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 ефективності та оптимізації технологій міжпроцесної взаємодії у серверних додатках
(тема)

Виконав:
студент (ка) 2 курсу, групи ІПЗм–22–2

Свєтлінський О.А.
(прізвище, ініціали)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)
Тип програми освітньо–наукова

Керівник проф. Лєсна Н.С.
(посада, прізвище, ініціали)

Допускається до захисту
Зав. кафедри _____ З.В.Дудар
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерних наук
Кафедра програмної інженерії
Рівень вищої освіти другий (магістерський)
Спеціальність 121 – Інженерія програмного забезпечення
Тип програми освітньо–наукова програма
Освітня програма Інженерія програмного забезпечення
(шифр і назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«___» _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Свєтлінському Олегу Андрійовичу
(прізвище, ім’я, по батькові)

1. Тема роботи «Дослідження ефективності та оптимізації технологій міжпроцесної взаємодії у серверних додатках»

Затверджена наказом по університету від 29.03. 2024р. № 250 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 11.06.2024

3. Вихідні дані до роботи технології IPC, об’єктно–орієнтовне програмування, електронні ресурси за обраною тематикою, пояснювальна записка, мови програмування C++, Python, середовище програмування QtCreator, IDLE

4. Перелік питань, що потрібно опрацювати в роботі
мета роботи, аналіз предметної області, постановка задачі, дослідження технологій міжпроцесної взаємодії, вивчення можливостей їх використання, аналіз ефективності та оптимізації кожної з них, написання програмних рішень, проведення експериментів та аналіз отриманих результатів

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі та постановка задачі	30.03 – 02.04.24	*виконано*
2	Аналіз та вибір API для дослідження	02.04 – 03.04.24	*виконано*
3	Аналіз та моделювання предметної області	03.04 – 04.04.24	*виконано*
4	Планування експериментів	04.04 – 08.04.24	*виконано*
5	Програмна реалізація кожного з обраних для дослідження API	09.04 – 16.04.24	*виконано*
6	Експериментальні дослідження	17.04 – 19.04.24	*виконано*
7	Аналіз результатів експериментальних досліджень та розробка рекомендацій	20.04 – 25.04.24	*виконано*
8	Написання та оформлення статті та тез доповіді	17.03 – 27.03.24	*виконано*
9	Підготовка пояснювальної записки	28.04 – 22.05.24	*виконано*
10	Підготовка презентації та доповіді	22.05 – 28.05.24	*виконано*
11	Нормоконтроль	30.05 – 01.06.24	*виконано*
12	Рецензування	01.06 – 06.06.24	*виконано*
13	Занесення диплома в електронний архів	08.06.2024	*виконано*
14	Попередній захист	08.06.2024	*виконано*
15	Допуск до захисту у зав. кафедри	09.06.2024	*виконано*

Дата видачі завдання 29 безерня 2024р.

Студент (ка) _____ Свєтлінський О.А.
(підпис)

Керівник роботи _____ проф. Лєсна Н.С.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Пояснювальна записка містить: 85 с., 15 рис., 18 табл., 17 джерел.

ТЕХНОЛОГІЇ МІЖПРОЦЕСНОГО ЗВ'ЯЗКУ, ІМЕНОВАНІ КАНАЛІ, UNIX–СОКЕТИ, БЕК–ЕНД, СПІЛЬНА ПАМ'ЯТЬ, ВІДДАЛЕНІ ОБ'ЄКТИ QT, ЕФЕКТИВНІСТЬ КОДУ, ЧЕРГИ ПОВІДОМЛЕНЬ.

Об'єктом дослідження є технології міжпроцесної взаємодії при передачі даних у рамках однієї системи, а саме серверного додатку.

Метою роботи є дослідження та аналіз ефективності та оптимізації існуючих технології міжпроцесної взаємодії, виділення критеріїв та методів для проведення порівняльного аналізу.

У результаті роботи розглянуто існуючі технології та протоколи міжпроцесної взаємодії для передачі даних, досліджено їх особливості, переваги та недоліки та принципи роботи, описано та продемонстровано методи порівняння та запропоновано формули для обчислення числових показників.

INTER–PROCESS COMMUNICATION, NAMED PIPES, UNIX SOCKETS, BACK–END, SHARED MEMORY, QT REMOTE OBJECTS, CODE EFFICIENCY, MESSAGE QUEUES.

The object of the study is the technology of interprocess interaction during data transfer within the framework of one system, namely, a server application.

The purpose of the work is research and analysis of the efficiency and optimization of existing technologies of interprocess interaction, selection of criteria.

As a result of the work, the existing technologies and protocols of inter–process interaction for data transfer were considered, their features, advantages and disadvantages and working principles were investigated, methods of comparison were described and demonstrated, and formulas for calculating numerical indicators were proposed.

Заява щодо самостійного виконання кваліфікаційної роботи та можливості її публікації в електронному архіві відкритого доступу ElArKhNURE.

Я, Свєтлінський Олег Андрійович, студент гр. ІПЗм–22–2, здобувач вищої освіти на другому (магістерському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Дослідження ефективності та оптимізації технологій міжпроцесної взаємодії у серверних додатках», 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. Усі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений(на) 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

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

IPC (Inter Process Communications) – комп'ютерна технологія, обмін даними між потоками одного та/або різних процесів.

D–Bus (Desktop Bus) – система міжпроцесної комунікації, яка дозволяє застосункам в операційній системі спілкуватися один з одним.

QtRO (Qt Remote Object) – це модуль взаємодії між процесами, який був розроблений для Qt.

Qt – фреймворк для розробки кроссплатформенного програмного забезпечення мовою програмування C++.

OS (Operating System) – Операційна система. Це базовий комплекс програм, що виконує керування апаратною складовою комп'ютера або віртуальної машини.

AWS (Amazon Web Services) – є дочірньою компанією Amazon.com, що надає платформу хмарних обчислень в оренду приватним особам, компаніям та урядам на основі платної підписки.

HTTPS (HyperText Transfer Protocol Secure) – протокол передачі даних, що використовується в комп'ютерних мережах.

IPMI (Intelligent Platform Management Interface) – інтелектуальний інтерфейс управління платформою, призначений для автономного моніторингу та управління функціями, вбудованими безпосередньо в апаратне та мікропрограмне забезпечення серверних платформ.

GUI (Graphical User Interface) – тип інтерфейсу, який дає змогу користувачам взаємодіяти з електронними пристроями через графічні зображення та візуальні вказівки.

TCP (Transmission Control Protocol) – протокол, що призначений для керування передаванням повідомлень та даних у комп'ютерних мережах, працює на транспортному рівні моделі OSI.

OSI (Open Systems Interconnection) – абстрактна мережева модель для комунікацій і розроблення мережевих протоколів.

FIFO (first in, first out) – спосіб організації та маніпулювання даними щодо часу та пріоритетів. Також може описувати одну з технології IPC.

СУБД (Система управління базами даних) – набір взаємопов'язаних даних і програм для доступу до цих даних.

API (application programming interface) – інтерфейс програмування застосунків. Набір визначень підпрограм, протоколів взаємодії та засобів для створення програмного забезпечення.

Mb (Mega byte) – одиниця вимірювання обсягу даних. Рівна 1 048 576 (220) байт або 1024 кілобайт.

ВСТУП

У сучасному світі, де інформаційні технології стають важливою складовою кожного аспекту життя, ефективність та оптимізація технологій міжпроцесної взаємодії є невід'ємною частиною розвитку галузі. Якщо умовних десять років назад, на одній машині могли працювати пару десятків програм або процесів, то сьогодні ця кількість може перевищувати сотні. Особливо це стосується саме серверних рішень, бо саме для них це відіграє велику роль.

Спостерігаючи швидкі темпи розвитку інформаційних технологій, необхідно зосередити увагу на питаннях оптимізації та підвищення ефективності, які відіграють вирішальну роль у забезпеченні стабільності та високої продуктивності серверних програм. Зростання обсягів обробки даних та збільшення завдань, що стоять перед серверними програмами, породжують необхідність поліпшення багатьох аспектів, одним з яких є – міжпроцесна взаємодія.

Дослідження та оптимізація сучасних технологій у цьому контексті не лише підвищують ефективність серверних додатків, а й сприяють економії ресурсів та забезпеченню більшої стабільності роботи систем. Застосування відповідних технічних рішень при міжпроцесній взаємодії дозволяє підвищити пропускну здатність, скоротити час відгуку та оптимізувати використання ресурсів. Оптимально налаштовані технології взаємодії дозволяють суттєво покращити роботу серверних програм, забезпечивши високу швидкість та ефективність обробки інформації.

У цьому дослідженні розглядається актуальність питань оптимізації міжпроцесної взаємодії у сучасних серверних додатках, а також вивчаються потенційні напрямки вдосконалення цих технологій. Аналіз та ефективна оптимізація технологій міжпроцесної взаємодії – важливий етап забезпечення стабільної та продуктивної роботи серверних додатків у сучасному високотехнологічному середовищі.

Метою даної роботи є глибоке порівняння існуючих протоколів спілкування між процесами у контексті серверних додатків.

Для того, щоб отримати результат, що б задовольнив мету, необхідно вирішити набір наступних питань:

– проаналізувати існуючі протоколи та способи обміну даними між процесами в одній системі;

– визначитись з методами та критеріями для порівняння технологій та сформувати план аналізу технологій;

– розробити програмні застосунки для проведення експериментів у відповідному контексті;

– отримані дані використати для виміру обраних метрик для порівняння технологій;

– проаналізувати отримані результати;

– сформулювати рекомендації створення способів міжпроцесної взаємодії;

– запропонувати потенційне розширення дослідження та охарактеризувати доречність роботи у майбутньому.

Об'єктом дослідження є ефективність та доцільність використання технологій міжпроцесної взаємодії у серверних додатках.

Предметом дослідження є технології міжпроцесної взаємодії.

Методами дослідження є вимірювання показників ефективності за критеріями та обчислення за допомогою запропонованих математичних формул для обчислення кожного з показників.

Результати роботи можна успішно використовувати у створенні чи подальшому аналізі нових технологій міжпроцесної взаємодії, покращення вже існуючих рішень чи оптимізацію існуючих серверних додатків.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз предметної галузі

Міжпроцесорна взаємодія (англ. inter–process communication, IPC) відіграє важливу роль у світі операційних систем, дозволяючи різним процесам спілкуватися та співпрацювати один з одним. Ця технологія має вирішальне значення для багатозадачності, дозволяючи процесам спілкуватися, синхронізувати свою діяльність і ефективно співпрацювати. Існують різні методи та способи реалізації взаємодії в сучасних операційних системах, та робота буде сфокусована на використання IPC саме для системи Linux через те, що більшість серверних рішень використовують саме цю операційну систему. Це видно зі статті «Ubuntu Linux continues to rule the cloud» [1]. Ubuntu, що є дистрибутивом Linux домінує на ринку хмарних серверних технологій, що показано на рисунку 1.1.1, взятому зі статі вище. Зараз велика кількість серверних рішень переходить та зразу розробляються «на хмарі», тому обираємо саме дослідження на Linux.

Рисунок 1.1 – Розподілення ОС на хмарному хостингу AWS (за даними [1])

IPC є головним механізмом, який операційні системи використовують для полегшення або встановлення зв’язку між внутрішніми процесами. У багатозадачному середовищі кілька процесів виконуються одночасно, а IPC служить як міст, який дозволяє їм обмінюватися інформацією та координувати свої дії.

З основних задач міжпроцесної взаємодії можна виділити наступні:

– обмін даними: процеси часто вимагають спільного використання даних.

Взагалі, одна з непомітних, але вкрай важливих особливостей сучасних операційних систем – це багатопроцесорність. Один розробник може створити перший процес, інший – другий процес, та вони можуть спілкуватися між собою, створюючи систему, що може виконувати більше роботи, аніж два процеси окремо. Одним з прикладів є текстовий процесор, який може передати дані процесу принтера, щоб створити друковану копію або веб–камера, що постійно передає медіа–дані, може обмінюватися ними з процесором–архіватором для автоматичного та прозорого для користувача стиснення даних;

– синхронізація: процесам може знадобитися синхронізувати свою діяльність для стабільної та тривалої роботи. Це може бути або багатопоточний додаток, або два різних процеси у системі. Наприклад, маємо додатковий процес між текстовим процесом та принтером для синхронізації їх роботи, а тобто текстові дані з невідомою швидкістю потраплять до нового процесу, а він, один за одним, передає їх до принтеру для стабільної печаті і все це через IPC;

– комунікація: процеси можуть вимагати комунікації для різних цілей, таких як обмін інформацією, сигналізація та обробка помилок. У якомусь сенсі це схоже на HTTPS, якщо говорити про звичайні сервер–клієнт системи. Прикладом комунікації за допомогою міжпроцесорни технологій є сигнал від драйверу присутності вентиляторів на серверній платі передати про відсутність половини елементів до графічного інтерфейсу користувача–адміністратора. Продовжуючи приклад, за сигналом про відсутність вентиляторів, процес підтримки температури може припиняти свою роботу та генерувати критичні помилки до інтерфейсів користувачів–адміністраторів, наприклад до IPMI [2];

– спільне використання ресурсів: IPC допомагає керувати такими ресурсами, як доступ до файлів, пам’яті або апаратних пристроїв, і ділитися ними між процесами. Таким чином один і той самий файл можуть безпечно використовувати зразу декілька процесів. У даному контексті «файл» може бути не тільки умовний текстовий документ або медіа, а й драйвери периферійних пристроїв такі як принтери, веб–камери, проектори, тощо. Невдале використання таких драйверів може негативно сказатися на стані самих пристроїв, тому таке спільне

використання є невід’ємною частиною вдалого програмного забезпечення [3].

Описавши основні задачі міжпроцесних технологій, бачимо, що більше всього отримують користь від ефективно налагодженої IPC, або страждають, від неефективної, наступні технології:

– канали командної оболонки (англ**.** Shell Pipelines): в Unix–подібних системах командна оболонка використовує канали для з’єднання виводу однієї команди з вводом іншої. Зазвичай це відбувається з–за використанням символу «|». Це дозволяє швидко фільтрувати вивід деяких команд, чи записувати до файлу інформацію, написавши команду в одну стрічку, коли інакше це потребувало б цілого скрипта на умовній мові bash [4];

– графічні інтерфейси користувача (англ. graphical user interface, GUI): програми графічних інтерфейсів використовують IPC для обробки подій, таких як надсилання повідомлень між вікнами або процесами. Існують фреймворки, де кожне вікно – це окремий процес, тому майже чи не кожний «клік» користувача супроводжується масивом повідомлень між процесами. Легко зрозуміти, що підвищення ефективності міжпроцесної взаємодії може зменшити час очікування користувача та зробити програму систему більш чуйною. Окрім того, кажучи саме про серверні рішення, багато критичних повідомлень повинні швидко та чітко потрапити до адміністраторів задля найскорішого полагодження апаратного забезпечення. У прикладі з вентиляторами, пошкодженим може бути центральний процесор, а це зайві витрати та ризики для власників серверів;

– зв'язок між сервером і клієнтом: IPC важливий у додатках клієнт–сервер. Ця архітектура може використовуватися не тільки меж різними машинами. Клієнти та сервери спілкуються через сокети, канали або інші механізми передачі даних між процесорами, наприклад DBus [5]. Прикладом такої архітектури є СУБД MySQL, встановлена на сервері з back–end додатком. Скоріш за все, вони будуть обмінюватися інформацією через unix–сокет;

– багатопоточність: у багатопоточних додатках потоки в межах процесу повинні спілкуватися та синхронізуватись за допомогою механізмів IPC, таких як семафори та м’ютекси. Можливе використання і інших механізмів, але це буде не

оптимально, тому що ця область є доволі самобутньою. Через це вона не буде розглянута у дослідженні;

– розподілені системи: міжпроцесорна взаємодія має вирішальне значення для розподілених обчислень, де процеси можуть виконуватися на різних машинах. Передача повідомлень зазвичай використовується для обчислення великих об’ємів даних, або спільною роботою над медіа. Одним з таких прикладів є історія про те, як рендерили кінофільм «Володар Перстнів» [6]. Для поточного дослідження цей аспект враховуватися не буде через те, що зазвичай серверне рішення – це додаток на одній машині, а для обміну інформацією між різними машинами, зазвичай використовується більш придатні для цього протоколи, як HTTPS і також для розподілених систем грає важливу роль тип та архітектура мережі, до якою входять обрані машини, а це потребує окремного дослідження та свої критеріїв.

Серед потенційних проблем у міжпроцесному спілкуванні або просто особливостей, які обов’язково треба мати на увазі, слід зазначити наступний список пунктів:

– стан гонитви (англ. race condition): процеси мають бути ретельно синхронізовані, щоб уникнути умов змагання, які можуть призвести до пошкодження даних. Це наріжний камінь багатопоточних додатків чи багатопроцесорних систем. Ця особливість піднімає питання про доцільність великої кількості малих повідомлень чи декількох великих, чи взагалі, доцільності втручати до задачі декількох виконавців, чи краще зупинитися на одному;

– безпека даних: неавторизований доступ до спільних даних може становити велику загрозу безпеці даних чи всій системі в цілому. Зловмисник може використовувати вразливості додатків, що мають доступ до критичних даних, як логін та пароль користувачів, та вкрасти їх. Суть полягає у тому, що обмін критичними даними це, з одного боку, зручно та корисно, а з іншого – потенційно небезпечно, тому що треба слідкувати за безпекою усіх компонентів обміну даними;

– складність: частіше міжпроцесорні технології вважаються доволі низькорівневими, які може бути важко використовувати, а помилки можуть

призвести до нестабільності системи. Розробники часто посилаються на такі протоколи як «темний ліс», де все має значення, але нічого не зрозуміло. Це критичний елемент додатків, які деколи працюють на критичних рівнях операційної системи, тому помилки слід звести до мінімуму. Таким чином складність є подвійною проблемою, тому що просто зробити працездатний код – недостатньо. Є високорівневі альтернативи, як TCP або DBus, але через їх абстрактність може бути зниження швидкості передачі даних;

– накладні витрати на продуктивність: деякі механізми IPC, особливо передача повідомлень, можуть призвести до накладних витрат на продуктивність через копіювання даних і зміну контексту. У комп'ютерах все має свою ціну у часі або пам'яті та не буває суто гарних рішень.

1.2 Постановка задачі

Маючи розуміння того, як і для чого можна використовувати міжпроцесорні технології та які недоліки та переваги вони мають, необхідно проаналізувати наявну інформацію про те, які конкретні протоколи, модулі, технології існують та які з них треба обрати для дослідження.

Можливими показниками успішної роботи кожного з методів, які можна виміряти та які потенційно будуть впливати на остаточний вибір розробників, є:

– доступність програмного забезпечення. Те, як легко його відключити до проекту. На C++, мові програмування що найчастіше використовується для розробки міжпроцесорних технологій, це окрема тема, яка може «з'їсти» не один день розробки, та не одного розробника;

– рівень абстракції технології. Те, як легко її зрозуміти і на якому рівні можна її використовувати. Існує очевидна різниця між використанням сирих масивів байтів, та відправкою повідомлень з пакетами з окремі ідентифікаційні номери, тощо;

– максимальний розмір одиниці даних або пакету. Можем відправити одне повідомлення, розбивши на пакети, чи одним пакетом. Звичайно, один пакетом

буде швидше, простіше та надійніше, але різні технології мають різні максимальні розміри пакетів чи буферів;

– середній час обміну даними. IPC схожі за своєю ідеєю на комунікацію у мережі Інтернет, тому як і у дослідженнях ефективності технологій передачі даних по мережі [7], швидкість обміну даними є чи найважливішою характеристикою.

– безпечність використання при паралельному виконанні. Цей показник відповідає на питання, як: чи потребує технологія додаткових механізмів безпеки або чи треба використовувати додаткові людино–години на тестування технології у паралельному програмуванні.

Беручи до уваги усі показники вище та аналіз області міжпроцесорних технологій, у рамках проведення дослідження ефективності необхідно вирішити наступні завдання:

– розглянути методи існуючих IPC, які можуть бути використані у серверних додатках;

– обрати ті технології, що найкраще підійдуть для проведення порівняльного аналізу;

– виділити критерії, за якими можна буде найкраще порівняти технології та які зможуть покрити більшість особливостей;

– визначитись з метриками та чинниками, що впливають на кожний з показників, а також визначити ступінь впливу кожного з них, вказати пріоритетність метрик у досліджені;

– сформувати план для проведення експерименту задля отримання експериментальних даних, створити програмні тестові середовища для проведення вимірювань для кожного з критеріїв та технологій;

– провести експеримент, проаналізувати отримані результати та задокументувати знахідки;

– надати рекомендації та результат аналізу щодо використання конкретних способів обміну даними між процесорами, а також сформулювати можливі потенційні напрями розширення дослідження.

2 ОГЛЯД ІСНУЮЧИХ ТЕХНОЛОГІЙ МІЖПРОЦЕСНОЇ ВЗАЄМОДІЇ

2.1 Класифікація технологій міжпроцесної взаємодії

Існує три головних напрями технологій IPC, які вже були частково покриті у попередньому розділі: напрями спілкування, сигналів та синхронізації. Схематичне представлення цього наведено на рисунку 2.1.

Напрям комунікації в міжпроцесних комунікаційних технологіях визначає, як процеси можуть обмінюватися інформацією та взаємодіяти один з одним. Це передбачає передачу даних між процесами, обмін повідомленнями.

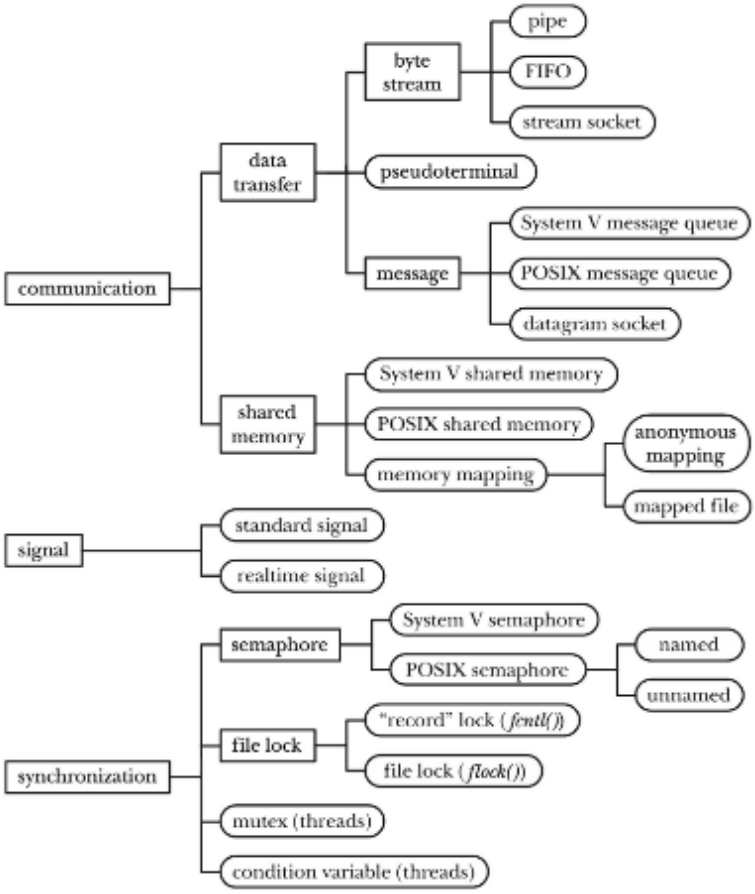

Рисунок 2.1 – Головні напрями технологій IPC (за даними [8])

Цей напрям є основним для дослідження, тому що більшість технологій IPC припадає саме нього та порівняння рішень має великий сенс, через велику різницю у реалізації потреб цього напряму.

Сигнали в міжпроцесному спілкуванні використовуються для передачі

простих повідомлень або сигналів між процесами. Сигнали можна використовувати для сповіщення процесів про певні події або зміни в системі.

Синхронізація в міжпроцесному зв'язку є важливим аспектом для забезпечення правильного порядку операцій і уникнення конфліктів. Механізми синхронізації дозволяють процесам чекати один одного, спільно використовувати ресурси або блокувати певні операції до досягнення певної умови. Інструменти синхронізації включають м'ютекси, семафори, бар'єри та інші механізми, які забезпечують взаємодію та координацію між процесами.

Цей напрям знаходиться «осторонь», так як все, що потрібно для забезпечення безпечного спілкування, вже існує, усі технології розроблені та продумані. Тут не має нових альтернатив чи конкурентних рішень, тому і сенсу робити дослідження нема.

Таким чином основним напрямком технологій міжпроцесної взаємодії, який буде розглянутий в досліджені, є напрям комунікації, передачі даних чи сигнальних повідомлень між процесами.

Також під час проведення аналізу предметної області було знайдено, що існують різні «рівні» технологій IPC. Не було знайдено офіційних понять, тому задля розуміння подальшого матеріалу дослідження, вводимо поняття «низькорівневі» та «високорівневі» технології міжпроцесної взаємодії.

Технології низького рівня в міжпроцесному зв'язку тісно пов'язані з операційною системою та прямим зв'язком між процесами. Ці технології зазвичай вбудовані в операційну систему або використовуються як системні виклики. Вони забезпечують простий, ефективний і прямий спосіб обміну даними та керування процесами. Приклади включають в себе: сигнали, семафори, спільну пам’ять або канали.

Однак використання низькорівневих технологій може вимагати великої уваги до деталей і складних механізмів для синхронізації та управління ресурсами. Вони забезпечують прямий контроль над ресурсами та зв’язком, але можуть вимагати багато кодування та ручного керування.

Технології високого рівня в міжпроцесному зв’язку базуються на технологіях

низького рівня, але додають абстракції та інтерфейси вищого рівня, щоб полегшити їх використання та зменшити складність. Ці технології забезпечують більше абстракцій і зручні інтерфейси для програмістів, полегшуючи розробку та підтримку міжпроцесного зв’язку.

Приклади технологій високого рівня включають протоколи, які використовують мережевий зв’язок, наприклад TCP/IP, або фреймворки, такі як DBus, які дозволяють процесам спілкуватися через шину повідомлень за допомогою абстракцій об’єктів і методів. Вони забезпечують зручний і стандартизований спосіб обміну даними, але можуть мати певні витрати порівняно з технологіями нижчого рівня.

Не можна нехтувати зручністю розробки серверних додатків з використанням тієї чи іншої технології, тому потрібно дослідити чи впливає додавання абстракції високорівневих рішень на швидкість та якість передачі даних і якщо так, то наскільки це є критичним. Звичайно чим потенційно можна було б знехтувати, але тоді маємо ситуації «чорної коробки», коли код роботу робить, а як – ніхто не знає та при виявлені помилки постає комплексна задача с якою не завжди можна швидко та правильно справитися.

2.2 Огляд Pipes

Технологія каналів чи труб (англ. Pipes) – це односторонній обмін даними від одного процесу до іншого. Процес–виробник створює новий канал за допомогою системного виклику, це створює два файлових дескрипторів: один для читання з каналу і один для запису до нього. Процес–виробник ділиться дескриптором з процесом–споживачем.

Це, так званий, напівдуплексний канал, тому кожен процес має бути закритий перед використанням іншого. Напівдуплексний вид зв'язку або каналу передачі даних, де обмін інформацією можливий тільки в один бік у певний момент часу. У такому каналі учасники можуть відправляти або приймати дані, але не обидва одночасно.

Схема роботи каналу з використанням системних викликів Linux за допомогою мови програмування С наведено на рисунку 2.2.

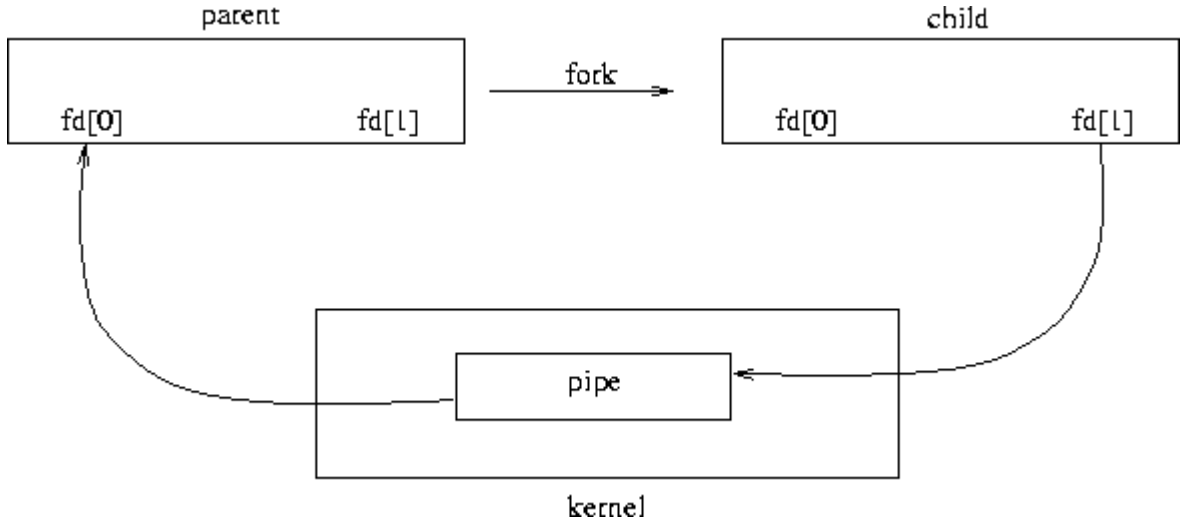

Рисунок 2.2 – Схематиче представлення технології Pipes (за даними [9])

2.3 Огляд FIFOs

FIFO (First–in–First–Out) чи іменований канал (англ. Named pipe) – це розширення традиційної концепції каналу. Традиційний канал «безіменний» і існує стільки ж, скільки і сам процес. Це «тимчасова» структура, яка видна тільки тип процесам, які її використовують. Іменований канал може існувати доти, поки система працює, за межами життя процесу. Канал можна побачити у файловій системі та видалити, якщо він більше не використовується. Як правило, іменований канал відображається як файл, і процеси зазвичай підключаються до нього для взаємодії між процесами.

Файл іменованого каналу – це особливий тип файлу в локальному сховищі, який дозволяє двом або більше процесам спілкуватися один з одним шляхом читання та запису з файлу. Такий файл вводиться у сховище шляхом спеціального системного виклику операційної системи. Після того, як файл було створено, будь–який процес може відкрити його для читання або запису, як і звичайний файл, однак він має бути відкритий з обох кінців одночасно, перш ніж ви зможете продовжувати виконувати над ним будь–які операції введення чи виведення.

Таким чином, це є технологія дуплексного зв’язку, хоча частіше за все її використовують саме у напівдуплексному режимі, як заміну безіменному каналові.

2.4 Огляд Unix Sockets

Unix сокети (англ Unix Sockets) – наступний крок розвитку IPC на основі каналів. Сокети також можуь бути представлені як спеціальні файлі у локальному сховищі, але використовують окремий інтерфейс для роботи з передачі даних, відмінний від інтерфейсу роботи з файлами.

Щоб використовувати сокети Unix, процеси повинні створити сокет, налагодити його на очікування з'єднання (для виробника) або встановити з'єднання (для споживача). Після встановлення зв’язку процеси можуть взаємодіяти через цей сокет обмінюючись даними. На відміну від каналів, сокети використовують буферизацію у обох напрямках передачі даних, що дозволяє менше займатися синхронизацією поміж користувачів, та робить процес зчитування та запису більш стабільним для обох користувачів з’єднання, але недоліком є зайві копії дані, що неминуче створюються при перенесенні даних з та до буферів. Таким чином використання одночасно великої кількості сокетів не є рекомендованим. Це наведено на рисунку 2.3.

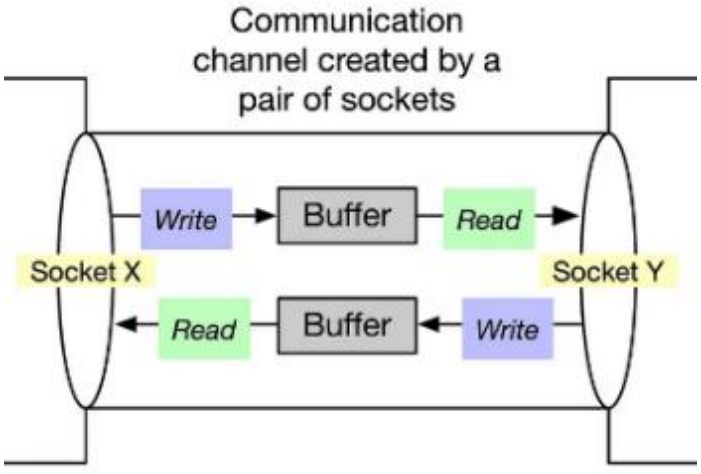

Рисунок 2.3 – Схематичне представлення технології Unix Sockets (за даними [10])

Unix сокети є потужним засобом міжпроцесної взаємодії та використовуються для реалізації різних широких сценаріїв та можуть бути розширені до використання спілкування між різними машинами, що виходить на рамки дослідження, але все ж таки є великою перевагою. Це може бути включене до наступних досліджень на поточну тематику.

2.5 Огляд Message Queues

Черги повідомлень (англ. Message Queues) є ще однією більш розвиненою формою каналів. Вони дозволяють процесам надсилати стандартизовані структури – повідомлення, один одному і їх також як і Unix сокети можна використовувати тільки для обміну даними між процесами, які працюють на одній машині. У випадку з декількома машинами, потрібно звертатися або до інших технологій, або до специфічної реалізації поточної.

У черзі кожне повідомлення має свій пріоритет і повідомлення витягуються з черги в порядку пріоритету, тобто чергу повідомлень можна назвати чергою з пріоритетами (англ. Priority Queue). Це дозволяє процесам визначати пріоритети для надсилання важливих повідомлень і гарантує, що критичні повідомлення не блокуються менш важливими повідомленнями у черзі.

Таким чином це є одним з яскравих прикладів технології напряму спілкування, який може працювати у напрямку сигналів, що є великою перевагою серед інших низькорівнених технологій.

Ця технологія забезпечує гнучкий і масштабований метод зв’язку між процесами, оскільки повідомлення можна надсилати й отримувати асинхронно, дозволяючи процесам продовжувати виконання, поки вони очікують надходження повідомлень.

Основним недоліком є те, що він може створити додаткові накладні витрати, оскільки повідомлення потрібно копіювати між адресними просторами, а чергою має керувати операційна система, щоб забезпечити її синхронізацію та узгодженість у всіх процесах.

Наочним прикладом такого представлення реалізації технології наведено на рисунку 2.4, як два процеси використовують один «message queue» для комунікації.

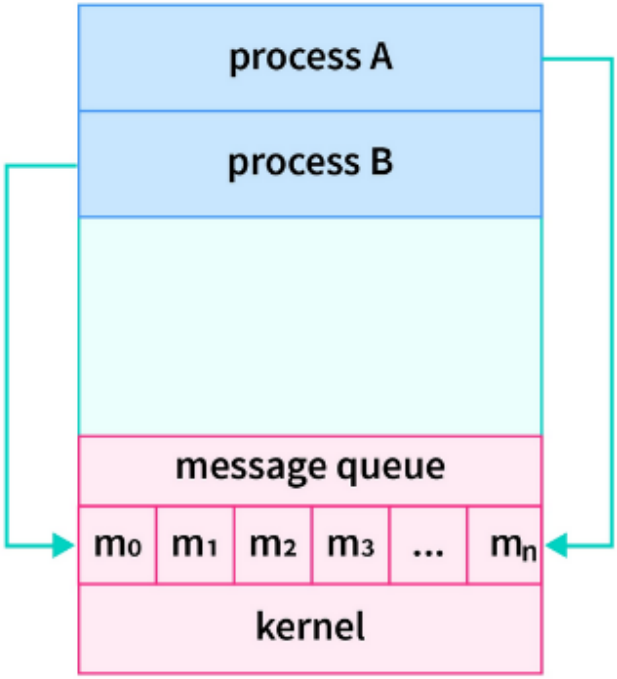

Рисунок 2.4 – Схематичне представлення технології Message Queue (за даними [11])

2.6 Огляд Shared Memory

Спільна пам'ять (англ. Shared Memory) — це технологія міжпроцесної взаємодії при якій виділяється область пам'яті, що є доступна для декількох процесів одночасно. Це дозволяє процесам взаємодіяти один з одним, читаючи та записуючи дані із оперативної області пам'яті.

Це швидкий та ефективний спосіб взаємодії процесів, але її може бути складно використовувати, якщо процеси не синхронізовані належним чином. Так чи інакше, будь яка інша технологія IPC, використовує або спільну, або додаткову пам’ять, тому спільна пам’ять – це найпряміший спосіб який тільки може існувати для передачі даних між процесами і одночасно це найнебезпечніший та найскладніший спосіб.

Декілька процесів можуть взаємодіяти через загальну пам'ять, де один процес вносить зміни одночасно, а потім ці зміни перевіряються іншими. Очевидно, що додаткові методи синхронізації чи протоколи обов’язково повинні бути додані до реалізації обміну даними. Також потрібно зазначити, що загальна пам'ять не використовується ядром.

2.7 Огляд TCP

Протокол TCP (Transmission Control Protocol) – це протокол обміну текстовими повідомленнями з архітектурою сервер–клієнт на основі сокетів. Насамперед це протокол зв’язку високого рівня, який забезпечує загальний віддалений інтерфейс між програмами, який можна легко інтегрувати в будь–яку програму з мінімальними зусиллями.

Сам протокол є стандартним для веб–середовища: серверна програма відповідає на запит, ініційований клієнтською програмою. Команди для окремих процесів можуть бути легко визначені серверною програмою та зареєстровані. Вхідне повідомлення від клієнта перевірятиметься на зареєстрований набір команд, аналізуватиметься та призначатиметься певні параметри, таким чином розробник може зосередитися на реалізації кожної команди.

TCP є повністю незалежним від платформи, стандартним і реалізованим у всіх операційних системах та багатьох інших пристроях.

У якомусь плані TCP для Unix сокетів це теж саме, що Pipes для Shared Memory – більше абстракції для зручного написання коду та тестування з додатковими обмеженнями, наприклад, у розмірі повідомлення.

2.8 Огляд D–Bus

D–Bus (Desktop Bus) – це високнева система міжпроцесного зв'язку з низькими витратами, оскільки використовує двійковий протокол і не вимагає перетворення на текстовий формат і з нього. D–Bus також розроблений, щоб

уникнути затримок та забезпечення асинхронної роботи.

D-Bus доволі простий у використанні, оскільки він працює з повідомленнями, а не потоками байтів і автоматично вирішує багато складних проблем IPC, у цьому він схожих на середнє між TCP та Message Queue.

Основний протокол D-Bus — це протокол одно–до–одного або клієнт–сервер, тобто це система, в якій один процес взаємодіє з іншим процесом, однак основним призначенням протоколу є шина повідомлень, що потенційно може зробити зв’язок стабільніше, але біль повільним.

Шина повідомлень (англ. Message Bus) – це спеціальний окремий процес, який приймає з'єднання багатьох інших процесів і пересилає повідомлення між ними.

Таким чином створюється архітектура одно–до–багатьох. Приклад схематичного представлення шини повідомлень наведено на рисунку 2.5, де наявна одна спільна шина на 5 різних процесів.

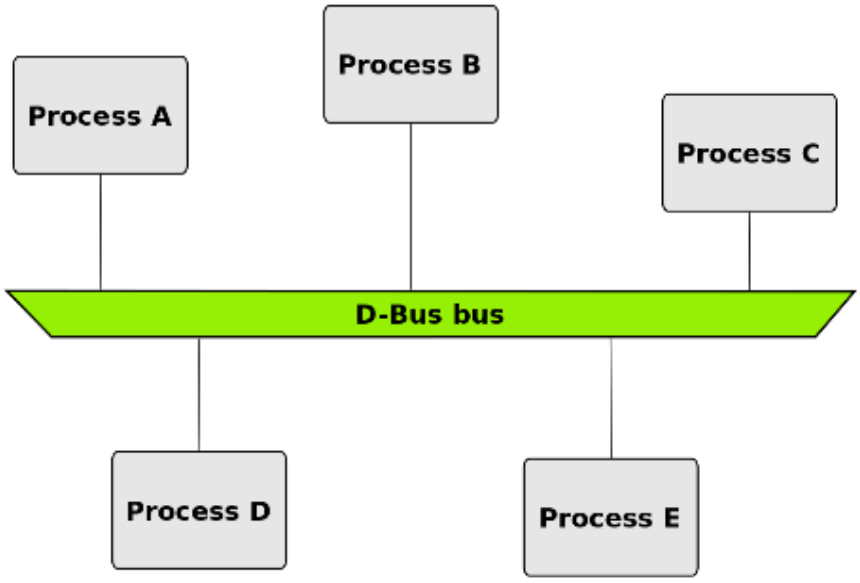

Рисунок 2.5 – Схематичне представлення шини повідомлень D–Bus (за даними [12])

D-Bus також можна використовувати для оповіщення про зміни в системі, наприклад, повідомлення, коли камеру підключено до комп'ютера.

Так як D–Bus є протоколом з багатьом реалізацій, насамперед далі буде йти

мова про реалізацію у програмному пакеті systemd, а саме – sd–dbus. Реалізація підтримує шину повідомлень та увесь протокол D–Bus.

2.9 Огляд Qt Remote Objects

Віддалені об’єкти Qt (англ. Qt Remote Objects, QtRO) – це модуль міжпроцесного зв’язку, розроблений для фрейморку Qt. Qt – це крос–платформений фреймворк для мови програмування C++, що включає у себе цілий ряд можливостей.

Цей модуль розширює існуючу функціональність Qt для легкого обміну інформацією між процесами або машинами. Очевидно, що для його використання, потрібно і серверний додаток мати написаний на Qt, але для все одно є сенс додати його до дослідження. Можливо, саме переваги QtRO зможуть стати наріжним каменем у розробці наступного серверного рішення.

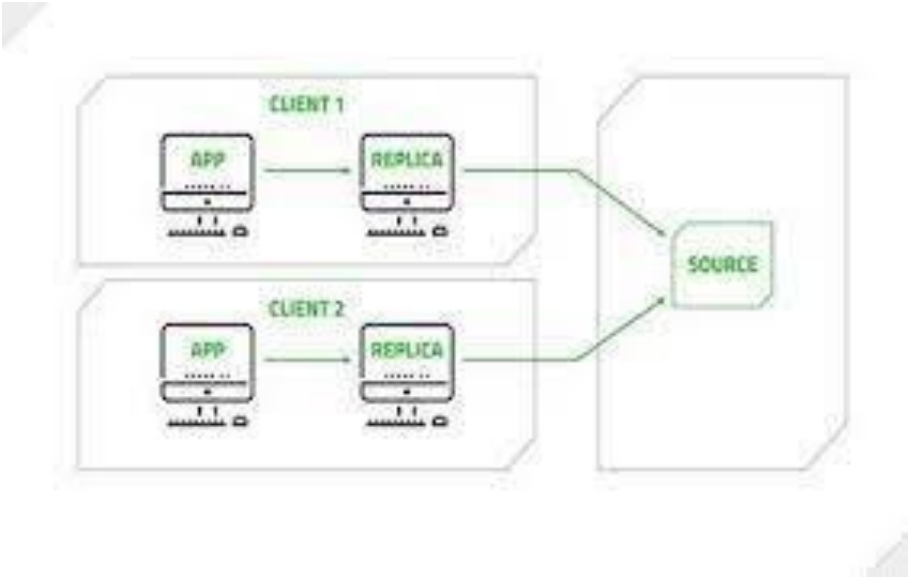

Рисунок 2.6 – Приклад використання Реплік у QtRO (за даними [13])

Метою QtRO є відповідність очікуваному API Qt, навіть якщо справжній стандартний об’єкт фреймворку знаходиться в іншому процесі. Метод, викликаний копією об’єкта, що є реплікою в QtRO, пересилається на вихідний об’єкт, що є джерелом в QtRO для обробки. Це продемонстровано на рисунку 2.6. Кожна репліка отримує оновлення джерела або зміни властивостей або видаються

сигнали. Таким чином стандартні способи комунікації в Qt використовуються окремими процесами між собою.

2.10 Перелік обраних технологій для дослідження

Серед оглянутих вище технологій не всі можуть підійти до експерименту. Потенційно кожну IPC можна розглянути та порівняти, але деякі з них або повторюють принцип роботи, або не є актуальними у більшості ситуацій. Наприклад Qt Remote Object є дуже специфічним прикладом, який неможливо інтегрувати у довільний проект. Це технологія, яка повинна стояти у центрі архітектури програмної системи, а тому і вибору не може бути.

З іншого боку, є технології, які потребують багато додаткового коду для оптимальної праці, наприклад, Shared Memory. Просто її використовувати не має можливості, через максимальну простоту технології. Потрібно створювати новий протокол, використовувати додаткову IPC для сигналювання зміни даних, перевірку цілісності пам’яті, захист від одночасного доступу к даних, тощо. Таким чином, Shared Memory може бути проаналізована у наступних ітераціях дослідження, а замість неї можна використовувати Message Queue з допоміжної бібліотеки boost. Ця технологія як раз використовує Shared Memory у центрі своєї реалізації.

Схожа ситуація і з Pipes – вони максимально прості, через що виникають труднощі з зручним використанням технології. FIFOs мають той самий принцип роботи, але вони більш прості у використані, більш легкі в аналізі та також вбудовані у Linux.

Таким чином, дослідження буде проводитися для наступних технологій міжпроцесної взаємодії: FIFOs, Unix Sockets, Message Queues, TCP, D–Bus;

3. МЕТОДИ ТА КРИТЕРІЇ ПОРІВНЯЛЬНОГО АНАЛІЗУ

3.1 Обґрунтування методів дослідження

Науковим дослідженням можна назвати вивчення явищ та процесів, аналіз впливу різноманітних факторів на них, а також вивчення взаємодії явищ з метою отримання доведених рішень з максимальним ефектом переконливості, корисних для практики та науки. Метод наукового дослідження визначає необхідність і місце застосування індукції та дедукції, аналізу та синтезу, порівняння досліджень як теоретичних, так і практичних.

Теорія у поточному дослідженні представляє собою доступну інформація про технології міжпроцесних взаємодій, а точніше, міжпроцесних обмінів даних: особливості, принципи роботи основних представників технічного напряму, їх можливо реалізація, їх переваги та недоліки при використані для серверних рішень.

Взагалі існує декілька методів проведення досліджень. У поточному – емпіричний метод був обраний задля порівняння різних технологій у обраному контексті. Саме цей метод є найкращим серед інших через необхідність реальних вимірювань, бо саме те, як поведуть себе обрані технології на практиці і вирішить їх відносне положення у дослідженні: що працює краще, а що – не дуже.

У свою чергу, методологією є сукупність методів або їхня певна послідовність, прийнята при описі дослідження. Логічний метод пізнання був обраний головним, який дозволяє використовувати аналітичні інтерпретації задля пояснення подій чи явищ, описувати проблеми та встановлювати шляхи їх рішення під час вирішення емпіричних чи теоретичних завдань.

3.2 Методи порівняння за критерієм швидкості обміну даними

Чи не найважливіша характеристика технології міжпроцесорної взаємодії – це швидкість передачі даних. На цей параметр можуть впливати дуже багато факторів під час роботи операційної системи і сама архітектура технології, але у досліджені вони були знехтувані задля спрощення результатів експериментів.

Дослідити вплив кожного аспекту на швидкість обміну даними потребувало б окремого звіту, критеріїв та методів з іншим підходом. Через те, представлені вище технології будуть проаналізовані у ближчих до «ідеальних» умовах, які майже не будуть змінюватися від альтернативи до альтернативи. Загальне співвідношення архітектури та протоколів повинно бути знайдено і за таких умов.

Усього будуть проведені декілька експериментів для кожної з альтернатив з різними параметрами запуску тестового додатку, такі як: кількість повідомлень, розмір повідомлення, розмір буферу. Самі параметри представлені у наступному розділі. Для кожного з набору будуть проведені 5 запусків, після чого буде замірений час передачі даних для кожного з них T_1, T_2, .. T_5 відповідно. Використовуючи відносну шкалу, тому що маємо час, кількісне значення критерія буде знайдено як середнє відносних швидкостей за формулою 3.1.

$$TransTime_i = \frac{1}{5}\sum_{k=0}^{5} T_{ik}, \quad 3.1)$$

де $TransTime_i$ – кількісне значення критерія альтернативи,

i –індекс альтернативи альтернатива,

k – індекс експерименту для різних наборів параметрів,

T_{ik}– середній час обміном даних для набору параметрів k.

Хоча критерій швидкості є вкрай важливим, не бажано нехтувати іншими критеріями. Умовна технологія спільної пам’яті може бути дуже швидкою, але все одно потрібно буде налагоджувати синхронізацію та свій протокол для безпечної передачі даних, тому ця швидкість буде не актуальна.

3.3 Методи порівняння за критерієм величини затримки отримання повідомлення

Існує безліч задач, які можуть бути вирішені за допомогою технологій міжпроцесної взаємодії. Велика частина таких задач – передача великих об’ємів

даних у якомога короткі проміжки часу, що може бути вирішена багатьма способами, у тому числі і буферизацією. На зворотному боці – повідомлення що не займають багато даних, але потребують чи не негайного транспортування між процесами. Буферизація для такої задачі є зайвою та навіть може завдати шкоди.

Прикладами таких повідомлень часто є сигналювання про помилки при роботі з обладнанням, потенційні загрози безпеки програмного забезпечення, аномальні зчитування датчиків температури чи напруги чи просто задачі, які потребують такий міжпроцесний зв'язок, швидкістю якого можна було б нехтувати.

Різні технології по–різному реалізують передачу повідомлень та сигнулювання про появу нових. D–Bus, наприклад, покладається на виклик процесу–сервера шини використовуючи Unix сокети за основу передачі даних, фільтрацію та пошук вірного сокета для заданого клієнта, перевірку прав доступу та інше.

Якщо це все відбувається один раз для великого повідомлення – не значні витрати, але при відправки сотень або тисяч малих, тоді можемо бачити, як загальний час передачі стає непомірно великим, чого, потенційно, не будемо спостерігати у більш простих технологіях. Таким чином це ще один критерій, за яким можна буде оцінювати ефективність IPC.

Як і для швидкості, будуть проведені п'ять експериментів з різними параметрами. Для кожного з набору будуть проведені 5 запусків, проаналізовані вихідні файли, знайдена середня затримка для повідомлення t_1, t_2, .. t_n та знайдене середнє значення для набору параметр відповідно за формулою 3.2.

$$MsgLatTime_i = \frac{1}{5}\sum_{k=0}^{5}\frac{\sum_{m}^{n} t_{mk}}{n}, \qquad 3.2)$$

де MsgLatTime $_i$ – кількісне значення критерія альтернативи,

i –альтернатива,

k – індекс експерименту для набору параметрів,

n – загальна кількість повідомлень у експерименті,

m – індекс повідомлення у експерименті для набору параметрів,

t_{mk}– величина затримки для набору параметрів k для повідомлення m.

Критерій величини затримки отримання повідомлення може бути найвпливовішим з критеріїв, але неможливо робити остаточний вибір працездатності IPC, не порівнявши його з загальною швидкістю передачі даних. Постає питання балансу затримки та пропускної здатності реалізації технології.

3.4 Методи порівняння за критерієм легкості в інтеграції

Існує два напрями розробки серверного додатку, які є важливими для дослідження: розробка додатку спочатку з обраною технологією обміну даними між процесами та інтеграція технології до існуючого коду.

У першому варіанті архітектура додатку буде відображати обрану технологію та код повинен працювати органічно та без несподіваних проблем. У другому ж варіанті можливі потенціальні великі проблеми. Так, інтеграція черги повідомлень дуже сильно відрізняється від D–Bus, а умовний QtRO взагалі може не підійти до серверного рішення через його специфічність. Потрібно з перших ітерацій мати цю технологію у центрі планування, інакше буде невигідно все переробляти під неї.

Таким чином, легкість та прозорість у інтеграції технології може вплинути на її загальну оцінку поміж інших альтернатив. Яким не була ефективна та зручна технологія, якщо її неможливо інтегрувати або це потребуватиме рефакторингу більшої частини коду – її не можна назвати найкращою альтернативою. Звичайно, цей критерій використовуватиме якісну шкалу.

Як і в випадку з безпечністю технології, формуємо множину пар: {властивість, кількість балів} і назвемо його EI. Так як це зазвичай відповідає якісній шкалі, потрібно переробити її на кількісну, тому використовуємо систему балів. Кожна за технологій буде мати свою множину EI_i. Властивості, що припадають до альтернативи надходять до множини EI_i.

Отже, кількісне значення критерія буде дорівнювати сумі балів властивостей, що притаманні альтернативі за формулою 3.3:

$$Intg_i = \sum_{k=0}^{n} \mathrm{EI}_i[k], \qquad 3.3)$$

де $\mathrm{Intg_i}$ – кількісне значення критерія альтернативи,

i –альтернатива,

k – властивість,

n – максимальна кількість властивостей, що відповідає альтернативі,

$\mathrm{EI_i}$ – під–множина властивостей альтернативи.

Множина властивостей буде наступна:

– масштабовані структури повідомлень – 2 бали;

– присутня у стандартній бібліотеці – 1 бал;

– не має додаткових залежностей – 4 балів;

– зручний API – 3 бали;

– виклики функцій не блокують поток – 5 бали;

Бали виставлені з оглядом на досвід розробки серверних застосунків. У подальших дослідженнях властивості та їх оцінки можуть бути відкориговані.

3.5 Методи порівняння за критерієм безпечності у використані

Більшість розглянутих вище технології міжпроцесної взаємодії працюють напряму чи через абстракцію з «ядром» системи, будь то спільна пам’ять напряму, чи черга повідомлень через інтерфейс. Таким чином, у разі, якщо якась помилка виникне, то це потенційно зможе дуже негативно вплинути на роботу усієї серверної машини, чого не можна допускати в жодному разі.

Річ йде про несанкціонований перезапуск операційної системи, термінове завершення випадкових процесів, пошкодження ділянок оперативної пам’яті чи видалення файлів, тощо. Не треба плутати з безпечністю даних, яка є дуже важливою у веб–розробці та одна ця тема є основою для багатьох окремих

досліджень. Існують багато методів досягання безпеки при розробці веб–застосунку [14].

Таким чином критерії безпечності у використанні полягає у тому, чи може розробник, написавши неякісний код, завдати шкоди системі та процесам окрім поточного. Цей критерій має вимірюватися шкалою найменувань, але це унеможливлює порівняння альтернатив за цим критерієм, таким чином використовуємо зразу кількісну шкалу.

Для отримання кількісного значення критерія для альтернативи, формуємо пари: {властивість, кількість балів} і назвемо їх SP. Альтернативи матимуть свою множину SP_i, що є під–множиною SP. Якщо альтернатива має окрему властивість з множини SP, вона потрапляє до множини альтернативи SP_i за формулою 3.4:

$$Sec_i = \sum_{k=0}^{n} SP_i[k], \tag{3.4}$$

де Sec_i – кількісне значення критерія альтернативи,

i –альтернатива,

n – максимальна кількість властивостей, що відповідає альтернативі,

k – властивість,

SP_i – під–множина властивостей альтернативи.

Таким чином, кількісне значення критерія буде дорівнювати сумі балів властивостей, що притаманні альтернативі.

Множина властивостей буде наступна:

– додаткові перевірки валідності даних зі сторони інтерфейсу – 5 бали;

– перевірка стану отримувача даних – 2 бали;

– існування механізму синхронізації між процесами – 5 балів;

– наявні структури повідомлень, замість роботи з байтами – 3 бали;

– буферизація даних – 2 бали;

– стійкість з’єднання між процесами – 5 балів.

Бали виставлені з оглядом на досвід розробки серверних застосунків. У подальших дослідженнях властивості та їх оцінки можуть бути змінені.

4. СТВОРЕННЯ ПРОГРАМНОЇ СИСТЕМИ ДЛЯ ПІДГОТОВКИ ЕКСПЕРИМЕНТУ

4.1 Функціональні вимоги

Для того, щоб виконати практичну частину дослідження, необхідно розробити декілька програмних застосунків з графічним інтерфейсом користувача, один з яких буде імітувати роботу сервера з надсилання повідомлень, а інший – приймати ці повідомлення та виводити статистику швидкості та якості отримання даних. Між цими застосунками потрібно реалізувати список вище зазначених міжпроцесорних технологій. Система буде працювати за принципом сервер–клієнт, де клієнт – це додаток генерації повідомлень, тому доступ у користувача буде тільки на ньому.

Генератор повідомлень повинен надавати такі можливості:

– користувач обирає кількість повідомлень, яку відправлятиме додаток;

– користувач може виставити обмеження у розмірі повідомлення та буферу;

– користувач обирає яку технологію міжпроцесної взаємодії використовуватиме система.

У такому разі, додаток, який буде прийматиме повідомлення, повинен реалізовувати наступні функції:

– користувачеві повинно показуватися стан сервера: чи підключений клієнт, чи існує вільне місце у системі для генерації звітів;

– користувач бачить шлях за яким зберігаються файли–звіти;

– користувач може скачати загальні чи специфічні для технології файлі у форматі .csv для подальшого аналізу.

Необхідно створити простий графічний інтерфейс для того, що б проводити експерименти було якомога простіше та дані, отримані в наслідок експериментів були дійсними.

4.2 Створення тестових додатків для проведення експерименту

Основною цілю проведення експерименту є оцінка критеріїв швидкості обміну повідомленнями та легкості інтеграції міжпроцесорних технологій, список яких був вище зазначений у розділі 3.

Вимірювання швидкості є чи не найважливішою характеристикою для обрання найкращої IPC. Занадто довгий процес обміну даними може сповільнити роботу серверної частини доволі суттєво, що навіть кінцевий користувач помітить перебої у роботи веб–додатку чи мобільного застосунку. Гарно налагоджене використання IPC є непомітним ні для рядового програміста, ні для користувача, але помилки у виборі технології можуть стати критичними.

Мета цього вимірювання у умовах приближених до реальних – забезпечення найпрозоріший процес обміну даними між процесами незалежно від загрузки на обчислювану машину, що забезпечить безперебійний досвід користувача користуванням програмної системи.

Вимірювання легкості інтеграції технології на перший погляд можна знехтувати, але за інших рівних, набагато швидше та простіше використовувати інструменти які вже є у стандартних бібліотеках, чи технології, які обзавелися безліччю інструкцій та порад. Для користувачів буде непомітно напряму проблеми з інтеграцією, але це опосередковано впливає на якість коду, його стресостійкість, час виконання, кількість потенціальних проблем, тощо. Усі гроші, що можуть бути направлені на покращення рівня коду, могли б були направлені на вивчена UI/UX графічного інтерфейсу, чи реклами програмної систему у цілому, хоча це виходить за рамки дослідження.

Метою вимірювання легкості інтеграції технологій міжпроцесної взаємодії є полегшення розробки програмного застосунку та унеможливлення появи помилок у роботі коду.

Для оцінки швидкості роботи різних технологій, потрібно зробити низку вимірювань для кожної з них, а саме:

– час та розмір останнього переданого повідомлення;

– середній час повідомлення за весь час роботи додатку;

– середній час повідомлення, залежно від їх середнього розміру;

– весь час передачі повідомлень та загальний їх розмір у кілобайтах.

Слід зазначити, що час передачі повідомлення складається з двох частин: запису та зчитування. В залежності від технології, можуть бути різні накладні витрати у часі та пам’яті на це.

Таким чином матимемо усі необхідні дані для подальшого аналізу швидкодії різних технологій міжпроцесної взаємодії, а для оцінки легкості у інтеграції буде заповнена таблиця з критеріями представленими у розділі 4.5, враховуючи досвід програмування програмної системи.

Для проведення таких експериментів, буде створено два програмних додатки на основі фреймворку Qt. Qt – це кросплатформний фреймворк розробки програмного забезпечення. Він широко використовується для створення різноманітних додатків, у тому числі графічних інтерфейсів користувача, вбудованих системних програм, мобільних додатків тощо. Qt надає набір інструментів і бібліотек, які спрощують розробку програм і роблять їх кросплатформними, хоча у нашому разі все буде базуватися на Linux.

Більшість коду буде написано на C++, використовуючи стандартну бібліотеку STL, до якої входитимуть модулі <chrono>, <sockets>, <threads> та інші. Це модулі покриють роботу з заміром часу, розміру повідомлень, стабільності роботи додатків, логування тощо. Вони побудують каркас додатку, до якого буде додано модулі для кожної з IPC.

До каркасу потрібно буде додати:

– підтримку стандартні графічні елементи задля надання можливості користувачеві виставляти обмеження для експерименту;

– загальні класи, що будуть використовуватися для кожної з IPC, такі як: таймери, логування, запис даних до файлів;

– шаблон для додавання нових технологій міжпроцесерної взаємодії, задля потенційного повторення експерименту з іншими вхідними даними, параметрами чи альтернативами;

– можливість відображення користувачеві статусу передачі повідомлень у реальному часі;

Основним класом подімлень є клас ipc_msg. Частково його код виглядає наступним чином:

```
class ipc_msg
{
public:
    ipc_msg (int i = 0, std::string n = "")
        : id(i), data(n)
        , time_created(0), time_transfered(0)
        , time_diff(0)
        {}

    uint16_t id;
    std::string data;
    uint64_t time_created;
    uint64_t time_transfered;
    uint64_t time_diff;

    void setTimeCreated();
    void setTimeTransfered();
    void setTimeDiff();
}
```

У собі він зберігає ідентифікаційний номер, саме повідомлення та три часові величини, які необхідні для експерименту за замірюванням витрат на їх передачу. Фукнція setTimeCreated() викликається клієнтською частиною перед відправкою повідомлення на сервер, функції setTimeTransfered() та за нею setTimeDiff() викликаються сервером після отримання та десеріалізації повідомлення. Приклад функції setTimeCreated() наведений далі.

Як бачимо з фрагменту коду нижче, використовується Unix–час [15] у мікросекундах. Мілісекунди не показали би жодного результату, через швидкість роботи IPC та малих розмірів повідомлення, а наносекунди мали б надто великі значення. Код повторюється у setTimeTransfered().

```
void ipc_msg::setTimeCreated()
{
    auto since_epoch =
            std::chrono::system_clock::now().time_since_epoch();
```

```
    time_created =
       std::chrono::duration_cast
              <std::chrono::microseconds>(since_epoch).count();
}
```

Іншим базовим класом є ipc_base. Це абстрактний клас, від якого унаслідуються усі класи з реалізаціями конкретних технологій. QObject, Q_OBJECT, signals, slots – це все особливості фреймворку Qt [13].

```
class ipc_base : public QObject
{
    Q_OBJECT

public:
    ipc_base() {}
    virtual ~ipc_base() {}

    virtual bool setupServer() {}
    virtual bool setupClient() {}
    virtual bool write(ipc_msg& ipc_message) {}

    uint64_t maxBufferSize = MAX_BUFFER;
    uint64_t maxMsgLength = 0;

signals:
    void readReady(ipc_msg& ipc_message);

protected slots:
    virtual bool read() {}
};
```

У цілому, використовуємо шаблон «Спостерігач» [16]. Сервер чекає від ipc_base нових повідомлень, після чого реагує на них.

На відміну від write() яку клієнтська сторона може викликати у будь який час, сервер не може викликати read() через специфічність міжпроцесного зв’язку, тому ця функція скрита від зовнішнього використання. Сервер повинен підписатися на сигнал readReady задля отримання повідомлень.

Алгоритм при встановлені зв’язку наступний: серверна частина стартує, викликає setupServer() для IPC що будуть використовуватися, запускає клієнт, клієнт викликає setupClient() таким ж чином, клієнт починає генерацію та передачу повідомлень. Приклад функції setupClient для черги повідомлень наведений нижче:

```
bool ipc_msgq::setupServer()
{
    boost::interprocess::message_queue::remove(msgq_name.c_str());
    msg_queue = std::make_shared
          <boost::interprocess::message_queue>
        (
            boost::interprocess::open_or_create,
            msgq_name.c_str(),
            maxBufferSize / (maxMsgLength + IPC_MSG_DATA_EXTRA) + 1,
            maxMsgLength + IPC_MSG_DATA_EXTRA
        );

    auto readLoop = [this]()
    {
        while(!stop)
        {
            read();
        }
    };
    QtConcurrent::task(std::move(readLoop)).spawn();

    return true;
}
```

Особливість цієї реалізації є створення додаткового потоку задля постійного зчитування черги. Черга повідомлень є блокуючою технологією, тому виклик read() у основному потоці повністю заморозив би сервер, поки не прийде нове повідомлення, а так як це один з багатьох одночасних каналів зв’язку – усі інші не змогли б передати жодного байту. У цьому ж разі очевидні переваги шаблону «Спостерігач».

Як було сказано раніше, сервер сам запускає клієнт для кращої автоматизації експериментів. Таким чином сервер автоматично передає параметри експерименту та IPC що будуть тестуватися. Код запуска, конвертації статусів технологій у флаг та передачі даних наведений нижче:

```
QTimer::singleShot(2000, this, [this]()
{
    QString program = "../Client/Client";

    QStringList arguments;
    arguments << "-n" << QString::number(maxMsgNumber);
    arguments << "-l" << QString::number(maxMsgLength);
    arguments << "-b" << QString::number(maxBufferSize);
```

```
    int ipcBitFlag = 0;
    if (statusIpcs["fifo"])
       ipcBitFlag |= 1 << 0;
    if (statusIpcs["msgq"])
       ipcBitFlag |= 1 << 1;
    if (statusIpcs["dbus"])
       ipcBitFlag |= 1 << 2;
    if (statusIpcs["unix"])
       ipcBitFlag |= 1 << 3;
    if (statusIpcs["tcp_"])
       ipcBitFlag |= 1 << 4;
    arguments << "-i" << QString::number(ipcBitFlag);

    QProcess *myProcess = new QProcess(this);
    myProcess->start(program, arguments);
});
```

Клієнт використовує аргументи інтерфейсу командного рядка для усіх параметрів. Для передачі масиву технологій використовується одна цілочисельна зміна, яка грає роль побітового прапора: кожен біт відповідає за одну технологію. Прочитуючи по біту зі змінної можемо отримати даних, які звичайно пакують до восьми окремих змінних.

Це може виглядати не дуже зрозуміло, але є доволі ефективним засобом економії пам’яті та кількості параметрів, особливо коли використовується інтерфейс командного рядка. Код наведений зі серверної сторони, але клієнт аналогічно зчитує інформацію. Усі операції з операційною систему дуже зручно упаковані у класи фрейморком Qt.

```
ipc_type,overall_transfer_time(µs),message_number,message_data_length,buffer_size(byte)
dbus,795352,10000,500,1500
fifo,461969,10000,500,1500
msgq,477738,10000,500,1500
tcp_,536097,10000,500,1500
unix,2740423,10000,500,1500
```

Рисунок 4.1 – Зміст конфігураційного файлу (створено самостійно)

Що ж до отримання експериментальний даних, сервер створює .csv файли для кожного з IPC та один загальний конфігураційний файл.

Як бачимо на рисунку 4.1 сервер записує тип IPC, загальний час обмін даними та необхідні параметри, такі як кількість повідомлень, їх розмір та розмір буферу. Окрім цього маємо рисунок 4.2 зі структурами повідомлень, де останні три

числа – мікросекунди у форматі часу Unix. Саме ці дані і є результатом дослідження.

```
id,data,time_created(μs),time_transfered(μs),time_diff(μs)
1,TEST_DATA_TEST_DATA_1,1716876484950894,1716876484950906,12
2,TEST_DATA_TEST_DATA_2,1716876484950911,1716876484950932,21
3,TEST_DATA_TEST_DATA_3,1716876484950917,1716876484950936,19
```

Рисунок 4.2 – Зміст файлу з отриманими даними по IPC (створено самостійно)

На кожен тест приходяться 2 * кількість запущених технологій + 1 файлів та їх загальний розмір може досягати гігабайтів, при екстремальних значеннях параметрів. Цього можна було б запобігти, просто не додаючи самі дані, які передає клієнт, до файлі, а просто писати час та розмір, але для перевірки тестового додатку на працездатність, зберігання конкретних даних, які отримав сервер може сильно допомогти при виникненні несподіваних проблем з якістю зв’язку між сервером та клієнтом.

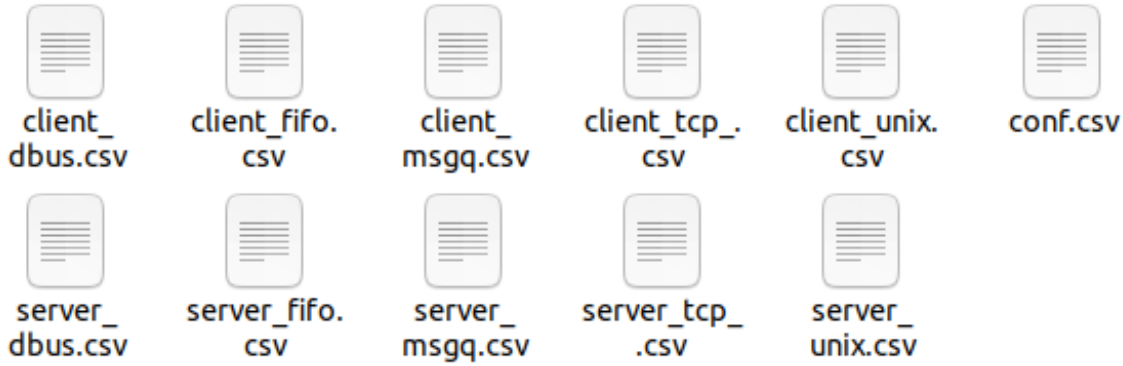

Рисунок 4.3 – Структура папки з файлами, згенерованими тестовим додатком (створено самостійно)

На рисунку 4.3 зображено як виглядає результат одного запуску тестового додатку. Були використані усі 5 технологій міжпроцесної взаємодії. Імена файлів є статичними для легкого аналізу їх через стороні застосунки, як, наприклад, скрипт Python.

5. ОПИС ПРОВЕДЕНИХ ДОСЛІДЖЕНЬ

5.1 Дослідження швидкості обміну даними

Після того як тестовий додаток був створений та перевірений на стабільність обміну даними між процесами, вимірюємо швидкість їх передачі для усіх технологій міжпроцесної взаємодії, що були обрані у розділі 2.10, а саме: FIFOs, Unix Sockets, Message Queues, TCP, D–Bus.

Експеримент складається з 25 повних запусків тестового додатку: по 5 разів на кожну групу параметрів. Використані параметри наведені у таблиці 5.1. Ці параметри були обрані експериментально, як середнє за часом проведення експериментів та використання ресурсів системи, на якій проводилися експерименти. Наприклад, виконання останнього набору тестів потребує декілька гігабайтів вільного місця на диску через великі об’єми даних.

Таблиця 5.1 – Параметри передачі даних

Назва набору тестів	Кількість повідомлень	Розмір повідомлення (кількість символів)	Розмір буферу (байти)
Початковий	10000	500	1500
Кількість повідомлень	25000	500	1500
Розмір повідомлення	10000	1500	4000
Розмір буферу	10000	500	15000
Всі параметри	30000	2000	15000

Розмір буферу потрібен бути більше за розмір повідомлення приблизно на 1000, що було знайдено експериментально. Це потребує специфікація реалізації TCP сокетів, тому що для усіх інших технологій потенційно можливо використання

буферу максимально приближеному до розміру самого повідомлення.

Таким чином алгоритм проведення експерименту для кожного набору даних наступний:

– обрати параметри для набору тестів;

– запустити тестовий додаток 5 разів;

– зібрати тестові дані, що знаходяться у згенерованих .csv файлах для усіх IPC;

– обчислити середні показники для 5 наборів даних;

– додати результативні значення до таблиці 5.1.2.

Для дослідження швидкості обміну даними достатньо інформації записаної до конфігураційного файлу. Після 25 запусків системи та знаходження середніх значень, маємо таблицю 5.2.

Таблиця 5.2 – Усереднені швидкості передачі даних технологіями (Kb/ms)

	Початковий	Кількість повідомлень	Розмір повідомлення	Розмір буферу	Всі параметри
D–Bus	5.92	6.75	10.38	5.72	11.00
FIFO	9.29	10.49	13.44	9.36	13.19
Message Queue	8.53	9.07	12.15	9.10	13.27
TCP	9.47	9.81	13.02	9.36	13.55
Unix Sockets	6.42	8.40	11.94	6.33	12.65

Отримані дані у вище не можна порівнювати між собою через вагому різницю між параметрами експериментів. Ціль такого підходу – подивитися на технології у різних ситуаціях та з різними обмеженнями, тому що одна й та же технологія може дуже по різному себе поводити у різних системах. Для того, що б можна було порівнювати дані між різними наборами тестів, потрібно нормалізувати.

Для нормування оцінок використовуємо формулу «з урахуванням min–max». Нормалізуємо за стовпцями, тобто за наборами тестів за формулою 5.1:

$$f = \frac{f_i - f_{min}}{f_{max} - f_{min}}, \quad (5.1)$$

де f – нормалізоване значення,

f_i – оригінальне значення,

f_{min} – мінімальне значення у наборі,

f_{max} – максимальне значення у наборі.

Таким чином маємо таблицю 5.3 за нормалізованими результатами експерименту за дослідженнями швидкості обміну даними.

Таблиця 5.3 – Нормалізовані показники швидкості передачі даних

	Початковий	Кількість повідомлень	Розмір повідомлення	Розмір буферу	Всі параметри
D–Bus	0.00	0.00	0.00	0.00	0.00
FIFO	0.95	1.00	1.00	1.00	0.86
Message Queue	0.74	0.62	0.58	0.93	0.89
TCP	1.00	0.82	0.86	1.00	1.00
Unix Sockets	0.14	0.44	0.51	0.17	0.65

Маючи показники приведені усі до однакової нормалізації, можна знайти середнє значення для кожної з технології за формулою 5.2:

$$f_{avg} = \frac{\sum_{k=0}^{n} f_k}{n}, \quad (5.2)$$

де f_{avg} – середнє значення,

k – індекс значення,

n – кількість наборів тестів, у даному випадку – 5,

f_k – нормалізоване значення.

Результати обчислень критерія наведені у таблиці 5.4.

Таблиця 5.4 – Векторний опис критерія швидкості обміну даними

D–Bus	FIFO	Message Queue	TCP	Unix Sockets
0.00	0.96	0.75	0.94	0.38

5.2 Дослідження величини затримки отримання повідомлення

Задля отримання даних щодо величини затримки не потрібно проводити додаткові експерименти через те, що усі необхідні дані уже наявні після дослідження швидкості обміну даними. Джерело інформації про затримки – файли, що створює сервер для кожного з запуску тестів для кожної з технологій. Таким чином маємо 25 файлів з даними, об'єми яких варіюються між 5 Mb та 50 Mb кожен. Загалом файли мають доволі великий об’єм, але у сьогоднішні часи це не повинно бути проблемою. Така надмірність може зекономити час у випадку, коли щось пішло не так, або автоматизувати перевірку цілісності передачі даних.

Клієнт запису час генерації повідомлення, відсилає його, сервер отримує, записує час отримання та різницю між позначками часу, після чого ця інформація потрапляє до файлу.

Алгоритм роботи з даними схожий на алгоритм з дослідженням швидкості, але ускладняється пошуком середньої затримки для одного повідомлення, тому що повідомлень було запущено багато тисяч під час експерименту і у кожного є свої часові позначки.

Обчислювати мегабайти інформації вручну не має сенсу, тому був створений малий скрипт на Python за допомогою модуля pandas [17]. Скрипт зчитує мегабайти даних, робить обчислення та виводить результати у термінал. Так як це проводитиметься пару разів, немає потрібності у генерації додаткових файлів..

Приклад проміжний обчислень для одного файлу, у даному випадку для Tcp, наведений на рисунку 5.1.

```
server_tcp_.csv
FullTime:  529012
MsgLength:  500
MsgNumber:  10000

BytesPerUs:  9.451581438606308
KBytesPerMs:  9.230059998638973
MsgMeanTimeUs:  782.3247
BufferSizeKb:  14.6484375
```

Рисунок 5.1 – Проміжний результат роботи допоміжного скрипта на Python (створено самостійно)

Остаточні результати експерименту, як і з дослідженням швидкості обміну даними, наведені у таблиці 5.5.

Таблиця 5.5 – Усереднені величини затримки отримання повідомлення (µs)

	Початковий	Кількість повідомлень	Розмір повідомлення	Розмір буферу	Всі параметри
D–Bus	364.02	384.50	331.30	376.38	425.52
FIFO	43.54	40.77	55.66	43.07	59.44
Message Queue	78.96	67.28	75.55	53.23	53.76
TCP	801.16	926.44	857.59	833.94	1088.38
Unix Sockets	74.76	49.41	77.67	87.12	82.92

На відміну від попереднього дослідження, тут зразу бачимо велику різницю між показниками технологій. TCP та D–Bus у декілька разів показують гірші значення за альтернативи, чому буде пояснення у розділі с аналізом отриманих даних. Наступний крок – використання формули з попереднього дослідження нормалізації даних. Результат наведений у таблиці 5.6.

Таблиця 5.6 – Нормалізовані величини затримки отримання повідомлення

	Початковий	Кількість повідомлень	Розмір повідомлення	Розмір буферу	Всі параметри
D–Bus	0.42	0.39	0.34	0.42	0.36
FIFO	0.00	0.00	0.00	0.00	0.01
Message Queue	0.05	0.03	0.02	0.01	0.00
TCP	1.00	1.00	1.00	1.00	1.00
Unix Sockets	0.04	0.01	0.03	0.06	0.03

Використовуємо інверсію даних за формулою 5.3, бо це затримка:

$$f = 1 - f_i, \qquad 5.3)$$

де f – інвертоване значення,

f_i – нормалізоване значення.

Результати перетворення можна побачити у таблиці 5.7.

Таблиця 5.7 – Інверті величини затримки отримання повідомлення

	Початковий	Кількість повідомлень	Розмір повідомлення	Розмір буферу	Всі параметри
D–Bus	0.58	0.61	0.66	0.58	0.64
FIFO	1.00	1.00	1.00	1.00	0.99
Message Queue	0.95	0.97	0.98	0.99	1.00
TCP	0.00	0.00	0.00	0.00	0.00
Unix Sockets	0.96	0.99	0.97	0.94	0.97

Як результат дослідження величини затримки, маємо результуючу таблицю 5.8 с записаними значеннями для кожної з альтернатив.

Таблиця 5.8 – Векторний опис критерія затримки повідомлення

D–Bus	FIFO	Message Queue	TCP	Unix Sockets
0.61	1.00	0.98	0.00	0.97

5.3 Дослідження легкості в інтеграції

Якщо технологія є дуже ефективною, але її тяжко включити до проекту, підтримувати та аналізувати на предмет помилок або неточностей, то стає питання, чи насправді вона «ефективна». Під час розробки було проаналізована легкість у інтеграції міжпроцесорних технологій за критеріями представлених у розділі 3.4.

Результати дослідження, отриманих експериментальним шляхом та відкориговані у процесі ознайомлення з джерелами про дані технології, представлені у таблиці 5.9.

Таблиця 5.9 – Властивості легкості в інтеграції технології

	D–Bus	FIFO	Message Queue	TCP	Unix Sockets
Масштабовані структури повідомлень	Так	Ні	Ні	Так	Так
Присутня у стандартній бібліотеці	Ні	Так	Ні	Так	Так
Не має додаткових залежностей	Ні	Так	Ні	Так	Так
Зручний API	Так	Ні	Ні	Так	Так
Виклики функцій не блокують поток	Так	Ні	Ні	Так	Так

Застосовуємо коефіцієнти з розділу 3.4 та маємо таблицю 5.10 з обчисленою сумою балів для кожної з альтернатив.

Таблиця 5.10 – Кількість баллів за властивості безпечності у альтернатив

	D–Bus	FIFO	Message Queue	TCP	Unix Sockets
Масштабовані структури повідомлень	2	0	0	2	2
Присутня у стандартній бібліотеці	0	1	0	1	1
Не має додаткових залежностей	0	4	0	4	4
Зручний API	3	0	0	3	3
Виклики функцій не блокують поток	5	0	0	5	5
Сума балів:	10	5	0	15	15

Використовуючи формулу 5.1 з дослідження швидкості знаходимо нормалізовані значення суми балів для кожної з технології. Результати обчислень знаходять у таблиці 5.11.

Таблиця 5.11 – Векторний опис критерія легкості в інтеграції

D–Bus	FIFO	Message Queue	TCP	Unix Sockets
0.66	0.33	0	1	1

5.4 Дослідження безпечності

У процесі розробки тестового програмного додатку виявлялися проблеми з різними технологіями та результатами їх вирішення, поглибленим ознайомленням з кодовой базою технологій та аналізом документації реалізації конкретних IPC є таблиця 5.12 з наведеними критеріями з розділу 3.5. Бачимо, що кожна з технологій використовує буферизацію даних у тому чи іншому вигляді, але принцип її роботи буде відрізнятися у кожній з них. Message Queue, наприклад, зберігає умовну кількість повідомлень, коли як Tcp користується максимальним розміром пакету даних, а для FIFO – це максимальний розмір байтів, які не переповнюють дескриптор.

Таблиця 5.12 – Властивості безпечності технологій

	D–Bus	FIFO	Message Queue	TCP	Unix Sockets
Додаткові перевірки валідності даних зі сторони інтерфейсу	Так	Ні	Ні	Ні	Ні
Перевірка стану отримувача даних	Так	Ні	Ні	Так	Так
Наявні структури повідомлень, замість роботи з байтами	Так	Ні	Ні	Ні	Ні
Буферизація даних	Так	Так	Так	Так	Так
Стійкість з’єднання між процесами	Так	Ні	Ні	Так	Так

Після застосування коефіцієнтів з розділу 3.5 отримуємо дані у таблиці 5.13 для альтернатив. Також додаємо суму балів до таблиці для зручності у подальшому пошуку векторного опису альтернативи. Більше всього балів отримує D–Bus. Ця технологія використовує додаткові процеси для правильної роботи, тому в неї є ресурси для підвищеною безпечністю. Найменше всього мають FIFO та Message Queue, які є низькорівневими технологіями.

Таблиця 5.13 – Кількість баллів за властивості безпечності у альтернатив

	D–Bus	FIFO	Message Queue	TCP	Unix Sockets
Додаткові перевірки валідності даних зі сторони інтерфейсу	5	0	0	0	0
Перевірка стану отримувача даних	2	0	0	2	2
Наявні структури повідомлень, замість роботи з байтами	3	0	0	0	0
Буферизація даних	2	2	2	2	2
Стійкість з’єднання між процесами	5	0	0	5	5
Сума балів:	15	2	2	9	9

Як і з дослідженням легкості в інтеграції, потрібно нормалізувати суму балів технологій для порівняння результатів з іншими критеріями. Для цього використовуємо ту ж саму формулу 5.1 та записуємо результати обчислень до таблиці 5.14.

Таблиця 5.14 – Векторний опис критерія безпечності

D–Bus	FIFO	Message Queue	TCP	Unix Sockets
1	0.13	0.13	0.6	0.6

Бачимо, що D–Bus сильно виграє між інших альтернатив, але загалом бали розподілилися по усій шкалі.

Наявних результатів повністю вистачає для формування висновку щодо ефективності роботи представлених технологій міжпроцесної взаємодії, допускаючи їх використання у серверних додатках або середовищах приближених до них.

6. АНАЛІЗ РЕЗУЛЬТАТІВ ДОСЛІДЖЕНЬ

Під час проведення експерименту було досліджено 5 технологій міжпроцесної взаємодії за чотирма критеріями, що потребувало 25 запусків тестового додатку та аналізу файлів з вихідними даними.

За критерієм швидкості бачимо середні значення у таблиці 5.2 та загальні нормалізовані у таблиці 5.4, візуалізація яких представлена на рисунку 6.1.

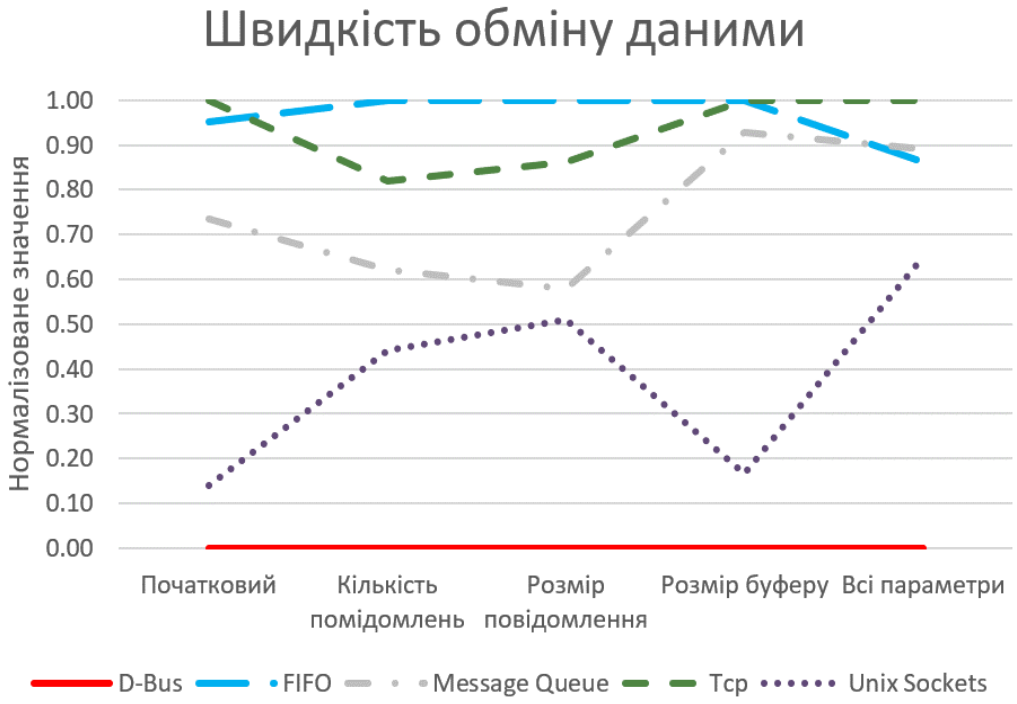

Рисунок 6.1 – Візуалізація відносної швидкості передачі даних технологіями (створено самостійно)

Як бачимо з рисунку, остаточно можна сказати, що D–Bus програє у будь якому випадку, Message Queue та Unix Sockets є не найстабільнішими та лідирують FIFO та Tcp. FIFO є чи не найпростішим методом IPC, не маючи жодних зайвих операцій у своїй реалізації, не дивно що воно є одним з найшвидших, але Tcp, в теорії, повинно було бути повільнішим чи на рівні з Unix Sockets, бо у їх основі лежать одна ідея – сокети. Можливо подальший аналіз розкриє цей момент у більших деталях.

Щодо D–Bus, не є сюрпризом що ця технологія є найповільнішою, бо кожне

повідомлення, яких під час експерименту було більше десяти тисяч, проходило не тільки через клієнт на сервер, а ще й через процес–сервер–d–bus, що й займається переадресацією повідомлень різним процесам. Окрім того, валідація повідомлень, перевірка прав доступу та відкритість каналів витрачає додатковий час, результати чого ми і бачимо на рисунку 6.1.

Це порівняння нормалізованих значень, а щодо абсолютних, можна подивитися на рисунок 6.2. D–Bus у середньому лишу у півтора рази повільніший за FIFO, щодо усіх інших – розрив ще менше, тому можна сказати, що усі технології мають більш–менш однакову пропуску здатність.

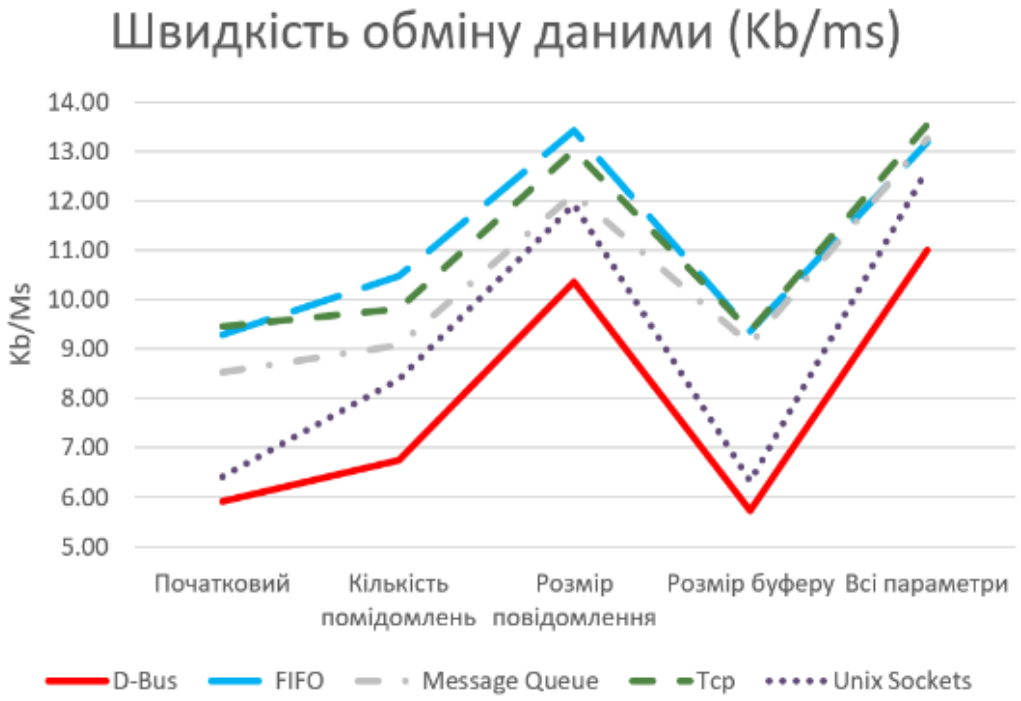

Рисунок 6.2 – Візуалізація абсолютної швидкості передачі даних технологіями (створено самостійно)

Слід тільки зазначити, що вона значно підвищується при збільшенні розміру самого повідомлення, що говорить про те, що основна частина часу йде саме на переключення з повідомлення на повідомлення, а не транспортування даних. Таким чином відправка одного пакету у 1 мегабайт буде швидше, аніж відправка 100 пакетів по 10 кілобайт. Відправка повідомлення включає до себе багато системних викликів, змін контексту, перевірок, створення додаткових структур, виділення

тимчасової пам’яті, тому не дивно, що зробити увесь круг один раз швидше, аніж декілька разів.

Розібравшись з швидкістю, перейдемо до величин затримки отримання повідомлень. Результати експерименту бачимо у таблицях 5.7 та 5.8. Візуалізація нормалізованих значень бачимо на рисунку 6.3.

Слід зазначити, що дані були взяті з таблиці 5.6, тобто вони ще не є інвертними – на рисунку Tcp має найбільшу затримку, а FIFO – найменшу, а не навпаки. Розберемо, що бачимо на рисунку.

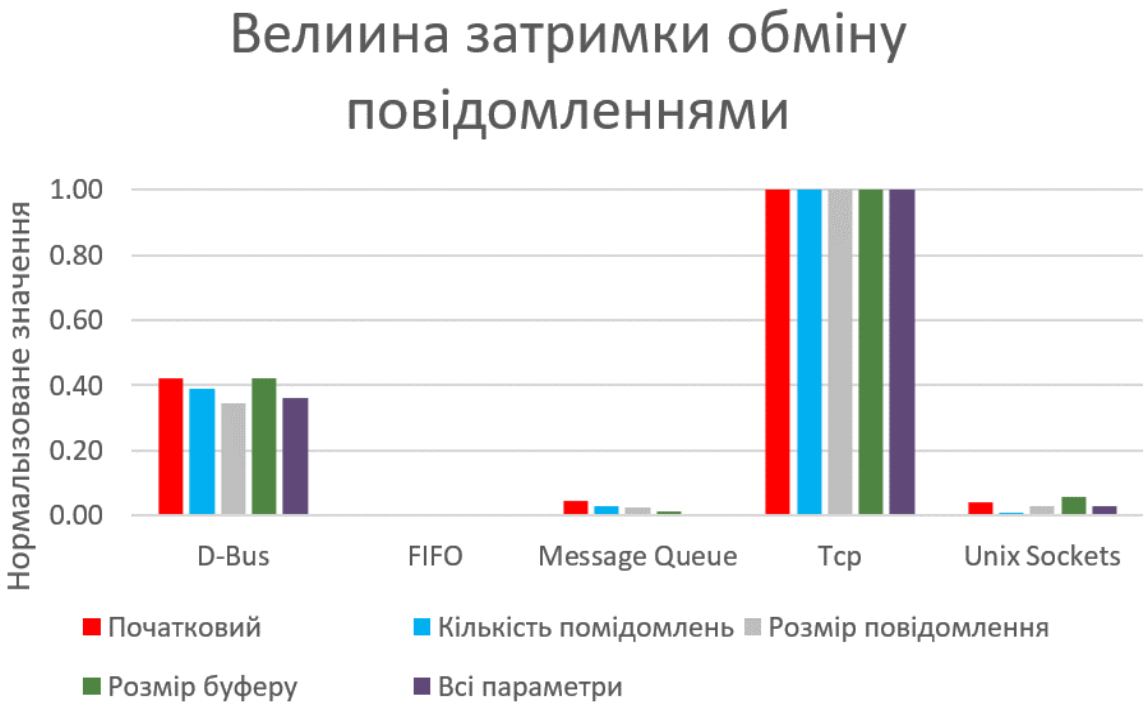

Рисунок 6.3 – Візуалізація відносної величини затримки обміну повідомленнями (створено самостійно)

Найменша затримка у FIFO, Message Queue та Unix Sockets. Ці технології використовують тільки два процеси та їх архітектура має на увазі роботу тільки локально на одній машині, тому не дивно, що затримка у них найменша. Щодо D–Bus, як і було сказано, усі повідомлення проходять через третій процес та додаткові перевірки, тому затримка більша, але найповільнішим виявився Tcp.

Через те, що ця технологія створена для спілкування по мережі інтернет, має сенс спочатку накопичити в буфері достатньо повідомлень, а потім їх всіх разом

відправити. Результатом буде велика затримка для кожного з повідомлень, але загальна висока швидкість передачі даних, що і бачимо на рисунку 6.3. Таким чином, Tcp платить за високу швидкість високою затримкою.

Це щодо нормалізованих значень, що ж до абсолютних, бачимо їх на рисунку 6.4. Усі виміри були створенні у мікросекундах, що бачимо з рисунку. На відміну від швидкості передачі даних, графіки затримки майже однакові. Різниця затримок між FIFO та TCP – у середньому 800 мікросекунд, або FIFO має меншу затримку ніж Tcp у 20 разів.

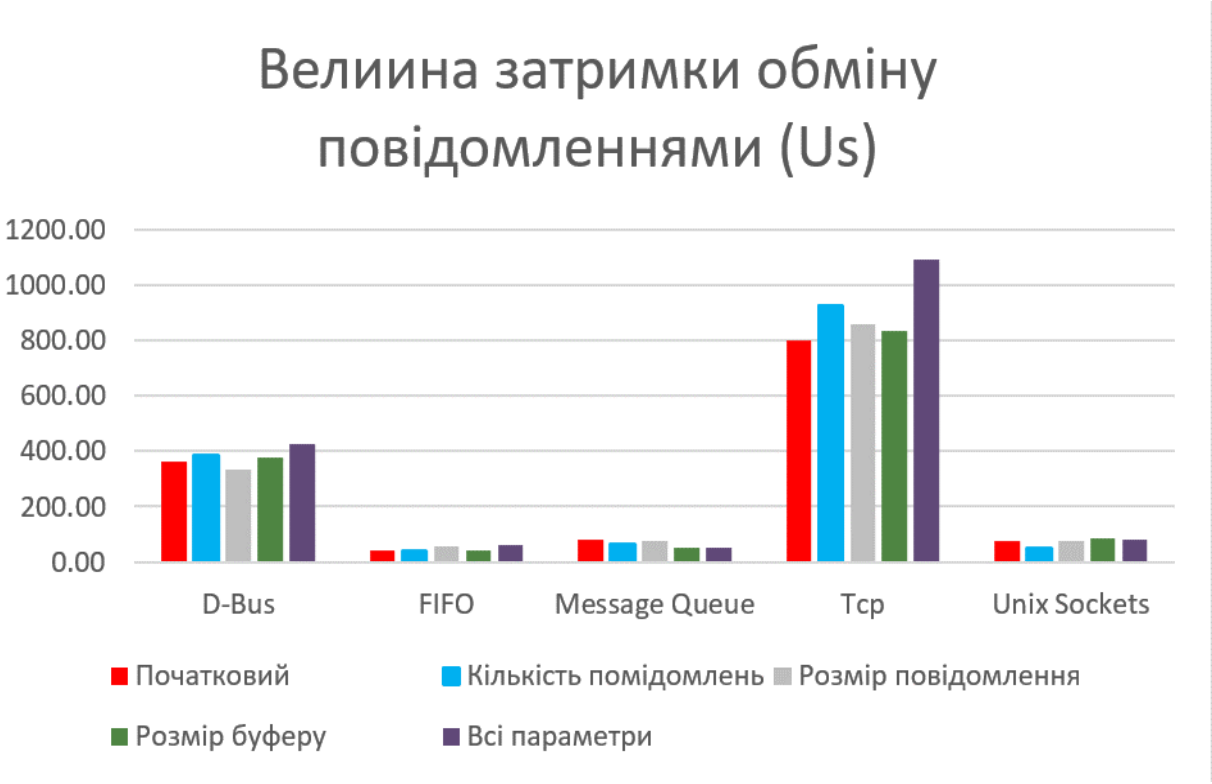

Рисунок 6.4 – Візуалізація абсолютної затримки обміну повідомленнями (створено самостійно)

Таким чином, можна зробити висновок, що FIFO, Message Queue та Unix Sockets слід використовувати, коли треба якомога швидше передати якесь критичне повідомлення чи команду, а Tcp для стабільного обміну інформації як то інформаційних повідомлень, наборів даних, можливо файлів. Є можливість досягти Unix Sockets результатів Tcp, якщо використовувати більший буфер та додати його обробку до коду, що не було зроблено у тестовому додатку.

Проаналізувавши найголовніші критерії дослідження, можемо перейти до загального огляду технологій. Використовуючи дані з таблиць 5.4, 5.8, 5.11 та 5.14, будуємо таблицю 6.1.

Таблиця 6.1 – Векторний опис технологій за критеріями

	D–Bus	FIFO	Message Queue	TCP	Unix Sockets
Критерій швидкості обміну даними	0	0.96	0.75	0.94	0.38
Критерій величини затримки отримання повідомлення	0.61	1	0.98	0	0.97
Критерій легкості в інтеграції	0.66	0.33	0	1	1
Критерій безпечності	1	0.13	0.13	0.6	0.6
Всього:	2.27	2.42	1.86	2.54	2.95

Маючи векторний опис альтернатив, можемо визначити множину Парето. При визначені її, оцінюємо, чи є альтернативи, які гірші за будь яким критерієм. При вирішенні поточної задачі, було виявлено «Message Queue», що не надійшла до множини недомінуючих альтернатив. Це не є показником, що альтернатива не має жодного оптимального використання у серверних додатках, але у поточному дослідженні с поточними критеріям – вона менш ефективна за інші альтернативи. Таким чином, маємо таблицю 6.2, як результат виключення невдалих альтернатив і дали працюємо з чотирма альтернативами, а саме: D–Bus, FIFO, TCP та Unix Sockets.

Для остаточного розрахунку корисності альтернатив, використовуємо адитивну лінійну згортку з ваговими коефіцієнтами за 4 критеріями та 4 альтернативами. Враховуючи, що величина затримки повідомлень знаходиться на порядку мікросекунд, а при використані справжнього серверного апаратного забезпечення, можливо і наносекунд, ваговий критерій для величини затримки не повинен бути високий, але потенційно, для іншого типу дослідження, саме цей критерій буде найважливішим.

Таблиця 6.2 – Векторний опис з множиною домінуючих альтернатив

	D–Bus	FIFO	TCP	Unix Sockets
Критерій швидкості обміну даними	0	0.96	0.94	0.38
Критерій величини затримки отримання повідомлення	0.61	1	0	0.97
Критерій легкості в інтеграції	0.66	0.33	1	1
Критерій безпечності	1	0.13	0.6	0.6
Всього:	2.27	2.42	2.54	2.95

Також вага критерію стосується легкості в інтеграції: цим не можна нехтувати, але і робити остаточний вибір сильно на цьому базуючись неможна, тому маємо наступні вагомы коефіцієнти у таблиці 6.3. Потенційно їх можна буде змінити, для проведення інших досліджень, або для іншого кінцевого використання технологій – десь окрім серверних рішень. Зміна вагомих коефіцієнтів сильно вплине на остаточне рішення щодо ефективності альтернатив.

Таблиця 6.3 – Вагомі коефіцієнти критеріїв

Критерій швидкості обміну даними	Критерій величини затримки отримання повідомлення	Критерій легкості в інтеграції	Критерій безпечності
0.4	0.25	0.1	0.25

Маємо найвпливовіший – критерій швидкості, потім порівну критерій затримки та безпечності, а в легкості інтеграції ставимо найнижче, через свою специфіку.

Останній крок – зробити саму згортку. Загальна формула для згортки однієї альтернативи виглядає наступним чином за формулою 6.1:

$$X_i = \sum_{j=0}^{n} \alpha_{ij}\beta_j, \quad 6.1)$$

де Xi – значення згортки для і–тої альтернативи,

n – кількість критеріїв, у нашому випадку це 4,

α_{ij} – значення нормованого критерію за альтернативною,

β_j – ваговий коефіцієнт для критерію.

Маємо таблицю 6.4 з результатами згортки:

Таблиця 6.4 – Векторний опис з результатами згортки

	D–Bus	FIFO	TCP	Unix Sockets	Вагомі коефіцієнти
Критерій швидкості обміну даними	0	0.96	0.94	0.38	0.4
Критерій величини затримки отримання повідомлення	0.61	1	0	0.97	0.25
Критерій легкості в інтеграції	0.66	0.33	1	1	0.1
Критерій безпечності	1	0.13	0.6	0.6	0.25
Результати згортки:	0.47	0.70	0.58	0.63	

За результатами згортки найефективнішою технологією міжпроцесної взаємодії виявилася FIFO, але це не є показником, що її треба завжди використовувати. За безпечністю лідирує D–Bus, за легкістю в інтеграції сокети та Message Queue може мати своє унікальне використання. Для різних задач будуть свої найефективніші технології, але у даному досліджені це FIFO.

7. ВИКОРИСТАННЯ РЕЗУЛЬТАТІВ У НАУКОВІЙ І ПРАКТИЧНІЙ ДІЯЛЬНОСТІ

Під час проведення дослідження було проаналізовано за запропонованими критеріями набор технологій міжпроцесної взаємодії, відкориговані експериментальні дані задля кращого порівняння, проведено адитивну лінійну згортку за вагомими коефіцієнтами для найкращого порівняння альтернатив.

Результати цього експерименту показали, що найкращою технологією за показниками швидкості, мінімальної затримки, легкості в інтеграції та безпеки є труби FIFO, які вбудовані у стандартну бібліотеку Linux. Вони мають найкращу швидкість обміну даними та найменшу затримку, хоча через простоту принципу роботу вони не є найлегшими у використанні.

У ході дослідження було представлено уніфікований підхід до порівняння технологій міжпроцесної взаємодії та тестовий додаток, який можна легко масштабувати під додаткові реалізації технологій для подальших досліджень. Цей результат можливо використати у практичній діяльності:

– під час розробки чи планування нової технології міжпроцесної взаємодії, для тестування "на льоту" ітерацій розробки, використовуючи різні вхідні параметри для найширшого огляду можливостей технології;

– розробниками програмних систем, для находження найкращого рішення для їх задач;

– задля покращення власної, можливо ліцензованої, технології взаємодії, використовуючи підходи представлених альтернатив.

Дослідження можна розширити додавши нові підходи для виміру швидкості та затримки отримання повідомлень, провести його на системі з більш стабільною операційною системою, задля найменших шумових вимірювань.

Залежно від задач, можна замінити запропоновані формули вимірювань критерії, збільшити вибірку тестових наборів, замінити вагомі коефіцієнти чи зовсім їх позбутися.

Перспективним напрямком дослідження було б додавання усіх відомих

технологій міжпроцесної взаємодії у одному дослідженні та впровадження порівняння реалізації одних технологій, але на різних операційних системах, бо біло проаналізовано реалізація тільки на Linux з потенційним розширенням критеріїв оцінювання.

Має сенс провести подібне дослідження для мобільних та IoT застосунків, хоча воно може бути ускладнене специфікою підходу розробки програмних систем на даних архітекторах.

Не менш важливим є дослідження технологій для обміну даними не на одній машина, а між декількома, так звані «розподілені обчислення». Таке дослідження мало би схожу специфіку та могло б використати більшість напрацювань, представлених тут. Тестовий додаток, з мінімальними змінами у коді, зміг би повністю покрити потреби проведення додаткових експериментів швидкості та затримки між повідомленнями та нові скрипти для аналізу вихідних файлі могли б покращити якість результативних даних.

ВИСНОВКИ

У результаті проведення дослідження для кваліфікаційної роботи було проаналізовано предметну область та було розглянуто обрані технології міжпроцесної взаємодії напрямку передачі даних. Було описано принципи їх роботи, також знайдено переваги та недоліки для кожної з технологій, сформульовані основні відмінності один від одного.

Як результат, було сформовано звіт дослідження, до якого ввійшли сформульовані вимірні критерії, задля порівняння технологій, було детально описано та аргументовано використання кожного з них та при яких випадках ними можна було б нехтувати. Метриками, обраними для порівняння технологій, стали:

– швидкість обміну даними;

– величина затримки повідомлення;

– легкість в інтеграції;

– безпечність у використані.

У поточному дослідженні було запропоновано актуальні варіанти вимірювання кожної з представлених метрик, були сформульовані критерії та формули, які були використані для обчислення числових значень цих метрик.

Був використаний тестових додаток для проведення експериментів, після яких записані тестові дані, що були проаналізовані, оброблені та завдяки використанню адитивної лінійної згортки з використанням вагомих коефіцієнтів було знайдено найкращу альтернативу з представлених технологій.

Було досліджено, що технологія міжпроцесної взаємодії FIFO найближче за інші альтернативи задовольняє представлені критерії. Технологія має найвищу швидкість обміну даних, через простий принцип роботи, але страждає від складності у реалізації та рівня безпечності, однак недоліки не завадили отримати найкращий результат згортки.

Кваліфікаційна робота пройшла апробацію на 28–му Міжнародному молодіжному форумі «Радіоелектроніка та молодь у XXI столітті» конференції «Інформаційні інтелектуальні системи» (Матеріали статті наведено у додатку В).

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Ubuntu Linux continues to rule the cloud [Електронний ресурс] – URL: https://www.zdnet.com/article/ubuntu–linux–continues–to–rule–the–cloud/ (дата звернення: 12.10.2023).

2. Intelligent Platform Management Interface A Clear and Concise Reference. – 5STARCooks: Gerardus Blokdyk, 2023. – 312 с.

3. Advanced Programming in the UNIX Environment (3rd Edition). – Addison–Wesley Professional: W. Richard Stevens, 2013. – 61–93 с.

4. Характеристики Bash. Кишеньковий довідник системного адміністратора. – Науковий Світ: Арнольд Роббінс, 2023 – 152 с.

5. Linux. Кишеньковий довідник. – Науковий Світ: Скотт Граннеман, 2019 – 464 с.

6. Making of ‘Lord of the Rings’ [Електронний ресурс] – URL: https://variety.com/2016/film/news/lord–of–the–rings–making–of–backstory–business–1201936646 (дата звернення: 22.11.2023).

7. Sharonova N., Kyrychenko I., Shapovalova D. Comparative Analysis of Instant Messaging Protocols and Technologies for Effective Communication in Computer–Mediated Environments CEUR Workshop Proceedings, 2023, 3396, pp. 102–117.

8. Interprocess Communication and Synchronization [Електронний ресурс] – URL: https://medium.com/@adilrk/interprocess–communication–and–synchronization–305922b39daa – (дата звернення: 05.12.2023).

9. Introduction To Linux – Create Pipe With C In Linux [Електронний ресурс] – URL: https://www.engineersgarage.com/introduction–to–linux–create–pipe–with–c–in–linux–part–12–15/ – (дата звернення: 18.11.2023).

10. Getting Started With Unix Domain Sockets [Електронний ресурс] – URL: https://medium.com/swlh/getting–started–with–unix–domain–sockets–4472c0db4eb1– (дата звернення: 06.10.2023).

11. Inter Process Communication (IPC) [Електронний ресурс] – URL: https://www.scaler.com/topics/operating–system/inter–process–communication–in–os/ – (дата звернення: 01.10.2023).

12. D–Bus Tutorial [Електронний ресурс] – URL: https://www.linux.com/topic/networking/d–bus–tutorial/ – (дата звернення:19.09.2023).

13. Qt Remote Object Concepts [Електронний ресурс] – URL: https://doc.qt.io/qt–6/qtremoteobjects–index.html – (дата звернення: 22.11.2023).

14. Kachko O., N. Bilous , Semerkov V. Research on methods for secure web applications development Information Technologies in Innovation Busi–ness (ITIB), 7–9 October, 2015, Kharkiv, Ukraine Proceedings of ITIB, p.26–27, ISBN 978–966–659–214–2.

15. Unix time [Електронний ресурс] – URL: https://en.wikipedia.org/wiki/Unix_time – (дата звернення: 15.04.2024).

16. C++ Software Design: Design Principles and Patterns for High–Quality Software 1st Edition. – O'Reilly Media: Klaus Iglberger, 2022 – 435 с.

17. Python for Data Analysis. Data Wrangling with Pandas, NumPy, and Jupyter. 3rd Edition. – O'Reilly: Wes McKinney, 2022 – 552 с.