllerInfo::typeFromNumber(number);

srand(static\_cast<unsigned int>(type));

}

void Controller::startSending(int sInterval) noexcept{

timerId = startTimer(sInterval \* 1000);

}

void Controller::stopSending() noexcept{

killTimer(timerId);

timerId = -1;

}

void Controller::timerEvent(QTimerEvent \*){

generateAndSend();

}

void Controller::sendHello() noexcept{

ControllerInfoMessage data = {};

data.controllerNumber = number;

try {

data.time = time(nullptr);

int sentBytes = socket->send(reinterpret\_cast<char\*>(&data),

sizeof(ControllerInfoMessage));

emit sent(sentBytes);

}

catch (const QString &msg) {

emit errorRaised(msg);

}

}

void Controller::generateAndSend() noexcept

{

auto type = ControllerInfo::typeFromNumber(number);

ControllerDataMessage data = {};

if (type == ControllerInfo::TYPE\_1) {

data.speed1 = 0 + rand() % ((20 + 1) - 0);

data.temp1 = 0 + rand() % ((1000 + 1) - 0);

data.temp2 = 0 + rand() % ((500 + 1) - 0);

data.mass = 0 + rand() % ((1000 + 1) - 0);

data.length = 0 + rand() % ((100 + 1) - 0);

} else {

data.speed1 = 0 + rand() % ((10 + 1) - 0);

data.speed2 = 0 + rand() % ((40 + 1) - 0);

data.mass = 0 + rand() % ((200 + 1) - 0);

data.temp2 = 0 + rand() % ((1000 + 1) - 0);

data.length = 0 + rand() % ((50 + 1) - 0);

}

data.controllerNumber = number;

data.time = time(nullptr) - timediff;

try {

int sentBytes = socket->send(reinterpret\_cast<char\*>(&data),

sizeof(ControllerDataMessage));

emit sent(sentBytes);

}

catch (const QString &msg) {

emit errorRaised(msg);

}

}

void Controller::onDataRecieved(SOCKET, char \*buffer, int) noexcept{

auto message = reinterpret\_cast<Message\*>(buffer);

if (message->type == Message::CONTROLLER\_TIMEDIFF) {

auto timediffMessage = reinterpret\_cast<TimeDiffMessage\*>(message);

timediff += timediffMessage->timediff;

}

}

Клас ControllerWindow:

#include "ControllerWindow.h"

#include "ui\_ControllerWindow.h"

#include <QSettings>

ControllerWindow::ControllerWindow(QWidget \*parent) :

QWidget(parent),

ui(new Ui::ControllerWindow),

winsock(nullptr),

client(nullptr),

controller(nullptr)

{

ui->setupUi(this);

ui->disconnectButton->hide();

ui->connectedLabel->hide();

ui->stopSendingButton->hide();

ui->sendingLabel->hide();

systemLogger = new Logger(ui->systemLog);

recieveLogger = new Logger(ui->recieveLog);

sendLogger = new Logger(ui->sendLog);

try {

winsock = new WinSock;

client = new ClientSocket(winsock);

controller = new Controller(client);

} catch (const QString &msg) {

systemLogger->write(msg);

return;

}

QObject::connect(ui->ipInput, SIGNAL(textChanged(const QString &)),

this, SLOT(checkConnectPossibility()));

QObject::connect(ui->portInput, SIGNAL(textChanged(const QString &)),

this, SLOT(checkConnectPossibility()));

QObject::connect(ui->connectButton, SIGNAL(clicked()),

this, SLOT(connect()));

QObject::connect(ui->disconnectButton, SIGNAL(clicked()),

this, SLOT(disconnect()));

QObject::connect(ui->startSendingButton, SIGNAL(clicked()),

this, SLOT(startSending()));

QObject::connect(ui->stopSendingButton, SIGNAL(clicked()),

this, SLOT(stopSending()));

auto eventManager = client->getEventManager();

QObject::connect(eventManager, SIGNAL(errorRaised(const QString &)),

systemLogger, SLOT(write(const QString &)));

QObject::connect(eventManager, &SocketEventManager::socketClosed,

this, &ControllerWindow::disconnect);

QObject::connect(eventManager, &SocketEventManager::dataRecieved,

this, &ControllerWindow::onDataRecieved);

QObject::connect(controller, SIGNAL(errorRaised(const QString &)),

systemLogger, SLOT(write(const QString &)));

QObject::connect(controller, &Controller::sent,

this, &ControllerWindow::onDataSent);

QSettings settings("settings.ini", QSettings::IniFormat);

ui->ipInput->setText(settings.value("ip", "").toString());

ui->portInput->setText(settings.value("port", "").toString());

}

ControllerWindow::~ControllerWindow(){

QSettings settings("settings.ini", QSettings::IniFormat);

settings.setValue("ip", ui->ipInput->text());

settings.setValue("port", ui->portInput->text());

if (controller != nullptr) {

delete controller;

controller = nullptr;

}

if (client != nullptr) {

client->close();

delete client;

client = nullptr;

}

delete winsock;

delete sendLogger;

delete recieveLogger;

delete systemLogger;

delete ui;

}

void ControllerWindow::checkConnectPossibility() noexcept{

auto ipRegexp = QRegExp("^([0-9]{1,3}[.]){3}[0-9]{1,3}$");

auto portRegexp = QRegExp("^[0-9]{1,5}$");

bool ipOk = ui->ipInput->text().contains(ipRegexp);

bool portOk = ui->portInput->text().contains(portRegexp);

ui->connectButton->setEnabled(ipOk && portOk);

}

void ControllerWindow::connect() noexcept{

try {

short number = static\_cast<short>(ui->clientSpinBox->value());

auto ip = ui->ipInput->text().toStdString();

auto port = ui->portInput->text().toStdString();

client->connect(ip, port);

controller->setNumber(number);

controller->sendHello();

systemLogger->write(QString("Controller number %1 was connected.").arg(number));

ui->connectButton->hide();

ui->disconnectButton->show();

ui->disconnectedLabel->hide();

ui->connectedLabel->show();

ui->ipInput->setEnabled(false);

ui->portInput->setEnabled(false);

ui->clientSpinBox->setEnabled(false);

ui->startSendingButton->setEnabled(true);

ui->sentStructures->setNum(0);

}

catch (const QString &msg) {

systemLogger->write(msg);

}

}

void ControllerWindow::disconnect() noexcept{

if (ui->sendingLabel->isVisible()) stopSending();

client->close();

int number = ui->clientSpinBox->value();

systemLogger->write(QString("Controller number %1 was disconnected.").arg(number));

ui->disconnectButton->hide();

ui->connectButton->show();

ui->connectedLabel->hide();

ui->disconnectedLabel->show();

ui->ipInput->setEnabled(true);

ui->portInput->setEnabled(true);

ui->clientSpinBox->setEnabled(true);

ui->startSendingButton->setEnabled(false);

}

void ControllerWindow::startSending() noexcept{

auto sInterval = ControllerInfo::sIntervalFromNumber(controller->getNumber());

controller->startSending(sInterval);

ui->startSendingButton->hide();

ui->stopSendingButton->show();

ui->sendingLabel->show();

systemLogger->write("Data sending was started.");

}

void ControllerWindow::stopSending() noexcept{

controller->stopSending();

ui->sendingLabel->hide();

ui->stopSendingButton->hide();

ui->startSendingButton->show();

systemLogger->write("Data sending was stopped");

}

void ControllerWindow::onDataRecieved(SOCKET, char \*, int bytes) noexcept{

recieveLogger->write(QString("%1 bytes recieved.").arg(bytes));

}

void ControllerWindow::onDataSent(int bytes) noexcept{

int sentStructures = ui->sentStructures->text().toInt() + 1;

ui->sentStructures->setText(QString("%1").arg(sentStructures));

sendLogger->write(QString("%1 bytes was sent.").arg(bytes));

}

main.cpp:

#include "ControllerWindow.h"

#include <QApplication>

#include <QIcon>

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

QApplication a(argc, argv);

QIcon icon(":icon.png");

ControllerWindow window;

window.setWindowIcon(icon);

window.show();

return a.exec();

}

Додаток В

Лістинг ПЗ робочих станцій

Клас ClientWindow:

#include "ClientWindow.h"

#include "ui\_ClientWindow.h"

#include <vector>

#include <QSettings>

using namespace std;

ClientWindow::ClientWindow(QWidget \*parent) :

QWidget(parent),

ui(new Ui::ClientWindow),

winsock(nullptr),

socket(nullptr)

{

ui->setupUi(this);

ui->type1Table->horizontalHeader()->setSectionResizeMode(QHeaderView::Stretch);

ui->type1Table->verticalHeader()->setSectionResizeMode(QHeaderView::ResizeToContents);

ui->type2Table->horizontalHeader()->setSectionResizeMode(QHeaderView::Stretch);

ui->type2Table->verticalHeader()->setSectionResizeMode(QHeaderView::ResizeToContents);

ui->disconnectButton->hide();

ui->connectedLabel->hide();

ui->proccessingLabel->hide();

ui->recievedLabel->hide();

ui->fromDate->setDate(QDate::currentDate());

ui->fromTime->setTime(QTime::currentTime());

ui->toDate->setDate(QDate::currentDate());

ui->toTime->setTime(QTime::currentTime());

systemLogger = new Logger(ui->systemLog);

recieveLogger = new Logger(ui->recieveLog);

sendLogger = new Logger(ui->sendLog);

try {

winsock = new WinSock;

socket = new ClientSocket(winsock);

} catch (const QString &msg) {

systemLogger->write(msg);

return;

}

QObject::connect(ui->ipInput, SIGNAL(textChanged(const QString &)),

this, SLOT(checkConnectPossibility()));

QObject::connect(ui->portInput, SIGNAL(textChanged(const QString &)),

this, SLOT(checkConnectPossibility()));

QObject::connect(ui->connectButton, SIGNAL(clicked()),

this, SLOT(connect()));

QObject::connect(ui->disconnectButton, SIGNAL(clicked()),

this, SLOT(disconnect()));

QObject::connect(ui->recieveButton, SIGNAL(clicked()),

this, SLOT(requestData()));

auto eventManager = socket->getEventManager();

QObject::connect(eventManager, SIGNAL(errorRaised(const QString &)),

systemLogger, SLOT(write(const QString &)));

QObject::connect(eventManager, &SocketEventManager::socketClosed,

this, &ClientWindow::disconnect);

QObject::connect(eventManager, &SocketEventManager::dataRecieved,

this, &ClientWindow::onDataRecieved);

QSettings settings("settings.ini", QSettings::IniFormat);

ui->ipInput->setText(settings.value("ip", "").toString());

ui->portInput->setText(settings.value("port", "").toString());

}

ClientWindow::~ClientWindow(){

QSettings settings("settings.ini", QSettings::IniFormat);

settings.setValue("ip", ui->ipInput->text());

settings.setValue("port", ui->portInput->text());

if (socket != nullptr) {

delete socket;

socket = nullptr;

}

if (winsock != nullptr) {

delete winsock;

winsock = nullptr;

}

delete sendLogger;

delete recieveLogger;

delete systemLogger;

delete ui;

}

void ClientWindow::sendHello() noexcept{

Message msg;

msg.type = Message::WORKSTATION\_HELLO;

msg.time = static\_cast<unsigned long>(time(nullptr));

try {

int bytes = socket->send(reinterpret\_cast<char\*>(&msg), sizeof(Message));

onDataSent(bytes);

} catch (const QString &msg) {

systemLogger->write(msg);

}

}

void ClientWindow::addRecord(const WorkstationAnswer \*answer) noexcept{

QTableWidget \*table;

QLabel \*countLabel;

vector<QString> columns;

QDateTime datetime = QDateTime::fromSecsSinceEpoch(answer->data.time);

auto time = datetime.toString("dd.MM.yyyy hh:mm:ss");

columns.push\_back(time);

if (answer->dataType == ControllerInfo::TYPE\_1) {

table = ui->type1Table;

countLabel = ui->type1Count;

columns.push\_back(QString("%1").arg(answer->data.speed1));

columns.push\_back(QString("%1").arg(answer->data.temp1));

columns.push\_back(QString("%1").arg(answer->data.temp2));

columns.push\_back(QString("%1").arg(answer->data.mass));

columns.push\_back(QString("%1").arg(answer->data.length));

} else {

table = ui->type2Table;

countLabel = ui->type2Count;

columns.push\_back(QString("%1").arg(answer->data.speed1));

columns.push\_back(QString("%1").arg(answer->data.speed2));

columns.push\_back(QString("%1").arg(answer->data.mass));

columns.push\_back(QString("%1").arg(answer->data.temp2));

columns.push\_back(QString("%1").arg(answer->data.length));

}

int row = table->rowCount();

table->insertRow(row);

countLabel->setNum(row + 1);

for (size\_t i = 0; i < columns.size(); ++i) {

int col = static\_cast<int>(i);

table->setItem(row, col, new QTableWidgetItem(columns[i]));

}

}

void ClientWindow::checkConnectPossibility() noexcept{

auto ipRegexp = QRegExp("^([0-9]{1,3}[.]){3}[0-9]{1,3}$");

auto portRegexp = QRegExp("^[0-9]{1,5}$");

bool ipOk = ui->ipInput->text().contains(ipRegexp);

bool portOk = ui->portInput->text().contains(portRegexp);

ui->connectButton->setEnabled(ipOk && portOk);

}

void ClientWindow::connect() noexcept{

try {

auto ip = ui->ipInput->text().toStdString();

auto port = ui->portInput->text().toStdString();

socket->connect(ip, port);

sendHello();

systemLogger->write("The workstation was connected.");

ui->connectButton->hide();

ui->disconnectButton->show();

ui->disconnectedLabel->hide();

ui->connectedLabel->show();

ui->ipInput->setEnabled(false);

ui->portInput->setEnabled(false);

ui->fromDate->setEnabled(true);

ui->fromTime->setEnabled(true);

ui->toDate->setEnabled(true);

ui->toTime->setEnabled(true);

ui->recieveButton->setEnabled(true);

}

catch (const QString &msg) {

systemLogger->write(msg);

}

}

void ClientWindow::disconnect() noexcept{

socket->close();

systemLogger->write("The workstation was disconnected.");

ui->disconnectButton->hide();

ui->connectButton->show();

ui->connectedLabel->hide();

ui->disconnectedLabel->show();

ui->ipInput->setEnabled(true);

ui->portInput->setEnabled(true);

ui->fromDate->setEnabled(false);

ui->fromTime->setEnabled(false);

ui->toDate->setEnabled(false);

ui->toTime->setEnabled(false);

ui->recieveButton->setEnabled(false);

ui->proccessingLabel->hide();

}

void ClientWindow::requestData() noexcept{

WorkstationRequest request;

QDateTime from(ui->fromDate->date(), ui->fromTime->time());

QDateTime to(ui->toDate->date(), ui->toTime->time());

request.from = static\_cast<unsigned long>(from.toSecsSinceEpoch());

request.to = static\_cast<unsigned long>(to.toSecsSinceEpoch());

request.time = static\_cast<unsigned long>(time(nullptr));

int bytes = socket->send(reinterpret\_cast<char\*>(&request), sizeof(WorkstationRequest));

onDataSent(bytes);

ui->recieveButton->setEnabled(false);

ui->recievedLabel->hide();

ui->proccessingLabel->show();

ui->type1Table->clearContents();

ui->type1Table->setRowCount(0);

ui->type2Table->clearContents();

ui->type2Table->setRowCount(0);

ui->type1Count->setNum(0);

ui->type2Count->setNum(0);

}

void ClientWindow::onDataSent(int bytes) noexcept{

sendLogger->write(QString("%1 bytes was sent.").arg(bytes));

}

void ClientWindow::onDataRecieved(SOCKET, char \*buffer, int bytes){

recieveLogger->write(QString("%1 bytes was recieved.").arg(bytes));

auto msg = reinterpret\_cast<Message\*>(buffer);

if (msg->type == Message::WORKSTATION\_ANSWER) {

while (bytes > 0) {

auto answer = reinterpret\_cast<WorkstationAnswer\*>(buffer);

bytes -= sizeof(WorkstationAnswer);

buffer += sizeof(WorkstationAnswer);

if (answer->finish) {

ui->proccessingLabel->hide();

ui->recievedLabel->show();

ui->recieveButton->setEnabled(true);

} else addRecord(answer);

}

}

}

main.cpp:

#include "ClientWindow.h"

#include <QApplication>

#include <QIcon>

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

QApplication a(argc, argv);

QIcon icon(":icon.png");

ClientWindow window;

window.setWindowIcon(icon);

window.show();

return a.exec();

}