Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп'ютерних систем та мереж
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

магістр
(назва освітнього ступеня)

на тему: Методи та засоби планування обчислювальних завдань в комп'ютерній системі.

Виконав(ла): студент(ка) *VI* курсу, групи *СІм-62*
спеціальності *123 «Комп'ютерна інженерія»*
(шифр і назва спеціальності)

		Люлька А.В.
	(підпис)	(прізвище та ініціали)
Керівник		Луцків А.М.
	(підпис)	(прізвище та ініціали)
Нормоконтроль		Луцик Н.С.
	(підпис)	(прізвище та ініціали)
Завідувач кафедри		Осухівська Г. М.
	(підпис)	(прізвище та ініціали)
Рецензент		Гладьо Ю.Б.
	(підпис)	(прізвище та ініціали)

Тернопіль 2024

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет *комп'ютерно-інформаційних систем і програмної інженерії*
(повна назва факультету)

Кафедра *комп'ютерних систем та мереж*
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
Осухівська Г.М.
(підпис) (прізвище та ініціали)
«11» листопада 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня *магістр*
(назва освітнього ступеня)

за спеціальністю *123 «Комп'ютерна інженерія»*
(шифр і назва спеціальності)

студенту Люльці Андрію Вікторовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Методи та засоби планування обчислювальних завдань в комп'ютерній системі.

Керівник роботи Луцків Андрій Мирославович канд. тех. наук, доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «*29*» *листопада* *2024* року № *4/7-1144*

2. Термін подання студентом завершеної роботи 16.12.2024

3. Вихідні дані до роботи *наукові літературні джерела*

4. Зміст роботи (перелік питань, які потрібно розробити)
1. Аналіз предметної області дослідження
2. Теоретична частина
3. Практична частина
4. Охорона праці та безпека в надзвичайних ситуаціях

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
1. Актуальність і мета дослідження.
2. Алгоритм роботи послідовного розподілення Round-Robin.
3. Діаграма класів планувальника.
4. Схема баз даних.
5. Блок-схеми алгоритмів роботи основних модулів.
6. Схема роботи безперервного хешування.
7. Загальна схема роботи планувальника.
8. Висновки.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Охорона праці	*Осухівська Г.М., доцент*	05.12.2024	
Безпека в надзвичайних ситуаціях	*Стручок В.С., ст. викладач каф. Обладнання харчових технологій*	21.11.2024	

7. Дата видачі завдання 11.11.2024р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	*Аналіз існуючих методів та засобів розв'язання науково-технічних проблем, що відповідають тематиці кваліфікаційної роботи.*	20.11-30.11.24	*Виконано*
2.	*Обґрунтування актуальності дослідження*	01.12-06.12.24	*Виконано*
3.	*Аналізу предмету дослідження та предметної області*	08.12-13.12.24	*Виконано*
4.	*Оформлення розділу "Аналіз технічного завдання"*	14.12-15.12.24	*Виконано*
5.	*Оформлення розділу "Проектна частина"*	16.12-17.12.24	*Виконано*
6.	*Оформлення розділу "Практична частина"*	18.12-19.12.24	*Виконано*
7.	*Оформлення розділу "Охорона праці та безпека в надзвичайних ситуаціях"*	20.12-21.12.24	*Виконано*
8.	*Нормоконтроль*	11.12-14.12.24	*Виконано*
9.	*Попередній захист кваліфікаційної роботи магістра*	16.12.24	*Виконано*
10.	*Захист кваліфікаційної роботи магістра*	23.12.24	*Виконано*

Студент _____ Люлька Андрій Вікторович
(підпис) (прізвище та ініціали)

Керівник роботи _____ Луцків Андрій Мирославович
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Люлька А. В. Методи та засоби планування обчислювальних завдань в комп’ютерній системі: робота на здобуття кваліфікаційного ступеня магістра: спец. 123 — комп’ютерна інженерія / Луцків А. М.. Тернопіль: Тернопільський національний технічний університет імені Івана Пулюя, 2024. — 74 с.

Ключові слова: планувальники, обчислювальні одиниці, операційні системи, розподілені системи, балансувальники навантаження

Кваліфікаційна робота присвячена дослідженню роботи планувальників завдань в різних системах. Результатом проведення робіт є створений алгоритм з принципом безперервного хешування. Для випробовування даної розробки на реальних пристроях потрібно налаштувати цей алгоритм для роботи в конкретній комп’ютерній системі.

Результати цієї роботи можуть бути використані для імплементації логіки планування обчислювальних завдань в комп’ютерних системах. Розроблений алгоритм може суттєво пришвидшити процес планування, більше того, він дозволить максимально суттєво знизити перенаправлення виконуваних процесів від одної обчислювальної одиниці до іншої в разі необхідності.

ANOTATION

Liulka A.V. Methods and tools for scheduling computational tasks in a computer system. Master's Graduation Thesis: speciality 123 — Computer engineering / supervisor A.M. Lutskiv. Ternopil: Ternopil Ivan Puluj National Technical University, 2024. — 74 p.

Keywords: schedulers, calculation units, operational systems, distributed systems, load balancers

The qualification work is devoted to the study of the work of task schedulers in various systems. The result of the work is the creation of an algorithm with the principle of continuous hashing. To test this development on real devices, it is necessary to configure this algorithm to work in a specific computer system.

The results of this work can be used to implement the logic of scheduling computing tasks in computer systems. The developed algorithm can significantly speed up the planning process, moreover, it will allow to significantly reduce the redirection of executed processes from one computing unit to another if necessary.

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ, ОДИНИЦЬ, СИМВОЛІВ, ТЕРМІНІВ І ПОЗНАЧЕНЬ.

КС – комп’ютерна система

БШ – безперервне хешування

AI – штучний інтелект

Core – обчислювальне ядро

Quantum – час одного такту процесора

Хеш-функція – функція генерації хеш значення

Хеш-код – числове значення згенероване на основі вхідних даних

CFS – Completely Fair Scheduler

ЗМІСТ

ВСТУП

Актуальність теми. У сучасних комп’ютерних системах існують сотні активних процесів, з різними станами та пріоритетами, процесор з певною кількістю ядер немає змоги виконувати всі завдання паралельно, тому є необхідність застосування алгоритму правильного розподілення ресурсів поточної комп’ютерної системи на виконання послідовності процесів створених для вирішення тих чи інших завдань. Для того, щоб забезпечити максимально ефективне планування процесів, з урахуванням типу системи, обмеження ресурсів, часу виконання та інших факторів, використовуються планувальники виконання обчислювальних завдань.

Попри велику кількість планувальників, які застосовуються в різних системах, кожен з них має свою низку переваг та недоліків. Певні планувальники є максимально ефективними для систем з конкретними характеристиками апаратної складової. Існують планувальники, які призначені безпосередньо для роботи з сильно обмеженою кількістю ресурсів, також існують планувальники, які є максимально ефективними при необмеженій кількості ресурсів. Існує велика кількість сфер застосування планувальників, а саме операційні системи (Linux, Windows, LynxOs, RTOS, QNX), системи керуванням розподілених систем (Kubernetes, Docker Swarm), програми віртуалізації (Hyper-V, VirtualBox, VMWare).

Попит на планувальники є високий, оскільки, вони дають змогу знайти найефективніший шлях розподілу обчислювальних ресурсів для вирішення певної задачі.

Мета дослідження. Обґрунтувати вибір універсального алгоритму планування обчислювальних завдань з врахуванням обсягу обчислювальних ресурсів та врахуванням розрахунків про потенційне навантаження за сталу одиницю часу й імплементувати його.

Завдання кваліфікаційної роботи:

− Проаналізувати наявні вирішення для планування завдань між доступними обчислювальними ресурсами, виявити їх переваги та недоліки;

- Детально дослідити особливості та необхідність застосування певних алгоритмів планування в конкретних комп'ютерних системах;

- Дослідити особливість планування виконання процесів в операційних системах та розподілених системах;

− Розробити універсальне програмне забезпечення, для забезпечення максимально ефективного планування процесів серед доступних обчислювальних одиниць та зробити порівняльну характеристику з існуючими підходами.

Об'єкт дослідження. Процес планування виконання послідовних та паралельних процесів в обчислювальних системах.

Предмет дослідження. Планувальники та балансувальники навантаження комп'ютерних систем, призначення для розподілення завдань та балансування навантаження.

Методи дослідження. Метод моделювання, який використовується для створення моделі системи з обчислювальними ресурсами та послідовністю процесів, які повинні бути розподіленні та заплановані моделлю досліджуваного планувальника. Метод абстрагування, що дозволяє дослідити загальні властивості та ефективність роботи об'єкта дослідження, без його прикріплення до конкретної комп'ютерної системи.

Наукова новизна отриманих результатів. Наукова новизна отриманих результатів полягає у створенні максимально швидкого алгоритму планування вхідних процесів, який повинен виконувати планування процесів за сталу одиницю часу без залежності від кількості вхідних та запланованих процесів, та бути відмовостійким у разі неможливості будь якої випадкової обчислювальної одиниці продовжувати виконання запланованих завдань.

Практичне значення одержаних результатів. Розроблена модель планувальника процесів, потенційно може бути застосована в різних операційних

системах, в обчислювальних системах керування розподіленими сервісами (мікросервісами), в системах комп’ютерних системах розподілення навантаження вхідного потоку процесів між наявними обчислювальними ресурсами.

Публікації. Результати дослідження апробовано на XIII науково-технічній конференції Тернопільського національного технічного університету імені Івана Пулюя «Актуальні задачі сучасних технологій», XII міжнародній науково-технічній конференція молодих учених та студентів «Інформаційні моделі, системи та технології», у вигляді тез конференцій.

1. А. Луцків, А. Люлька Застосування методу безперервного хешування у плануванні виконання послідовних процесів. Актуальні задачі сучасних технологій: Праці XIII наук.-техн. конф. (Тернопіль, 11-12 грудня 2024 р.), Тернопіль, 2024. С. 410.

2. А. Луцків, А. Люлька Застосування алгоритмів балансування навантаження в процесах сучасних операційних систем. Інформаційні моделі, системи та технології: Праці XII наук.-техн. конф. (Тернопіль, 18-19 грудня 2024 р.), Тернопіль, 2024. С. 410.

Структура роботи. До складу кваліфікаційної роботи магістра входить пояснювальна записка та графічна частина. Пояснювальна записка включає вступ, 4 розділи, висновок, список використаної літератури та додатки. Обсяг роботи: пояснювальна записка – 74 арк. Формату А1, графічна частина – 6 аркушів формату А1.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ДОСЛІДЖЕННЯ

1.1. Обґрунтування доцільності створення ефективного планувальника обчислювальних задач у комп'ютерній системі

Доволі часто виникають задачі створення власних систем планування і керування виконанням обчислювальних завдань під особливості предметної області. У рамках професійної діяльності автор даної роботи здійснює розроблення системи виконання обчислювальних завдань у сфері біологічних досліджень. Користувачем системи виступає біолог, якому вона надає можливість моделювати біологічні процеси. Система реалізована на базі операційної системи Windows 10 і може працювати у хмарному середовищі і/або локально. Ці біологічні процеси транслюються та компілюються у певні DLL-бібліотеки, які й мають мати можливість до виконання. Доволі часто вченому потрібно мати можливість одночасно працювати з кількома різними процесами моделювання, а вони можуть мати різну обчислювальну складність. У даному випадку система має надати можливість ефективно використати наявні системні ресурси і максимально швидко та надійно виконати запущені задачі. Прикладом таких систем можуть служити системи метакластерних обчислень грід-системи [24] та спільно-комп'ютингові (крауд-комп'ютингові) системи типу BOINC [25]. Водночас дані системи не можуть бути інтегровані з уже наявною програмно-апаратною платформою у розробленні якої бере участь автор з точки зору відсутньої сумісності, ряді випадків надмірної складності наведених систем та ліцензійної несумісності.

Задача зі створення планувальника обчислювальних завдань для такої системи і є кінцевою метою даного дослідження. Для досягнення цієї мети буде здійснено огляд предметної області, а саме будуть проаналізовані алгоритми та методи планування обчислювальних завдань.

1.2. Обґрунтування та аналіз особливостей планування процесів

Планування обчислювальних процесів є дуже важливою задачею, оскільки переважна більшість комп'ютерних систем є багатозадачними й підтримують багатопроцесне опрацювання даних. Ефективний план виконання завдань, мінімізує час виконання, підвищує продуктивність та забезпечує оптимізацію використання обчислювальних ресурсів комп'ютерних систем. Від алгоритму планування напряму залежить продуктивність та стабільність роботи системи.

Існують два способи організації виконання процесів, а саме:

- Послідовний.
- Паралельний.

Послідовний спосіб передбачає організацію виконання процесів одного за одним. Цей метод виділяється надзвичайно простотою організації та керування з строгим дотриманням порядку виконання. Приклад послідовного виконання можна побачити на рис. 1.1, тобто всі завдання виконуються одне за одним. Такий метод виконання процесів ефективний коли нам важливо дотримуватись чіткого порядку виконання, наприклад коли виконання наступної задачі залежить від результату виконання попередньої.

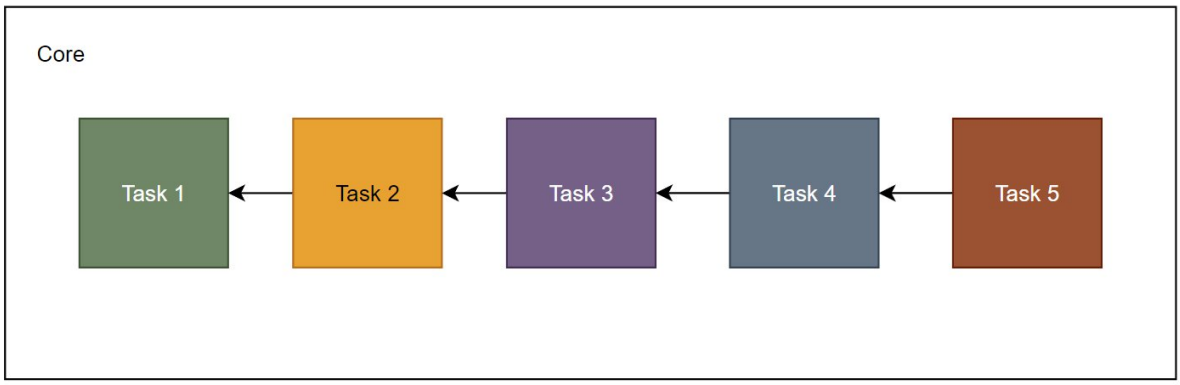


Рис. 1.1. Приклад виконання послідовно запланованих процесів

Послідовне виконання є частим способом планування процесів у базах даних або системах реального часу.

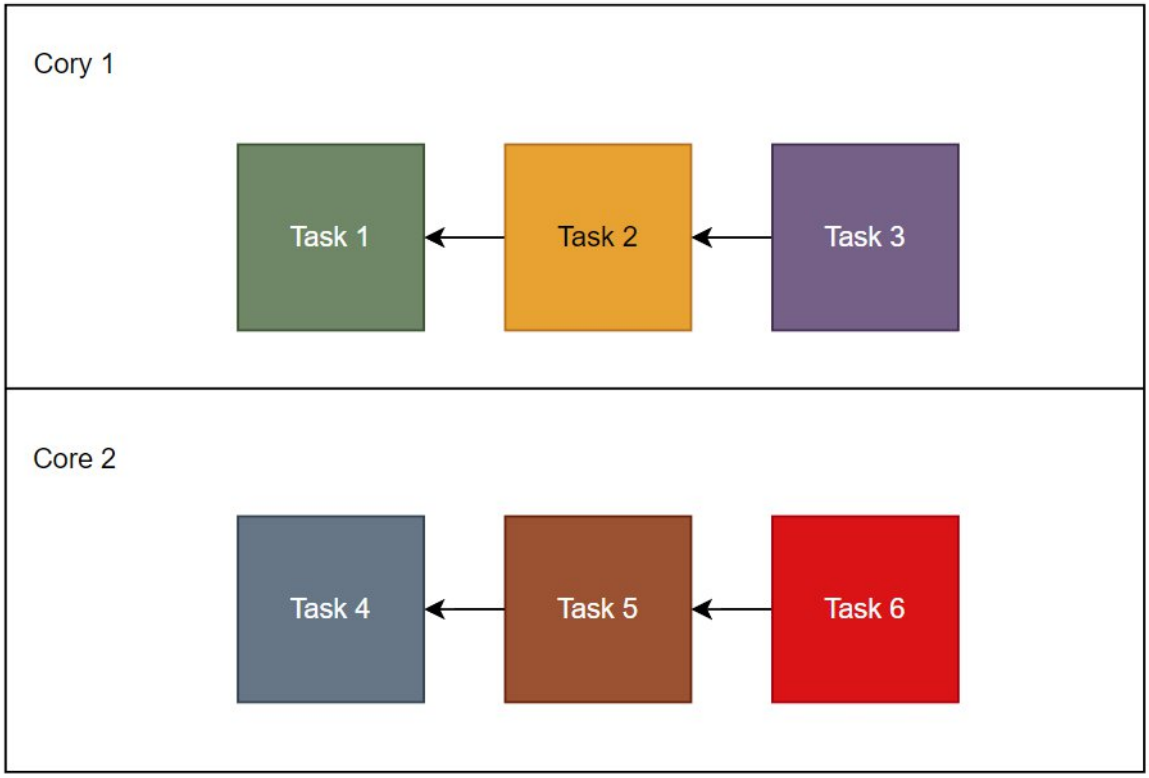


Рис. 1.2. Приклад виконання паралельно запланованих процесів

Паралельне виконання процесів, на відміну від послідовного, вимагає наявності додаткової фізичної чи віртуальної обчислювальної одиниці, наприклад ноди чи ядра. У даній роботі під терміном *обчислювальна одиниця* (compute unit) розуміємо довільний обчислювальни засіб, яким може виступати процесор, ядро процесора, обчислювальний вузол або “нода” (за умови використання кластера), спеціалізований обчислювач (DSP-процесор, FPGA-плата, GPU-процесор або його ядро тощо). При такому підході планування послідовності виконання декілька завдань можуть виконуватись одночасно. Приклад паралельного планування можна побачити на рис. 1.2, три завдання виконуються на кожній обчислювальній одиниці. Одночасне виконання багатьох завдань призводить до істотного збільшення продуктивності роботи комп’ютерної системи, проте вимагає більшої потужності апаратного забезпечення, більшого об’єму обчислювальних ресурсів та переважно більшого обсягу вбудованої пам’яті.

Такий тип планування завдань є однією з домінуючих парадигм у комп’ютерних системах на сьогоднішній день.

Паралельне виконання процесів часто використовується у багатоядерних комп’ютерних системах, кластерних обчисленнях, хмарних платформах та ін.

Попри низку переваг, паралельне планування виконання завдань є значно складнішим процесом, та має певні потенційні проблеми, які можуть виникнути, у порівнянні з послідовним, а саме:

- Синхронізація доступу до спільних ресурсів системи обчислювальними одиницями.
- Створення механізму уникнення стану гонитви (Race condition).
- Балансування навантаження.
- Уникнення взаємних “живих” та “мертвих” блокувань (dead/live lock).
- Узгодження результатів виконання для кінцевого представлення.

При плануванні виконання паралельних процесів слід використовувати динамічні підходи, які мають змогу адаптуватись до змінних умов виконання завдань.

У випадку відсутності додаткової обчислювальної одиниці, при великій необхідності часткового виконання декількох завдань одночасно, використовується принцип одночасності, який показаний на рис. 1.3.

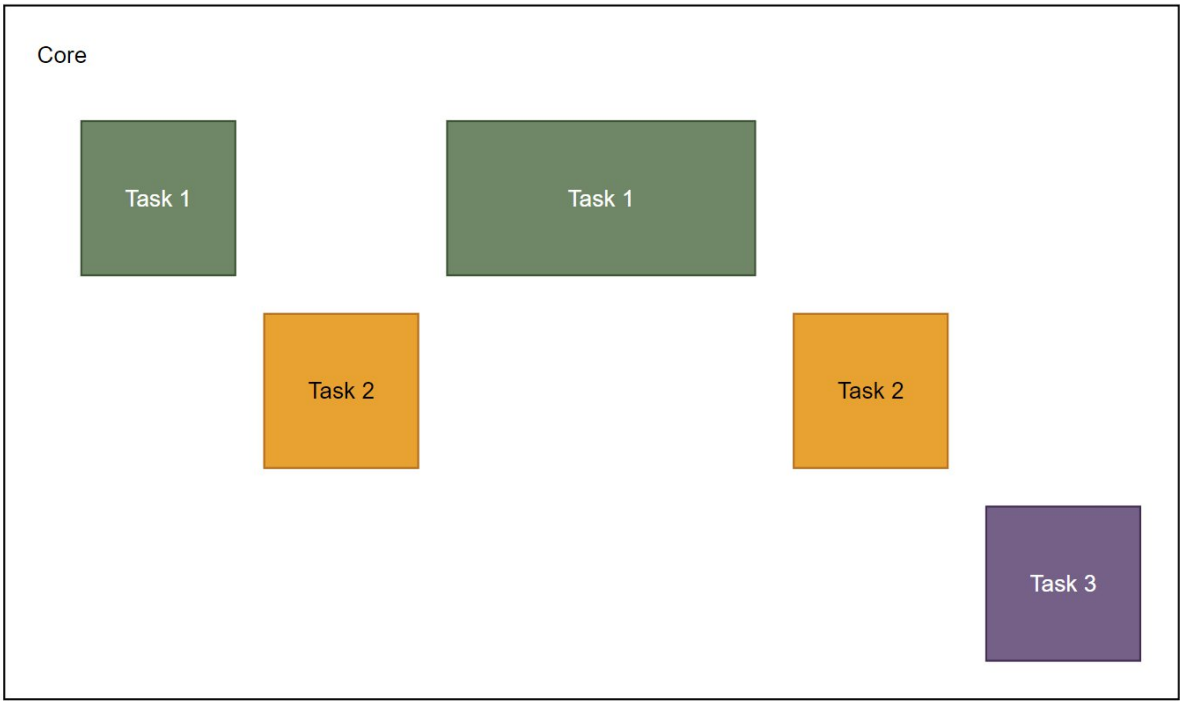


Рис. 1.3. Приклад одночасного виконання запланованих процесів

При одночасному виконанні, обчислювальна одиниця має змогу переключатись між виконуваними завданнями. При такому способі, відбувається часткове виконання всіх запланованих завдань і створюється симуляція того що ці процеси виконуються одночасно, при залучені лише однієї обчислювальної одиниці.

Вибір правильної тактики планування виконання процесів є надзвичайно важливим процесом, оскільки від цього залежить швидкодія та продуктивність цілої комп'ютерної системи. Планування виконання процесів є дуже важливим для сучасних технологій і подальше вдосконалення алгоритмів планування сприятиме оптимізації та пришвидшенню розв'язання складних обчислювальних завдань у різних сферах, від автоматизації невеликих організацій та установ до управління великими хмарними системами.

1.3. Аналіз планувальників виконання процесів та їх типів

Для того щоб виконувати планування процесів у системах, був створений центральний елемент обчислювальної системи під назвою "планувальник".

Планувальники зазвичай створюють для того щоб, грамотно розподіляти навантаження між ресурсами комп'ютеної системи (як балансувальники навантаження), дозволяючи різним компонентам системи ділитись ресурсами ефективно, без взаємних блокувань. Планувальники є невід'ємною частину будь яких комп'ютерних систем, оскільки ця концепція дає можливість створювати багатозадачність з одною обчислювальною одиницею.

Планувальник має одну або декілька цілей, а саме:

- Створити максимальну пропускну здатність (загальний обсяг роботи зроблений за одиницю часу).
- Мінімізувати час очікування (проміж часу між готовністю завдання до виконання і часом початку виконання).
- Зменшити затримку або час відповіді (проміж часу між готовністю завдання до виконання і повним завершенням або наданням першої відповіді керованого процесу у випадку інтерактивної взаємодії з планувальником).
- Збільшити справедливість розподілення ресурсів (однакову кількість ресурсів на всі процеси або більшу кількість ресурсів більш пріоритетним і важливим процесам).

Зокрема, в операційних системах, планувальник - це модуль який обирає наступні обчислювальні процеси для запуску та наступні процеси, які отримають допуск до системи. Планувальник процесів визначає який процес виконувати в кожен момент часу, він зазвичай має можливість призупиняти або виконувані процеси, продовжувати зупинені процеси, повертати їх назад у чергу процесів на виконання та почати виконувати нові процеси. Операційні системи можуть мати до трьох різних типів планувальників:

- Довгостроковий планувальник (також відомий як планувальник високого рівня).
- Середньостроковий планувальник.
- Короткостроковий планувальник.

Довгостроковий планувальник (long-term scheduler) – відповідає за визначення процесів, які мають дозвіл на потрапляння у чергу готовності. Тобто

коли в операційній системі відбувається спроба виконати процес, то допуск до існуючих процесів або приймається або затримується довгостроковим планувальником. Тому довгостроковий планувальник фактично диктує операційній системі, які процеси варто виконувати, а які ні, ступінь одночасності та паралелізму – чи багато чи мало процесів можуть виконуватись паралельно та одночасно, як буде відбуватись розподіл між I/O та CPU. Основне завдання цього планувальника, це дати зрозуміти чи достатньо доступних CPU ресурсів в поточний момент часу, для того щоб виконати вхідну чергу процесів.

Загалом, більшість процесів можна описати як I/O-bound або CPU-bound. I/O-bound процеси, це ті які виконують взаємодію здебільшо з системою вводу та виводу (тобто пам’яттю комп’ютера), та майже не залучені в обчисленнях. CPU-bound процеси, виконують здебільшо обчислення, та рідко взаємодіють з системою вводу/виводу.

Середньостроковий планувальник (medium-term scheduler) – тимчасово видаляє процеси з основної (оперативної пам’яті) та поміщає в другорядну пам’ять (жорсткий диск або SSD накопичувач). Цей планувальник може замінити неактивний процес, малопріоритетний процес, процес з великою кількістю помилок або процес який займає надто багато пам’яті, для того щоб звільнити пам’ять для інших процесів, та повернути ці процеси назад коли більше місця доступно в системі або процес розблокований та більше не очікує на ресурс.

У деяких системах середньостроковий планувальний може виконувати роль довгострокового планувальника.

Короткостроковий планувальник (short-term scheduler) – визначає, який саме з процесів виділених в оперативній пам’яті буде виконуватись після завершення процесором такту та перериванням або завершенням виконання процесу для виконання поточного процесу. Такі планувальники приймають такі рішення набагато частіше, ніж довгострокові та середньострокові планувальники, оскільки рішення щодо наступної запланованої задачі потрібно приймати відразу після кожного відрізку часу (time slice), які є дуже короткі. Цей планувальник може бути випереджувальним, тобто готовим примусово перервати виконувану

задачу з виконання центральним процесором, з наміром продовжити її виконання пізніше, та поставити на виконання нову задачу. Також цей планувальник може бути не випереджувальним, в такому випадку планувальник не може примусово перервати виконання поточної задачі, та повинен дочекатись її завершення.

Інший компонент, залучений у планування процесів в операційних систем, це “Диспетчер”. Цей модуль надає контроль центральному процесору до процесів, які були обрані короткостроковим планувальником. Тут відбувається контроль контекстів виконуваних процесів, тобто коли виконується переривання, диспетчер зберігає стан виконуваного процесу або потоку, що перед тим виконувався, та завантажує контекст нового процесу. Він має бути швидкий, оскільки виконується протягом кожного перемикання центрального процесора.

За принципом роботи, планувальник завдань використовує принцип теорії масового обслуговування (Queueing theory).

Перед тим як бути виконаним, завдання здебільшо потрапляє в чергу завдань, після чого, планувальник за принципом FIFO (First-in, First-out) розподіляє завдання між відповідними обчислювальними одиницями. За принципом FIFO, перше завдання, яке приходить в чергу на виконання, буде оброблене планувальником скоріше, ніж те яке приходить пізніше. Приклад роботи принципу FIFO можна побачити на рис. 1.4.

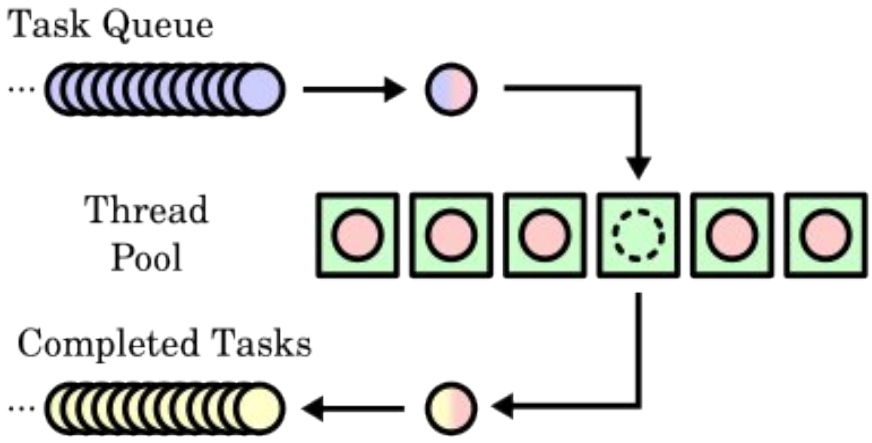


Рис. 1.4. FIFO принцип пулу потоків

1.4. Особливості роботи планувальника виконання процесів без пріоритету

При відсутності будь-якого пріоритету у всіх вхідних процесах планувальника, планувальник не виконує сортування процесів за пріоритетами, оскільки, всі процеси рівнозначні за важливістю виконання. При такому підході не має розділення на “більш важливі” та “менш важливі”. У системі з планувальником який виконує всі процеси без врахування пріоритету, всі процеси виконуються по черзі в порядку їх надходження в чергу завдань, тобто за принципом FIFO. Кожен процес одержить рівну кількість процесорних ресурсів, що при певних обставинах дозволяє пришвидшити роботу системи, оскільки, планувальник не витрачає час на визначення пріоритетів та правильне розподілення ресурсів за пріоритетністю.

Приклад планування виконання процесів без пріоритету можна побачити на рис. 1.5, всі процеси мають пріоритет за замовчуванням “1”, тому виконуються в порядку надходження в чергу.

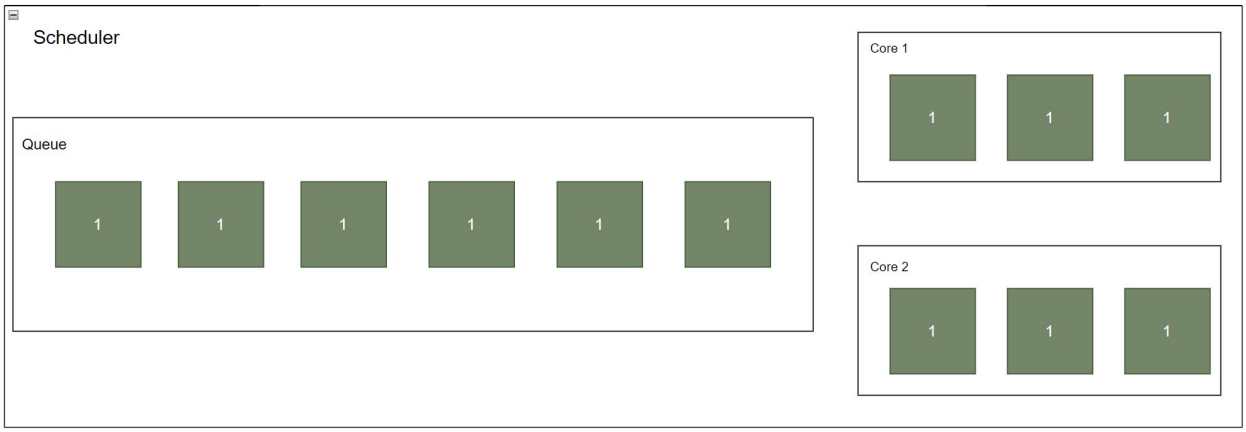


Рис. 1.5. Планування виконання процесів без пріоритету

1.5. Особливості роботи планувальника виконання процесів з пріоритетом

Планування процесів, які мають певний пріоритет полягає в тому, що планувальник виконує сортування процесів за їхнім пріоритетом, та не використовує принцип FIFO. Кожен вхідний процес у черзі має певний пріоритет,

цей пріоритет зазвичай визначається при створені цього процесу системою, тобто статично або динамічно, тобто може змінюватись в процесі виконання в залежності від різних сторонніх факторів (час очікування, змінні умови та ін.). Такий підхід планування має низку переваг, оскільки, дозволяє надає змогу системі ефективніше розподіляти обчислювальні ресурси, надаючи перевагу більш пріоритетним процесам на противагу менш пріоритетним, що може мати критично важливе значення у високопродуктивних та високонавантажених комп'ютерних системах.

При плануванні виконання процесів з пріоритетом, процеси з вищим пріоритетом отримують доступ до обчислювальних ресурсів раніше та швидше, ніж процеси з нижчим пріоритетом.

Приклад підходу виконання процесів з пріоритетом, можна побачити на рис. 1.6. На рисунку видно, що кожен вхідний процес має різний пріоритет, цей пріоритет вказаний числом та кожне число візуалізоване різним кольором. Обчислювальні одиниці "Core 1" та "Core 2", виконують ці вхідні процеси сортуючи за пріоритетом, тобто чим більше число процесу, тим раніше цей процес буде оброблений.

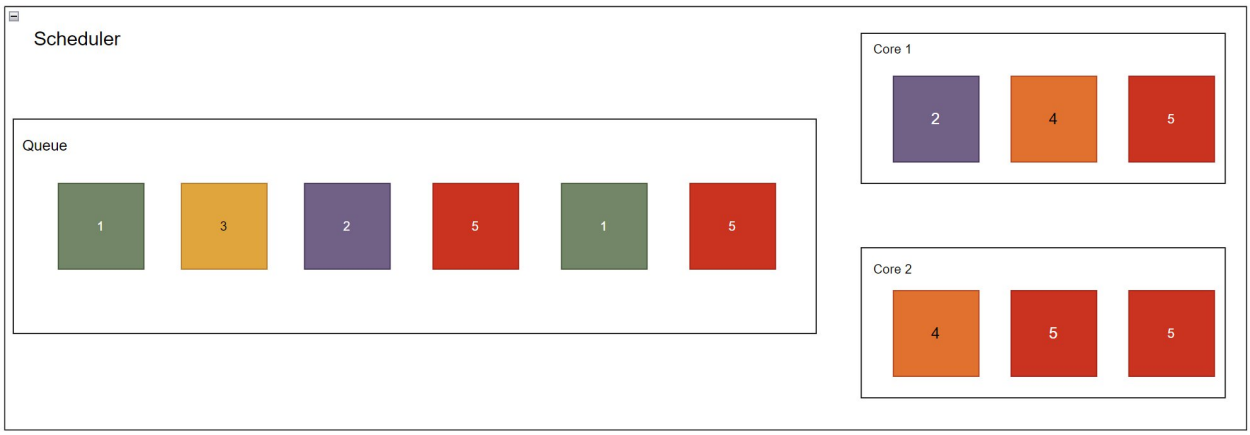


Рис. 1.6. Планування виконання процесів з пріоритетом

Підхід до планування виконання процесів з певним пріоритетом є більш частим рішенням більшості операційних та розподілених систем.

1.6. Алгоритми планування обчислювальних завдань в Unix-подібних операційних системах.

Unix-подібні операційні системи це операційні системи, які мають схожий принцип та особливість роботи до оригінальної Unix системи. Ці операційні системи використовують схожий спосіб керування процесами, ієрархічну структуру файлової системи, систему керування користувачам, інтерфейс командного рядка та ін., при тому інколи без використання вихідного коду система Unix.

Прикладами Unix-like операційних систем є:

- Серія BSD (FreeBSD, DragonFlyBSD, NetBSD, OpenBSD).
- macOS.
- Xenix Os.
- Linux.
- Solaris.
- AIX IBM.

Найбільшу популярність серед всіх Unix-подібних операційних систем, здобула операційна система, яка була розроблена Лінусом Торвальдсом у 1991 році, і отримала назву Linux, який на сьогоднішній день має велику кількість різних дистрибутивів (CentOS, Debian, Ubuntu, Kali, Fedora та ін.).

При розробці ядра Linux були використані різні алгоритми планування розподілення обчислювальних ресурсів між процесами. Планувальник Linux є невід'ємною частиною ядра, оскільки від правильності планування, безпосередньо залежить продуктивність та ефективність роботи операційної системи. На рис. 1.7 видно, що планувальник процесів є частиною та одним з центральних компонентів ядра Linux.

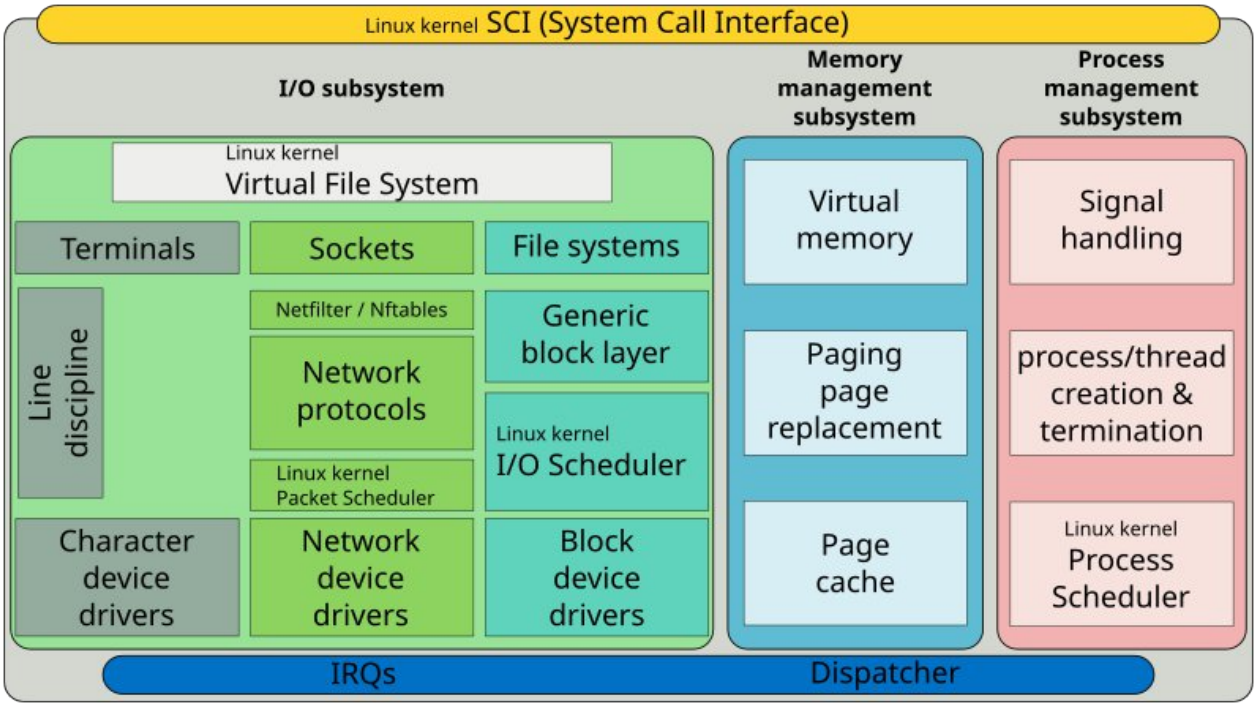


Рис. 1.7. Схема системних викликів ядра linux

Планувальник обчислювальних завдань в операційній системі linux, зазнав чимало змін від 1991 року. Його алгоритм удосконалювався, та інколи повністю замінювався на новий, більш оптимальний та продуктивний. Основними планувальниками, які були використані в ядрі Linux є: O(n) scheduler, O(1) scheduler, Completely Fair Scheduler, BF scheduler.

1.6.1. Планувальник O(n). Планувальник O(n) використовувався в Linux ядрі в версіях з 2.4 до 2.6. Алгоритм планувальника полягав в тому, що спочатку планувальник ділить процесор на певні епохи. У середині кожної епохи, кожен процес може виконуватись час заданий як time slice. Якщо процес не використовує повністю наданий час, тоді в наступній епосі, цьому процесу буде надано більше часу на половину. Недоліком цього алгоритму було те, що при збільшенні кількості процесів, зменшувалась швидкість планування, оскільки, при плануванні нового процесу планувальник повинен був ітеративно пройти весь список запланованих процесів.

На рис. 1.8 зображено візуальне представлення алгоритму роботи планувальника O(n).

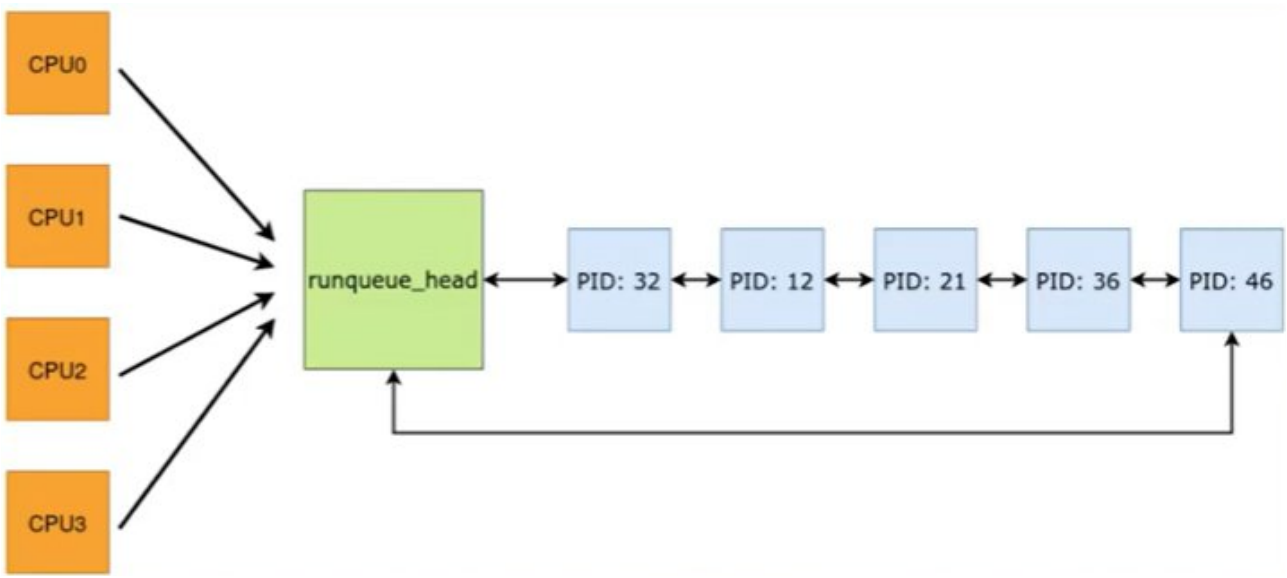


Рис. 1.8. Алгоритм роботи планувальника O(n) scheduler

1.6.2. Планувальник O(1). Планувальник O(1), з’явився в ядрі Linux у 2003 році, з релізом версії ядра 2.6.0. Його перевагою стало те, що він мав змогу планувати процеси до виконання за сталу одиницю часу незалежно від того, скільки процесів вже були заплановані або виконуються. Основна ідея алгоритму планувальника полягає в тому, що ми даємо спеціальну одиницю quantum кожному процесу, потім він переміщується в масив протермінованих процесів. Відразу як всі активні завдання використовують свій час quantum, масиви міняються місцями. Оскільки, доступ до цих масивів відбувається через вказівники, то планувальник просто перемикає вказівники на массив завдань. Активний масив завдань стає протермінованим масивом, а новостворений протермінований масив стає активним масивом. Найбільша проблема цього алгоритму полягала в тому, що процес визначення неінтерактивних та інтерактивних процесів був надто складний та ненадійний.

На рис. 1.9 зображено візуальне представлення алгоритму роботи планувальника O(1).

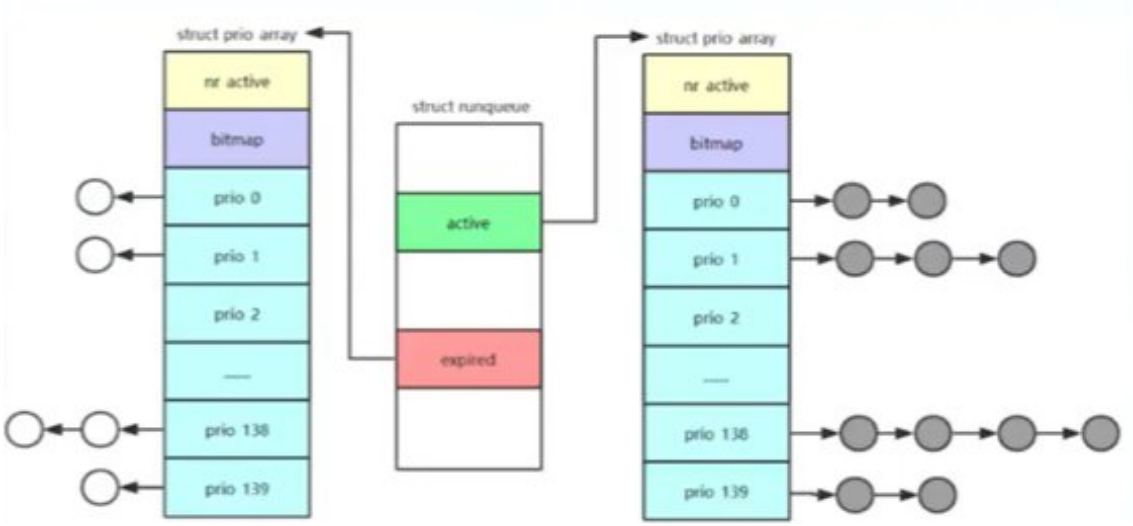


Рис. 1.9. Алгоритм роботи планувальника O(1) scheduler

1.6.3. Планувальник Completely Fair Scheduler. Планувальник CFS (Completely Fair Scheduler) був запроваджений у ядрі Linux у 2007 році, з релізом версії ядра 2.6.23. Його основна ідея полягає в тому, що, вся вхідна черга завдань, балансується в “червоно-чорному дереві” (англ. red-black tree - алгоритм балансування дерева елементів, який використовується в лгоритмах сортування), при тому балансування відбувається таким чином, що всі завдання з найменшим slice time переміщаються в ліву частину, а з найбільшим slice time в праву.

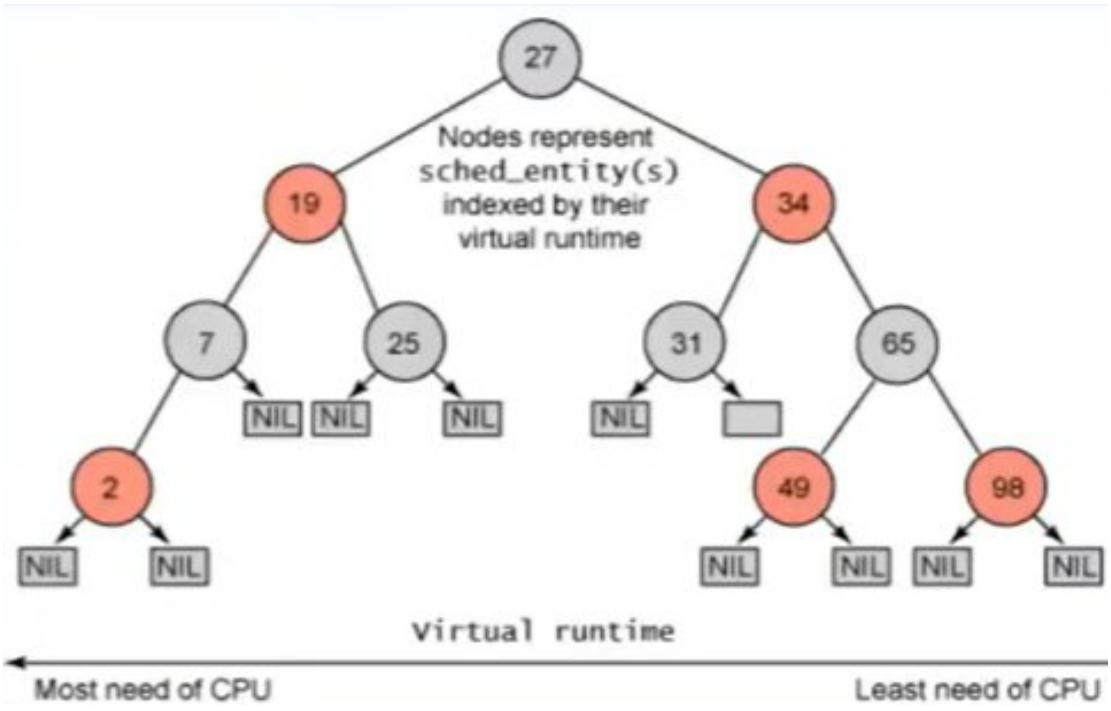


Рис. 1.10. Червоно-чорне дерево планувальника Completely Fair Scheduler

На рис. 1.10 зображено візуальне представлення червоно-чорного дерева, яке використовується в планувальнику Completely Fair Scheduler.

1.6.4. Планувальник BFS. Планувальник BFS, з'явився в ядрі Linux у 2009 році. За основу цього алгоритму було взяло принцип "Найраніший прийнятний віртуальний термін першого планування" (Earliest eligible virtual deadline first scheduling). Відмінність поточного алгоритму від всіх попередніх полягає в тому, що він використовує простий алгоритм, який не потребує додаткового коригування параметрів налаштування для адаптації продуктивності для певних пікових обчислювальних навантажень.

Було доведено, що планувальник BFS сприяє покращенню роботи Linux робочого стола на комп'ютерах з меншою кількістю ядер, ніж 16.

1.7. Планувальник системи керування розподіленими системами Kubernetes

Одним з найвідоміших рішень у сфері розподілених систем є рішення від компанії Google під назвою Kubernetes, яке дозволяє керувати великими розподіленими системами які зазвичай складаються з невеликих модулів. Kubernetes є відкритою системою автоматичного розгортання, масштабування та управління застосунками в контейнерах. З метою полегшення процесу управління обчислювальними ресурсами, була створена віртуальна одиниця обчислювальних ресурсів, під назвою "нода", яка є головним обчислювальним елементом Kubernetes.

При розробці Kubernetes, було завдання створити алгоритм для забезпечення правильності розподілення ресурсів віртуальних обчислювальних елементів (нод) та модулів системи (под), оскільки, кластери можуть складатися з багатьох віртуальних обчислювальних одиниць та багатьох модулів. Рішенням для даної проблеми, було створення Kubernetes планувальника, який містив в собі алгоритм розподілення модулів системи та ресурсів віртуальних обчислювальних

одиниць, закріплених за поточним кластером. Схема роботи Kubernetes планувальника зображена на рисунку 1.11.

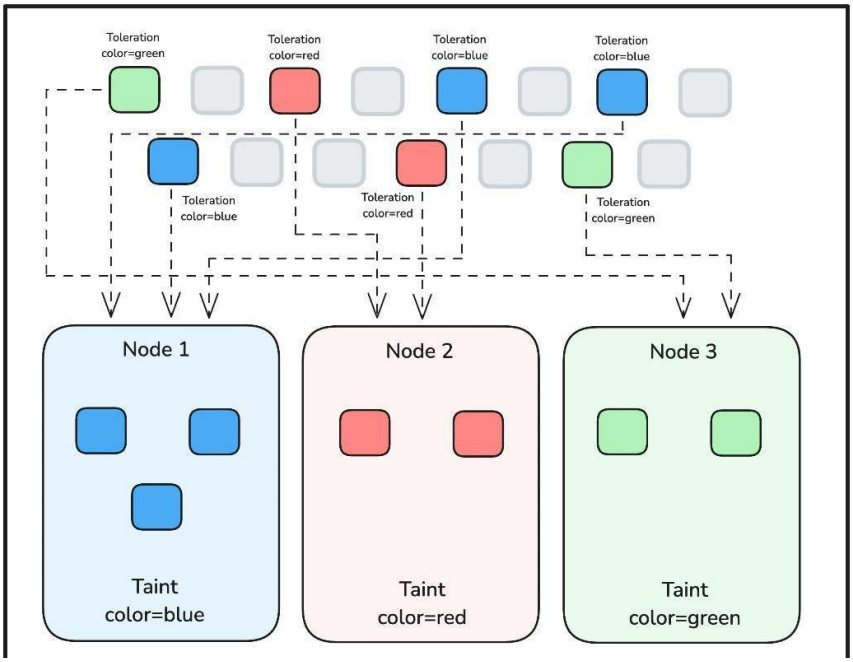


Рис. 1.11. Схема розподілення ресурсів у планувальнику Kubernetes

Загальний принцип алгоритму планувальника розподіленими сервісами Kubernetes наступний, коли в кластері створюється системний модуль (пода), планувальник дивиться чи немає спеціальних налаштувань кластера щодо зміни налаштувань за замовчуванням роботи планувальника. Якщо ніяких змін немає, то планувальник вибирає найбільш сумісну обчислювальну одиницю (ноду) з списку доступних та прикріплює до системного модуля (поди).

1.8. Висновки до розділу

Проведено детальний аналіз предметної області обраної теми. Детально досліджено особливості роботи алгоритмів існуючих планувальників операційних систем та систем керування розподіленими сервісами. Проаналізовано тактики і методи планувань існуючих алгоритмів з врахуванням переваг недоліків кожного з них.

РОЗДІЛ 2
ТЕОРЕТИЧНА ЧАСТИНА

2.1. Сучасні операційні системи та системи керування розподіленими сервісами

За останні роки операційні системи відчутно еволюціонували. Еволюція спричинена необхідністю задовільнити попит на обчислювальні потреби, від персональних комп'ютерів та мобільних телефонів і до корпоративних серверів. На них значною мірою вплинули історичні системи, зокрема Unix та його нащадки. Це призвело до створення надійних, масштабованих та безпечних платформ. Сьогодні існує дуже велика кількість різних операційних систем, зокрема Linux, планувальники завдань якого були проаналізовані у розділі 1. Значна частина емпіричного досвіду використання тих чи інших алгоритмів планування процесів прийшла до нас у результаті реалізації їх у даних операційних системах, які є багатозадачними та багатокористувацькими, а також можуть характеризуватись різним відгуком (англ. response time): настільні, серверні та операційні системи реального часу.

2.2. Особливість роботи планувальників комп'ютерних систем

При плануванні обчислювальних завдань алгоритм повинен бути максимально гнучким, тобто він має бути готовим до неочікуваної поведінки і раптових змін. Оскільки, припускається що досліджуваний алгоритм планування обчислювальних завдань буде працювати на різних обчислювальних одиницях для планування процесів, випадки раптового переключення у разі переривань чи раптового виведення з ладу є можливими з великою ймовірністю.

При плануванні обчислювальних завдань, потрібно враховувати такі речі як гарячі точки (англ. hotspots). Приклад гарячої точки можна побачити на рис. 2.1, де ядро "Core 1" виконує набагато більше завдань за інші ядра.

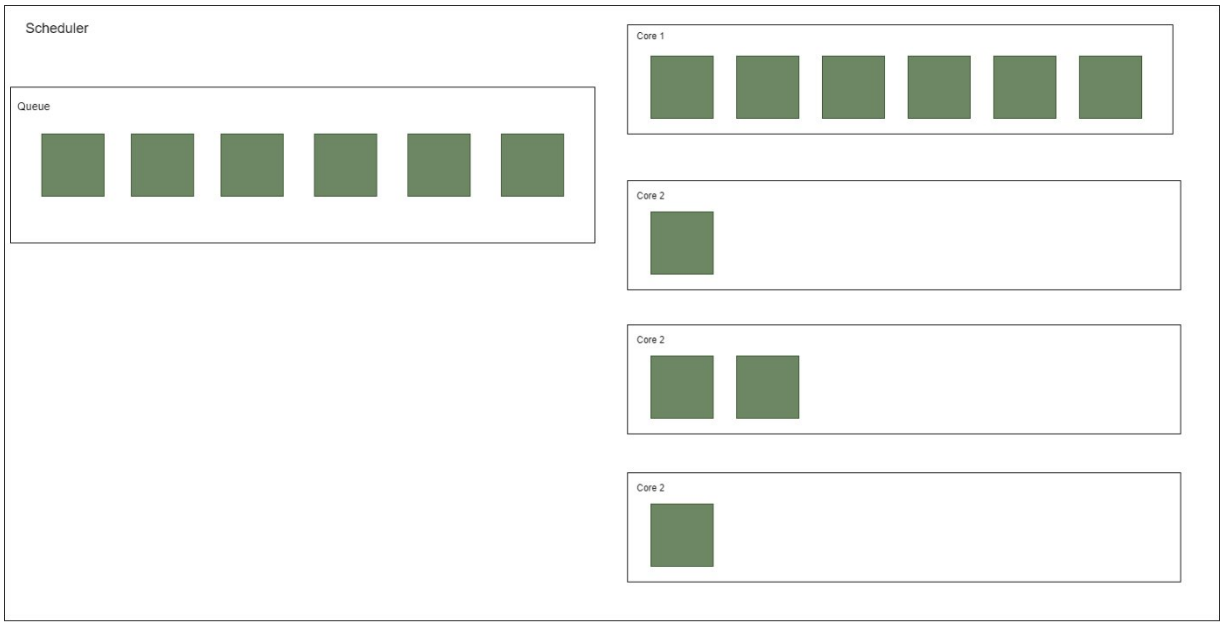


Рис. 2.1. Гаряча точка планувальника обчислювальних завдань

Гарячі точки – це окремі обчислювальні одиниці планувальника/балансувальника навантаження, які є значно більше навантажені відносно інших. Наявність гарячих точок свідчить про неправильно організований та спроєктований алгоритм планування або про відсутність передбачення перебалансування у випадку подібних проблем. У таких випадках обчислювальні одиниці, які сильно навантажені, можуть сповільнятись у швидкодії, аварійно завершувати операції з помилками та поводитися дуже непередбачувано.

Слід також враховувати спорідненість процесу (process affinity), тобто необхідність конкретного процесу бути закріпленим до певної обчислювальної одиниці, у такому випадку ця обчислювальна одиниця не може бути перервана або переключена на інші операції, поки ця операція, яка прикріплена до неї не буде завершена. Існують два типи спорідненості:

- М’яка спорідненість – система планування виконання дає запит на закріплення процесу до конкретного ядра до завершення його виконання.
- Жорстка спорідненість - система планування виконання примусово закріплює процес до конкретного ядра до завершення його виконання.

2.3. Алгоритм планування послідовних процесів Round-Robin

Round-Robin - це алгоритм, створення послідовності обробки процесів обчислювальною одиницею (оброблювальником). Принцип роботи цього алгоритму наступний, спочатку планувальник Round-Robin визначає квант часу виконання окремої частини обчислювального завдання - quantum (в планувальниках операційних систем, цей термін має назву - time slice) як час виконання процесором однієї задачі. Кожен процес на вхід має певний час виконання (burst time), спочатку цей час виконання quantum присвоюється кожному процесу який вже готовий до виконання, поки йде процес виконання першого такту обробника тривалістю *1 * quantum*, всі решта процеси добавляються в чергу на виконання. Якщо після завершення виконання процесу його виконання ще не завершене, тобто *(burst time - (1 * quantum)) > 0,* це означає, що процес повинен циклічно повторити своє виконання, цей процес переміщується назад в чергу виконання, така операція проводиться для кожного виконуваного процесу. Якщо процес виконано *(burst time - (1 * quantum)) <= 0*, то це означає що процес завершив своє виконання і для нього повторне виконання не виконується. Принцип роботи алгоритму доволі простий, та дозволяє рівномірно розподілити процесорний час між всіма доступними процесами. Принцип Round-Robin може застосовуватись також і до інших проблем планування, таких як планування пакетів даних в комп'ютерних мережах або розподілення ресурсів для обробки повідомлень в брокерах повідомлень, таких як Apache Kafka та RabbitMQ.

Приклад планування процесів за алгоритмом Round-Robin можна побачити на рис. 2.2. Задано 5 процесів: P1, P2, P3, P4, P5, кожен з яких має свою назву та свої певні параметри. Кожен вхідний процес має параметр BT (burst time), тобто час виконання цього процесу, та AT (arrival time), тобто час надходження цього процесу в чергу на виконання. Попередньо заданий час одного такту процесора quantum рівний числу 3. Спочатку виконуються в чергу на виконання ставляться всі процеси, час прибуття який менший за *1 * quantum*, це процеси P1, P2, P3, вони будуть виконуватись першими у порядку значення їхнього AT.

TIME SLICE = 4

PROCESS	ARRIVAL TIME	BURST TIME	
		TOTAL	REMAINING
P1	0	8	8
P2	1	6	6
P3	3	3	3
P4	5	2	2
P5	6	4	4

READY QUEUE:

P1	P2	P3	P1	P4	P5	P2

GANTT CHART:

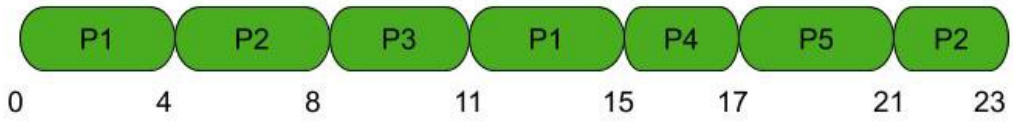
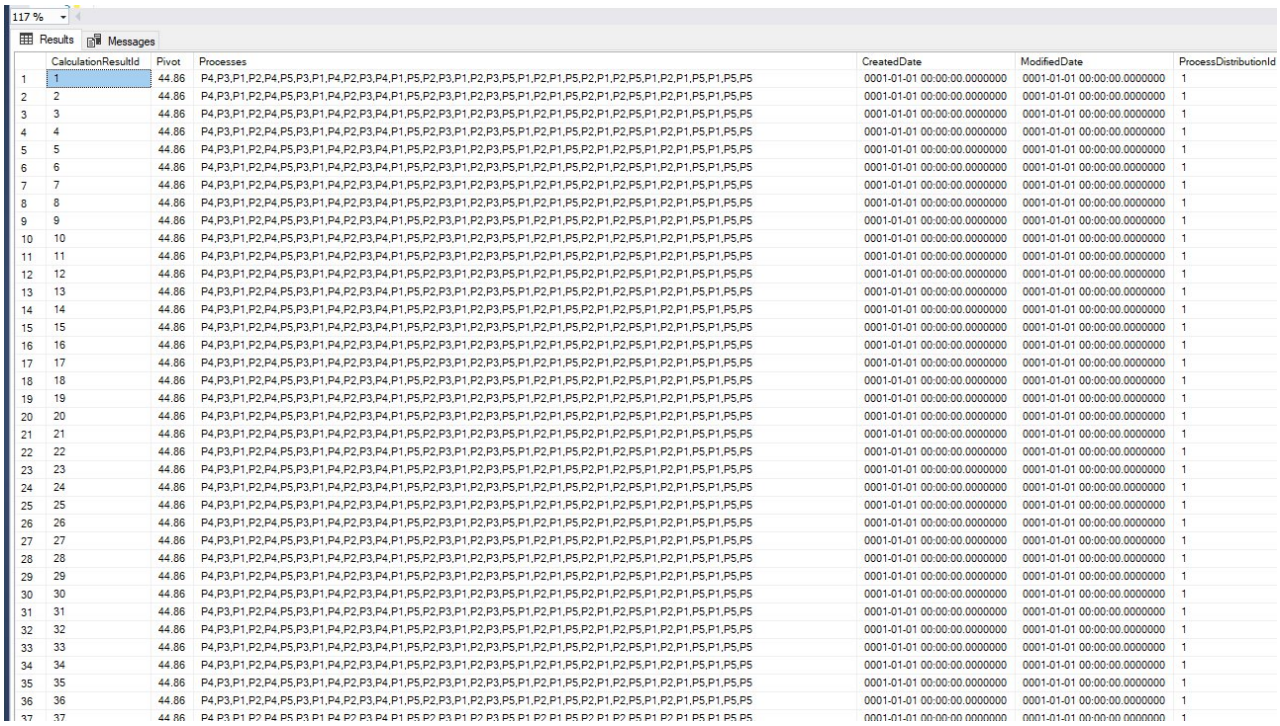


Рис. 2.2. Схема алгоритму виконання Round-Robin

Відбувається перевірка чи всі процеси були виконанні повністю, в поточному випадку процес P1 не виконаний повністю, тобто *BT(P1) – (1* quantum) > 0.* Тому наступний процес після перших трьох це знову P1. Після цього, проводиться аналіз всіх процесів у черзі, час виконання яких менший за *2 * quantum*, оскільки це вже другий такт роботи процесора, то відповідно перший множник у нас 2. Вибираємо всі процеси, які залишились, AT яких повинно бути менше 6, це процеси P4 та P5, ставимо їх наступними в чергу на виконання. Залишився ще незавершений процес P3, тому ставимо його наступним у чергу. Останній процес який залишився, це ще раз довиконати процес P5. В кінці, ми матимемо наступну послідовність: P1, P2, P3, P1, P4, P5, P3, P5.



117 %

Results | Messages

	CalculationResultId	Pivot	Processes	CreatedDate	ModifiedDate	ProcessDistributionId
1	1	44.86	P4,P3,P1,P2,P4,P5,P3,P1,P4,P2,P3,P4,P1,P5,P2,P3,P1,P2,P3,P5,P1,P2,P1,P5,P2,P1,P2,P5,P1,P2,P1,P5,P1,P5,P5	0001-01-01 00:00:00.0000000	0001-01-01 00:00:00.0000000	1
2	2	44.86	P4,P3,P1,P2,P4,P5,P3,P1,P4,P2,P3,P4,P1,P5,P2,P3,P1,P2,P3,P5,P1,P2,P1,P5,P2,P1,P2,P5,P1,P2,P1,P5,P1,P5,P5	0001-01-01 00:00:00.0000000	0001-01-01 00:00:00.0000000	1
3	3	44.86	P4,P3,P1,P2,P4,P5,P3,P1,P4,P2,P3,P4,P1,P5,P2,P3,P1,P2,P3,P5,P1,P2,P1,P5,P2,P1,P2,P5,P1,P2,P1,P5,P1,P5,P5	0001-01-01 00:00:00.0000000	0001-01-01 00:00:00.0000000	1
4	4	44.86	P4,P3,P1,P2,P4,P5,P3,P1,P4,P2,P3,P4,P1,P5,P2,P3,P1,P2,P3,P5,P1,P2,P1,P5,P2,P1,P2,P5,P1,P2,P1,P5,P1,P5,P5	0001-01-01 00:00:00.0000000	0001-01-01 00:00:00.0000000	1
5	5	44.86	P4,P3,P1,P2,P4,P5,P3,P1,P4,P2,P3,P4,P1,P5,P2,P3,P1,P2,P3,P5,P1,P2,P1,P5,P2,P1,P2,P5,P1,P2,P1,P5,P1,P5,P5	0001-01-01 00:00:00.0000000	0001-01-01 00:00:00.0000000	1
6	6	44.86	P4,P3,P1,P2,P4,P5,P3,P1,P4,P2,P3,P4,P1,P5,P2,P3,P1,P2,P3,P5,P1,P2,P1,P5,P2,P1,P2,P5,P1,P2,P1,P5,P1,P5,P5	0001-01-01 00:00:00.0000000	0001-01-01 00:00:00.0000000	1
7	7	44.86	P4,P3,P1,P2,P4,P5,P3,P1,P4,P2,P3,P4,P1,P5,P2,P3,P1,P2,P3,P5,P1,P2,P1,P5,P2,P1,P2,P5,P1,P2,P1,P5,P1,P5,P5	0001-01-01 00:00:00.0000000	0001-01-01 00:00:00.0000000	1
8	8	44.86	P4,P3,P1,P2,P4,P5,P3,P1,P4,P2,P3,P4,P1,P5,P2,P3,P1,P2,P3,P5,P1,P2,P1,P5,P2,P1,P2,P5,P1,P2,P1,P5,P1,P5,P5	0001-01-01 00:00:00.0000000	0001-01-01 00:00:00.0000000	1
9	9	44.86	P4,P3,P1,P2,P4,P5,P3,P1,P4,P2,P3,P4,P1,P5,P2,P3,P1,P2,P3,P5,P1,P2,P1,P5,P2,P1,P2,P5,P1,P2,P1,P5,P1,P5,P5	0001-01-01 00:00:00.0000000	0001-01-01 00:00:00.0000000	1
10	10	44.86	P4,P3,P1,P2,P4,P5,P3,P1,P4,P2,P3,P4,P1,P5,P2,P3,P1,P2,P3,P5,P1,P2,P1,P5,P2,P1,P2,P5,P1,P2,P1,P5,P1,P5,P5	0001-01-01 00:00:00.0000000	0001-01-01 00:00:00.0000000	1
11	11	44.86	P4,P3,P1,P2,P4,P5,P3,P1,P4,P2,P3,P4,P1,P5,P2,P3,P1,P2,P3,P5,P1,P2,P1,P5,P2,P1,P2,P5,P1,P2,P1,P5,P1,P5,P5	0001-01-01 00:00:00.0000000	0001-01-01 00:00:00.0000000	1
12	12	44.86	P4,P3,P1,P2,P4,P5,P3,P1,P4,P2,P3,P4,P1,P5,P2,P3,P1,P2,P3,P5,P1,P2,P1,P5,P2,P1,P2,P5,P1,P2,P1,P5,P1,P5,P5	0001-01-01 00:00:00.0000000	0001-01-01 00:00:00.0000000	1
13	13	44.86	P4,P3,P1,P2,P4,P5,P3,P1,P4,P2,P3,P4,P1,P5,P2,P3,P1,P2,P3,P5,P1,P2,P1,P5,P2,P1,P2,P5,P1,P2,P1,P5,P1,P5,P5	0001-01-01 00:00:00.0000000	0001-01-01 00:00:00.0000000	1
14	14	44.86	P4,P3,P1,P2,P4,P5,P3,P1,P4,P2,P3,P4,P1,P5,P2,P3,P1,P2,P3,P5,P1,P2,P1,P5,P2,P1,P2,P5,P1,P2,P1,P5,P1,P5,P5	0001-01-01 00:00:00.0000000	0001-01-01 00:00:00.0000000	1
15	15	44.86	P4,P3,P1,P2,P4,P5,P3,P1,P4,P2,P3,P4,P1,P5,P2,P3,P1,P2,P3,P5,P1,P2,P1,P5,P2,P1,P2,P5,P1,P2,P1,P5,P1,P5,P5	0001-01-01 00:00:00.0000000	0001-01-01 00:00:00.0000000	1
16	16	44.86	P4,P3,P1,P2,P4,P5,P3,P1,P4,P2,P3,P4,P1,P5,P2,P3,P1,P2,P3,P5,P1,P2,P1,P5,P2,P1,P2,P5,P1,P2,P1,P5,P1,P5,P5	0001-01-01 00:00:00.0000000	0001-01-01 00:00:00.0000000	1
17	17	44.86	P4,P3,P1,P2,P4,P5,P3,P1,P4,P2,P3,P4,P1,P5,P2,P3,P1,P2,P3,P5,P1,P2,P1,P5,P2,P1,P2,P5,P1,P2,P1,P5,P1,P5,P5	0001-01-01 00:00:00.0000000	0001-01-01 00:00:00.0000000	1
18	18	44.86	P4,P3,P1,P2,P4,P5,P3,P1,P4,P2,P3,P4,P1,P5,P2,P3,P1,P2,P3,P5,P1,P2,P1,P5,P2,P1,P2,P5,P1,P2,P1,P5,P1,P5,P5	0001-01-01 00:00:00.0000000	0001-01-01 00:00:00.0000000	1
19	19	44.86	P4,P3,P1,P2,P4,P5,P3,P1,P4,P2,P3,P4,P1,P5,P2,P3,P1,P2,P3,P5,P1,P2,P1,P5,P2,P1,P2,P5,P1,P2,P1,P5,P1,P5,P5	0001-01-01 00:00:00.0000000	0001-01-01 00:00:00.0000000	1
20	20	44.86	P4,P3,P1,P2,P4,P5,P3,P1,P4,P2,P3,P4,P1,P5,P2,P3,P1,P2,P3,P5,P1,P2,P1,P5,P2,P1,P2,P5,P1,P2,P1,P5,P1,P5,P5	0001-01-01 00:00:00.0000000	0001-01-01 00:00:00.0000000	1
21	21	44.86	P4,P3,P1,P2,P4,P5,P3,P1,P4,P2,P3,P4,P1,P5,P2,P3,P1,P2,P3,P5,P1,P2,P1,P5,P2,P1,P2,P5,P1,P2,P1,P5,P1,P5,P5	0001-01-01 00:00:00.0000000	0001-01-01 00:00:00.0000000	1
22	22	44.86	P4,P3,P1,P2,P4,P5,P3,P1,P4,P2,P3,P4,P1,P5,P2,P3,P1,P2,P3,P5,P1,P2,P1,P5,P2,P1,P2,P5,P1,P2,P1,P5,P1,P5,P5	0001-01-01 00:00:00.0000000	0001-01-01 00:00:00.0000000	1
23	23	44.86	P4,P3,P1,P2,P4,P5,P3,P1,P4,P2,P3,P4,P1,P5,P2,P3,P1,P2,P3,P5,P1,P2,P1,P5,P2,P1,P2,P5,P1,P2,P1,P5,P1,P5,P5	0001-01-01 00:00:00.0000000	0001-01-01 00:00:00.0000000	1
24	24	44.86	P4,P3,P1,P2,P4,P5,P3,P1,P4,P2,P3,P4,P1,P5,P2,P3,P1,P2,P3,P5,P1,P2,P1,P5,P2,P1,P2,P5,P1,P2,P1,P5,P1,P5,P5	0001-01-01 00:00:00.0000000	0001-01-01 00:00:00.0000000	1
25	25	44.86	P4,P3,P1,P2,P4,P5,P3,P1,P4,P2,P3,P4,P1,P5,P2,P3,P1,P2,P3,P5,P1,P2,P1,P5,P2,P1,P2,P5,P1,P2,P1,P5,P1,P5,P5	0001-01-01 00:00:00.0000000	0001-01-01 00:00:00.0000000	1
26	26	44.86	P4,P3,P1,P2,P4,P5,P3,P1,P4,P2,P3,P4,P1,P5,P2,P3,P1,P2,P3,P5,P1,P2,P1,P5,P2,P1,P2,P5,P1,P2,P1,P5,P1,P5,P5	0001-01-01 00:00:00.0000000	0001-01-01 00:00:00.0000000	1
27	27	44.86	P4,P3,P1,P2,P4,P5,P3,P1,P4,P2,P3,P4,P1,P5,P2,P3,P1,P2,P3,P5,P1,P2,P1,P5,P2,P1,P2,P5,P1,P2,P1,P5,P1,P5,P5	0001-01-01 00:00:00.0000000	0001-01-01 00:00:00.0000000	1
28	28	44.86	P4,P3,P1,P2,P4,P5,P3,P1,P4,P2,P3,P4,P1,P5,P2,P3,P1,P2,P3,P5,P1,P2,P1,P5,P2,P1,P2,P5,P1,P2,P1,P5,P1,P5,P5	0001-01-01 00:00:00.0000000	0001-01-01 00:00:00.0000000	1
29	29	44.86	P4,P3,P1,P2,P4,P5,P3,P1,P4,P2,P3,P4,P1,P5,P2,P3,P1,P2,P3,P5,P1,P2,P1,P5,P2,P1,P2,P5,P1,P2,P1,P5,P1,P5,P5	0001-01-01 00:00:00.0000000	0001-01-01 00:00:00.0000000	1
30	30	44.86	P4,P3,P1,P2,P4,P5,P3,P1,P4,P2,P3,P4,P1,P5,P2,P3,P1,P2,P3,P5,P1,P2,P1,P5,P2,P1,P2,P5,P1,P2,P1,P5,P1,P5,P5	0001-01-01 00:00:00.0000000	0001-01-01 00:00:00.0000000	1
31	31	44.86	P4,P3,P1,P2,P4,P5,P3,P1,P4,P2,P3,P4,P1,P5,P2,P3,P1,P2,P3,P5,P1,P2,P1,P5,P2,P1,P2,P5,P1,P2,P1,P5,P1,P5,P5	0001-01-01 00:00:00.0000000	0001-01-01 00:00:00.0000000	1
32	32	44.86	P4,P3,P1,P2,P4,P5,P3,P1,P4,P2,P3,P4,P1,P5,P2,P3,P1,P2,P3,P5,P1,P2,P1,P5,P2,P1,P2,P5,P1,P2,P1,P5,P1,P5,P5	0001-01-01 00:00:00.0000000	0001-01-01 00:00:00.0000000	1
33	33	44.86	P4,P3,P1,P2,P4,P5,P3,P1,P4,P2,P3,P4,P1,P5,P2,P3,P1,P2,P3,P5,P1,P2,P1,P5,P2,P1,P2,P5,P1,P2,P1,P5,P1,P5,P5	0001-01-01 00:00:00.0000000	0001-01-01 00:00:00.0000000	1
34	34	44.86	P4,P3,P1,P2,P4,P5,P3,P1,P4,P2,P3,P4,P1,P5,P2,P3,P1,P2,P3,P5,P1,P2,P1,P5,P2,P1,P2,P5,P1,P2,P1,P5,P1,P5,P5	0001-01-01 00:00:00.0000000	0001-01-01 00:00:00.0000000	1
35	35	44.86	P4,P3,P1,P2,P4,P5,P3,P1,P4,P2,P3,P4,P1,P5,P2,P3,P1,P2,P3,P5,P1,P2,P1,P5,P2,P1,P2,P5,P1,P2,P1,P5,P1,P5,P5	0001-01-01 00:00:00.0000000	0001-01-01 00:00:00.0000000	1
36	36	44.86	P4,P3,P1,P2,P4,P5,P3,P1,P4,P2,P3,P4,P1,P5,P2,P3,P1,P2,P3,P5,P1,P2,P1,P5,P2,P1,P2,P5,P1,P2,P1,P5,P1,P5,P5	0001-01-01 00:00:00.0000000	0001-01-01 00:00:00.0000000	1
37	37	44.86	P4,P3,P1,P2,P4,P5,P3,P1,P4,P2,P3,P4,P1,P5,P2,P3,P1,P2,P3,P5,P1,P2,P1,P5,P2,P1,P2,P5,P1,P2,P1,P5,P1,P5,P5	0001-01-01 00:00:00.0000000	0001-01-01 00:00:00.0000000	1

Рис. 2.3. Повна вибірка можливих комбінацій алгоритму Round-Robin

На рис. 2.3 зображена вибірка можливих комбінацій Round-Robin.

Загалом існують дві версії алгоритму Round-Robin, а саме:

- Round-Robin – виконується без будь-якого пріоритету, вважається що пріоритет в усіх вхідних процесів рівний.
- Weighted Round-Robin – виконується з врахуванням того що кожен процес має свій заданий пріоритет.

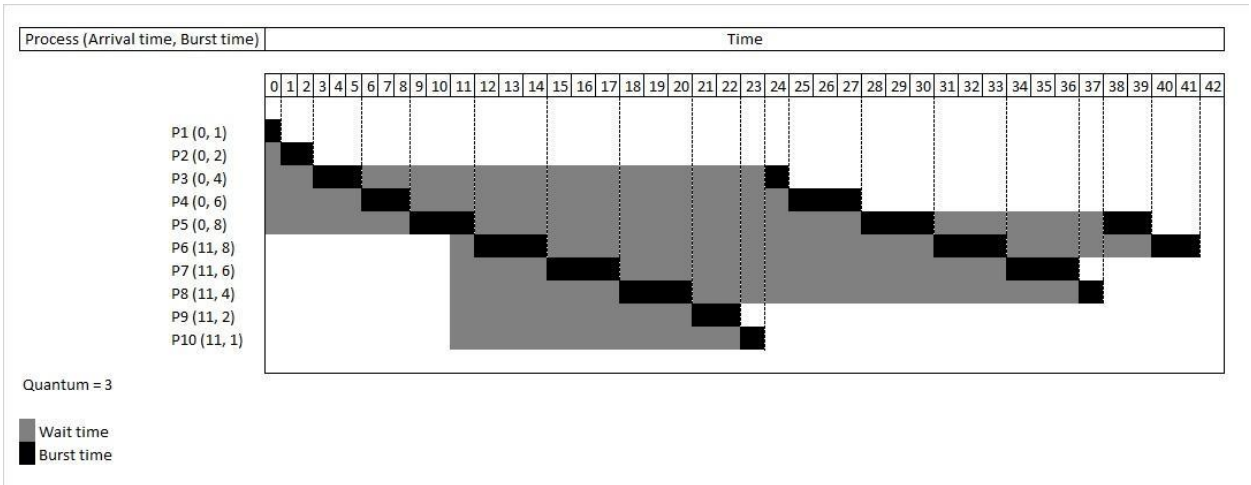


Рис. 2.4. Схема часових інтервалів виконання алгоритму планувальника Round-Robin

Приклад візуального відображення часових проміжків виконання алгоритму Round-Robin можна побачити на рис. 2.4.

2.4. Алгоритм планування процесів на основі хеш-функцій

Хеш-функції – це функції, що можуть довільне значення (набір символів або число) фіксованого або змінного розміру з вхідного набору даних. Вхідний набір даних при проходженні через хеш-функцію перетворюються на хеш-значення. Складність роботи ефективної хеш-функції є O(1). Хеш значення зазвичай використовуються для того щоб розподіляти та індексувати ці значення в хеш-таблицях відповідно до їхніх хешів. Приклад роботи хеш-функції можна побачити на рис. 2.5.

Хеш-функції разом з хеш-таблицями, надають змогу отримання даних з колекції за ключем за сталий проміжок часу (при відсутності колізій). Такі сховища потребують трішки більше місця, оскільки окрім даних, ще зберігають хеш-стрічки ключів.

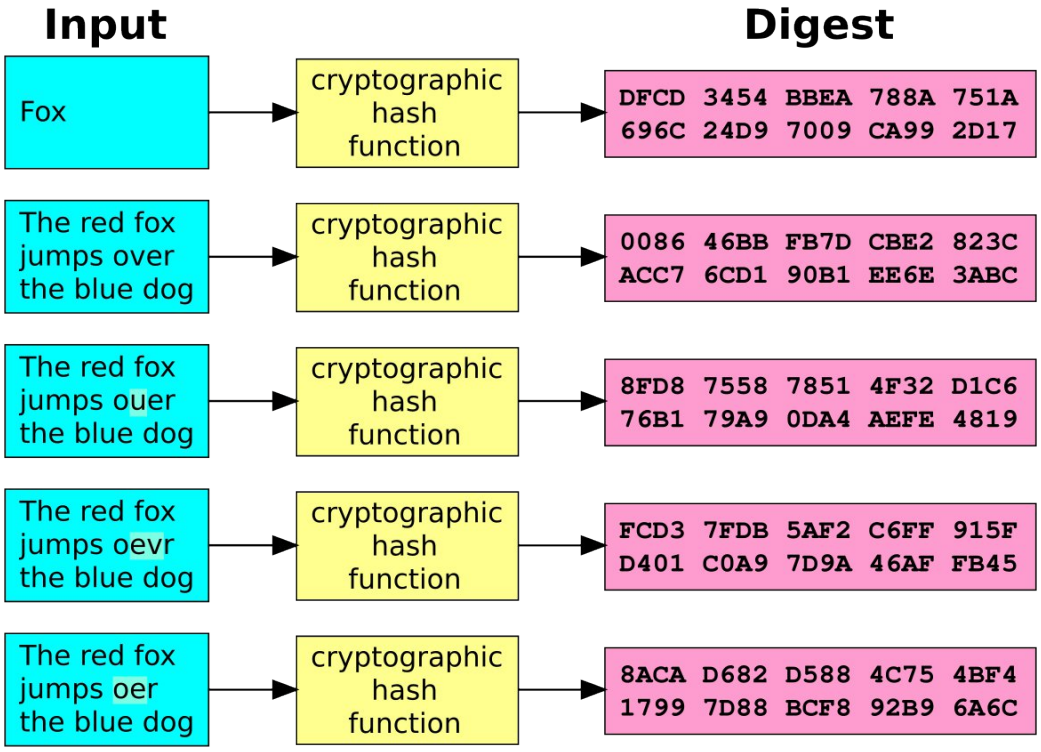


Рис. 2.5. Схема роботи хеш-функції

Хеш-таблиці є ефективним, з точки зору швидкодії, способом збереження та доступу до даних, що дозволяє уникнути залежності часу роботи функції доступу до впорядкованих та невпорядкованих списків або структурованих дерев, від розміру цих списків та інших сторонніх факторів.

У найгіршому випадку хеш-функції, може статись колізія, це випадок коли при різних вхідних значеннях генерується однаковий хеш код. Колізії сприяють погіршенню швидкодії роботи сховищ даних, які використовують хеш-функції не стійкі до колізій, тому сучасні хеш-функції намагаються створювати таким чином, щоб хеш значення було унікальне для кожного вхідного ключа і ніколи не повторювалось. Приклад відтворення випадку колізії, можна побачити на рис. 2.6, у цьому випадку в нас використовується стрічка з іменами, як ключ для генерації хеш стрічки, для імен “Michael” та “Toby” згенерувалося однаковим хеш значення, а саме “2”. Існує два способи вирішувати колізію, такі як:

- Регенерація хеш значення з додаванням певного значення (переважно ціле число).
- Розміщувати елемент для якого відбулась колізія, як наступний елемент зв’язного списку поточного бакету (бакет - англ. контейнер, який містить елемент списку).

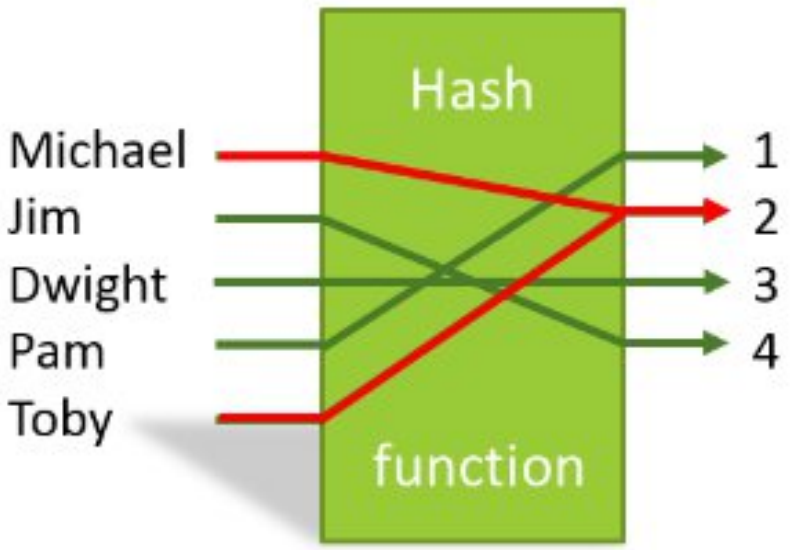


Рис. 2.6. Колізія у хеш-функції

Усі хеш-функції повинні бути криптостійкими. Тобто при отриманні хеш-значень, в жодному разі не повинно бути можливості повернути початкове значення цього хеш коду. Хеш-функції повинні перетворювати вхідні дані в хеш коди, і це перетворення одностороннє. Якщо є можливість отримати початкове значення до застосування хеш-функції, то такі хеш-функції є нерекомендованими до використання.

У досліджуваному планувальнику хеш-функції використовуються для генерування хеш-коду вхідного процесу для планування, щоб мати змогу планувати будь-яку кількість вхідних процесів за сталу одиницю часу.

2.5. Застосування принципу безперервного хешування в сучасних планувальниках

У сфері комп'ютерних наук, принцип безперервного хешування це тип спеціальної техніки хешування, згідно якої при зміні розміру хеш-таблиці, тільки n/m ключів повинні бути перепризначені в середньому, де n – кількість ключів і m – кількість слотів. У більшій кількості традиційних хеш-таблиць, зміна номеру слотів масиву призводить до того, що всі ключі повинні бути перепризначені.

Принцип безперервного хешування використовує модульну арифметику. В математиці, модульна арифметика це система арифметики для цілих чисел, де числа нумеруються по колу, коли досягають певного значення, це число називається модулем. Приклад кола модульної арифметики видно на рис. 2.7.

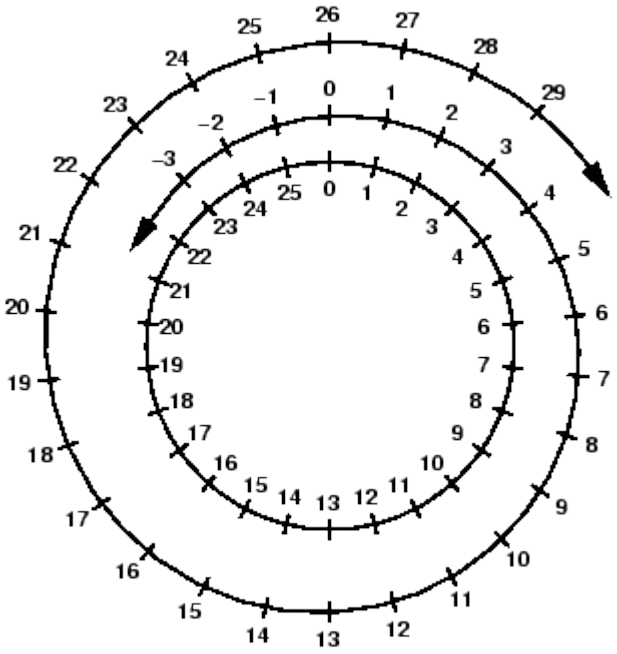


Рис. 2.7. Коло модульної арифметики

У поточному алгоритмі планування як модуль береться максимальне число типу змінної Integer, тобто число 2147483647. Після досягнення цього числа, відлік починається спочатку. Цей принцип модульної арифметики схожий за принципом годинника, що має всього 12 позначень години або та колом, що має всього 360 градусів. Два випадкових цілих числа a та b називають конгруентними модулями m, якщо $a-b=m$.

Підхід безперервного хешування є частим розв′язком проблеми балансування навантаження. Для прикладу, якщо існує певне сховище даних (Blob), яке повинне бути прикріплене до одного з n серверів на кластері. В даному випадку можна застосувати принцип безперервного хешування. Нехай згенерований хеш значення буде рівним β, тоді кількість серверів на кластері, буде рівна n. Тепер нам потрібно виконати модульну операцію з кількістю серверів n, щоб знайти конкретний сервер, на якому буде розміщене сховище даних. У поточному прикладі буде використовуватись коло на 360 градусів. Для того щоб визначити сервер, на який буде розміщене сховище даних, можна використати формулу 2.1.

$$q = \beta \% n \,, \tag{2.1}$$

де q – позиція обчислювальної одиниці в колі розподілу;

β – значення згенерованого хеш значення;

n – кількість обчислювальних одиниць.

Принцип безперервного хешування допомагає уникнути проблем, коли в нас виникає збій роботи одного з серверів, що в кінцевому результаті призводить до повного припинення його роботи. У такому випадку, навантаження та будь-які операції з цим сервером повинні припинитись. А завдання які були прикріплені до сервера, що припинив роботу, повинні перейти до сусіднього сервера. Якщо взяти приклад з колом, то можна припустити що n = 360, тоді для того щоб знайти приналежність до певної частини кола, нам потрібно згенерувати хеш-код IP-адреси запиту прикріплення сховища даних або унікальний ідентифікатор самого сховища. Далі використовуючи формулу 2.1, знайти потрібну частину кола, нехай, значення згенероване хеш-функцією β = 400, у такому випадку q = 400 % 360 = 40. Отже, місце на колі розподілу, в нас q = 40, тепер нам потрібно модулярно рухатись по колу проти годинникової стрілки, для того щоб знайти відповідний сервер, до якого буде прикріплено це сховище даних, для цього зазвичай використовують лінійний алгоритм пошуку зі складністю O(n) або двійковий зі складністю O(log N).

У поточному алгоритмі планування виконання завдань, принцип безперервного хешування використовується для швидкого та ефективного перерозподілу виконуваних процесів конкретної обчислювальної одиниці, в разі її неможливості продовжувати виконувати цю послідовність процесів. При створенні нової обчислювальної одиниці їй присвоюється унікальний ідентифікатор з якого пізніше генерується хеш-код з допомогою хеш-функції. Це хеш-значення використовується як орієнтир на колі модульної арифметики, для розподілення ресурсів обчислювальної одиниці між виконуваними процесами. Коло модульної арифметики алгоритму можна побачити на рисунку 2.8 з трьома обчислювальними одиницями.

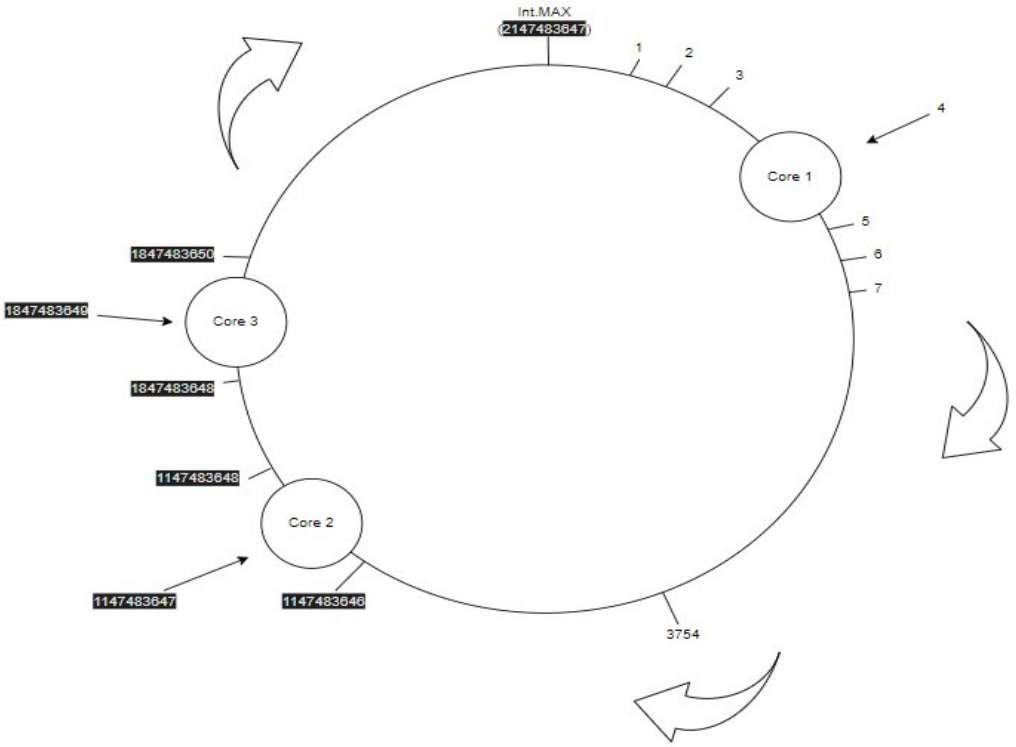


Рис. 2.8. Коло модульної арифметики безперервного хешування

Принцип генерації хеш коду з ідентифікатору процесу був обраний, через те, що алгоритм генерування хеш коду є швидким та виконується за сталу одиницю часу. У досліджуваному алгоритмі планування використовувалась хеш функція, яка генерувала хеш значення у діапазоні від 0 до 2^{32}. На рис. 2.9 зображено візуально принцип роботи функції генерації хеш коду з вхідної стрічки.

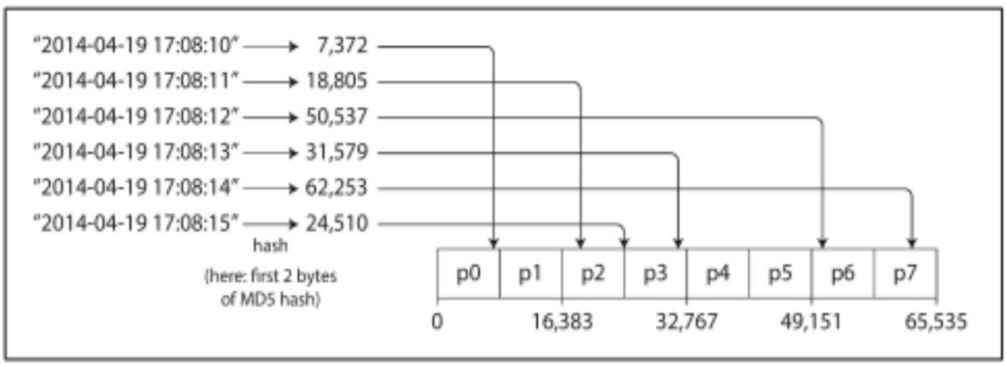


Figure 6-3. Partitioning by hash of key.

Рис. 2.9. Розділення за хеш кодом ключа [1]

На рис. 2.10 показано схему діапазону хеш значень кожної обчислювальної одиниці, за які вона є відповідальною.

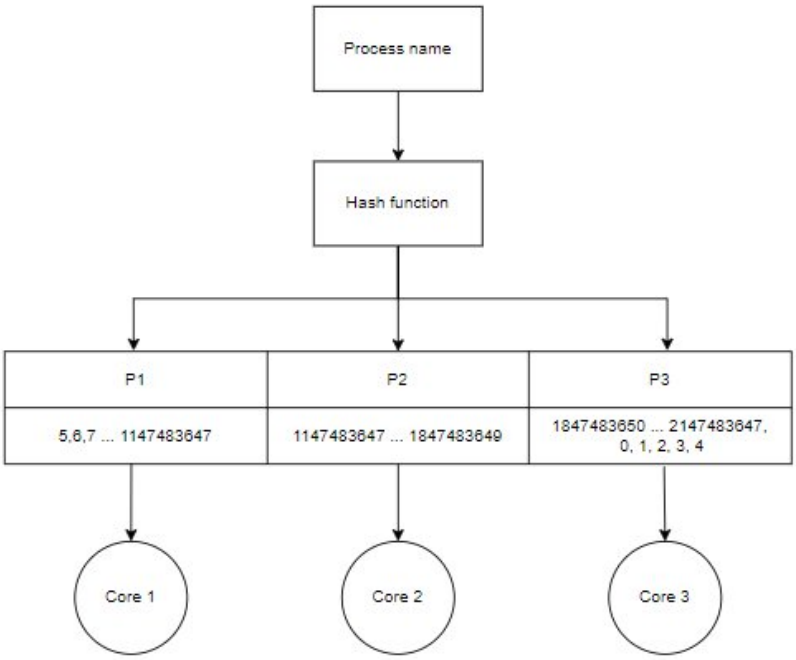


Рис. 2.10. Розділення діапазон відповідальності хеш значень

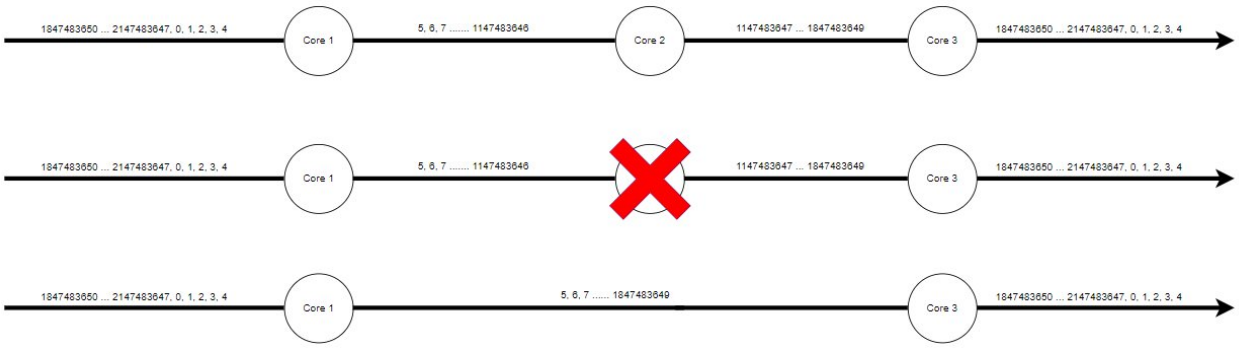


Рис. 2.11. Перерозподіл у випадку неможливості обчислювальної одиниці продовжувати роботу

У поточному прикладі розглянуто планувальник, у якому є три обчислювальні одиниці, а саме R1, R2, R3. Обчислювальна одиниця R1, відповідає за діапазон хеш значень від 5 до 1147483647, R2 – від 1147483647 до 1847483649, R3 – від 1847483650 до 2147483647, 0, 1, 3 4. Тобто коли генерується хеш код вхідного процесу, то за цими діапазонами планувальник визначає, на якій

обчислювальній одиниці відбудеться обробка цього процесу. У випадку переривання або критичної помилки будь-якої обчислювальної одиниці, всі виконувані процеси переносяться на сусідню обчислювальну одиницю, цей випадок продемонстровано на рис. 2.11. Коли обчислювальна одиниця перестає функціонувати, діапазон хеш кодів 1147483647-1847483649 переноситься на обчислювальну одиницю R1, тобто обчислювальна одиниця R1 стає відповідальною за діапазон хеш значень від 5 до 1847483649.

2.6. Теорія систем масового обслуговування

Для того, щоб ефективно організувати чергу збереження завдань та їх обробку у певному порядку в планувальнику, було використано принцип роботи системи масового обслуговування. Теорія системи масового обслуговування, повинна пришвидшити та підвищити продуктивність роботи системи планування обчислювальних завдань.

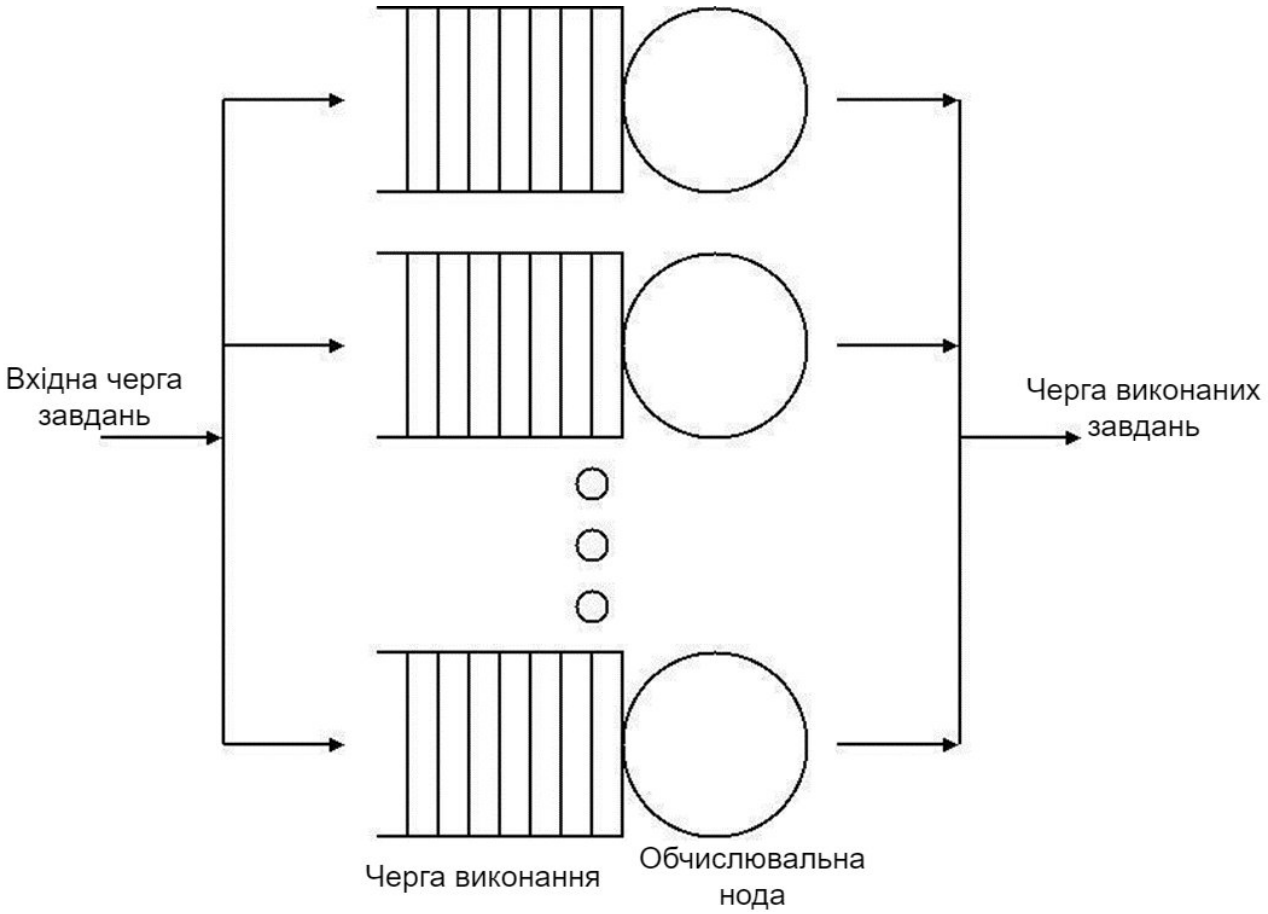


Рис. 2.12. Модель теорії масового обслуговування

Теорія масового обслуговування це математичний напрям дослідження черг очікування. Черги повинні бути спроєктовані таким чином, щоб час очікування був передбачуваним. Приклад схематичного зображення моделі масового обслуговування можна побачити на рис. 2.12. Дана теорія вважається розділом дисципліни дослідження операцій, оскільки результати часто використовуються при прийнятті бізнес рішень про необхідну кількість ресурсів для надання певного сервісу.

Теорія масового обслуговування є великою областю дослідження науки управління. З допомогою науки управління, різні організації та установи можуть вирішувати бізнес-проблеми, використовуючи математичні та наукові підходи. Аналіз масового обслуговування є ймовірнісним аналізом черг, тому всі результати є також ймовірнісними, а не детермінованими. Ймовірність що n елементів буде в черзі в певний момент часу, середня кількість елементів в усіх чергах, середня кількість елементів в черзі очікування і ймовірність виходу з ладу певного обробника, це є характеристики, які модель масового обслуговування здатна обчислювати. Загальна мета аналізу черг, це обраховувати попередньо згадані характеристики для поточної системи та спробувати створити декілька альтернатив інших підходів розподілення елементів між чергами для знаходження способів покращення. Дослідники можуть зібрати обрахунки декількох альтернативних способів планування та на основі статистики та аналізу результатів обрати оптимальний спосіб для виконання конкретного завдання. При обранні оптимального шляху, організація матиме змогу зменшити витрати, зменшити час очікування та покращити ефективність.

Існує два типи систем масового обслуговування за кількістю черг, а саме:

- Одночергова система.
- Система з багатьма чергами.

Також ці системи поділяються на два типи за часом обслуговування, це:

- Система з постійним часом.
- Система з невизначеним часом.

2.7. Часова складність планування обчислювальних завдань

У процесі проведення дослідження, було детально досліджено залежність збільшення кількості операцій (навантаження обчислювальних одиниць) та збільшення пам’яті, від збільшення кількості процесів і збільшення кількості обчислювальних одиниць системи планування. Було проведено дослідження можливості створення алгоритму зі складністю O(1) планування процесу виконання, тобто час виконання має бути сталий, у незалежності від кількості попередньо запланованих процесів.

2.8. Особливості реалізації планувальника завдань в мові програмування C# платформи .NET

Розглянемо, як реалізовано планувальник завдань платформи .NET з урахуванням сутностей мови C#. Для зменшення складності доступу роботи з фоновими завданнями на мові програмування C# був створений спеціальний тип, під назвою Task. Цей клас інкапсулює логіку доступу до фонових потоків, він включає логіку доступу, планування чи скасування виконання завдання.

Для того щоб координувати роботу декількох завдань, був створений додатковий клас TaskScheduler. У цьому класі реалізована логіка виконання декількох завдань типу Task, є можливість задання певного пріоритету для кожної виконуваної задачі, а також видалити певні завдання, якщо їхнє виконання більше не потрібне. Всі завдання зберігаються в черзі виконання та виконуються за принципом FIFO якщо всі мають однаковий пріоритет, проте якщо поточне завдання має більший пріоритет ніж завдання яке знаходиться перед ним, то це завдання буде виконане швидше. У планувальнику також реалізована логіка скасування виконуваної задачі, це реалізовано з використанням класу CancellationToken, який створюється класом CancellationTokenSource. Екземпляр цього класу сповіщає фонове завдання, що його виконання більше не потрібне і

це завдання викликає виняток, який називається TaskCancelledException, що призводить до завершення його виконання.

2.9. Висновки до розділу

У даному розділі було детально досліджено всі технології та інструменти, які були використані під час розробки планувальника обчислювальних завдань. Було проведено аналіз наявних планувальників сучасних операційних систем. Проаналізовано алгоритми Round-Robin та Weighted Round-Robin, обґрунтовано доцільність використання хеш функцій та алгоритму безперервного хешування в сучасних планувальниках.

РОЗДІЛ 3
ПРАКТИЧНА ЧАСТИНА

3.1. Структура системи планування

Досліджувана система базується на використанні зазначеного вище алгоритму планування та здійснює керуванням виконання обчислювальних процесів, використовуючи вказані налаштування та вказану кількість обчислювальних одиниць.

Досліджувана система складається з двох частин, а саме:

- Модуль планування послідовних процесів.
- Модуль балансування та планування паралельних процесів.

Основним завданням модуля планування послідовних процесів є прийняття послідовності процесів з певною інформацією, аналізу цих процесів та генерації коефіцієнта ефективності для кожної комбінації послідовного виконання цих процесів. Коли коефіцієнти згенеровані, користувач отримує на виході послідовність, що має найбільший коефіцієнт.

Модуль балансування та планування паралельних процесів має складнішу структуру в порівнянні з попереднім модулем. У цьому модулі реалізована логіка планування та балансування навантаження між послідовністю процесів та, заданими попередньо, обчислювальними одиницями. Для роботи з цим модулем ми повинні створити певну послідовність обчислювальних одиниць. Потім ми повинні відправити в систему планування процес, цей процес буде автоматично напрямлений на виконання до конкретної обчислювальної одиниці.

3.2. Опис системи, яку використовує планувальник

На поточному етапі розробки та дослідження було створено систему планування виконання обчислювальних завдань використовуючи процеси операційної системи Windows як обчислювальні одиниці. Коли створюється запит

на створення обчислювальної одиниці, система планування ініціює екземпляр windows процесу та запускає його виконання та створює об'єкт в пам’яті для його виконання.

Windows-процес є екземпляром виконуваної програми, який виконується на операційній системі. Процес як виконуваний екземпляр включає виконуваний код, дані, системні ресурси, які необхідні для виконання. Ці процеси є мультизадачними і можуть керувати виконанням різних задач паралельно. Приклад структури одного такого процесу можна побачити на рис. 3.1.

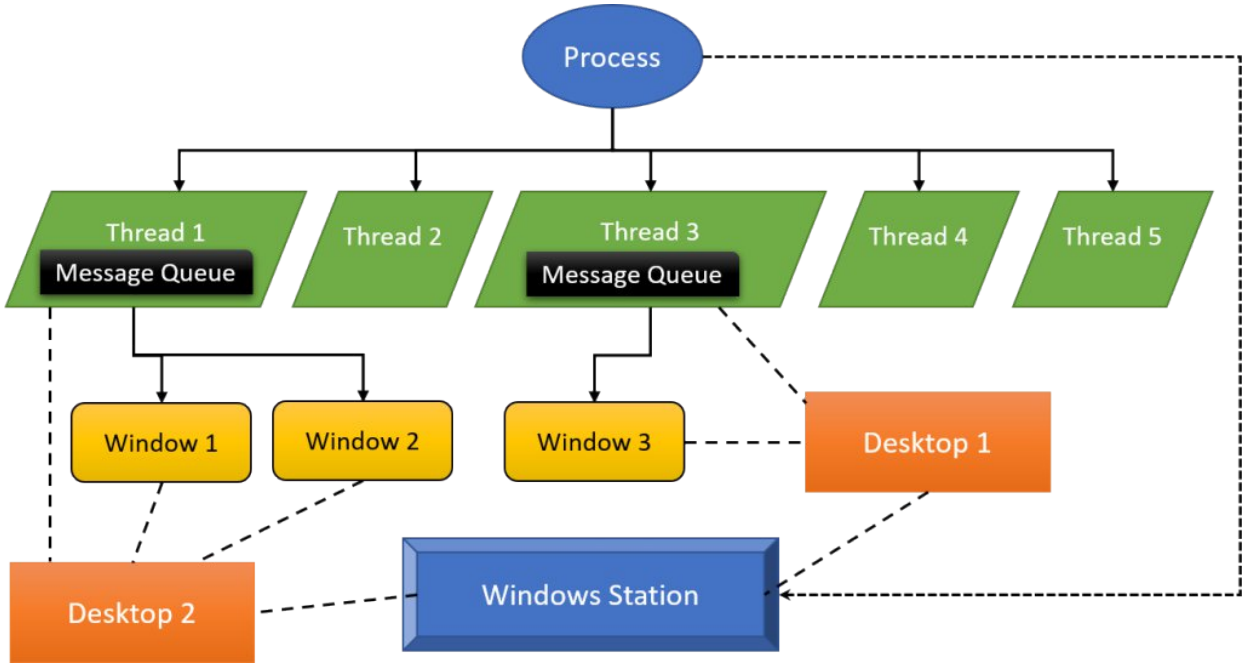


Рис. 3.1. Структура компонентів Windows-процесу

Запуск Windows-процесу відбувався на мові програмування C#. Щоб запустити такий процес нам потрібно знайти скомпільований екземпляр програми з використанням фреймворку ASP .NET Core та запустити його. Скомпільований код лежить в папці bin в папці з програмою, нам потрібно знайти файл з розширенням “.dll” та виконати його як новий Windows процес, вказавши вільний порт при його запуску. Щоб переконатись що процес запущений, нам потрібно перейти в меню “Services” та переконатись що новий процес був створений та перебуває в статусі “Running”, якщо процесу немає, це означає що він завершив своє виконання з певною помилкою.

3.3. Предметна область та модель системи планування

Предметна область комп’ютерної системи планування виконання процесів спроєктована таким чином, щоб забезпечити максимальну гнучкість її застосування до різних комп’ютерних та інших обчислювальних систем. Схематичне представлення предметної області системи можна побачити на рис. 3.2.

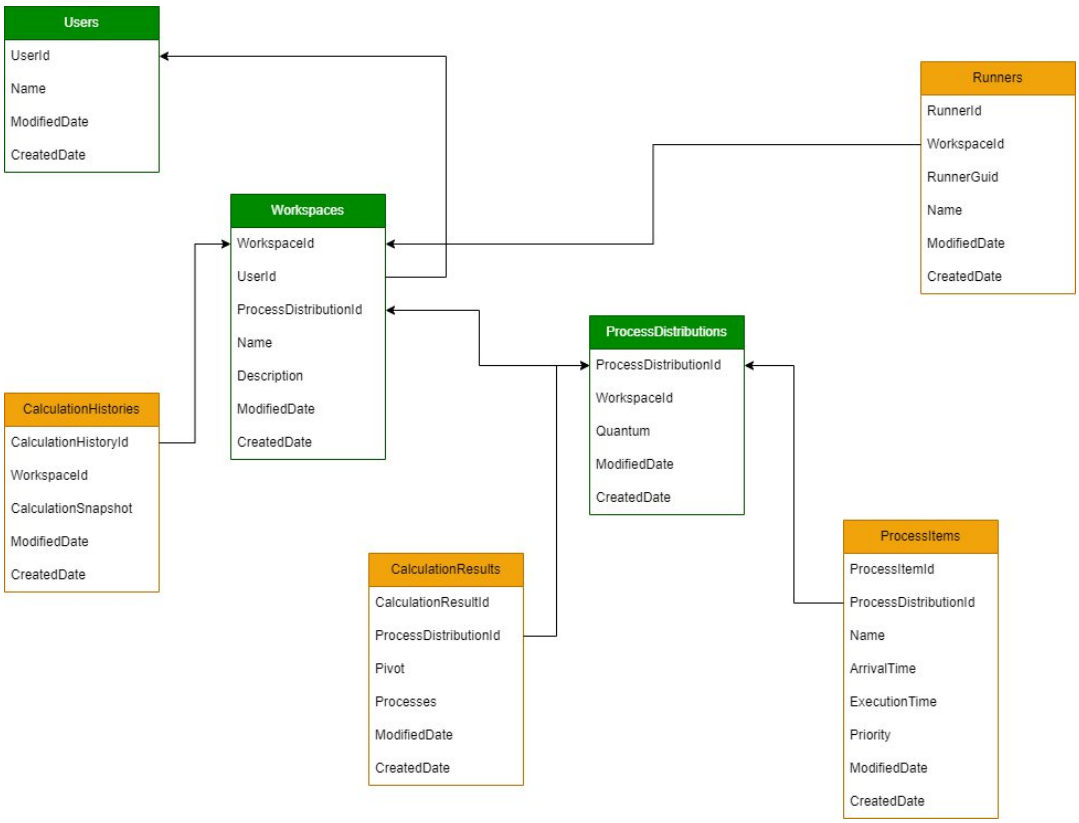


Рис. 3.2. Схема бази даних предметної області

Основними компонентами системи планування обчислювальних завдань є наступні компоненти:

- Users.
- Workspaces.
- CalculationHistories.
- ProcessDistributions.
- CalculationResults.

- Runners.
- ProcessItems.

Система була спроектована таким чином, щоб алгоритм планування обчислювальних завдань міг бути застосованим у будь-якій системі з мінімальною кількістю змін, це може бути як операційна система, так і система планування конкретного додатку.

Сутність Users використовується для розділення доступу до конкретної кількості сутностей Workspace для організації планування. Для того щоб можна було розпочати роботу з системою планування потрібно створити сутність Workspace. У випадку необхідності проведення аналізу ефективності виконання певної послідовності процесів з певним пріоритетом нам потрібно створити сутність ProcessDistribution. Зв'язок між сутностями Workspace та ProcessDistribution є один до одного, тобто для одного Workspace може бути закріплений тільки один екземпляр ProcessDistribution. Сутність ProcessItem потрібна для того щоб мати послідовність об'єктів на основі яких і буде проведений аналіз для визначення найоптимальнішої послідовності виконання. При створенні кожного ProcessItem ми повинні вказати певні критерії для аналізу, а саме:

- Ім'я.
- Час прибуття.
- Час виконання.
- Пріоритет.

Сутності CalculationResult та CalculationHistory потрібні для зберігання результатів визначення найоптимальнішої послідовності виконання та збереження результату попередніх операцій відповідно.

Сутність Runners потрібна для роботи іншого модуля системи, а саме планування та балансування процесів для паралельного виконання між певними обчислювальними одиницями. Ця сутність необхідна для збереження стану та інформації про всі працюючі обчислювальні одиниці, щоб у випадку збою роботи

планувальника, система мала змогу повернути всі активні обчислювальні одиниці назад, без необхідності їхнього ручного перестворення.

Коли сутність Workspace створена, то для неї створюється спеціальний екземпляр класу RunnerOrchestrator який потрапляє в клас OrchestratorManager, який завжди живе в пам’яті й відповідає патерну Singleton. Для того щоб переконатись, що RunnerScheduler завжди присутній для кожного робочого місця, у системі завжди запущений фоновий потік який з інтервалом у 10 секунд перевіряє чи кожне робоче місце має відповідний планувальник в пам’яті, якщо ні, то створює новий та відповідно якщо робоче місце видалене, то видаляє відповідний планувальник з пам’яті. Приклад схематичного представлення роботи маніпулятора планувальниками типу OrchestratorManager можна побачити на рис. 3.3.

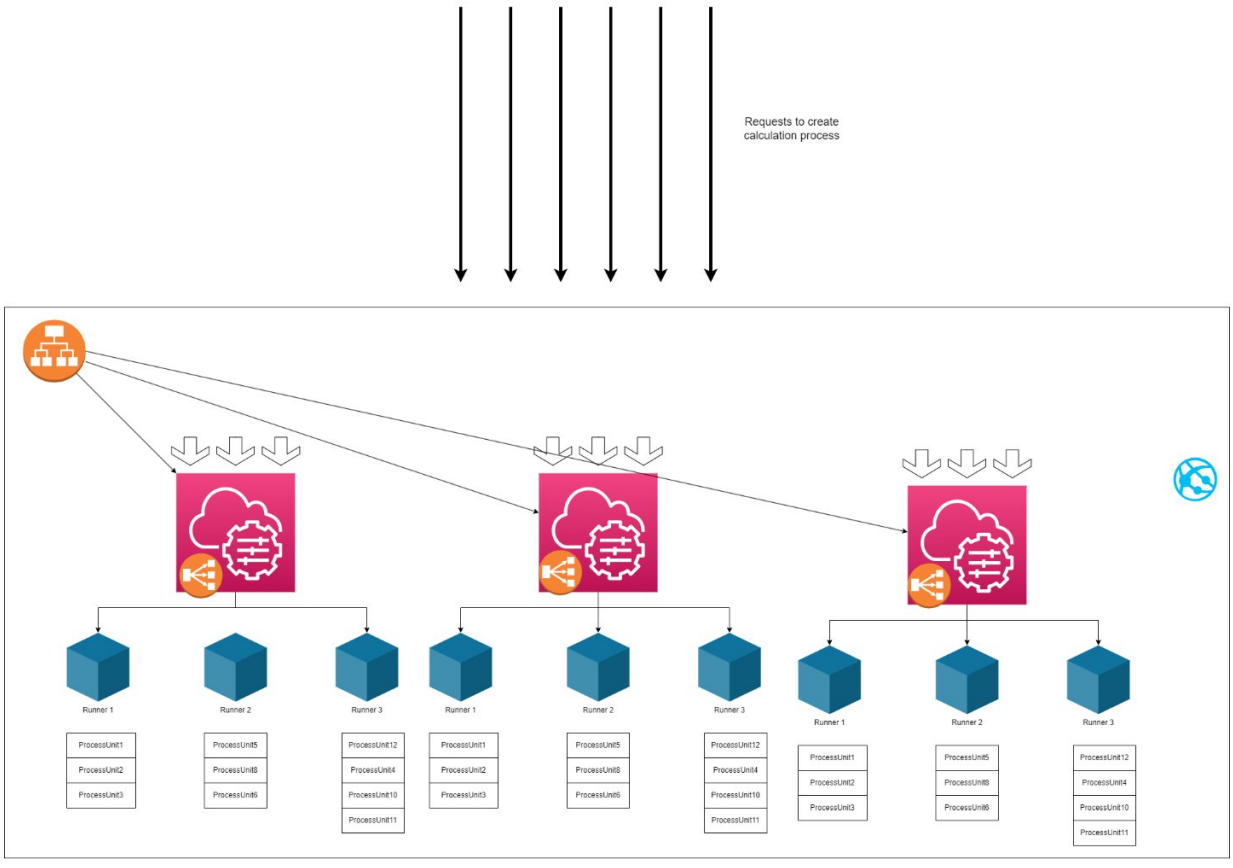


Рис. 3.3. Схема роботи маніпулятора планувальниками системи

Кожен з створених планувальників містить певні обчислювальні одиниці Runners, кожна з який може виконувати певну кількість обчислювальних завдань. Для того щоб зберігати чергу виконуваних завдань, всі процеси які закріпляються до конкретної обчислювальної одиниці розміщуються у черзі виконання. Ці процеси у черзі виконуються за алгоритмом Weighted Round-Robin, згідно цього алгоритму всі процеси виконуються рівномірно розподіляючи час обчислювальної одиниці за пріоритетом. Вибір відповідної обчислювальної одиниці вибирається за хеш-значенням вхідного процесу. Якщо створюються гарячі точки (hotspots), система автоматично запускає алгоритм перебалансування, який повинен перерозподілити завдання між обчислювальними одиницями, для того щоб вирівняти навантаження. Приклад виконання процесів одної обчислювальної одиниці можна бачити на рис. 3.4.

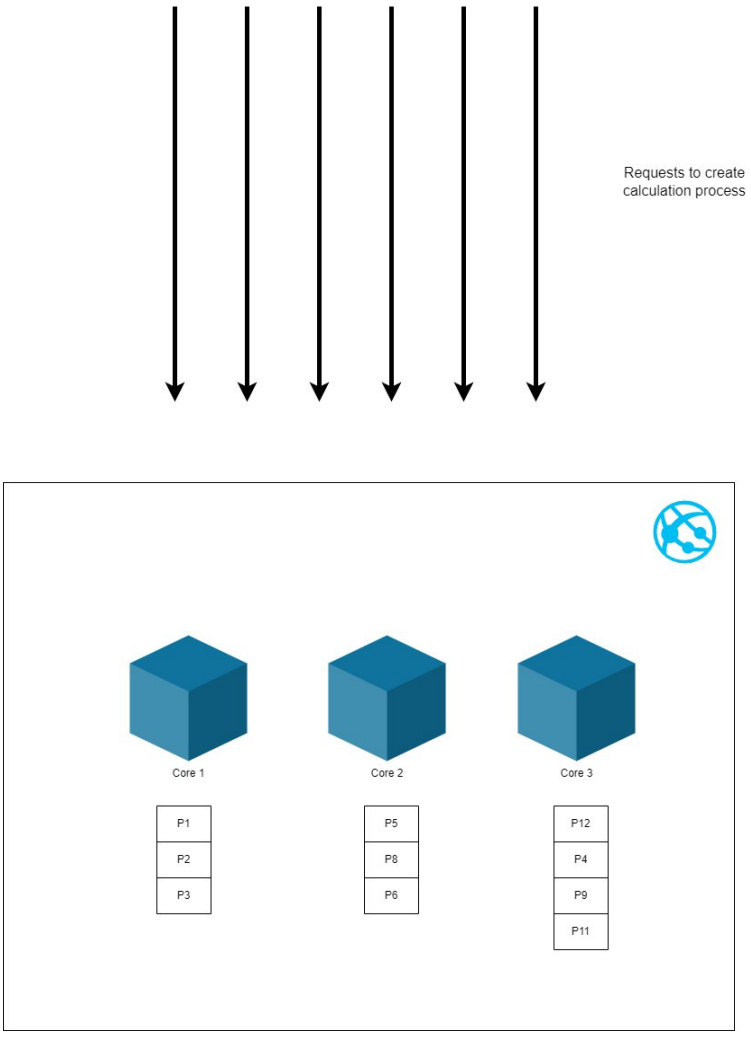


Рис. 3.4. Схема роботи конкретного планувальника

3.4. Алгоритмічна складова логіки роботи основних модулів

В системі використовувався алгоритм Weighted Round-Robin. Блок-схему алгоритму роботи цього алгоритму можна побачити на рис. 3.5.

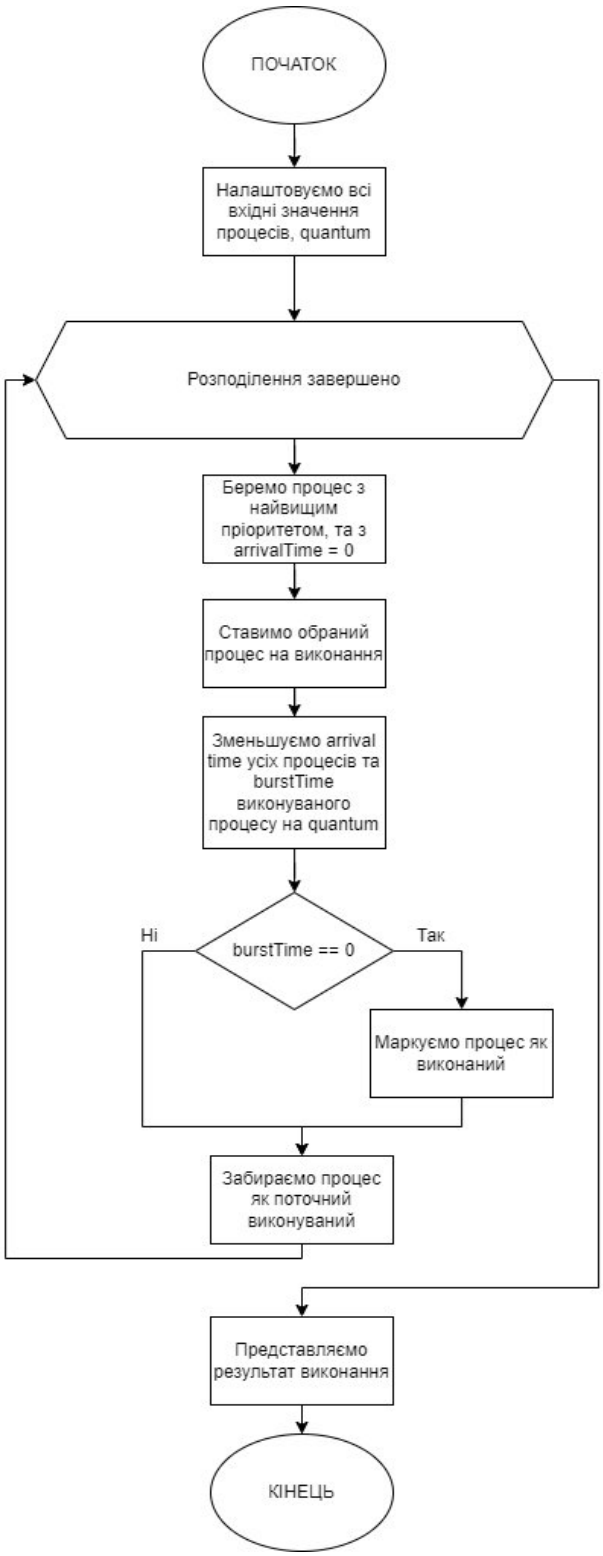


Рис. 3.5. Блок-схема алгоритму послідовного планування

Цей алгоритм використовувався для планування послідовного виконання процесів, а також проведення аналізу можливих комбінацій ефективного виконання алгоритму використовувався алгоритм Weighted Round-Robin.

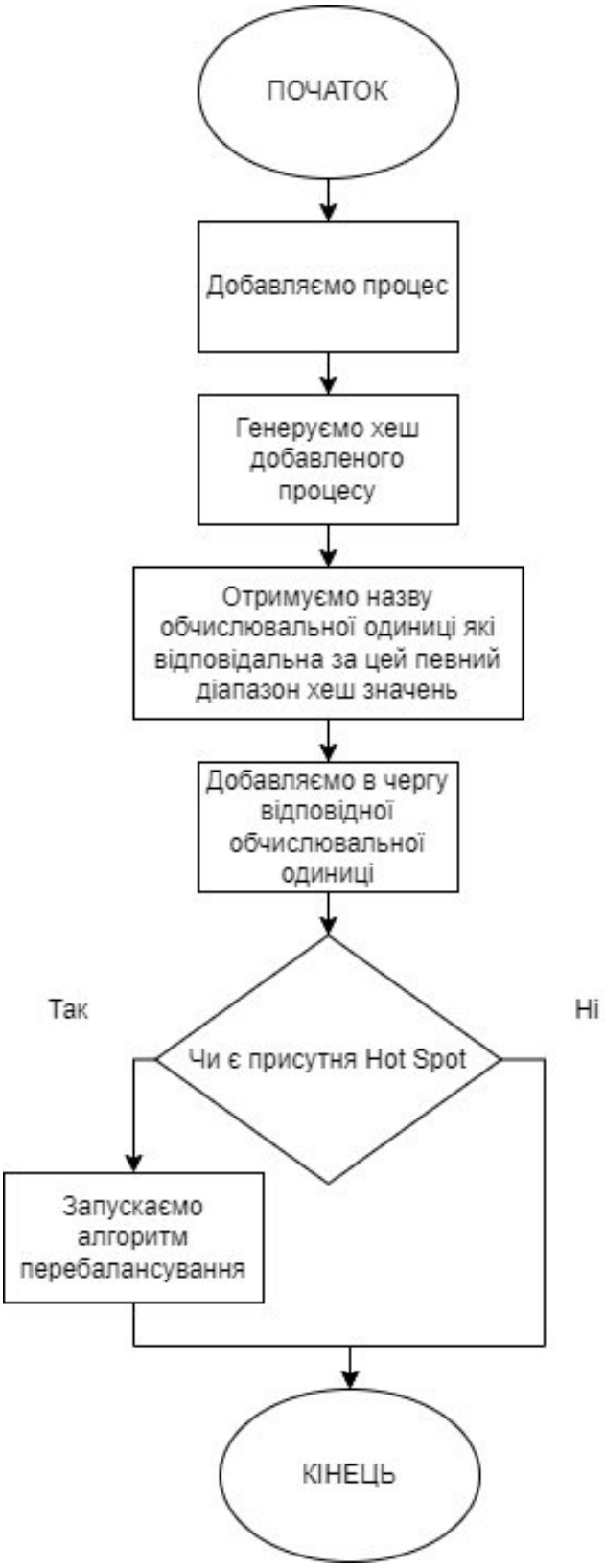


Рис. 3.6. Блок-схема алгоритму добавлення нового процесу на виконання

Кожен процес системи повинен бути доданий окремо, блок-схему алгоритму додавання нового процесу, можна побачити на рис. 3.6. Під час додавання в чергу спочатку генерується хеш-код вхідного процесу на основі назви цього ж процесу. Хеш-значення це ціле число в діапазоні від 0 до 2^{32}.

Коли хеш-значення згенероване, система автоматично обирає відповідну обчислювальну одиницю за принципом “Безперервного хешування”. Коли обчислювальна одиниця обрана, ми додаємо процес у чергу цієї обчислювальної одиниці на виконання. У кінці ми проводимо перевірку на наявність гарячих точок (hotspots) у системи, якщо такі точки присутні, ми проводимо операцію ребалансуванння.

У системі також існує можливість запуску алгоритму перебалансування процесів вручну. Тобто коли гарячих точок не помічено, проте поточне розподілення процесів між обчислювальними одиницями не є задовільним. Блок-схему алгоритму ручного перебалансування можна побачити на рисунку 3.7.

Рис. 3.7. Блок-схема запуску алгоритму ребалансування

Блок-схема алгоритму визначення найефективнішої послідовності видно на рисунку 3.8.

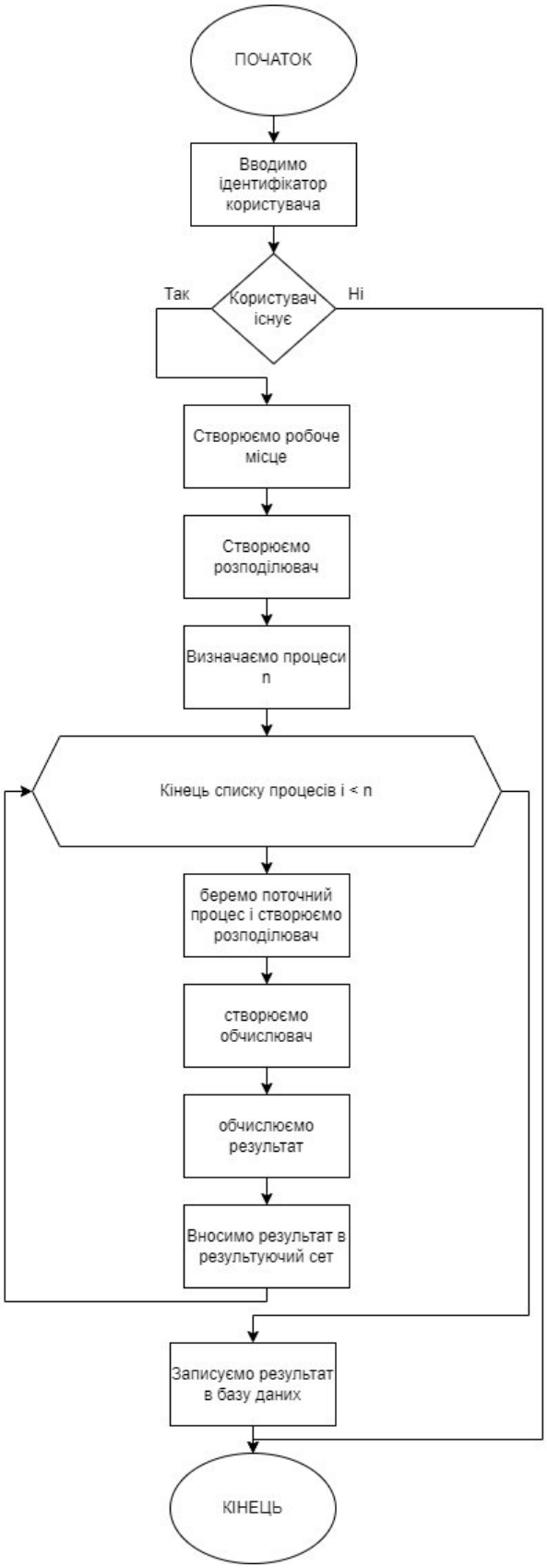


Рис. 3.8. Блок-схема алгоритму визначення найефективнішої послідовності

3.5. Програмне забезпечення основних модулів

Система складається з двох модулів, у першому модулі відбувається реалізована логіка, яка дає можливість користувачу вказати на вхід певну послідовність процесів, система проаналізує цю послідовність, запустить алгоритм визначення найоптимальнішої послідовності.

Другий модуль використовується для організації планування паралельно виконуваних обчислювальних завдань. Тут є можливість задання певних обчислювальних одиниць, система сформує певні діапазони хеш-значень за які кожна обчислювальна одиниця буде відповідальною. Коли користувач або система задають певні процеси на вхід, система планування повинна автоматично визначити, яка обчислювальна одиниця буде виконувати цей процес.

3.5.1 Визначення оптимальної послідовності виконання процесів. Модуль планування виконання послідовних процесів був реалізований на основі відомого алгоритму Weighted Round-Robin, у лістингу 3.1 можна побачити реалізацію цього алгоритму на мові програмування C#.

```
        public IReadOnlyCollection<string> Distribute()
        {
            Queue<string> resultiveFlow = new Queue<string>();
            int t = 0;
            while (!IsDistributionCempleted())
            {
                var                readyProcesses                =
_processes.GetReadyUnfinishedOrderedProcesses()
                    .Except(_currentProcess)
                    .Except(_processReadyQueue);
                foreach (var readyProcess in readyProcesses)
                {
                    readyProcess!.Reset();
    readyProcess.UpdateEstimatedExecution(_quantum);
                    _processReadyQueue.Enqueue(readyProcess);
                }
                if (IsExecutionCurrentCyclePassed())
                {
                    if (IsFullyFinished(_currentProcess!))
                    {
    _fullyFinishedProcesses.Add(_currentProcess!.Name);
                    }
                    _currentProcess = default;
                }
```

```
                bool isNextProcess = _processReadyQueue.TryPeek(out
_);
                if (_currentProcess is null && isNextProcess)
                {
                    _currentProcess = _processReadyQueue.Dequeue();
                    _currentProcess.PrepareForExecution(_quantum);
                    resultiveFlow.Enqueue(_currentProcess.Name);
                }
                else if (_currentProcess is null && !isNextProcess)
                {
                    MoveWaitingProcessesArrivalTimeStepForward();
                    continue;
                }
                _currentProcess!.Execute(_iterationStep);
                MoveWaitingProcessesArrivalTimeStepForward();
            }
            return resultiveFlow;
        }
        private void MoveWaitingProcessesArrivalTimeStepForward()
        {
            var waitingProcesses = _processes.Select(x => x.Value)
                .Except(_currentProcess)
                .Except(_processReadyQueue);
            foreach (var process in waitingProcesses)
            {
                process!.Wait(_iterationStep);
            }

        }
```

Рис. 3.9. Лістинг коду планування послідовності процесів

Як видно на рис. 3.9, реалізований метод Distribute який викликається при необхідності згенерувати послідовність виконуваних процесів за досліджуваним алгоритмом. У нас виконується цикл While, у тілі якого система визначає виконуваний процес, ним стає той значення ArrivalTime якого є найнижчим. При кожному виконанні циклу ArrivalTime кількість всіх невиконуваних процесів зменшується на задаче число quntum, що являє собою час одного такну обчислювальної одиниці, для виконуваного процесу зменшується число BurstTime. Коли ArrivalTime та BurstTime всіх процесів послідовності рівні 0, то метод IsDistributionCompleted повертає true, що викликає завершення виконання алгоритму.

3.5.2 Планування вхідного потоку процесів між паралельними обчислювальними одиницями. Модуль планування та розподілення вхідного потоку процесів призначений для розподілення навантаження між обчислювальними одиницями. Коли процес повинен бути запланований, планувальник генерує хеш значення, та на його основі визначає, яка обчислювальна одиниця обробляє цей діапазон вхідних хеш значень

```
    public void Rebalance(List<string> coreIds)
    {
        if (!coreIds.Any())
        {
            _balancingRangeUnits.Clear();
            return;
        }
        var balancingUnitsCopy = _balancingRangeUnits.DeepClone();
        try
        {
            _balancingRangeUnits.Clear();

            var coresNumber = coreIds.Count;

            var processAssignmentRange = MaxHashValue / coresNumber;

            var startPoint = MinHashValue;

            for (int i = 0; i < coreIds.Count; i++)
            {
                var balancingUnit = new BalancingRangeUnit(
                    hashFrom: startPoint + 1,
                    hashTo: startPoint + processAssignmentRange,
                    runnerId: coreIds[i]);
                _balancingRangeUnits.Add(balancingUnit);
                startPoint += processAssignmentRange;
            }
        }
        catch (Exception ex)
        {
         _logger.LogError($"Exception happened during Rebalancing.
Exception message: {ex.Message}");
            _balancingRangeUnits = balancingUnitsCopy;
        }
    }
    public string GetRunnerId(int processHash)
    {
        var runnerRangeUnit = _balancingRangeUnits
            .SingleOrDefault(x => processHash.IsInRange(x.hashFrom,
x.hashTo));
        if (runnerRangeUnit is null)
```

```
        {
            throw new Exception("Balancing item does not exist !!");
        }
        return runnerRangeUnit.runnerId;
    }
```

Рис. 3.10. Лістинг коду планування алгоритму балансування навантаження між обчислювальними одиницями

3.6. Тестування роботи планувальника

Система планування виконання послідовних процесів реалізована у вигляді програми, яка працює на операційній системі Windows 10. Для керування системою використовується інструмент swagger, вікно якого можна побачити на рис. 3.11.

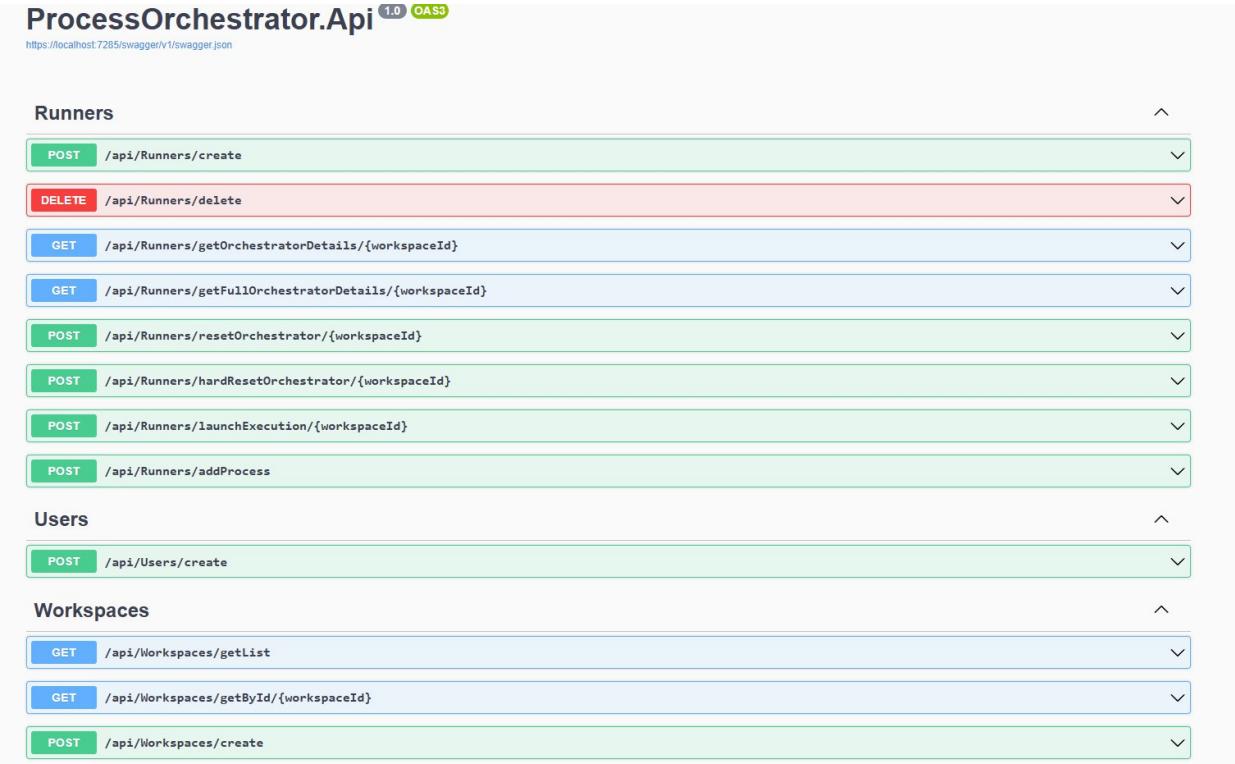


Рис. 3.11. Вікно Swagger

Оскільки, в системі передбачено можливість керування різними обчислювальними контекстами, це було зроблено через сутність Workspace яка кріпиться до кожного користувача окремо. Кожен користувач може мати

необмежену кількість робочих місць. Кожне робоче місце має свій власний планувальник, який живе як Singleton об’єкт в пам’яті системи. Також існує потік, який виконується в циклі кожні 10 секунд та перевіряє чи кожен Workspace має відповідного планувальника в пам’яті, якщо ні, то створює його.

Для того щоб отримати повну інформацію про планувальник, там потрібно виконати запит на кінцеву точку “getFullOrchestratorDetails”. Приклад виконання такого запиту можна побачити на рис. 3.10. Планувальник на рис. 3.12 ще не виконувався, він немає виконаних процесів та обчислювальних одиниць.

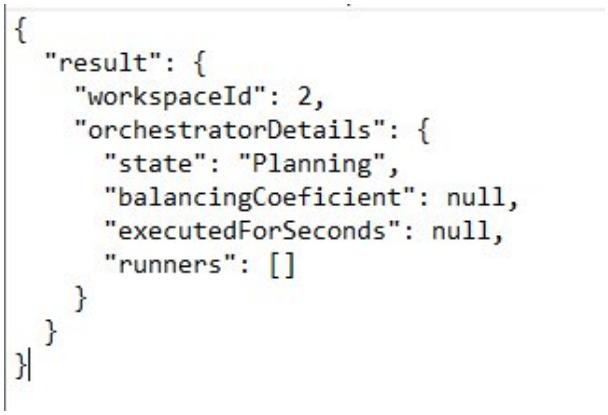


Рис. 3.12. Результат виконання запиту “getFullOrchestratorDetails”

Для визначення ефективності розподілення використовується обрахунок середньоквадратичного відхилення кількості процесів на кількість обчислювальних одиниць. Для обчислення середньої кількості процесів на одній обчислювальній одиниці потрібно використати формулу:

$$avg = \frac{n_p}{n_r} \quad , \tag{3.1}$$

де n_p – кількість процесів;

n_r – кількість обчислювальних одиниць;

Для обчислення середньоквадратичного відхилення використовується формула:

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n} (x_i - avg)^2}{n}}, \tag{3.2}$$

де x_i – кількість процесів обчислювальної одиниці;

avg – середня кількість процесів на обчислювальній одиниці;

n – кількість обчислювальних одиниць.


```
{
  "result": {
    "workspaceId": 2,
    "orchestratorDetails": {
      "state": "Finished",
      "balancingCoeficient": 3.2727833889689544,
      "executedForSeconds": 6.2372185,
      "runners": [
        {
          "id": "a3044233-25b5-45d9-bcbc-e5256b99469c",
          "name": "R15",
          "isActive": true,
          "balancingRange": {
            "hashFrom": 1,
            "hashTo": 214748364
          },
          "processes": [],
          "finishedProcesses": [],
          "state": "Idle",
          "errorMessage": null
        },
        {
          "id": "751730bc-ebc4-4e89-af5c-aca357abdee7",
          "name": "R17",
          "isActive": true,
          "balancingRange": {
            "hashFrom": 214748365,
            "hashTo": 429496728
          },
          "processes": [],
          "finishedProcesses": [
            {
              "name": "P33",
              "executionTime": 0
            },
            {
              "name": "P34",
              "executionTime": 0
            },
            {
              "name": "P49",
              "executionTime": 0
            },
            {
              "name": "P50",
              "executionTime": 0
            },
            {
              "name": "P53",
              "executionTime": 0
            },
            {
```


Рис. 3.13. Результат виконання запиту “getFullOrchestratorDetails” після калькуляції

На рис. 3.13 зображено приклад виконання випадку коли у нас є 10 обчислювальних одиниць та 30 процесів, які виконувались на цих обчислювальних одиницях. Було автоматично обраховано коефіцієнт правильності розбалансування, коефіцієнт = 3.2727833889689544. Планувальник

має змогу обраховувати час виконання процесів. Для прикладу на рисунку 3.12 було взято 30 процесів, час виконання кожного процеса становить 10 одиниць, припускається що одна одиниця рівна 100 мілісекундам. Час виконання = 6.237 с.

Було проведено дослідження з різними коефіцієнтами розбалансування.

Таблиця 3.1

Результати виконання планувальника з 10 обчислювальними одиницями

Execution time sec	Planning coefficient
5.1838071	2.848001248
6.231885	3.272783389
4.032772	2.48191721
5.1877478	2.917380865
6.2371895	3.242084378
4.1537053	2.985148424
5.2029255	3.018461713
6.2478531	2.985148424
5.1924304	2.951459149
7.2766022	3.242084378

У таблиці 3.1 продемонстровано результати проведення дослідження 10-ти обчислень. Для проведення експерименту потрібно задати певні початкові налаштування, а саме:

- Кількість обчислювальних одиниць.
- Кількість процесів.
- Час виконання одного процесу.

Для поточного експерименту було взято 10 обчислювальних одиниць та 30 процесів, час виконання кожного процесу був встановлений як 10.

Обрахування коефіцієнту планування відбувається шляхом визначення середньоквадратичного відхилення поточної запланованої послідовності. Припускається, що при однакових початкових умовах всіх активних

обчислювальних одиниць та певної кількості процесів з однаковим пріоритетом, процеси повинні бути рівномірно розподіленими між всіма обчислювальними одиницями, тобто чим більший коефіцієнт розподілення, тим гірше розподілення відбулось.

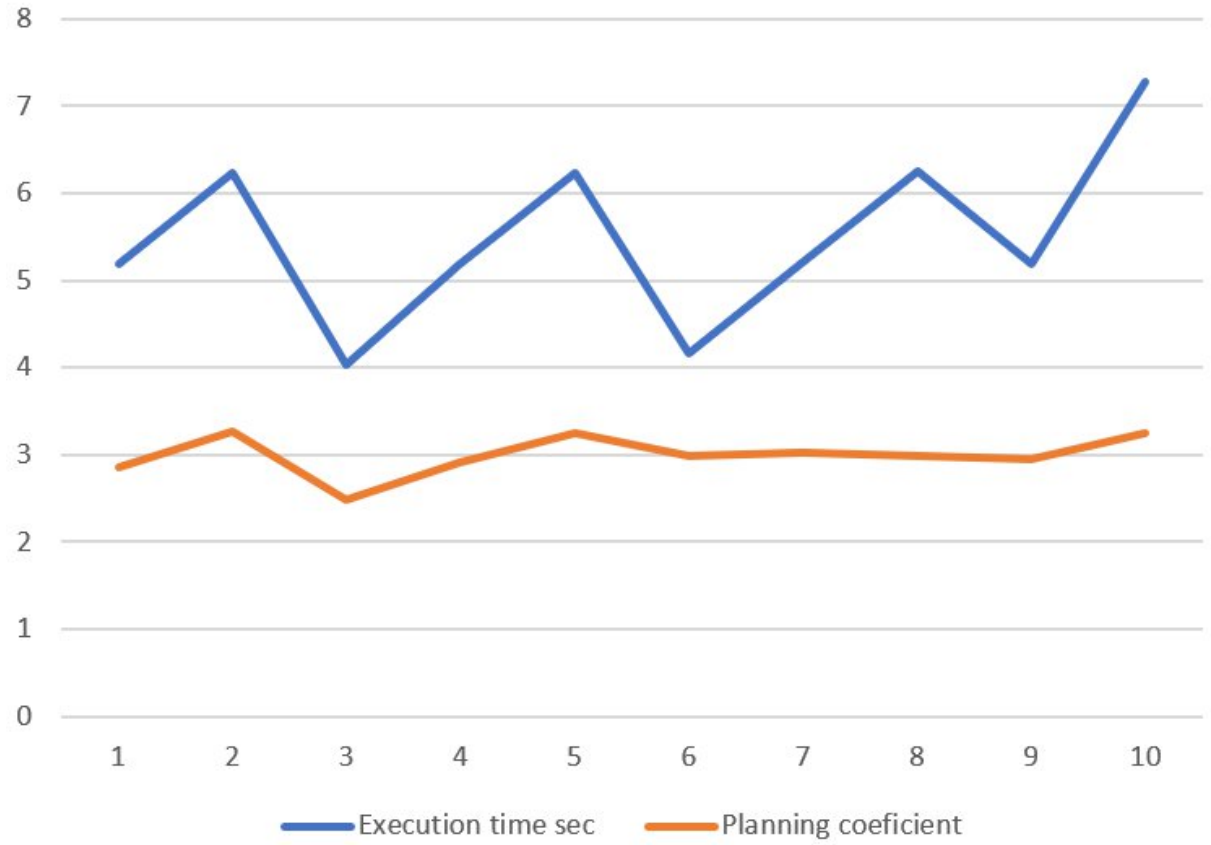


Рис. 3.14. Експеримент з 10 обчислювальних одиниць

Графік зміни часу виконання та коефіцієнту розбалансування можна побачити на рисунку 3.14. Після проведення дослідження було виявлено, що при наближенні до створення гарячих точок в системі, сильно зростає коефіцієнт балансування. При ідеальному випадку розбалансування вхідного потоку процесів між обчислювальними одиницями, коефіцієнт балансування повинен бути рівний 0. Чим більше цей коефіцієнт відхиляється від 0, тим неправильніше відбувся розподіл процесів між обчислювальними одиницями. Є випадки, коли неможливо досягнути ідеального випадку розбалансування, наприклад коли в нас є 3 обчислювальні одиниці та 7 процесів, одна обчислювальна одиниця точно буде

виконувати більше завдань на 1 ніж інші, в ідеальному випадку. Тому потрібно лише намагатись зменшити цей коефіцієнт.

Таблиця 3.2

Результати виконання планувальника з 5 обчислювальними одиницями

Execution time sec	Planning coefficient
7.2559988	5.901506399
7.3161862	5.9352993
8.2865888	6.068589439
10.3741541	6.513660858
8.305612	5.968900885
7.2561251	5.867518877
5.1846922	6.665416549
9.3227913	6.605132684
8.2904059	6.068589439
7.2719326	5.9352993

У таблицю 3.2 внесено результати виконання планування з 5 обчислювальними одиницями.

Рис. 3.15. Експеримент з 5 обчислювальних одиниць

На рисунку 3.15 зображено графік результатів виконання планування обчислення з 30 процесами та 5 обчислювальними одиницями.

Таблиця 3.3

Результати виконання планувальника з різною кількістю обчислювальних одиниць

Execution time sec	Planning coefficient	Runners count
6.2314036	3.148191721	10
8.3154112	3.723946533	9
7.2860206	3.784324393	8
8.3275759	4.510006511	7
9.3571347	5.2	6
8.3206423	6.068589439	5
10.3784709	7.649691352	4
12.488502	10.03377629	3
15.6074876	14.93333333	2
31.1501942	0	1

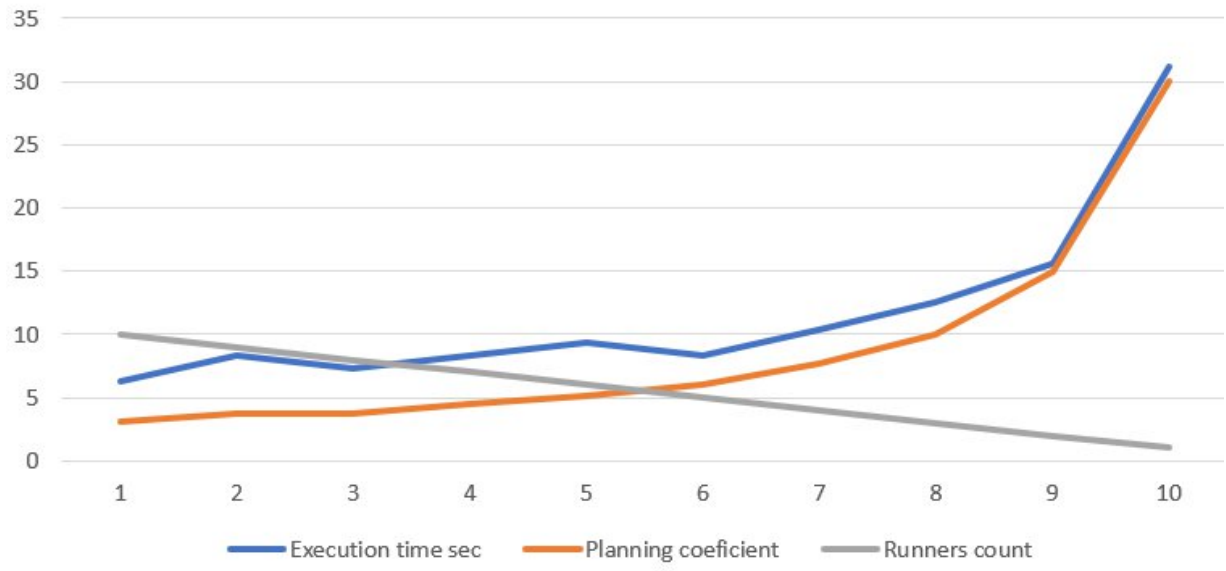


Рис. 3.16. Експеримент з кількістю обчислювальних одиниць від 1 до 10

На рисунку 3.16 зображено графік зміни швидкості обрахунку та коефіцієнта балансування від кількості обчислювальних одиниць.

3.7. Висновки до розділу

У поточному розділі було реалізовано та протестовано алгоритм балансування процесів між різною кількістю обчислювальних одиниць. Було досліджено необхідність обрахунку коефіцієнту балансування при розподіленні процесів. У результаті проведення експериментів було визначено, що при зменшенні коефіцієнта балансування, час виконання однакової кількості процесів зменшується, також досліджено що при збільшенні кількості обчислювальних одиниць, час виконання зменшується.

РОЗДІЛ 4
ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1. Охорона праці

Метою кваліфікаційної роботи магістра є створення власного алгоритму планування процесів з врахуванням всіх недоліків існуючих алгоритмів. Оскільки для проведення розробки та використання системи передбачене використання комп'ютерної техніки, а саме комп'ютера та периферійних пристроїв, то обов'язковим є дотримання вимог з охорони праці та техніки безпеки.

Для забезпечення максимальної ефективності роботи, а також підтримання безпеки колективу працівників з розробки програмного забезпечення, необхідно організувати безпечні умови праці.

Для того щоб організувати захист від негативного впливу екранів дотримано вимог НПАОП 0.00-7.15-18, який затверджений наказом Міністерства соціальної політики України 14.02.2018 № 207.

Під час виконання кваліфікаційної роботи магістра, робоче місце облаштоване згідно наведених вимог та відповідає організаційним, ергономічним та вимогам з пожежної безпеки.

У будівлі, де виконувався проект і приміщення його потенційної експлуатації, було дотримано вимог державних будівельних норм ДБН В.1.1-7-2016, а також визначення категорій приміщень, будинків та зовнішніх установок за вибухопожежною та пожежною небезпекою ДСТУ Б.В.1.1-36:2016. Приміщення де розроблявся алгоритм відносяться до категорії "Д зниженопожежонебезпечна" згідно ДСТУ Б В.1.1-36:2016.

У приміщеннях, де розташовуються робочі місця користувачів ПК потрібно забезпечити відповідність вимогам санітарних норм і правил [28]. В усіх інших приміщеннях потрібно встановлювати теплові пожежні сповіщувачі. Приміщення, де розміщені робочі місця операторів керування обчислювальними планувальниками, мають бути оснащені вогнегасниками, кількість яких

визначається згідно з вимогами ДСТУ 4297:2004 «Пожежна техніка. Технічне обслуговування вогнегасників». Загальні технічні вимоги і з урахуванням граничнодопустимих концентрацій вогнегасної рідини відповідно до вимог НАПБ А.01.001-2014. Приміщення, в яких розміщуються робочі місця операторів сервера загального призначення, обладнуються системою автоматичної пожежної сигналізації та засобами пожежогасіння відповідно до вимог ДБН В.2.5-56:2014, НАПБ А.01.001-2014. Проходи до засобів пожежогасіння мають бути вільними.

У робочій зоні виробничих приміщень вміст шкідливих речовин не повинен перевищувати граничнодопустимих концентрацій, встановлених наказ МОЗ № 1596 «Про затвердження гігієнічних регламентів допустимого вмісту хімічних і біологічних речовин у повітрі робочої зони».

Параметри мікроклімату в межах робочої зони повинні відповідати вимогам Санітарних норм мікроклімату виробничих приміщень ДСН 3.3.6.042- 99. Рівень шуму на робочих місцях повинен відповідати нормам, встановленим Санітарними нормами виробничого шуму, ультразвуку та інфразвуку ДСН 3.3.6.037-99.

Окрім, забезпечення оптимальних показників мікроклімату, необхідно передбачити ще й оптимальні показники шуму та вібрації на робочих місцях. Рівень вібрації на робочих місцях не повинен перевищувати норм, встановлених Державними санітарними нормами виробничої загальної та локальної вібрації ДСН 3.3.6.039-99.

Параметри електромагнітних полів на робочих місцях повинні відповідати вимогам Державних санітарних норм і правил при роботі з джерелами електромагнітних полів, затверджених наказом Міністерства охорони здоров’я України від 18 грудня 2002 року № 476, зареєстрованих у Міністерстві юстиції України 13 березня 2003 року за № 203/7524 (ДСН 3.3.6.096-2002).

Граничні величини шуму на робочих місцях регламентуються ДСН 3.3.6.037 – 99 „Державні санітарні норми виробничого шуму, ультразвуку та інфразвуку”. В ньому закладено принцип встановлення певних параметрів шуму, виходячи з класифікації приміщень та їх використання для трудової діяльності. Окрім цього, на робочих місцях працівників необхідно забезпечити дотримання вимог НПАОП

0.00-7.15-18 «Вимоги щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями».

Забороняється смітити на робочих місцях матеріалами, деталями і предметами, які не використовується у процесів роботи.

Температура нагрітих поверхонь устаткування та огороджень не повинна перевищувати +43 °С згідно з вимогами ДСТУ EN 563-2001 «Безпечність машин. Температури поверхонь, доступних для дотику. Ергономічні дані для встановлення граничних значень температури гарячих поверхонь».

До роботи не будуть допущені особи, які не пройшли навчання з техніки безпеки. Даний кабінет задовольняє вимоги щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями що, відображені в НПАОП 0.00-7.15-18.

Таким чином, проведено аналіз основних питань і заходів, які стосуються охорони праці та можуть впливати на здоров’я і життя операторів керування обчислювальними процесами.

4.2. Стійкість роботи комп'ютерної системи під час надзвичайних ситуацій воєнного часу

Під час надзвичайних ситуацій воєнного стану, стійкість роботи комп’ютерної системи є надзвичайно важливим аспектом, щоб забезпечити безперервну роботу гуманітарних організацій, військових підрозділів, комерційних підприємств та державних установ. Війна сприяє створенню великої кількості викликів для інформаційних технологій, від фізичного пошкодження обладнання до створення системи захисту від масштабних кіберзагроз. У подібних обставинах комп’ютерна система повинна залишатись працездатною та продовжувати забезпечення надійності, безпеки та доступності інформації.

Існують наступні ключові аспекти забезпечення стійкості роботи комп'ютерної системи під час надзвичайних ситуацій воєнного часу, такі як:

- Фізичний захист.

- Кібербезпека.
- Енергетична автономність.
- Мережева стійкість.

Ключовим аспектом стійкості є фізичний захист інфраструктури. Сервери, комунікаційні вузли та центри обробки даних необхідно розташовувати у безпечних місцях, з захистом від прямих фізичних атак, таких як диверсія чи бомбардування. Для того щоб зменшити ризик повної втрати критично важливих даних, використовується геореплікація. Геореплікація це технологія яка синхронізує дані між декількома центрами обробки даних.

Кібербезпека в умовах війни набуває ще більшого значення. Існує чимало прикладів кіберзлочинів, а саме: атаки на державні та військові ресурси, перехоплення конфіденційної інформації та поширення шкідливого програмного забезпечення. Для захисту від кіберзагроз необхідно використовувати багаторівневу систему безпеки: система виявлення вторгнень, шифрування даних, антивірусні програми та фаєрволи. Необхідно проводити регулярне навчання персоналу основам кібербезпеки, для того щоб ознайомити їх як діяти при тій чи іншій кіберзагрозі.

Енергетична автономність є одною з найбільш важливих умов функціонування комп’ютерних систем у умовах дефіциту електроенергії. Для забезпечення автономності роботи комп’ютерних систем при відсутності центрального електропостачання використовують джерела безперебійного живлення, дизельні та бензинові генератори та сонячні панелі. Оптимізація енергоспоживання допомагає зменшити залежність від зовнішніх джерел енергії.

Мережева стійкість досягається за допомогою використання децентралізованих рішень, таких як peer-to-peer мережі, альтернативні канали зв’язку, супутникові системи та VPN для захисту передавання даних. Все це дозволяє забезпечувати зв’язок навіть у випадку пошкодження основної інфраструктури.

Потрібно також завжди враховувати портативність і модульність обладнання. Використання компактних серверів та сховищ, які можна легко

транспортувати дозволяє легко змінити поточне розміщення комп'ютерної системи в умовах воєнного стану при виникненні надзвичайної ситуації. Модульні комп'ютерні системи можна без проблем розгорнути у новій локації за короткий проміжок часу, що забезпечує адаптивність у надзвичайних ситуаціях.

Надзвичайні ситуації мають прямий вплив на стійкість автоматизованих комп'ютерних систем [26].

Є можливість створення єдиної комп'ютерної мережі, яка забезпечує автономне і спільне функціонування складових цієї системи та взаємозв'язок з іншими інформаційними системами, які діють в Україні і за кордоном [27].

Таким чином забезпечення стійкості комп'ютерної системи під час надзвичайних ситуацій воєнного стану вимагає комбінації технологічних, соціальних та організаційних рішень. Тільки комплексний підхід, що включає повне резервування, енергетичну автономність, адаптація до нових умов та надійний кіберзахист має змогу гарантувати ефективну роботу комп'ютерних систем навіть у найскладніший обставинах війни.

ВИСНОВКИ

У кваліфікаційній роботі проаналізовано наявні вирішення для планування завдань між доступними обчислювальними ресурсами, а саме алгоритмічне та програмне забезпечення.

Проаналізовано особливості планування виконання процесів в сучасних операційних системах Linux та розподілених системах оркестрації контейнерів Kubernetes.

Досліджено можливість застосування алгоритму безперервного хешування в системах планування обчислювальних завдань. На основі детального аналізу переваг та недоліків наявного алгоритмічного забезпечення обґрунтовано вибір оптимального для конкретної програмної системи планування обчислювальних завдань, яке враховує її особливості. Такими алгоритмами є Round-Robin та consistent hashing.

Розроблено універсальне програмне забезпечення, для забезпечення максимально ефективного планування процесів серед доступних обчислювальних одиниць. Здійснено тестування роботи цього програмного забезпечення, проведено обчислювальний експеримент й проаналізовано його результати.

Також у роботі розглянуто питання стійкості роботи комп'ютерної системи під час надзвичайних ситуацій воєнного часу та охорони праці досліджуваної системи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Martin Kleppmann Designing Data Intensive Applications. United States, 2017. P. 204

2. Толок А.О. Крюковська О.А. Безпека життєдіяльності: Навч. посібник. 2011. 175 с.

3. Зеркалов Д. Охорона праці в галузі: Загальні вимоги. Навчальний посібник. К.: Основа. 2011. 356 с.

4. Andrew Tanenbaum Modern Operating Systems. United States, 2015. P. 113

5. A. Silberschatz, P. Baer Galvin, G. Gagne Operating System Concepts Essentials. United States. 2014. P. 561

6. А. Луцків, А. Люлька Застосування методу безперервного хешування у плануванні виконання послідовних процесів. Актуальні задачі сучасних технологій: Праці XIII наук.-техн. конф. (Тернопіль, 11-12 грудня 2024 р.), Тернопіль, 2024. С. 410.

7. А. Луцків, А. Люлька Застосування алгоритмів балансування навантаження в процесах сучасних операційних систем. Інформаційні моделі, системи та технології: Праці XII наук.-техн. конф. (Тернопіль, 18-19 грудня 2024 р.), Тернопіль, 2024. С. 237.

8. Луцик Н.С., Луцків А.М., Осухівська Г.М., Тиш Є.В. Методичні рекомендації до виконання кваліфікаційної роботи магістра. Тернопіль, ТНТУ. 2024. 17с.

9. Jeffrey Richter CLR via C#. Київ, 2012. 273 с.

10. Роберт Мартін. Чиста Архітектура. Київ, 2020. 135с.

11. Марк Д. C# 11 .NET 7 Фундаментальні основи багатоплатформної розробки. Київ, 2023р. 131с.

12. Технічна документація EntityFramework. URL: https://www.learnentityframeworkcore.com/ (дата звернення 13.10.2024)

13. Зайцев В., Цибаєв Є. Комп’ютерні системи реального часу: Навчальний посібник. Київ «КПІ ім. Ігоря Сікорського», 2019. 78 с.

14. S. Hutchison Scheduling for high performance computing with reinforcement learning. United States. 2024. P. 156

15. S. Shaharuddin, A. Zomaya Scheduling in Parallel Computing Systems. United States. 1999. P. 275

16. Луцків А. Імітаційне моделювання циклічних випадкових процесів. Львів, 2006. 184 с.

17. Макмілан М. Data Structures and Algorithms with JavaScript. Київ, 2017. 56 с.

18. Системи підтримки прийняття рішень. Х. : Інжек, 2006. 231 с.

19. Волошин, О. Ф. Моделі та методи прийняття рішень : навч. посіб. для студ. вищ. навч. закл. 2-ге вид., перероб. та допов. К. : Видавничо-поліграфічний центр "Київський університет", 2020. 269 с.

20. Пушкар О. І. Системи підтримки прийняття рішень: навч. Посібник. Харків : Інжек 2006. 187 с.

21. Failover Clustering in Windows Server. . URL: https://learn.microsoft.com/en-us/windows-server/failover-clustering/failover-clustering-overview (дата звернення: 01.12.2024).

22. Kernel Virtual Machine. URL: https://linux-kvm.org/page/Main_Page (дата звернення: 18.10.2024).

23. Details About Hardware Virtualization. URL: https://docs.oracle.com/en/virtualization/virtualbox/6.0/admin/hwvirt-details.html (дата звернення: 14.12.2024).

24. The National Grid Infrastructure. URL: http://ung.bitp.kiev.ua/ua/ (дата звернення: 16.12.2024).

25. Комп’ютингові системи типу Boinc. URL: https://boinc.berkeley.edu/ (дата звернення: 08.12.2024).

26. Стручок В.С. Техноекологія та цивільна безпека. Частина «Цивільна безпека». Навчальний посібник. Тернопіль: ТНТУ. 2022. 21 с.

27. Стручок В.С. Безпека в надзвичайних ситуаціях: Навчальний посібник Тернопіль: ТНТУ. 2016. 37 с.

28. Управління інспекційної діяльності у Тернопільській області Південно-Західного міжрегіонального управління Державної служби з питань праці URL: https://te.dsp.gov.ua/robota-v-ofisi-osnovni-sanitarno-gigiyenichni-vymogy/ (дата звернення: 12.12.2024).

29. Луцик Н.С., Луцків А.М., Осухівська Г.М., Тиш Є.В. Програма та методичні рекомендації з проходження практики за тематикою кваліфікаційної роботи для студентів спеціальності 123 «Комп'ютерна інженерія» другого (магістерського) рівня вищої освіти усіх форм навчання. Тернопіль. ТНТУ. 2024. 41 с.

30. Луцик Н. С., Луцків А. М., Осухівська Г. М., Тиш Є. В. Методичні рекомендації до виконання кваліфікаційної роботи магістра для студентів спеціальності 123 «Комп'ютерна інженерія» другого (магістерського) рівня вищої освіти усіх форм навчання. Тернопіль. ТНТУ. 2024. 47 с.

31. Варавін А.В., Лещишин Ю.З., Чайковський А.В. Методичні вказівки до виконання курсового проєкту з дисципліни «Дослідження і проєктування комп'ютерних систем та мереж» для здобувачів другого (магістерського) рівня вищої освіти спеціальності 123 «Комп'ютерна інженерія» усіх форм навчання. Тернопіль. ТНТУ, 2024. 36 с.

ДОДАТОК А
Тези конференції

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Тернопільський національний технічний університет імені Івана Пулюя (Україна)
Університет імені П'єра і Марії Кюрі (Франція)
Маріборський університет (Словенія)
Технічний університет у Кошице (Словаччина)
Вільнюський технічний університет ім. Гедимінаса (Литва)
Наукове товариство ім. Т.Шевченка

АКТУАЛЬНІ ЗАДАЧІ СУЧАСНИХ ТЕХНОЛОГІЙ

Збірник
тез доповідей

XIII Міжнародної науково-практичної конференції молодих учених та студентів
11-12 грудня 2024 року

УКРАЇНА
ТЕРНОПІЛЬ – 2024

УДК 004.9

А. Луцків, А. Люлька
(Тернопільський національний технічний університет імені Івана Пулюя, Україна)

ЗАСТОСУВАННЯ МЕТОДУ БЕЗПЕРЕРВНОГО ХЕШУВАННЯ У ПЛАНУВАННІ ВИКОНАННЯ ПОСЛІДОВНИХ ПРОЦЕСІВ

UDC 004.9

A. Lutskiv, A. Liulka

APPLICATION OF THE CONTINUOUS HASHING METHOD IN PLANNING THE EXECUTION OF SEQUENTIAL PROCESSES

Забезпечення ефективного планування процесів є ключовим завданням для сучасних багатозадачних обчислювальних систем. Одним із перспективних підходів для підвищення ефективності планування є використання методу безперервного хешування. Цей метод дозволяє оптимізувати управління послідовними процесами шляхом ефективного розподілу ресурсів та мінімізації затримок під час виконання завдань. Графічне представлення методу безперервного хешування можна побачити на рисунку 1.

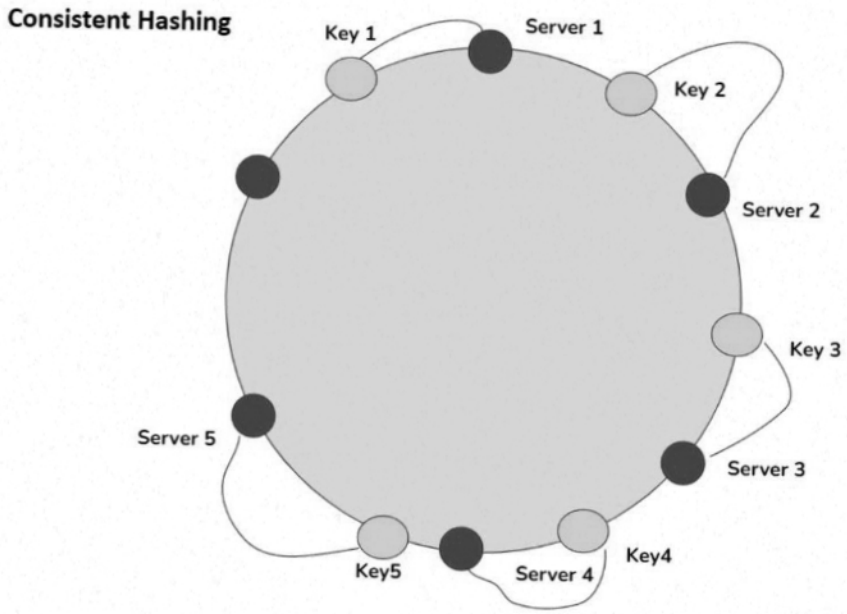


Рисунок 1. Принцип безперервного хешування

Безперервне хешування забезпечує динамічний розподіл процесів по доступних ядрах, уникаючи перевантаження окремих ядер і сприяючи рівномірному розподілу обчислювальних потужностей. Це особливо важливо в умовах змінних навантажень та обмежених ресурсів, де традиційні методи планування можуть бути менш ефективними. Однією з ключових переваг методу є його здатність забезпечувати адаптивне перенаправлення процесів у разі змін у системі, що робить його стійким до непередбачуваних змін.

Даний алгоритм повинен розподіляти вхідну чергу процесів між доступними ядрами та у разі неможливості певного ядра продовжувати роботу, перебалансовувати всі його процеси на наступне у списку ядро.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
імені ІВАНА ПУЛЮЯ
НАУКОВЕ ТОВАРИСТВО ім. ШЕВЧЕНКА

М А Т Е Р І А Л И

XII НАУКОВО-ТЕХНІЧНОЇ КОНФЕРЕНЦІЇ

«ІНФОРМАЦІЙНІ МОДЕЛІ, СИСТЕМИ ТА ТЕХНОЛОГІЇ»

18-19 грудня 2024 року

ТЕРНОПІЛЬ
2024

УДК 004.9
А. Луцків, А. Люлька
(Тернопільський національний технічний університет імені Івана Пулюя, Україна)

ЗАСТОСУВАННЯ АЛГОРИТМІВ БАЛАНСУВАННЯ НАВАНТАЖЕННЯ В ПРОЦЕСАХ СУЧАСНИХ ОПЕРАЦІЙНИХ СИСТЕМ

UDC 004.9
A. Lutskiv, A. Liulka

APPLICATION OF LOAD BALANCING ALGORITHMS IN THE PROCESSES OF MODERN OPERATING SYSTEMS

Забезпечення ефективного використання ресурсів є критично важливим завданням для сучасних операційних систем (ОС). Алгоритми балансування навантаження виступають ключовим інструментом у розподілі процесів між доступними ресурсами, зокрема процесорами, ядрами та серверами, забезпечуючи їх оптимальну роботу. Ці алгоритми дозволяють мінімізувати затримки, запобігати перевантаженню окремих ресурсів та забезпечувати справедливий розподіл обчислювальної потужності.

Для правильного розподілу навантаження між обчислювальними одиницями, в комп'ютерних системах використовуються певні алгоритми. А саме:

- Round Robin – алгоритм послідовного розподілу навантаження;
- Least connections – алгоритм розподілу навантаження до обчислювальних одиниць з найменьшою кількістю активних підключень;
- Least Response Time – алгоритм переадресування запитів до обчислювальної одиниці з найкоротшим часом відповіді;
- IP Hash – алгоритм розподілу на основі хешу IP адреси;
- Weighted Round Robin – варіація Round Robin алгоритму з можливістю призначення ваги обчислювальної одиниці для подальшого визначення можливого навантаження цієї обчислювальної одиниці;
- Random – алгоритм випадкового розподілення;
- Dynamic Load Balancing – алгоритм визначення навантаження для кожної обчислювальної одиниці базуючись на метриках, навантаженні центрального процесора та використанні пам'яті;

Дуже важливу роль також відіграють алгоритми планування виконання процесів всередині одної обчислювальної одиниці. Існують основні алгоритми планування процесів всередині кожної обчислювальної одиниці, а саме:

- Round Robin;
- Shortest Job Next;
- First-Come, First-Served;
- Multilevel Queue Scheduling;

У сучасних операційних системах планувальники необхідні для розподілення навантаження процесів між ядрами центрального процесора. Такий принцип дозволить іншим ядрам обробити чергу процесів у випадку:

- Збою сусіднього ядра;
- Негайним переключенням ядра не невідкладну та довготривалу задачу;