

МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ЛЬВІВСЬКА ПОЛІТЕХНІКА

Кафедра ЕОМ



АВТОМАТИЗОВАНЕ ПРОЕКТУВАННЯ
КОМП'ЮТЕРНИХ СИСТЕМ

Task 2. SW <> HW (FEF)

Виконав:

Ст. гр. КІ-402

Середа Д.А.

Прийняв: Федак П.Р.

Львів 2024

Мета: Створити просту схему зв'язку SW(клієнт) <-> UART <-> HW(сервер).

Завдання:

1. Create a simple communication schema SW(client) <-> UART <-> HW(server).
2. The client should send a message to the server. The server should modify the message and send it back to the client.
3. Create YML file with next features:
 - a. build all binaries (create scripts in folder ci/ if need);
 - b. run tests;
 - c. create artifacts with binaries and test reports;
4. Required steps.

Варіант:

19	tik-tac-toe 3x3	XML
----	-----------------	-----

Теоретичні відомості

Комунікаційна схема SW (клієнт) <-> UART <-> HW (сервер):

Комунікаційна схема між клієнтом і сервером через UART (Universal Asynchronous Receiver-Transmitter) дозволяє обмін повідомленнями між програмним забезпеченням (SW) та апаратним забезпеченням (HW). UART є послідовним інтерфейсом передачі даних, що широко використовується в системах з низьким енергоспоживанням. У цій схемі клієнт (SW) надсилає повідомлення через UART до сервера (HW), який обробляє або модифікує його та відправляє назад на клієнт. UART працює за асинхронним протоколом, тобто передача даних не потребує синхронізації, що спрощує апаратну реалізацію.

Передача і модифікація повідомлень: Згідно з вимогами, клієнт (SW) має передати певне повідомлення серверу (HW), який змінює його (наприклад, додає префікс чи змінює текст) і надсилає змінене повідомлення назад клієнту. Це може бути реалізовано за допомогою протоколу обміну, де клієнт посилає запит, а сервер відповідає зі зміненими даними. Це допомагає не тільки перевірити двосторонню комунікацію, але й дає змогу серверу виконати певну обробку даних.

Файл конфігурації YAML:

YML-файли (YAML Ain't Markup Language) часто використовуються для опису конфігурацій та робочих процесів автоматизації, особливо в контексті CI/CD (Continuous Integration/Continuous Deployment). У цьому завданні YML файл має містити наступні етапи:

- Збірка всіх бінарних файлів: Цей етап передбачає компіляцію всього коду, а для цього можна створити відповідні скрипти у папці `ci/`.
- Запуск тестів: Автоматизовані тести дозволяють перевірити роботу системи на різних етапах. Тести можна створити для перевірки успішності зв'язку, коректності обробки повідомлень та передачі даних через UART.
- Створення артефактів: Результати збірки (бінарні файли) та звіти тестів мають бути збережені як артефакти, що дозволяє їх використовувати в наступних етапах або аналізувати у разі виникнення проблем.

Деталі реалізації

```
src > C++ main.cpp > ...
1  #include <Arduino.h>
2
3  void setup() {
4      Serial.begin(9600);
5      while (!Serial) {
6          ;
7      }
8      Serial.println("Server ready...");
9  }
10
11 void loop() {
12     if (Serial.available() > 0) {
13         String message = Serial.readStringUntil('\n');
14         String modifiedMessage = message + " [Modified by Server]";
15         Serial.println(modifiedMessage);
16     }
17     delay(100);
18 }
19
```

Рис.1. Реалізація сервера

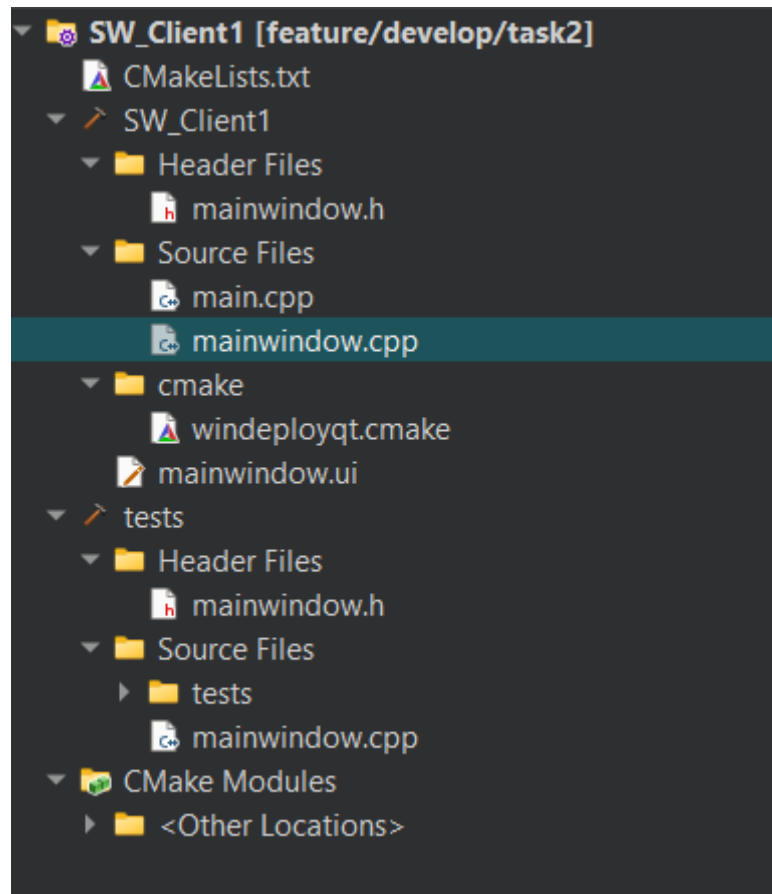


Рис.2. Реалізація клієнтської частини

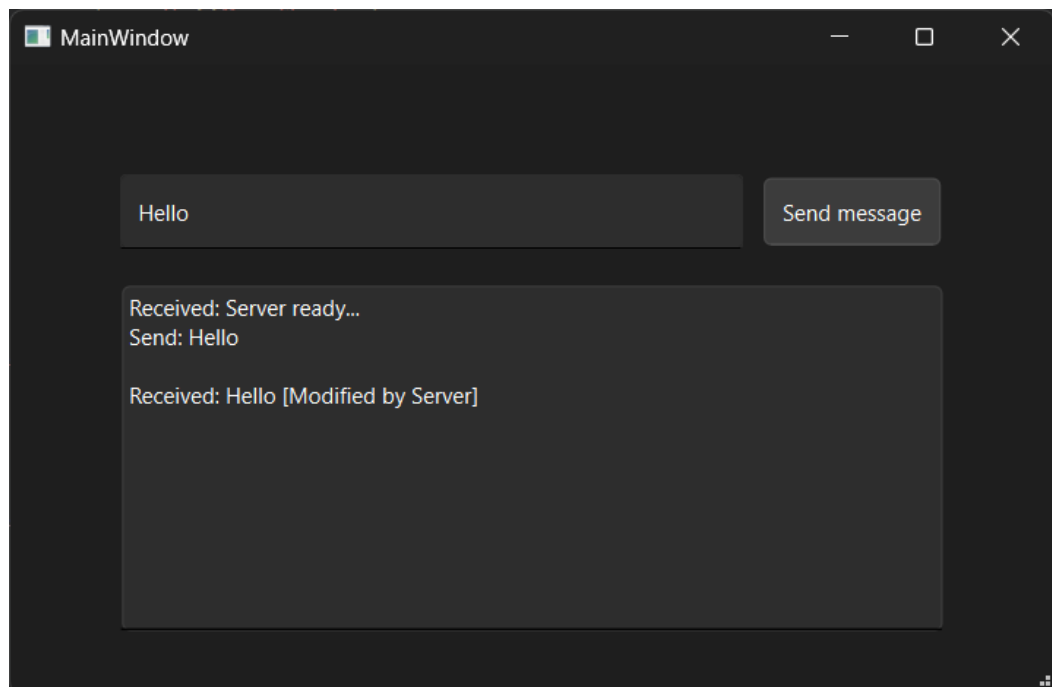


Рис.3. Інтерфейс для комунікації між клієнтом та сервером

```

1  name: CI Qt CMake (Windows) and PlatformIO
2
3  on:
4    push:
5      branches:
6        - '*'
7    pull_request:
8      branches:
9        - '*'
10
11  env:
12    BUILD_TYPE: Debug
13
14  jobs:
15    build-sw_client:
16      runs-on: windows-latest
17
18      steps:
19        # Step 1: Checkout the repository
20        - name: Checkout repository
21          uses: actions/checkout@v3
22
23        # Step 2: Install Qt
24        - name: Install Qt
25          uses: jurplel/install-qt-action@v3
26          with:
27            qtverson: '==3.1.*'
28            version: '6.8.0'
29            host: 'windows'
30            target: 'desktop'
31            arch: 'win64_msvc2022_64'
32            modules: 'qtserialport'
33
34        # Step 3: Configure CMake
35        - name: Configure CMake
36          run: cmake ${{github.workspace}}/sw_client -B ${{github.workspace}}/sw_client/build -DCMAKE_BUILD_TYPE=${{env.BUILD_TYPE}}
37

```

Рис.4. Файл конфігурації YML

```

***** Start testing of TestMainWindow *****
Config: Using QTest library 6.8.0, Qt 6.8.0 (x86_64-little_endian-llp64 shared (dynamic))
PASS   : TestMainWindow::initTestCase()
PASS   : TestMainWindow::testSerialPortSetup()
PASS   : TestMainWindow::testSendMessage()
PASS   : TestMainWindow::testReceiveMessage()
PASS   : TestMainWindow::cleanupTestCase()
Totals: 5 passed, 0 failed, 0 skipped, 0 blacklisted, 32ms
***** Finished testing of TestMainWindow *****

```

Рис.5. Перевірка роботи тестів

Task 2 #32 Re-run all jobs ...

Summary

Jobs

- build-sw_client**
- build-server

Run details

Usage

Workflow file

Annotations

1 warning

build-sw_client

succeeded 2 days ago in 2m 13s

Search logs

- Set up job 3s
- Checkout repository 6s
- Install Qt 1m 1s
- Configure CMake 33s
- Build 20s
- Run tests with CTest debug 3s
- Upload sw_client Build Artifacts 5s
- Post Install Qt 0s
- Post Checkout repository 1s
- Complete job 0s

Рис. 6. Процес автоматизованої збірки та тестування клієнта, усі етапи завершені успішно

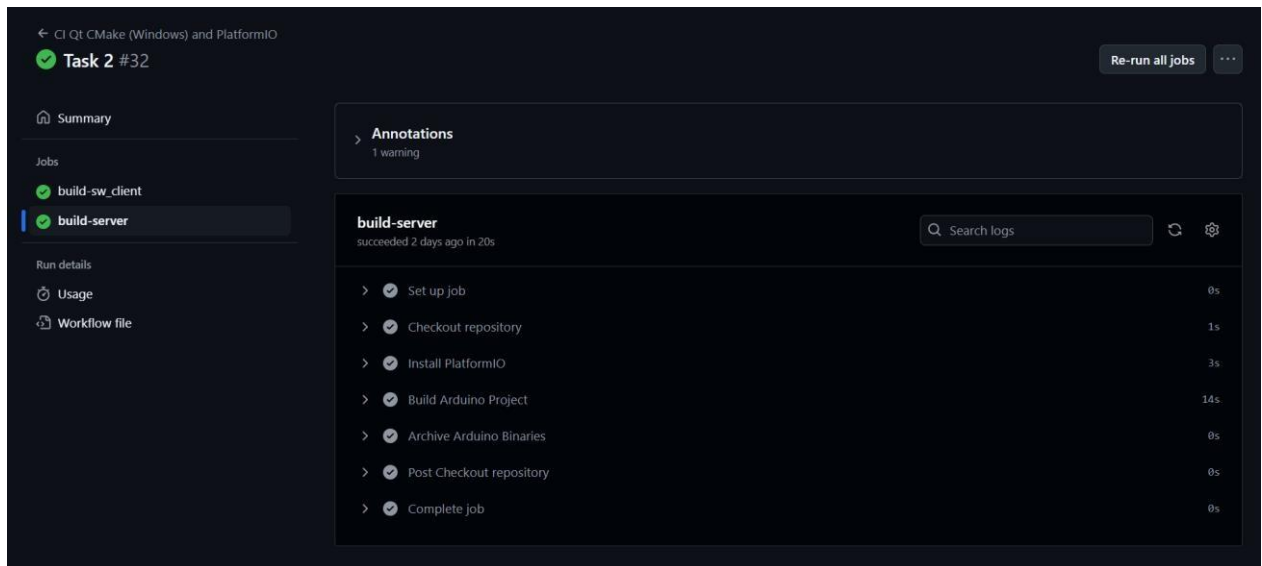


Рис. 7. Процес автоматизованої збірки та тестування сервера, усі етапи завершені успішно







Artifacts		
Produced during runtime		
Name	Size	
 arduino-binaries	536 KB	 
 sw_client-binaries	39 MB	 

Рис. 8. Артефакти, створені під час виконання завдання

Висновок:

У рамках виконання завдання було успішно реалізовано комунікаційну схему SW (клієнт) <-> UART <-> HW (сервер), де клієнт надсилає повідомлення серверу, а сервер модифікує його та повертає назад клієнту. Для автоматизації збірки та тестування проєкту було створено YAML файл, який налаштовує процеси CI/CD. YAML конфігурація включає етапи збору бінарних файлів, запуску тестів і створення артефактів, що містять результати збірки та звіти тестів. В процесі виконання збірки були налаштовані всі необхідні етапи, такі як налаштування середовища, встановлення залежностей, конфігурація CMake, виконання збірки, запуск тестів і збереження артефактів. На основі отриманих результатів (бінарні файли та тестові звіти) можна оцінити коректність реалізації комунікаційної схеми та перевірити якість роботи програмного забезпечення.

Це завдання продемонструвало переваги автоматизації процесів розробки та тестування, що зменшує можливість людських помилок і підвищує ефективність роботи команди розробників.

Список використаної літератури:

1. GitHub Docs, <https://docs.github.com/>
2. CSAD instructions for practical tasks and coursework