Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

Розробка методів і програмних засобів дослідження тредів та процесів в операційній системі Linux

Виконав: студент 2-го курсу
групи 2ПІ-22м_ спеціальності
121 – Інженерія програмного забезпечення
Яворський Дмитро Григорович

Керівник: к.т.н., доц. каф. ПЗ Хошаба О.М.
«12» грудня 2023 р.

Опонент: к.т.н., доц. каф. ЗІ Лукічов В.В.
«12» грудня 2023 р.

Допущено до захисту
Завідувач кафедри ПЗ
д.т.н., проф. Романюк О. Н.
(прізвище та ініціали)
«12» грудня 2023 р.

Вінниця ВНТУ – 2023

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти ІІ-й (магістерський)
Галузь знань 12 – Інформаційні технології
Спеціальність 121 – Інженерія програмного забезпечення
Освітньо-професійна програма – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О. Н.
« _19_ » вересня 2023 р.

ЗАВДАННЯ
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Яворському Дмитру Григоровичу

1. Тема роботи – розробка методів і програмних засобів дослідження тредів та процесів в операційній системі Linux.

Керівник роботи: Хошаба Олександр Мирославович, к.т.н., доцент кафедри ПЗ, затверджені наказом вищого навчального закладу від « 18 » вересня 2023 р. № 247.

2. Строк подання студентом роботи
5 грудня 2023 р.

3. Вихідні дані до роботи: використання CPU на виконання процесів та тредів - не більше 10%, загальне навантаження на систему (upload) - не більше 0.5, загальна кількість тредів в одному процесі - не більше 30, загальна кількість процесів - не більше 20, використання пам'яті - не більше 4 ГБайт, пріоритет для процесів - середній або низький, загальна кількість сторонніх процесів в системі - не більше 200, кількість запитів до програмного засобу - не менш 300 запитів за секунду, визначення розміру обробки запитах на серверах - не більш ніж 3000 мс.

4. Зміст розрахунково-пояснювальної записки: вступ, обгрунтування вибору методу аналізу, постановка задачі та аналіз методів дослідження тредів та процесів в операційній системі Linux, дослідження особливостей роботи тредів та процесів в операційній системі Linux, запропонований метод дослідження роботи тредів, програмна реалізація методів дослідження тредів та

процесів, структурно-функціональні особливості основних модулів програмного засобу, економічна частина, висновки, додатки.

5. Перелік графічного матеріалу: вступ, актуальність теми, вибір методів аналізу, постановка задачі та аналіз методів дослідження тредів та процесів в операційній системі Linux, дослідження особливостей роботи тредів та процесів в операційній системі Linux, запропонований метод дослідження роботи тредів, наукова новизна одержаних результатів, економічний розділ, висновки.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Хошаба О. М., к.т.н., доцент кафедри ПЗ	19.09.23	01.12.23
5	Кавецький В.В., к.е.н., доц. кафедри ЕПВМ	22.10.23	01.12.23

7. Дата видачі завдання 19 вересня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Постановка задачі дослідження тредів та процесів в операційній системі Linux.	19.09.2023-02.10.2023	вик
2	Аналіз та виконання методів дослідження тредів та процесів в операційній системі Linux.	03.10.2023-16.10.2023	вик
3	Дослідження особливостей роботи тредів та процесів в операційній системі Linux.	17.10.2023-28.10.2023	вик
4	Програмна реалізація методів дослідження тредів та процесів в операційній системі Linux.	29.10.2023-12.11.2023	вик
5	Економічна частина	13.11.2023-01.12.2023	вик

Студент _____ **Яворський Д. Г.**
(підпис) (прізвище та ініціали)

Керівник магістерської кваліфікаційної роботи _____ **Хошаба О. М.**
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

УДК 681.3.06

Яворський Д.Г. Розробка методів і програмних засобів дослідження тредів та процесів в операційній системі Linux: магістерська кваліфікаційна робота зі спеціальності 121 Інженерія програмного забезпечення, освітня програма – інженерія програмного забезпечення. Вінниця: ВНТУ, 2023. 127 с.

На укр. мові. Бібліогр. : 36 назв ; рис. : 15; табл. 19.

В магістерській роботі була визначена постановка задачі дослідження тредів та процесів в операційній системі Linux. Виконано аналіз та показані методи дослідження тредів та процесів в операційній системі Linux.

Проведено дослідження особливостей роботи тредів та процесів в операційній системі Linux, де запропоновано метод обчислень зображення Мандельброта як фрактала на основі моделі багатопоточності процесів або тредів. Виконано програмна реалізація методів дослідження тредів та процесів в операційній системі Linux, де визначені вимоги щодо розробки системи управління тредів та процесів, створені структурно-функціональні особливості основних модулів програмного засобу.

Виконано проектування, розробка та впровадження основних модулів програмного засобу.

Ключові слова: методи дослідження тредів та процесів, операційна система Linux, сервера, робочі станції, корпоративні мережі.

ABSTRACT

Yavorskyi D.G. Development of methods and software for monitoring threads and processes in the Linux operating system : master's qualification thesis on the specialty 121 Software engineering, educational program - software engineering. Vinnytsia: VNTU, 2023. 123 with.

In Ukrainian language. Bibliographer : 36 titles; Fig. : 15; table 19.

In the master's thesis, the formulation of the task of researching threads and processes in the Linux operating system was defined. The analysis is performed, and the methods of researching threads and processes in the Linux operating system are shown.

A study of the peculiarities of the work of threads and processes in the Linux operating system was carried out, where a method of calculating the Mandelbrot image as a fractal based on the multithreading model of processes or threads was proposed. The software implementation of thread and process research methods in the Linux operating system was carried out, where the requirements for the development of the thread and process management system were determined, and the structural and functional features of the main modules of the software tool were created.

The design, development, and implementation of the main modules of the software tool have been completed.

Keywords: thread and process research methods, Linux operating system, servers, workstations, corporate networks.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. Розробка методів та програмних засобів для дослідження тредів (потоків) та процесів в операційній системі Linux була актуальною і важливою галуззю ще з прошлого століття. Так як Linux є однією з найпопулярніших операційних систем у світі, то вона використовується на багатьох серверах, вбудованих системах, мобільних пристроях та багатьох інших пристроях. Тому вивчення та аналіз трейдів та процесів в Linux важливо для розробників, адміністраторів систем, дослідників та інженерів [1].

До основних причин актуальності та важловості досліджень та розробки в цій галузі відноситься багато з чинників. Наприклад, існує підвищена складність систем завдяки тому, що сучасні операційні системи Linux стають все більш досконалими з наявністю великої кількості процесів та трейдів, які взаємодіють між собою. Тому, дослідження цих компонентів допомагає зрозуміти та вирішувати проблеми з продуктивністю та стабільністю системи.

Розробка засобів для моніторингу та аналізу процесів і тредів допомагає відлагоджувати програми та виявляти помилки, що виникають в системі. Ефективне управління обчислювальними ресурсами, такими як пам'ять, процесорні ядра та інші дуже важливі для забезпечення оптимальної продуктивності системи.

З подальшим розвитком операційних систем Linux виникають нові технології та можливості, такі як контейнеризація (наприклад, Docker та Kubernetes) і віртуалізація. Дослідження цих нововведень вимагає розробки спеціалізованих інструментів. Тому, є багато проектів з відкритим кодом, таких як strace, lsof, top, ps, htop і інших, що допомагають в аналізі і моніторингу процесів та тредів в Linux [1-3].

Таким чином, актуальність розробки методів та програмних засобів для дослідження тредів та процесів в операційній системі Linux залишається важливою, оскільки ця область є ключовою для забезпечення ефективності, безпеки та стабільності Linux-систем у сучасному інформаційному середовищі.

Мета та завдання дослідження:

Метою роботи є підвищення ефективності використання потоків та процесів в операційній системі Linux за рахунок використання методів та механізмів багатопоточності.

У відповідності до поставленої мети потрібно виконати такі **завдання**:

- сформувати задачі дослідження тредів та процесів в операційній системі Linux;

- виконати аналіз методів дослідження тредів та процесів в операційній системі Linux;

- розглянути аналіз планування процесів та тредів в операційній системі Linux;

- провести дослідження тредів та процесів в операційній системі Linux;

- визначити особливості дослідження тредів та процесів в операційній системі Linux;

- запропоновати метод дослідження роботи процесів та тредів;

- розробити вимоги щодо розробки системи управління тредів та процесів;

- визначити структурно-функціональні особливості основних модулів програмного засобу;

- виконати роботи з реалізації методу досліджень, де здійснити проектування, розробку та впровадження основних модулів програмного засобу.

Об’єктом дослідження є процеси обробки даних в операційній системі Linux.

Предметом дослідження є методи та засоби підвищення ефективності використання потоків та процесів в операційній системі Linux.

Методи дослідження: теорія чисел та чисельних методів, лінійна алгебра, системний аналіз процесу обробки даних, моделі якості роботи, математичні методи та моделі апроксімації даних.

Наукова новизна одержаних результатів:

- вперше запропоновано метод оцінки продуктивності заданої кількості потоків, особливість якого полягає у виконанні обчислень зображення

Мандельброта як фрактала, що визначений множиною точок на комплексній площині, і, як наслідок, дає можливість визначити математичну модель багатопоточності процесів або тредів;

- здобуло подальший розвиток метод обчислень зображення Мандельброта, який, на відміну від існуючих, полягає у його використані з метою дослідження процесів в операційній системі Linux, що дозволяє визначати математичні моделі часу побудови фракталу в багатопотокових програмних засобах.

Практична цінність отриманих результатів. Практичне значення отриманих результатів полягає в тому, що на основі отриманих в магістерській кваліфікаційній роботі теоретичних положень запропоновано багатопоточний алгоритм та розроблено програмні засоби з підвищення ефективності використання потоків та процесів в операційній системі Linux.

Особистий внесок здобувача. У магістерській кваліфікаційній роботі усі результати дослідження здобуті автором даної роботи самостійно. У роботі [1], опублікованій самостійно, автору належить формування постановки проблеми, мети роботи та основної частини.

Апробація матеріалів магістерської кваліфікаційної роботи. Основні положення магістерської кваліфікаційної роботи доповідалися та обговорювалися на Всеукраїнській науково-технічній конференції молодих вчених, аспірантів та студентів "Комп’ютерні ігри і мультимедіа як інноваційний підхід до комунікації - 2023", (Одеса, 2023).

Публікації. Основні результати досліджень опубліковано в науковій праці у матеріалах конференції.

РОЗДІЛ 1
ОБГРУНТУВАННЯ ВИБОРУ МЕТОДУ АНАЛІЗА

1.1. Задачі дослідження тредів та процесів в операційній системі Linux

Розробка методів та програмних засобів для дослідження тредів (потоків) та процесів в операційній системі Linux була актуальною і важливою галуззю. Linux є однією з найпопулярніших операційних систем у світі, і вона використовується на багатьох серверах, вбудованих системах, мобільних пристроях та багатьох інших пристроях. Тому вивчення та аналіз тредів та процесів в Linux важливо для розробників, адміністраторів систем, дослідників та інженерів. Тому, до основних задач досліджень та розробки програмних засобів відносяться наступні (рис. 1.1).

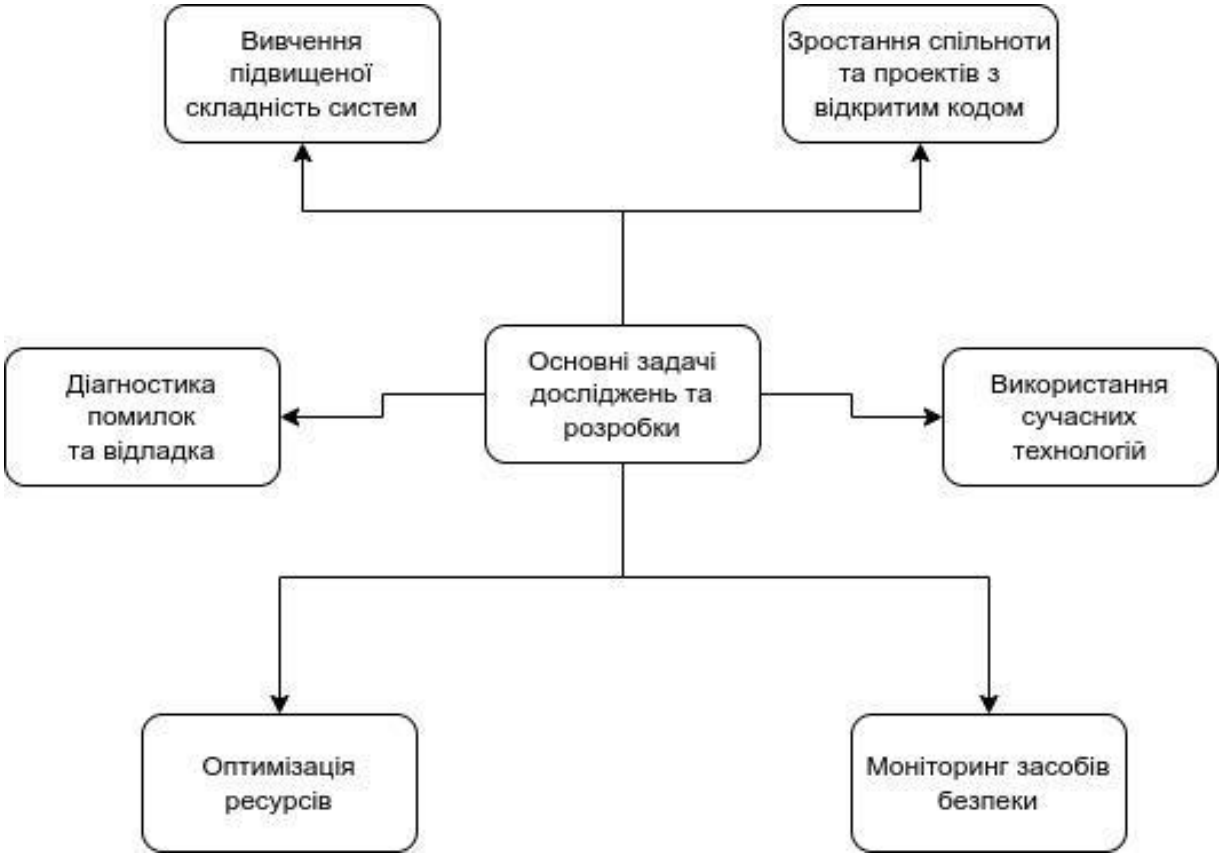


Рисунок 1.1 - Основні задачі досліджень та розробки програмних засобів дослідження тредів та процесів в операційній системі Linux

Основні задачі досліджень та розробки програмних засобів дослідження тредів та процесів в операційній системі Linux більш детально. Сучасні операційні системи Linux стають все більш складними з великою кількістю процесів та трейдів, які взаємодіють один з одним. Дослідження цих компонентів допомагає зрозуміти та вирішувати проблеми з продуктивністю, безпекою та стабільністю системи. Вивчення підвищеної складності систем є важливим завданням у галузі інформаційних технологій і операційних систем, включаючи Linux [2,3].

Таблиця 1.1

Основні задачі досліджень та розробки програмних засобів дослідження тредів та процесів в операційній системі Linux

Задача	Опис задачі
Підвищена складність систем	Сучасні Linux-системи мають багато процесів і тредів, що потребує вивчення та аналізу для підвищення продуктивності і стабільності.
Діагностика помилок та відладка	Розробка засобів для моніторингу та аналізу допомагає виявляти та виправляти помилки в програмах і системі.
Оптимізація ресурсів	Ефективне управління ресурсами, такими як пам'ять та процесорні ядра, є ключовим для забезпечення оптимальної продуктивності.
Моніторинг безпеки	Аналіз трейдів і процесів допомагає виявляти потенційні загрози та захищати систему від атак та порушень безпеки.
Сучасні технології	Розвиток нових технологій, таких як контейнеризація та віртуалізація, вимагає розробки спеціалізованих інструментів для аналізу та управління.
Зростання спільноти та проектів	Багато проектів з відкритим кодом допомагають у моніторингу та аналізі процесів і тредів у Linux, і ця спільнота продовжує зростати.

З ростом кількості процесів, тредів та ресурсів, що використовуються в операційній системі, зростає і ймовірність виникнення конфліктів і нестабільності. Вивчення підвищеної складності допомагає ідентифікувати потенційні джерела проблем і запобігти їх виникненню.

Розуміння та управління роботою багатьох процесів і тредів може допомогти оптимізувати використання ресурсів, таких як центральний процесор і оперативна пам'ять, що підвищує продуктивність системи в цілому.

Ефективне розподілення ресурсів полягає у вивченні складних взаємодій між процесами та тредами, що дозволяє ефективно розподіляти обмежені ресурси між різними додатками та завданнями. При виникненні помилок або неправильної поведінки програм важливо вміти відлагоджувати їх, а для цього необхідно розуміти, як процеси та треди взаємодіють один з одним.

З ростом обсягу та складності проектів і систем, зокрема у веб-розробці та обробці даних, розуміння підвищеної складності стає ключовим фактором для забезпечення їх масштабованості. Вивчення підвищеної складності допомагає виявляти потенційні точки вразливості та зрозуміти можливі атаки, що можуть бути використані для порушення безпеки системи.

Усі ці фактори підкреслюють важливість вивчення та розуміння підвищеної складності систем, включаючи операційні системи, такі як Linux. Це допомагає забезпечити ефективну роботу системи, підвищити її надійність та безпеку, і забезпечити задоволення потреб користувачів і розробників. Розробка засобів для моніторингу та аналізу процесів і тредів допомагає відлагоджувати програми та виявляти помилки, що виникають в системі.

Тому, діагностика помилок та відладка програмного коду є невід'ємною частиною розробки програмного забезпечення і важливою навичкою для розробників. Діагностика помилок і відладка допомагають виявити та усунути дефекти та помилки в програмному коді перед випуском продукту. Це забезпечує високу якість та надійність програми, що є важливим для задоволення потреб користувачів і підтримки репутації розробника чи компанії.

Відладка допомагає зменшити витрати часу та ресурсів на пошук і виправлення помилок у вже написаному коді. Це особливо важливо в великих проектах, де велика кількість розробників працює над різними частинами програми. Відладка допомагає розробникам швидше зрозуміти причину помилок та шляхи їх виправлення. Це підвищує продуктивність розробників і дозволяє швидше внести необхідні зміни.

Діагностика і відладка дозволяють переконатися, що програма веде себе вірно в усіх ситуаціях і виконує очікувану функціональність. Це особливо важливо для критичних систем, таких як медичні програми, автономні автомобілі або фінансові системи. Відладка є важливою частиною процесу навчання розробників. Вона допомагає їм краще розуміти мову програмування, інструменти розробки та особливості конкретного проекту. Діагностика і відладка можуть допомогти виявити потенційні вразливості програми, які можуть бути використані для атаки. Це важливо для забезпечення безпеки програми та захисту від несанкціонованого доступу.

Загалом, діагностика помилок та відладка програмного коду є необхідними етапами в розробці програмного забезпечення, оскільки вони сприяють забезпеченню якості, ефективності та безпеки програми, а також покращують навички розробників і зменшують витрати на підтримку.

Ефективне управління ресурсами (пам'яттю, процесорними ядрами та іншими ресурсами) важливо для забезпечення оптимальної продуктивності системи. Оптимізація ресурсів є критично важливою для ефективності та продуктивності систем, особливо в обчислювальному середовищі.

Обчислювальні ресурси, такі як центральний процесор, оперативна пам'ять і дисковий простір, є обмеженими. Оптимізація ресурсів дозволяє максимально використовувати їх потенціал і зменшити витрати. Якщо система оптимізована для ефективного використання ресурсів, це може допомогти зменшити потребу в закупівлі нового обладнання або розширенні існуючого, що економить гроші [2,3].

Оптимізована система працює швидше і ефективніше, що забезпечує кращу продуктивність для користувачів і додатків. Це особливо важливо в обчислювальних задачах та додатках з великим навантаженням. Оптимізація ресурсів може допомогти уникнути перевантажень і витоків пам'яті, що можуть призвести до збоїв та падінь системи.

Оптимізація ресурсів також допомагає зменшити споживання енергії, що є важливим для зниження викидів CO2 та зменшення витрат на електроенергію.

В мобільних пристроях та вбудованих системах ресурси обмежені, тому оптимізація ресурсів є критично важливою для забезпечення їхньої працездатності та продуктивності. Оптимізація ресурсів дозволяє системі легко масштабуватися і забезпечувати доступність для багатьох користувачів.

Усі ці фактори підкреслюють необхідність оптимізації ресурсів в інформаційних технологіях і комп'ютерних системах. Це допомагає забезпечити більшу продуктивність, зменшення витрат та покращення якості обчислювальних систем.

Важливість забезпечення безпеки в системах Linux важко переоцінити. Дослідження та аналіз тредів та процесів допомагає виявляти потенційні загрози та атаки. Моніторинг засобів безпеки є критично важливим елементом для забезпечення інформаційної безпеки в сучасному світі, де кіберзагрози і потенційні атаки постійно зростають.

Моніторинг засобів безпеки дозволяє виявляти вразливості в інфраструктурі та програмному забезпеченні до того, як зловмисники використають їх для атаки. Це дозволяє вчасно вжити заходів для запобігання або реагування на потенційні загрози.

Моніторинг дозволяє виявляти незвичайну або аномальну активність, яка може бути зв'язана з атаками, і негайно реагувати на неї для обмеження шкоди. Постійний моніторинг допомагає виявити проблеми та вирішити їх до того, як вони призведуть до відмови системи або втрати даних. Це сприяє підтримці стабільності та надійності системи.

Моніторинг може допомогти встановити джерела атак і шляхи, якими зловмисники проникають у систему, що дозволяє приймати відповідні заходи для запобігання подібним атакам у майбутньому.

Багато галузей, таких як фінанси, охорона здоров'я та галузі, пов'язані з обробкою особистих даних, вимагають відповідності певним стандартам безпеки. Моніторинг допомагає забезпечити відповідність цим стандартам. Моніторинг дозволяє ретельно аналізувати інциденти безпеки, включаючи їхні наслідки та джерела. Це допомагає вивчити і вдосконалити заходи безпеки.

Велика частина організацій зберігає важливу і конфіденційну інформацію, і моніторинг допомагає забезпечити її захист від незаконного доступу.

Відсутність ефективного моніторингу засобів безпеки може призвести до серйозних наслідків, включаючи втрату даних, порушення конфіденційності та інтегрітету інформації, фінансові втрати і пошкодження репутації організації. Тому моніторинг безпеки є важливою складовою стратегії інформаційної безпеки в будь-якому організаційному середовищі.

З розвитком операційних систем Linux виникають нові технології та можливості, такі як контейнеризація (наприклад, Docker та Kubernetes) і віртуалізація. Дослідження цих нововведень вимагає розробки спеціалізованих інструментів. Використання сучасних технологій є ключовим фактором для досягнення успіху в сучасному світі [2-4].

Сучасні технології дозволяють автоматизувати багато рутинних завдань і процесів, що підвищує продуктивність працівників і зменшує час виконання завдань. Використання сучасних технологій дозволяє розробляти продукти і послуги вищої якості, з більшими можливостями і функціональністю.

Організації, які активно використовують сучасні технології, можуть зберігати свою конкурентоспроможність, оскільки вони швидше адаптуються до змін на ринку і вдосконалюють свої продукти та послуги. Сучасні технології дозволяють оптимізувати бізнес-процеси і зменшити витрати на обладнання, ресурси та інфраструктуру.

Сучасні технології забезпечують зручні та ефективні способи комунікації між співробітниками, клієнтами і партнерами, що сприяє кращому спілкуванню та співпраці. Вони відкривають нові можливості для створення і впровадження інноваційних продуктів та послуг, що дозволяє підприємствам стати лідерами у своїх галузях.

Сучасні технології аналізу даних дозволяють збирати, обробляти і аналізувати великі обсяги інформації, що допомагає приймати кращі рішення на основі даних. Використання сучасних технологій дозволяє забезпечити захист від кіберзагроз і зберегти конфіденційність даних. Такі технології дозволяють працювати та отримувати доступ до інформації з будь-якого місця і на будь-якому пристрої, що сприяє зручності та ефективності роботи.

Сучасний світ постійно змінюється, і тільки за допомогою сучасних технологій можна відповідати на ці зміни і залишатися актуальним. Усі ці фактори підкреслюють важливість використання сучасних технологій для підтримки ефективності, конкурентоспроможності та інноваційності в різних сферах діяльності, від бізнесу і освіти до охорони здоров'я та наукових досліджень [2,3].

В теперішній час є багато проектів з відкритим кодом, таких як strace, lsof, top, ps, htop і багато інших, що допомагають в аналізі і моніторингу процесів та тредів в Linux. Розробка та підтримка таких проектів є актуальною задачею.

Розвиток спільноти та проектів з відкритим кодом має велике значення для багатьох сфер, включаючи розробку програмного забезпечення, науку, освіту, інновації та багато інших. Проекти з відкритим кодом надають можливість широкому колу людей безкоштовно використовувати інструменти і ресурси для розробки програмного забезпечення. Це робить знання та технології доступними для всіх, незалежно від фінансових можливостей.

Проекти з відкритим кодом сприяють співпраці і колективній розробці програмного забезпечення. Розробники з різних куточків світу можуть об'єднатися для створення і вдосконалення програм з відкритим кодом, що призводить до розробки високоякісних продуктів. Відкритий код сприяє

інноваціям та швидкому розвитку технологій. Широкий загальний доступ до коду стимулює створення нових ідей і рішень.

Багато компаній користуються відкритими проектами для розробки власного програмного забезпечення. Відкритий код дозволяє зменшити витрати на розробку та підтримку програм, а також прискорює вихід на ринок. Спільноти з відкритим кодом створюють можливість для обміну досвідом та навчання. Розробники можуть вчити один одного та розвивати свої навички.

Відкритий код дозволяє багатьом експертам аналізувати інформацію щодо безпеки і стабільності програм. Це сприяє вчасному виявленню і виправленню помилок і вразливостей. Відкритий код сприяє демократії в технологіях, де рішення приймаються спільно і не обмежені комерційними інтересами. Відкритий код дозволяє створювати програмне забезпечення, яке може бути використано на різних платформах і операційних системах.

Усі ці фактори підкреслюють необхідність розвитку спільнот та проектів з відкритим кодом як важливого джерела інновацій, знань і можливостей для розробників, підприємств і суспільства в цілому.

Таким чином, актуальність розробки методів та програмних засобів для дослідження тредів та процесів в операційній системі Linux залишається високою, оскільки ця область є ключовою для забезпечення ефективності, безпеки та стабільності Linux-систем у сучасному інформаційному середовищі.

1.2. Аналіз методів дослідження тредів та процесів

Аналіз методів дослідження тредів та процесів в операційній системі Linux часто базується на використанні різних утиліт та інструментів, які надають інформацію про процеси та потоки. Утиліти надають користувачам та адміністраторам зручний спосіб аналізу та моніторингу системи без необхідності написання власного програмного коду або використання складних методів [3-5].

Ці утиліти спрощують аналіз та моніторинг процесів та потоків у Linux і надають інформацію, яка допомагає користувачам та адміністраторам розуміти, як система працює та виявляти можливі проблеми.

Взаємодія між процесами та тредами в операційній системі Linux відбувається за допомогою різних механізмів та інструментів, які дозволяють процесам та тредам обмінюватися даними, спілкуватися та синхронізуватися (рис. 1.2). Розглянемо деякі з основних методів взаємодії між тредами та процесами в операційній системі Linux.

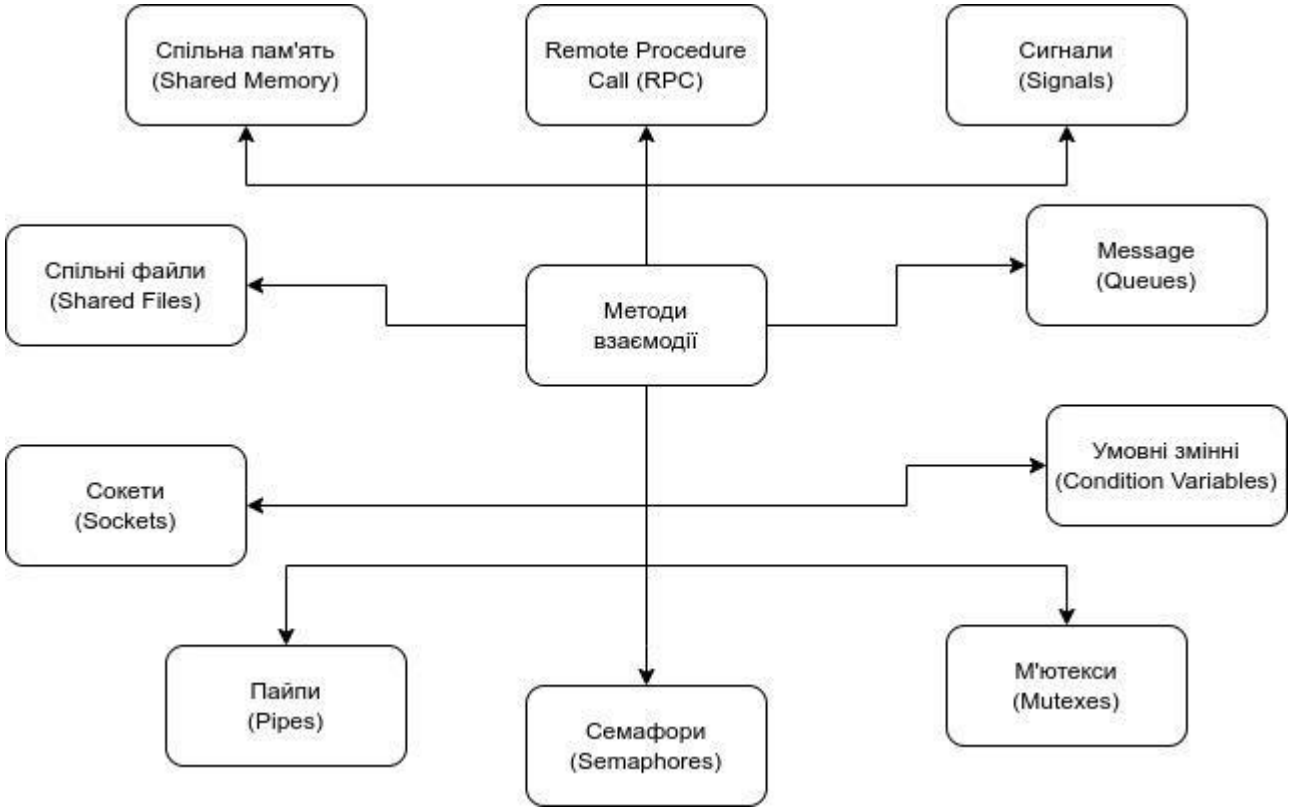


Рисунок 1.2 - Взаємодія між процесами та тредами в операційній системі Linux

Спільна пам'ять дозволяє декільком процесам чи тредам отримувати доступ до одного та самого регіону пам'яті. Процеси можуть читати та записувати дані у спільний регіон пам'яті для обміну інформацією. Взаємодія між процесами та тредами за допомогою спільної пам'яті (Shared Memory) в операційній системі Linux передбачає спільний доступ до регіону пам'яті, який може бути використаний для обміну даними між різними процесами або тредами (табл. 1.2).

Для створення спільної пам'яті спершу необхідно створити регіон спільної пам'яті. Це може бути зроблено за допомогою функцій, таких як shmget() або

mmap(). Під час створення обраного регіону пам'яті встановлюються розмір та унікальний ключ для ідентифікації спільної пам'яті.

Після створення ділянки спільної пам'яті процеси або треди можуть прикріпити (підключити) цю ділянку до свого адресного простору. Це робиться за допомогою функцій, таких як shmat() або mmap(). Після прикріплення пам'яті до адресного простору процесу/треду, вони отримують доступ до цієї ділянки.

Таблиця 1.2

Методи взаємодії між процесами та тредами в операційній системі Linux

Механізм / Інструмент	Опис
Спільна пам'ять (Shared Memory)	Декілька процесів чи тредів отримують доступ до одного та самого регіону пам'яті для обміну даними.
Спільні файли (Shared Files)	Декілька процесів чи тредів можуть ділитися доступом до одного та самого файлу, щоб обмінюватися даними.
Сокети (Sockets)	Дозволяють процесам та тредам взаємодіяти через мережу, навіть якщо вони працюють на різних машинах.
Пайпи (Pipes)	Для спрощеного обміну даними між процесами чи тредами. Включають іменовані та анонімні пайпи.
Семафори (Semaphores)	Використовуються для синхронізації доступу до спільних ресурсів та дозволу на виконання певних дій.
М'ютекси (Mutexes)	Використовуються для синхронізації доступу до критичних секцій коду, дозволяють лише одному процесу чи треду отримувати доступ до ресурсу в один момент часу.
Умовні змінні (Condition Variables)	Дозволяють процесам чи тредам чекати на виникнення певних умов та сповіщати про зміни в стані.
Черги повідомлень (Message Queues)	Дозволяють процесам та тредам надсилати повідомлення один одному через системний меседж-сервіс.
Сигнали (Signals)	Сповіщають процеси чи треди про виникнення певних подій або помилок, і дозволяють їм

	взаємодіяти відповідно до цих подій.
Remote Procedure Call (RPC)	Дозволяє викликати функції або методи, які знаходяться в інших процесах чи на інших машинах.

Потім, коли ділянка спільної пам'яті прикріплений до адресного простору різних процесів або тредів, вони можуть звертатися до цієї області для читання та запису даних. Тому, процеси/треди можуть читати та записувати дані в цю ділянку, і ці зміни будуть видимими для всіх, хто прикріпив цю область пам'яті.

Після завершення роботи з областю спільної пам'яті процеси/треди повинні відокремити (відключити) його від свого адресного простору за допомогою функцій, таких як shmdt() або munmap(). Це важливо для звільнення ресурсів та попередження витоку пам'яті.

Ділянка спільної пам'яті може бути видалена після завершення використання за допомогою функції, такої як shmctl() з командою IPC_RMID. Видалення звільняє ресурси і більше не дозволяє процесам/тредам звертатися до цієї області [4,6].

Цей механізм спільної пам'яті дозволяє декільком процесам або тредам обмінюватися даними безпосередньо у швидкому режимі. Важливо правильно керувати прикріпленням та відокріпленням пам'яті, а також забезпечити синхронізацію доступу до спільної пам'яті, щоб уникнути конфліктів та гарантувати цілісність даних.

Декілька процесів чи тредів можуть ділитися доступом до одного та самого файлу. Це використовується для обміну даними між процесами.

Взаємодія між процесами та тредами за допомогою спільних файлів в операційній системі Linux передбачає спільний доступ до одного та того ж файлу або файлів, що дозволяє обмінюватися даними. Для цього спершу потрібно створити або відкрити файл, який буде використовуватися для спільної взаємодії. Це може бути файл звичайного розміру або спеціально створений файл, наприклад, через системний виклик open().

Різні процеси або треди можуть відкривати цей файл для читання та запису за допомогою системного виклику open() або бібліотечних функцій, таких як fopen(). Важливо відкривати файл у режимі, що дозволяє багатьом процесам одночасно читати і записувати.

Після відкриття файлу процеси або треди можуть читати та записувати дані в цей файл за допомогою системних викликів read() і write() або функцій, таких як fread() і fwrite(). Дані можуть бути зчитані або записані в будь-який час, коли це потрібно.

Для уникнення конфліктів і гарантування цілісності даних можуть використовуватися механізми синхронізації, такі як м'ютекси (mutexes) чи семафори. Вони дозволяють процесам чи тредам координувати доступ до спільного файлу.

Після завершення роботи з файлом процеси або треди повинні закрити файл за допомогою системного виклику close() або функції fclose(). Це важливо для звільнення ресурсів та попередження витоку файлових ресурсів.

Взаємодія за допомогою спільних файлів дозволяє різним процесам або тредам обмінюватися даними, а також спільно використовувати файли для зберігання інформації. Важливо враховувати, що доступ до файлу може бути синхронізованим для запобігання конфліктам при одночасному читанні та записі [4-6].

Сокети дозволяють процесам та тредам взаємодіяти через мережу. Вони можуть бути використані для обміну даними між процесами на різних машинах або на одній машині через мережевий стек.

Взаємодія між процесами та тредами за допомогою сокетів в операційній системі Linux передбачає використання мережевих сокетів для обміну даними між різними процесами або навіть між процесами на різних машинах через мережу [7].

Для цього, спершу потрібно створити сокет в обох процесах, які будуть взаємодіяти. Для цього використовуються системні виклики, такі як socket(). При

цьому визначаються тип сокету (наприклад, TCP або UDP) та адреса і порт для зв'язку.

В одному з процесів (зазвичай серверному) потрібно прийняти з'єднання, в іншому (клієнтському) потрібно встановити з'єднання. Для TCP це зазвичай включає bind(), listen() та accept() на сервері та connect() на клієнті.

Після встановлення з'єднання процеси можуть обмінюватися даними через сокети. Для цього використовуються системні виклики, такі як send() і recv() для TCP, або sendto() і recvfrom() для UDP. Дані надсилаються на одному кінці з'єднання і приймаються на іншому кінці. Після завершення обміну даними з'єднання та сокети слід закрити за допомогою системного виклику close().

Сокети можуть бути використані для локальної взаємодії між процесами на одній машині або для мережевої взаємодії між процесами на різних машинах через TCP/IP мережу. Цей механізм є потужним і дозволяє процесам взаємодіяти зі значними відстанями та через різні мережеві технології. Важливо враховувати безпеку і аутентифікацію, коли використовуються сокети для взаємодії між процесами чи між різними системами, щоб забезпечити конфіденційність та цілісність даних.

Пайпи використовуються для спрощеного обміну даними між процесами. Є іменовані пайпи та анонімні пайпи (pipe). Взаємодія між процесами та тредами за допомогою пайпів (pipes) в операційній системі Linux передбачає використання спеціальних каналів для передачі даних в одному напрямку між процесами чи тредами. Пайпи зазвичай використовуються для взаємодії між батьківськими та дочірніми процесами, а також для обміну даними між процесами, які спільно працюють у складних програмах.

Для цього, спершу потрібно створити пайп для обміну даними між процесами чи тредами. В операційній системі Linux це може бути зроблено за допомогою системного виклику pipe().

Після створення пайпа він розділяється на два дескриптори: один для читання інший для запису. Один кінець пайпа призначений для відправлення даних (зазвичай називається "write end"), а інший кінець для отримання даних

(зазвичай називається "read end"). Далі, потрібно створити процеси чи треди, які планують взаємодіяти за допомогою пайпа.

Процеси чи треди можуть взаємодіяти через пайп, записуючи дані в один кінець і читаючи дані з іншого кінця. Для запису та читання використовуються системні виклики, такі як write() для запису даних у пайп і read() для читання даних з пайпа. Після завершення обміну даними процеси чи треди повинні закрити пайпи за допомогою системних викликів close(), щоб звільнити ресурси [8,9].

Взаємодія за допомогою пайпів зазвичай обмежена в одному напрямку, тобто дані передаються від одного кінця пайпа до іншого. Це робить пайпи корисними для реалізації потоків даних між процесами або тредами, де один процес генерує дані, інший процес чи тред їх обробляє.

Семафори використовуються для синхронізації доступу до спільних ресурсів. Вони дозволяють процесам чекати, поки інший процес завершить дію над спільним ресурсом. Взаємодія між процесами та тредами за допомогою семафорів в операційній системі Linux передбачає використання спеціальних об'єктів семафорів для синхронізації доступу до спільних ресурсів або критичних секцій коду. Семафори дозволяють встановлювати та перевіряти рівень доступу та використовуються для управління взаємовиключенням між процесами або тредами.

Для цього, спершу потрібно створити семафор за допомогою системного виклику sem_init(). При цьому вказується початковий рівень доступу (зазвичай 1, якщо це бінарний семафор) та інші параметри.

Після завершення взаємодії семафор можна закрити за допомогою системного виклику sem_destroy(), щоб звільнити ресурси. Це не обов'язково, і семафор може залишитися доступним для інших процесів та тредів після завершення поточного.

Тому, процеси або треди можуть використовувати семафор для встановлення доступу до спільних ресурсів або критичних секцій коду. Для забезпечення взаємовиключення процеси використовують системні виклики

sem_wait() для зниження значення семафора (захоплення) та sem_post() для інкрементації значення семафора (звільнення).

Семафори можуть використовуватися для синхронізації доступу до спільних ресурсів. Наприклад, якщо багато процесів намагаються звернутися до ресурсу, семафор може контролювати доступ, дозволяючи тільки одному процесу одночасно отримувати доступ до ресурсу. Після завершення взаємодії семафор може бути звільнений за допомогою системного виклику sem_post(), щоб інші процеси чи треди могли використовувати його.

Семафори допомагають уникнути гонок за ресурсами і забезпечують правильну синхронізацію між процесами чи тредами. Вони корисні для взаємодії, де потрібно керувати доступом до спільних ресурсів або розділити ресурси між процесами чи тредами.

М'ютекси використовуються для синхронізації доступу до критичних секцій коду. Тільки один процес або тред може отримати доступ до ресурсу, захищеного м'ютексом, в один момент часу. Взаємодія між процесами та тредами за допомогою м'ютексів (mutexes) в операційній системі Linux передбачає використання спеціальних об'єктів синхронізації для захисту критичних секцій коду або спільних ресурсів від одночасного доступу декількох процесів або тредів. М'ютекси дозволяють створювати критичні секції коду, які можуть бути виконані лише одним процесом або тредом одночасно.

Для цього, спершу потрібно створити м'ютекс за допомогою системного виклику, такого як pthread_mutex_init() для м'ютексів в середовищі POSIX або sem_init() для м'ютексів в більш загальному випадку. М'ютекси можуть бути ініціалізовані з різними атрибутами, включаючи режими блокування (recursive або non-recursive) та атрибути безпеки.

Критична секція коду - це область, в якій процеси або треди мають обмежений доступ. Перед входом в критичну секцію коду процеси або треди викликають функцію блокування м'ютекса, таку як pthread_mutex_lock() для POSIX-м'ютекса. Якщо м'ютекс вільний, виклик блокування буде успішним і процес або тред отримає доступ до критичної секції.

Тому, процеси або треди можуть виконувати код в критичній секції. Цей код захищений м'ютексом, і тому лише один процес або тред може виконувати його одночасно. Після завершення виконання коду в критичній секції процес або тред повинен викликати функцію розблокування м'ютекса, таку як pthread_mutex_unlock(). Це дозволить іншим процесам чи тредам отримати доступ до критичної секції коду.

Після завершення взаємодії м'ютекс може бути знищений за допомогою функції, такої як pthread_mutex_destroy(). Це необов'язково, і м'ютекс може залишитися доступним для подальших використань.

М'ютекси є потужним інструментом для забезпечення взаємовиключення та синхронізації між процесами та тредами. Вони допомагають уникнути гонок за ресурсами та інших проблем, пов'язаних з одночасним доступом до спільних даних [9,10].

Умовні змінні використовуються для сповіщення процесів або тредів про зміну умови або стану. Вони дозволяють реалізувати біль складні сценарії синхронізації. Взаємодія між процесами та тредами за допомогою умовних змінних (condition variables) в операційній системі Linux передбачає використання спеціальних змінних для синхронізації та сповіщення про події між процесами чи тредами. Умовні змінні зазвичай використовуються в комбінації з м'ютексами для управління послідовністю виконання та доступом до спільних ресурсів.

Тому, спершу потрібно створити умовну змінну за допомогою системного виклику, такого як pthread_cond_init() для POSIX-умовної змінної. Умовні змінні пов'язані з певним м'ютексом, і вони використовуються для блокування тредів, поки не виконується певна умова. Перед входом в критичну секцію коду або перед перевіркою умови потрібно, щоб процес чи тред міг захопити м'ютекс, за допомогою функції, такої як pthread_mutex_lock(). При цьому, м'ютекси допомагають уникнути гонок за ресурсами. Тоді, процес чи тред перевіряє умову, яку він очікує, щоб продовжити виконання коду.

Якщо така умова не виконана, процес або тред викликає функцію pthread_cond_wait(). Ця функція звільняє м'ютекс і блокує поточний процес чи тред, доки інший процес не викличе pthread_cond_signal() або pthread_cond_broadcast() на цій самій умовній змінній.

Далі, інший процес чи тред, який відповідає за зміну стану, викликає pthread_cond_signal(), щоб сповістити процес чи тред, який очікує на умовній змінній, що умова стала виконуватися. Якщо потрібно сповістити всі процеси чи треди, які очікують на цій умовній змінній, використовується pthread_cond_broadcast(). Після отримання сповіщення процес чи тред виходить з блокування, викликаючи pthread_cond_wait(). Він може відразу захопити м'ютекс і продовжити виконання.

Після завершення роботи процес чи тред повинні викликати pthread_mutex_unlock() для звільнення м'ютекса, щоб інші процеси чи треди могли отримати доступ до умовної змінної.

Таким чином, умовні змінні корисні для управління послідовністю виконання і сповіщення про події між процесами чи тредами. Вони дозволяють програмам робити зміну стану на основі умов та сповіщати інші частини програми про цю зміну стану.

Черги повідомлень (Message Queues) дозволяють процесам та тредам надсилати повідомлення один одному через системний меседж-сервіс. Взаємодія між процесами та тредами за допомогою черг повідомлень (message queues) в операційній системі Linux передбачає використання спеціальних черг для обміну повідомленнями між процесами. Цей механізм дозволяє процесам або тредам обмінюватися даними та повідомленнями через спільну чергу.

Для цього, спершу потрібно створити чергу повідомлень за допомогою системного виклику, такого як msgget(). При створенні черги потрібно вказати унікальний ключ і права доступу для процесів, які будуть взаємодіяти з чергою.

Процес або тред, який "хоче" надіслати повідомлення, використовує системний виклик msgsnd(). Він вказує ідентифікатор черги, тип повідомлення та дані, які потрібно надіслати. Інший процес чи тред, який хоче отримати

повідомлення, використовує системний виклик msgrcv(). Він також вказує ідентифікатор черги та тип повідомлення, який він очікує. Коли повідомлення в черзі з відповідним типом стає доступним, воно зчитується.

Після завершення взаємодії чергу можна видалити за допомогою системного виклику msgctl(). Це необов'язково, і черга може залишитися доступною для подальших взаємодій.

Черги повідомлень дозволяють процесам або тредам обмінюватися структурованими повідомленнями. Вони корисні в ситуаціях, де потрібно передавати дані між процесами та тредами безпосередньо і декількома способами. Цей механізм також дозволяє вирішити проблеми синхронізації та обробки повідомлень в послідовному порядку.

Сигнали використовуються для сповіщення процесів або тредів про виникнення певних подій або помилок. Вони можуть бути використані для обробки інших процесів. Взаємодія між процесами та тредами за допомогою сигналів (signals) в операційній системі Linux передбачає використання спеціальних повідомлень для сповіщення про події та передачі сигналів одному процесу або треду іншого. Сигнали дозволяють процесам чи тредам взаємодіяти та обробляти події, такі як завершення процесу, натискання клавіші та інші події.

Один процес чи тред надсилає сигнал іншому процесу чи треду за допомогою системного виклику, такого як kill() або pthread_kill(). У випадку kill(), відправник вказує ідентифікатор процесу чи треду-одержувача та код сигналу. Сигнали можуть бути стандартними сигналами (наприклад, SIGINT для переривання) або користувацькими сигналами, які можуть мати будь-який код [8-10].

Процес чи тред, який отримав сигнал, повинен мати обробник сигналу, що визначає, як реагувати на сигнал. Обробники сигналів можуть бути функціями, які викликаються при отриманні певного сигналу. Процес чи тред може зареєструвати обробник сигналу за допомогою системних викликів, таких як signal() або sigaction().

Коли сигнал надходить, виконується обробка сигналу, яка визначена у встановленому обробнику сигналу. Обробка може включати в себе реакції на подію, такі як завершення процесу, повідомлення про помилку або інші дії, залежно від типу сигналу та програмної логіки.

В деяких випадках сигнали можуть бути переслані іншим процесам чи тредам, які вже відправляли сигнали. Це дозволяє організувати ланцюгові реакції на сигнали.

Таким чином, сигнали дозволяють процесам та тредам обмінюватися повідомленнями та реагувати на події у системі. Вони корисні для реалізації різних видів взаємодії і синхронізації між процесами та тредами в операційній системі Linux.

Взаємодія між процесами та тредами за допомогою Remote Procedure Call (RPC) передбачає використання віддалених процедур для виконання коду на віддаленому вузлі або процесі в мережі. RPC дозволяє програмам здійснювати виклики методів або функцій на віддалених вузлах, ніби ці функції були викликані локально.

Для цього, спочатку потрібно визначити, які процедури або функції будуть викликані віддалено. Для цього зазвичай використовують мови інтерфейсу, такі як IDL (Interface Definition Language), які визначають сигнатури віддалених процедур і їх параметри. На основі описів віддалених процедур генерується код, який дозволяє виконувати віддалені виклики. Цей код може бути автоматично згенерований за допомогою інструментів, які використовують IDL [12].

На вузлі, який має виконувати віддалені процедури, запускається сервер, який слухає запити від клієнтів. Тоді клієнт викликає віддалену процедуру, як якому-небудь локальному методу або функції. Однак цей виклик виконується на сервері.

Віддалені параметри передаються через мережу на сервер, разом з інформацією про те, яка віддалена процедура повинна бути викликана. Сервер отримує запит та виконує віддалену процедуру. Результат виклику повертається

на клієнта через мережу. Результат виклику повертається на клієнта, як якимось локальним значенням.

RPC дозволяє легко реалізовувати взаємодію між процесами або тредами на віддалених вузлах, незалежно від того, де вони фізично розташовані. Цей підхід дуже корисний для розробки розподілених систем, де програми на різних вузлах повинні взаємодіяти та обмінюватися даними. RPC також дозволяє приховати деталі мережевого взаємодії, спрощуючи розробку програм, які використовують віддалені ресурси чи послуги.

Таким чином, розглянуті механізми та інструменти дозволяють процесам та тредам взаємодіяти, обмінюватися даними та координувати свою роботу в операційній системі Linux. Вибір конкретного механізму залежить від потреб конкретної задачі та вимог до взаємодії між процесами.

1.3. Аналіз планування процесів та тредів в операційній системі Linux

Планування процесів і тредів в операційній системі Linux виконується за допомогою різних утиліт та алгоритмів, які визначають порядок виконання процесів і тредів (рис. 1.3). Розглянемо деякі основні аспекти планування за допомогою різних механізмів та утиліт.

У Linux існує концепція "пріоритету" процесу, який визначається за допомогою "nice value". Більший nice value вказує на менший пріоритет, тоді як менший nice value вказує на вищий пріоритет. nice або renice утиліти дозволяють користувачам змінювати пріоритети процесів. Встановлення пріоритету процесу в операційній системі Linux використовує параметр, відомий як "Nice Value". Це значення використовується для визначення пріоритету процесу в системі. Nice Value визначається у діапазоні від -20 до +19, де менше значення вказує на вищий пріоритет, і навпаки. Загальна ідея полягає в тому, що процеси з меншими Nice Values отримують більше обчислювального часу і ресурсів в системі, оскільки вони вважаються більш "дружніми" або пріоритетними для системи (табл. 1.3).

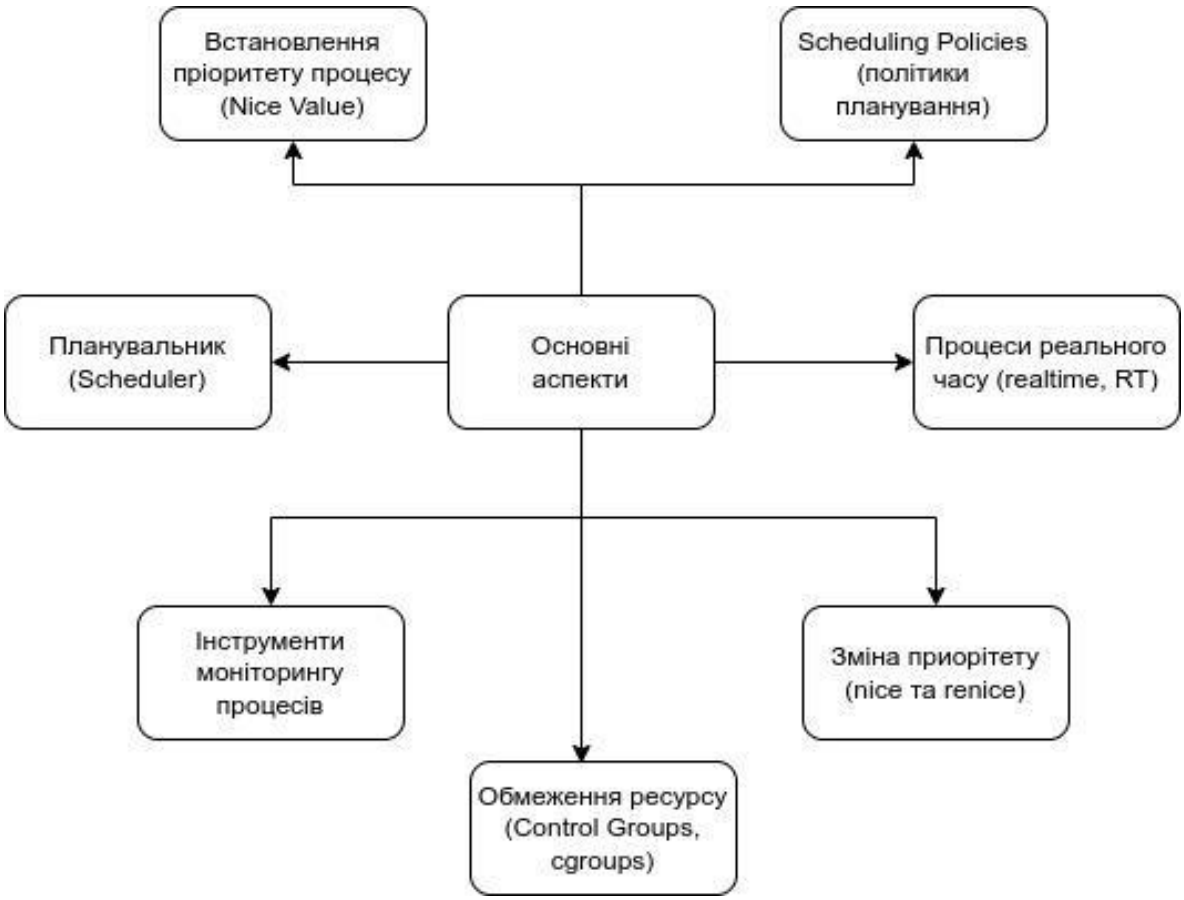


Рисунок 1.3 - Основні аспекти планування за допомогою утиліт

Основні поняття встановлення пріоритету процесу (Nice Value) полягає в наступному. Діапазон значень Nice Value зазвичай змінюється від -20 до +19, де -20 відповідає найвищому пріоритету, а +19 - найнижчому. Для зміни Nice Value може використовуватися команда nice. Зазвичай користувачі можуть зменшити значення Nice Value, але для збільшення значення Nice Value може знадобитися спеціальний статус адміністратора (наприклад, за допомогою sudo).

Зменшення Nice Value збільшує пріоритет процесу, надаючи йому більше обчислювального часу, вищий пріоритет та пріоритетну доступність до ресурсів. Збільшення Nice Value знижує пріоритет процесу, зменшуючи його доступ до ресурсів.

Зміна Nice Value може бути корисною для контролю використання процесорних ресурсів та управління роботою процесів. Наприклад, процеси, що вимагають великої обчислювальної потужності, можуть бути запущені з меншим Nice Value, щоб отримати більше обчислювального часу.

Зміна Nice Value дозволяє користувачам та адміністраторам керувати пріоритетом процесів у системі і забезпечити більш ефективне використання ресурсів, враховуючи потреби різних задач.

Операційна система використовує планувальник для визначення того, який процес або тред має виконуватися в даний момент. У Linux існує кілька планувальників, таких як Completely Fair Scheduler (CFS) для процесів та Multiple Queue Scheduler (MQ) для тредів. Планувальник (Scheduler) в операційній системі відповідає за призначення процесам та тредам часу процесора та ресурсів для їх виконання. Він визначає порядок, в якому процеси виконуються, і відповідає за вирівнювання і використання ресурсів процесора для забезпечення справедливості та ефективності виконання задач.

Таблиця 1.3

Основни аспекти планування за допомогою утиліт

Аспект	Опис
Встановлення пріоритету процесу	Процесам присвоюються пріоритети (Nice Value), де менше значення вказує на вищий пріоритет. Nice Value може бути змінений за допомогою команди nice.
Планувальник	Операційна система використовує планувальник для вибору, який процес повинен виконуватися в наступному циклі планування. Існують різні політики планування, такі як Completely Fair Scheduler (CFS) в Linux.
Інструменти моніторингу процесів	Утиліти, такі як top, htop, ps, дозволяють моніторити та аналізувати роботу процесів та використання ресурсів.
Обмеження ресурсу (Control Groups)	Control Groups або cgroups дозволяють обмежувати ресурси, які можуть використовувати групи процесів, дозволяючи більш ефективно управляти ресурсами.
Зміна приоритету (nice та renice)	Користувачі можуть змінювати пріоритети процесів за допомогою команди nice або renice, щоб контролювати, як ресурси розподіляються між процесами.

Процеси реального часу (realtime, RT)	Для додатків, які вимагають негайної відповіді, існують процеси реального часу. Вони мають вищий пріоритет та гарантовані ресурси для виконання.
Scheduling Policies (політики планування)	Операційна система підтримує різні політики планування, такі як SCHED_FIFO, SCHED_RR, SCHED_OTHER. Кожна політика має свої правила і призначення.

При цьому, планувальник використовує різні політики планування для визначення, який процес отримує доступ до процесора та наскільки довго. Приклади політик включають Completely Fair Scheduler (CFS), Round Robin (RR) і багато інших. Процесам присвоюються пріоритети, які використовуються для вирівнювання пріоритетів та призначення часу процесора. Зазвичай низький пріоритет відповідає більш важливим задачам, а високий пріоритет - менш важливим.

Головне завдання планувальника - мінімізувати час очікування процесів та максимізувати використання процесора. Він розподіляє обчислювальний час між процесами, дозволяючи їм виконувати свої завдання відповідно до їх пріоритетів [11].

Планувальник намагається забезпечити справедливий доступ до ресурсів процесора для всіх процесів. Він уникатиме ситуацій, де один процес займає весь процесор, ущемляючи інші.

Планувальник може реагувати на зміну стану системи і розподіляти ресурси відповідно до потреб. Наприклад, він може збільшувати пріоритет підвищеного пріоритету задач. Планувальник реалізований в ядрі операційної системи і працює як частина ядра. Він виконується в фоновому режимі, обробляючи запити на планування процесів.

Планувальник є важливим компонентом операційної системи, оскільки він дозволяє управляти виконанням процесів і забезпечувати ефективне та справедливе використання ресурсів процесора. Він грає ключову роль у забезпеченні продуктивності та стійкості операційної системи.

Утиліти, такі як top, htop, ps, надають інформацію про активні процеси та їхній пріоритет. Користувачі можуть спостерігати за працездатністю процесів і змінювати їхні пріоритети за необхідності. Інструменти моніторингу процесів в операційній системі Linux є важливими для відстеження та аналізу активності процесів, використання ресурсів та виявлення проблем у системі. Вони надають інформацію про поточний стан процесів, таку як CPU, пам'ять, дисковий ввід/вивід та мережева активність.

Ці інструменти дозволяють адміністраторам і користувачам отримувати інформацію про роботу процесів і ресурсів системи, що допомагає виявляти проблеми, оптимізувати використання ресурсів та контролювати стан системи.

Обмеження ресурсу (Control Groups, cgroups) є важливою функцією Linux дозволяє обмежувати ресурси, доступні для групи процесів, такі як CPU, пам'ять і введено-виведено (I/O). Вона дозволяє управляти виділеними ресурсами та контролювати виконання процесів.

Обмеження ресурсів або Control Groups (cgroups) - це механізм в операційній системі Linux, який дозволяє обмежувати ресурси, доступні для групи процесів. Це потужний інструмент для контролю ресурсів, таких як процесорний час, пам'ять, введення-виведення, мережевий трафік та інші, що може бути корисним для розподілу ресурсів між процесами та групами процесів [11].

Групи процесів cgroups дозволяють об'єднувати процеси в групи та створювати ієрархію груп. Кожна група може мати свої обмеження ресурсів. За допомогою cgroups можна встановлювати ліміти на ресурси, які можуть використовувати групи процесів. Наприклад, можливо обмежити кількість процесорного часу, доступну для певної групи.

Ізоляція ресурсів за допомогою cgroups дозволяють ізолювати групи процесів одна від одної. Це може бути корисним для уникнення впливу одних процесів на інші в мультикорпусних системах.

Засоби моніторингу та статистики за допомогою cgroups надають механізми для моніторингу використання ресурсів кожної групи. Тому, можливо

відстежувати статистику про використання процесорного часу, пам'яті, мережі та інших параметрів.

Для динамічного керування можливо динамічно змінювати обмеження ресурсів для груп процесів, що дозволяє адаптувати ресурси в реальному часі відповідно до потреб.

Застосування в контейнерах за допомогою cgroups широко використовуються в контейнерах, таких як Docker. Вони дозволяють ізолювати та обмежувати ресурси, які доступні для контейнерів.

Таким чином, використання cgroups дозволяє оптимізувати використання ресурсів, підвищити стійкість системи та дозволяє розробникам та адміністраторам системи зміцнити контроль над ресурсами, які використовують процеси та групи процесів.

При цьому, команди nice і renice дозволяють користувачам змінювати пріоритети процесів. nice встановлює пріоритет при запуску процесу, а renice дозволяє змінювати пріоритет існуючого процесу. Зміна приоритету процесу за допомогою команд nice та renice - це спосіб контролювати пріоритет виконання процесу в операційній системі Linux. Пріоритет визначає, наскільки процес є "дружнім" для інших процесів, і впливає на те, як обчислювальний час процесора розподіляється між процесами.

Основні інструменти для зміни пріоритету процесу полягають у використанні команди nice для запуску процесу зі зміненим пріоритетом. Вона приймає значення від -20 до +19, де -20 є найвищим пріоритетом, а +19 - найнижчим. Щоб встановити пріоритет, необхдіно ввести nice перед командою. Наприклад:

nice -n 10 some_command

Це запустить some_command з пріоритетом 10.

Команда renice використовується для зміни пріоритету вже існуючого процесу. Вона також приймає значення від -20 до +19 і ідентифікатор процесу (PID). Наприклад:

renice -n 5 -p 12345

Ця команда змінить пріоритет процесу з PID 12345 на 5.

Зміна пріоритету за допомогою nice та renice корисна для користувачів та адміністраторів систем, які хочуть контролювати використання процесорних ресурсів і надавати пріоритет важливим завданням або, навпаки, обмежувати ресурси менш важливим задачам. Те, яким чином ця команда впливає на продуктивність системи, залежить від її розумного використання.

Linux підтримує realtime процеси, які мають вищий пріоритет і гарантований доступ до CPU ресурсів. Це корисно для додатків, де важлива точність виконання в реальному часі. Процеси реального часу (Real-time processes або RT processes) - це особливий тип процесів в операційній системі, які мають високий пріоритет і гарантовану можливість виконання в обмеженому часі. Вони призначені для систем, де важливо забезпечити негайне та детерміноване виконання певних завдань без затримок.

Процеси реального часу мають надзвичайно високий пріоритет в порівнянні з іншими процесами в системі. Це дозволяє їм відразу отримувати доступ до процесора. Важливою особливістю RT процесів є гарантований час виконання їх завдань. Це означає, що система гарантує, що RT процес виконається в обмеженому часовому інтервалі.

В загальному випадку, процеси реального часу часто використовуються в системах, де детермінована відповідь на події є критично важливою, таких як системи управління реакцією на аварії, автопілоти літаків та інші вбудовані системи. Тому, процеси RT можуть мати дедлайни, що вказують максимальний допустимий час виконання задачі.

При цьому, для керування RT процесами використовуються спеціалізовані алгоритми планування, які дозволяють гарантувати готовність процесу вчасно та відразу [12].

Зміна пріоритету RT процесів зазвичай вимагає спеціальних дозволів і уважної обробки, оскільки ці процеси мають пріоритет перед усіма іншими. Процеси реального часу широко використовуються в сферах, де важлива безпека

та надійність, таких як медичні прилади, автомобільна промисловість, аерокосмічна промисловість та інші критичні застосування.

Таким чином, процеси реального часу грають ключову роль у системах, де важлива гарантована реакція на події та мінімізація затримок. Однак важливо ретельно керувати та налагоджувати їх, оскільки помилки в їх конфігурації можуть призвести до серйозних наслідків.

Linux має різні політики планування, такі як SCHED_FIFO, SCHED_RR та SCHED_OTHER, які визначають, як планувальник взаємодіє з процесами. Це означає, що політики планування (Scheduling Policies) в операційній системі Linux визначають правила і алгоритми, за якими процесорний час розподіляється між різними процесами та тредами. Вони впливають на те, як операційна система вирішує, які процеси мають пріоритет і доступ до процесора в данний момент часу.

При цьому, Completely Fair Scheduler (CFS) є стандартною політикою планування в Linux і призначена для забезпечення справедливого розподілу процесорного часу між процесами. Вона використовує алгоритм ваги та балансує завдання на основі їхніх пріоритетів. Ця політика надає справедливу можливість доступу до процесора для всіх процесів.

Політика RR використовує циклічний алгоритм планування, де кожен процес отримує певну квант часу для виконання. Якщо процес не завершиться за відведений час, він переходить до черги інших процесів.

Політика RT використовується для процесів реального часу, де важливий гарантований та детермінований час виконання. Процеси RT мають надзвичайно високий пріоритет та важливість для системи.

Prioritized Based (PRI) дозволяє користувачам встановлювати власні пріоритети для процесів. Вони можуть бути змінені динамічно або вручну.

Політика планування "Batch" призначена для процесів, які не потребують високого пріоритету та можуть виконуватися в фоновому режимі. Політика "Idle" призначена для процесів, які можуть виконуватися тільки тоді, коли немає інших активних задач.

Multi-Level Queue (MLQ) використовує декілька рівнів черг планування для різних типів процесів. Наприклад, користувачі та системні процеси можуть мати різні рівні пріоритету.

Таким чином, кожна з цих політик використовується для конкретних вимог і сценаріїв. Вибір правильної політики планування важливий для досягнення оптимальної продуктивності і стабільності системи. В залежності від потреб і конкретних задач, можуть використовуватися різні політики для різних процесів та груп процесів.

Тому, планування процесів і тредів в Linux - це складний процес, який виконується операційною системою на основі різних алгоритмів та параметрів. Користувачі можуть впливати на планування, змінюючи пріоритети та використовуючи інші методи контролю над виконанням процесів.

1.4. Задачі дослідження

В магістерській роботі необхідно виконати такі задачі.

1. Сформувати задачі дослідження тредів та процесів в операційній системі Linux.

2. Виконати аналіз методів дослідження тредів та процесів в операційній системі Linux.

3. Розглянути аналіз планування процесів та тредів в операційній системі Linux.

4. Провести дослідження тредів та процесів в операційній системі Linux.

5. Визначити особливості дослідження тредів та процесів в операційній системі Linux.

6. Запропоновати метод дослідження роботи процесів та тредів.

7. Розробити вимоги щодо розробки системи управління тредів та процесів.

8. Визначити структурно-функціональні особливості основних модулів програмного засобу.

9. Виконати роботи з реалізації методу досліджень, де здійснити проектування, розробку та впровадження основних модулів програмного засобу.

1.5. Висновки

В першому розділі проведено обгрунтування вибору методу аналізу тредів та процесів в операційній системі Linux, де були визначені задані дослідження. Змістовно описано аналіз методів дослідження тредів та процесів в операційній системі Linux які базуються на використанні різних утиліт та інструментів, що надають детальну інформацію про процеси та потоки. Розглянуто аналіз планування процесів та тредів в операційній системі Linux.

РОЗДІЛ 2
ОСОБЛИВОСТІ ДОСЛІДЖЕНЬ ТРЕДІВ ТА ПРОЦЕСІВ В ОПЕРАЦІЙНІЙ СИСТЕМІ LINUX

2.1. Особливості дослідження тредів в операційній системі Linux

Дослідження тредів (потоків) в операційній системі Linux має свої особливості та вимоги. Основні особливості дослідження тредів в Linux включають наступні компоненти (рис. 2.1).

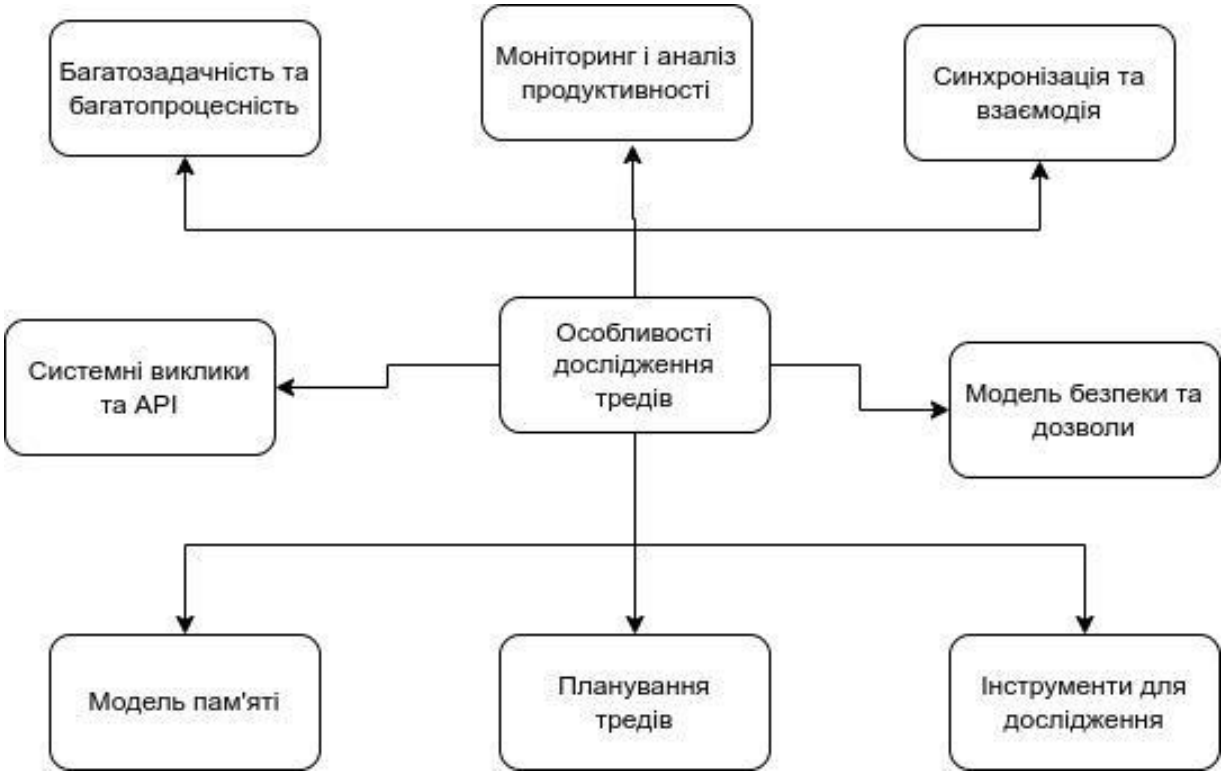


Рисунок 2.1 - Основні особливості дослідження тредів в Linux

Linux - це багатозадачна операційна система, яка підтримує багатопроцесність. Це означає, що дослідження тредів потребує розуміння взаємодії та конкуренції між різними процесами та їхніми тредами.

Ретельніше опишимо особливості дослідження багатозадачності та багатопроцесності в операційній системі Linux. Багатозадачність вказує на можливість операційної системи виконувати одночасно більше одного завдання або процесу. Основні особливості дослідження багатозадачності в Linux полягають в наступному (табл. 2.1).

Таблиця 2.1

Особливості дослідження тредів в операційній системі Linux

Особливість	Опис
Багатозадачність та багатопроцесність	Linux - багатозадачна та багатопроцесна ОС, що вимагає вивчення взаємодії та конкуренції між процесами та тредами.
Системні виклики та API	Розуміння системних викликів та API для створення, управління та спілкування з тредами.
Модель пам'яті	Розуміння моделі пам'яті та проблем спільного доступу до ресурсів між тредами.
Планування тредів	Вивчення принципів планування тредів, включаючи приоритети та розподіл часу.
Інструменти для дослідження	Використання інструментів, таких як GDB, strace, valgrind, ltrace для аналізу та налагодження тредів.
Модель безпеки та дозволи	Розуміння системи дозволів та їх впливу на доступ до ресурсів.
Синхронізація та взаємодія	Вивчення методів синхронізації та взаємодії між тредами, таких як м'ютекси, семафори.
Моніторинг і аналіз продуктивності	Здатність моніторити та аналізувати продуктивність тредів та їх взаємодію з ресурсами.

Linux підтримує багатопроцесність за допомогою процесів та тредів. Дослідження включає вивчення створення, управління та комунікації між ними. Дослідження пов'язане з розумінням принципів планування задач (процесів) для ефективного використання ресурсів, таких як процесорний час та пам'ять.

Важливим аспектом є вивчення контекстів перемикання між процесами або тредами, коли операційна система вирішує, якому процесу або треду надати можливість виконання [12,14].

Багатопроцесність означає, що операційна система може підтримувати одночасну роботу більше одного процесора або ядра. Вивчення можливостей операційної системи для роботи з багатьма процесорами або ядрами, включає механізми мультипроцесності та мультиплексування задач. Дослідження

багатопроцесності також включає в себе аналіз масштабованості додатків та збалансованості навантаження між процесорами або ядрами.

Вивчення різних моделей паралельного програмування, таких як потоки даних, потоки завдань, мультипроцесорні та мультикомп'ютерні обчислення.

Тому, дослідження багатозадачності та багатопроцесності в Linux є важливим для розуміння та оптимізації роботи програмних додатків у вимогливих середовищах, де одночасно виконується багато задач або використовується багато ресурсів.

Дослідження тредів в Linux вимагає розуміння системних викликів і API, які використовуються для створення, управління та спілкування з тредами. Важливо знати, як викликати функції, такі як pthread_create, pthread_join, pthread_mutex_lock тощо.

Розглянемо докладніше особливості дослідження системних викликів та API в операційній системі Linux. Системні виклики є способом, яким програми взаємодіють з ядром операційної системи. Дослідження системних викликів включає в себе розуміння основних функцій, які можна викликати з програмного коду, щоб отримати доступ до операційних ресурсів (наприклад, файлів, мережі, пам'яті тощо).

Для кожної операції існує відповідний системний виклик. Дослідження включає вивчення списку доступних системних викликів, їхніх аргументів і зразків виклику.

Особливості безпеки, такі як контроль доступу до системних викликів і аутентифікація користувачів, також важливі для дослідження. Розуміння, як забезпечується безпека та ідентифікація користувачів, може допомогти у виявленні потенційних загроз та слабких місць.

Для відлагодження програм на рівні системних викликів можуть використовуватися інструменти, такі як strace або GDB. Дослідження включає в себе розуміння, як можна використовувати ці інструменти для відстеження викликів і виявлення проблем наступним чином.

API в Linux включає в себе багато бібліотек і функцій, які можна використовувати для розробки програм. Дослідження API передбачає розуміння доступних бібліотек, їх функцій та параметрів.

Для дослідження API важливо використовувати документацію, яка надається розробниками операційної системи. Це допомагає зрозуміти, як використовувати функції та які аргументи передавати.

Деякі API можуть бути специфічними для певних дистрибутивів Linux або архітектур. Дослідження таких особливостей важливо для забезпечення кросплатформеності додатків. Розуміння API дозволяє ефективно відлагоджувати і оптимізувати програми, враховуючи використання ресурсів та взаємодію з системою.

Дослідження системних викликів та API важливо для розробки високопродуктивних і надійних програм для Linux, а також для забезпечення їхньої безпеки та відповідності стандартам.

В Linux, як і в інших Unix-подібних системах, існує розширена підтримка роботи зі спільною пам'яттю між тредами. Дослідження вимагає розуміння моделі пам'яті та проблем, пов'язаних із забезпеченням безпеки спільного доступу до ресурсів.

Дослідження моделі пам'яті в операційній системі Linux є важливим аспектом для розробників та дослідників, оскільки вона визначає, як програми отримують доступ до пам'яті і взаємодіють з нею.

Основні сегменти пам'яті, такі як стек і купа, досліджуються для розуміння способу виділення та звільнення пам'яті, а також для виявлення можливих переповнень стеку або сегмента купи.

Дослідження сегментів даних і коду включає в себе вивчення доступу до змінних, констант, функцій та інших об'єктів у пам'яті. Це важливо для оптимізації та безпеки програм. Дослідження може включати в себе відлагодження програми для виявлення помилок у роботі з пам'яттю, таких як витік пам'яті або доступ до звільненої пам'яті.

Розуміння моделі пам'яті в умовах багатозадачності і конкурентності важливо для уникнення гонок за ресурсами та інших конфліктів взаємодії. Дослідження включає в себе вивчення функцій та системних викликів, які дозволяють програмам взаємодіяти з пам'яттю, таких як malloc, free, mmap, munmap і багато інших.

В контексті багатозадачності і багатопроцесності дослідження моделі пам'яті включає вивчення, як кожен процес або потік взаємодіє зі своєю власною пам'яттю та може чи не може отримувати доступ до пам'яті інших процесів [12].

Дослідження моделі пам'яті допомагає виявити і вирішити проблеми безпеки, такі як переповнення буфера або вразливості, пов'язані з роботою з пам'яттю.

Таке дослідження може включати в себе оптимізацію роботи з пам'яттю для досягнення кращої продуктивності програм. Дослідження моделі пам'яті є ключовим етапом у вивченні і розробці програм для Linux, оскільки вона визначає, як програми взаємодіють з пам'яттю та ресурсами системи. Розуміння цих особливостей допомагає розробникам створювати безпечні, ефективні і стабільні програми.

Linux має власну систему планування тредів, і дослідження включає в себе аналіз та вивчення принципів планування тредів, таких як пріоритети, розподіл часу тощо.

Дослідження планування потоків (тредів) в операційній системі Linux є важливим аспектом для розробників та дослідників. Планування тредів визначає, як операційна система розподіляє обчислювальні ресурси між потоками, які працюють паралельно.

Таке дослідження включає в себе розуміння алгоритмів планування, таких як Round Robin, Completely Fair Scheduler (CFS) або Real-Time Scheduler. Кожен алгоритм має свої особливості і визначає, як потоки отримують доступ до процесора.

Вивчення того, як встановлюються пріоритети для потоків і як розподіляються ваги, допомагає розуміти, як вирішується конфлікт за ресурсами.

Дослідження може включати в себе вимірювання часу виконання потоків та аналіз того, наскільки воно відповідає плану. Важливо досліджувати затримку і обривання потоків для підтримки вимог до реального часу. Дослідження включає в себе вивчення того, як потоки розподіляються між різними ядрами CPU і як вони використовують пам'ять. Розуміння того, як блокуючі операції впливають на планування та продуктивність потоків.

Вивчення механізмів синхронізації, таких як мутекси та семафори, для уникнення гонок за ресурсами. Дослідження може включати в себе аналіз можливих станів, що призводять до deadlocks або livelocks.

Використання інструментів профілювання для аналізу використання процесорного часу та інших ресурсів потоками. Дослідження може включати в себе методи балансування навантаження для оптимізації роботи потоків.

Таке дослідження планування тредів в операційній системі Linux допомагає розробникам створювати програми, які ефективно використовують ресурси системи, мінімізують затримки та відповідають вимогам до продуктивності і надійності.

Щоб досліджувати треди в Linux, корисно мати доступ до інструментів для аналізу та налагодження, таких як GDB (GNU Debugger), strace, valgrind, ltrace тощо.

Дослідження інструментів для аналізу та діагностики процесів і тредів в операційній системі Linux є ключовим завданням для розробників та адміністраторів систем.

Інструменти для визначення, які частини програми витрачають більше часу та ресурсів. Наприклад, Valgrind, Perf, GProf. Виявлення витоків пам'яті та аналіз використання пам'яті. Наприклад, Valgrind, Massif.

Інші програмні засоби такі які виконують відлагодження та аналіз помилок, наприклад GDB необхідно як дебагер для відлагодження програм, аналізу стеку викликів та відстеження значень змінних.

Програмний засіб Strace виконує відстеження системних викликів, що допомагає виявити проблеми з введенням/виведенням та файловою системою.

Ltrace робить відстеження бібліотек, що використовуються програмою, для аналізу викликів функцій.

Аналіз ресурсів і продуктивності виконується за допомогою програм Top і Htop, які роблять відображення використання CPU і пам'яті процесами та потоками в реальному часі.

Програма Vmstat необхідна для моніторингу використання віртуальної пам'яті та інших системних ресурсів.

Програма Iostat необхідна для відстеження використання дискового простору та операцій з введенням/виведенням.

Аналіз конкурентності та синхронізації виконується за допомогою Pthreads, де досліджується відстеження діяльності потоків та блокувань у програмі. Програма MutexChecker здійснює виявлення помилок синхронізації за допомогою мутексів.

Аналіз мережевої активності виконується за допомогою наступних програм. Wireshark виконує аналіз пакетів мережевого трафіку для виявлення проблем мережі та взаємодії програм. Програма Netstat здійснює відстеження мережевих з'єднань та портів.

Для дослідження аналізу безпеки та аудиту виконуються Lynis і OpenVAS для аналізу безпеки системи та виявлення потенційних вразливостей. AIDE і Tripwire необхідні для виявлення змін в системних файлових структурах та контроль цілісності файлів.

Моніторинг ресурсів та подій здійснюється за допомогою Syslog і rsyslog. Вони здійснюють збір і аналіз системних журналів подій. Інші програми, такі як Nagios і Zabbix виконують моніторинг стану системи та сервісів в реальному часі [12-14].

Аналіз процесів та тредів здійснюється за допомогою Ps і top. Вони виводять інформацію про активні процеси та потоки.

Програма lsof здійснює виявлення відкритих файлових дескрипторів процесами.

Таким чином, дослідження і використання цих інструментів допомагає здійснювати контроль та аналіз різних аспектів процесів і тредів в операційній системі Linux, що сприяє вдосконаленню продуктивності та стабільності системи [14].

Також, Linux має розвинуту модель безпеки з контролем доступу до ресурсів. Дослідження тредів вимагає розуміння системи дозволів та їх впливу на доступ до об'єктів. Дослідження моделі безпеки та дозволів в операційній системі Linux є важливим завданням для забезпечення безпеки і контролю доступу до ресурсів системи.

Дослідження моделей доступу до ресурсів дуже важливий у випадку використання таких механізмів як дискреційний (DAC) та мандаторний (MAC) контроль доступу. Linux підтримує обидві ці моделі, і дослідження дозволяє зрозуміти, як вони застосовуються до файлів, процесів та інших об'єктів.

Дослідження методів автентифікації користувачів (наприклад, пароль, ключі) і визначення їхніх прав доступу. Розуміння і налаштування політик безпеки, таких як SELinux, AppArmor або аналогічних систем, які обмежують дії процесів та користувачів.

Дослідження прав доступу до файлів та каталогів, що визначаються за допомогою бітів доступу (читання, запис, виконання) для власника, групи та інших користувачів.

Вивчення розширених списків контролю доступу таких як ACL (Access Control Lists) дозволяють виконати більш гнучкий контроль над правами доступу до файлів та каталогів.

Ієрархічні права важливі для дослідження та розуміння того, як ієрархія файлової системи впливає на спадкування прав доступу. Дослідження журналу аудиту для відстеження подій, що стосуються безпеки та аналізу дій користувачів та процесів.

Вивчення логів подій для виявлення потенційних загроз безпеці та вчасної реакції на них.

При цьому, поширено використовуються LDAP і Kerberos для дослідження інтеграції з системами централізованої ідентифікації та автентифікації. Firewall і IDS/IPS виконується для налаштування систем захисту мережі та виявлення вторгнень.

Розробка та тестування власних політик безпеки досліджується за допомогою SELinux та AppArmor з метою вивчення та створення власних політик безпеки для програм та служб.

Таким чином, дослідження моделі безпеки та дозволів в операційній системі Linux сприяє покращенню безпеки системи, забезпеченню відповідності правилам безпеки та захисту важливих даних і ресурсів.

Дослідження включає в себе вивчення методів синхронізації та взаємодії між тредами, таких як м'ютекси, семафори, атомарні операції тощо. Дослідження синхронізації та взаємодії процесів і тредів в операційній системі Linux є ключовим аспектом для забезпечення коректної та ефективної роботи багатозадачних програм.

При цьому, використовуються наступні моделі синхронізації. Семафори для дослідження використання семафорів для координації доступу до ресурсів, які мають обмежений доступ. М’ютекси - для розуміння принципів роботи м’ютексів та їх використання для забезпечення взаємовиключного доступу до об'єктів. Дослідження використання умовних змінних для реалізації чекання та сигналізації між процесами.

Міжпроцесорна взаємодія досліджується на рівні пайпів та FIFO для вивчення роботи інтерпроцесорних каналів та іменованих каналів (FIFO) для обміну даними між процесами.

Дослідження використання сокетів для мережевої взаємодії між процесами, як на одній системі, так і через мережу. Спільний доступ до пам'яті (Shared Memory) пов'язано з розумінням принципів використання спільної пам'яті для обміну даними між процесами.

Визначення моделей взаємодій виконується за допомогою продюсер-консумент для Дослідження використання моделі "продюсер-консумент" для

спільного доступу до обмежених ресурсів. Читач-письменник використовується для вивчення взаємодії між процесами, які читають та записують дані до спільного ресурсу. Патерн "Обробник" (Observer) для дослідження різних способів реалізації патерна "Обробник" для асинхронної взаємодії між об'єктами.

Тестування синхронізації та взаємодії виконується на рівні юніт-тестів з метою створення тестів для перевірки правильності синхронізаційних механізмів та взаємодії між процесами. Стрес-тести виконуються для вивчення поведінки системи в умовах великого навантаження та конкуренції між процесами. Аналіз помилок здійснюється для виявлення та виправлення помилок синхронізації та взаємодії для забезпечення стабільності програми.

Асинхронна та багатопотокова взаємодія необхідна для Callback-функції з метою дослідження використання callback-функцій для асинхронної обробки подій [14,15]. Пул потоків (Thread Pool) необхідний для розуміння принципів роботи та використання пулів потоків для ефективної обробки багатьох завдань одночасно.

Таким чином, дослідження синхронізації та взаємодії грає критичну роль у розробці багатозадачних і багатопроцесних програм в операційній системі Linux. Воно допомагає уникнути гонок, дедлоків та інших проблем синхронізації, забезпечуючи вірну та ефективну роботу програми.

Для дослідження тредів важливо вміти моніторити їхню роботу та аналізувати продуктивність. Інструменти, які надає система, також можуть бути використані для аналізу ресурсів. Дослідження моніторингу та аналізу продуктивності в операційній системі Linux має важливе значення для забезпечення ефективності та оптимізації роботи системи.

Для дослідження важливими є вивчення інструментів, які надає сама операційна система Linux для моніторингу, такі як top, htop, vmstat, sar, iostat та інші. Дослідження сторонніх інструментів для моніторингу, таких як Nagios, Zabbix, Prometheus, Grafana, які дозволяють стежити за роботою системи та додатків у реальному часі.

Робота таких важливих параметрів моніторингу як центральний процесор (CPU) необхдіно для вивчення завантаженості процесора, розподілу завдань між ядрами та визначення причин перевищення обсягу обчислювальних ресурсів. Робота оперативної пам'яті (RAM) використовується для аналізу використання пам'яті, виявлення витоків пам'яті та вирішення проблем з обсягом доступної RAM.

Моніторинг швидкості читання/запису на диск необхідно для визначення обсягу вільного простору, оптимізація роботи з файловими системами.

Вивчення обсягу мережевого трафіку досліджується для виявлення найбільш активних мережевих з'єднань та протоколів.

Досить важливими є використання інструментів для профілювання програмного коду з метою виявлення вузьких місць та непотрібних операцій.

Оптимізація запитів до баз даних необхідна для покращення продуктивності баз даних шляхом оптимізації запитів, використання кешування і індексів. Моніторинг додатків потрібен для слідкування за роботою конкретних додатків, виявлення проблем в їх архітектурі та оптимізація роботи.

Таким чином, масштабування інфраструктури важливе для визначення потреби у розширенні ресурсів (процесора, RAM, дискового простору) з метою забезпечення зростаючої продуктивності.

Виявлення аномалій та проблем за допомогою аналізу даних моніторингу та виявлення аномалій вказують на можливі проблеми з продуктивністю.

Оптимізація і прогнозування полягає в розробці стратегій оптимізації та прогнозування для забезпечення стійкої та ефективної роботи системи у майбутньому.

Тому, дослідження моніторингу та аналізу продуктивності в операційній системі Linux дозволяє виявляти, вирішувати та попереджувати проблеми з продуктивністю, що є важливим для забезпечення ефективної роботи програм та інфраструктури.

Таким чином, дослідження тредів у Linux вимагає глибокого розуміння операційної системи, її інфраструктури та інструментів, що підтримуються.

2.2. Особливості дослідження процесів в операційній системі Linux

Дослідження процесів в операційній системі Linux - це важлива складова роботи системного адміністратора, програміста або дослідника, яка дозволяє розуміти і вивчати, як процеси працюють в операційній системі Linux. Розглянемо деякі особливості дослідження цих процесів (рис. 2.2).

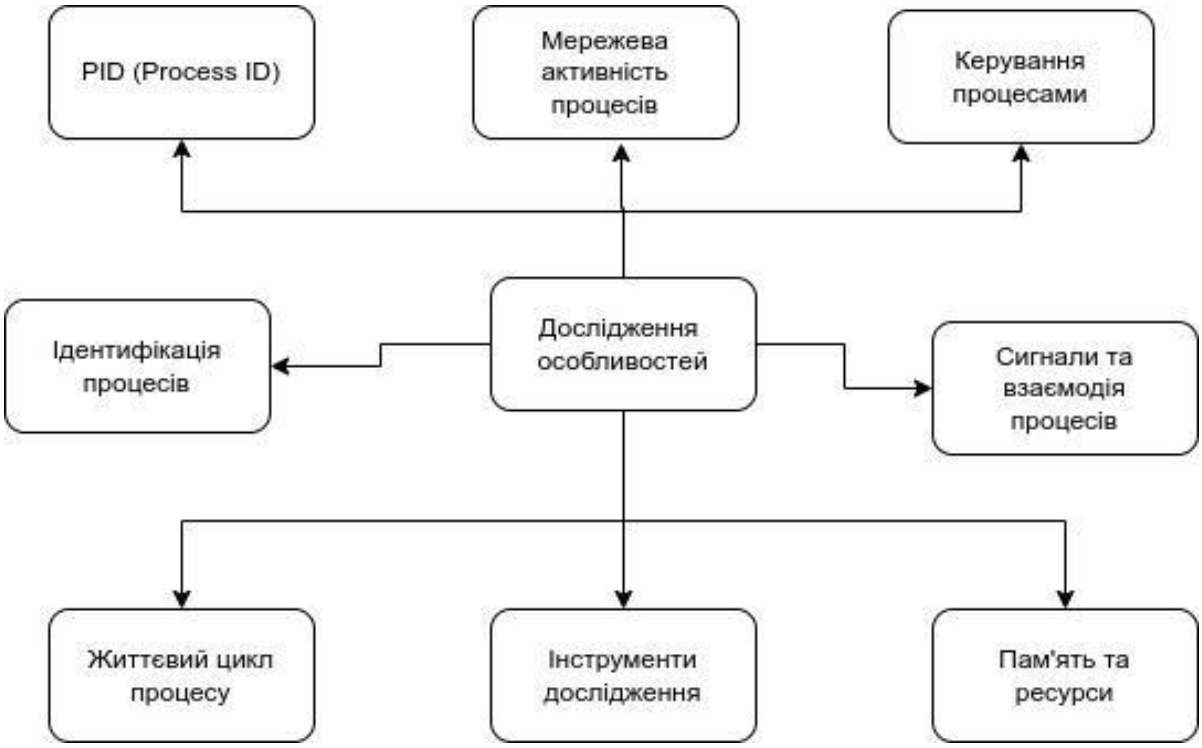


Рисунок 2.2 - Деякі особливості дослідження процесів в операційній системі Linux

В Linux все, що існує, відбувається через процеси. Процеси - це програми або задачі, які виконуються в операційній системі. Загальне розуміння процесів в Linux - це ключовий аспект для розуміння роботи операційної системи. Процеси є основними одиницями обробки завдань в Linux і виконують різні функції, включаючи запуск програм, управління ресурсами та взаємодію з іншими процесами. Розглянемо основні аспекти, які допоможуть краще описати процеси в Linux.

Кожен процес в Linux має унікальний ідентифікатор, який називається PID. Це числове значення, яке ідентифікує конкретний процес. PID надає засоби для посилання на конкретний процес під час управління ним.

Процеси проходять різні стани в своєму життєвому циклі. Основні стани включають створення (creation), готовність (ready), виконання (running), очікування (waiting), завершення (terminated) та знищення (termination). Процеси можуть переходити між цими станами залежно від їхнього стану та взаємодії з системою [14-16].

Процеси в Linux можуть бути створені іншими процесами. Батьківський процес, який створює інший процес, називається батьківським процесом, а процес, який був створений, - дочірнім. Це утворює ієрархію процесів, де дочірні процеси спадають від свого батька.

Кожний процес має свою власну область пам'яті, включаючи стек, купу та сегмент даних. Вона також має свій власний PID, середовище, дескриптори файлів та інші ресурси. Кожний процес має обмежений доступ до ресурсів, що допомагає визначити, як вони взаємодіють з іншими процесами та системою.

Процеси можуть взаємодіяти один з одним та з системою через різні механізми, такі як сигнали, черги, спільний доступ до пам'яті і файли. Це дозволяє процесам співпрацювати та обмінюватися інформацією.

Процеси можуть завершувати свою роботу, виходячи з програми або через відомий сигнал (наприклад, SIGTERM). Після завершення процесу його ресурси звільняються, і він стає зомбі-процесом, поки батьківський процес не отримає інформацію про його завершення. При цьому можливо використовувати різні інструменти і команди, такі як ps, top, htop, kill, strace тощо, для моніторингу та керування процесами в Linux (табл. 2.2).

Таблиця 2.2

Опис деяких особливостей дослідження процесів в операційній системі Linux

Фактор	Опис
PID (Process ID)	Унікальний ідентифікатор процесу, що використовується для ідентифікації кожного процесу в системі.
Ідентифікація процесів	Процеси ідентифікуються за допомогою їхнього PID та імені. Інші атрибути також використовуються для ідентифікації та управління процесами.
Життєвий цикл процесу	Процеси проймають різні стани від створення до завершення: створення, виконання, призупинення, завершення тощо.
Інструменти дослідження	Різні інструменти, такі як ps, top, htop, lsof, strace, netstat, iotop, vmstat, iostat тощо, використовуються для аналізу та вивчення процесів.
Пам'ять та ресурси	Процеси використовують пам'ять (фізичну та віртуальну) та різні ресурси (CPU, I/O, мережу тощо) для виконання своїх завдань. Можливе обмеження ресурсів.
Сигнали та взаємодія процесів	Процеси можуть надсилати та отримувати сигнали для спілкування та контролю. Це важливо для взаємодії та відлагодження.
Керування процесами	Операційна система відповідає за створення, планування, призупинення, відновлення та завершення процесів. Адміністратори та користувачі також можуть впливати на цей процес.
Мережева активність процесів	Процеси можуть взаємодіяти з мережею через мережеві порти та протоколи, надсилаючи та отримуючи дані через мережеві інтерфейси. Можна використовувати інструменти, такі як netstat та tcpdump, для моніторингу мережевої активності.

Загалом розуміння процесів в Linux допомагає керувати ресурсами системи, виявляти проблеми та аналізувати їх для підвищення продуктивності та надійності операційної системи.

Кожен процес має унікальний ідентифікатор PID (Process ID). Ідентифікація процесів в операційній системі Linux - це процес призначення унікальних числових значень кожному активному процесу PID (Process ID).

Ця ідентифікація дозволяє системі та користувачам однозначно визначити та посилатися на конкретні процеси.

Кожен процес має унікальний PID, який ніколи не повторюється в межах системи в данний момент часу. Це означає, що ні два процеси в системі не можуть мати однаковий PID. PID призначається процесам системою при їхньому створенні. Перший процес, який створюється під час завантаження системи, має PID 1 і називається "init" (в більш нових версіях Linux замінили на "systemd"). Подальші процеси отримують PID в порядку їхнього створення.

PID використовується для ідентифікації конкретних процесів під час їхнього керування. Наприклад, ви можете використовувати команду kill, щоб надіслати сигнал до конкретного процесу за його PID для припинення роботи цього процесу.

Тому, у системі існують обмеження на значення PID. Зазвичай, максимальне значення PID обмежено 32 767 або більше, в залежності від конфігурації системи. Після досягнення максимального значення, PID починають переводитися знову від 1, переповнюючи значення. При цьому, можливо перевірити PID активних процесів, використовуючи різні команди та інструменти, такі як ps, top, htop, pgrep, pstree, або команди статусу системи. Ці інструменти надають інформацію про PID, назву процесу, власника тощо.

Розуміння ідентифікації процесів важливе для ефективного керування та моніторингу процесів в операційній системі Linux. PID - це ключовий параметр, що дозволяє взаємодіяти з конкретними процесами і виконувати операції, такі як запуск, припинення, відновлення, надсилання сигналів і багато інших.

Процеси в Linux проходять різні стани в життєвому циклі, включаючи створення, готовність, виконання, очікування, завершення і знищення. Дослідження цих станів може допомогти вам розуміти, як процеси взаємодіють між собою та операційною системою. Життєвий цикл процесу - це послідовність станів, через які процес проходить в операційній системі від моменту його створення до завершення або призупинення. Розуміння життєвого циклу процесу

важливо для ефективного моніторингу, керування та відлагодження програм в Linux.

Процес починає своє існування під час створення. Це стан, в якому процес отримує свій PID та ініціалізується. Після цього він може переходити в стан готовності. У стані готовності процес готовий до виконання, але його виконання ще не почалося. Процес чекає, коли CPU буде готовий виконувати його інструкції. Зазвичай цей стан виникає, коли багато процесів конкурують за ресурси процесора. Виконання - це стан, в якому процес активно виконується на процесорі. В цей момент він виконує свої інструкції і взаємодіє з системою.

Після виконання певних операцій або подій, процес може перейти в стан очікування, де він тимчасово призупинений і чекає на подію або ресурс. Наприклад, процес може очікувати завершення введення/виведення або отримання сигналу. Процес завершується, коли він виконав всі необхідні дії і закінчив свою роботу. Після завершення процес може повернути результати своєї роботи та бути видаленим з системи.

Інколи процес може застрягти в якому-небудь стані, де він не може продовжити виконання і не завершується. Це може статися через програмні помилки або проблеми з ресурсами. Деякі процеси можуть спати або бути призупиненими до певного події або запуску. Вони не витрачають ресурси CPU під час сплячого стану.

Після завершення процесу він може залишити свій слід у вигляді зомбі-процесу. Зомбі - це процес, який завершив виконання, але має ще певну інформацію для батьківського процесу. Він чекає, доки батьківський процес не отримає цю інформацію, і тоді він буде повністю видалений з системи.

Розуміння життєвого циклу процесу допомагає відстежувати та аналізувати роботу програм в системі, виявляти потенційні проблеми, такі як зависання або зомбі-процеси, і керувати ресурсами оптимальним чином.

В Linux існує багато інструментів для дослідження процесів, таких як ps, top, htop, strace, lsof, pstree та інші. Вони дозволяють переглядати і аналізувати інформацію про процеси в реальному часі. Інструменти дослідження в

операційній системі Linux - це програми та команди, які допомагають користувачам та системним адміністраторам аналізувати та вивчати різні аспекти системи, включаючи процеси, ресурси, мережеву активність та багато інше. Ці інструменти допомагають вирішувати проблеми, оптимізувати роботу системи та здійснювати моніторинг.

Розглянемо декілька інструментів дослідження в Linux та їхні основні функції [14-16].

ps - це команда, що дозволяє переглядати інформацію про активні процеси в системі. При цьому, можливо вивести список процесів, їхні PID, стани, споживану пам'ять та іншу інформацію.

top та htop - це інтерактивні інструменти для моніторингу системи в реальному часі. Вони надають інформацію про завантаженість процесора, використання пам'яті та інші системні метрики.

lsof - це команда, що дозволяє переглядати відкриті файлові дескриптори для процесів. Ви можете дізнатися, які файли та ресурси використовуються процесами.

strace - це інструмент для трасування системних викликів, який дозволяє вивчати, які системні виклики виконуються процесами. Це корисно для відлагодження та аналізу роботи програм.

netstat та ss - це інструменти для аналізу мережевої активності. Вони дозволяють переглядати інформацію про мережеві підключення, порти та інші мережеві параметри.

iotop - допомагає відстежувати введення/виведення (I/O) на диски та переглядати, які процеси використовують дисковий ресурс.

pstree - це інструмент, який допомагає візуалізувати ієрархію процесів в системі, показуючи їхній зв'язок з батьківськими та дочірніми процесами.

vmstat - надає інформацію про використання віртуальної пам'яті, включаючи статистику обміну та активність диска.

iostat - дозволяє аналізувати використання системи вводу/виводу (I/O) та моніторити активність диска.

Ці інструменти допомагають користувачам і адміністраторам отримувати інсайти в роботу системи, виявляти проблеми, оптимізувати ресурси та вести моніторинг для забезпечення ефективності та стабільності операційної системи Linux. Вибір конкретного інструменту залежить від конкретних завдань та завдань, які необхідно виконати.

Процеси в Linux використовують пам'ять і інші ресурси системи. Дослідження споживання пам'яті, центрального процесора та інших ресурсів процесом може допомогти в оптимізації роботи системи.

В операційній системі Linux пам'ять та ресурси відіграють важливу роль у функціонуванні і виконанні процесів. Розуміння цих понять допомагає у вирішенні питань ефективності та надійності системи.

Оперативна пам'ять (RAM) являє собою фізичну пам'ять комп'ютера, яка використовується для зберігання виконуваного коду та даних процесів в реальному часі. RAM надає швидкий доступ до даних, які використовуються процесами під час їхнього виконання.

Linux використовує віртуальну пам'ять, яка розширює можливості фізичної RAM, дозволяючи динамічно розподіляти пам'ять між процесами та диском. Віртуальна пам'ять дозволяє запускати більше процесів, ніж фізична пам'ять може вмістити [19], і забезпечує підтримку обміну даними між RAM і диском.

Пам'ять поділяється на невеликі однакові частини, які називаються сторінками пам'яті. Система керування пам'яттю Linux використовує техніку сторінок для ефективного виділення та використання пам'яті.

Пам'ять ядра (Kernel Memory) - це область пам'яті, яка використовується операційною системою для зберігання ядра Linux та важливих системних даних [12,14].

До ресурсів (Resources) операційної системи Linux відноситься центральний процесор (CPU), який є обчислювальним ресурсом та виконує інструкції процесів. Linux керує розподілом CPU між процесами, використовуючи алгоритми планування.

Також є ресурси вводу/виводу (I/O), що включають в себе пристрої для зчитування та запису даних, такі як диски, мережеві карти та інші пристрої. Вони використовуються процесами для комунікації з ними. Мережеві ресурси включають в себе мережеві інтерфейси та ресурси, які дозволяють процесам взаємодіяти з мережею.

Файлові системи, як ресурси, дозволяють зберігати та отримувати доступ до файлів та каталогів. Ресурси файлової системи включають в себе доступ до файлів, дозволи на читання та запис, обмеження місця на диску і багато інших параметрів.

Розуміння пам'яті та ресурсів допомагає користувачам та адміністраторам ефективно керувати ресурсами системи, моніторити їх використання та виявляти проблеми. Користувачі і адміністратори повинні бути уважними до розподілу ресурсів між процесами, щоб забезпечити стабільну та продуктивну роботу операційної системи Linux.

Процеси можуть взаємодіяти один з одним за допомогою сигналів. Дослідження цієї взаємодії допомагає розуміти, як взаємодіють різні компоненти системи.

Сигнали та взаємодія процесів - це важливий аспект операційних систем, включаючи Linux, який дозволяє процесам спілкуватися один з одним та реагувати на різні події. Сигнали - це спеціальні повідомлення, які процеси відправляють один одному або ядро операційної системи для ініціювання певних дій або повідомлення про події.

Сигнали - це короткі повідомлення, що використовуються для сповіщення про події або запити на взаємодію між процесами. Це може бути щось таке, як запит на завершення процесу або повідомлення про помилку. Linux підтримує різні типи сигналів, включаючи сигнали для завершення (наприклад, SIGTERM), зупинки (наприклад, SIGSTOP), перезапуску (наприклад, SIGHUP) та багато інших.

Сигнали можуть бути згаснуті або оброблені, тому процеси можуть згасити (ігнорувати) сигнали або встановити власні обробники сигналів, які виконують певні дії при отриманні сигналу.

Процес може надіслати сигнал іншому процесу, використовуючи команди, такі як kill або killall. Це дозволяє керувати процесами, наприклад, припиняти їх виконання.

Процеси можуть використовувати сигнали для комунікації між собою. Наприклад, один процес може надсилати сигнал іншому процесу для повідомлення про певну подію або дію. Розробники використовують сигнали для відлагодження програм. Наприклад, сигнал SIGUSR1 може бути використаний для виклику власного коду в процесі для відлагодження або збору додаткової інформації.

Сигнали можуть бути ігноровані або перехоплені. Тому, процеси можуть встановити власні обробники сигналів, які виконують певні дії при отриманні сигналу. Наприклад, власний обробник може взяти додаткові дії, після чого виконання процесу може продовжитися.

Таким чином, певні сигнали не можуть бути перехоплені або згаснуті. Це важливо для забезпечення стабільності системи та безпеки.

Сигнали - це потужний механізм для взаємодії процесів та контролю їхнього поведінки в операційній системі Linux. Вони дозволяють процесам спілкуватися, взаємодіяти та відлагоджувати програми, що важливо для стабільності та продуктивності системи.

В Linux є можливість керування процесами, такими як запуск, призупинення, відновлення, припинення та видалення. Дослідження методів керування процесами також є важливою частиною вивчення операційної системи. Керування процесами - це важливий аспект операційних систем, включаючи Linux, який охоплює всі дії та стратегії, пов'язані з управлінням виконанням процесів. Важливо керувати процесами для забезпечення стабільності та ефективності операційної системи.

Процеси створюються в системі під час запуску програм або ініціації операційною системою [16,17]. Кожен процес отримує унікальний ідентифікатор PID (Process ID).

Операційна система відповідає за розподіл процесорного часу між активними процесами. Існують різні алгоритми планування, які визначають, які процеси отримують доступ до CPU в якому порядку.

Процеси можуть бути призупинені (зупинені) і відновлені пізніше. Це корисно для економії ресурсів або розвантаження системи від процесів, які тимчасово не потрібні. Процеси завершуються після завершення своєї роботи або через внутрішній або зовнішній запит. Операційна система звільняє ресурси, які були призначені для завершеного процесу.

Процеси можуть взаємодіяти один з одним або з ядром операційної системи для обміну даними або взаємодії. Синхронізація дозволяє контролювати порядок виконання процесів та уникнути конфліктів.

Операційна система може накладати обмеження на ресурси, доступні для процесів, такі як обмеження пам'яті, кількість використаних CPU часу тощо.

Операційна система дозволяє користувачам взаємодіяти з процесами через командний рядок, графічний інтерфейс та інші способи. Тому, адміністратори системи та користувачі можуть моніторити та керувати процесами за допомогою інструментів, таких як ps, top, kill, htop, тощо.

Операційна система встановлює механізми безпеки та права доступу для захисту процесів і ресурсів від несанкціонованого доступу та зловмисних дій. Керування процесами - це важливий аспект операційних систем і грає важливу роль у забезпеченні стабільності та продуктивності системи. Воно дозволяє ефективно розподіляти та керувати ресурсами, забезпечувати безпеку та контролювати виконання процесів.

Також, існує необхідність досліджувати мережеву активність процесів за допомогою інструментів, таких як netstat або ss, щоб з'ясувати, як процеси взаємодіють з мережею. Мережева активність процесів в операційній системі Linux означає взаємодію між процесами та мережею, коли процеси відправляють

або отримують дані через мережеві інтерфейси. Це важливий аспект для багатьох додатків, які потребують мережевого зв'язку, таких як веб-сервери, поштові сервери, браузери та багато інших.

Процеси можуть надсилати дані через мережу на інші системи або отримувати дані з мережі. Це може бути взаємодія з іншими комп'ютерами, веб-сервісами або внутрішніми мережевими службами. Для взаємодії з мережею процеси використовують мережеві порти. Кожний мережевий порт призначений для конкретної служби або протоколу. Наприклад, HTTP сервери зазвичай прослуховують порт 80, а сервери електронної пошти використовують порт 25.

Кожен комп'ютер у мережі має свою IP-адресу, яка ідентифікує його в мережі. Процеси можуть надсилати дані на конкретні IP-адреси та порти для забезпечення комунікації з іншими системами. Для взаємодії в мережі процеси використовують різні мережеві протоколи, такі як TCP/IP, UDP, HTTP, FTP, SSH і багато інших. Кожен протокол визначає правила обміну даними між системами.

Для аналізу та моніторингу мережевої активності процесів у Linux можна використовувати інструменти, такі як netstat, ss, tcpdump, Wireshark та інші. Ці інструменти дозволяють відстежувати мережевий трафік, перевіряти відкриті порти, аналізувати підключення та багато іншого.

Забезпечення безпеки мережевої активності - це важливий аспект. Це включає в себе встановлення правил брандмауера, шифрування даних, аутентифікацію та інші заходи для захисту мережевої комунікації від несанкціонованого доступу та атак.

Таким чином, мережева активність процесів важлива для забезпечення взаємодії та обміну даними в мережі. Розуміння цього аспекту допомагає адміністраторам та розробникам забезпечувати стабільну та безпечну мережеву активність своїх програм та служб.

Тому, загальна мета дослідження процесів в Linux - це зрозуміти, як операційна система керує процесами, використовує ресурси та забезпечує надійну та ефективну роботу системи. Для цього може знадобитися використання різних інструментів і технік аналізу, включаючи моніторинг,

трасування системних викликів, аналіз системних журналів та багато інших методів.

2.3. Запропонований метод дослідження роботи тредів

Найбільш поширене застосування багатопотокової обробки – це рішення потребують тривалої обробки завдань без гальмування візуального інтерфейсу користувача, для чого зазвичай достатньо одного – двох паралельно працюючих потоків в одному процесі. Але в складніших обчислювальних додатках для ефективного вирішення їхніх завдань може знадобитися кілька десятків робочих потоків, що паралельно працюють.

Тому, для рішення паралельних завдань з метою значного прискорення в обробці та відображення даних можна досягти при використанні комп'ютерів з багатоядерним процесором.

Розглянемо приклад дослідження багатопоточності завдань кількох тредів. Одним із поширених сценаріїв використання багатопоточності є обробка великих обчислювальних завдань у фонових потоках. Це важливо для створення високопродуктивних програм, де користувач може продовжувати взаємодію з інтерфейсом, навіть коли триває тривала обробка складних даних.

У деяких складніших обчислювальних завданнях може знадобитися значна кількість паралельних робочих потоків для ефективного вирішення задачі. Це особливо актуально на багатоядерних процесорах, де безліч потоків можна використовувати для прискорення обробки даних технологічних процесів [16].

Найбільш вдалим у разі багатопотокової обробки даних є приклад із зображенням Мандельброта (рис. 2.3). У наведеному прикладі може бути деяке зображення Мандельброта, де потрібно виконання великої кількості обчислень для кожної точки на екрані. При цьому, розпаралелювання роботи між групою паралельних потоків, які обробляють дані, може значно прискорити процес їх обробки.

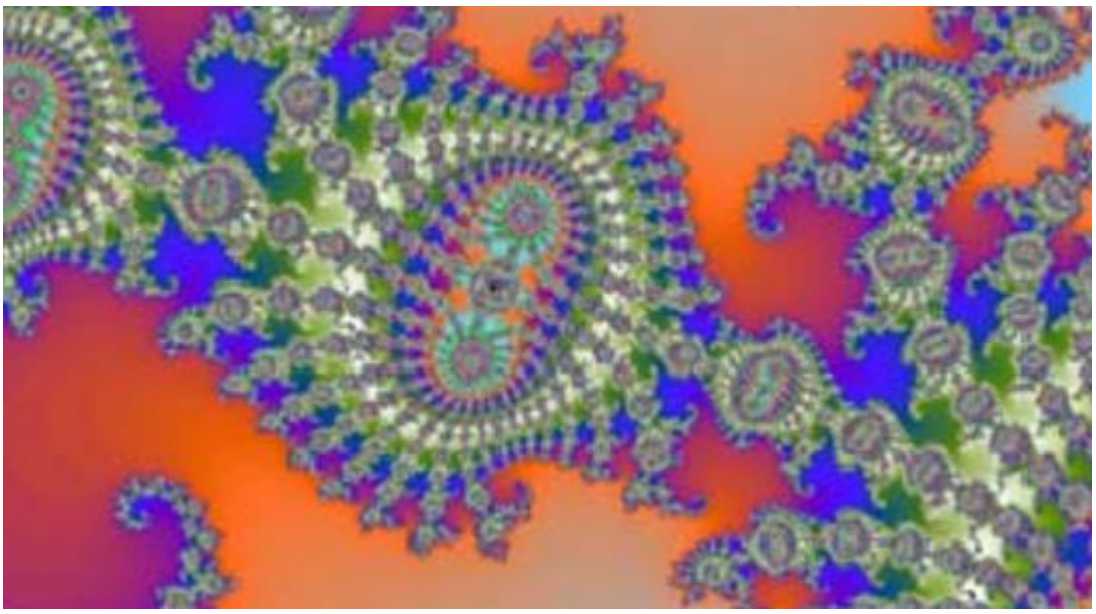


Рисунок 2.3 - Багатопотокове оброблення даних на прикладі побудови зображення Мандельброта

На рис. 2.3 представлено зображення множини Мандельброта [5] - фрактала, визначеного як безліч точок на комплексній площині, для яких не йде в нескінченність ітеративна послідовність наступного виду:

$Z_0 = 0$

$Z_{n+1} = Z_{n+2} + C$

Якщо отримувати наведене на малюнку зображення послідовно за допомогою одного потоку, то на його отримання та відображення на екрані пристрою знадобиться близько 1 секунди.

Так як ітерації для отримання зображення ведуться рядково, обчислення можна розподілити між групою потоків, що паралельно працюють, які ітерують сусідні рядки даного зображення. В результаті час отримання множини в заданих межах і його відображення на екрані може зменшитися в кілька разів, якщо Linux-пристрій побудований на базі кристала з кількома ядрами.

Для цього кожен потік може ітерувати сусідні рядки зображення, що дозволяє скоротити час обробки і покращити продуктивність програми.

У зв'язку з цим реалізовують у додатку автоматичне масштабування зображення в залежності від розміру вікна браузера. Крім того, після обчислення

координат необхідної точки зображення встановлюються нові межі множини Мандельброта із заданим коефіцієнтом масштабування, зазвичай рівним 8, і за допомогою тієї ж групи паралельно працюючих потоків обчислюється новий фрактал, що підтримує співвідношення сторін поточного розміру вікна.

Таким чином, описаний функціонал програми, що працює на платформі Linux і пов'язаного з відображенням фракталів Мандельброта, є гарним прикладом візуалізації та взаємодії із зображенням у реальному часі. У цьому випадку програма автоматично адаптує зображення фрактала під розмір вікна браузера і дозволяє користувачеві керувати межами множини Мандельброта за допомогою торкання точок на зображенні.

При зміні розмірів вікна браузера програма динамічно перераховує та масштабує зображення фрактала, щоб він відповідав новим розмірам вікна. Це дозволяє забезпечити коректне відображення навіть за зміни розміру вікна.

Для прискорення розрахунків програма використовує групу паралельно працюючих потоків для обчислення нових фракталів. Це дозволяє використовувати багатопоточність для збільшення продуктивності та покращення чуйності програми.

В цілому, описаний функціонал є гарним прикладом застосування багатопоточності та інтерактивної взаємодії з графічними даними в Linux-додатку. Він дозволяє створити важливий для дослідження продуктивності візуальний графічний додаток.

Для вирішення цього завдання багато сучасних пристроїв використовують багатоядерні процесори. При цьому ефективне використання багатопоточності дозволяє повністю задіяти обчислювальні ресурси, що призводить до значного збільшення продуктивності.

Важливо зауважити, що правильне розпаралелювання обчислень вимагає врахування таких різних аспектів, таких як синхронізація доступу до загальних даних та оптимальне розподілення навантаження між потоками. Якщо це зроблено правильно, багатопотоковість може значно покращити продуктивність обчислювальних додатків.

Однак, при вирішенні подібних завдань, пов'язаних з багатопоточністю, можуть виникати проблеми, які вирішуються за допомогою багатопоточного процесу та полягають у наступному.

В операційній системі Unix (Linux, Android) будь-який компонент сервісу, який виконує трудомістку за часом завдання в основному потоці (також відомому як UI-потік), може викликати блокування всієї програми до завершення цього завдання. Це відбувається тому, що основний потік відповідає за оновлення інтерфейсу користувача і обробку введення користувача. Тому, якщо він блокується виконанням довгої операції, то додаток перестає відповідати на дії користувача.

Коли основний потік заблоковано, операційна система Unix може попередити користувача з повідомленням «Додаток не відповідає». При цьому користувач може бути змушений закрити додаток. Така поведінка програми є небажаною.

Щоб уникнути блокування основного потоку та забезпечити високу продуктивність програми, трудомісткі завдання слід виконувати в окремих робочих потоках виконання (наприклад, у фонових потоках або у фонових сервісах). Це дозволяє основному потоку продовжувати роботу без переривань, а результати трудомістких операцій можуть бути інтегровані в інтерфейс користувача після завершення.

Вирішення цієї проблеми на методологічному рівні полягає в наступному. Для розробки програмного забезпечення для Linux та інших Unix-подібних операційних систем є суворі правила, які забороняють виконання трудомістких операцій в основному потоці. Це правило є актуальним для всіх додатків, незалежно від операційної системи. Виконання трудомістких операцій в основному потоці може блокувати інтерфейс користувача і знижувати якість програмного продукту. Тому важливо виконувати такі операції в окремих потоках чи процесах.

Наступне правило забороняє оновлення елементів інтерфейсу користувача з інших потоків. Це також принцип, що також поширюється на розробку в

операційній системі Linux та інших платформ. Тому, у багатьох GUI-бібліотеках, що використовуються для створення графічних інтерфейсів на Linux, таких як Qt або GTK, доступ до елементів інтерфейсу здійснюється з основного потоку (звичайно званого "головним циклом подій"). Спроби оновлення інтерфейсу з інших потоків можуть призвести до непередбачуваних проблем та помилок.

Вирішення проблеми на програмному рівні зазвичай досягається шляхом використання механізмів синхронізації та повідомлень між робочим потоком та основним потоком.

В ОС Linux та інших багатозадачних операційних системах можна використовувати різні способи для цієї мети, включаючи наступні.

Handler і Looper, де створюється об'єкт Handler переважно потоці і передається їх у робочий потік. Робочий потік може використовувати цей Handler для надсилання повідомлень або завдань до основного потоку, які будуть виконуватися в контексті основного потоку.

RunOnUiThread(), де існує метод runOnUiThread(), який дозволяє виконати код на основному потоці з фонового потоку. Це дозволяє оновлювати елементи інтерфейсу користувача безпечно.

Використання механізмів синхронізації застосовується для запобігання конкуренції між потоками обробки даних та забезпечення синхронізації між ними. При цьому можна використовувати механізми міжпроцесної взаємодії, такі як блокування, м'ютекси, семафори та інші.

Використання асинхронних завдань дозволяє також застосовувати класи AsyncTask або бібліотеку RxJava для зручної роботи з асинхронними завданнями, включаючи оновлення інтерфейсу з фонових завдань.

Таким чином, дуже важливе дотримання цих принципів для забезпечення безпеки та стабільності багатозадачних додатків. Неправильна взаємодія між робочими потоками та основним потоком може викликати конкуренцію у використанні даних, блокування та помилки, що може спричинити некоректну поведінку програми. Виконання багатопоточного алгоритму (рис. 2.4) полягає у виконанні наступних кроків.

Крок 1. Визначення трудомістких завдань, який полягає у відокремленні операцій, що можуть займати багато часу або ресурсів комп'ютера.

Крок 2. Створення додаткових робочих потоків виконання або процесів для реалізації виконання цих трудомістких завдань. Ці потоки можуть бути фоновими потоками або фоновими процесами (сервісами), що не впливають на основний UI-потік.

Крок 3. Передача трудомістких завдань в окремі потоки з метою переміщення виконання трудомістких операцій з основного потоку у ці додаткові робочі потоки або процеси. Це дозволяє основному потоку не блокуватися під час виконання цих операцій.

Крок 4. Застосування механізмів синхронізації для взаємодії між робочими потоками та основним потоком. Це може включати в себе передачу повідомлень або задач з робочих потоків до основного потоку для подальшого виконання в контексті основного потоку.

Крок 5. Обмін даними та інтеграція передачі результатів від робочих потоків до основного потоку для подальшої інтеграції цих результатів у користувацький інтерфейс після завершення операцій у робочих потоках.

Крок 6. Використання відповідних методів або інструментів для створення та управління потоками виконання, наприклад, використання Handler і Looper, або методів типу runOnUiThread() для доступу до основного UI-потоку з фонових потоків.

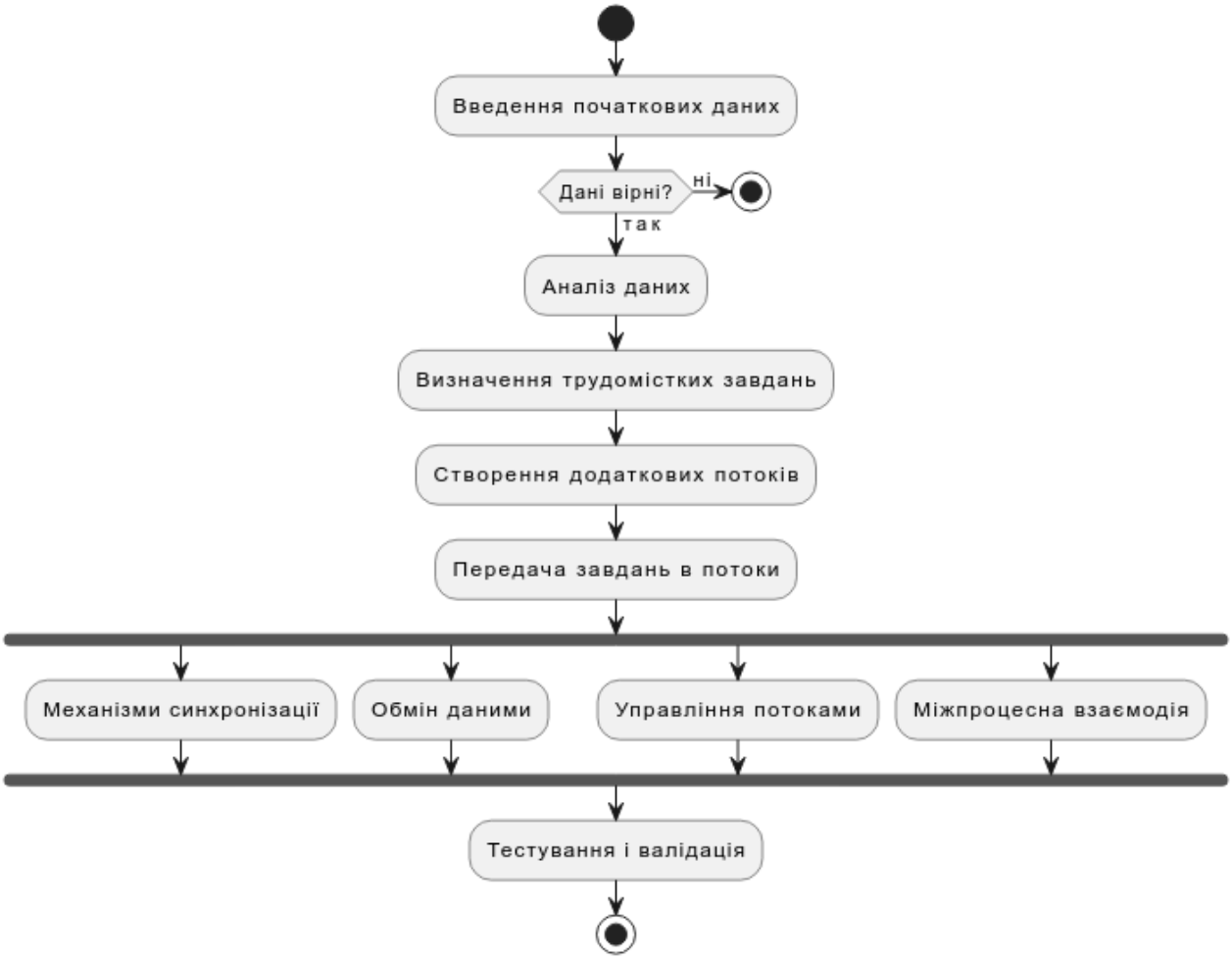


Рисунок 2.4 - UML діаграма багатопоточної обробки даних під час побудови зображення Мандельброта

Крок 7. Використання механізмів міжпроцесної взаємодії для координації роботи між різними процесами чи потоками.

Крок 8. Тестування та валідація тестів для перевірки правильності та ефективності роботи з багатопоточними процесами, виявлення та виправлення можливих проблем.

Наведений багатопоточний алгоритм базується на основних принципах розробки для Unix-подібних операційних систем, які передбачають виконання трудомістких операцій у фонових потоках та забороняють блокування основного потоку, що відповідає за інтерфейс користувача.

Для дослідження роботи процесів і тредів використовувалася операційна система Linux та персональний комп'ютер із процесором s1366 Intel Core i7-950

3.06-3.36GHz 4/8 8MB DDR3 800/1066 130W (4 ядра, 8 потоків). У результаті дослідження було отримано такі дані (табл. 2.3).

Таблиця 2.3

Час обробки даних, що отримані від фракталів залежно від кількості робочих потоків, що використовуються

Кількість потоків	Час отримання фракталу (мс)	Кількість потоків	Час отримання фракталу (мс)
1	1,43	7	0,28
2	0,78	8	0,27
3	0,54	9	0,26
4	0,42	10	0,25
5	0,35	11	0,24
6	0,31	12	0,24

Дані із табл. 2.3 були отримані за допомогою паралельних обчислень на основі потоків, що використовується в додатках для ОС Linux, базується на властивостях і методах класи AsyncTask, що параметризується. При такому способі всі тривалі за часом обчислення виконуються одним або декількома робочими потоками, що дозволяє виконувати обробку паралельно з кількома процесорними ядрами.

Тому, вся робота із синхронізації робочих потоків, виконання потоконебезпечних ділянок програми, а також візуалізація результатів обробки на екрані пристрою виконуються основним потоком, з якого починається робота при запуску програми.

З наданих у табл. 2.3 даних видно, як час обробки даних (отримання фракталу) залежить від кількості робочих потоків, що використовуються для виконання паралельних завдань. При цьому важливо відзначити таке.

Збільшення паралельних потоків знижує час виконання, тобто зі збільшенням кількості потоків від 1 до 12, час отримання фрактал значно

скорочується. Це є класичним прикладом прискорення виконання завдання за рахунок використання методу розпаралелювання. В даному випадку використання багатьох потоків дозволяє виконувати обчислення більш ефективно і швидко.

Зменшення часу нелінійно, тобто зменшення часу виконання не є лінійним із збільшенням кількості потоків. Наприклад, різниця в часі виконання між 1 і 2 потоками набагато більша, ніж різниця між 9 і 10 потоками. Це з накладними витратами управління потоками і змаганням ресурси процесора.

Закон Амдаля стверджує, що прискорення виконання завдання обмежується найповільнішою частиною цього завдання, навіть якщо інші частини розпаралельовані. В даному випадку, при вирішенні цього завдання прискорення обмежене складністю обчислень та продуктивністю процесора.

Також з табл. 2.3 видно, що при використанні 4 потоків час виконання досягло відносної плато, і подальше збільшення числа потоків призводить до більш незначного прискорення. Це свідчить про наявність обмежуючих чинників, як-от апаратні ресурси чи складність алгоритму.

Тому оптимальним числом потоків для цього завдання може бути близько 4-5, оскільки з додатковим збільшенням потоків прискорення стає менш суттєвим. Також на основі наданих даних з табл. 2.3 можна зробити висновок, що використання множини робочих потоків може значно прискорити виконання завдання, але рівень прискорення та оптимальна кількість потоків залежать від конкретної задачі, а також від апаратних ресурсів та інших факторів.

На основі даних часу побудови фракталу, що показані табл. 2.3 та методу найменших квадратів отримана математична модель багатопоточності тредів з рівнянням y = 1.26 * x^(-0.73).

Результати порівняльної характеристики отриманих даних (результатів досліджень побудови фракталу) та рівняння моделі багатопоточності тредів показані на рис. 2.5.

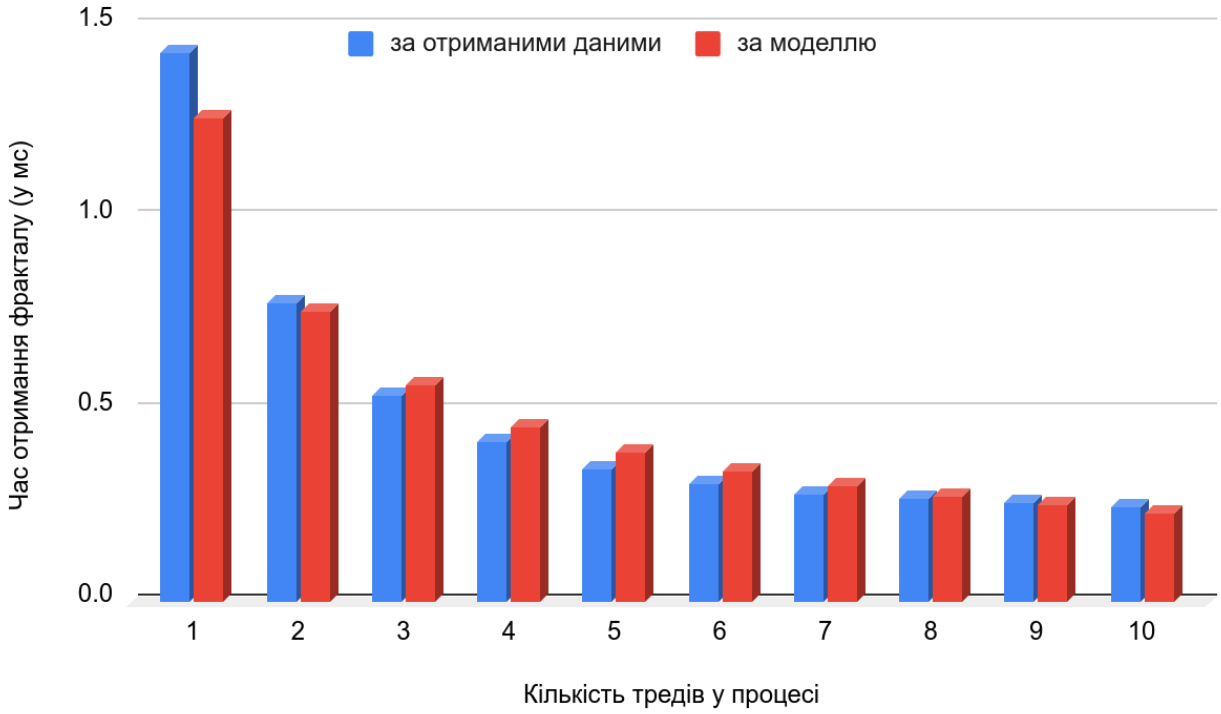


Рисунок 2.5 - Порівняльна характеристика отриманих даних (синій колір) та рівняння моделі багатопоточності тредів (червоний колір)

Таким чином, для перевірки ефективності методу розпаралелювання обробки даних за допомогою тредів створено програмну реалізацію для ОС Linux, яка вимагала тривалих обчислень за допомогою робочих потоків. Потім, в основному потоці, формувалися дані, що використовувались у подальших дослідженнях.

Проведені виміри часу роботи програми за різної кількості використовуваних паралельно працюючих потоків показали високу ефективність використання методу багатопоточних обчислень в ОС Linux. Так, на чотирьох-ядерному процесорі Intel Core i7-950 прискорення роботи програми склало: при 2-х робочих потоках – у 1.8 рази, при 3-х робочих потоках – у 2.6 рази, при 4-х робочих потоках – у 3.4 рази порівняно з часом виконання програми з використанням лише одного робочого потоку.

2.4. Висновки

В другому розділі проведено дослідження тредів та процесів в операційній системі Linux. Визначені особливості дослідження тредів та процесів в в операційній системі Linux.

Запропонований метод дослідження роботи процесів та тредів, де для перевірки ефективності методу розпаралелювання обробки даних було створена програмну реалізацію для ОС Linux, яка вимагала тривалих обчислень за допомогою робочих потоків. Потім, в основному потоці, формувалися дані, що використовувались у подальших дослідженнях. Проведені виміри часу роботи програми за різної кількості використовуваних паралельно працюючих потоків показали високу ефективність використання методу багатопоточних обчислень в ОС Linux.

РОЗДІЛ 3.
РЕАЛІЗАЦІЯ МЕТОДУ ДОСЛІДЖЕННЯ ТРЕДІВ ТА ПРОЦЕСІВ

3.1. Визначення вимог щодо розробки системи управління тредів та процесів

Під час розробки системи управління тредів та процесів в операційній системі Linux дуже важливо визначити вимоги до такої автоматизованної системи (рис. 3.1).

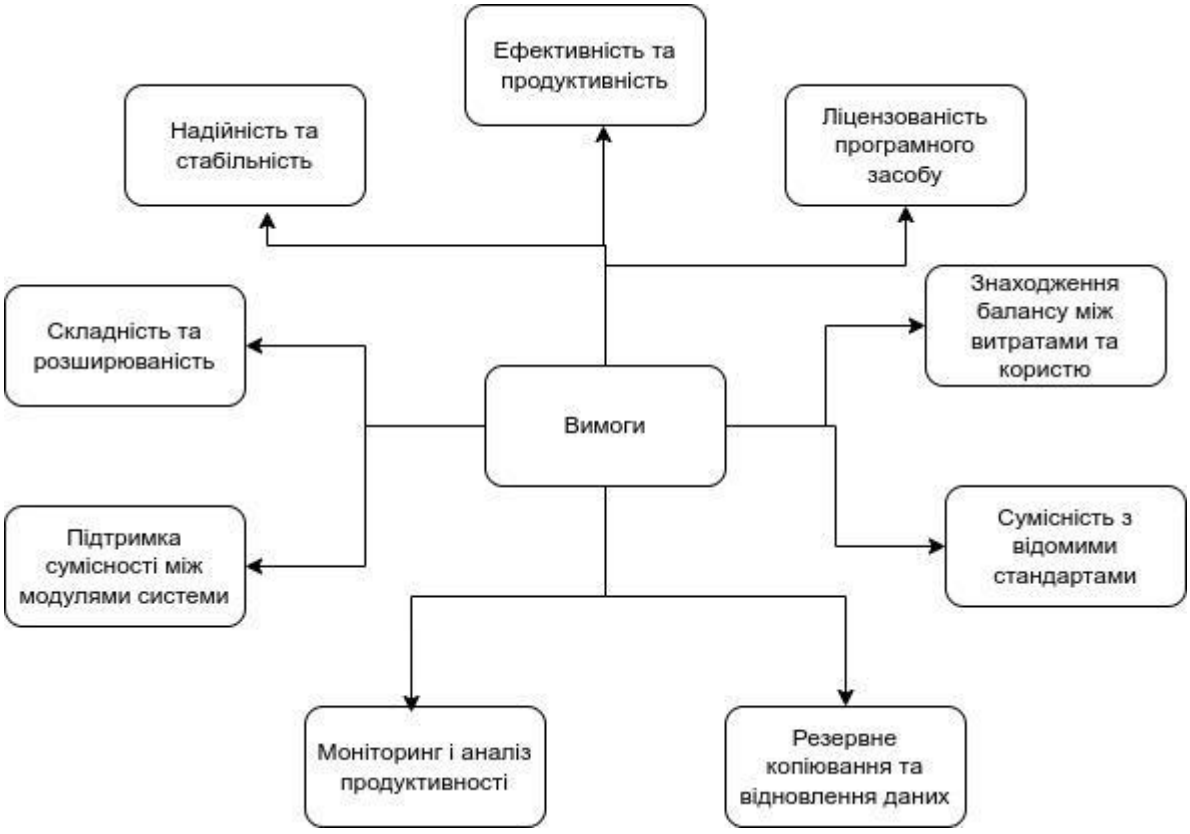


Рисунок 3.1 - Вимоги до розробки системи управління тредів та процесів в операційній системі Linux

Опишемо їх більш ретельно. На основі визначених вимог, автоматизована система повинна оптимізувати управління тредами та процесами, забезпечуючи швидку реакцію на запити і мінімізуючи зайві обчислювальні витрати.

Вимога до ефективності та продуктивності в розробці системи управління тредами та процесами в операційній системі Linux передбачає досягнення

максимальної продуктивності та оптимального використання ресурсів при виконанні операцій з тредами та процесами (табл. 3.1).

Таблиця 3.1

Вимоги до розробки системи управління тредами та процесами в операційній системі Linux

Вимоги	Опис вимог
Ефективність та продуктивність	Мінімізація накладних витрат на управління тредами і процесами. Забезпечення швидкодії та ефективності операцій з управлінням.
Надійність та стабільність	Забезпечення стійкості системи до помилок та відмов. Моніторинг та виявлення аномалій та аварій.
Складність та розширюваність	Зручний інтерфейс користувача для налаштування та моніторингу тредів і процесів. Можливість легкого розширення функціоналу системи.
Підтримка сумісності	Забезпечення сумісності з різними версіями операційної системи Linux. Підтримка різних архітектур та режимів роботи.
Безпека і автентифікація	Забезпечення контролю доступу до функцій управління тредами та процесами. Можливість аутентифікації користувачів та ролей.
Моніторинг і аналіз продуктивності	Збір та аналіз даних щодо використання ресурсів процесора, пам'яті та інших ресурсів системи. Генерація звітів і логів для аналізу продуктивності та виявлення проблем.
Резервне копіювання та відновлення	Можливість резервного копіювання конфігурацій та стану системи управління тредами і процесами. Здатність відновлювати систему після відмови.
Документація та підтримка	Доступна та зрозуміла документація з використання та налаштування системи. Можливість отримання технічної підтримки та оновлень.
Сумісність із стандартами	Відповідність стандартам і протоколам управління процесами та тредами.
Баланс між	Забезпечення оптимального співвідношення між

витратами та користю	витратами на розробку та підтримку системи та користю, яку вона приносить.
Спільнота розробників та користувачів	Залучення спільноти розробників і користувачів для спільної розробки, тестування та підтримки системи.
Ліцензування	Використання відкритих ліцензій, таких як GNU GPL, для забезпечення відкритості та доступу до вихідного коду.

При цьому, система повинна бути дизайнована так, щоб мінімізувати накладні витрати, пов'язані з управлінням тредами та процесами. Це включає в себе швидкість створення, завершення та перемикання тредів та процесів, а також мінімізацію використання CPU, пам'яті та інших ресурсів.

Також, система повинна забезпечувати швидку реакцію на запити користувачів або інших системних компонентів, що стосуються створення, завершення, призупинення або відновлення тредів та процесів. Затримки можуть призвести до негативного впливу на продуктивність та реакцію системи на зміни в навантаженні.

Система має ефективно використовувати оперативну та віртуальну пам'ять для забезпечення мінімального використання ресурсів. Це включає в себе ефективне виділення та звільнення пам'яті для процесів та тредів. Реалізація алгоритмів керування тредами та процесами повинна бути оптимізованою для виконання операцій з мінімальною обчислювальною та часовою складністю.

Автоматизована система повинна бути масштабованою, щоб забезпечити ефективну роботу навіть при великих навантаженнях. Вона повинна витримувати ріст кількості тредів та процесів без значного зниження продуктивності. При цьому, система повинна мати засоби моніторингу та аналізу продуктивності для виявлення та усунення можливих проблем та бутівлювання оптимізацій.

Важливо оптимізувати доступ до файлової системи для тредів та процесів, оскільки робота з файлами може бути дорогою за ресурсами операцією. Система

повинна ефективно підтримувати багатозадачність, де багато тредів та процесів одночасно конкурують за ресурси.

Ці вимоги до ефективності та продуктивності допомагають забезпечити, що система управління тредами та процесами в операційній системі Linux працює швидко та ефективно, використовуючи ресурси оптимальним чином і задовольняючи потреби користувачів та системних завдань. Важливо, щоб система була стійкою до помилок та відмов, і мала можливість виявлення та відновлення після аварій.

Вимога до надійності та стабільності в розробці системи управління тредами та процесами в операційній системі Linux передбачає створення системи, яка здатна працювати безперебійно та стійко в різних умовах, мінімізувати ризик виникнення помилок та відмов, а також надавати можливість виявлення та відновлення після аварій. Тому, система повинна бути стійкою до відмов та помилок, які можуть виникнути під час роботи. Це включає в себе обробку некоректних запитів, уникнення ситуацій блокування, та здатність продовжувати працювати при незначних проблемах.

Виявлення та відновлення помилок відбувається, коли система повинна мати засоби виявлення та відновлення помилок. Це може включати в себе механізми реєстрації та логування помилок, а також засоби автоматичного відновлення після аварій.

Забезпечення безпеки та надійності управління процесами та тредами для запобігання конфліктам, переповненням буфера, та іншим помилкам. Система повинна бути здатна відновлювати свою роботу після відмови, включаючи відновлення стану процесів та тредів та продовження роботи.

Засоби моніторингу та аналізу відмов допомагають виявляти та аналізувати ситуації, де стійкість системи може бути порушена. Система повинна проходити тестування та верифікацію, щоб впевнитися в її стійкості та надійності перед впровадженням в реальному середовищі.

Для цього, система повинна мати заходи захисту від зловживання та атак, які можуть призвести до відмови. Забезпечення захисту даних та

конфіденційності процесів і тредів, щоб запобігти несанкціонованому доступу та витоку даних. Система повинна бути відмовостійкою, що означає, що вона може працювати навіть у випадку відмови окремих компонентів чи ресурсів.

Ці вимоги до надійності та стабільності допомагають створити систему, яка може надійно виконувати свої функції навіть у вимогливих умовах та забезпечувати неперервну роботу системи управління тредами та процесами в операційній системі Linux.

Для виконання вимог складності та розширюваності потрібен зручний інтерфейс для користувачів для зручного налаштування і моніторингу. Також важлива можливість легко розширювати функціонал системи. Вимога до складності та розширюваності в розробці системи управління тредами та процесами в операційній системі Linux передбачає створення системи, яка буде легко розуміти, модифікувати та розширювати. Ця вимога допомагає забезпечити гнучкість та довгострокову життєздатність системи. Для зрозумілості та легкістю використання система повинна бути легко зрозумілою для розробників і адміністраторів. Документація повинна бути доступною та інформативною, а інтерфейс користувача повинен бути інтуїтивно зрозумілим.

Для модульності та роздільності обов'язків вважається, що система повинна бути розділеною на модулі, кожен з яких відповідає за конкретні функціональність. Це дозволить впроваджувати зміни в одному модулі без впливу на інші, що сприяє легкості розширення та обслуговування.

Можливість налаштування передбачає, що система повинна надавати можливість налаштовувати різні параметри та поведінку без зміни вихідного коду. Це допомагає адаптувати систему до різних вимог користувачів і сценаріїв використання. Система повинна мати засоби для додавання нового функціоналу без значних змін в існуючому коді. Це дозволяє впроваджувати нові можливості, коли з'являються нові вимоги.

Можливість розробляти та впроваджувати плагіни і розширення для системи, які можуть забезпечувати додатковий функціонал без змін в основному коді. Тому, використання системи завдатків (івентів) для реагування на події і

зміни в системі. Це дозволяє легко додавати нові обробники подій та реагувати на різні становища.

Можливість використовувати різні інтерфейси для взаємодії з системою, включаючи командний рядок, API, веб-інтерфейси та інші.

Підтримка зворотної сумісності із старими версіями і іншими додатками, які можуть використовувати систему.

Надані вимоги до складності та розширюваності допомагають забезпечити, що система управління тредами та процесами в операційній системі Linux буде зручною для розробників, легко адаптуватиметься до змінних вимог, та матиме потенціал для подальшого розширення та покращення.

Вимога щодо підтримки сумісності між модулями системи управління тредами та процесами в операційній системі Linux передбачає, що різні компоненти системи повинні співпрацювати між собою та з іншими додатками та модулями без проблем та конфліктів. Ця вимога важлива для забезпечення правильної роботи системи та інтеграції з іншими програмними продуктами.

Тому, система повинна дотримуватися стандартів та специфікацій, які визначають правила взаємодії між компонентами системи та іншими системами. Наприклад, POSIX-стандарт для операційних систем UNIX/Linux. Для цього, система повинна бути сумісною із різними версіями операційної системи Linux. Вона повинна правильно працювати на різних релізах ядра та бібліотек.

Розробники системи повинні забезпечити стабільність та сумісність інтерфейсів програмного забезпечення (APIs) та додаткових модулів. Зміни в API повинні бути ретельно розглянуті і документовані, а збереження сумісності між різними версіями системи - дотримане. Система повинна надавати інтерфейси для інтеграції з іншими додатками та модулями, такими як бібліотеки, плагіни, сервіси, інші операційні системи, а також мережеві протоколи.

Доступна та докладна документація щодо інтерфейсів, специфікацій та методів інтеграції допомагає розробникам зрозуміти, як користуватися системою та інтегрувати її з іншими компонентами. Перевірка сумісності системи з іншими

додатками та модулями під час розробки та перед випуском допомагає виявити та усунути проблеми сумісності. Система повинна бути розділена на модулі, які можна легко замінювати або розширювати без змін в інших частинах системи. Це полегшує інтеграцію нових функцій та модулів.

Під час інтеграції з іншими системами повинні дотримуватися високі стандарти безпеки, щоб запобігти потенційним загрозам та атакам.

Ці вимоги до сумісності допомагають створити систему управління тредами та процесами в операційній системі Linux, яка може ефективно співпрацювати з іншими компонентами та забезпечувати інтеграцію з різними додатками і системами без конфліктів та проблем.

Контроль доступу до функцій управління та можливість аутентифікації користувачів і ролей надає важливі можливості для забезпечення безпеки.

Вимога до безпеки і автентифікації користувачів у системі управління тредами та процесами в операційній системі Linux визначає необхідність забезпечити захист від несанкціонованого доступу, недопущення витоку конфіденційних даних, та забезпечення ідентифікації та автентифікації користувачів.

Для цього, система повинна забезпечувати процес автентифікації користувачів, щоб перевірити їхню ідентичність. Це включає в себе перевірку логінів і паролів, а також можливість використання більш сильних методів автентифікації, таких як біометричні дані або двофакторна аутентифікація. Також, система повинна надавати механізми для авторизації користувачів і управління їхніми правами доступу до ресурсів системи, включаючи можливість обмежити доступ до об'єктів на основі ролей і правил.

Дані, що оброблюються системою управління тредами та процесами, повинні бути захищені від несанкціонованого доступу і витоку. Використання шифрування та інших методів захисту даних є важливими складовими безпеки. Система повинна підтримувати можливість ведення журналу подій (audit log), який фіксує дії користувачів та системних подій. Це допомагає виявляти та відслідковувати несанкціонований доступ та аномальні події.

Розробники повинні вживати заходів для запобігання вразливостям, таким як переповнення буфера або ін'єкція коду, які можуть бути використані для атак. Система повинна мінімізувати атаковану поверхню, зменшуючи надлишкові сервіси та забезпечуючи закриті порти та служби.

Системні процеси та служби повинні мати механізми автентифікації та авторизації для запобігання несанкціонованому втручанню. Розробники системи повинні впроваджувати постійні покращення безпеки шляхом оновлення залежностей, виправлення вразливостей, та аудиту системи на вразливості. Система повинна забезпечувати контроль над доступом до ресурсів, таких як файлові системи, мережеві порти, додатки, інтерфейси тощо. При цьому, така система повинна бути захищеною від вірусів, троянів та інших шкідливих програм.

Забезпечення безпеки і автентифікації користувачів в системі управління тредами та процесами в операційній системі Linux є надзвичайно важливим для забезпечення захищеної і стійкої роботи системи і захисту конфіденційності даних та ресурсів.

Система повинна збирати та аналізувати дані щодо використання ресурсів для моніторингу продуктивності та виявлення проблем.

Вимога до моніторингу і аналізу продуктивності в системі управління тредами та процесами в операційній системі Linux передбачає можливість стеження за роботою системи, виявлення проблем продуктивності, аналізу та оптимізації її ресурсів та шляхів вдосконалення.

Система повинна надавати інструменти для моніторингу ресурсів, таких як центральний процесор (CPU), пам'ять, дисковий простір, мережевий трафік та інші, щоб збирати дані про використання ресурсів. Для збору та агрегації даних система повинна забезпечувати можливість збору та агрегації даних моніторингу, щоб отримувати загальний вигляд на продуктивність системи.

Система повинна аналізувати зібрані дані та надавати статистику та звіти щодо продуктивності, виявлення проблем та пропозиції щодо оптимізації. Для

цього, система повинна мати механізми для виявлення проблем продуктивності та надсилання повідомлень адміністраторам або операторам системи.

Підтримка візуалізації даних, така як графіки та діаграми, допомагає адміністраторам отримувати зрозумілу та інформативну інформацію про продуктивність системи. Можливість моніторингу роботи окремих додатків та сервісів, які працюють в системі, є важливою для виявлення проблем у власних додатках.

Здатність відслідковувати мережевий трафік і з'єднання для виявлення проблем мережі та аналізу використання мережевих ресурсів. Спостереження за використанням пам'яті та центрального процесора для виявлення можливих витоків пам'яті або завантаження процесора є також важливими.

Під час збереження історичних даних система повинна забезпечити можливість збереження історичних даних моніторингу для аналізу трендів і порівняння з минулими даними. Також, до цих функцій входять можливість моніторити та аналізувати продуктивність на різних рівнях системи, включаючи окремі процеси, треди, та весь сервер.

Ці вимоги до моніторингу та аналізу продуктивності допомагають забезпечити ефективне відстеження та вдосконалення продуктивності системи управління тредами та процесами в операційній системі Linux.

Можливість резервного копіювання конфігурацій та стану системи є важливою, а також можливість відновлення після відмови є також важливими. Вимога до резервного копіювання та відновлення даних в системі управління тредами та процесами в операційній системі Linux передбачає можливість створення резервних копій системних ресурсів, конфігураційних файлів, та інших важливих даних, а також їхнє відновлення в разі виникнення непередбачених проблем або втрати даних.

При цьому, система повинна надавати можливість створення регулярних резервних копій даних, включаючи конфігураційні файли, бази даних, системні налаштування та інші важливі ресурси. Також, система повинна дозволяти користувачам обирати, які дані вони хочуть резервувати, і як часто це повинно

відбуватися. Наприклад, користувач повинен мати можливість налаштувати резервне копіювання щоденно, щотижня або за іншим графіком.

Зберігання резервних копій в безпечному місці забезпечуються коли резервні копії повинні зберігатися в безпечному місці, щоб уникнути їхньої втрати або пошкодження в разі аварії.

Можливість відновлення даних відбувається, коли система повинна надавати можливість відновити дані з резервних копій у випадку втрати чи пошкодження основних даних. Тоді, з'являється можливість створення інкрементальних резервних копій, які зберігають лише змінені або нові дані, допомагає зекономити місце для зберігання та зменшити час копіювання.

Забезпечення шифрування резервних копій даних допомагає зберегти їх конфіденційність та запобігти несанкціонованому доступу мається, коли система повинна зберігати історію резервних копій, щоб користувачі могли відновлювати дані з різних моментів часу.

Моніторинг та повідомлення про стан резервного копіювання відбувається, коли механізми моніторингу, які спостерігають за процесом резервного копіювання та надсилають повідомлення в разі виникнення проблем, допомагають забезпечити надійність цього процесу.

Відновлення в аварійних ситуаціях з'являється, коли система повинна мати можливість відновлення даних в аварійних ситуаціях, таких як відмова жорсткого диска або вірусна атака.

Розглянуті вимоги до резервного копіювання та відновлення даних допомагають забезпечити захист важливих інформаційних ресурсів та можливість відновити їх в разі непередбачених проблем або втрати.

Для реалізаціх вимог необхідна доступна та зрозуміла документація щодо використання і налаштування системи, а також можливість отримання технічної підтримки. З цієї метою існують вимоги щодо документації та підтримки користувачів у системі управління тредами та процесами в операційній системі Linux передбачає створення докладної документації та надання ефективної підтримки для користувачів та адміністраторів системи. Це допомагає

користувачам краще розуміти та використовувати систему, а також отримувати допомогу у вирішенні можливих проблем. При цьому, система повинна мати детальну документацію, яка включає в себе наступні елементи.

Посібник користувача, де виконується опис встановлення, налаштування та основних функцій системи для користувачів.

Документація API, якщо система надає програмний інтерфейс (API), то цей інтерфейс повинен бути документований для розробників.

Посібник адміністратора, де описані інструкції щодо налаштування та управління системою для адміністраторів.

Довідка із використання, де є короткі інструкції або пояснення для швидкого доступу користувачів до основних операцій та команд.

Онлайн-ресурси для надання додаткових відомостей, таких як веб-сайт, форуми, чати підтримки, де користувачі можуть знайти інформацію, отримати відповіді на запитання та звернутися до спеціалістів для допомоги.

Таким чином, документація щодо вимог повинна регулярно оновлюватися, особливо при випуску нових версій системи або виправленні помилок. Надання навчальних матеріалів, таких як відеоуроки, навчальні курси та приклади, що допомагають користувачам швидше оволодіти системою.

Також, необхідні наявність служби технічної підтримки, яка може надавати відповіді на складні запитання та вирішувати проблеми, що виникають у користувачів, інформування користувачів про важливі оновлення, виправлення вразливостей та інші критичні питання безпеки, механізми для збору зворотнього зв'язку від користувачів щодо проблем, скарг та пропозицій щодо поліпшень системи, підтримка різних мов. Для цього, документація та підтримка повинні бути доступні на різних мовах, особливо якщо система використовується в міжнародному середовищі.

Організація тренінгів та консультацій для користувачів та адміністраторів системи дозволяє забезпечити доступу ще для людей з обмеженими можливостями, де враховуються потреби користувачів з різними видами обмежених можливостей у документації та інтерфейсах.

Ці вимоги до документації та підтримки користувачів допомагають забезпечити ефективне використання системи управління тредами та процесами в операційній системі Linux та задоволення потреб користувачів у надійній допомозі та інформації.

Важливими є вимоги щодо сумісності з відомими стандартами, де система повинна відповідати стандартам управління процесами та тредами. Вимога до сумісності з відомими стандартами в системі управління тредами та процесами в операційній системі Linux передбачає необхідність дотримання відповідних індустріальних та міжнародних стандартів для забезпечення сумісності з іншими системами, інтероперабельності та забезпечення стандартизованого інтерфейсу для користувачів та розробників.

При цьому, враховується POSIX-сумісність, де система повинна дотримуватися стандартів POSIX (Portable Operating System Interface), таких як POSIX.1, POSIX.2, POSIX.4, тощо. Це забезпечує сумісність з іншими UNIX-подібними операційними системами та дозволяє переносити програми між ними.

Стандарти мов програмування передбачають підтримку стандартів мов програмування, таких як C, C++, Python, Java, тощо, для розробки програм та додатків для системи, забезпечення сумісності з мережевими протоколами та стандартами, такими як TCP/IP, HTTP, FTP, SSH, SMTP, тощо, для забезпечення інтероперабельності у мережевому середовищі.

Використання відомих стандартів для інтерфейсу користувача, таких як X Window System (X11) для графічного інтерфейсу або стандарти оболонок командного рядка, якщо застосовується мають велике значення для розробки вимог, де важливими є такі стандарти безпеки як стандарти криптографії та захисту даних, для забезпечення безпеки системи та інформації та використання стандартів для забезпечення безпеки мережевого зв'язку, таких як TLS/SSL для шифрування даних.

Міжнародна локалізація та інтернаціоналізація передбачає підтримку до стандартів локалізації (локальні формати дати, часу, валюти тощо) та

інтернаціоналізації (підтримка мовних рядків) для використання системи в різних регіонах світу.

До цих стандартів обміну даними входить дотримання правил для обміну даними, таких як JSON, XML, CSV, для забезпечення сумісності з іншими додатками та системами, забезпечення сумісності між різними архітектурами та платформами Linux (x86, ARM, itd.), якщо це застосовано.

Сумісність з існуючими стандартами галузі необхідна для дотримання відомих стандартів у відповідних галузях, таких як стандарти веб-розробки, стандарти медичних інформаційних систем тощо. Сумісність з відомими стандартами допомагає забезпечує інтероперабельність, спрощує розробку додатків для системи та дозволяє користувачам і розробникам легше взаємодіяти з системою.

Важливо, щоб система була вартісно-ефективною, забезпечуючи оптимальний співвідношення між витратами на розробку та користю.

Вимога до знаходження балансу між витратами та користю в системі управління тредами та процесами в операційній системі Linux передбачає необхідність оптимізації використання ресурсів та забезпечення ефективної роботи системи без надмірного споживання ресурсів.

Система повинна ефективно використовувати ресурси, такі як процесорний час, пам'ять, мережевий трафік та інші, для забезпечення найкращої продуктивності при мінімальних витратах.

Моніторинг та аналіз витрат вимагає від системи можливість моніторити та аналізувати витрати ресурсів для виявлення можливих проблем та оптимізації.

Динамічне управління ресурсами надає можливість динамічно налаштовувати виділені ресурси для процесів та тредів на основі їхніх потреб і пріоритетів. Оптимізація завантаження системи зменшує час завантаження системи та споживання ресурсів при завантаженні. Використання кешу для збереження проміжних результатів та зменшення обчислювальних витрат.

Підтримка віртуалізації та контейнеризації забезпечує можливості використання віртуалізації та контейнеризації для оптимізації ресурсів у

віртуальних середовищах, надає високу доступність, враховує загальні вимоги щодо надійності та забезпечення мінімізації перерв у роботі системи.

Підтримка масштабованості надає можливість розширювати систему для обробки більшої кількості процесів та тредів без падінь продуктивності, зберігає стійку продуктивність, забезпечує сталу продуктивність системи навіть при зростанні обсягу роботи та навантаження.

Врахування вартості ресурсів дозволяє підрахувати вартість використання ресурсів (наприклад, електроенергії) та мінімізує ці витрати, знаходить балансу між витратами та користю, що допомагає оптимізувати роботу системи, зменшити витрати та забезпечити найкращу продуктивність та ефективність.

Вимога до наявності спільноти розробників та користувачів в системі управління тредами та процесами в операційній системі Linux передбачає створення та підтримку активної спільноти людей, які розробляють, тестують, вдосконалюють та використовують цю систему. Особливо важливі наявність онлайн-форумів, чатів та інших платформ для спілкування та обговорення питань щодо системи, що допомагає обміну досвідом, надає можливість користувачів та розробників звертатися один до одного для отримання допомоги, вирішення проблем та обміну кращими практиками.

Публічний доступ до вихідного коду системи сприяє співпраці, внесенню виправлень та вдосконалень від зовнішніх розробників.

Документація і навчання створюють та підтримують процеси документообігу для розробників і користувачів, навчальних матеріалів та навчальних курсів.

Спільнота розробників та користувачів відіграє важливу роль у підтримці та розвитку системи управління тредами та процесами в операційній системі Linux. Вона сприяє покращенню продуктивності, якості та безпеки цієї системи.

Вимога щодо ліцензованості програмного засобу в системі управління тредами та процесами в операційній системі Linux передбачає визначення правового статусу та умов використання програмного продукту, зокрема, чи це

вільне програмне забезпечення з відкритим вихідним кодом або комерційний продукт.

Тому, програмний продукт повинен мати визначену ліцензію, яка визначає права та обов'язки користувачів. У разі вільного програмного забезпечення з відкритим вихідним кодом, це може бути, наприклад, ліцензія GPL (GNU General Public License) або MIT License. Якщо програма має відкритий вихідний код, ліцензія повинна забезпечувати доступ користувачів до вихідного коду програми та їхню можливість модифікувати та розповсюджувати його.

Якщо програмний продукт є комерційним, то ліцензія повинна визначати умови для комерційного використання, включаючи вартість ліцензії та обмеження. Ліцензія повинна розглядати питання патентних прав та можливість використання програми без порушення патентних прав третіх осіб. Ліцензія повинна визначати авторські права на програмний продукт та умови їх використання.

Умови ліцензії повинні чітко визначати відповідальність розробників та гарантії (якщо такі є) щодо функціональності програмного продукту. Якщо передбачається надання актуалізацій та підтримки, ліцензія повинна визначити умови та тривалість цієї підтримки.

Ліцензія може містити обмеження, такі як обмеження кількості користувачів або пристроїв, на яких можна використовувати програмний продукт.

Вимога до ліцензуваності програмного засобу важлива для забезпечення законного використання системи та визначає правовий статус користувачів та розробників. Розробники повинні чітко визначити умови ліцензування та надавати користувачам можливість ознайомитися з цими умовами перед використанням програмного продукту.

Таким чином, розглянуті вимоги визначають ключові параметри і характеристики системи управління тредами та процесами в операційній системі Linux, яка відповідає потребам користувачів і організацій.

3.2. Структурно-функціональні особливості основних модулів програмного засобу

Основні модулі програмного засобу з управління тредами та процесами в операційній системі Linux мають ряд важливих властивостей та структурно-функціональних особливостей (рис. 3.2).

Опишемо ключові аспекти, які можуть бути характерними для цих модулів (табл. 3.2).

Модуль керування процесами і тредами в операційній системі Linux має важливу структурно-функціональну особливість, яка полягає в здатності створювати, контролювати та координувати процеси та треди в системі.

Функціонал структурної організації включає в себе структури даних та об'єкти для представлення процесів і тредів. Одній з основних функцій цього модуля є підтримка ієрархії процесів, де кожен процес може мати батьківський процес та дочірні процеси. Ця структура дозволяє групувати та керувати процесами логічно.

Функціонал створення процесів і тредів надає API для створення нових процесів та тредів, включаючи можливість клонування існуючих. Це дозволяє розробникам створювати нові завдання або задачі для виконання.

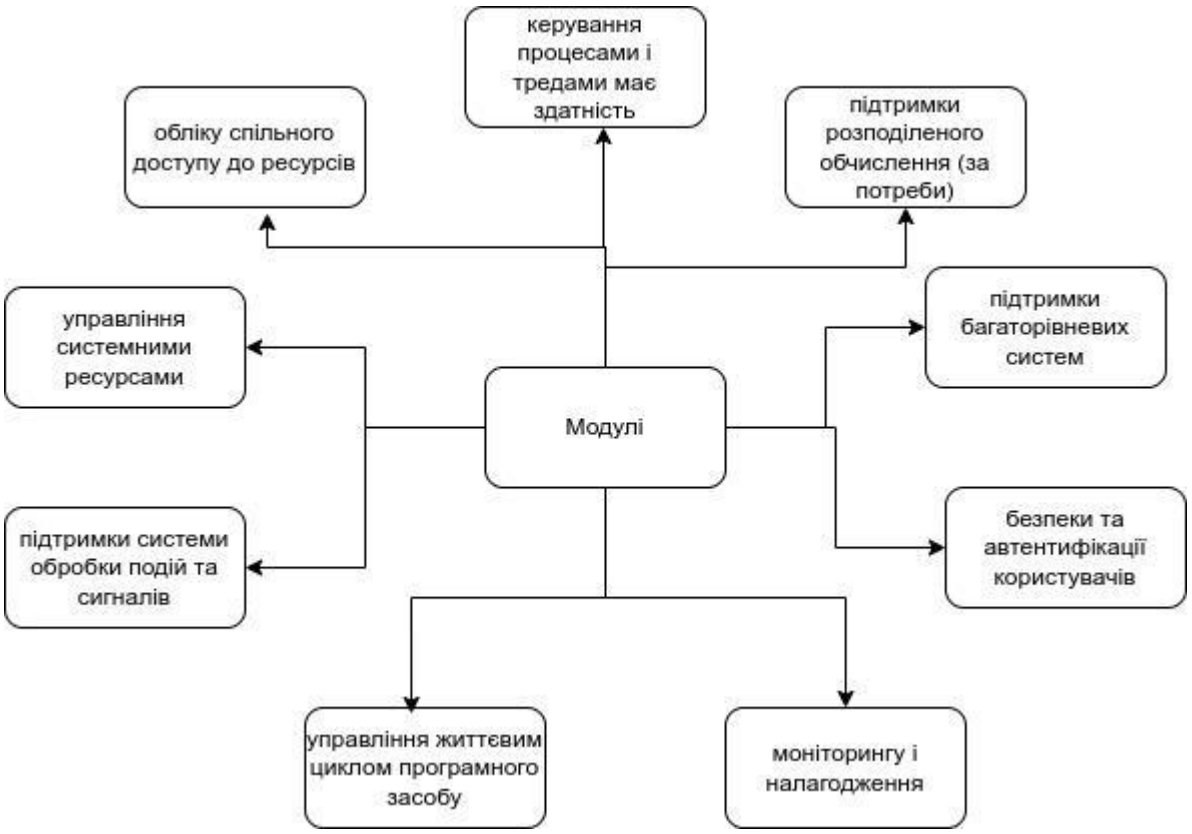


Рисунок 3.2 - Структурно-функціональні особливості модулів програмного засобу з управління тредами та процесами в операційній системі Linux

Модуль має систему планування для призначення процесам та тредам пріоритетів виконання та вирішення конфліктів. Планувальник визначає, який процес або тред виконується в даний момент.

Функціонал синхронізації та взаємодії надає механізми синхронізації, такі як семафори, блокування, м'ютекси і черги, для забезпечення взаємодії між процесами і тредами. Це дозволяє уникнути гонок та конфліктів при спільному доступі до ресурсів.

Модуль керує системними ресурсами, такими як пам'ять і процесорний час, і може накладати обмеження на їхнє використання для кожного процесу або треду. Система обробки сигналів дозволяє процесам та тредам реагувати на події в операційній системі, такі як завершення інших процесів або таймери.

Таблиця 3.2

Опис основних модулів програмного засобу для управління тредами та процесами в операційній системі Linux

Призначення (функціонал) модулів	Властивості та особливості
Керування процесами і тредами	- Створення, запуск, призупинення та завершення процесів і тредів. - Призначення пріоритетів для регулювання їхньої важливості.
Спільний доступ до ресурсів	- Механізми синхронізації та взаємодії (семафори, блокування, м'ютекси). - Розподілення пам'яті та обмін даними між процесами або тредами.
Системні ресурси	- Управління системними ресурсами (пам'ять, процесорний час, введення/виведення). - Моніторинг і статистика використання ресурсів.
Обробка подій та сигналів	- Система обробки подій та сигналів для реагування на події в ОС. - Реєстрація обробників сигналів.
Управління життєвим циклом	- Створення і завершення процесів і тредів за потребою програми. - Управління залежностями між ними.
Моніторинг і налагодження	- Засоби відладки, моніторингу та аналізу продуктивності. - Аналіз використання ресурсів.
Безпека та автентифікація	- Заходи для забезпечення безпеки процесів і тредів (обмеження доступу, автентифікація).
Підтримка багаторівневих систем	- Можливість створювати багаторівневі програми з різними рівнями тредів і процесів.
Документація та інтерфейси програмування	- Наявність документації та API для розробників для створення та управління процесами і тредами.
Можливості відлагодження	- Інструменти для відлагодження, профілювання та аналізу процесів і тредів.

Функціонал системи управління життєвим циклом дозволяє створювати, призупиняти, завершувати і перезапускати процеси та треди відповідно до їхніх потреб та залежностей. Моніторинг та аналіз продуктивності надає засоби для

моніторингу та аналізу продуктивності процесів та тредів, включаючи можливість збору статистики та інформації про використання ресурсів.

Модуль включає в себе заходи для забезпечення безпеки процесів і тредів, включаючи обмеження доступу до системних ресурсів і механізми автентифікації користувачів.

Ця структурно-функціональна особливість модуля керування процесами і тредами в операційній системі Linux дозволяє ефективно та надійно управляти обчислювальними задачами, забезпечувати ізоляцію та безпеку, а також керувати ресурсами для виконання різних завдань у системі.

Модуль обліку спільного доступу до ресурсів має механізми синхронізації та взаємодії між процесами і тредами, включаючи семафори, блокування, м'ютекси та інші. Також, до основних функцій цього модуля входить розподілення пам'яті і обмін даними між процесами або тредами.

Модуль обліку спільного доступу до ресурсів в операційній системі Linux відіграє важливу роль у забезпеченні ефективного та безконфліктного доступу до системних ресурсів, таких як пам'ять, файли, пристрої тощо.

Функціонал ve синхронізації надає механізми синхронізації, такі як семафори, блокування та м'ютекси, які дозволяють процесам і тредам координувати свою роботу та уникати гонок за ресурсами.

Модуль підтримує систему черг та буферів, яка дозволяє процесам та тредам взаємодіяти через обмін даними через черги, забезпечуючи безпечний та послідовний доступ до ресурсів.

Функціонал управління доступом до файлів та ресурсів дозволяє встановлювати правила доступу до файлів, директорій та інших ресурсів, щоб обмежити доступ та забезпечити конфіденційність та безпеку. Модуль надає можливість виявлення конфліктів та вжиття заходів для їх вирішення, таких як блокування ресурсів або повідомлення про помилки.

Функціонал моніторингу та журналювання доступу може вести журнал доступу до ресурсів, щоб відстежувати, які процеси та треди мають доступ до ресурсів, і аналізувати їхню активність. Модуль дозволяє багатьом процесам і

тредам одночасно спільно використовувати ресурси системи, забезпечуючи відокремленість та безпеку.

Функціонал повторного використання ресурсів сприяє ефективному використанню ресурсів шляхом їхнього повторного використання та перерозподілу між процесами та тредами. Також, є підтримка атомарних операцій, що дозволяють виконувати послідовні дії над ресурсами без можливості втрати даних або створення гонок.

Функціонал захисту від переповнення буфера допомагає уникати переповнення буфера шляхом встановлення обмежень на об'єм даних, які можуть бути записані чи прочитані з ресурсу. Модуль спрощує розробку конкурентних програм, де багато процесів і тредів одночасно звертаються до спільних ресурсів.

Ця структурно-функціональна особливість модуля обліку спільного доступу до ресурсів допомагає забезпечити безконфліктний та ефективний доступ до ресурсів в операційній системі Linux, що є ключовим для стабільної та надійної роботи системи та додатків.

Модуль з управління системними ресурсами має механізми управління системними ресурсами, такими як пам'ять, процесорний час, введення/виведення, мережа та інші, моніторинг і статистику використання ресурсів має наступний функціонал.

В загальному випадку, модуль з управління системними ресурсами в операційній системі Linux відіграє критичну роль у забезпеченні ефективного та справедливого розподілу обчислювальних, пам'ятевих та інших ресурсів між процесами та тредами.

Функціонал розподілення пам'яті керує пам'яттю, яка виділяється та звільняється процесам та тредам. Він включає в себе механізми для виділення стеків, кучі та інших областей пам'яті, а також може визначати правила обміну даними між процесами через пам'ять.

Модуль відстежує використання процесорного часу кожним процесом та тредом і призначає їм пріоритети для планування виконання. Він забезпечує справедливу та ефективну розподіл процесорного часу.

Управління введенням/виведенням включає в себе системи для керування введенням/виведенням, які дозволяють процесам та тредам взаємодіяти з пристроями вводу/виводу (наприклад, диски, мережеві інтерфейси).

Моніторинг і статистика в даному модулі надає можливість ведення статистики щодо використання ресурсів, включаючи витрати пам'яті, використання CPU і обсяг введення/виведення. Ця інформація може бути корисною для аналізу продуктивності та вирішення проблем.

Керування файлами та дисками дозволяє керувати доступом до файлів і дисків, забезпечуючи захист даних та встановлюючи правила для зчитування, запису та видалення файлів.

Обмін ресурсами між процесами дозволяє процесам та тредам обмінюватися ресурсами, такими як сокети для мережевого взаємодії або інтерпроцесовий обмін повідомленнями.

Модуль має можливість встановлювати ліміти на використання ресурсів для кожного процесу, щоб запобігти перевантаженню системи або зловживанню ресурсами.

Автоматичне відновлення ресурсів може відновлювати ресурси, які були використані і звільнені попередніми процесами, для подальшого використання. Механізми очікування та блокування надають засоби для процесів та тредів очікувати на наявність ресурсів або блокувати виконання, якщо ресурси недоступні.

Підтримка розподіленого обчислення може підтримувати розподілені обчислення, дозволяючи обмінювати ресурсами та даними між вузлами.

Ця структурно-функціональна особливість модуля з управління системними ресурсами в операційній системі Linux є важливою для забезпечення ефективного та справедливого використання ресурсів у системі, що сприяє оптимальній роботі процесів та тредів.

Модуль підтримки системи обробки подій та сигналів в операційній системі Linux грає ключову роль у забезпеченні взаємодії між процесами та тредами, а також у виявленні та реагуванні на події, такі як помилки, сигнали від користувачів або інші системні події.

Модуль включає в себе систему сигналів, яка дозволяє процесам та тредам надсилати та отримувати сигнали для сповіщення про події або помилки. Сигнали можуть бути використані для обробки винятків, переривань та інших асинхронних подій.

Реакція на сигнали надається за допомогою механізмів для реакції на сигнали, включаючи можливість встановлення обробників сигналів (signal handlers), які викликаються при отриманні певного сигналу. Це дозволяє програмістам забезпечувати власну обробку сигналів.

Система черг та повідомлень може включати систему черг та повідомлень, яка дозволяє процесам та тредам обмінюватися повідомленнями і подіями, що спрощує комунікацію між ними. Планування подій включає підсистему планування подій, яка дозволяє процесам запланувати обробку подій у майбутньому, вказуючи час виконання та функцію-обробник події. Модуль також може надавати можливість відстежувати події в системі, такі як створення, завершення або переривання процесів та тредів, і реагувати на них відповідно до встановлених правил.

Система логування та журналювання включає систему логування та журналювання подій, яка допомагає відстежувати та аналізувати події у системі для відладки, аудиту та аналізу продуктивності.

Обробка винятків і помилок полягає у визначені обробників винятків та обробки помилок, які можуть виникати під час виконання програми. Відслідковування системних подій дозволяє відслідковувати системні події, такі як введення/виведення даних, мережеву активність, стан пристроїв тощо.

Механізми блокування та очікування надають можливість блокувати виконання процесів та тредів до настання певної події або сигналу.

Ця структурно-функціональна особливість модуля підтримки системи обробки подій та сигналів в операційній системі Linux допомагає керувати взаємодією між різними частинами системи, реагувати на події та спрощувати розробку програм, які вимагають асинхронної поведінки.

Модуль управління життєвим циклом програмного засобу має можливість створювати та завершувати динамічно процеси або треди відповідно до потреби програми та виконувати управління залежностями між процесами або тредами. Він відіграє важливу роль у керуванні запуском, виконанням, зупинкою та видаленням програм та процесів.

Цей модуль дозволяє запускати нові програми та процеси у системі. Він може виконувати різні завдання перед запуском, такі як завантаження необхідних бібліотек, встановлення змінних середовища, та інше.

Функціонал модуль забезпечує можливість керувати процесами, включаючи можливість створювати, призупиняти, відновлювати та припиняти їх виконання. Модуль дозволяє відстежувати стан та ресурси, які використовуються кожним процесом, а також може надавати інформацію про їхню активність.

Модуль дозволяє створювати та керувати системними службами та демонами, які запускаються при завантаженні системи та працюють в фоновому режимі. Він також може включати системний планувальник завдань, який дозволяє планувати та автоматизувати виконання завдань у певні часові моменти або за певних умов.

Функціонал модуль визначає та керує життєвим циклом процесів, включаючи їхнє створення, запуск, виконання, зупинку і видалення. Модуль дозволяє видаляти ресурси, пов'язані з завершеними процесами, такі як пам'ять, файли та інші ресурси.

Завантаження ядра та ініціалізація системи включає в себе процес завантаження ядра операційної системи та ініціалізації системних служб і ресурсів. Модуль дозволяє керувати життєвим циклом додатків, забезпечуючи їхню ініціалізацію, виконання та завершення. Відладка та діагностика проблем

надає інструменти для відладки та діагностики проблем, пов'язаних з життєвим циклом програм та процесів.

Ця структурно-функціональна особливість модуля управління життєвим циклом програмного засобу в операційній системі Linux є важливою для забезпечення ефективного та стабільного функціонування програм та процесів у системі, а також для автоматизації процесів управління ними.

Модуль моніторингу і налагодження має засоби для відладки та моніторингу роботи процесів і тредів, включаючи можливість виводу логів, профілювання та аналізу продуктивності. Також, цей модуль має засоби аналізу використання ресурсів. Він відіграє важливу роль у відстеженні та аналізі роботи системи, виявленні помилок, відладці програмного коду та забезпеченні ефективної роботи системи.

Збір інформації про систему забезпечує можливість збору різноманітної інформації про стан системи, такої як використання CPU, обсяг пам'яті, обсяг введення/виведення даних, мережевий трафік тощо.

Функціонал аналіу продуктивності надає інструменти для аналізу продуктивності системи та окремих програм. Це допомагає виявити ресурсозатратність та можливі місця для оптимізації. Відстеження роботи програм може відстежувати роботу окремих програм та процесів, включаючи їхній життєвий цикл, використання ресурсів та інші параметри.

Модуль дозволяє виконувати аналіз виконання програмного коду, включаючи відстеження викликів функцій, вимірювання часу виконання, аналіз стеку викликів та інше. Відладка програм надає інструменти для відладки програмного коду, включаючи можливість встановлення точок зупинки (breakpoints), відстеження значень змінних, виведення повідомлень та стеку викликів.

Функціонал виявлення та реакція на помилки може автоматично виявляти помилки в роботі програм та сповіщати про них. Також він може надавати можливість реакції на помилки, наприклад, аварійне завершення програми. Ще відстеження подій та сигналів дозволяє відстежувати системні події та сигнали,

такі як завантаження системи, зупинка процесів, помилки файлової системи тощо.

Модуль може забезпечувати можливість журналювання подій та інформації про роботу системи для подальшого аналізу та аудиту. Віддалене управління та діагностика може підтримувати віддалене управління та діагностику системи, що дозволяє адміністраторам віддалено аналізувати та налагоджувати роботу системи.

Ця структурно-функціональна особливість модуля моніторингу і налагодження є важливою для забезпечення надійності, продуктивності та ефективності операційної системи Linux, а також для виявлення та вирішення проблем, які можуть виникнути під час її роботи.

Модуль безпеки та автентифікації користувачів в операційній системі Linux відіграє важливу роль у забезпеченні захисту системи від несанкціонованого доступу та збереженні конфіденційності та цілісності даних.

Модуль надає можливість аутентифікації користувачів, перевіряючи їхню ідентичність через паролі, ключі або інші методи. Це дозволяє визначити, чи має користувач доступ до системи. Управління правами доступу дозволяє визначати права доступу для користувачів і груп до різних ресурсів, таких як файли, директорії, пристрої та сервіси. Це забезпечує контроль над тим, що користувачі можуть та не можуть робити у системі.

Функціонал авторизації та управління сеансами керує сеансами користувачів і може включати механізми авторизації, які визначають права доступу до різних дій та ресурсів під час активного сеансу. Шифрування та безпека даних забезпечує можливість шифрування даних, що передаються та зберігаються на системі, для захисту від несанкціонованого доступу та перехоплення.

Аудит та журналювання подій може включати систему аудиту та журналювання подій для відстеження та реєстрації дій користувачів і системних подій. Це допомагає виявити та реагувати на потенційні загрози безпеці.

Функціонал захисту від атак має заходи безпеки для захисту системи від різних видів атак, таких як переповнення буфера, введення через веб-додатки, вторгнення та інші. Управління сертифікатами та ключами дозволяє керувати сертифікатами та ключами для забезпечення безпеки з'єднань та обміну даними. Існує також біометрична автентифікація, що підтримує методи біометричної автентифікації, такі як відбиток пальця або розпізнавання обличчя.

Система внутрішньої безпеки забезпечується захистом від внутрішніх загроз, включаючи привілейовані акаунти та можливість виявлення підозрілих дій користувачів.

Ця структурно-функціональна особливість модуля безпеки та автентифікації користувачів є критично важливою для забезпечення цілісності та безпеки операційної системи Linux та збереження даних інформації, що в ній зберігається.

Модуль підтримки багаторівневих систем має можливість створення багаторівневих програм, де кілька рівнів тредів та процесів співіснують та співпрацюють. До його складу також належать інструменти для відлагодження, профілювання та аналізу процесів і тредів. Він відіграє ключову роль у забезпеченні безпеки даних та управлінні доступом до них в обмеженому оточенні, де існують різні рівні конфіденційності та доступу.

Даний модуль використовує модель безпеки, що визначає правила та політику для управління доступом користувачів до різних рівнів даних. Зазвичай це базується на концепції обов'язкового контролю доступу (Mandatory Access Control - MAC).

Цей модуль дозволяє призначати лейбели та класифікації до об'єктів, таких як файли, процеси та ресурси. Це допомагає ідентифікувати рівень конфіденційності та доступу для кожного об'єкта.

Функціонал керування доступом визначає, як користувачі з різними лейбелами можуть отримувати доступ до об'єктів. Це включає правила доступу, такі як "прочитати", "записати", "виконати" тощо. Модуль забезпечує захист від

несанкціонованого виходу даних на рівень, що нижчий за їхній рівень конфіденційності. Це допомагає запобігти витоку даних.

Модуль може включати систему аудиту та журналювання подій, щоб реєструвати спроби доступу та інші події для аналізу безпеки. Модуль дозволяє ізолювати процеси на різних рівнях безпеки, щоб запобігти несанкціонованому обміну інформацією між ними.

Функціонал правил перевірки безпеки використовує певні правила перевірки безпеки для визначення, чи можуть об'єкти та суб'єкти отримувати доступ один до одного. Цей модуль дозволяє визначати, які права доступу можуть бути успадковані від батьківських об'єктів до дочірніх.

Ця структурно-функціональна особливість дозволяє операційній системі Linux працювати в умовах з підвищеним рівнем безпеки та забезпечувати ізоляцію даних та ресурсів між різними рівнями

Таким чином, загальна структура та функціональність модулів може змінюватися в залежності від конкретної реалізації та завдань програмного засобу які будуть висуватись в майбутньому. Однак ці основні властивості і функції допомагають управляти процесами і тредами в операційній системі Linux та розвивати різноманітні програми і системи на її основі у теперішній час.

3.3. Проектування, розробка та впровадження основних модулів програмного засобу

Проектування основних модулів програмного засобу для управління тредами та процесами в операційній системі Linux вимагає ретельного підходу і врахування важливих аспектів (табл. 3.3). Для цього потрібно визначити функціональності основних модулів. Для вимог, що позначені у підрозділі 3.1 функціональності основних модулів для виконання проектування програмного засобу будуть наступні.

Таблиця 3.3

Основні етапи проектування, розробки та впровадження основних модулів програмного засобу

Етапи розробки	Основні дії та процеси
Проектування	1. Визначення функціональності модулів. 2. Архітектурне проектування і визначення інтерфейсів. 3. Вибір технологій та інструментів.
Розробка	1. Написання коду для кожного модуля згідно специфікацій. 2. Модульне тестування кожного модуля. 3. Інтеграція модулів та системне тестування. 4. Відлагодження та оптимізація коду.
Впровадження	1. Розгортання програмного засобу на цільових системах. 2. Системне тестування та перевірка роботи. 3. Підтримка та вдосконалення. 4. Моніторинг та оптимізація продукту.

Спочатку необхідно чітке визначення функціональних вимог до кожного модуля, де розглядаються всі можливі операції, які повинен виконувати програмний засіб для управління тредами та процесами.

Під час архітектурного проектування необхідно розробити архітектурний план програмного засобу, визначивши структуру модулів та їх взаємозв'язок та порядок обміну даними між модулями та внутрішньою архітектурою програмного засобу.

Визначення інтерфейсів передбачає створення чітких інтерфейсів для кожного модуля, включаючи списки доступних функцій, параметри та повернені значення та структури даних, які будуть передаватися між модулями.

Вибір технологій та інструментів має на меті врахування, які технології та інструменти найкраще підходять для реалізації кожного модуля та розгляд можливість використання бібліотек або фреймворків, які спростять розробку.

Розробка деталей модулів передбачає написання детальних специфікацій для кожного модуля, включаючи опис функцій, алгоритмів та структур даних, розробку коду для кожного модуля, керуючись розробленими специфікаціями.

Тестування модулів має на меті виконання модульного тестування для кожного модуля окремо, переконавшись, що кожна функція працює коректно та використання тестових даних, щоб впевнитися в правильності роботи.

Для інтеграції та системного тестування необхідно інтегрувати модулі разом та виконати системне тестування всього програмного засобу, перевірте, чи взаємодіють модулі правильно і чи виконуються всі функції.

Відлагодження та оптимізація необхідно для знаходження та виправлення помилок (багів) у коді кожного модуля, виконання оптимізації коду для підвищення продуктивності та ефективності.

При цьому, готується технічна документація для кожного модуля, включаючи опис інтерфейсів та приклади використання.

Для впровадження та підтримки розгортається програмний засіб на цільових серверах або системах, потім надається підтримка користувачам та адміністраторам системи, вирішуйте їхні запити та виявлені помилки.

Моніторинг та оптимізація програмного засобу виконується в реальному часі, де вживаються заходи для оптимізації його роботи. Надалі підтримується програмний засіб та розробляються поновлення для покращення функціональності та безпеки.

Цей підхід дозволяє ретельно розробити та впровадити основні модулі для управління тредами та процесами в операційній системі Linux і забезпечити їхню ефективну та стабільну роботу.

Етап проектування модулів програмного засобу для управління тредами та процесами в операційній системі Linux є критично важливим для успішної розробки системи (рис. 3.3). Цей етап передбачає ретельне визначення функціональності модулів, архітектурне проектування та визначення інтерфейсів, а також вибір технологій та інструментів.

Рисунок 3.3 - Основні складові етапу проектування

На цьому етапі визначається, які саме функції та задачі мають виконувати модулі системи. Розробники повинні чітко зрозуміти, які операції повинні бути доступні для управління тредами та процесами, включаючи створення, завершення, призупинення, відновлення тощо. Необхідно також визначити необхідні алгоритми та обробники подій.

Архітектурне проектування і визначення інтерфейсів передбачає наступні дії. На цьому етапі розробляється загальна архітектура програмного засобу, включаючи розподіл функцій між модулями, структуру та взаємозв'язки між ними. Кожен модуль повинен мати чітко визначені інтерфейси, які вказують, як взаємодіяти з іншими модулями та системою в цілому.

Вибір технологій та інструментів передбачає пошук технологій та інструментів, які найкраще підходять для реалізації функціональності модулів. Враховуються такі аспекти, як мова програмування, бібліотеки, фреймворки, бази даних та інші залежності. Також обираються інструменти для версійного керування, тестування, логування та налагодження.

Після завершення цих етапів, команда розробників має чітке уявлення про те, як будуть працювати модулі програмного засобу та як вони будуть

взаємодіяти між собою та з системою Linux. Ця важлива підготовча робота допоможе уникнути потенційних проблем під час реалізації та тестування системи.

Етап розробки модулів програмного засобу для управління тредами та процесами в операційній системі Linux включає наступні основні компоненти (рис. 3.4).

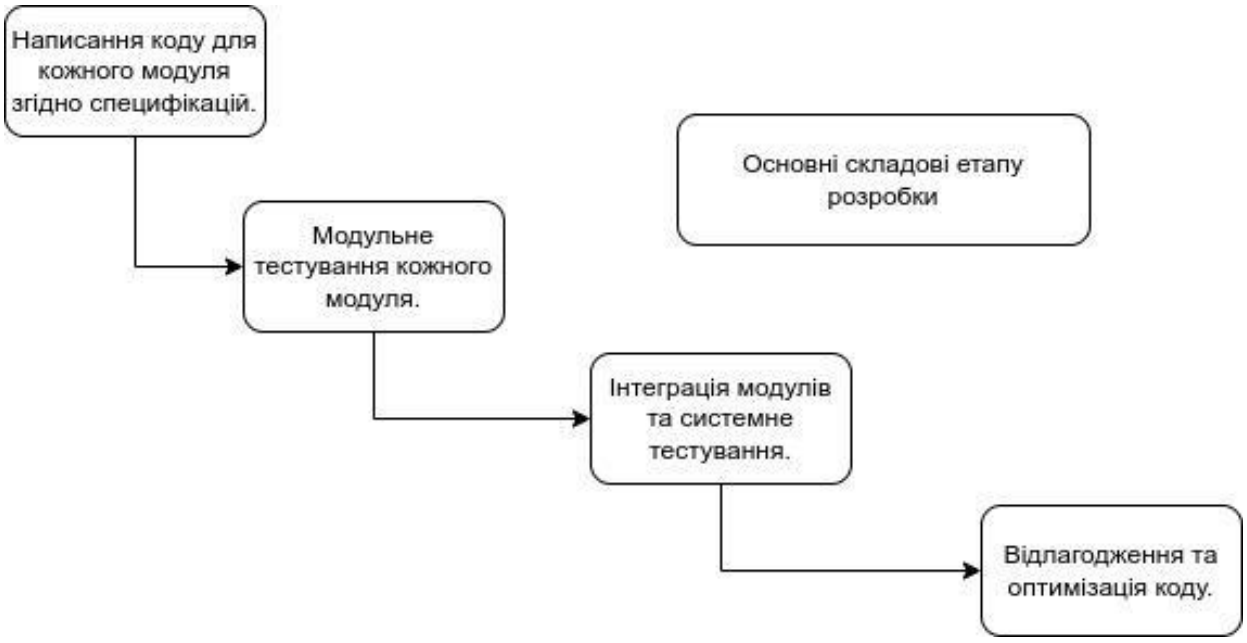


Рисунок 3.4 - Основні складові етапу розробки

На етапі написання коду для кожного модуля згідно специфікацій передбачає, що розробники пишуть код для кожного окремого модуля програмного засобу. Код повинен бути написаний відповідно до специфікацій та вимог, визначених на етапі проектування. Кожен модуль має бути самодостатнім та дотримуватися кращих практик програмування.

Після написання коду для кожного модуля важливо виконати його модульне тестування. Модульне тестування передбачає перевірку окремих функцій та методів, щоб переконатися, що вони працюють правильно та відповідають специфікаціям.

Тестові випадки створюються для кожного модуля, а потім виконуються для перевірки коректності роботи коду. Виявлені помилки виправляються, і тестування повторюється до досягнення очікуваних результатів.

Ці два компоненти є ключовими у розробці, оскільки вони допомагають переконатися, що код модулів відповідає вимогам, а також виявляють та виправляють помилки на ранніх стадіях розробки. Модульне тестування допомагає забезпечити надійність та стабільність кожного модуля перед їх інтеграцією в систему.

Етап розробки модулів програмного засобу для управління тредами та процесами в операційній системі Linux включає наступні основні компоненти.

На етапі інтеграція модулів та системне тестування розроблені модулі інтегруються в єдину систему. Це означає, що різні компоненти програмного засобу об'єднуються, і їх взаємодія перевіряється для впевненості, що вони співпрацюють правильно.

Системне тестування проводиться для перевірки роботи всієї системи як єдиної сутності. Тестові випадки повинні включати різні сценарії взаємодії з тредами, процесами та іншими компонентами операційної системи.

Відлагодження та оптимізація коду передбачає наступне.

Відлагодження (дебагінг) - це процес виявлення, аналізу та виправлення помилок в програмному коді. Розробники використовують різні інструменти для відлагодження, такі як інтерактивні середовища розробки, логування та аналізатори помилок.

Оптимізація - це процес покращення продуктивності та ефективності програмного засобу. Після виявлення та виправлення помилок розробники можуть працювати над оптимізацією алгоритмів та структур даних для досягнення кращої продуктивності.

Ці два компоненти важливі для забезпечення якості та ефективності програмного засобу. Після успішної інтеграції модулів та системного тестування, відлагодження та оптимізація дозволяють покращити роботу

програми, виявити та виправити залишкові помилки, а також досягти кращої продуктивності в процесі виконання.

Етап впровадження модулів програмного засобу для управління тредами та процесами в операційній системі Linux включає наступні основні компоненти (рис. 3.5).

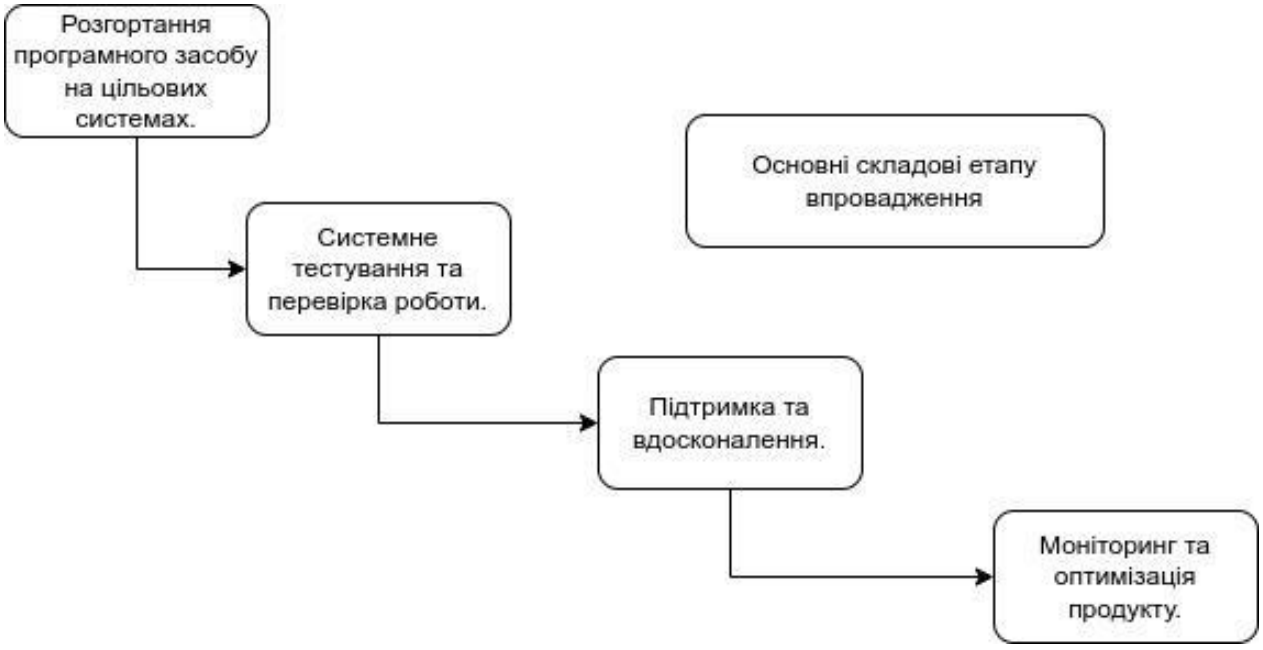


Рисунок 3.5 - Основні складові етапу впровадження

На цьому етапі розроблену систему розгортають на цільових серверах або комп'ютерах, де вона буде використовуватися. Розгортання може включати установку програмних пакетів, налаштування конфігураційних файлів та інші операції, необхідні для правильної роботи системи.

Після розгортання системи проводиться системне тестування для перевірки її коректності та стабільності в реальних умовах. Тестування повинно включати різні сценарії використання, включаючи навантаження на систему та взаємодію з іншими програмами та службами.

Під час тестування виявлені помилки та проблеми повинні бути документовані та виправлені. Перевірка на відповідність вимогам і специфікаціям також є важливою частиною цього етапу.

Після успішного впровадження та системного тестування програмного засобу можна ввести його в експлуатацію для реального використання. У цьому етапі важливо враховувати безпеку та моніторинг роботи програми для

забезпечення надійності та продуктивності вирішення завдань з управління тредами та процесами в операційній системі Linux.

Етап впровадження модулів програмного засобу для управління тредами та процесами в операційній системі Linux включає наступні основні компоненти (рис. 3.5).

Після впровадження важливо забезпечувати підтримку та обслуговування програмного засобу. Це включає в себе реагування на звернення користувачів, вирішення проблем, пов'язаних з роботою системи, а також підтримку інтеграції з іншими програмами та операційними системами.

Постійне вдосконалення програмного засобу також є важливим елементом підтримки. Це може включати в себе випуск оновлень з виправленнями помилок, вдосконаленням функціональності та відповідь на нові вимоги користувачів.

Таким чином, після впровадження важливо забезпечувати моніторинг роботи програмного засобу. Це включає в себе стеження за продуктивністю, надійністю та безпекою системи. Моніторинг допомагає вчасно виявляти проблеми та реагувати на них.

Оптимізація продукту полягає в пошуку можливостей для покращення продуктивності та ефективності програмного засобу. Це може включати в себе оптимізацію алгоритмів, структур даних та ресурсів. Тому, після впровадження продукту, підтримка та постійне вдосконалення є необхідними для забезпечення його надійності та відповідності потребам користувачів. Моніторинг і оптимізація дозволяють забезпечити ефективну роботу системи та реагувати на зміни в умовах використання.

3.4. Висновки

В третьому розділі були виконані роботи з реалізацію методу досліджень, де розроблені вимоги щодо розробки системи управління тредів та процесів. Визначені структурно-функціональні особливості основних модулів програмного засобу. Виконано проектування, розробка та вповадження основних модулів програмного засобу.

РОЗДІЛ 4.
ЕКОНОМІЧНИЙ РОЗДІЛ

Виконання науково-дослідної роботи завжди передбачає отримання певних результатів і вимагає відповідних витрат. Результати виконаної роботи завжди дають нам нові знання, які в подальшому можуть бути використані для удосконалення та/або розробки (побудови) нових, більш продуктивних зразків техніки, процесів та програмного забезпечення.

Дослідження на тему «Розробка методів і програмних засобів дослідження тредів та процесів в операційній системі Linux» може бути віднесено до фундаментальних і пошукових наукових досліджень і спрямоване на вирішення наукових проблем, пов’язаних з практичним застосуванням. Основою таких досліджень є науковий ефект, який виражається в отриманні наукових результатів, які збільшують обсяг знань про природу, техніку та суспільство, які розвивають теоретичну базу в тому чи іншому науковому напрямку, що дозволяє виявити нові закономірності, які можуть використовуватися на практиці.

Для цього випадку виконаємо такі етапи робіт:

1) здійснимо проведення наукового аудиту досліджень, тобто встановлення їх наукового рівня та значимості;

2) проведемо планування витрат на проведення наукових досліджень;

3) здійснимо розрахунок рівня важливості наукового дослідження та перспективності, визначимо ефективність наукових досліджень.

4.1 Оцінювання наукового ефекту

Основними ознаками наукового ефекту науково-дослідної роботи є новизна роботи, рівень її теоретичного опрацювання, перспективність, рівень розповсюдження результатів, можливість реалізації. Науковий ефект НДР на тему «Розробка методів і програмних засобів дослідження тредів та процесів в операційній системі Linux» можна охарактеризувати двома показниками: ступенем наукової новизни та рівнем теоретичного опрацювання.

Заплановані результати дослідження полягають у досягненні таких показників як: формалізація методів підвищення ефективності серверів у корпоративній мережі, визначення продуктивності сервірсів у корпоративній мережі - не менш 300 запитів за секунду, визначення розміру обробки запитах на серверах - не більш ніж 3000 мс, визначення кількості відмов в обробці запитів на сервері - не більш ніж 100 відмов за секунду, визначення кількості коллізій на мережевому інтерфейсі серверу - не більш ніж 2 за годину, визначення кількості віртуальних машин на одному серверу - не більш ніж 10.

Аналіз ресурсів і продуктивності виконується за допомогою програм Top і Htop, які роблять відображення використання CPU і пам'яті процесами та потоками в реальному часі. Програма Vmstat необхідна для моніторингу використання віртуальної пам'яті та інших системних ресурсів. Програма Iostat необхідна для відстеження використання дискового простору та операцій з введенням/виведенням. Аналіз конкурентності та синхронізації виконується за допомогою Pthreads, де досліджується відстеження діяльності потоків та блокувань у програмі. Програма MutexChecker здійснює виявлення помилок синхронізації за допомогою мутексів. Аналіз мережевої активності виконується за допомогою наступних програм. Wireshark виконує аналіз пакетів мережевого трафіку для виявлення проблем мережі та взаємодії програм. Програма Netstat здійснює відстеження мережевих з'єднань та портів. Для дослідження аналізу безпеки та аудиту виконуються Lynis і OpenVAS для аналізу безпеки системи та виявлення потенційних вразливостей. AIDE і Tripwire необхідні для виявлення змін в системних файлових структурах та контроль цілісності файлів. Моніторинг ресурсів та подій здійснюється за допомогою Syslog і rsyslog. Вони здійснюють збір і аналіз системних журналів подій. Інші програми, такі як Nagios і Zabbix виконують моніторинг стану системи та сервісів в реальному часі. Аналіз процесів та тредів здійснюється за допомогою Ps і top. Вони виводять інформацію про активні процеси та потоки. Програма lsof здійснює виявлення відкритих файлових дескрипторів процесами.

Значення показників ступеня новизни і рівня теоретичного опрацювання науково-дослідної роботи в балах наведені в табл. 4.1 та 4.2.

Таблиця 4.1

Показники ступеня новизни науково-дослідної роботи виставлені експертами

Ступінь новизни	Характеристика ступеня новизни	Значення ступеня новизни, бали		
		Експерти (ПІБ, посада)		
		1	2	3
Принципово нова	Робота якісно нова за постановкою задачі і ґрунтується на застосуванні оригінальних методів дослідження. Результати дослідження відкривають новий напрям в даній галузі науки і техніки. Отримані принципово нові факти, закономірності; розроблена нова теорія. Створено принципово новий пристрій, спосіб, метод	0	0	0
Нова	Отримана нова інформація, яка суттєво зменшує невизначеність наявних значень (по-новому або вперше пояснені відомі факти, закономірності, впроваджені нові поняття, розкрита структура змісту). Проведено суттєве вдосконалення, доповнення і уточнення раніше досягнутих результатів	59	56	53
Відносно нова	Робота має елементи новизни в постановці задачі і методах дослідження. Результати дослідження систематизують і узагальнюють наявну інформацію, визначають шляхи подальших досліджень; вперше знайдено зв’язок (або знайдено новий зв’язок) між явищами. В принципі відомі положення розповсюджені на велику кількість об’єктів, в результаті чого знайдено ефективне рішення. Розроблені більш прості способи для досягнення відомих результатів. Проведена часткова раціональна модифікація (з ознаками новизни)	0	0	0

Традиційна	Робота виконана за традиційною методикою. Результати дослідження мають інформаційний характер. Підтверджені або поставлені під сумнів відомі факти та твердження, які потребують перевірки. Знайдено новий варіант рішення, який не дає суттєвих переваг в порівнянні з існуючим	0	0	0
Не нова	Отримано результат, який раніше зафіксований в інформаційному полі, та не був відомий авторам	0	0	0
Середнє значення балів експертів		56,0		

Згідно отриманого середнього значення балів експертів ступінь новизни характеризується як нова, тобто отримана нова інформація, яка суттєво зменшує невизначеність наявних знань (по-новому або вперше пояснені відомі факти, закономірності, впроваджені нові поняття, розкрита структура змісту) та проведено суттєве вдосконалення, доповнення і уточнення раніше досягнутих результатів.

Таблиця 4.2

Показники рівня теоретичного опрацювання науково-дослідної роботи виставлені експертами

Характеристика рівня теоретичного опрацювання	Значення показника рівня теоретичного опрацювання, бали		
	Експерт (ПІБ, посада)		
	1	2	3
Відкриття закону, розробка теорії	0	0	0
Глибоке опрацювання проблеми: багатоаспектний аналіз зв’язків, взаємозалежності між фактами з наявністю пояснень, наукової систематизації з побудовою евристичної моделі або комплексного прогнозу	62	64	63
Розробка способу (алгоритму, програми), пристрою, отримання нової речовини	0	0	0
Елементарний аналіз зв’язків між фактами та наявною гіпотезою, класифікація, практичні рекомендації для окремого випадку тощо	0	0	0
Опис окремих елементарних фактів, викладення досвіду, результатів спостережень, вимірювань тощо	0	0	0
Середнє значення балів експертів	63,0		

Згідно отриманого середнього значення балів експертів рівень теоретичного опрацювання науково-дослідної роботи характеризується як глибоке опрацювання проблеми: багатоаспектний аналіз зв'язків, взаємозалежності між фактами з наявністю пояснень, наукової систематизації з побудовою евристичної моделі або комплексного прогнозу.

Показник, який характеризує рівень наукового ефекту, визначаємо за формулою **[Козловський, Лесько, Кавецький]:**

$$E_{нау} = 0,6 \cdot k_{нов} + 0,4 \cdot k_{теор}, \quad (4.1)$$

де $k_{нов}, k_{теор}$ - показники ступеня новизни та рівня теоретичного опрацювання науково-дослідної роботи, $k_{нов} = 56,0, k_{теор} = 63,0$ балів;

0,6 та 0,4 – питома вага (значимість) показників ступеня новизни та рівня теоретичного опрацювання науково-дослідної роботи.

$$E_{нау} = 0,6 \cdot k_{нов} + 0,4 \cdot k_{теор} = 0,6 \cdot 56,0 + 0,4 \cdot 63,00 = 58,80 \text{ балів.}$$

Визначення характеристики показника $E_{нау}$ проводиться на основі висновків експертів виходячи з граничних значень, які наведені в табл. 4.3.

Таблиця 4.3

Граничні значення показника наукового ефекту

Досягнутий рівень показника	Кількість балів
Високий	70…100
Середній	50…69
Достатній	15…49
Низький (помилкові дослідження)	1…14

Відповідно до визначеного рівня наукового ефекту проведеної науково-дослідної роботи на тему «Розробка методів і програмних засобів дослідження тредів та процесів в операційній системі Linux», даний рівень становить 58,80 балів і відповідає статусу - середній рівень. Тобто у даному випадку можна вести мову про потенційну фактичну ефективність науково-дослідної роботи.

4.2 Розрахунок витрат на здійснення науково-дослідної роботи

Витрати, пов’язані з проведенням науково-дослідної роботи на тему «Розробка методів і програмних засобів дослідження тредів та процесів в операційній системі Linux», під час планування, обліку і калькулювання собівартості науково-дослідної роботи групуємо за відповідними статтями.

4.2.1 Витрати на оплату праці

До статті «Витрати на оплату праці» належать витрати на виплату основної та додаткової заробітної плати керівникам відділів, лабораторій, секторів і груп, науковим, інженерно-технічним працівникам, конструкторам, технологам, креслярам, копіювальникам, лаборантам, робітникам, студентам, аспірантам та іншим працівникам, безпосередньо зайнятим виконанням конкретної теми, обчисленої за посадовими окладами, відрядними розцінками, тарифними ставками згідно з чинними в організаціях системами оплати праці.

Основна заробітна плата дослідників

Витрати на основну заробітну плату дослідників ($З_о$) розраховуємо у відповідності до посадових окладів працівників, за формулою **[Козловський, Лесько, Кавецький]**:

$$З_о = \sum_{i=1}^{k} \frac{M_{ni} \cdot t_i}{T_p}, \qquad (4.2)$$

де k – кількість посад дослідників залучених до процесу досліджень;

M_{ni} – місячний посадовий оклад конкретного дослідника, грн;

t_i – число днів роботи конкретного дослідника, дн.;

T_p – середнє число робочих днів в місяці, T_p=21 дні.

$$З_о = 16560{,}00 \cdot 21 / 21 = 16560{,}00 \ грн.$$

Проведені розрахунки зведемо до таблиці.

Таблиця 4.4

Витрати на заробітну плату дослідників

Найменування посади	Місячний посадовий оклад, грн	Оплата за робочий день, грн	Число днів роботи	Витрати на заробітну плату, грн

Керівник проекту	16560,00	788,57	21	16560,00
Інженер-розробник програмних засобів	15900,00	757,14	21	15900,00
Всього				32460,00

Основна заробітна плата робітників

Витрати на основну заробітну плату робітників ($З_р$) за відповідними найменуваннями робіт НДР на тему «Розробка методів і програмних засобів дослідження тредів та процесів в операційній системі Linux» розраховуємо за формулою:

$$З_р = \sum_{i=1}^{n} C_i \cdot t_i , \qquad (4.3)$$

де C_i – погодинна тарифна ставка робітника відповідного розряду, за виконану відповідну роботу, грн/год;

t_i – час роботи робітника при виконанні визначеної роботи, год.

Погодинну тарифну ставку робітника відповідного розряду C_i можна визначити за формулою:

$$C_i = \frac{M_M \cdot K_i \cdot K_c}{T_p \cdot t_{зм}} , \qquad (4.4)$$

де M_M – розмір прожиткового мінімуму працездатної особи, або мінімальної місячної заробітної плати (в залежності від діючого законодавства), приймемо $Мм$=6700,00 грн;

K_i – коефіцієнт міжкваліфікаційного співвідношення для встановлення тарифної ставки робітнику відповідного розряду (табл. Б.2, додаток Б) **[Козловський, Лесько, Кавецький]**;

K_c – мінімальний коефіцієнт співвідношень місячних тарифних ставок робітників першого розряду з нормальними умовами праці виробничих об’єднань і підприємств до законодавчо встановленого розміру мінімальної заробітної плати.

T_p – середнє число робочих днів в місяці, приблизно T_p = 21 дн;

$t_{зм}$ – тривалість зміни, год.

C_1 = 6700,00 · 1,10 · 1,35 / (21 · 8) = 59,22 *грн.*
$З_{р1}$ = 59,22 · 6,45 = 381,99 *грн.*

Таблиця 4.5

Величина витрат на основну заробітну плату робітників

Найменування робіт	Тривалість роботи, год	Розряд роботи	Тарифний коефіцієнт	Погодинна тарифна ставка, грн	Величина оплати на робітника грн
Установка електронно-обчислювального обладнання	6,45	2	1,10	59,22	381,99
Підготовка робочого місця розробника програмного забезпечення	4,50	2	1,10	59,22	266,50
Всього					648,49

Додаткова заробітна плата дослідників та робітників

Додаткову заробітну плату розраховуємо як 10 … 12% від суми основної заробітної плати дослідників та робітників за формулою:

$$З_{дод} = (З_о + З_р) \cdot \frac{Н_{дод}}{100\%}, \quad (4.5)$$

де $Н_{дод}$ – норма нарахування додаткової заробітної плати. Приймемо 10%.

$З_{дод}$ = (32460,00 + 648,49) · 10 / 100% = 3310,85 грн.

4.2.2 Відрахування на соціальні заходи

Нарахування на заробітну плату дослідників та робітників розраховуємо як 22% від суми основної та додаткової заробітної плати дослідників і робітників за формулою:

$$З_н = (З_о + З_р + З_{дод}) \cdot \frac{Н_{зп}}{100\%} \quad (4.6)$$

де $Н_{зп}$ – норма нарахування на заробітну плату. Приймаємо 22%.

Зн = (32460,00 + 648,49 + 3310,85) · 22 / 100% = 8012,26 грн.

4.2.3 Сировина та матеріали

До статті «Сировина та матеріали» належать витрати на сировину, основні та допоміжні матеріали, інструменти, пристрої та інші засоби і предмети праці, які придбані у сторонніх підприємств, установ і організацій та витрачені на проведення досліджень за темою «Розробка методів і програмних засобів дослідження тредів та процесів в операційній системі Linux».

Витрати на матеріали на даному етапі проведення досліджень в основному пов'язані з використанням моделей елементів та моделювання роботи і досліджень за допомогою комп'ютерної техніки та створення експериментальних математичних моделей або програмного забезпечення, тому дані витрати формуються на основі витратних матеріалів характерних для офісних робіт.

Витрати на матеріали (M), у вартісному вираженні розраховуються окремо по кожному виду матеріалів за формулою:

$$M = \sum_{j=1}^{n} H_j \cdot Ц_j \cdot K_j - \sum_{j=1}^{n} B_j \cdot Ц_{вj}, \qquad (4.7)$$

де H_j – норма витрат матеріалу j-го найменування, кг;

n – кількість видів матеріалів;

$Ц_j$ – вартість матеріалу j-го найменування, грн/кг;

K_j – коефіцієнт транспортних витрат, ($K_j = 1,1 \ldots 1,15$);

B_j – маса відходів j-го найменування, кг;

$Ц_{вj}$ – вартість відходів j-го найменування, грн/кг.

$M_1 = 3 \cdot 92,00 \cdot 1,04 - 0 \cdot 0 = 287,04$ *грн.*

Проведені розрахунки зведемо до таблиці.

Таблиця 4.6

Витрати на матеріали

Найменування матеріалу, марка, тип, сорт	Ціна за 1 кг, грн	Норма витрат, кг	Величина відходів, кг	Ціна відходів, грн/кг	Вартість витраченого матеріалу, грн

Тека для паперів CALIPSO BOX	92,00	3	0	0	287,04
Папір для записів Calipso Parers Light A5	112,00	4	0	0	465,92
Офісний папір Calipso Plus A4-500-80	212,00	3	0	0	661,44
Органайзер офісний Calipso Office	156,00	3	0	0	486,72
Картридж для принтера Canon LBP6500	1129,00	1	0	0	1174,16
Канцелярське приладдя (набір офісного працівника)	193,00	3	0	0	602,16
Диск оптичний NewLine CD-RW	27,00	4	0	0	112,32
USB Flesh-пам'ять Kingston 16 GB	169,00	1	0	0	175,76
Всього					3965,52

4.2.4 Розрахунок витрат на комплектуючі

Витрати на комплектуючі *($К_в$)*, які використовують при проведенні НДР на тему «Розробка методів і програмних засобів дослідження тредів та процесів в операційній системі Linux» відсутні.

4.2.5 Спецустаткування для наукових (експериментальних) робіт

До статті «Спецустаткування для наукових (експериментальних) робіт» належать витрати на виготовлення та придбання спецустаткування необхідного

для проведення досліджень, також витрати на їх проектування, виготовлення, транспортування, монтаж та встановлення.

Балансову вартість спецустаткування розраховуємо за формулою:

$$B_{спец} = \sum_{i=1}^{k} Ц_i \cdot C_{пр.i} \cdot K_i \ , \qquad (4.8)$$

де $Ц_i$ – ціна придбання одиниці спецустаткування даного виду, марки, грн;

$C_{пр.i}$ –кількість одиниць устаткування відповідного найменування, які придбані для проведення досліджень, шт.;

K_i – коефіцієнт, що враховує доставку, монтаж, налагодження устаткування тощо, ($K_i = 1,10…1,12$);

k – кількість найменувань устаткування.

$B_{спец} = 52899,00 \cdot 1 \cdot 1,04 = 55014,96$ *грн.*

Отримані результати зведемо до таблиці:

Таблиця 4.7

Витрати на придбання спецустаткування по кожному виду

Найменування устаткування	Кількість, шт	Ціна за одиницю, грн	Вартість, грн
Персональний комп'ютер із процесором s1366 Intel Core i7-950 3.06-3.36GHz 4/8 8MB DDR3 800/1066 130W (4 ядра, 8 потоків)	1	52899,00	55014,96
Всього			55014,96

4.2.6 Програмне забезпечення для наукових (експериментальних) робіт

До статті «Програмне забезпечення для наукових (експериментальних) робіт» належать витрати на розробку та придбання спеціальних програмних засобів і програмного забезпечення, (програм, алгоритмів, баз даних) необхідних для проведення досліджень, також витрати на їх проектування, формування та встановлення.

Балансову вартість програмного забезпечення розраховуємо за формулою:

$$B_{прг} = \sum_{i=1}^{k} Ц_{iпрг} \cdot C_{прг.i} \cdot K_i \ , \qquad (4.9)$$

де $Ц_{iпрг}$ – ціна придбання одиниці програмного засобу даного виду, грн;

$C_{прг.i}$ – кількість одиниць програмного забезпечення відповідного найменування, які придбані для проведення досліджень, шт.;

K_i – коефіцієнт, що враховує інсталяцію, налагодження програмного засобу тощо, (K_i = *1,10…1,12);*

k – кількість найменувань програмних засобів.

$B_{прг} = 852{,}00 \cdot 1 \cdot 1{,}01 = 860{,}52$ *грн.*

Отримані результати зведемо до таблиці:

Таблиця 4.8

Витрати на придбання програмних засобів по кожному виду

Найменування програмного засобу	Кількість, шт	Ціна за одиницю, грн	Вартість, грн
Програмний продукт аналізу ресурсів і продуктивності Top і Htop	1	852,00	860,52
Програмний продукт Vmstat	1	520,00	525,20
Програмний продукт Iostat	1	420,00	424,20
Програмний продукт аналізу мережевої активності Wireshark	1	350,00	353,50
Програмний продукт дослідження аналізу безпеки та аудиту Lynis і OpenVAS	1	650,00	656,50
Програмний продукт моніторингу ресурсів та подій Syslog і rsyslog	1	212,00	214,12
Інше програмне забезпечення	1	300,00	303,00
Всього			3337,04

4.2.7 Амортизація обладнання, програмних засобів та приміщень

В спрощеному вигляді амортизаційні відрахування по кожному виду обладнання, приміщень та програмному забезпеченню тощо, розраховуємо з використанням прямолінійного методу амортизації за формулою:

$$A_{обл} = \frac{Ц_б}{T_в} \cdot \frac{t_{вик}}{12}, \quad (4.10)$$

де $Ц_б$ – балансова вартість обладнання, програмних засобів, приміщень тощо, які використовувались для проведення досліджень, грн;

$t_{вик}$ – термін використання обладнання, програмних засобів, приміщень під час досліджень, місяців;

$T_в$ – строк корисного використання обладнання, програмних засобів, приміщень тощо, років.

$A_{обл} = (8550{,}00 \cdot 1) / (5 \cdot 12) = 142{,}50$ *грн.*

Проведені розрахунки зведемо до таблиці.

Таблиця 4.9

Амортизаційні відрахування по кожному виду обладнання

Найменування обладнання	Балансова вартість, грн	Строк корисного використання, років	Термін використання обладнання, місяців	Амортизаційні відрахування, грн
Оргтехніка	8550,00	5	1	142,50
Пакет прикланого програмного забезпечення розробки	8520,00	3	1	236,67
Персональний комп'ютер інженера-дослідника програмного забезпечення	28999,00	3	1	805,53
Персональний комп'ютер	52100,00	3	1	1447,22

системи обчислення даних				
Приміщення дослідної лабораторії	416500,00	30	1	1156,94
Пристрій виводу інформації	6850,00	5	1	114,17
Робоче місце розробника програмного забезпечення	8210,00	7	1	97,74
Всього				4000,77

4.2.8 Паливо та енергія для науково-виробничих цілей

Витрати на силову електроенергію (B_e) розраховуємо за формулою:

$$B_e = \sum_{i=1}^{n} \frac{W_{yi} \cdot t_i \cdot Ц_e \cdot K_{вni}}{\eta_i}, \quad (4.11)$$

де W_{yi} – встановлена потужність обладнання на визначеному етапі розробки, кВт;

t_i – тривалість роботи обладнання на етапі дослідження, год;

$Ц_e$ – вартість 1 кВт-години електроенергії, грн; (вартість електроенергії визначається за даними енергопостачальної компанії), приймемо $Ц_e = 7,50$ грн;

$K_{вni}$ – коефіцієнт, що враховує використання потужності, $K_{вni} < 1$;

η_i – коефіцієнт корисної дії обладнання, $\eta_i < 1$.

$B_e = 0,45 \cdot 120,0 \cdot 7,50 \cdot 0,95 / 0,97 = 405,00$ *грн.*

Проведені розрахунки зведемо до таблиці.

Таблиця 4.10

Витрати на електроенергію

Найменування обладнання	Встановлена потужність, кВт	Тривалість роботи, год	Сума, грн
Персональний комп'ютер із процесором s1366 Intel	0,45	120,0	405,00

Core i7-950 3.06-3.36GHz 4/8 8MB DDR3 800/1066 130W (4 ядра, 8 потоків)			
Оргтехніка	0,26	15,0	29,25
Персональний комп'ютер інженера-дослідника програмного забезпечення	0,40	160,0	480,00
Персональний комп'ютер системи обчислення даних	0,25	160,0	300,00
Пристрій виводу інформації	0,23	6,0	10,35
Робоче місце розробника програмного забезпечення	0,10	160,0	120,00
Всього			1344,60

4.2.9 Службові відрядження

До статті «Службові відрядження» дослідної роботи на тему «Розробка методів і програмних засобів дослідження тредів та процесів в операційній системі Linux» належать витрати на відрядження штатних працівників, працівників організацій, які працюють за договорами цивільно-правового характеру, аспірантів, зайнятих розробленням досліджень, відрядження, пов’язані з проведенням випробувань машин та приладів, а також витрати на відрядження на наукові з’їзди, конференції, наради, пов’язані з виконанням конкретних досліджень.

Витрати за статтею «Службові відрядження» розраховуємо як 20…25% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{св} = (З_о + З_р) \cdot \frac{Н_{св}}{100\%}, \qquad (4.12)$$

де $Н_{св}$ – норма нарахування за статтею «Службові відрядження», приймемо $Н_{св} = 20\%$.

$B_{св} = (32460,00 + 648,49) \cdot 20 / 100\% = 6621,70$ *грн.*

4.2.10 Витрати на роботи, які виконують сторонні підприємства, установи і організації

Витрати за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації» розраховуємо як 30…45% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{сп} = (З_о + З_р) \cdot \frac{Н_{сп}}{100\%}, \qquad (4.13)$$

де $Н_{сп}$ – норма нарахування за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації», приймемо $Н_{сп}$= 30%.

$B_{сп} = (32460,00 + 648,49) \cdot 30 / 100\% = 9932,55$ *грн.*

4.2.11 Інші витрати

До статті «Інші витрати» належать витрати, які не знайшли відображення у зазначених статтях витрат і можуть бути віднесені безпосередньо на собівартість досліджень за прямими ознаками.

Витрати за статтею «Інші витрати» розраховуємо як 50…100% від суми основної заробітної плати дослідників та робітників за формулою:

$$I_в = (З_о + З_р) \cdot \frac{Н_{ів}}{100\%}, \qquad (4.14)$$

де $Н_{ів}$ – норма нарахування за статтею «Інші витрати», приймемо $Н_{ів}$ = 50%.

$I_в = (32460,00 + 648,49) \cdot 50 / 100\% = 16554,25$ *грн.*

4.2.12 Накладні (загальновиробничі) витрати

До статті «Накладні (загальновиробничі) витрати» належать: витрати, пов’язані з управлінням організацією; витрати на винахідництво та раціоналізацію; витрати на підготовку (перепідготовку) та навчання кадрів; витрати, пов’язані з набором робочої сили; витрати на оплату послуг банків; витрати, пов’язані з освоєнням виробництва продукції; витрати на науково-технічну інформацію та рекламу та ін.

Витрати за статтею «Накладні (загальновиробничі) витрати» розраховуємо як 100…150% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{нзв} = (З_о + З_р) \cdot \frac{Н_{нзв}}{100\%}\ , \qquad (4.15)$$

де $Н_{нзв}$ – норма нарахування за статтею «Накладні (загальновиробничі) витрати», приймемо $Н_{нзв}$ = 100%.

$В_{нзв}$ = (32460,00 + 648,49) · 100 / 100% = 33108,49 грн.

Витрати на проведення науково-дослідної роботи на тему «Розробка методів і програмних засобів дослідження тредів та процесів в операційній системі Linux» розраховуємо як суму всіх попередніх статей витрат за формулою:

$$B_{заг} = З_о + З_р + З_{дод} + З_н + М + К_в + В_{спец} + В_{прг} + А_{обл} + В_е + В_{св} + В_{сп} + І_в + В_{нзв}. \qquad (4.16)$$

$В_{заг}$= 32460,00 +648,49 +3310,85 +8012,26 +3965,52 +0,00 +55014,96 + 3337,04 + 4000,77 +1344,60 +6621,70 +9932,55 +16554,25 +33108,49 = 178311,47 грн.

Загальні витрати *ЗВ* на завершення науково-дослідної (науково-технічної) роботи та оформлення її результатів розраховується за формулою:

$$ЗВ = \frac{B_{заг}}{\eta}, \qquad (4.17)$$

де η - коефіцієнт, який характеризує етап (стадію) виконання науково-дослідної роботи, приймемо η=0,9.

ЗВ = 178311,47 / 0,9 = 198123,86 грн.

4.3 Оцінювання важливості та наукової значимості науково-дослідної роботи

Оцінювання та доведення ефективності виконання науково-дослідної роботи фундаментального чи пошукового характеру є достатньо складним процесом і часто базується на експертних оцінках, тому має вірогідний характер.

Для обґрунтування доцільності виконання науково-дослідної роботи на тему «Розробка методів і програмних засобів дослідження тредів та процесів в операційній системі Linux» використовується спеціальний комплексний показник, що враховує важливість, результативність роботи, можливість впровадження її результатів у виробництво, величину витрат на роботу.

Комплексний показник K_P рівня науково-дослідної роботи може бути розрахований за формулою:

$$K_P = \frac{I^n \cdot T_C \cdot R}{B \cdot t}, \qquad (4.18)$$

де I – коефіцієнт важливості роботи. Приймемо $I = 4$;

n – коефіцієнт використання результатів роботи; $n = 0$, коли результати роботи не будуть використовуватись; $n = 1$, коли результати роботи будуть використовуватись частково; $n = 2$, коли результати роботи будуть використовуватись в дослідно-конструкторських розробках; $n = 3$, коли результати можуть використовуватись навіть без проведення дослідно-конструкторських розробок. Приймемо $n=2$;

T_C – коефіцієнт складності роботи. Приймемо $T_C = 3$;

R – коефіцієнт результативності роботи; якщо результати роботи плануються вище відомих, то $R = 4$; якщо результати роботи відповідають відомому рівню, то $R = 3$; якщо нижче відомих результатів, то $R = 1$. Приймемо $R = 3$;

B – вартість науково-дослідної роботи, тис. грн. Приймемо B =198123,86 грн;

t – час проведення дослідження. Приймемо t = 0,08 років, (1 міс.).

Визначення показників *I, n, T_C, R, B, t* здійснюється експертним шляхом або на основі нормативів [Козловський, Лесько, Кавецький].

$$K_P = \frac{I^n \cdot T_C \cdot R}{B \cdot t} = 4^2 \cdot 3 \cdot 3 \,/\, 198{,}1 \cdot 0{,}08 = 8{,}72.$$

Якщо $K_P > 1$, то науково-дослідну роботу на тему «Розробка методів і програмних засобів дослідження тредів та процесів в операційній системі Linux»

можна вважати ефективною з високим науковим, технічним і економічним рівнем.

4.4 Висновки

Витрати на проведення науково-дослідної роботи на тему «Розробка методів і програмних засобів дослідження тредів та процесів в операційній системі Linux» складають 198123,86 грн. Відповідно до проведеного аналізу та розрахунків рівень наукового ефекту проведеної науково-дослідної роботи на тему «Розробка методів і програмних засобів дослідження тредів та процесів в операційній системі Linux» є середній, а дослідження актуальними, рівень доцільності виконання науково-дослідної роботи $K_P > 1$, що свідчить про потенційну ефективність з високим науковим, технічним і економічним рівнем.

ВИСНОВКИ

В першому розділі проведено обгрунтування вибору методу аналізу тредів та процесів в операційній системі Linux, де були визначені задані дослідження. Змістовно описано аналіз методів дослідження тредів та процесів в операційній системі Linux які базуються на використанні різних утиліт та інструментів, що надають детальну інформацію про процеси та потоки. Розглянуто аналіз планування процесів та тредів в операційній системі Linux.

В другому розділі проведено дослідження тредів та процесів в операційній системі Linux. Визначені особливості дослідження тредів та процесів в в операційній системі Linux.

Запропонований метод дослідження роботи процесів та тредів, де для перевірки ефективності методу розпаралелювання обробки даних було створена програмну реалізацію для ОС Linux, яка вимагала тривалих обчислень за допомогою робочих потоків. Потім, в основному потоці, формувалися дані, що використовувались у подальших дослідженнях. Проведені виміри часу роботи програми за різної кількості використовуваних паралельно працюючих потоків показали високу ефективність використання методу багатопоточних обчислень в ОС Linux.

В третьому розділі були виконані роботи з реалізацію методу досліджень, де розроблені вимоги щодо розробки системи управління тредів та процесів. Визначені структурно-функціональні особливості основних модулів програмного засобу. Виконано проектування, розробка та вповадження основних модулів програмного засобу.

Список використаної літератури

1. Yavorskyi D., Khoshaba O. Increasing game software performance due to threads and processes in the Linux operating system //III Всеукраїнська науково-технічна конференція молодих вчених, аспірантів та студентів "Комп’ютерні ігри і мультимедіа як інноваційний підхід до комунікації - 2023". Збірник матеріалів - Одеса, 2023. – С. 154-155.

2. Завертайло К.С., Хошаба О.М. Підвищення продуктивності в операційних системах шляхом вирішення конфліктних ситуацій між процесами /На шляху до індустрії 4.0:інформаційні технології, моделювання, штучний інтелект, автоматизація. Монографія.Одеса, 2021. -С.115-125.

3. Khoshaba O. M. Automated system to support the process of servicing maintenance stations [Електронний ресурс] / O. M. Khoshaba, D. Y. Chernega // Матеріали L науково-технічної конференції підрозділів ВНТУ, Вінниця, 10-12 березня 2021 р. – Електрон. текст. дані. – 2021. – Режим доступу: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2021/paper/view/12252.

4. Хошаба Олександр. Деякі рішення проблем управління потоками даних у хмарних середовищах [Електронний ресурс] / О. Хошаба, О. Романюк // Матеріали XV міжнародної конференції "Контроль і управління в складних системах (КУСС-2020)", м. Вінниця, 8-10 жовтня 2020 р.– Електрон. текст. дані. – Вінниця : ВНТУ, 2020. – Режим доступу: http://ir.lib.vntu.edu.ua//handle/123456789/30635.

5. Поджаренко В.О., Кучерук В.Ю., Севастьянов В.М. Основи мікро-процесорної техніки. Навчальний посібник. - Вінниця: ВНТУ, 2006. - 226 с.

6. Тарарака В.Д. Т19 Архітектура комп'ютерних систем: навчальний посібник. – Житомир : ЖДТУ, 2021. – 383 с.

7. Buyya R., Calheiros R. N., Son J., Dastjerdi A. V., Yoon Y. Software-defined cloud computing: Architectural elements and open challenges/ Advances in

Computing, Communications and Informatics (ICACCI) International Conference. – 2019. – 1–12 pp.

8. Хвостівська Л.В., Хвостівський М.О. Методичні вказівки для виконання лабораторних робіт з дисципліни “Архітектура ПК” для студентів спеціальностей 163 Біомедична інженерія та 172 Радіотехніка та телекомунікації. Тернопіль: ТНТУ, 2020. 146 с.

9. Hayes, B Cloud Computing (CC) Communications of the ACM. [Електронний ресурс] – Режим доступу до ресурсу: – http://delivery.acm.org/ 10.1145/1370000/1364786 – Дата доступу: 13.10.2023.

10. Singh S., Chana I. A Survey on Resource Scheduling in Cloud Computing: Issues and Challenges/ Journal of Grid Computing. – 2019. – 1–48 pp.

11. Hameed A., Khoshkbarforoushha A., Ranjan R., Jayaraman P. P., Kolodziej J., Bala-ji P., Zeadally S., Malluhi Q. M., Tziritas N., Vishnu A. A survey and taxonomy on energy efficient resource allocation techniques for cloud computing systems/ Computing. – 2021. – 1–24pp.

12. Баженов В. А. Інформатика. Комп’ютерна техніка. Комп’ютерні технології: підручник. – 3-е видання. – К.: Каравела, 2019.– 640 с.

13. Інформатика і комп’ютерна техніка: конспект лекцій / укладач А. В. Булашенко. – Суми: Сумський державний університет, 2022. – 232 с.

14. Madni S. H. H., Latiff M. S. A., Coulibaly Y. Resource scheduling for infrastructure as a service (IaaS) in cloud computing: Challenges and opportunities/ Journal of Network and Computer Applications. – 2022. – vol. 68. – 173–200 pp.

15. Карачка А., Дудко О. Архітектура комп'ютерів: навч. посіб. Тернопіль: Економічна думка, 2020. 181 с.

16. Мельник А.О. Архітектура комп’ютера : навч. посіб. Луцьк: Волинська обласна друкарня, 2020. 470 с.

17. Clark C., Fraser K., Hand S., Hansen J. G., Jul E., Limpach C., Pratt I., Warfield A. Live migration of virtual machines/ Proceedings of the 2nd conference

on Symposium on Networked Systems Design & Implementation.– 2018. – vol. 2. – 273–286 pp.

18. Цифрові пристрої та мікропроцесори. Організація та функціонування: навч. посібн. / О. М. Рисований, С. О. Соколов, І. С. Зиков, В. В. Скородєлов; /під ред. Рисованого О. М. – Харків : ХВУ, 2022. – 328 с.

19. Злобін Г. Г., Рикалюк Р. Є. Архітектура та апаратне забезпечення ПЕОМ: навч. посіб. Київ: Каравела, 2021. 224 с.

20. Рибалов Б.О. Архітектура комп'ютерів: Посібник до виконання лабораторних робіт./ Б.О. Рибалов; Одеська національна академія харчових технологій, 2019. – 43 с.

21. Wood T., Shenoy P. J., Venkataramani A., Yousif M. S. Black-box and Gray-box Strategies for Virtual Machine/ NSDI. – 2021. – vol. 7. – 17–19 pp.

22. Муляр В. П. Архітектура ЕОМ: лабораторний практикум. Луцьк : ВежаДрук, 2021. 112 с.

23. Daniel J. Barrett. Linux Pocket Guide: Essential Commands. (English Edition) 3rd Edition, Kindle Edition. 2018. 439p.

24. William Shotts. The Linux Command Line, 2nd Edition: A Complete Introduction. No Starch Press; 2nd edition (7 Mar. 2019) 504p. ISBN-10 1593279523, ISBN-13 978-1593279523

25. Jason Cannon. Linux for Beginners: An Introduction to the Linux Operating System and Command Line (English Edition) Kindle Edition. 2022.- 204 p

26. Christine Bresnahan, Richard Blum. Linux Command Bible 4e Paperback – 9 Jan. 2021. Wiley; 4. edition (9 Jan. 2021). 816p.

27. Jason Cannon. Command Line Kung Fu: Bash Scripting Tricks, Linux Shell Programming Tips, and Bash One-liners (English Edition) Kindle. 2022. - 956p.

28. Jason Cannon. Linux Administration: The Linux Operating System and Command Line Guide for Linux Administrators (English Edition). 2021. - 204 p.

29. Richard Petersen. Linux: The Complete Reference, Sixth Edition (Complete Reference Series). 2020.-830p.

30. Brian Ward. How Linux Works, 3rd Edition: What Every Superuser Should Know Paperback – 19 April 2021. - 464 p.

31. W. Stevens, Stephen Rago. Advanced Programming in the UNIX Environment (Addison-Wesley Professional Computing Series) Paperback – Illustrated, 14 May 2018. - 1032p.

32. Love Robert. Linux Kernel Development (Developer's Library) (English Edition) 3rd Edition, Kindle Edition. 2019.-731p.

33. Michael Kerrisk. The Linux Programming Interface: A Linux and UNIX System Programming Handbook. Hardcover – 1 Oct. 2020. - 1580 p.

34. Ellen Siever, Stephen Figgins, Robert Love, Arnold Robbins. Linux in a Nutshell: A Desktop Quick Reference Paperback – 21 Oct. 2019.-942p.

35. Операційні системи: [Електронний ресурс]: навч. посіб. для студ.спеціальності 123 «Комп’ютерна інженерія» / В. Г. Зайцев, І. П. Дробязко; КПІ ім. Ігоря Сікорського. – Електронні текстові дані (1 файл: 3 Мбайт). – Київ: КПІ ім. Ігоря Сікорського, 2019. – 240 с.

36. Методичні вказівки до виконання економічної частини магістерських кваліфікаційних робіт / Уклад. : В. О. Козловський, О. Й. Лесько, В. В. Кавецький. – Вінниця : ВНТУ, 2021. – 42 с.

ДОДАТКИ

Додаток А. Технічне завдання

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
д.т.н., проф. О. Н. Романюк
"19" вересня 2023 р.

Технічне завдання
на магістерську кваліфікаційну роботу «Розробка методів і програмних засобів дослідження тредів та процесів в операційній системі Linux» за спеціальністю
121 – Інженерія програмного забезпечення

Керівник магістерської кваліфікаційної роботи:
к.т.н., доцент О.М. Хошаба
"19" 09 2023 р.

Виконав:
студент гр. 2ПІ-22м Д. Г. Яворський
"19" 09 2023 р.

Вінниця – 2023 року

1. Найменування та галузь застосування

Магістерська кваліфікаційна робота: «Розробка методів і програмних засобів дослідження тредів та процесів в операційній системі Linux».

Галузь застосування – програмні засоби дослідження тредів та процесів в операційній системі Linux.

2. Підстава для розробки.

Підставою для виконання магістерської кваліфікаційної роботи (МКР) є індивідуальне завдання на МКР та наказ № 247 від 18 вересня 2023 року ректора по ВНТУ про закріплення тем МКР.

3. Мета та призначення розробки.

Метою роботи є підвищення ефективності використання потоків та процесів в операційній системі Linux за рахунок використання методів та механізмів багатопоточності.

Призначення роботи – розробка методів і засобів підвищення ефективності використання ефективності використання потоків та процесів в операційній системі Linux.

3 Вихідні дані для проведення НДР

Перелік основних літературних джерел, на основі яких буде виконуватись МКР.

1. Yavorskyi D., Khoshaba O. Increasing game software performance due to threads and processes in the Linux operating system //III Всеукраїнська науково-технічна конференція молодих вчених, аспірантів та студентів "Комютерні ігри і мультимедіа як інноваційний підхід до комунікації - 2023". Збірник матеріалів - Одеса, 2023. – С. 154-155.
2. Завертайло К.С., Хошаба О.М. Підвищення продуктивності в операційних системах шляхом вирішення конфліктних ситуацій між процесами /На

шляху до індустрії 4.0:інформаційні технології, моделювання, штучний інтелект, автоматизація. Монографія.Одеса, 2021. -С.115-125.

3. Khoshaba O. M. Automated system to support the process of servicing maintenance stations [Електронний ресурс] / O. M. Khoshaba, D. Y. Chernega // Матеріали L науково-технічної конференції підрозділів ВНТУ, Вінниця, 10-12 березня 2021 р. – Електрон. текст. дані. – 2021. – Режим доступу: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2021/paper/view/12252
4. Хошаба Олександр. Деякі рішення проблем управління потоками даних у хмарних середовищах [Електронний ресурс] / О. Хошаба, О. Романюк // Матеріали XV міжнародної конференції "Контроль і управління в складних системах (КУСС-2020)", м. Вінниця, 8-10 жовтня 2020 р.– Електрон. текст. дані. – Вінниця : ВНТУ, 2020. – Режим доступу: http://ir.lib.vntu.edu.ua//handle/123456789/30635.

4. Технічні вимоги

Використання CPU на виконання процесів та тредів - не більше 10%, загальне навантаження на систему (upload) - не більше 0.5, загальна кількість тредів в одному процесі - не більше 30, загальна кількість процесів - не більше 20, використання пам'яті - не більше 4 ГБайт, пріоритет для процесів - середній або низький, загальна кількість сторонніх процесів в системі - не більше 200, кількість запитів до програмного засобу - не менш 300 запитів за секунду, визначення розміру обробки запитах на серверах - не більш ніж 3000 мс.

5. Конструктивні вимоги.

Конструкція пристрою повинна відповідати естетичним та ергономічним вимогам, повинна бути зручною в обслуговуванні та керуванні.

Графічна та текстова документація повинна відповідати діючим стандартам України.

6. Перелік технічної документації, що пред’являється по закінченню

робіт:

– пояснювальна записка до МКР;

– технічне завдання;

– лістинги програми.

7. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

8. Стадії та етапи розробки:

№ з/п	Назва етапів магістерської кваліфікаційної Роботи	Строк виконання етапів роботи
1	Постановка задачі дослідження тредів та процесів в операційній системі Linux.	19.09.2023-02.10.2023
2	Аналіз та виконання методів дослідження тредів та процесів в операційній системі Linux.	03.10.2023-16.10.2023
3	Дослідження особливостей роботи тредів та процесів в операційній системі Linux.	17.10.2023-28.10.2023
4	Програмна реалізація методів дослідження тредів та процесів в операційній системі Linux.	29.10.2023-12.11.2023
5	Економічна частина	13.11.2023-01.12.2023

9. Порядок контролю та прийняття.

Виконання етапів магістерської кваліфікаційної роботи контролюється керівником згідно з графіком виконання роботи. Прийняття магістерської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком

Додаток Б. Протокол перевірки навчальної (кваліфікаційної) роботи

Назва роботи: **Розробка методів і програмних засобів дослідження тредів та процесів в операційній системі Linux**

Тип роботи: кваліфікаційна робота

Підрозділ : кафедра програмного забезпечення, ФІТКІ, 1ПІ – 21мз

Науковий керівник: к.т.н. доц. Хошаба О. М.

Unicheck	
Оригінальність	**97,5%**
Схожість	2,5 %

Аналіз звіту подібності

■ **Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.**

□ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.

□ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Заявляю, що ознайомлений з повним звітом подібності, який був згенерований Системою щодо роботи «Розробка методів і програмних засобів дослідження тредів та процесів в операційній системі Linux».

Особа, відповідальна за перевірку _____ Черноволик Г. О.

Опис прийнятого рішення: **допустити до захисту**

Автор роботи _____ Яворський Д.Г.

Керівник роботи _____ Хошаба О.М.

Додаток В. Лістинг програми

Фронтенд основного модуля:

```
<!DOCTYPE html>
<html lang="uk">
<head>
  <meta charset="UTF-8">
  <title>Головний модуль</title>
</head>
<body>
  <h1>Головний модуль</h1>
  <nav>
    <ul>
      <li><a href="#" onclick="loadModule('processManagement')">Керування процесами і тредами</a></li>
      <li><a href="#" onclick="loadModule('resourceAccess')">Спільний доступ до ресурсів</a></li>
      <li><a href="#" onclick="loadModule('systemResources')">Системні ресурси</a></li>
      <li><a href="#" onclick="loadModule('eventHandling')">Обробка подій та сигналів</a></li>
      <li><a href="#" onclick="loadModule('lifecycleManagement')">Управління життєвим циклом</a></li>
      <li><a href="#" onclick="loadModule('monitoringDebug')">Моніторинг і налагодження</a></li>
    </ul>
  </nav>
  <div id="moduleContent">
    <!-- Сюди буде завантажуватися вміст модуля -->
  </div>
```

```
<script>
  function loadModule(moduleName) {
    const moduleContent = document.getElementById('moduleContent');
    // Тут можна виконати запит до бекенду для отримання даних для відповідного модуля

    // Приклад імітації завантаження вмісту модуля
    switch (moduleName) {
      case 'processManagement':
        moduleContent.innerHTML = "<h2>Керування процесами і тредами</h2><p>- Створення, запуск, призупинення та завершення процесів і тредів.<br>- Призначення пріоритетів для регулювання їхньої важливості.</p>";
        break;
      case 'resourceAccess':
        moduleContent.innerHTML = "<h2>Спільний доступ до ресурсів</h2><p>- Механізми синхронізації та взаємодії (семафори, блокування, м'ютекси).<br>- Розподілення пам'яті та обмін даними між процесами або тредами.</p>";
        break;
      // Додайте case для інших модулів
      default:
        moduleContent.innerHTML = "<p>Виберіть модуль для відображення</p>";
    }
  }
</script>
</body>
</html>
```

```
<body>
    <h1>Операційна система: Модуль керування</h1>

    <!-- Керування процесами і тредами -->
    <h2>Керування процесами і тредами</h2>
    <button onclick="createProcess()">Створити процес</button>
    <button onclick="suspendProcess()">Призупинити процес</button>
    <!-- Інші елементи для керування процесами -->

    <!-- Спільний доступ до ресурсів -->
    <h2>Спільний доступ до ресурсів</h2>
    <button onclick="createSemaphore()">Створити семафор</button>
    <!-- Інші елементи для роботи з ресурсами -->

    <!-- Системні ресурси -->
    <h2>Системні ресурси</h2>
    <div id="resourceUsage">
        <!-- Інформація про використання ресурсів -->
    </div>
    <!-- Інші елементи для управління ресурсами -->

    <!-- Обробка подій та сигналів -->
    <h2>Обробка подій та сигналів</h2>
    <button onclick="registerSignalHandler()">Зареєструвати обробник
сигналу</button>
    <!-- Інші елементи для роботи з подіями та сигналами -->

    <!-- Управління життєвим циклом -->
    <h2>Управління життєвим циклом</h2>
    <button onclick="createThread()">Створити тред</button>
```

```
    <!-- Інші елементи для управління життєвим циклом -->

    <!-- Моніторинг і налагодження -->
    <h2>Моніторинг і налагодження</h2>
    <div id="debugInfo">
        <!-- Тут буде інформація про відладку -->
    </div>
    <!-- Інші елементи для моніторингу та відладки -->

    <script src="scripts.js"></script> <!-- Підключення скрипту з логікою -->
</body>

function createProcess() {
    // Логіка для створення процесу
    // Виклик відповідної функції чи API для створення нового процесу
}

function suspendProcess() {
    // Логіка для призупинення процесу
    // Виклик відповідної функції чи API для призупинення процесу
}

function setPriority(processId, priority) {
    // Логіка для призначення пріоритету процесу
    // Виклик відповідної функції чи API для призначення пріоритету
}

function createSemaphore() {
    // Логіка для створення семафору
    // Виклик відповідної функції чи API для створення семафору
```

```
    }

    function allocateMemory(size) {
        // Логіка для розподілу пам'яті
        // Виклик відповідної функції чи API для розподілу пам'яті
    }

    function sendInterProcessData(data) {
        // Логіка для обміну даними між процесами або тредами
        // Виклик відповідної функції чи API для обміну даними
    }

    function manageResources() {
        // Логіка для управління системними ресурсами (пам'ять, процесорний час, введення/виведення)
        // Виклик відповідної функції чи API для управління ресурсами
    }

    function handleEventsAndSignals() {
        // Логіка для обробки подій та сигналів
        // Виклик відповідної функції чи API для обробки подій та сигналів
    }

    function manageLifeCycle() {
        // Логіка для управління життєвим циклом процесів та тредів
        // Виклик відповідної функції чи API для створення та завершення процесів
    }

    function monitorAndDebug() {
```

```
    // Логіка для моніторингу та відлагодження
    // Виклик відповідної функції чи API для моніторингу та відладки
}
```

Збірник maven (файл pom.xml)

```
<!-- Залежності Spring Boot -->
<dependencies>
    <!-- Spring Web для створення веб-додатків -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <!-- Додаткова залежність для роботи з MySQL -->
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <scope>runtime</scope>
    </dependency>
</dependencies>
```

Збірник grade (файл build.grade)

```
// Залежності Spring Boot
implementation 'org.springframework.boot:spring-boot-starter-web'

// Додаткова залежність для роботи з MySQL
runtimeOnly 'mysql:mysql-connector-java'
```

Файл application.properties:

```
spring.datasource.url=jdbc:mysql://localhost:3306/db_name
```

```
spring.datasource.username=db_username
spring.datasource.password=db_password
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.jpa.hibernate.ddl-auto=update
```

Файл application.yml:

```
spring:
  datasource:
    url: jdbc:mysql://localhost:3306/db_name
    username: db_username
    password: db_password
    driver-class-name: com.mysql.cj.jdbc.Driver
  jpa:
    hibernate:
      ddl-auto: update
```

Створення моделей даних та репозиторіїв:

```
import javax.persistence.*;
@Entity
@Table(name = "your_table_name")
public class YourEntity {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String field1;
    private String field2;
    // getters/setters
}
import org.springframework.data.jpa.repository.JpaRepository;
```

```
public interface YourEntityRepository extends JpaRepository<YourEntity, Long> {
    // Власні методи, які будуть виконувати певні операції з базою даних
}
```

Створення контролерів:

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/api")
public class YourController {
    private final YourEntityRepository repository;

    @Autowired
    public YourController(YourEntityRepository repository) {
        this.repository = repository;
    }

    @GetMapping("/entities")
    public List<YourEntity> getAllEntities() {
        return repository.findAll();
    }
    // Методи для збереження, оновлення, видалення сутностей та інше, як потрібно
}
```

Програмний код для тестування JUnit в Spring Boot:

```
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
```

```
import org.springframework.beans.factory.annotation.Autowired;
import
org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMockMvc;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.junit.jupiter.SpringExtension;
import org.springframework.test.web.servlet.MockMvc;
import org.springframework.test.web.servlet.request.MockMvcRequestBuilders;
import org.springframework.test.web.servlet.result.MockMvcResultMatchers;
import                                                                static
org.springframework.test.web.servlet.result.MockMvcResultHandlers.print;

@ExtendWith(SpringExtension.class)
@SpringBootTest
@AutoConfigureMockMvc
public class YourControllerTest {

    @Autowired
    private MockMvc mockMvc;

    @Test
    public void shouldReturnEntities() throws Exception {
        this.mockMvc.perform(MockMvcRequestBuilders.get("/api/entities"))
                .andExpect(MockMvcResultMatchers.status().isOk())

.andExpect(MockMvcResultMatchers.content().contentType("application/json"))
                .andDo(print());
    }
    // Інші тести для інших методів контролера, таких як тести на збереження,
оновлення, видалення і т.д.
}
```

Контролер модуля для керування процесами і тредами:

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.Optional;

@Service
public class ProcessService {

    private final ProcessRepository processRepository;

    @Autowired
    public ProcessService(ProcessRepository processRepository) {
        this.processRepository = processRepository;
    }

    public List<ProcessEntity> getAllProcesses() {
        return processRepository.findAll();
    }

    public Optional<ProcessEntity> getProcessById(Long id) {
        return processRepository.findById(id);
    }

    public ProcessEntity createProcess(ProcessEntity process) {
        return processRepository.save(process);
    }

    public ProcessEntity updateProcess(Long id, ProcessEntity updatedProcess) {
```

```
        if (processRepository.existsById(id)) {
            updatedProcess.setId(id);
            return processRepository.save(updatedProcess);
        }
        return null; // or throw an exception
    }

    public boolean deleteProcess(Long id) {
        if (processRepository.existsById(id)) {
            processRepository.deleteById(id);
            return true;
        }
        return false;
    }
    // Методи для призупинення, завершення процесів та інші дії, які потрібні
}
```

Тестування сервісів та контролерів за допомогою JUnit в поєднанні з Mockito для створення залежностей:

```
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.mockito.MockitoAnnotations;

import java.util.Collections;
import java.util.List;
import java.util.Optional;

import static org.junit.jupiter.api.Assertions.assertEquals;
```

```
import static org.mockito.ArgumentMatchers.anyLong;
import static org.mockito.Mockito.*;

public class ProcessServiceTest {

    @Mock
    private ProcessRepository processRepository;

    @InjectMocks
    private ProcessService processService;

    @BeforeEach
    void setUp() {
        MockitoAnnotations.initMocks(this);
    }

    @Test
    void getAllProcesses_ShouldReturnListOfProcesses() {
        // Arrange
        when(processRepository.findAll()).thenReturn(Collections.emptyList());

        // Act
        List<ProcessEntity> processes = processService.getAllProcesses();

        // Assert
        assertEquals(0, processes.size());
        verify(processRepository, times(1)).findAll();
    }

    @Test
```

```
    void getProcessById_ShouldReturnProcessWhenFound() {
        // Arrange
        ProcessEntity mockProcess = new ProcessEntity();
        mockProcess.setId(1L);

when(processRepository.findById(1L)).thenReturn(Optional.of(mockProcess));

        // Act
        Optional<ProcessEntity> process = processService.getProcessById(1L);

        // Assert
        assertEquals(mockProcess.getId(), process.get().getId());
        verify(processRepository, times(1)).findById(anyLong());
    }

    // Інші тести для різних методів сервісу, таких як createProcess,
updateProcess, deleteProcess
}
```

Контролер модуля для спільного доступу до ресурсів:

```
// Залежності Spring Boot
implementation 'org.springframework.boot:spring-boot-starter-web'
implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
// Залежність для роботи з MySQL
runtimeOnly 'mysql:mysql-connector-java'
spring.datasource.url=jdbc:mysql://localhost:3306/db_name
spring.datasource.username=db_admin
spring.datasource.password=db_admin
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.jpa.hibernate.ddl-auto=update
```

```
import javax.persistence.*;

@Entity
@Table(name = "semaphores")
public class SemaphoreEntity {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private int value;

    // геттери/сеттери
}

import org.springframework.data.jpa.repository.JpaRepository;
public interface SemaphoreRepository extends JpaRepository<SemaphoreEntity,
Long> {
    // додаткові методи, якщо потрібно
}
// Сервіс
@Service
public class SemaphoreService {

    private final SemaphoreRepository semaphoreRepository;

    @Autowired
    public SemaphoreService(SemaphoreRepository semaphoreRepository) {
        this.semaphoreRepository = semaphoreRepository;
    }
```

```
    // Методи для взаємодії з семафорами, блокуваннями, м'ютексами та іншими ресурсами
    // ...
}

// Контролер
@RestController
@RequestMapping("/api/semaphores")
public class SemaphoreController {

    private final SemaphoreService semaphoreService;

    @Autowired
    public SemaphoreController(SemaphoreService semaphoreService) {
        this.semaphoreService = semaphoreService;
    }

    // API endpoints для роботи з ресурсами
    // ...
}
```

Додаток Г. Ілюстративний матеріал

ІЛЮСТРАТИВНИЙ МАТЕРІАЛ ДО ЗАХИСТУ МАГІСТЕРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Слайд 1 – Тема, автор, науковий керівник бакалаврської дипломної роботи:

Розробка методів і програмних засобів дослідження тредів та процесів в операційній системі Linux

Виконав:
студент групи 2ПІ-22м Яворський Д.Г.

Керівник:
к.т.н., доц. каф. ПЗ Хошаба О.М.

Слайд 2 – Мета, об'єкт та предмет дослідження:

Метою роботи є підвищення ефективності використання потоків та процесів в операційній системі Linux за рахунок використання методів та механізмів багатопоточності.

Об’єктом дослідження є процеси обробки даних в операційній системі Linux.

Предметом дослідження є методи та засоби підвищення ефективності використання потоків та процесів в операційній системі Linux.

Слайд 3 – Задачі дослідження:

Основними задачами дослідження є:

- сформувати задачі дослідження тредів та процесів в операційній системі Linux;

- виконати аналіз методів дослідження тредів та процесів в операційній системі Linux;

- розглянути аналіз планування процесів та тредів в операційній системі Linux;

- провести дослідження тредів та процесів в операційній системі Linux;

- визначити особливості дослідження тредів та процесів в в операційній системі Linux;

- запропоновати метод дослідження роботи процесів та тредів;

- розробити вимоги щодо розробки системи управління тредів та процесів;

- визначити структурно-функціональні особливості основних модулів програмного засобу;

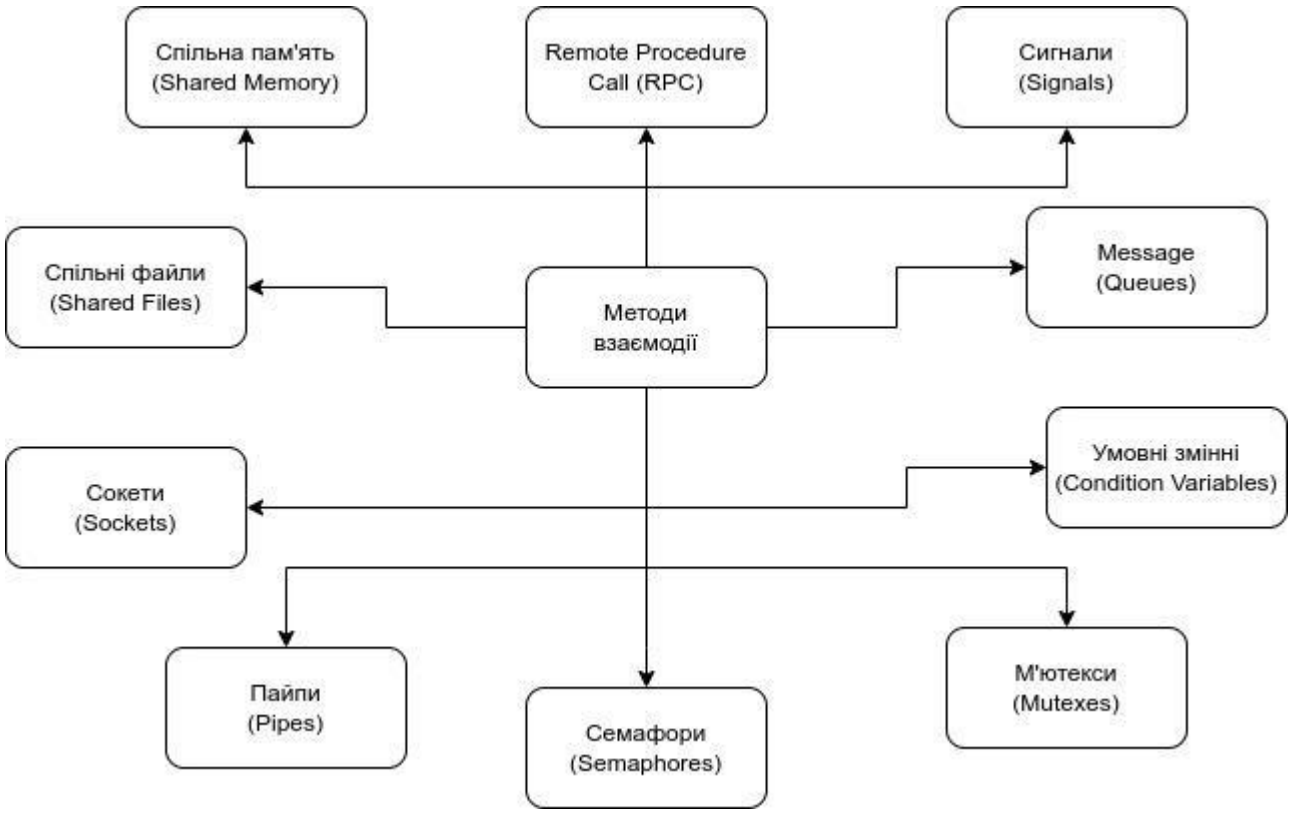
- виконати роботи з реалізації методу досліджень, де здійснити проектування, розробку та вповадження основних модулів програмного засобу.

Слайд 4 – Актуальність розробки:

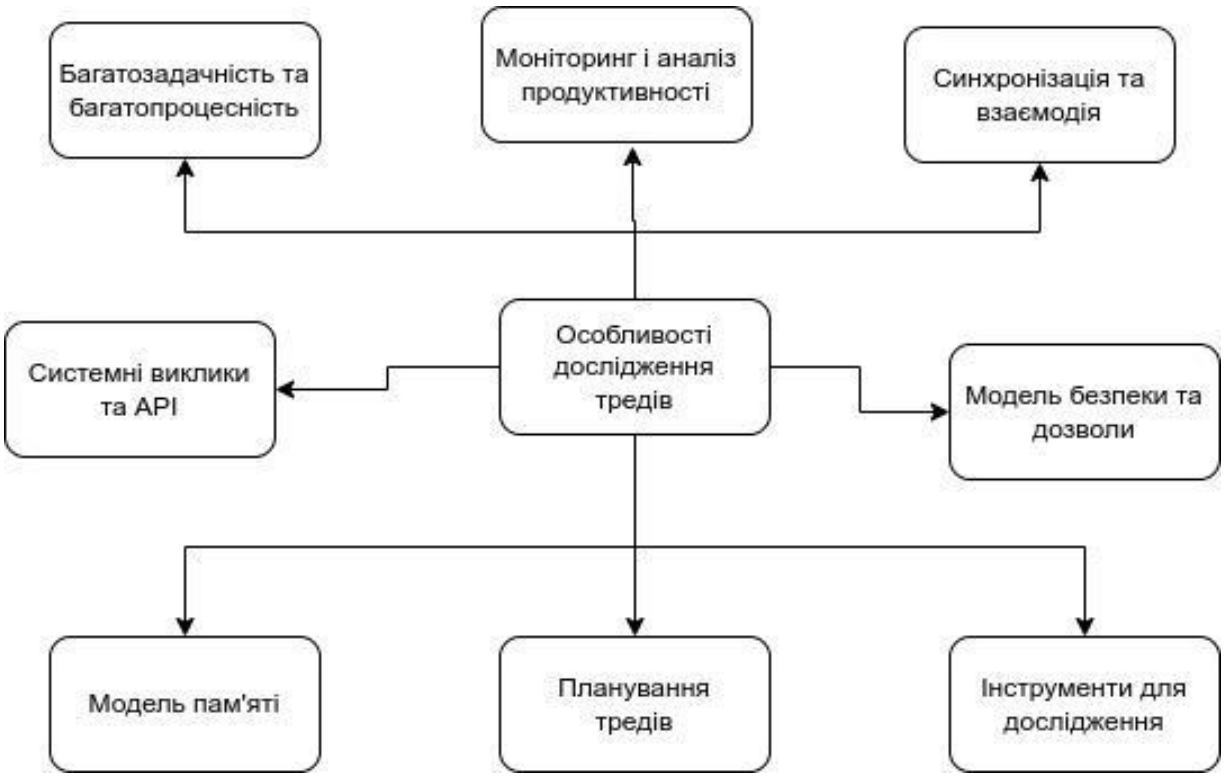
З подальшим розвитком операційних систем Linux виникають нові технології та можливості, такі як контейнеризація (наприклад, Docker та Kubernetes) і віртуалізація. Дослідження цих нововведень вимагає розробки спеціалізованих інструментів. Тому, є багато проектів з відкритим кодом, що допомагають в аналізі і моніторингу процесів та тредів в Linux.

Таким чином, актуальність розробки методів та програмних засобів для дослідження тредів та процесів в операційній системі Linux залишається важливою, оскільки ця область є ключовою для забезпечення ефективності, безпеки та стабільності Linux-систем у сучасному інформаційному середовищі.

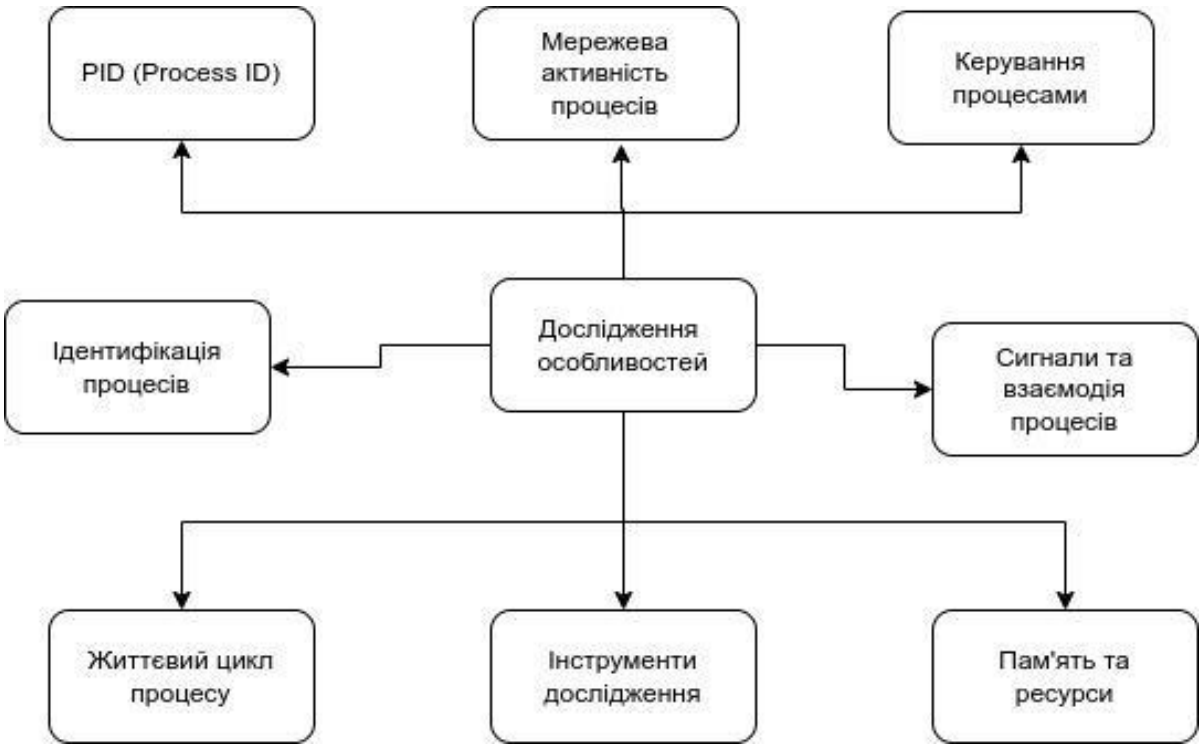
Слайд 5 – Взаємодія між процесами та тредами в операційній системі Linux



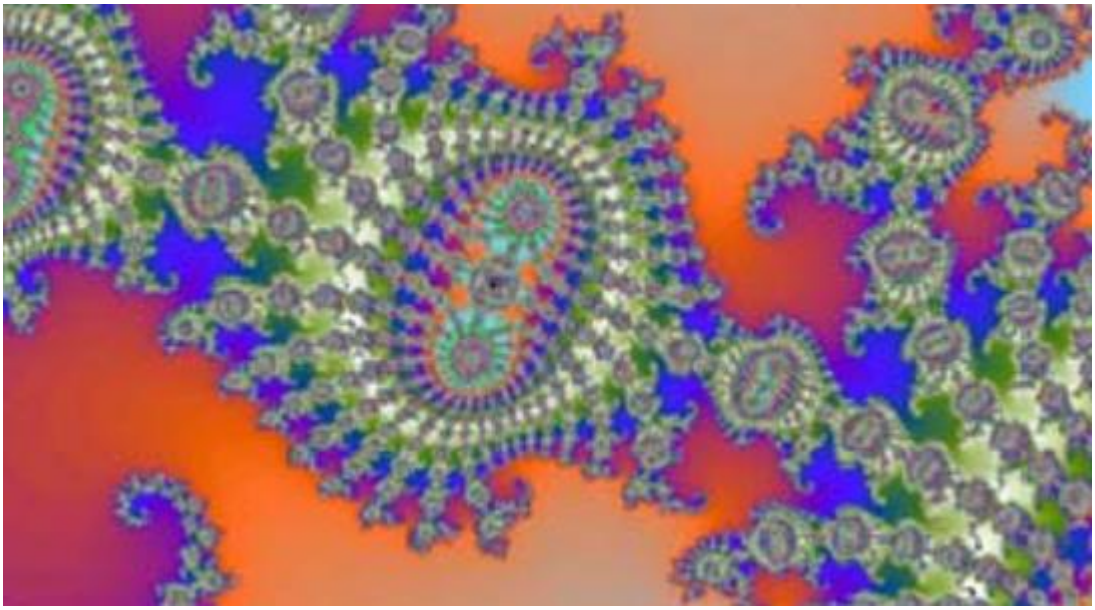
Слайд 6 — Основні особливості дослідження тредів в Linux



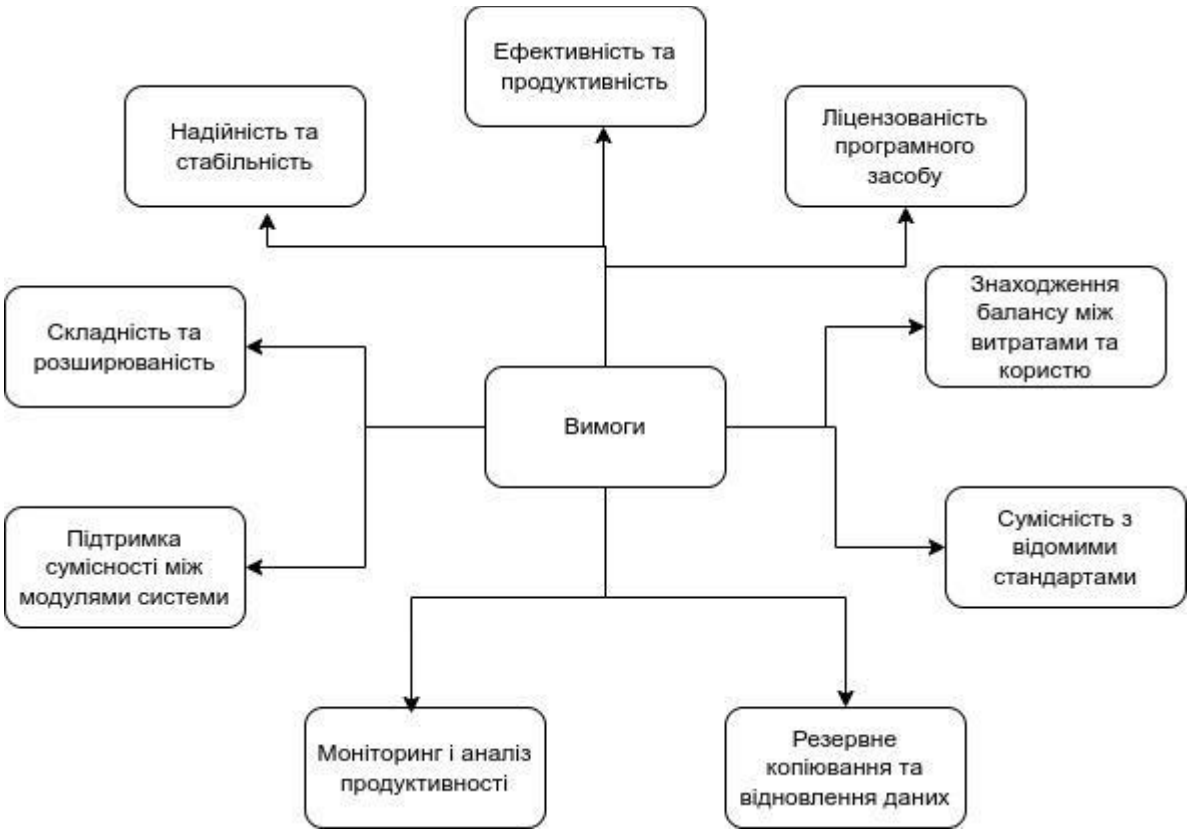
Слайд 7 — Деякі особливості дослідження процесів в операційній системі Linux



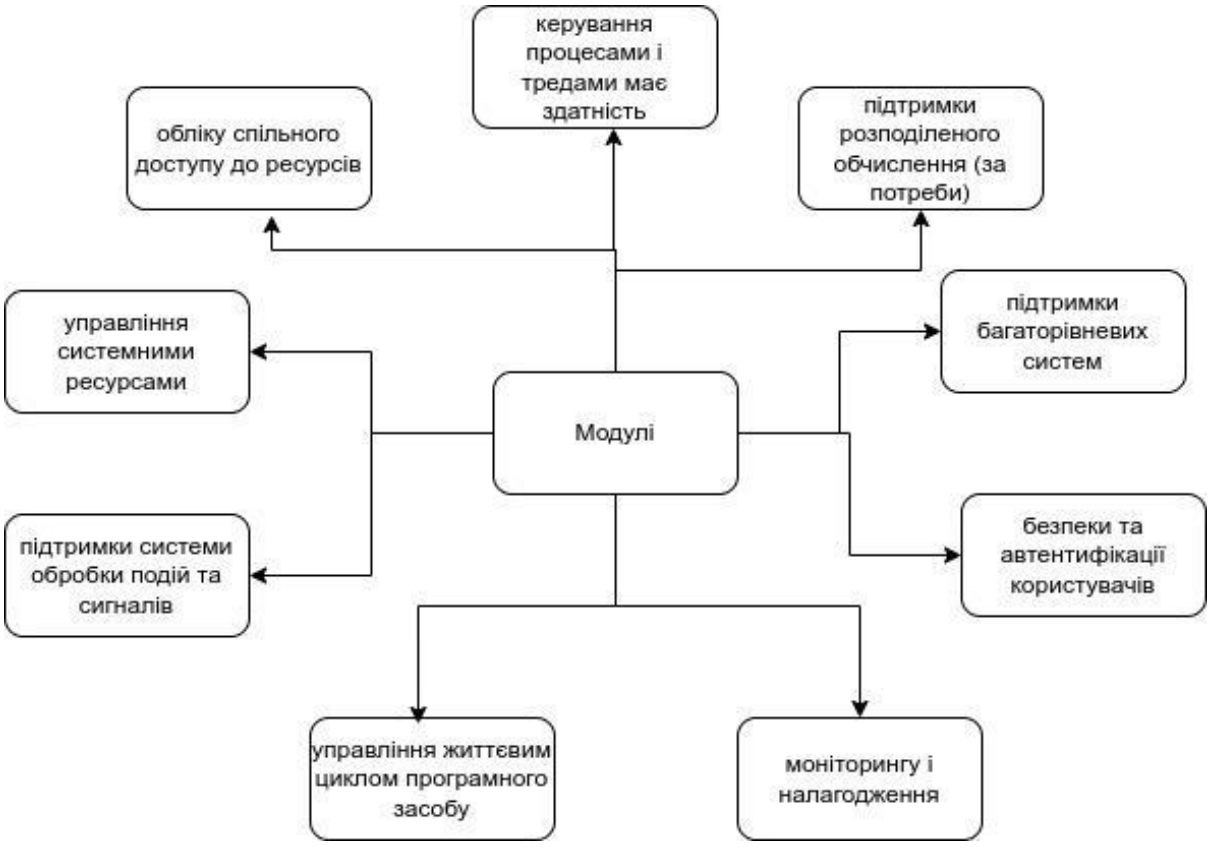
Слайд 8 — Багатопотокове оброблення даних на прикладі побудови зображення Мандельброта



Слайд 9 – Вимоги до розробки системи управління тредів та процесів в операційній системі Linux



Слайд 10 — Структурно-функціональні особливості модулів програмного засобу з управління тредами та процесами в операційній системі Linux



Слайд 11 – Наукова новизна одержаних результатів:

- вперше запропоновано метод оцінки продуктивности заданої кількості потоків, особливість якого полягає у виконанні обчислень зображення Мандельброта як фрактала, що визначений множиною точок на комплексній площині, і, як наслідок, дає можливість визначити математичну модель багатопоточності процесів або тредів;

- здобуло подальший розвиток метод обчислень зображення Мандельброта, який, на відміну від існуючих, полягає у його використані з метою дослідження процесів в операційній системі Linux, що дозволяє визначати математичні моделі часу побудови фракталу в багатопотокових програмних засобах.

Слайд 12 – Висновки:

В роботі:

- сформувані задачі дослідження тредів та процесів в операційній системі Linux;
- виконані аналіз методів дослідження тредів та процесів в операційній системі Linux;
- розглянуті аналіз планування процесів та тредів в операційній системі Linux;
- проведено дослідження тредів та процесів в операційній системі Linux;
- визначені особливості дослідження тредів та процесів в в операційній системі Linux;
- запропоновано метод дослідження роботи процесів та тредів;
- розроблені вимоги щодо розробки системи управління тредів та процесів;
- визначені структурно-функціональні особливості основних модулів програмного засобу;
- виконані роботи з реалізації методу досліджень, де здійснити проектування, розробку та вповадження основних модулів програмного засобу.

Основні положення магістерської кваліфікаційної роботи доповідалися та обговорювалися на Всеукраїнській науково-технічній конференції молодих вчених, аспірантів та студентів "Комютерні ігри і мультимедіа як інноваційний підхід до комунікації - 2023", (Одеса, 2023).

Основні результати досліджень опубліковано в науковій праці у матеріалах цієї конференції.