Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет _____Комп'ютерних наук_____
(повна назва)
Кафедра_____Програмної інженерії_____
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти - другий (магістерський)

Дослідження методів міжпроцесорної взаємодії для створення вбудованих систем
(тема)

Виконав: студент 2 курсу, групи ІПЗздм-19-2___
_____Кабанцев В.В._____
(прізвище, ініціали)

спеціальності 121- Інженерія програмного забезпечення
(код і повна назва спеціальності)

Освітньо-наукової програми_____
(тип програми)

Інженерія програмного забезпечення__
(повна назва освітньої програми)

Керівник _____доцент, к.т.н. Мазурова О.О._____
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри, проф. _____ З.В.Дудар

2021р.

Харківський національний університет радіоелектроніки

Факультет_Комп'ютерних наук_____
Кафедра_Програмної інженерії_____
Рівень вищої освіти - другий (магістерський)

Спеціальність _121-Інженерія програмного забезпечення_____
(код і повна назва)
Тип програми освітньо-наукова програма

Освітня програма_Інженерія програмного забезпечення_____

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 20 ___ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студенту _ Кабанцеву Володимиру Володимировичу_____
(прізвище, ім'я, по батькові)

1. Тема роботи__Дослідження методів міжпроцесорної взаємодії для створення вбудованих систем.
затверджена наказом університету від "_26_" __03_2020 р № ___34 Стз__
заповнюється вручну після отримання наказу
2. Термін подання студентом роботи до екзаменаційної комісії
22 травня 2021 р.
3. Вихідні дані до роботи *електронні ресурси за обраною тематикою, мінімальні вимоги до функціональності програми, загальні вимоги до архітектури системи, оптимізаційні методи та моделі,* середовище розробки Qt Creator, фреймворк Qt, gRPC, C/C++_____
4. Перелік питань, що потрібно опрацювати в роботі *аналіз предметної області і постановка задачі, аналіз вимог до програмної системи, UML-моделювання предметної області, дослідження типів міжпроцесорної взаємодії, аналіз та створення моделі прикладної системи, розробка алгоритму міжпроцесорної взаємодії, програмна реалізація системи, тестування та порівняння з існуючим аналогом.*_____
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслеників, плакатів) ____15 слайдів презентації_____

6 Консультанти розділів роботи

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата
Спецчастина	доцент, к.т.н. Мазурова О.О.		17.05.2021

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз проблемної області дослідження	21.01.2021–30.01.2021	Виконано
2	Розробка постановки задачі	02.02.2021–04.02.2021	Виконано
3	Дослідження існуючих методів міжпроцесорнох взаємодії	04.02.2021–14.02.2021	Виконано
4	Моделювання (концептуальне, UML) предметної області	14.02.2021–18.02.2021	Виконано
5	Розробка алгоритму міжпроцесорної взаємодії	18.02.2021–24.02.2021	Виконано
6	Створення коду	24.02.2021–до 20.03.21	Виконано
7	Тестування програми	20.03.21–4.04.21	Виконано
8	Проведення експериментів	4.04.21–12.04.21	Виконано
9	Написання тез доповіді на міжнародну конференцію	12.04.21–20.04.21	Виконано
10	Підготовка пояснювальної записки	20.04.21–8.05.21	Виконано
11	Підготовка презентації та доповіді	8.05.21–15.05.21	Виконано
12	Нормоконтроль	15.05.21	
13	Рецензування	17.05.21	
14	Занесення диплома в електронний архів	21.05.21	
15	Попередній захист	21.05.21	
16	Допуск до захисту у зав. кафедри	23.05.21	

Дата видачі завдання __21_ _____січня_____ 2021р.

Студент _____ Кабанцев В.В.

(підпис)

Керівник роботи _____ доц. Мазурова О.О.

(підпис)

РЕФЕРАТ / ABSTRACT

Кваліфікаційна робота магістра містить 70 стор, 19 рисунків, 3 таблиці, 4 розділи, 19 джерел.

ВБУДОВАНА СИСТЕМА, МІЖПРОЦЕСОРНА ВЗАЄМОДІЯ, МЕТОД ВЗАЄМОДІЇ, ОБМІН ДАНИМИ, RPC, GRPC, QT.

Метою роботи є дослідження методів взаємодії між процесами при побудові вбудованих систем.

Проведено дослідження методів міжпроцесорної взаємодії. Розроблено програмну систему, яка реалізує один із методів обміну даними вбудованої системи з зовнішньою системою, та порівняно її швидкодію з широко вживаною реалізацією.

Методи проєктування та розробки базуються на використанні кросплатформенного фреймворку Qt. Для порівняння використовується система gRPC.

EMBEDDED SYSTEM, INTER-PROCESS COMMUNICATION, COMMUNICATION METHOD, DATA EXCHANGE, RPC, GRPC, QT.

The aim of the work is to study the methods of interaction between processes in the development of embedded systems.

Research of interprocessor communication methods has been carried out. A software system was developed that implements one of the methods of data exchange of the embedded system with the external system and compares its performance with the widely used implementation.

Design and development methods are based on the use of a cross-platform Qt framework. The gRPC system is used for performance comparison.

Я, Кабанцев Володимир Володимирович, студент гр. ІПЗздм-19-2, здобувач вищої освіти на другому (магістерському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Дослідження методів міжпроцесорної взаємодії для створення вбудованих систем», що буде представлена в екзаменаційну комісію для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElAr KhNURE. Всі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

ЗМІСТ

Вступ 7
1 Аналіз проблемної області та постановка задач 9
1.1 Аналіз проблемної області дослідження 9
1.2 Аналіз існуючих аналогів 15
1.3 Постановка задачі 17
2 Опис прийнятих проєктних рішень 19
2.2 Вибір цільових ОС та допоміжного інструментарію 26
2.3 Розробити алгоритму взаємодії на базі обраного методу 28
2.4 Аналіз та моделювання предметної області для прикладної системи 30
2.5 Розробка архітектури програмної системи 32
3 Опис програмної реалізації 35
4 Експериментальне дослідження міжпроцесорної взаємодії 43
Висновки 47
Перелік джерел посилання 48
Додаток А Перелік джерел посилання за науковими напрямами керівника та науковців кафедри програмної інженерії 50
Додаток Б Звіт результатів перевірки кваліфікаційної роботи на унікальність тексту 51
Додаток В Слайди презентації 52
Додаток Г Лістинг коду 60
Додаток Д Тези доповіді 69
Додаток Ж Експертний висновок нормоконтроль 70

ВСТУП

Все частіше різноманітні комп’ютеризовані системи стають частиною побуту або роботи. Велика частина з таких систем відносяться до вбудованих, які можуть бути складовою побутових приладів (так званих смарт пристроїв), так і спеціалізованими пристроями для дозвілля, роботи, лікування або доглядом за здоров’ям. З розвитком обчислювальних можливостей та запитом користувачів росте складність програмної складової таких систем.

Для спрощення проектування та для кращого функціонування часто використовуються системи з декількома процесами (або потоками), які виконуються паралельно постійно або якийсь проміжок часу. Також, зі спрощенням доступу до Інтернету, виникла проблема взаємодії вбудованих систем з іншими системами через мережу.

Ціль даної роботи - дослідження методів міжпроцесорної взаємодії для взаємодії, вибір та реалізація одного з методів для побудови вбудованих систем, та порівняння його швидкодії з широковживаним аналогом.

Об’єктом дослідження є взаємодія між процесами в вбудованих системах. Предмет дослідження — існуючі методи міжпроцесороної взаємодії в межах однієї системи або через мережу.

Теоретичною і методологічною основою роботи є праці вітчизняних і зарубіжних вчених, в яких розглядаються практичні і теоретичні питання в області розробки вбудованих систем (див. додаток А), Під час кваліфікаційної роботи було проведено аналіз методів міжпроцесорної взаємодії, побудовано архітектуру системи взаємодії двох процесів, розташованих на різних обчислювальних пристроях об’єднаних мережею та розроблено програмне забезпечення, яке реалізує один із методів взаємодії цих процесів. Також проведено експериментальне дослідження методів.

Отримані результати можуть бути використані при побудові вбудованих систем, складність яких потребує використання декількох процесів між якими

потрібно налаштувати ефективну взаємодію, або систем, які потребують взаємодії з процесами через мережу.

Робота перевірена на унікальність тексту в мережі інтернет та базі ХНУРЕ (див. додаток Б).

За результатами кваліфікаційної роботи магістра було розроблено презентацію (див. додаток В).

Лістинг частини коду розроблюваної системи наведено в додатку Г. Було створено тези доповіді на конференції «Сучасна молодь в світі ІТ». Тези доповіді наведено в додатку Д. Робота перевірена на відповідність вимогам оформлення (див. додаток Ж).

1 АНАЛІЗ ПРОБЛЕМНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧ

1.1 Аналіз проблемної області дослідження

Комп’ютери та різноманітні автоматизовані системи глибоко проникли в життя сучасної людини. Вони допомагають вирішувати різноманітні задачі з якими люди стикаються щодня. Комп’ютеризовані системи в тим чи іншим шляхом керують більшістю інформаційних потоків.

Програмне забезпечення для комунікації зблизило людей та дозволяє ефективно організовувати роботу в розподілених командах, покращити якість дистанційного навчання, спростити спілкування використовується в роботі, навчанні, дозвіллі.

Програмне забезпечення зараз стало частиною побуту, в тому числі як складова частина автомобілів, побутової техніки, приладів медичного напрямку. Саме в таких комп’ютеризована частина називається вбудованою.

Вбудовані системи - це комбінація програмного та апаратного забезпечення і, можливо, додаткових механічних чи інших частин, створене для виконання конкретного функціоналу, на відміну від систем призначених для виконання широкого кола завдань, таких як персональний комп’ютер [1].

Обмеження в функціоналу дозволяє оптимізувати систему для зменшення вартості продукту, його розміру, збільшення надійності та швидкодії. В багатьох випадках ці параметри можуть стати вирішальними для його успіху на ринку.

Вбудовані системи широко використовуються в споживчій галузі, промисловості, медичній сфері, телекомунікаціях, автомобілебудуванні та військовій галузі [2].

В споживчій галузі це починаючі від мобільних телефонів та ігрових консолей до мікрохвильовок та пральних машин з електронним керуванням. Вбудовані системи допомагають побудувати систему з розумним термостатом, що дозволяє більш економно та точніше керувати температурою. Домашні системи автоматизації в яких можна керувати світлом, температурою системами безпеки

використовують вбудовані системи як сенсори та для управління.

Телекомунікаційна галузь використовує численну кількість вбудованих систем - від автоматичних телефонних станцій (АТС) до мобільних телефонів. Комп'ютерні мережі використовують маршрутизатори та мережеві мости для маршрутизації даних.

Розвиток автомобілебудування призвів до широкого використання вбудованих систем як складової частини транспортного засобу [3]. Окрім інформаційної функції, яку може давати система використовуючи GPS приймач, вона також може збільшувати безпечність транспортного засобу через додаткові системи активної безпеки - антиблокувальної системи (АБС), антибуксувальну систему, електронний контроль стійкості, автоматичної системи повного приводу. В різних електромобілях використовуються контролери моторів. З розвитком автопілоту також збільшується кількість вбудованих систем, які використовуються в автомобілі.

В медичній галузі вбудовані системи використовуються для моніторингу стану пацієнта, автоматична взаємодія з пацієнтом в разі необхідності - введення ліків, електричний імпульс в разі порушення серцевої діяльності, проведення закритого масажу серця, тощо [4].

Процес розробки вбудованих систем складається з таких етапів [5]:

– вибір апаратної складової - електроніка для виконання ПЗ, зовнішні інтерфейси, пристрої вводу та виводу інформації;
– розробка архітектури системи;
– вибір операційної системи;
– вибір платформи для розробки - мова програмування, інструменти розробки;
– розробка апаратної складової - компонування електроніки та тестування взаємодії її елементів;
– розробка ПЗ;
– верифікація та валідація функціоналу.

В деяких випадках вдається проводити розробку апаратної та програмної

складових одночасно. Для цього використовуються емулятори, систем які частково реалізують кінцеву платформу або на ще не готовий апаратній частині з емуляцією частин, які ще в розробці.

Для розробки вбудованих систем використовується багато різних методів розробки які можуть призвести до різної якості кінцевого ПЗ.

Часто для розробки таких систем використовується модельно-орієнтований підхід, коли система розбивається на предметно-орієнтовані моделі. Моделі абстрагують якусь частину ПЗ та дозволяють зменшити складність та зв'язність програмного продукту в цілому [6]. Перевикористання стандартизованих моделей дозволяє підвищити продуктивність розробки та спростити процес проектування та сприяє взаємодії між розробниками і командами (через стандартизацію термінологію та використання найкращих практик з різних предметних областей).

Завдяки поширенню методологій гнучкої розробки, для створення вбудованих систем також почали використовуються такі підходи [7]. Такі методології дозволяють більш повно задовольнити замовника за рахунок постійної з ним. Внесення змін в вимоги під час розробки дозволяє створити більш конкурентоспроможний продукт, використовувати нові технології, що також може позитивно відобразитись на якості кінцевого продукту. Також в системах орієнтованих на взаємодію з користувачем, таких як мультимедійна складова в автомобілі або окрема мультимедіа система можлива розробка з використанням методології BDD, коли сценарії користування можна використовувати як тести, керуючись якими і ведеться розробка [8].

Процес розробки вбудованих систем може бути дуже складним, в цьому випадку треба враховувати багато факторів - розуміння необхідного функціоналу; коректна побудова абстракцій над різними елементами ПЗ; правильний розподіл ресурсів, який дуже важливий в операційних реального часу; взаємодія процесів, які відповідають за обробку інформації від сенсорів або які відповідають за відображення або передачу даних.

Вибір способу взаємодії між процесами - це дуже важливий етап розробки вбудованих систем. Дуже часто вбудовані системи дуже обмежені в

обчислювальних потужностях, або мають їх розподіляти між різними процесами для нормального функціонування.

Під час роботи, процес в вбудованій системі може бути необхідно обмінюватись інформацією з іншими процесами як в рамках однієї ЕОМ (або однієї ОС для багатомашинних систем), так і на різних ЕОМ, пов’язаних між собою мережею [9]. Також є варіант, коли процес взаємодіє сам з собою - в такому випадку відбувається синхронізація або обмін даними між окремими потоками процесу.

Для зв’язку з зовнішними пристроями вбудовані системи можуть використовувати різні фізичні протоколи обміну даними орієнтованих на системи зі зниженим споживанням енергії або для роботи зі слабким сигналом тощо [10]. В даній роботі такі специфічні протоколи не розглядаються і огляд сконцентрований на програмній складовій.

Такий процес може виконуватись на різних операційних системах. Однією з задач магістерської роботи є огляд обміну між процесами, який можливий на переважній більшості операційних систем, оскільки це забезпечить універсальність результатів.

Всі методи для взаємодії між процесами можна розділити на 2 групи – методи для взаємодії в межах однієї ЕОМ та методи взаємодії між процесами, які знаходяться на різних ЕОМ.

Для взаємодії процесів в рамках однієї ЕОМ існують такі методи:

– файл – інформація записується в файл, до якого мають доступ інші процеси. Можливий варіант роботи з файлом який знаходиться в оперативній пам’яті;

– сегменти подільної пам'яті — засіб, що дозволяє декільком процесам сумісно використовувати (поділяти) фрагмент оперативної пам'яті з метою обміну даними; відправлення даних відбувається шляхом запису в пам'ять, отримання — читанням з пам'яті.

– черги повідомлень — пакети даних, що передаються між процесами з повідомленням отримувача про надходження пакету;

– сигнали — асинхронні повідомлення, що, зазвичай, не передають дані між процесами, а сповіщають про подію, на яку процес має відреагувати виконанням наперед визначеної дії (функції або команди залежно від використаних засобів програмування);

– неіменовані та іменовані канали передачі даних як синхронних (очікуваних) повідомлень; відправлення повідомлення відбувається подібно до операції запису в файл, отримання — подібно до читання даних з файлу, якщо канал порожній — процес, що очікує дані призупиняється до надходження даних в канал;

– локальні сокети – взаємодія відбувається через віртуальні сокети ядра;

– платформозалежний обмін повідомленнями – пакети даних що передаються через механізми які доступні в ОС або платформі, такі як D-Bus, Windows Communication Foundation.

Також до методів взаємодії процесів в межах однієї ЕОМ можна віднести семафори та м’ютекси.

В вбудованих системах часто використовуються ОС з так званими мікроядрами, в яких частина з вищезгаданих методів може бути недоступна.

Типова архітектура вбудованої системи представлена на рисунку 1.1.

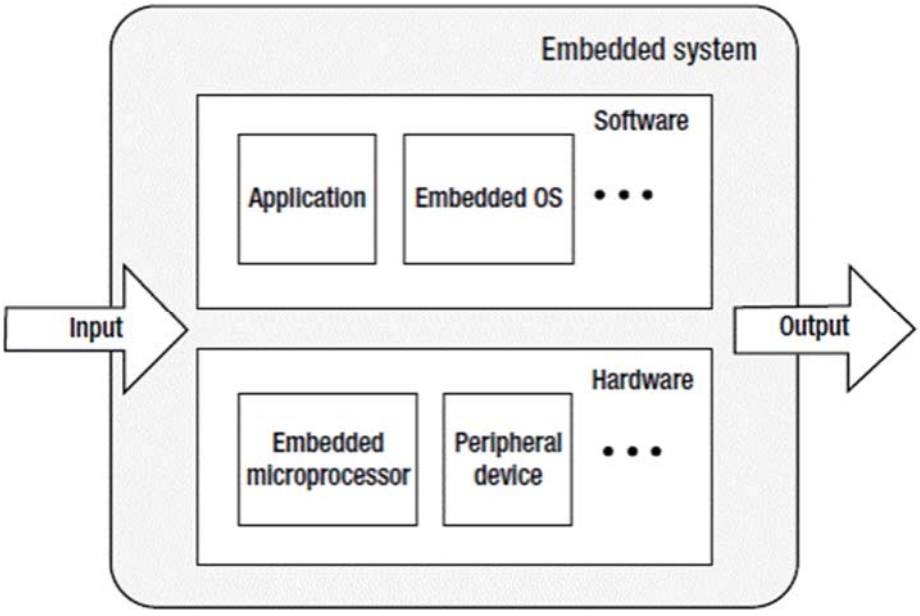

Рисунок 1.1 - Базова архітектура вбудованої системи

Така система включає в себе дві основні частини - програмну та апаратну. Апаратна частина - це процесор, пам’ять, шину передачі даних, периферійні пристрої, порти вводу/виводу та різноманітні контролери.

Програмна частина зазвичай включає спеціалізовану операційну систему, різноманітні застосунки та драйвери.

Існує декілька типів архітектури ПЗ які використовуються в розробці:

– простий цикл, в якому робляться виклики підпрограм, що виконують свою частину задачі;

– системи побудовані на перериваннях де задачі викликаються за допомогою подій. Такі події можуть генеруватися таймером із предустановленною частотою або контроллером при отриманні сигналів;

– системи сумісної багатозадачності, які схожі на простий цикл, з винятком що для додавання нового ПЗ до виконання необхідно додати його до черги інтерпретатора;

– системи витісняючої багатозадачності чи мультиядерні в яких частина низькорівневого коду перемикається між задачами або потомами і базується на таймері;

– мікроядра та екзоядра. Такі ядра розділяють пам’ять та перемикають потоки виконання, а користувацький шар процесів реалізує основні функції, такі як файлові системи, мережеві інтерфейси тощо [11];

– монолітні ядра, які створюють середовище аналогічне настільним операційним системам, таким як Лінукс або Microsoft Windows. Такі системи значно спрощують розробку програмного забезпечення але ціною збільшення вимог до апаратних ресурсів та можливого зменшення надійності.

– системи специфічні до предметної області, такі як AUTOSAR в автомобілебудування [12], або де є підвищені вимоги до безпеки та надійності недоступної в вищезгаданих архітектурах.

1.2 Аналіз існуючих аналогів

Для побудови масштабованих архітектур систем так званих інтернету речей часто використовується протокол DDS - data distribution services [13], наприклад в реалізації від RTI - RTI Connext DDS. Цей протокол орієнтований на дані, тобто кожне повідомлення включає контекстуальну інформацію, яка необхідна для застосунків щоб зрозуміти які дані вони отримують.

Схема обміну даними в середині протоколу представлена на рисунку 1.2.

Ця система дозволяє використовувати шаблон Спостерігач [14] для обміну даними - частина системи, яка збирає дані підписується на “тему” з даними які її цікавлять, а частина яка генерує - публікує дані під цією “темою”.

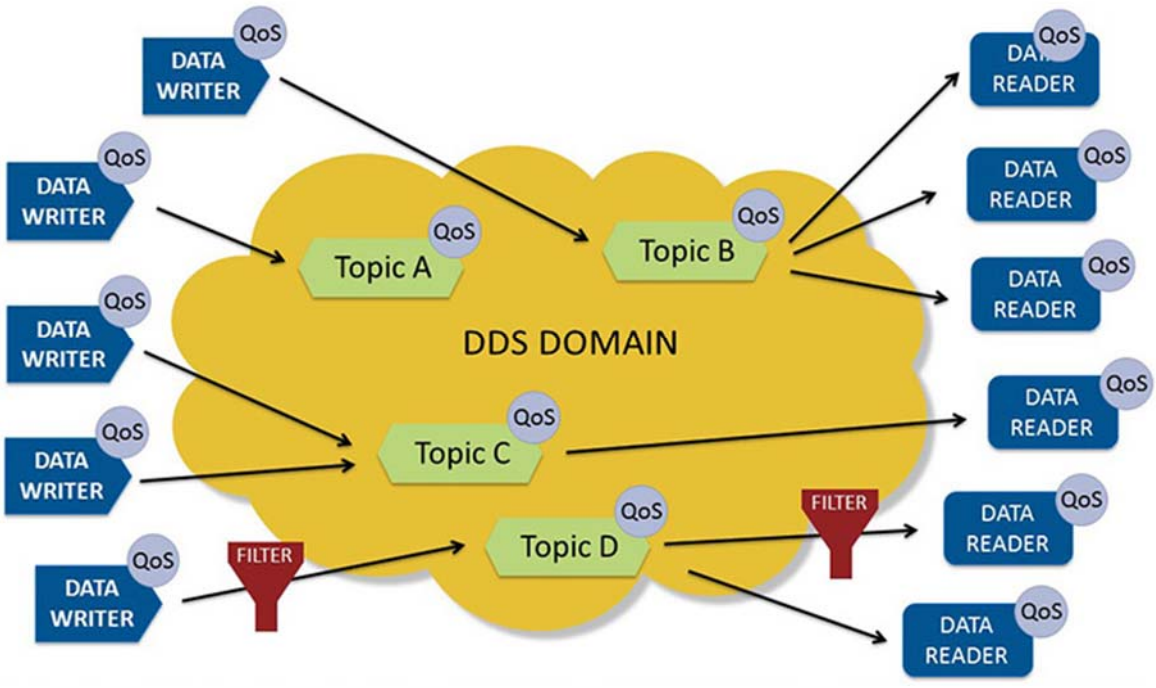

Рисунок 1.2 Схема передачі даних в DDS

Також цей протокол передбачає динамічне знаходження систем, які публікують і які підписані без необхідності в окремому сервері для їх з’єднання. Це дозволяє будувати гнучкі системи з простою архітектурою мережі. Але, для

використання цієї реалізації розробник повинен мати ліцензію а також платити пеню за кожен пристрій, який використовує цю реалізацію.

Але цілий ряд відомих постачальників інтернет сервісів використовують gRPC для взаємодії в своїх системах [15]. Це система для віддалених викликів з відкритим кодом, яку почала розробляти Google ще у 2015 році. Вона дозволяє виконувати віддалені виклики для систем написаних різними мовами програмування, яких підтримується більше 10. На рисунку 1.3 наведений приклад обміну повідомленнями між додатками написаних різними мовами програмування через gRPC.

Для опису інтерфейсів використовується IDL (interface description language) файли [16], з яких потім генерується реалізація для конкретної мови застосунку.

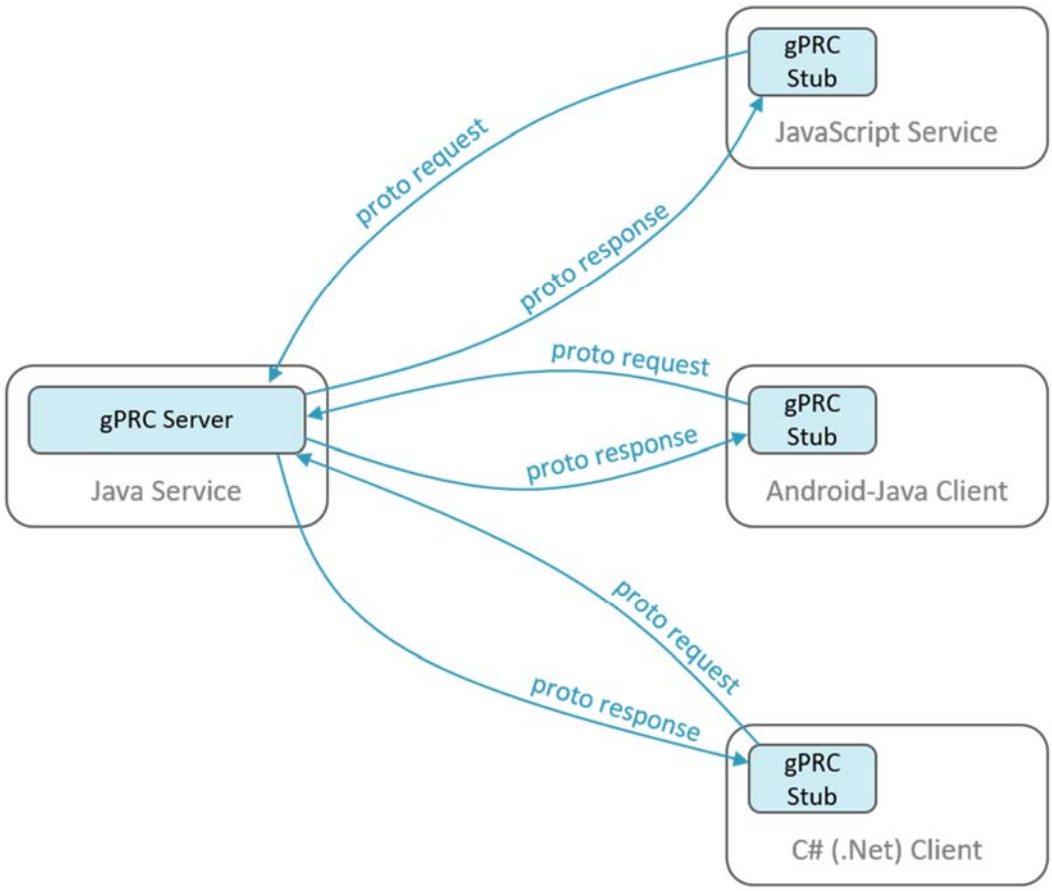

Рисунок. 1.3 - Взаємодія серверу і клієнтів з використанням gRPC

gRPC підтримує автентифікацію через токени та TLC [17], що дозволяє

побудувати безпечний канал передачі даних.

1.3 Постановка задачі

Метою роботи є дослідження методів міжпроцесорної взаємодії для створення вбудованих систем - їх типи, особливості застосування. Практичною складовою - є розробка системи для виконання віддалених процедур, яка б дозволяла взаємодіяти процесу на вбудованій системі з іншою вбудованою системою або з сервером, та порівняння її швидкодії з однією із широко відомих реалізацій систем для міжпроцесорної взаємодії. Система повинна бути виконана у вигляді двох застосунків, для клієнтської та серверної частини, які були б доступі для виконання на кількох популярних платформах.

На рисунку 1.4 представлена схематична взаємодія вбудованої системи з сервером.

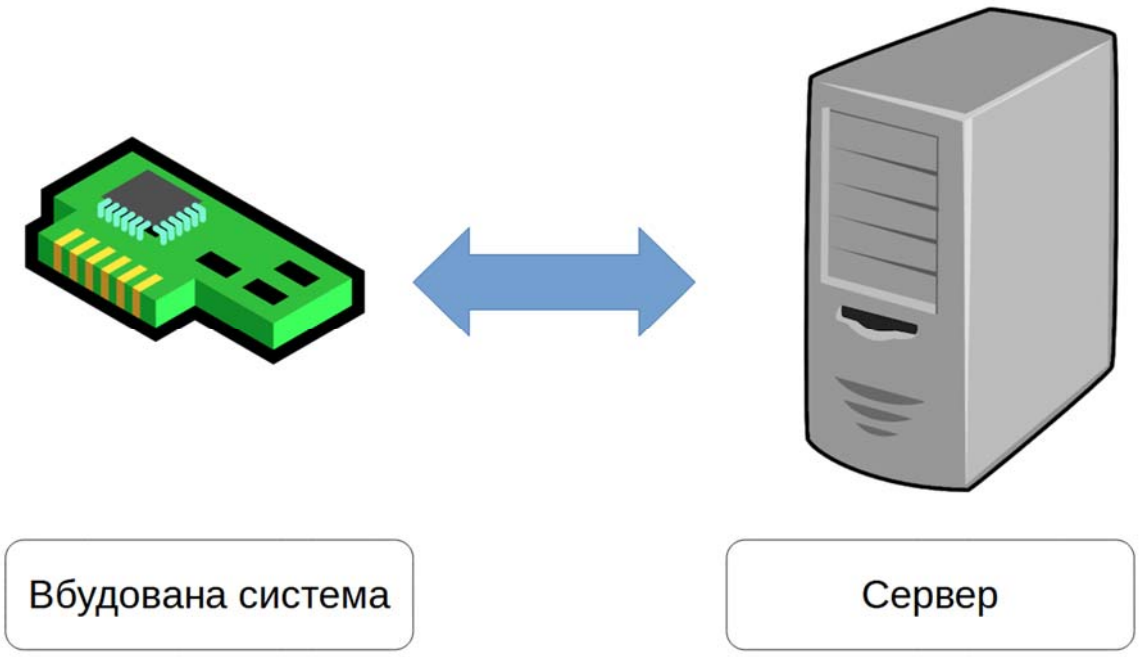

Рисунок 1.4 - Схематичне зображення взаємодії вбудованої системи з іншими системами

На початку роботи необхідно провести аналіз існуючої інформації про предметну область, визначити об’єкти, їх атрибути та зв’язки між об’єктами, клієнтів системи та їх вимоги.

Система повинна працювати стабільно на операційних системах які часто використовуються в побудові серверної частини так і на операційній системі які використовуються в вбудованих системах. З боку користувача, тобто розробника вбудованої системи або серверної частини, взаємодія через цей інструментарій повинна нагадувати взаємодію всередині системи – виклик методу або інший зрозумілий механізм.

Таким чином, можна поставити такі задачі:

– провести аналіз методів міжпроцесорної взаємодії та обрати метод для реалізації;

– вибрати цільові ОС, допоміжний інструментарій та розробити алгоритм на базі обраного методу;

– провести аналіз та моделювання предметної області для створення прикладної системи;

– спроектувати та програмно реалізувати вбудовану систему;

– розробити план проведення експериментального дослідження (обрати методи порівняння та критерії) та провести дослідження реалізованого методу та обраних аналогів;

– розробити рекомендації щодо використання методів міжпроцесорної взаємодії.

2 ОПИС ПРИЙНЯТИХ ПРОЄКТНИХ РІШЕНЬ

2.1 Аналіз методів міжпроцесорної взаємодії

Для взаємодії процесів в рамках однієї ЕОМ існують такі методи:

– взаємодія через семафори;
– взаємодія з використанням м’ютексів.
– взаємодія через файлову систему;
– взаємодія через сегменти подільної пам'яті;
– сигнали;
– передача даних через неіменовані та іменовані канали;
– взаємодія через локальні та нелокальні сокети;
– платформозалежний обмін повідомленнями.

Семафори, так само як і вироджена його версія м’ютекс, являється одним із найстаріших методів управлінням доступом процесів, що виконуються паралельно. Семафор являє собою об’єкт ядра операційної системи, який має вигляд лічильника в діапазоні від 0 до його максимального значення заданого при створенні. Від дозволяє здійснити над собою 2 операції:

– захоплення ресурсу, при цьому значення лічильника зменшується на одиницю, якщо він не нульовий. В іншому випадку процес блокується, допоки не звільниться хоча б один ресурс.
– звільнення ресурсу – значення лічильника збільшується на одиницю.

Реалізація м’ютекса, крім сталого значення лічильника - 0 або 1, також тримає в собі поняття володіння - звільнити м’ютекс може тільки процес який його захопив. Використання м’ютексів дозволяє організовувати безпечну роботу з областю пам’яті яка доступна декільком процесам.

Використовуючи файлову систему, процеси можуть передавати дані, розмір яких може змінюватись. Але, зазвичай, запис в файл може робити одночасно тільки один процес. Також може виникнути проблема цілісності зчитаних даних, якщо файл був перезаписаний під час читання іншим процесом. Тому, після

запису в файл процес інформує, що файл було записано і вже після цього клієнтські процеси можуть його зчитувати. Також файли можуть використовуватись для зберігання стану процесу задля його відновлення при наступному запуску.

Швидкодія файлової системи також впливає на взаємодію між процесами – треба брати до уваги, що запис та читання файлу займає набагато більше часу аніж робота з оперативною пам’яттю.

В вбудованих системах через обмеження по розміру та низьке споживання енергії використовуються твердотільні накопичувачі, в яких неконтрольоване активне використання файлової системи на запис може призвести до швидкої деградації, що може вилитись в пошкоджені дані на файловій системі.

Щоб обійти обмеження по швидкодії та зменшити вплив на накопичувач, можна використовувати файл, що відображається в оперативну пам’ять.

Спільна пам’ять. Спільною (або подільною) пам'яттю називають регіон комп'ютерної пам’яті, доступ до якої має декілька процесів одночасно. Потоки можуть ефективно обмінюватись даними, так як дані не копіюються додатково. Для цього один процес створює виділяє область в оперативній пам’яті, до якого можуть мати доступ інші процеси.

Комбінуючи використання спільної пам’яті з іншими методами взаємодії, такими як обмін повідомленнями або сигнали можна побудувати ефективну систему взаємодії де процеси мають менше обмежень в розмірі переданих даних без зменшення швидкодії.

Також використовуючи поведінковий шаблон Blackboard (дошка) [16] на базі спільної пам’яті можна створити надійну систему обміну даними, яка зменшить зв’язність компонентів та спростить додавання нових. Структура шаблону зображена на рисунку 2.1.

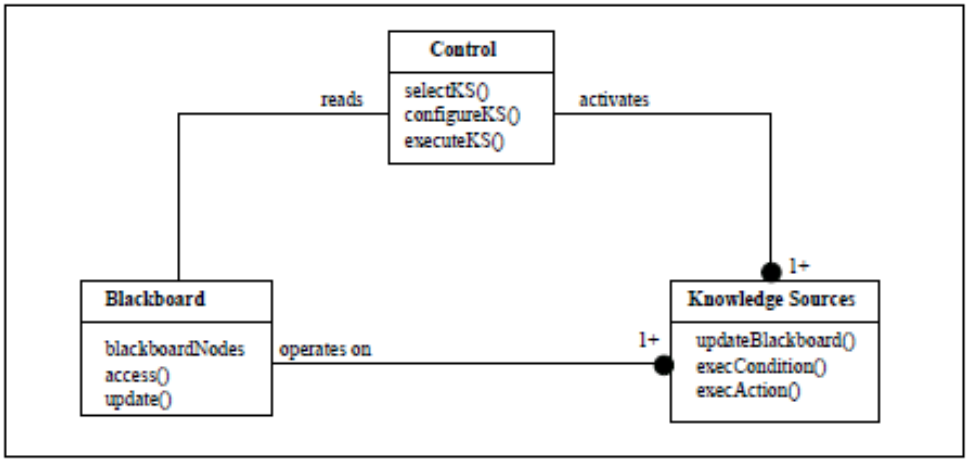

Рисунок 2.1 Взаємодія основних елементів в шаблоні Blackboard

Сигнали, як метод взаємодії, мають доволі обмежену сферу використання через малий розмір інформації що передається. Вони можуть використовуватись для сповіщення про якусь подію, наприклад про оновлення даних в доступному для отримувача місці – файлі, пам’яті або сокеті.

Іменовані канали мають подібну до файлів поведінку. Це односторонній спосіб взаємодії, при якому дані передаються за принципом FIFO – в якому порядку вони були записані, в такому вони і можуть бути зчитані іншим процесом без використання проміжного фалу. Для користування каналом потрібно знати його ім’я та відповідні права для його читання або запису. Використовуючи два різнонаправлені канали два процеси можуть налагодити повноцінний обмін даними.

До неіменованих каналів можна віднести стандартні потоки введення та виведення – програма, яка запускає іншу, створює неіменований канали, які наслідує створений процес. Також можлива ситуація коли декілька запущених процесів з’єднуються такими каналами і створюється конвеєр програмного забезпечення – кожний наступний процес обробляє дані отримані через неіменований канал від попереднього.

Зазвичай канали є частиною операційної системи та створюються в оперативній пам’яті.

Локальні та нелокальні сокети – це програмні інтерфейси для обміну даними

які є абстрактними об’єктами що представляють кінцеві точки з’єднання.

Локальні, або Unix, сокети відрізаються від нелокальних тим, що передача даних відбувається в рамках однієї системи без використання мережевого інтерфейсу і тільки за рахунок операцій ядра системи. Локальні сокети є частиною POSIX сумісної системи.

Сокети можуть бути клієнтськими або серверними – серверні очікують на з’єднання, а клієнтські його ініціюють. Після з’єднання сокети можуть використовуватись для обміну даними.

Типи сокетів залежать від протоколу, який використовується для транспорту даних:

– потокові сокети, в яких використовується протокол TCP, SCTP чи DCCP, які забезпечують послідовну (окрім як у випадку DCCP) передачу даних без обмеження в розмірах запису та без пошкодження даних

– датаграм сокети які використовують протокол UDP і для яких не потрібно з’єднання. Порядок та надійність не гарантовані в даному випадку, тому декілька пакетів відправлених з одного такого сокету можуть прибути у будь якому порядку або не прибути зовсім. Також можлива широкомовна передача – в цьому випадку, щоб отримати пакети приймаючий сокет не повинен бути пов’язаний з конкретним адресом, але це може залежати від конкретної реалізації

– сокети сирих даних які дозволяють відсилати пакети даних без автоматичного додавання протоколозалежного шару даних.

Обмін повідомленнями – використання черг повідомлень через канали які можуть контролюватись ОС. До таких систем можна віднести RPC, парадигму MPI, DDS та інші.

Якщо сокети є частиною більшості операційних систем, то для обміну повідомленнями недетермінованого розміру часто потрібні додаткові сервіси. Також є платформозалежні системи для передачі повідомлень, такі як D-Bus, Mailslots, MSMQ.

Деякі ознаки вище приведених методів можуть відрізнятись залежно від платформи. Порівняння, приведене на таблиці 2.1 базується на тому що цільовою системою є Raspberry Pi 4.

Таблиця 2.1 Порівняння методів взаємодії

Метод взаємодії	Швидкодія	Зручність	Основні недоліки
Семафори та м’ютекси	Висока	Синхронізація роботи потоків, контроль доступу	Неможливість передачі даних
Файлова система	Низька	Доступний великий обсяг, Погано підходить для ефективної взаємодії	Відносно повільний запис та читання
Спільна пам’ять	Висока	Можливість процесам працювати з одними змінними	Невеликий обсяг, необхідний непередбачуваний результат при одночасному записі в одну область
Сигнали	Висока	Проста синхронізація роботи	Неможливість передачі даних або дуже обмежений їх обсяг
Канали	Середня	Проста передача даних	Необхідний протокол для більш складної взаємодії
Сокети	Середня-низька	Проста передача даних, передача даних через мережу	Необхідний протокол для більш складної взаємодії
Обмін повідомленнями	Середня-низька	Гарна інкапсуляція, синхронізація роботи, передачі даних	Потрібно слідкувати за розміром повідомлення та уникати заповнення черг читання/запису

Найбільш зручним для ефективної взаємодії є обмін повідомленнями, який дозволяє організувати роботу багатьох процесів з низькою зв’язністю. Розглянемо цей метода детальніше.

POSIX сумісна системи мають вбудований інтерфейс для створення каналів обміну повідомленнями, але для створення каналу треба вказати розмір повідомлення та розмір черги, що ускладнює ефективний обмін даними розмір яких може сильно відрізнятись. Але в комбінації з іншими методами, такими як

сумісна пам’ять, або якщо всі користувачі каналу є частиною одної програми і тому мають сумісну пам’ять, можна значно зменшити розмір повідомлення та зменшити кількість копіювань при передачі даних.

Обмін повідомленнями може відбуватись в синхронному або асинхронному режимі.

При синхронному обміну процес, який відіслав повідомлення, буде заблокований, поки не отримає відповідь. Такий може бути неприйнятним, так як блокує основний процес і може, в деяких випадках навіть призвести до взаємного блокування процесів.

При асинхронному обміні отримувач може бути зайнятий, коли йому відправляють повідомлення. Відправник не заблокований після відправлення повідомлення, а саме повідомлення відправляється в чергу вхідних повідомлень отримувача. Після того як отримувач опрацює повідомлення він відправляє відповідь в чергу відповідей відправника. Такий метод обміну повідомленнями потребує додаткової пам’яті для черг повідомлень. Також можуть виникнути проблеми, якщо розмір виділеної пам’яті недостатньо для збереження всіх повідомлень, що може призвести до блоку відправника або втрати повідомлень. В першому випадку, коли відправник заблокований, може відбутися взаємне блокування. В другому втрата повідомлень приводе до ненадійної комунікації.

Синхронний обмін може бути побудований поверх асинхронного за допомогою алгоритму синхронізації. З іншої сторони асинхронна комунікація може бути побудована поверх синхронного обміну повідомленнями за допомогою допоміжних потоків.

Високорівневий обмін повідомленнями, такий як RPC допомагає приховати (інкапсулювати) транспортний рівень та зосередитись на даних які потрібно передати. RPC дає можливість процесу запущеному на одній ЕОМ звертатись до функцій (процедур) процесу що виконується на іншій ЕОМ, подібно до того, як процес звертається до власних локальних функцій. Це одна із форма клієнт-серверної взаємодії. Схема клієнт серверної взаємодії представлена на рисунку 2.2.

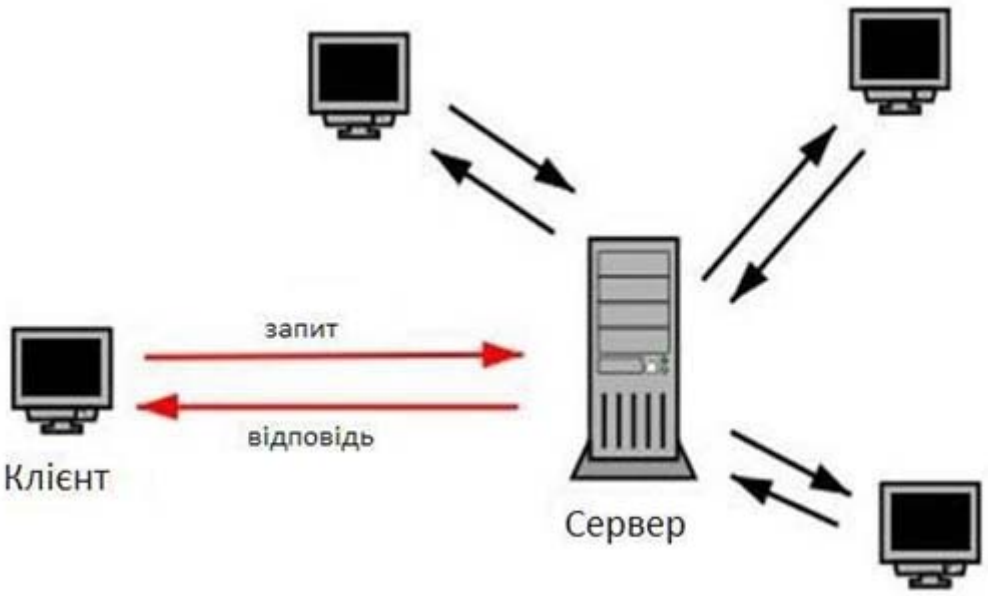

Рисунок 2.2 – Спрощена схема взаємодії між клієнтом та сервером

Завдяки таким методологіям як RPC, розробнику не потрібно хвилюватись про сам процес передачі запитів та відповідей. Наприклад у D-Bus є інтеграція с фреймворком Qt, завдяки якій можна користуватися передачею повідомлень між процесами через стандартну для фреймворку систему сигналів-слотів. Схема взаємодії процесів через D-Bus представлена на рисунку 2.3.

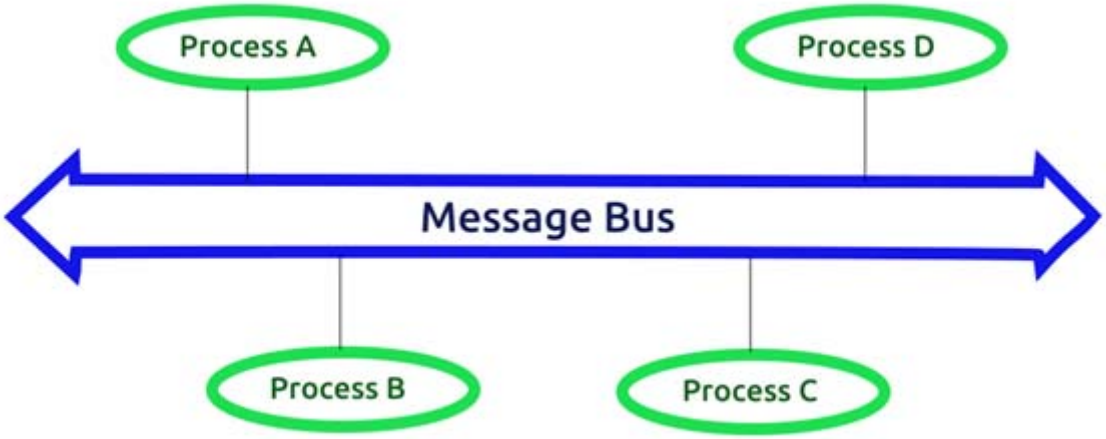

Рисунок 2.3 – Схема взаємодії між процесами через сервіс D-Bus

В парадигмі об’єктно орієнтованого програмування RPC являє собою RMI, тобто виклик віддалених методів. Модель RPC передбачає рівень локальної

прозорості, щоб виклик локальних процедур був таким які і виклику віддалених але з можливістю розділити їх, так як віддалені процедури зазвичай повільніше.

Важлива різниця між викликом локальної процедури і віддаленої це те що віддалений виклик може не вдатись через непередбачувані проблеми з мережею. Також важливо пам’ятати, що через вищезгадані проблеми процес який ініціював виконання віддаленої процедури може не дізнатись, чи була виконана процедура. На це треба зважати, якщо цільова процедура не являється ідемпотентною, тобто змінює стан системи, на якій виконується.

RPC це протокол запиту-відповіді. Виклик RPC ініціюється клієнтом, який відправляє повідомлення запиту конкретному віддаленому серверу на виконання визначеної процедури з переданими параметрами. Сервер повідомляє клієнта про отримання повідомлення та обробляє запит – виконує процедуру. Результат виконання відправляється клієнту у відповідь. На рисунку 2.4 представлено механізм виклику віддаленої процедури.

Реалізація взаємодії методом обміну повідомлень за парадигмою RPC завдяки гарній інкапсуляції та простому використанню може спростити створення розподілених вбудованих систем.

Отже для подальшої реалізації було обрано метод взаємодії через виклик віддаленого методу з передачею даних через мережеві сокети.

2.2 Вибір цільових ОС та допоміжного інструментарію

Більшість вбудованих систем використовують ОС побудовані на базі ядра Linux [17]. Одними із найвідоміших представників таких систем є Ubuntu, яку створено на базі вільного та відкритого програмного забезпечення, що дозволяє його використання як в навчальних так і комерційних цілях.

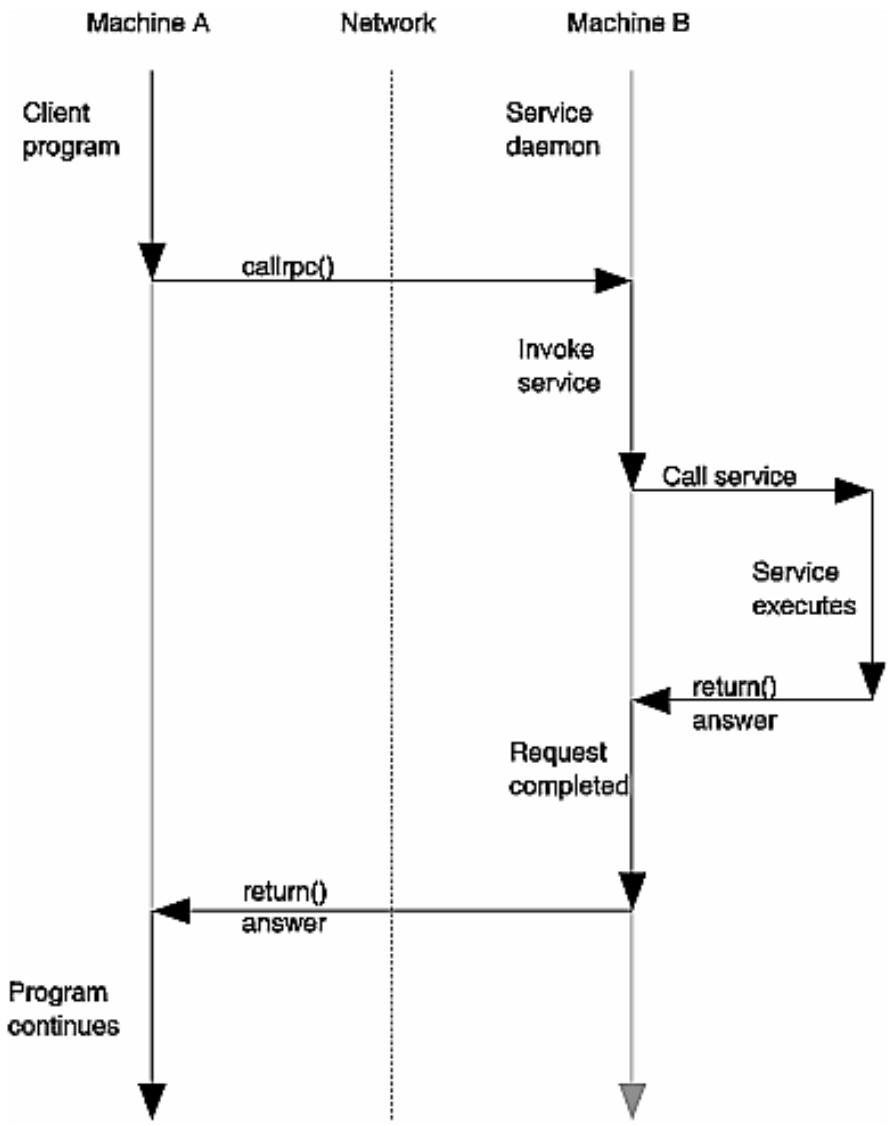

Рисунок 2.4 Механізм виклику віддаленої процедури

Так як комунікація потребує декількох систем в якості ОС для серверної частини також візьмемо Ubuntu, яка широковживана і в сегменті серверних систем.

Для побудови вбудованих систем часто використовується C++ який дозволяє більш чітко організувати вихідних код без суттєвого зменшення швидкодії кінцевої програми.

Щоб спростити реалізацію, зменшити кількість помилок та не зважати на різницю між системами використаємо кроспратформений фреймворк Qt, який дозволить використовувати один і той же вихідний код для компіляції на вихідних платформах. Qt підтримує компіляцію на цілому наборі відомих ОС та

використовується для створення вбудованих систем як на базі Linux так і QNX.

2.3 Розробити алгоритму взаємодії на базі обраного методу

Для розробки алгоритму обміну повідомленнями через виклик віддаленого методу треба виділити 2 частини – алгоритм для виклику віддаленої процедури та алгоритм обробки запитів на виконання, тобто відокремити клієнтську та серверну частини.

Так як взаємодія відбувається через мережевий інтерфейс, то один із перших кроків – встановлення адреси, яка буде вхідною точкою для запитів.

Клієнтська частина може мати як синхронну поведінку, коли виклик віддаленої процедури блокує процес, або асинхронну – процес не блокується а результат виконання доступний згодом як повідомлення або статус.

Синхронний алгоритм, який окрім того що краще відповідає поведінці виклику процедури, з використанням допоміжного потоку, може бути легко перетворений в асинхронний, тому зупинимось на ньому.

В розроблюваному алгоритмі можна виділити декілька основних стадій: серіалізація даних, транспортування серіалізованих даних, їх десереалізація, виклик методу. Після виклику результат виконання проходить проходить такий же шлях в кінці якого він повертається клієту

Алгоритм виклику віддаленої процедури зображено на рисунку 2.5

В загальному випадку алгоритм для серверної частини матиме вигляд як представлено на рисунку 2.6. Для випадку передачі великої кількості даних для кращої швидкодії має сенс додавати перед серіалізованими даними їх розмір, щоб серверна частина мала змогу визначити коли всі необхідні дані отримані.

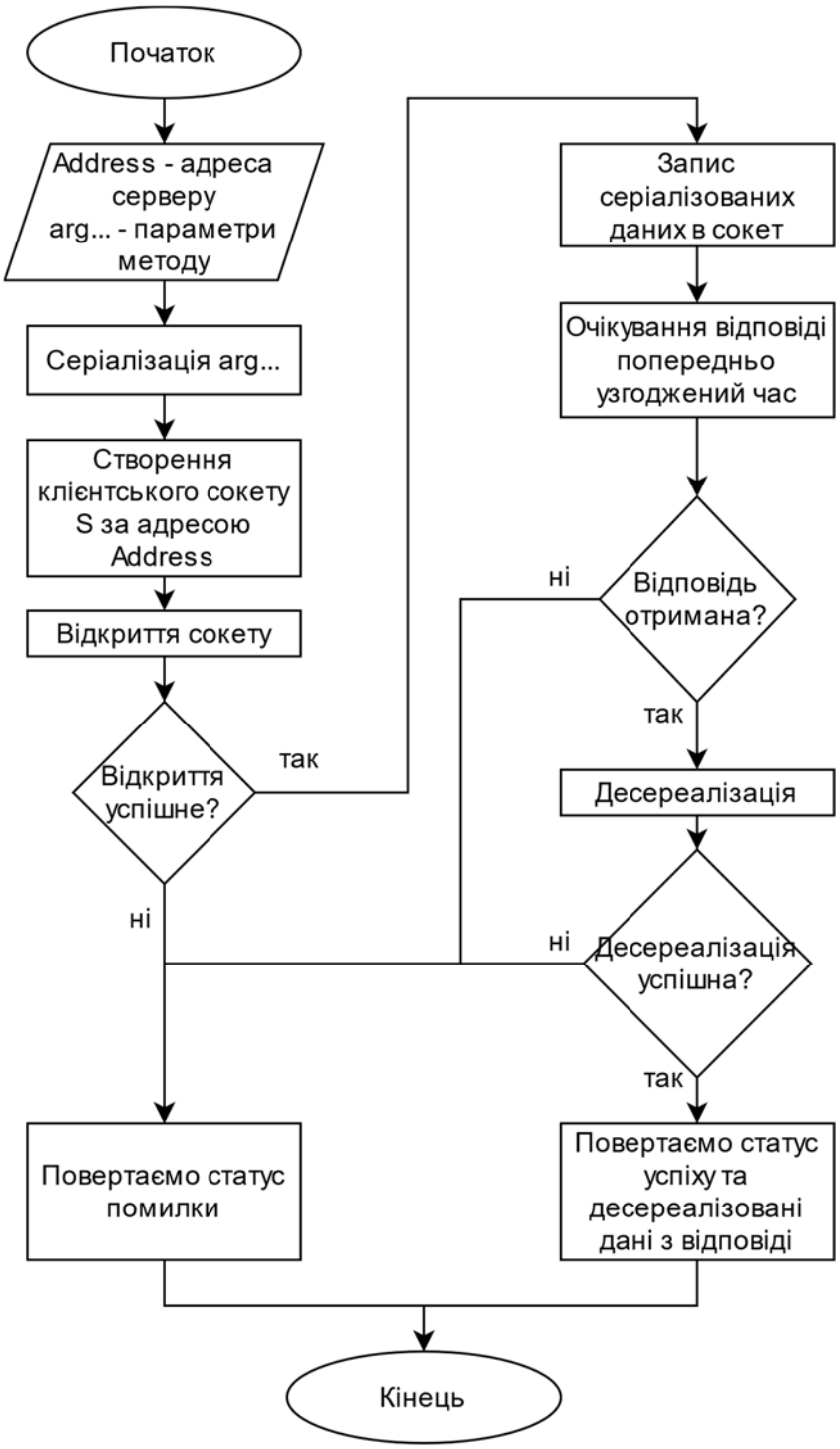

Рисунок 2.5 Алгоритм виклику віддаленої процедури

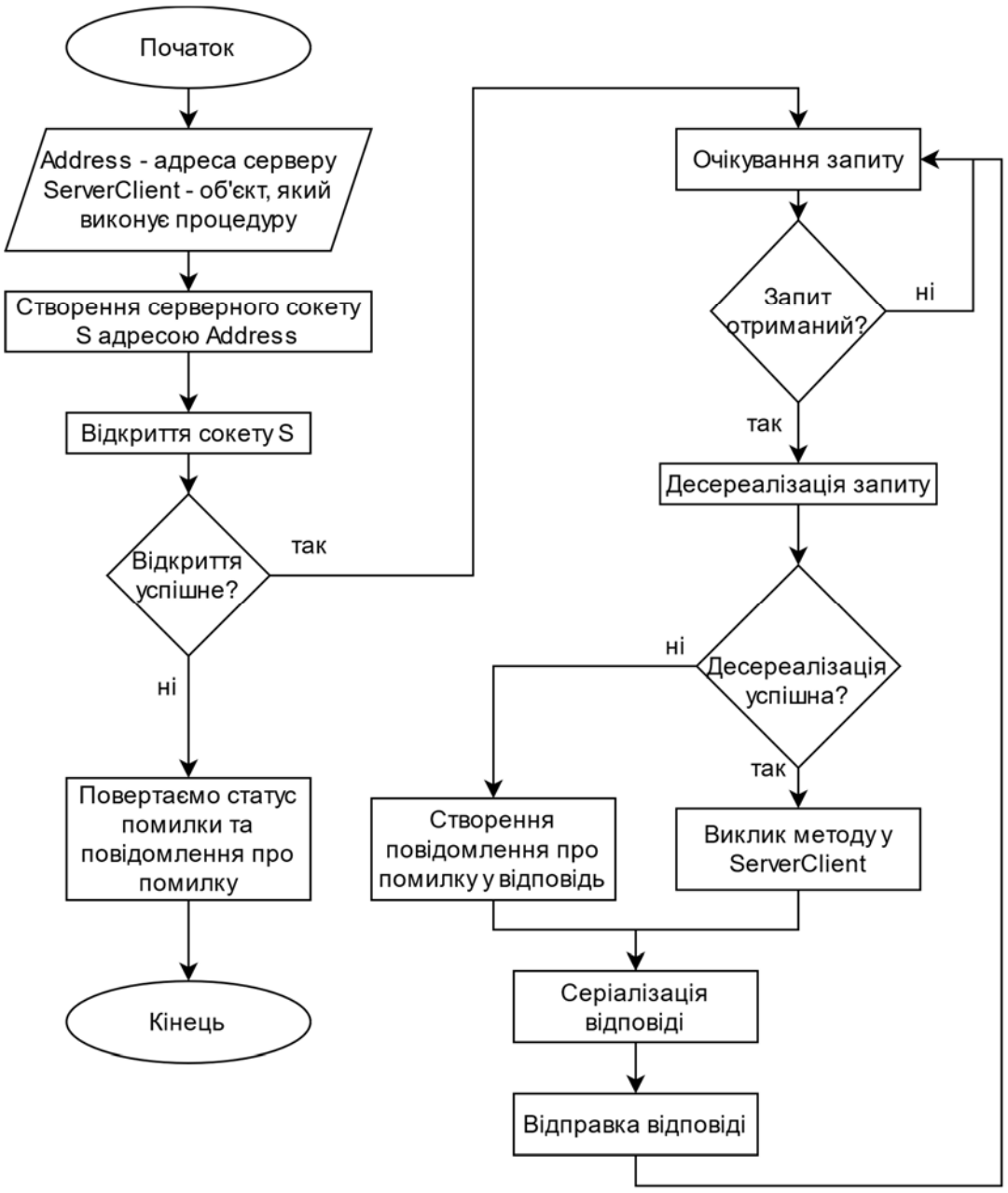

Рисунок 2.6 Алгоритм обробки запиту на виклик віддаленої процедури

Завершення роботи серверної частини може відбутись також через закриття сокету, яке для спрощення не було відображене в алгоритмі.

2.4 Аналіз та моделювання предметної області для прикладної системи

Однією з поширених областей використання вбудованих систем є медицина.

Вбудовані системи можуть застосовуватись для отримання та попереднього аналізу даних пов’язаних зі станом пацієнта — життєві показники, медичні зображення, інші показники сенсорів; автоматичної або напівавтоматичної взаємодії — введення медикаментів за необхідністю, проведення специфічних маніпуляцій, як то закритий масаж серця, відновлення серцевого ритму за допомогою електричних імпульсів тощо; моніторинг одного або декількох медичних параметрів — серцевої активності, пульсоксиметрія, моніторинг мозкової діяльності та інші.

Розглянемо приклад системи для моніторингу за життєвими показниками. В таких системах важливу роль може відіграє взаємодія з зовнішніми системами, через яку вона передає зібрані дані для подальшого аналізу.

За допомогою діаграми Use Case можна прослідкувати сценарій використання системи. Побудовану діаграму представлено на рисунку 2.7.

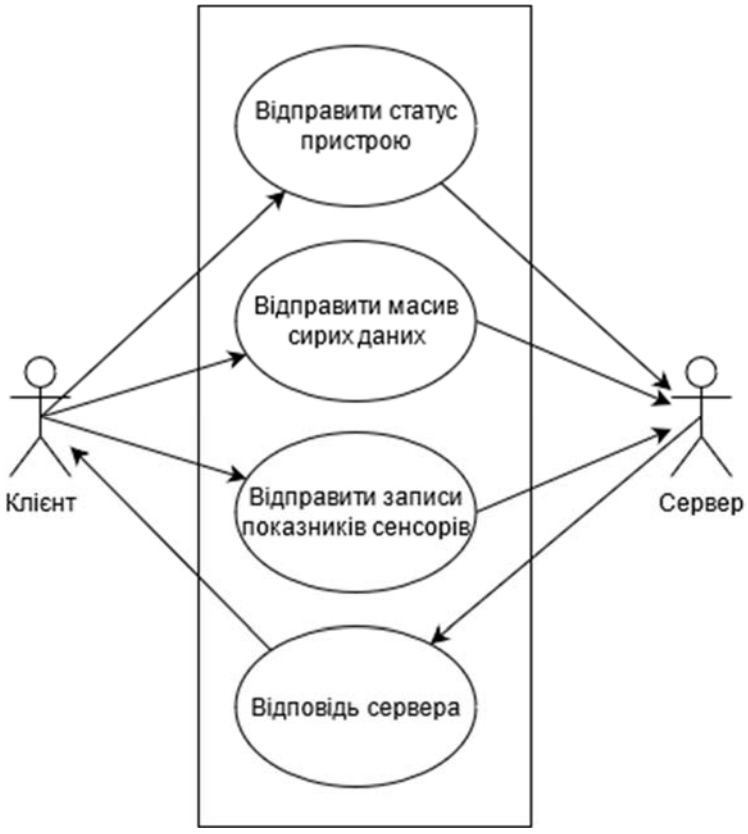

Рисунок 2.7 – Діаграма Use-Case

На даній діаграмі в якості клієнт виступає частина програмної системи, яка

потребує використання міжпроцесорної взаємоді. З неї можна виділити основні сутності, які задіяні при функціонуванні частини системи, яка розглядається в даній роботі.

2.5 Розробка архітектури програмної системи

Система для взаємодії між процесами, які знаходяться на різних обчислювальних система складається з двох частин – клієнтської, та серверної. Взаємодія між ними зображена на рисунку 2.8.

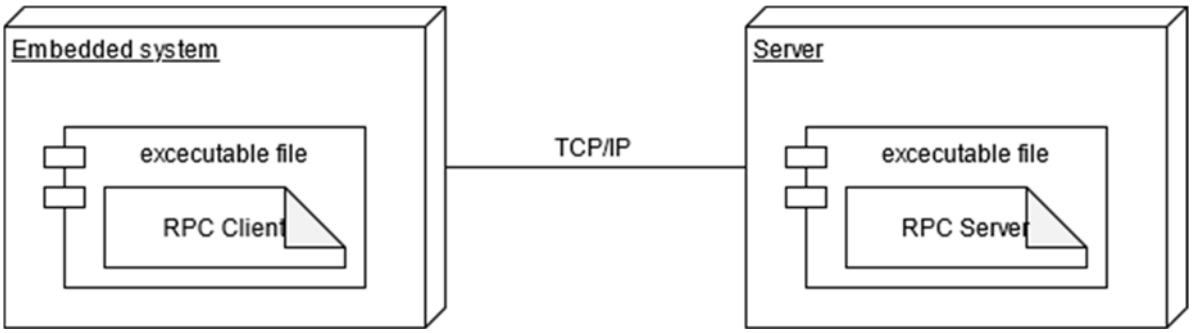

Рисунок 2.8 – Діаграма клієнт – серверної взаємодії

На цій діаграмі відображено основні обчислювальні вузли, що використовуються при роботі системи, та компоненти, що виконуються на них.

Обидві частини являють собою консольні додатки, написані за мові програмування C++ з використанням Qt. Частина коду, відповідальна за взаємодію між додатками спільна для обох додатків, та використовується через набір інтерфейсів, відображених на рисунку 2.9.

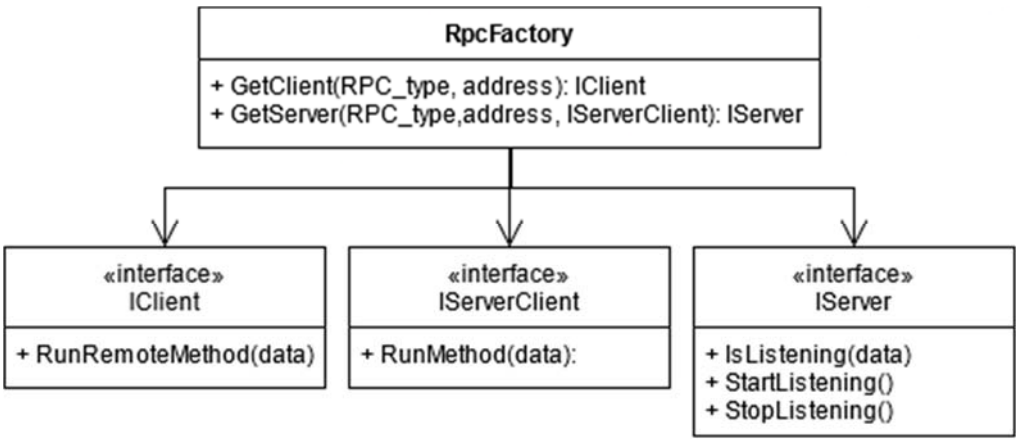

Рисунок 2.9 – Інтерфейси для міжпроцесорної взаємодії

Завдяки цим інтерфейсам, можна легко виокремити рівень який відповідальний за серіалізацію та транспорт даних та вносити в нього зміни без впливу на користувача інтерфейсів.

Використовуючи таких ці інтерфейси можна побудувати діаграму класів для взаємодії на базі TCP сокетів. Ця діаграма зображена на рисунку 2.10.

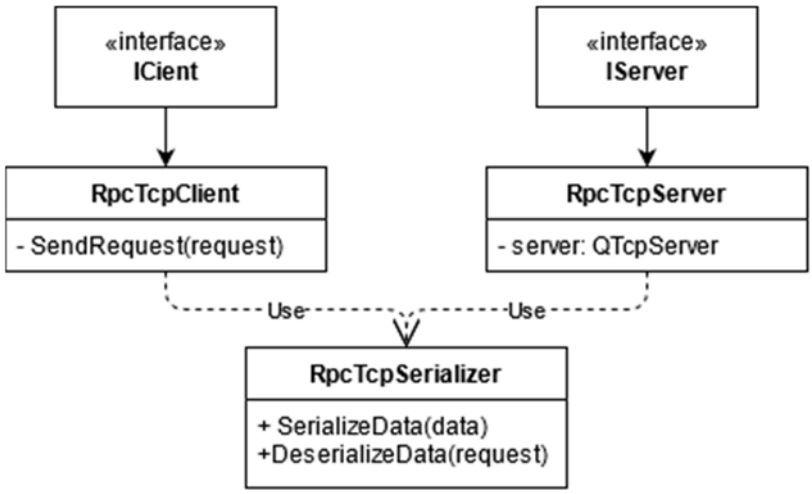

Рисунок 2.10 – Діаграма класів для RPC з використанням сокетів TCP

Для очікування вхідних з’єднань використовується Qt TCP сервер, який дозволяє спростити роботу з сокетами.

При реалізації взаємодії з використанням gRPC структура спрощується, так як можна використовувати вбудований в gRPC механізм маршалінгу-демаршалінгу – protobuf. Діаграма класів взаємодії представлена на рисунку 2.11.

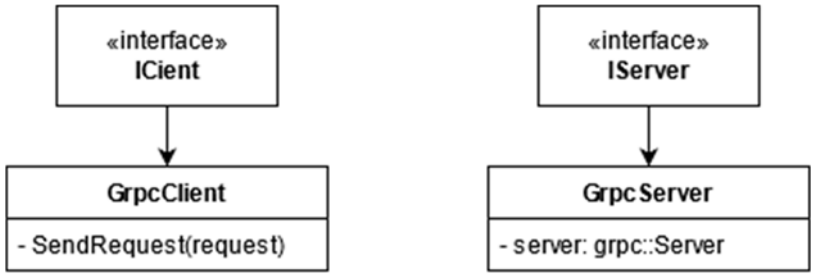

Рисунок 2.11 – Діаграма класів для RPC з використанням gRPC

На базі розробленої архітектури буда створено систему взаємодії процесів які можуть бути розташовані на різних платформах.

3 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

3.1 Вибір засобів розробки

Під час вибору засобів для написання розробленої системи потрібно враховувати наскільки вони актуальні в даній сфері, простоту в підтримці та доречності їх використання для даної розробки. Важливо пам’ятати, що однією з особливостей вбудованих систем є, зазвичай, обмежена обчислювальна спроможність через можливу мобільність девайсу, обмеженнях в фізичних параметрах та можливе живлення від батареї. Такі обмеження призводять до використання компільованих мов програмування, які не потребують додаткового оточення на пристрої.

Завдяки швидкодії, гарним контролем над використовуваною пам’яттю та підтримці об’єктно-орієнтованого програмування C++ часто використовується для побудови вбудованих систем. Ця мова вже довгий час є в переліку найуживаніших мов для створення додатків орієнтованих на швидкодію та має великий багаж різноманітних фреймворків та бібліотек для використання.

Частину системи можна зробити спільною для клієнтської і серверної частини є доцільним використовувати одну мову програмування для створення обох додатків.

Для реалізації міжпроцесорної взаємодії доцільно використовувати фреймворк, який вже реалізує деякі проміжні етапи. З огляду на надаваний функціонал, популярність, розвинену спільноту та доступність як для комерційної так і для розробки з відкритим кодом та підтримку вбудованих систем є правомірним обрати Qt як основним фреймворком для побудови вихідної системи.

Для порівняння взаємодії побудованої системи взаємодії було використано gRPC як відому систему для віддалених викликів, яка реалізована на мові C++ та не має тому може бути використана для побудови додатків на різноманітних платформах зі змогою їх взаємодії.

3.2 Реалізація серверної частини

Серверна частина складається з реалізації скелету віддаленого об’єкту який і виконує віддалені виклики, тобто реалізує собою віддалений об’єкт та є частиною передбачуваного клієнтського коду та вхідної функції, яка включає в себе запуск серверу з використанням вищезгаданої реалізації. Вхідна функція серверної частини має вигляд:

```
int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    //Skeleton implementation
    RpcGenericServer serverClient;
    //Passing skeleton implementation to the server
    auto tcpServer =
RpcServerFactory::Instance().GetRpcServer(serverClient,
RpcCommon::RPCTYPE_TcpSocket);
    auto grpcServer =
RpcServerFactory::Instance().GetRpcServer(serverClient,
RpcCommon::RPCTYPE_Grpc);
    return a.exec();
}
```

Код, який відповідальний за саму взаємодію винесено в окрему частину за приховано за фабрикою, в даному випадку серверної частини. Це дозволяє змінювати метод транспортну систему взаємодії без зміни клієнтського коду.

В якості файлів проєктів використовуються звичні для Qt фреймворку .pro файли. Через них можна сконфіскувати використання правильних бібліотек та включень в залежності від платформи.

3.3 Реалізація клієнтської частини

Клієнт в даній системі представляє собою віконний додаток, який має декілька вкладок, кожна з яких відповідає за окремий віддалений метод.

На рисунку 3.1 представлено процес роботи з додатком під час виклику віддаленого методу, який рапортує встановлений статус.

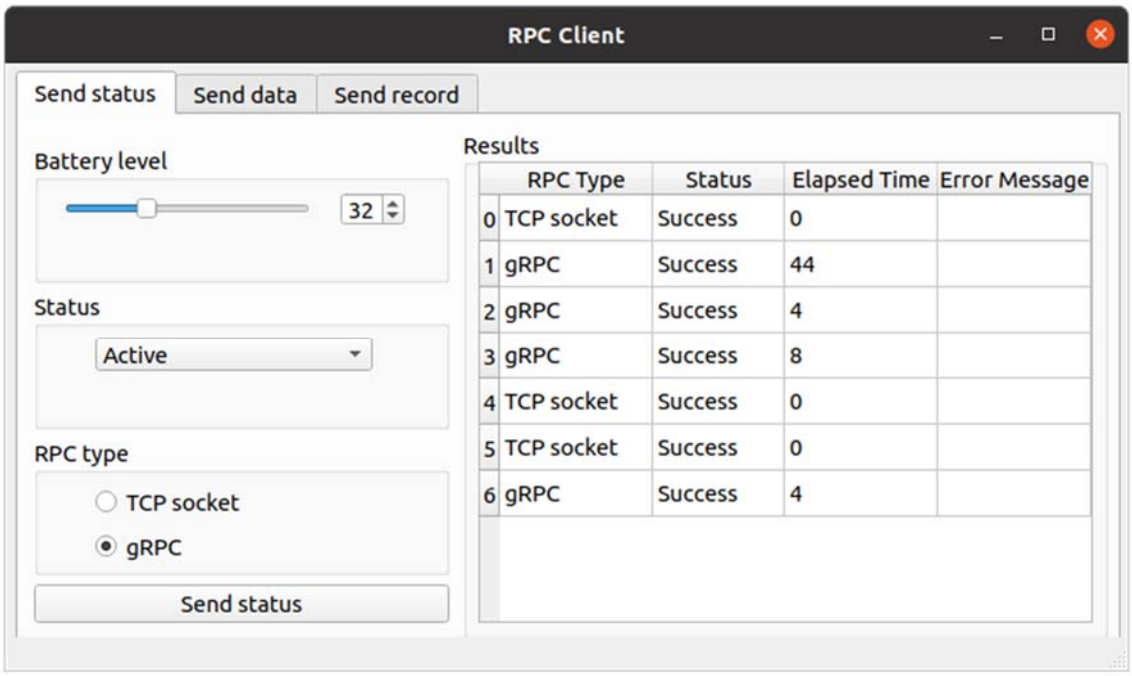

Рисунок 3.1 – Основне вікно клієнтського додатку

В даній частині клієнтський додаток дозволяє встановити бажані параметри статусу, вибрати метод, яким буде відбуватись взаємодія та викликати віддалений метод. Результати виклику зображені в правій частині вікна – тип взаємодії, відповідь серверу, час, затрачений на взаємодію, опис помилки, якщо виклик не вдався.

Для випадку передачі великої кількості бінарних даний реалізовано метод, робота з яким представлена на рисунку 3.2. Через кнопку «Load data» користувач має змогу завантажити необхідні дані, вибрати бажаний спосіб передачі та викликати віддалений метод з передачею цих даних. В таблиці справа представлено результати попередніх викликів.

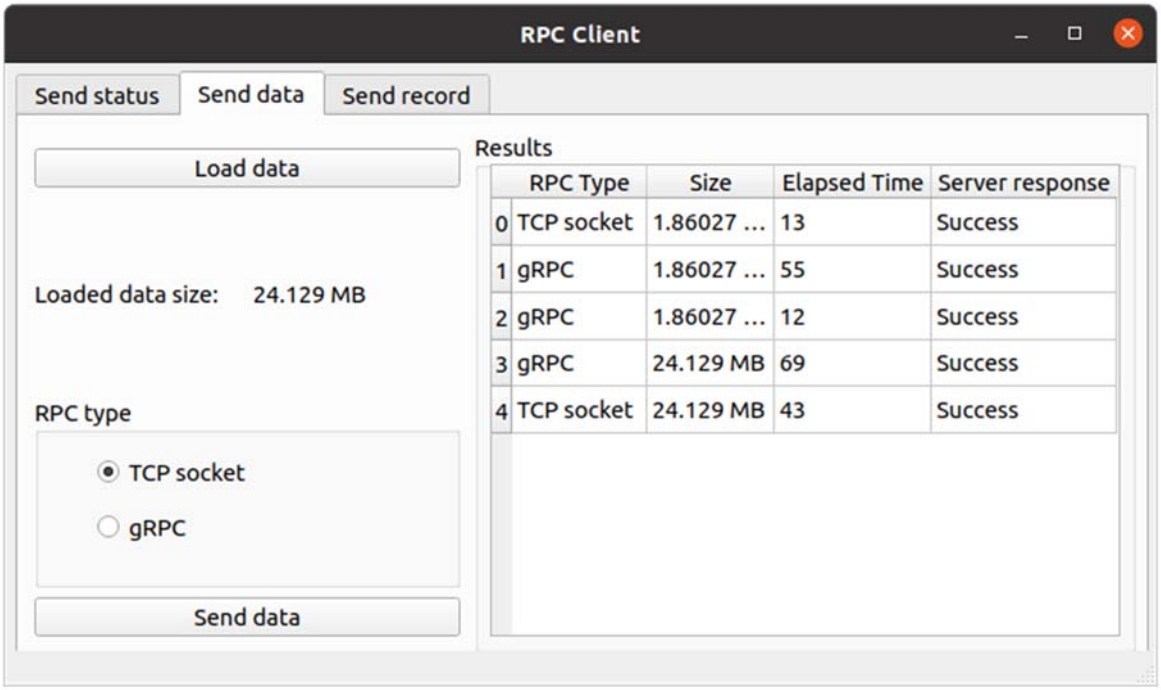

Рисунок 3.2 – Виклик віддаленого методу з передачею бінарних даних

Частина клієнтського додатку для передачі даних сенсорів пацієнта має дещо схожий вигляд. Користування нею відрізняється тим що, завантажити можна тільки вірно сформовані файли записів. Робота з цим методо представлена на рисунку 3.3.

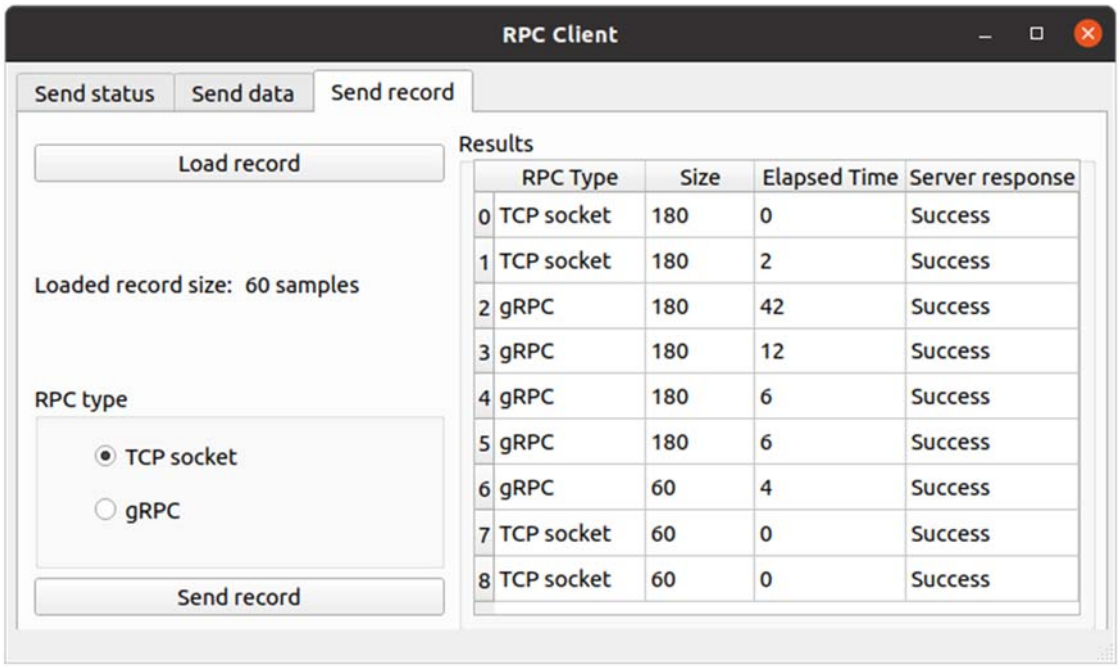

Рисунок 3.3 – Виклик віддаленого методу з передачею життєвих показників

Дані пацієнта представляють собою набор записів показників сенсорів в якийсь момент часу. В даній реалізації такий запис має такі поля:

– час зняття показників;
– пульс пацієнта;
– частота дихання;
– температура;
– систолічний тиск;
– діастолічний тиск.

Всі поля зберігають беззнакове число розміром в 4 байти. Цього достатньо для передачі показників без викривлення, а розмір дозволяє уникнути автоматичного вирівнювання структури компілятором.

Частина, відповідальна за взаємодію знаходиться в окремому каталозі і використовується і серверною і клієнтською частинами.

Нижче наведений приклад коду клієнтського додатку який відправляє серверу поточний статус пристрою:

```
ResponseStatus_e runStatusTestCycle(const RpcCommon::RpcType_e rpcType,
                                    const DeviceStatus_e status,
                                    const DeviceBatteryCharge battery)
{
    QSharedPointer<IClient> client =
RpcClientFactory::Instance().GetRpcClient(rpcType);
    if (client.isNull())
    {
        return ServerResponse_ConnectionError;;
    }

    return client->SendStatus(status, battery);
}
```

У випадку помилки при виклику віддаленого методу користувач отримує сповіщення з описом проблеми, яка виникла.

3.4 Реалізація міжпроцесорної взаємодії

Взаємодія реалізована у вигляді підпроєкту, який включається в вищезгаданих через включення в .pro файлах. Цей підпроєкт містить в собі опис

типів даних які використовуються під час взаємодії, інтерфейси які описують об’єкти взаємодії та реалізацію передачі взаємодії.

Взаємодія була реалізована 2 способами — з використанням gRPC та через передачу даних TCP сокетами з допомогою фреймворку Qt.

Якщо в gRPC за маршалізацію перед передачею даних за вмовчання відповідає Protobuf, то для випадку передачі даних через сокети був використаний вбудовану в Qt серіалізацію даних яка реалізована класом QDataStream. Ця реалізація дозволяє, за допомогою розбору структури на базові типи отримати потік даних який можна відновити на іншій платформі без втрати цілісності. Приклад маршалізації з використанням цього класу наведено нижче:

```
QDataStream &operator<<(QDataStream &stream, const VitalsSample_s &sample)
{
    return stream
            << sample.timestamp
            << sample.hr
            << sample.rr
            << sample.temperature
            << sample.systolic
            << sample.diastolic;
}
```

В спільній частині коду серіалізація-десеріалізація даних для передачі через сокети винесена в окремий клас.

Передача даних таким методом може бути представлена кодом нижче.

```
bool RpcTcpSerializer::WriteInfo(const StatusInfo_s &info)
{
    WriteHeader(RequestType::RequestType_StatusInfo);
    m_stream << info.deviceId;
    m_stream << info.status;
    m_stream << info.battery;

    return true;
}
```

На початку передачі даних записується заголовок, завдяки якому отримувач може перевірити, що дані передаються за очікуваним протоколом та параметри якого методу передаються. В даному випадку він включає в себе

Окремо можна виділити передачу даних великого розміру — починаючи від декількох мегабайт. В цьому випадку вбудованого буферу транспортної підсистеми може не вистачити і деякі дані можуть бути втрачені. Щоб уникнути такої ситуації можна передавати інформацію шматками. Такий підхід також дозволяє більш ефективно використовувати оперативну пам’ять учасників

взаємодії, поступово читаючи/записуючи дані перед відправкою або після отримання. В даному випадку це відноситься і до випадку з передачею бінарних даних і до передачі записів життєвих показників. Частина такого коду представлена нижче.

```
bool RpcTcpSerializer::WriteInfo(const DataInfo_s &info)
{
    WriteHeader(RequestType::RequestType_DataInfo);
    m_stream << info.deviceId;
    m_stream << info.data.size();

    int dataToSend = info.data.size();
    const char * data_ptr = info.data.data();
    qint64 sendData = -1;
    while (dataToSend > 0)
    {
        sendData = m_stream.writeRawData(data_ptr, dataToSend);
        if (sendData == -1)
        {
            return false;
        }
        dataToSend -= sendData;
        data_ptr += sendData;
    }

    return true;
    }
```

В даному випадку, щоб передати великий масив даних спочатку передається розмір цих даних, щоб приймаюча сторона мала змогу оцінити розмір даних та розуміти коли всі дані отримані. Приклад десеріалізації таких даних приведено в в наступній частині коду.

```
bool RpcTcpSerializer::ReadRecordInfo(RecordInfo_s &info)
{
    bool rv = false;

    m_stream.startTransaction();
    m_stream >> info.deviceId;
    int samplesToReceive = -1;
    m_stream >> samplesToReceive;
    if (!m_stream.commitTransaction())
    {
        return rv;
    }
    VitalsSample_s sample;
    QIODevice *device = m_stream.device();
    while (samplesToReceive > 0)
    {
        m_stream.startTransaction();
        m_stream >> sample;
        if (m_stream.commitTransaction())
        {
            info.record.append(sample);
            --samplesToReceive;
        }
        else if (!device->waitForReadyRead(2000))
        {
```

```
            return rv;
        }
    }

    rv = true;
    return rv;
    }
```

Для випадку передачі життєвих показників було додано функцію серіалізації-десеріалізації окремого об’єкту життєвих показників. Це дало змогу передавати цей тип даних тим само способом як і інші вбудовані типи, які підтримує QDataStream.

Для прийому даних використовувався QTcpServer, який при надходженні запиту на нове з’єднання має сигнал. Через систему слотів-сигналів Qt він був приєднаний до реалізації, яка і обробляє запит.

Для порівняння використовувався фреймворк gRPC. Його використання мало такі особливості: виклик віддаленого методу для передачі статусу реалізований в синхронному режимі; виклик з передачею бінарних даних реалізований в синхронному режимі з потоковою передачею даних порціями по 65500 байт; передача масиву життєвих показників реалізована з використанням параметру repeated для відповідного поля.

4 ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ МІЖПРОЦЕСОРНОЇ ВЗАЄМОДІЇ

4.1 Планування експериментів

При дослідженні розробленого алгоритму міжпроцесорної взаємодії в вбудованих системах було вирішено обрати декілька основних типів виклику віддаленого методу:

– виклик з передачею декількох параметрів - статус пристрою;

– виклик з передачею масиву бінарних даних;

– виклик з передачею масиву об’єктів – передача запису життєвих показників.

Якщо до виклик з передачею декількох параметрів значення параметрів не впливає на швидкість взаємодії, то для виклику з передачею даних та запису це має визначне значення. Тому для них було проведено окремі серії експериментів залежно від розміру даних які приймають участь в взаємодії.

Серії експериментів для взаємодії за передачею масиву бінарних даних мали такий розмір даних: 1мб, 5мб, 10мб, 15мб, 25мб, 50мб, 75мб, 100мб.

Для серії експериментів з передачею масиву показників сенсорів за розмір масиву брався час, за який дані показники могли накопичились за умови що показники знімались кожної секунди. Експерименти проводились з такими записами: 1 година, 6 годин, 12 годин, доба, 2 доби, 3 доби, тиждень, 2 тижні, 30 днів.

Кожна серія експериментів проводилась 100 разів. Ефективність кожної взаємодії перевірялась через час затрачений на виконання віддаленого методу.

Експеримент проводився з використанням двох платформ – клієнтської та серверної.

Клієнтська платформа на базі Raspberry Pi 4 Model B має наступні характеристики:

– центральний процесор Broadcom BCM2711;

– частота процесору 1.5 ГГц;
– 4 ядра;
– ОЗУ 2 Гб 3200 МГц.
– Серверна платформа була представлена віртуальним сервером з такими характеристиками:
– частота процесору 4.0 ГГц;
– 4 ядра;
– ОЗУ 3 Гб 1333 МГц.

Взаємодія відбувалась через мережу, до якої сервер був підключений через 1 Гб канал а клієнт через WiFi 5ГГц. Завдяки цьому можна краще зрозуміти як буде поводити себе така система при використанні в готовому приладі.

4.2 Результати експерименту

Результати досліджень серії експериментів для представлені нижче. Середній час взаємодії з передачею статусу відображено на рисунку 4.1

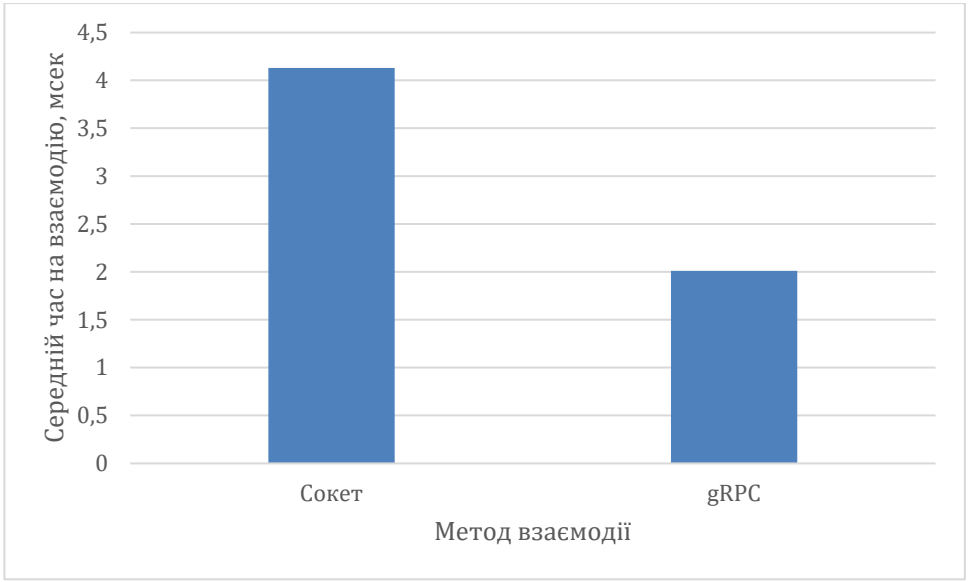

Рисунок 4.1 – Середній час виконання віддаленого методу з передачею статусу

Результати дослідження серії експериментів для взаємодії з передачею даних наведено в таблицях 4.1 та 4.2.

Таблиця 4.1 – Результати виміру швидкодії виклику віддаленого методу з передачею бінарних даних

Розмір переданих	Середній час виконання віддаленої процедури для	
	Сокет, мс	gRPC, мс
1	126,67	102,62
5	522,52	480,91
10	1014,39	951,22
15	1513,06	1433,36
25	2538,16	2431,77
50	5098,91	4863,26
75	7623,17	7099,71
100	10093,59	9504,3

Таблиця 4.2 – Результати виміру швидкодії виклику віддаленого методу з передачею масиву життєвих показників

Розмір запису, годин	Середній час виконання віддаленої процедури для методу	
	Сокет, мс	gRPC, мс
1	25,41	15,04
6	104,03	77,25
12	175,53	159,77
24	296,32	324,41
48	541,43	647,49
72	773,19	980,41
168	1705,83	2318,63
336	3261,1	4732,75
720	6918,89	10337,1

4.3 Рекомендації щодо використання методів між процесорної взаємодії

З отриманих даних можна казати реалізований метод в середньому вдвічі повільніший за gRPC для взаємодії з передачею невеликої кількості параметрів. Але швидкість виконання достатньо мала, щоб вважати обидві реалізації швидкими, тому що різниця між середніми значеннями 2 мілісекунди, що не має великий вплив на функціонування системи в цілому.

В передачі бінарних даних gRPC має невелику перевагу яка коливається в межах 5-15% затраченого часу.

Взаємодія з передачею масиву об’єктів має більшу швидкодію з використанням gRPC для відносно невеликого об’єму – до 12-15 годин запису. Після цього перевагу має реалізація з використанням серіалізації Qt.

Таким чином можна зробити висновок що реалізований метод взаємодії є достатньо ефективним, та може використовуватись як для прототипування так і як частина готового продукту. В той же час різниця в швидкодії може бути критичною для системи, в такому випадку краще використовувати gRPC для взаємодії з невеликою кількістю параметрів або при передачі бінарних даних. Для великих обсягів масивів об’єктів кращу швидкодію показала розроблена реалізація.

Також на вибір міжпроцесорної взаємодії може впливати платформа та мова програмування, яка використовується для реалізації процесів. Qt доступний на багатьох платформах, в тому числі Windows, Linux, QNX, але реалізація додатків з прямим використанням фреймворку доступна тільки в мовах програмування C++ та Python.

В той же час gRPC, на відміну від Qt має інтерфейси для використовування в понад 10 різних мовах програмування. Це дозволяє будувати різні частини системи використовуючи підходящу мову програмування зберігаючи можливість ефективно взаємодіяти з рештою системи.

ВИСНОВКИ

В даній кваліфікаційній роботі було досліджено набір методів міжпроцесорної взаємодії, їх особливості на вбудованих системах.

З проведеного аналізу можна виділити основні групи методів для взаємодії процесів, в тому числі і для вбудованих систем, самі методи, особливості їх застосування. Ці дані можна використовувати при проектуванні нових вбудованих систем або для оптимізації або розширенні функціоналу існуючих.

Було розроблено алгоритм міжпроцесорної взаємодії за протоколом виклику віддалених процедур та побудовано архітектуру прикладної системи, яка його використовує. Побудований алгоритм було порівняно з відомою системою для виклику віддалених процедур gRPC.

Алгоритм було реалізовано за допомогою мови програмування C++, фреймворку Qt. Під час роботи використовувався QtCreator.

За результатами експериментального дослідження було сформовано рекомендації щодо міжпроцесорної взаємодії з використання виклику віддалених процедур що можуть бути використані при проектуванні вбудованих систем які мають взаємодіяти з зовнішнім сервером, що поширено в медичній та побутовій сферах, IoT.

За результатами дослідження були опубліковані тези в рамках II конференції «Сучасна молодь в світі інформаційних технологій» (див. додаток Д).

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Barr M. Embedded Systems Glossary // Neutrino Technical Library [Електронний ресурс]: URL: https://barrgroup.com/Embedded-Systems/Glossary (дата звернення: 23.03.2021).

2. Barr M., Massa A. J. Programming embedded systems: with C and GNU development tools – O'Reilly, 2006. – 336 с.

3. Embedded Systems Role in Automobiles with Applications [Електронний ресурс]: URL: https://www.watelectronics.com/importance-of-embedded-systems-in-automobiles-with-applications/ (дата звернення: 23.03.2021).

4. Upadhyay A., Dhapola A. S. Embedded Systems and its Application in Medical Field // Emerging Trends in Computer Science and Information, 03.2015

5. Development Process of Embedded Systems [Електронний ресурс]: URL: https://www.watelectronics.com/embedded-systems-development-process/ (дата звернення: 23.03.2021).

6. OMG pursues new strategic direction to build on success of past efforts [Електронний ресурс]: URL: http://xml.coverpages.org/OMG-MDA2001-03-08a.htm (дата звернення: 23.03.2021).

7. Kaisti M., Rantala V., Mujunen T., Hyrynsalmi S. Agile methods for embedded systems development-A literature review and a mapping study // EURASIP Journal on Embedded Systems. 2013. № 15.

8. Bezsmertnyi O., Golian N., Golian V., Afanasieva I. Behavior Driven Development Approach in the Modern Quality Control Process // Problem of Infocommunications. Science and Technology (PIC S&T′2020), 10.2020.

9. Interprocess Communications // Windows documentation. Microsoft [Електронний ресурс]: URL: https://docs.microsoft.com/en-us/windows/win32/ipc/interprocess-communications (дата звернення: 23.03.2021).

10. Afanasieva I., Golian N., Hnatenko O., Daniiel Y., Onyshchenko K. Data exchange model in the Internet of Things concept // Telecommunications and Radio Engineering. №10(78) - 2019. P. 869-878.

11. Erlingsson Ú., Kyparlis A. Microkernels, [Електронний ресурс]: URL: https://www.cs.cornell.edu/home/ulfar/ukernel/ukernel.html (дата звернення: 23.03.2021).

12. AUTOSAR [Електронний ресурс]: URL: https://www.autosar.org/ (дата звернення: 24.03.2021).

13. Data Distribution Service (DDS) [Електронний ресурс]: URL: https://www.omg.org/spec/DDS/1.4/PDF (дата звернення: 24.03.2021).

14. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software – Addison Wesley, 1994. – 395 с.

15. gRPC [Електронний ресурс]: URL: https://grpc.io/ (дата звернення: 24.03.2021).

16. Bachmann F., Bass L., Clements P., Garlan D., Ivers J., Little R., Nord R., Stafford J. Documenting Software Architecture: Documenting Interfaces - Carnegie Mellon University, 2002. – 48 с.

17. Authentication | gRPC [Електронний ресурс]: URL: https://grpc.io/docs/guides/auth/ (дата звернення: 25.03.2021).

18. Blackboard Design Pattern // Microsoft TechNet. Microsoft. [Електронний ресурс]: URL: https://social.technet.microsoft.com/wiki/contents/articles/13215.blackboard-design-pattern.aspx (дата звернення: 25.03.2021).

19. Embedded Markets Study // AspenCore [Електронний ресурс]: URL: https://www.embedded.com/wp-content/uploads/2019/11/EETimes_Embedded_2019_Embedded_Markets_Study.pdf (дата звернення: 25.03.2021).