Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет навчально-науковий центр заочної форми навчання
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти перший (бакалаврський)

Система для моделювання багатопроцесорних алгоритмів планування

(тема)

Виконав:
здобувач 4 року навчання,
групи КІУКІз-21-1
Максим ОЛЕКСАНДРЕНКО
(власне ім'я, прізвище)

Спеціальність
123 «Комп'ютерна інженерія»
(код і повна назва спеціальності)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма
Комп'ютерна інженерія
(повна назва освітньої програми)

Керівник: ст. викл. Вячеслав РАДЧЕНКО
(посада, власне ім'я, прізвище)

Допускається до захисту

Завідувач кафедри ЕОМ _____ Андрій КОВАЛЕНКО
(підпис) (власне ім'я, прізвище)

2025 р.

Харківський національний університет радіоелектроніки

Факультет навчально-науковий центр заочної форми навчання

Кафедра електронних обчислювальних машин

Рівень вищої освіти перший (бакалаврський)

Спеціальність 123 «Комп'ютерна інженерія»
(код і повна назва)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Комп'ютерна інженерія
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
"____" _____ 20__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Олександренку Максиму Андрійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Система для моделювання багатопроцесорних алгоритмів планування

затверджена наказом по університету від " 05 " травня 2025 р. № 72 Стз

2. Термін подання здобувачем роботи до екзаменаційної комісії 16 червня 2025 р.

3. Вхідні дані до роботи мережі Петрі, багатопроцесорне середовище, алгоритми симуляція, GPenSIM, MATLAB, моделювання, продуктивність систем

4. Перелік питань, що потрібно опрацювати у роботі

Теоретичні основи роботи

Концепція процесів

Мережі Петрі та GPenSIM

Проєктування системи

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій 11

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Отримання теми кваліфікаційної роботи	05.05	
2	Аналіз літератури	05.05-19.05	
3	Побудова системи	20.05-10.06	
4	Тестування системи та отримання результатів	11.06-12.06	
5	Формування пояснювальної записки	13.06-14.06	
6	Перевірка на плагіат	15.06-17.06	
7	Рецензування роботи	17.06	
8	Подача роботи в ЕК	18.06	
9	Захист роботи	24.06	

Дата видачі завдання " 05 " травня 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ ст. викл. Вячеслав РАДЧЕНКО
(підпис) (посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 55 с., 9 рис., 1 табл., 1 дод., 25 джерел.

МЕРЕЖІ ПЕТРІ, БАГАТОПРОЦЕСОРНЕ СЕРЕДОВИЩЕ, АЛГОРИТМИ ПЛАНУВАННЯ, СИМУЛЯЦІЯ, GPENSIM, MATLAB, МОДЕЛЮВАННЯ, ПРОДУКТИВНІСТЬ СИСТЕМ.

Метою кваліфікаційної роботи є створення системи моделювання багатопроцесорних алгоритмів планування

У ході виконання кваліфікаційної роботи проведено дослідження ефективності різних алгоритмів планування завдань у багатопроцесорному середовищі шляхом моделювання за допомогою мереж Петрі. Моделювання реалізовано із використанням симулятора GPenSIM на платформі MATLAB. У зв’язку з поступовим зниженням ефективності закону Мура та зростанням кількості ядер у сучасних процесорах, питання оптимального планування стало особливо актуальним для забезпечення високої продуктивності обчислювальних систем.

ABSTRACT

Bachelor’s thesis: 55 pages, 9 figures, 1 tables, 1 appendices, 25 sources.

PETRI NETWORKS, MULTI-CPU ENVIRONMENT, SCHEDULING ALGORITHMS, SIMULATION, GPENSIM, MATLAB, MODELING, SYSTEMS PERFORMANCE.

The major goal of this thesis is to develop a simulation system for multiprocessor scheduling algorithms.

In the course of the qualification work, a study was conducted to evaluate the efficiency of various task scheduling algorithms in a multiprocessor environment using Petri net-based modeling. The simulation was implemented with the GPenSIM simulator on the MATLAB platform. Due to the gradual decline in the effectiveness of Moore's Law and the increasing number of cores in modern processors, the issue of optimal scheduling has become particularly relevant for ensuring high performance in computing systems.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

FCFS – First-Come, First-Served, хто перший прийшов, той перший обслуговується

SJF – Shortest Job First, найкоротше завдання перше

RR – Round Robin , круговий (циклічний) планувальник

MLFQ – Multi-Level Feedback Queue

CFS –повністю справедливий планувальник

CPU – центральний процесор

ВСТУП

Ефективне багатопроцесорне планування є ключовим фактором для оптимізації продуктивності паралельних обчислювальних систем. У цій кваліфікаційній роботі використовується потужність мереж Петрі та інструменту GPenSIM для моделювання та імітації різноманітних алгоритмів багатопроцесорного планування (базових алгоритмів, таких як «перший прийшов, перший обслуговується», «найкоротші завдання першими» та «циклічний процес», а також більш складних планувальників, таких як багаторівнева черга зворотного зв'язку та повністю справедливий планувальник Linux). У цій роботі представлено оцінку трьох ключових показників продуктивності багатопроцесорного планування (таких як час виконання, час відгуку та пропускна здатність) за різними алгоритмами планування. Однак основна увага приділяється розробці надійної платформи моделювання, що складається з модулів Петрі, для полегшення динамічного представлення паралельних процесів, що дозволяє нам досліджувати взаємодії та залежності в реальному часі в багатопроцесорному середовищі; більш просунуті та новіші планувальники можна протестувати за допомогою платформи моделювання, представленої в цій роботі.

1 ТЕОРЕТИЧНІ ОСНОВИ РОБОТИ

1.1 Постановка задачі

Ця робота має на меті дослідити та оцінити різноманітні алгоритми планування в багатопроцесорному середовищі з використанням мереж Петрі. Моделювання буде здійснюватися за допомогою симулятора мереж Петрі загального призначення (GPenSIM) [1], інструментарію, що працює на платформі MATLAB.

З огляду на постійне зниження ефективності закону Мура, виробники процесорів відреагували збільшенням кількості ядер для підтримки покращення продуктивності [2]. Зі зростанням кількості ядер ефективне планування в операційних системах стало критично важливим. На відміну від минулого, коли завдання планувалися на одному ядрі, сучасні операційні системи тепер повинні керувати кількома ядрами, де планувальник відіграє ключову роль у максимізації продуктивності процесора.

Основною метою проєкту, є розробка платформи моделювання з мережами Петрі та GPenSIM, щоб можна було протестувати та оцінити минулі, сучасні та майбутні (нові) алгоритми планування. Платформа моделювання складається з модулів Петрі (модульних мереж Петрі).

У цій роботі оцінюється п'ять різних алгоритмів планування в багатопроцесорному середовищі з використанням цієї платформи моделювання на основі модулів Петрі. У цій роботі оцінюються фундаментальні планувальники, такі як «хто перший прийшов, той перший обслуговується» (FCFS), «найкоротший планувальник» (SJF) та «круговий планувальник» (RR). Крім того, у цій роботі аналізуються більш складні планувальники, такі як багаторівнева черга зворотного зв'язку (MLFQ) та повністю справедливий планувальник Linux (CFS) [3]. Завдяки всебічному порівнянню різних робочих навантажень, наша мета полягає в оцінці

продуктивності цих планувальників, зосереджуючись на ключових показниках, таких як час виконання та час відгуку. З огляду на його реалізацію в ядрі Linux, CFS служить цінним орієнтиром для порівняння з менш складними алгоритмами планування, такими як FCFS, SJF та RR.

1.2 Огляд літературних джерел

Існує безліч робіт з багатопроцесорного планування. Для ґрунтовного вивчення рекомендуються такі книги [4-6].

Деякі роботи з багатопроцесорного планування зосереджені на порівнянні нових алгоритмів планування або продуктивності різних алгоритмів. У роботах [7, 8] тестуються нові планувальники на основі генетичних алгоритмів у багатопроцесорному середовищі. У роботі [9] використовується гібридний алгоритм оптимізації рою частинок як планувальник. У роботі [10] використовується алгоритм мережевого потоку, а в роботі [11] досліджується алгоритм упаковки в контейнери.

У літературному дослідженні також виявлено роботи, що досліджують оптимізацію багатопроцесорного планування шляхом зміни або впровадження нових політик. У роботі [12] досліджується вплив додавання обмежень термінів виконання завдань; подібний метод («межі термінів затримки») досліджується в [13]. У роботі [14] розглядається політика фіксованого пріоритету з межами використання, щоб зробити планування більш справедливим для конкуруючих процесів. У роботі [15] вивчається вплив попереднього замовлення («частково впорядкованих») завдань для оптимального планування. Нарешті, у роботі [16] розглядається особливість щодо завдань, які можуть бути відхилені зі штрафами, та те, як штрафи покращують загальні показники.

Третя група робіт зосереджена на апаратному забезпеченні. У роботі [17] досліджується застосування динамічного масштабування напруги (DVS) для зменшення динамічного енергоспоживання вбудованих

мультипроцесорів. У роботі [18] досліджується застосування інформації про спорідненість процесора та кешу в мультипроцесорній системі зі спільною пам'яттю. Нарешті, у роботі [19] розглядається планування багатопроцесорної роботи в системах, що характеризуються затримками (затримки є поширеними).

Новизна нашої роботи, хоча в цій роботі представлено оцінку трьох ключових показників продуктивності в багатопроцесорному плануванні (таких як час виконання, час відгуку та пропускна здатність) за різними алгоритмами планування, основна увага в цій роботі приділяється розробці надійної платформи моделювання, що складається з модулів Петрі. Розробляючи цю платформу, ми сподіваємося полегшити динамічне представлення паралельних процесів, що дозволить нам досліджувати взаємодії та залежності в реальному часі в багатопроцесорному середовищі, щоб можна було протестувати більш просунуті та новіші планувальники.

2 КОНЦЕПЦІЯ ПРОЦЕСІВ

У цьому розділі представлено концепцію процесів, добре відомої абстракції, що використовується в операційних системах. Крім того, у цьому розділі досліджуються різні алгоритми планування, які будуть реалізовані та протестовані в наступних розділах. Нарешті, у цьому розділі розглядається, як алгоритм планування, розроблений для одного процесора, може бути адаптований для багатопроцесорного планування.

2.1 Процес

Операційні системи, такі як Windows та Linux, використовують абстракцію, відому як процес, для виконання програм [3]. Ця абстракція створює ілюзію, що кожен процес має свій власний виділений процесор, навіть якщо він використовує один або кілька процесорів спільно з іншими процесами. Як правило, процес може існувати в одному з трьох різних станів:

- виконується, цей стан вказує на те, що процес активно виконується на процесорі;
- готовий, тут процес готовий до запуску, але ще не був обраний планувальником;
- заблоковано/очікує, у цьому стані процес зазвичай виконав певну операцію вводу/виводу та наразі заблоковано або очікує.

Процеси можуть складатися з одного або кількох потоків. Для спрощення в цій роботы розглядаються лише процеси з одним потоком.

Зазвичай операційна система планує процеси та потоки з невідомим часом виконання; отже, для моделювання робляться деякі припущення. Ми вважатимемо, що час виконання для кожного вхідного процесу відомий, що дозволяє нам моделювати однакове навантаження на різних алгоритмах планування. Слово «завдання» використовується в книзі, але в решті цієї

роботи воно буде використовуватися як взаємозамінне з термінами «процес» та «потік». Кожне завдання зберігатиме важливу інформацію для цілей моделювання, що дозволить генерувати метрики після завершення моделювання. Створені завдання охоплюватимуть наступне:

- PID, унікальний ідентифікатор завдання;
- час прибуття, момент створення завдання;
- час початку, початковий час планування завдання;
- час завершення, час завершення роботи;
- залишок часу, тривалість, що залишилася для виконання завдання;
- загальний час, загальний час, необхідний для виконання завдання;
- заплановано, кількість разів, коли завдання було заплановано (через перемикання контексту).
- перенесено, кількість перенесень завдання.

Для деяких специфічних алгоритмів планування необхідні додаткові поля.

2.2 First Come First Serve

Алгоритм планування за принципом «хто перший прийшов, той перший обслуговується» (FCFS) є одним із найпростіших і найзрозуміліших алгоритмів планування процесів, що використовуються в комп'ютерних операційних системах. Наш опис FCFS базується на описі в роботі [3]. У цьому алгоритмі процеси плануються на основі порядку їх надходження в чергу. Процесу, який надходить першим, надається пріоритет і планується першим, а потім наступні надходження йдуть у тому ж послідовному порядку. FCFS має низку переваг. По-перше, його простота дуже гідна похвали, що робить його легким для реалізації та розуміння, що робить його ідеальним вибором для початківців та простих сценаріїв планування. Крім того, FCFS за своєю суттю підтримує справедливість, гарантуючи, що процеси обслуговуються в порядку їх надходження, запобігаючи

голодуванню для будь-якого конкретного процесу. Крім того, відсутність необхідності враховувати пріоритети ще більше спрощує процес планування. Однак FCFS не позбавлений своїх обмежень. Це може призвести до вищого середнього часу очікування, ніж інші алгоритми планування, особливо якщо першими надходять тривалі процеси. Крім того, FCFS може бути не найефективнішим у використанні обчислювальної потужності процесора, головним чином, якщо коротші процеси надходять пізніше в чергу. Додатковим недоліком є потенційна поява ефекту конвою в алгоритмі FCFS, коли менші процеси затримуються через те, що процеси з тривалим часом виконання плануються першими. Це призведе до уповільнення одного процесу, що, своєю чергою, спричинить повільну роботу всієї групи процесів, що призведе до витрачання часу процесора та інших ресурсів.

2.3 Shortest Job First

Алгоритм планування «найкоротший час виконання завдань» (SJF) – це невиправний підхід до планування процесів, де процеси плануються на основі тривалості їх пакетної роботи процесора [3]. SJF вибирає процес з найкоротшим часом виконання процесора для виконання першим. Ця стратегія мінімізує середній час очікування, що призводить до ефективного використання процесора та покращення продуктивності системи. Однак SJF вимагає точного прогнозування або оцінки пакетної роботи процесора, що може бути складним завданням, особливо для нових або невідомих процесів. Крім того, довші процеси можуть відчувати дефіцит, якщо продовжується постійний приплив коротших процесів. В інтерактивних системах, де користувачі очікують швидких відповідей, SJF може бути неідеальним через непередбачуваний характер вводу даних користувача. SJF залишається важливою політикою планування, незважаючи на ці міркування, оскільки він може оптимізувати використання ресурсів та підвищити ефективність системи.

2.4 Round Robin

Алгоритм кругового планування – це широко використовувана техніка випереджувального планування процесів в комп'ютерних операційних системах [3]. У цьому підході кожному процесу призначається фіксований часовий інтервал, відомий як квант часу, протягом якого він може виконуватися на процесорі. Як тільки процес вичерпує свій квант часу, він переміщується в кінець черги готовності, дозволяючи наступному процесу в черзі отримати час процесора. Круговий метод має на меті забезпечити справедливий та рівний час процесора кожному процесу, запобігаючи монополізу процесора будь-яким окремим процесом. Однак цей алгоритм може створювати накладні витрати через часті перемикання контексту, пов'язані з переміщенням між процесами. Квант часу має бути ретельно обраний, щоб збалансувати досягнення швидкості реагування та мінімізацію накладних витрат на перемикання контексту. Кругове планування цінується за свою простоту, ефективність у системах розподілу часу та запобігання голодуванню процесів, забезпечуючи відносно рівномірний розподіл часу процесора між процесами. Одним з недоліків RR є те, що неможливо визначити пріоритети інтерактивних завдань, таких як введення за допомогою миші та клавіатури, навіть якщо він має хороший час відгуку порівняно з FCFS.

2.5 Багаторівнева черга зворотного зв'язку (MLFQ)

Пояснення алгоритму планування багаторівневої черги зворотного зв'язку (MLFQ) базується на описі в роботі [3]. MLFQ – це динамічний метод планування процесів на основі пріоритетів, який використовується в деяких операційних системах. MLFQ динамічно призначає процеси до різних черг з пріоритетом на основі їхньої нещодавньої поведінки, прагнучи оптимізувати час відгуку та час виконання. Цей алгоритм ініціює процеси в черзі з

найвищим пріоритетом та переходить до черг з нижчим пріоритетом на основі моделей використання процесора. MLFQ вивчає процеси та використовує їхню минулу поведінку для прогнозування майбутніх моделей. На основі цього пріоритет процесів буде змінено. Алгоритм забезпечує справедливість, постійно коригуючи пріоритети процесів у відповідь на їхню поведінку. Однак виявлено суттєві недоліки. Однією з основних проблем є потенціал для інтерактивних завдань монополізувати час процесора, залишаючи довготривалі завдання без ресурсів обробки. Крім того, алгоритм вразливий до маніпуляцій, що дозволяє хитрим користувачам обманювати планувальник та отримувати несправедливу частку процесора шляхом стратегічного використання операцій вводу/виводу. Іншим обмеженням є нездатність алгоритму адаптуватися до змін поведінки програми з часом, ігноруючи переходи від фаз, пов'язаних з процесором, до інтерактивних фаз. Налаштування MLFQ ретельно налаштовуються для подолання цих перешкод, таких як кількість черги, розміри часових інтервалів та частота підвищення пріоритету.

MLFQ дотримується правил, викладених у пунктах нижче [3]. Деякі правила були адаптовані для покращення сумісності в рамках моделі мережі Петрі.

Правило 1: оберіть завдання з найвищим пріоритетом.

Правило 2: якщо два завдання мають однаковий найвищий пріоритет, виконуйте їх по черзі.

Правило 3: призначайте найвищий пріоритет завданням, що потрапляють до черги.

Правило 4: якщо відведений для завдання час закінчився, зменште пріоритет завдання на один.

Правило 5: якщо завдання перебувають у черзі протягом певного періоду часу без виконання, підвищте їхній пріоритет до найвищого.

2.6 CFS

Опис повністю справедливого планувальника (CFS) базується на описі в роботі [3]. Спочатку ОС Linux використовувала надзвичайно складний MLFQ, відомий як планувальник O(1). Однак Лінус Торвальдс вважав його надто складним і важким для обдумування. У версії Linux 2.6.23 у 2007 році CFS, винайдений Інго Молнаром [20], став планувальником за замовчуванням. На відміну від традиційних планувальників, які покладаються на фіксовані часові інтервали, CFS надає пріоритет справедливості, прагнучи рівномірно розподілити обчислювальну потужність процесора між конкуруючими процесами. Він досягає цього за допомогою динамічного підходу на основі підрахунку, відомого як віртуальне середовище виконання (vruntime), де кожен процес накопичує vruntime пропорційно до свого фактичного часу виконання. Щоб забезпечити справедливий розподіл ресурсів процесора під час прийняття рішень щодо планування, CFS вибирає процес з найменшим vruntime для виконання наступним.

CFS використовує параметр sched_latency для визначення тривалості процесу перед перемиканням контексту. Налаштування sched_latency за замовчуванням зазвичай налаштовується на 48 мс, що призводить до відповідного time_slice 48мс. Фактичний time_slice також залежить від наявності інших процесів у черзі. Коли є n процесів, time_slice розраховується як

$$time\_slice = \frac{sched\_latency}{n}$$

Тут time_slice представляє час, протягом якого процес виконується перед тим, як його вимкнути. Однак, зі збільшенням кількості процесів, sched_latency може стати надзвичайно малою, що потенційно призведе до

перевищення витрат на перемикання контексту над виділеним часом. Щоб вирішити цю проблему, CFS вводить параметр під назвою min_granularity, який встановлює мінімальний часовий інтервал, протягом якого процес виконуватиметься перед вимкненням. Значення min_granularity за замовчуванням у Linux становить 6мс. Отже, часовий інтервал розраховується за допомогою рівняння

$$time\_slice = max\left(\frac{sched\_latency}{n}, min\_graunularity\right)$$

CFS надає пріоритет справедливому плануванню використання процесора та дозволяє детально контролювати пріоритет процесів за допомогою механізму, відомого як «niceness» (приємність), який призначає значення «nice» кожному процесу в системі. Параметр «nice» можна змінити, щоб виділити більшу частку процесора певним процесам. Цей параметр коливається від -20 до +19 для процесу, зі значенням за замовчуванням 0. Додатне значення nice призначає процесу нижчий пріоритет, тоді як від'ємне значення надає вищий пріоритет. Концепція nice надає користувачам або адміністраторам практичний спосіб впливати на розподіл ресурсів процесора для певних процесів. CFS пов'язує значення nice процесів з певними вагами, як показано в таблиці 2.1

Застосовуючи вагу заданого процесу k та ділячи її на сумарну вагу всіх процесів n, ми отримуємо рівняння – призначений час виконання завдання k.

$$time\_slice_k = \max\left(\frac{weight_k}{\sum_{i=0}^{n-1} weight_i} \cdot sched\_latency, min\_granularity\right)$$

Таблиця 2.1 – Відображення якості у вагу у формі $niceness \mapsto weight$

−20↦88761	−19↦71755	−18↦56483	−17↦46273	−16↦36291
−15↦29154	−14↦23254	−13↦18705	−12↦14949	−11↦11916
−10↦9548	−9↦7620	−8↦6100	−7↦4904	−6↦3906
−5↦3121	−4↦2501	−3↦1991	−2↦1586	−1↦1277
0↦1024	1↦820	2↦655	3↦526	4↦423
5↦335	6↦272	7↦215	8↦172	9↦137
10↦110	11↦87	12↦70	13↦56	14↦451
15↦36	16↦29	17↦23	18↦18	19↦15

CFS покладається на періодичне переривання таймера для оцінки частоти та вирішення питання про необхідність перемикання контексту. Через потенційні десяткові коми в time_slice k, певні процеси можуть виконуватися трохи довше, ніж передбачалося. Однак цей ефект має збалансуватися з часом, забезпечуючи приблизне використання процесорного часу. У нашому моделюванні ми спрощуємо це, округляючи значення.

Крім того, CFS коригує розрахунок vruntime на основі ваги, забезпечуючи справедливий розподіл ресурсів процесора з урахуванням якості та пріоритетів процесу. Ця інтелектуальна адаптація підтримує пропорційне розподілення процесора, зберігаючи справедливість, навіть коли значення якості відрізняються. Наприклад, процес з вищим значенням якості накопичуватиме vruntime зі збільшенням порівняно з процесом з нижчим значенням якості. Накопичення vruntime зображено в рівнянні, де $runtime_i$ представляє фактичний час процесора, що використовується процесом i, $weight_0$=1024 – вага за замовчуванням для рівня якості 0, як зазначено в таблиці 2.1, та $weight_i$ відповідає вазі процесу i.

$$vruntime_i = vruntime_i + \frac{weight_0}{weight_i} \cdot runtime_i$$

2.6.1 Red-Black Trees

Пошук наступного завдання для планування є ключовим аспектом будь-якого планувальника, особливо для CFS, який використовується в глобально поширеній операційній системі Linux. Базове рішення полягає в тому, щоб підтримувати всі дані в структурі даних, подібній до черги, та проходити через неї цикл, коли потрібно запланувати наступне завдання. Однак це просте рішення має часову складність $O(n)$, лінійно масштабуючись з кількістю завдань у черзі, яка в сучасних системах може коливатися від 100 до 1000. CFS вирішує цю проблему за допомогою структури даних, відомої як червоно-чорне дерево. Це збалансоване дерево є чергою пріоритетів, що дозволяє вставляти та видаляти завдання в $O(\log\ n)$ часова складність. Цей підхід масштабується набагато ефективніше, ніж використання фундаментальної структури даних, такої як масив. Важливо зазначити, що складна червоно-чорна деревоподібна структура даних не буде реалізована в симуляції.

2.6.2. Нові робочі місця та сплячі робочі місця

Коли нові завдання надходять у систему або переходять із заблокованого стану до стану готовності, можуть виникнути проблеми через відставання їхнього vruntime від активних завдань. Якщо нове завдання з vruntime, встановленим на нуль, зустрічає мінімальне vruntime, скажімо, 500 серед існуючих завдань, нове завдання може монополізувати процесор, доки його vruntime не наздожене його. Подібна ситуація може виникнути, коли завдання переходить із тривалого заблокованого/очікуючого стану.

Щоб вирішити проблему монополізації процесора, завдання, що надходять у систему, та заблоковані завдання, що переходять у стан готовності, повинні отримати нове vruntime. Це нове vruntime встановлюється на мінімальне vruntime для всіх поточних завдань. Таким

чином, ми запобігаємо монополізації процесора новонадійшлими або нерозблокованими завданнями та забезпечуємо справедливе планування між усіма активними завданнями.

2.7 Багатопроцесорне планування

Хоча в попередніх підрозділах детально розглядалися різні алгоритми планування в одноядерних системах, основний акцент у цій роботі робиться на моделюванні, спрямованому на планування в багатопроцесорному середовищі. Використовуючи один процесор, планування використовується для створення ілюзії того, що кожен процес у системі має свій власний процесор. Насправді всі процеси використовують один процесор, і концепції перемикання контексту та складних алгоритмів планування намагаються приховати цей факт. У багатопроцесорній системі паралелізм досяжний, оскільки два процеси можуть виконуватися одночасно на різних ядрах. Теоретично може здатися, що система з чотирма процесорами демонструватиме пропускну здатність, яка в чотири рази більша, ніж система з одним процесором. Однак на практиці це не так. Це відбувається тому, що кожен процесор має власні кеші та буфер перегляду трансляцій (TLB), доступний лише з пов'язаного з ним процесора. Процесори використовують кеші та TLB для підвищення продуктивності, використовуючи принцип локальності. Цей принцип стверджує, що процесор часто звертатиметься до одного й того ж набору місць пам'яті протягом певного періоду, зосереджуючись на адресах пам'яті, розташованих близько одна до одної, таких як цикли в програмі. Коли процес виконується на CPU1 і перемикається на CPU2, кеш і TLB необхідно оновити. Переміщення процесу з одного CPU на інший, відоме як міграція, вимагає оновлення кешу та TLB. Хоча моделювання не буде зосереджено безпосередньо на проблемах кешу та TLB, воно відстежуватиме частоту міграцій процесів, щоб отримати уявлення про те, як часто такі міграції можуть відбуватися [3]. На рисунку

2.1 зображено типову багатопроцесорну архітектуру, де кожен CPU має власний кеш, але пам'ять є спільною.

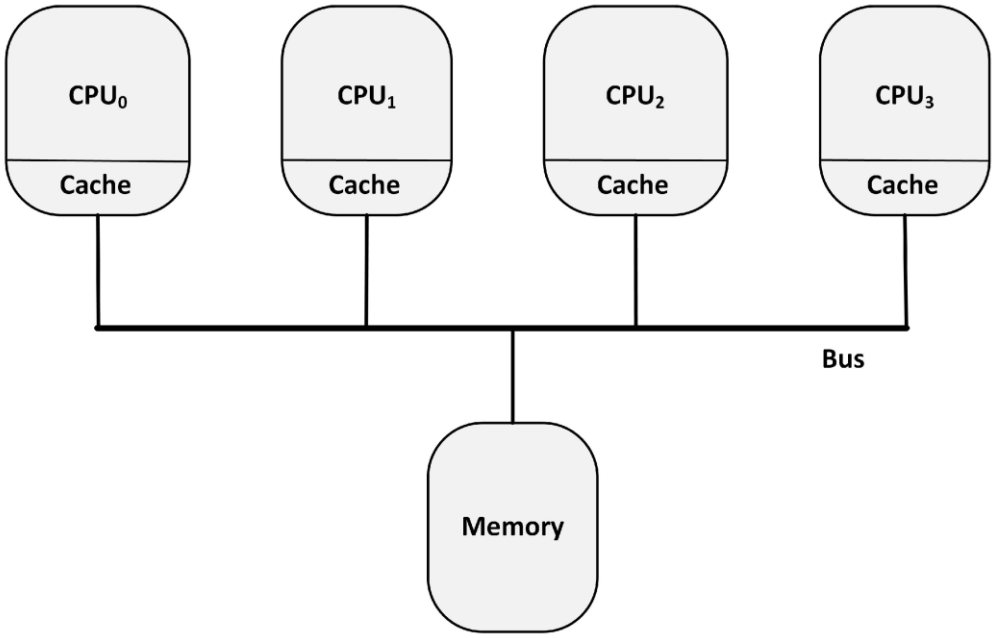

Рисунок 2.1 – Приклад спрощеної багатопроцесорної архітектури з чотирма процесорами та їхніми відповідними кешами, що спільно використовують пам'ять.

Існує два звичайні підходи до багатопроцесорного планування, обидва з яких пояснюються нижче.

Одна черга: під час переходу від планувальника з одним процесором до планувальника з кількома процесорами найпростіший підхід передбачає спільне використання однієї черги, до якої надходять усі процеси. Усі процесори використовують один і той самий алгоритм планування та відповідно вибирають завдання з черги. Хоча цей підхід простий для розуміння та легко реалізувати, він має суттєві недоліки. Основна проблема полягає в необхідності певної форми блокування, щоб запобігти запуску одного й того ж процесу двома процесорами. Введення блокування може значно знизити продуктивність, особливо з багатьма процесорами, оскільки кожен процесор повинен отримати блокування, знайти правильний процес та зняти блокування. Ці додаткові накладні витрати є небажаною властивістю.

Ще однією проблемою підходу з однією чергою є спорідненість кешу, оскільки процеси можуть виконуватися на іншому процесорі щоразу, коли вони плануються. На рисунку 2.2 показано, як може виглядати виконання за допомогою налаштування з однією чергою та чотирма процесорами.

Багато черг: підхід, який мінімізує конкуренцію за отримання блокувань, полягає у використанні черги для кожного процесора в системі. Якщо кількість процесорів дорівнює n, буде n черг. Коли завдання надходить у систему, воно зазвичай розміщується в черзі з найменшою кількістю завдань. Кожен процесор може планувати завдання у своїй локальній черзі, не турбуючись про спільне використання та синхронізацію. Цей підхід також допомагає зі спорідненістю кешу, оскільки завдання не перескакують між різними процесорами, а залишаються узгодженими на одному й тому ж. Однак цей підхід має недоліки, особливо з точки зору балансування навантаження. Наприклад, якщо два процесори мають загалом три завдання, де один процесор має одне завдання в черзі, а другий процесор має два завдання в черзі, то завдання в одному процесорі виконуватиметься більше, ніж два завдання в черзі для другого процесора. Ще одним недоліком багаточергового підходу є необхідність перерозподілу, якщо один процесор простоює, а інші мають кілька завдань у своїх чергах. На рисунку 2.3 показано, як може виглядати виконання в багаточерговій конфігурації з двома процесорами.

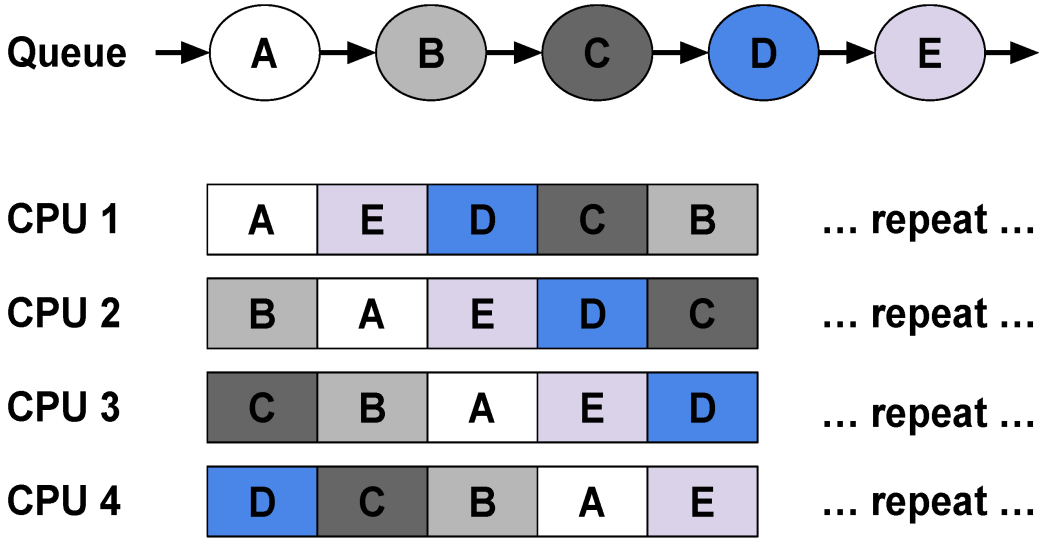

Рисунок 2.2 – Приклад багатопроцесорної системи з п'ятьма завданнями, що використовує єдину чергу.

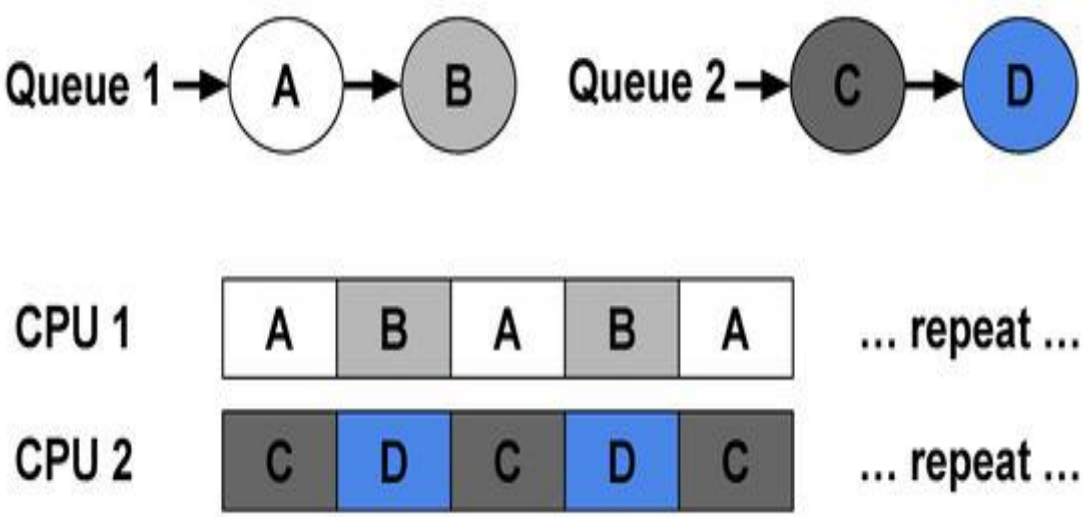

Рисунок 2.3 – Приклад багатопроцесорної системи з чотирма завданнями, що використовує багаточергову конфігурацію.

3 МЕРЕЖІ ПЕТРІ ТА GPENSIM

У цій роботі для моделювання AGV використовуються мережі Петрі, а точніше, модульні мережі Петрі.

3.1 Мережі Петрі

Мережа Петрі складається з двох елементів: місць та переходів. Перехід представляє подію або дію, що виконується об'єктом; подією може бути щось: змішування смузі, поєднання двох елементів, розбирання та заряджання. Місце представляє стан або позицію в системі. Наприклад, чи доступна машина, або яку відстань проїхав автомобіль. Дуга з'єднує місця та переходи; оскільки мережі Петрі є двочастинним графом, дуга не може з'єднувати елементи одного типу (наприклад, одне місце з іншим місцем). Токен представляє об'єкт, який переходить з одного місця в інше через спрацьовування переходу.

Мережа Петрі (P/T-мережа Петрі) – це чотиримерна мережа [21, 22]

$$PTPN = (P, T, F, M_0),$$

де,

P – множина місць, $P=\{p_1,p_2,\ldots,p_{np}\}$

T – набір переходів, $T=\{t_1,t_2,\ldots,t_{nt}\}$.

$P \cap T = \emptyset$.

F – множина спрямованих дуг; $F \subseteq (P \times T) \cup (T \times P)$. Вага дуги за замовчуванням, W, f_{ij} ($f_{ij} \in F$, дуга, що йде від p_i до t_j або з t_i до p_j) є однотонним, якщо не зазначено інше.

M – вектор-рядок позначок (токенів) на множині позицій.

$M=[M(p_1),M(p_2),\ldots,M(p_{np})] \in N^{n_p}$, M_0 це початкове маркування.

3.2 Модульні мережі Петрі

Модульна мережа Петрі — це мережа Петрі, яка складається з одного або кількох модулів Петрі та нуля або кількох міжмодульних з'єднувачів (IMC). Модуль Петрі подібний до будь-якої іншої мережі Петрі, але з деякими специфічними правилами:

- токени можуть потрапляти до модуля Петрі лише через його вхідні порти, які є переходами;
- токени можуть виходити з модуля Петрі лише через його вихідні порти, які також є переходами;
- локальні позиції та переходи модуля Петрі не можуть мати жодних прямих дуг з елементами поза модулем Петрі.

Модульні мережі Петрі мають дві важливі переваги порівняно з немодульними мережами Петрі [23], модульні мережі Петрі дозволяють незалежну розробку та тестування модулів Петрі. Після ретельного тестування окремих модулів Петрі вони з'єднуються за допомогою IMC для формування загальної моделі. Модулі Петрі можуть працювати на різних комп'ютерах. Отже, вони працюють швидше (вимагають менше часу на моделювання).

3.3 GPenSIM

У цій роботі для реалізації та моделювання моделі мережі Петрі використовується GPenSIM. GPenSIM дозволяє накладати логіку моделі на переходи через файли препроцесора та постпроцесора [24]. Також GPenSIM підтримує використання ресурсів, забезпечуючи компактну модель мережі Петрі для систем з великою кількістю ресурсів. Ресурс являє собою щось, що потрібно для роботи машини; наприклад, робот-маніпулятор може бути ресурсом.

GPenSIM — це набір інструментів MATLAB, GPenSIM знаходиться у

вільному доступі. Функція розфарбовування GPenSIM корисна для розробки моделей мереж Петрі для реальних дискретних систем [25]. Крім того, GPenSIM дозволяє моделювати великі дискретні системи за допомогою модульних мереж Петрі/

4 ПРОЄКТУВАННЯ СИСТЕМИ

У цьому розділі розглядається загальний архітектурний дизайн, модульний підхід та використання шаблонів проєктування та архітектурних шаблонів для організації системи.

4.1 Загальний дизайн

Під час моделювання різних алгоритмів планування в багатопроцесорному середовищі достатньо спрощеної системи порівняно зі складною операційною системою, такою як Windows або Linux. Ми пропустили з нашої моделі такі функції, як управління пам'яттю, операції вводу-виводу та деталі виконання програм. Виключення цих елементів було навмисним вибором, зумовленим їхньою складністю. Ми зосереджуємося виключно на мінімально необхідній інформації для обчислення метрик та належного планування завдань на їх задану тривалість. Методи, описані нижче, є важливими для полегшення моделювання різних планувальників.

Генерація завдань, перехід у мережі Петрі генеруватиме завдання з використанням певних значень. Характеристики завдань також залежатимуть від використаного алгоритму планування. Завдання можуть бути згенеровані випадковими, статичними або на основі зовнішніх наборів даних методами. Згодом усі згенеровані завдання будуть розміщені в глобальній черзі, доступній для всіх процесорів.

Черги, щойно згенеровані завдання розміщуються в глобальній черзі відповідно до описаної раніше процедури. Кожен процесор оснащений власною виділеною локальною чергою. Для системи, що складається з n процесорів, буде n локальних черг, що призведе до загальної кількості n+1 черги, включаючи глобальну чергу. Завдання переносяться з глобальної черги до локальної черги з найменшою кількістю завдань. У сценаріях, коли

кілька локальних черг мають однакову мінімальну кількість завдань, рішення щодо того, яка локальна черга отримає завдання, залишається за GPenSIM. В результаті кількість завдань у кожній локальній черзі залишається відносно постійною.

Перерозподіл завдань, за наявності простоюючого процесора він повинен мати можливість перенести завдання з однієї локальної черги до іншої. Міграція завдань залежить від відсутності завдань у глобальній та локальних чергах. За цих умов завдання з іншої локальної черги повернеться до глобальної черги та зазнає справедливого перерозподілу.

Планування, ЦП вибирають виконувані завдання з відповідних локальних черг для виконання протягом заданого періоду часу. Алгоритм планування залишається однаковим для всіх ЦП, зосереджуючись на завданнях у своїй локальній черзі. Вибір конкретного завдання залежить від реалізованого планувальника. Деякі алгоритми планування можуть вимагати додаткових полів у завданнях для полегшення правильного вибору наступного завдання.

Час, кожне завдання охоплює кілька полів, необхідних для розрахунків метрик, залежно від наявності доступу до часу. GPenSIM використовує годинник, який називається глобальним таймером, дискретне значення, що збільшується на 1/4 найкоротшого переходу випалу [24]. Глобальний таймер у GPenSIM буде використовуватися для призначення часу завданням. Обґрунтування цього вибору буде пояснено пізніше, коли ми заглибимося в переваги та недоліки використання цього вбудованого глобального таймера.

4.2 Модульний підхід

Багатопроцесорне планування – це складна тема, і створення комплексного модуля Петрі, який функціонує безперебійно, не є простим завданням. Обрана нами стратегія передбачає поступовий підхід. Ми починаємо з базової моделі з одним процесором, що використовує

найпростіший алгоритм планування, FCFS. Після того, як модуль з одним процесором продемонструє точну функціональність, ми переходимо до моделювання складніших алгоритмів планування в послідовному порядку: SJF → RR → MLFQ → CFS. Після успішної реалізації цих алгоритмів планування наша увага переходить до багатопроцесорного планування. Заключний етап включає зосередження на аспекті перерозподілу.

4.2.1 Один процесор

Початковий підхід передбачає моделювання одного процесора з мінімальними вимогами, еквівалентного планувальнику FCFS. Метою було створити модель з двома модулями — одним для локальної черги та іншим для процесора — та призначити всі обов'язки планування одному переходу в модулі локальної черги. Це гарантує, що загальний потік моделі залишається узгодженим, незалежно від реалізованого алгоритму планування. Початкова модель, зображена на рисунку 4.1, складається з двох модулів: Локальна черга та Процесор. Крім того, вона включає міжмодульні з'єднувачі (IMC), які інтегрують різні модулі, що дозволяє здійснювати моделювання.

Під час використання модулів Петрі є чотири окремі елементи [23]:

- вхідні порти: це переходи, через які токени можуть надходити в модуль з IMC, такі як tCpuQueue в модулі локальної черги на рисунку 4.1;
- вихідні порти, це переходи, де токени можуть переходити з модуля до певного місця в IMC, як-от tInitJob у модулі локальної черги на рисунку 4.1;
- локальні переходи, вони доступні лише зсередини модуля, оскільки не є вхідними чи вихідними портами.
- локальні місця, вони доступні лише зсередини модуля, оскільки вони не можуть отримувати токени від зовнішнього переходу, а зовнішній перехід не може брати токени від них.

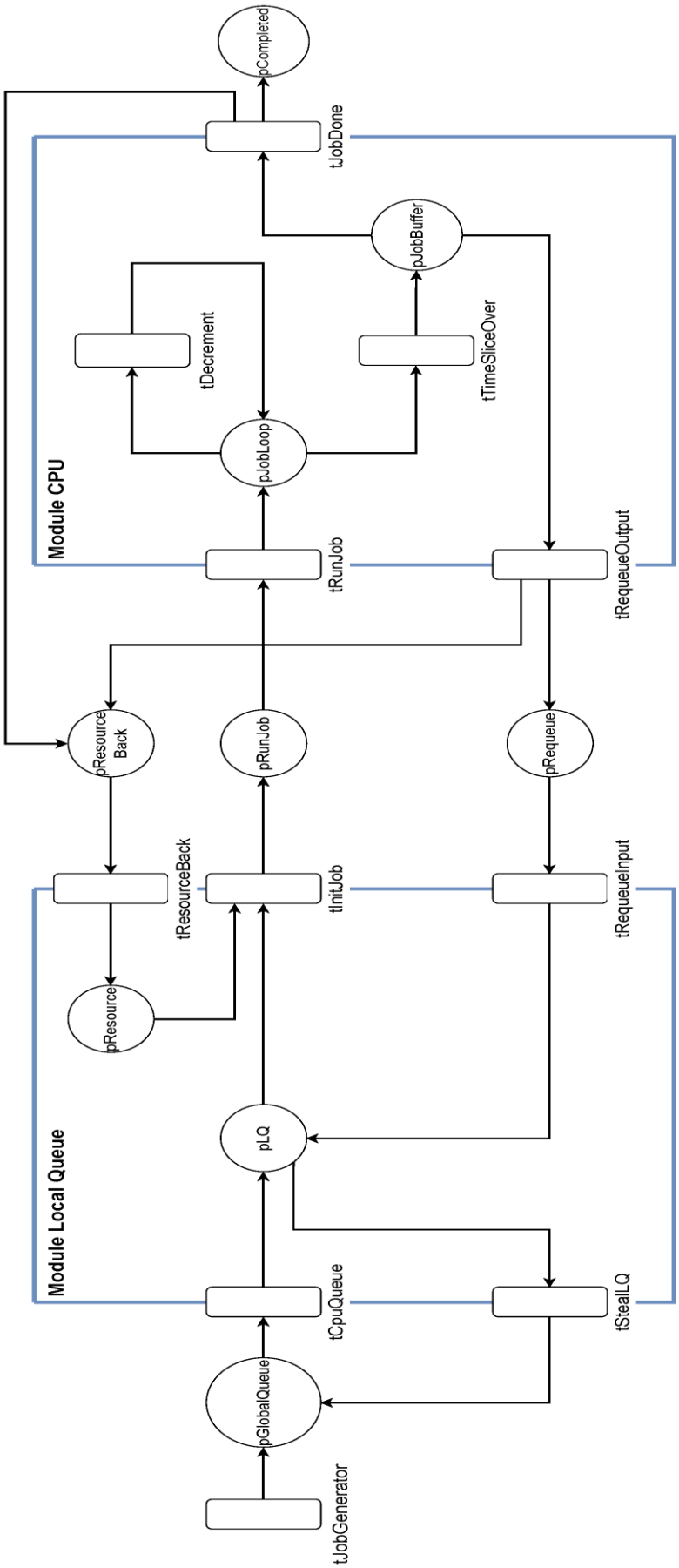

Рисунок 4.1 – Використання місця як ресурсу для запобігання одночасному виконанню кількох завдань у моделі.

Вхідні та вихідні порти на рисунку 4.1 є переходами, що перекриваються між модулем та IMC.

Потік системи, модель, описана в пунктах нижче.

Перший перехід у моделі, tJobGenerator, відповідає за розміщення токенів усередині місця pGlobalQueue, кольором яких відповідає його ідентифікатор завдання. Усі завдання, що надходять до pGlobalQueue, очікують на розміщення в місці pLQ. Хоча місце pGlobalQueue не є необхідним для одного процесора, воно є компонентом багатопроцесорної моделі, яка незабаром з'явиться.

Локальна черга pLQ – це місце, де алгоритм планування вибирає наступне завдання для планування. Перехід tStealLQ деактивований в однопроцесорній моделі, але використовується в багатопроцесорній моделі для перерозподілу завдань. Планувальник MLFQ містить додатковий перехід tBoostLQ , який підвищує швидкість завдань, що перебували в локальній черзі протягом тривалого періоду часу без планування.

Перехід tInitJob – це місце, де виконується алгоритм планування, щоб визначити наступне завдання для планування. Крім того, є місце pResource , яке використовує tInitJob , що гарантує, що в будь-який момент часу може виконуватися лише одне завдання. Без цього місця наявність кількох завдань у циклі завдань була б можливою. Це місце забезпечує взаємне виключення pJobLoop.

Після прибуття завдання до розташування pJobLoop, воно виконуватиметься протягом заданого інтервалу часу, визначеного реалізованим алгоритмом планування. Якщо інтервал часу не дорівнює нулю, перехід tDecrement виконає завдання протягом 1 одиниці часу та зменшить інтервал часу на 1 одиницю часу. Коли інтервал часу досягне нуля, виконається перехід tTimeSliceOver .

Якщо завдання не завершено, воно повертається до позиції pLQ та очікує перепланування. Після завершення завдання перехід tTimeSliceOver призначає токену колір «Готово». Перехід tJobDone перевіряє, чи мають якісь

токени в pJobBuffer колір «Готово», що означає завершення завдання та дозволяє розміщення в позиції pCompleted.

Після того, як завдання знову поміщається в чергу або виконується, отриманий ресурс у pResource повертається, що спрощує планування нового завдання.

Оновлена модель: модель, детально описана вище, використовує єдиний токен на місці pResource для емуляції спільного ресурсу, запобігаючи одночасному виконанню кількох завдань. GPenSIM має вбудований механізм ресурсів, спеціально розроблений для таких сценаріїв. Замість делегування розподілу ресурсів мережі Петрі, вища абстракція в GPenSIM бере на себе цю відповідальність. Рисунок 4.2 зображує ту саму модель, що й попередній розділ, але з використанням вбудованих ресурсів GPenSIM. Перехід tInitJob запитує доступ до процесора та має дозвіл на виконання лише за наявності ресурсу процесора; в іншому випадку він очікує та повторює спробу пізніше. Переходи tRequeueInput та tJobDone звільняють ресурс процесора, дозволяючи планувати інше завдання на процесорі. Все інше на рисунку 4.2 відповідає тому ж порядку дій, що й попередня модель.

Незалежно від реалізованого алгоритму планування, загальна модель залишається незмінною. Кожен алгоритм використовує ті самі модулі, як показано на рисунку 4.2. Логіка планування вбудована у файли препроцесора та постпроцесора. Переходом, відповідальним за рішення щодо планування, є tInitJob, який керує плануванням завдань на основі реалізованого алгоритму. Конкретні алгоритми планування потребують додаткових параметрів для прийняття рішень, які або включаються до основного файлу моделювання, або додаються до завдань, або і те, й інше.

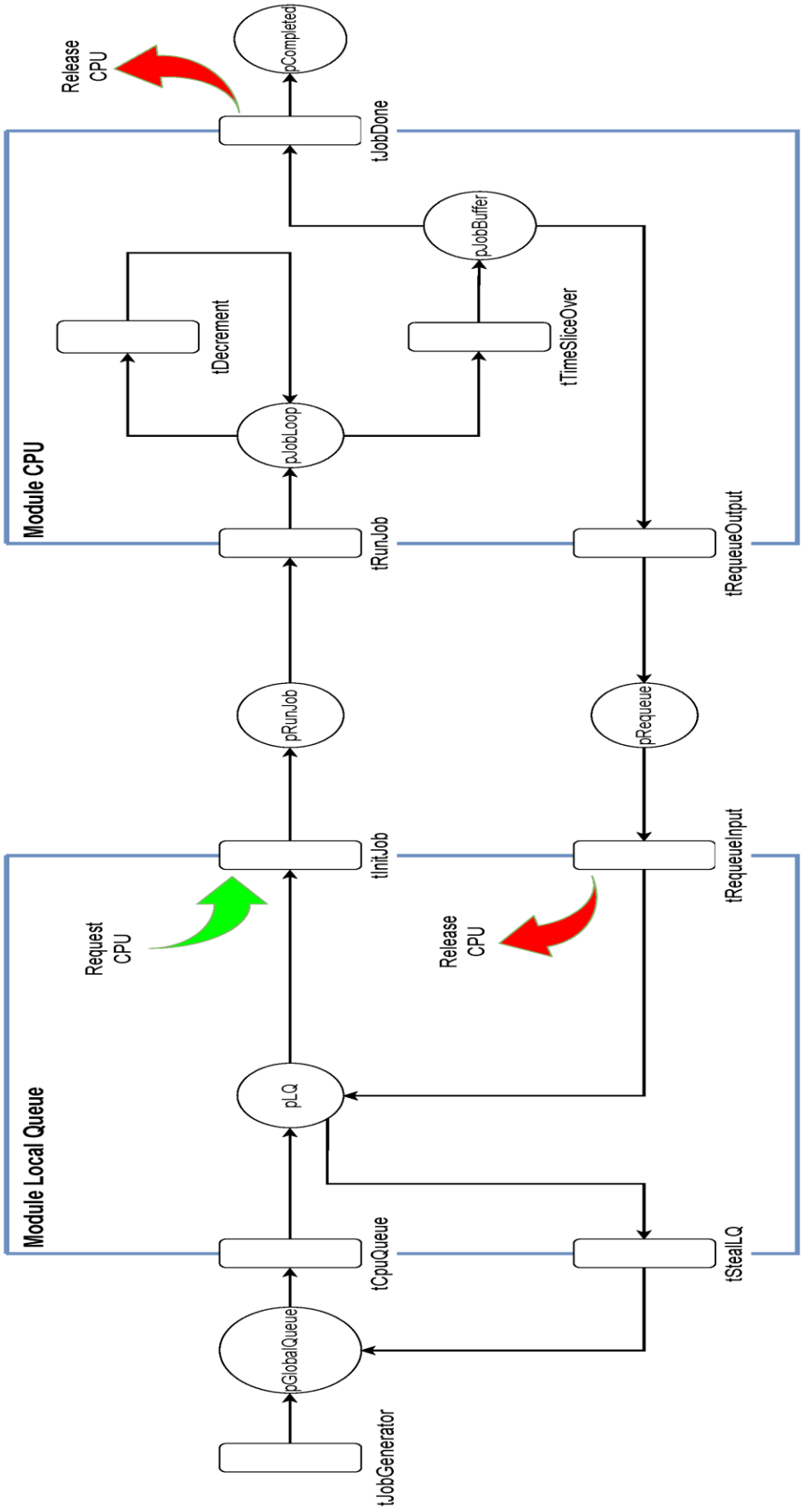

Рисунок 4.2 – Використання вбудованого ресурсу GPenSIM для спрощення моделі

4.2.2 Багатопроцесорний

Модель, зображена на рисунку 4.2, розроблена для одного процесора та ефективно функціонує в цьому контексті. Для моделювання алгоритмів планування в багатопроцесорному середовищі оригінальну модель необхідно реплікувати, щоб включити стільки екземплярів, скільки процесорів є в системі. Наприклад, якщо є чотири процесори, має бути чотири моделі, кожна з власним модулем локальної черги та модулем процесора. На рисунку 4.3 показано конфігурацію моделі з чотирма процесорами. Дублювання компонентів pGlobalQueue та tJobGenerator не є необхідним, оскільки вони повинні бути спільними для процесорів для розподілу завдань.

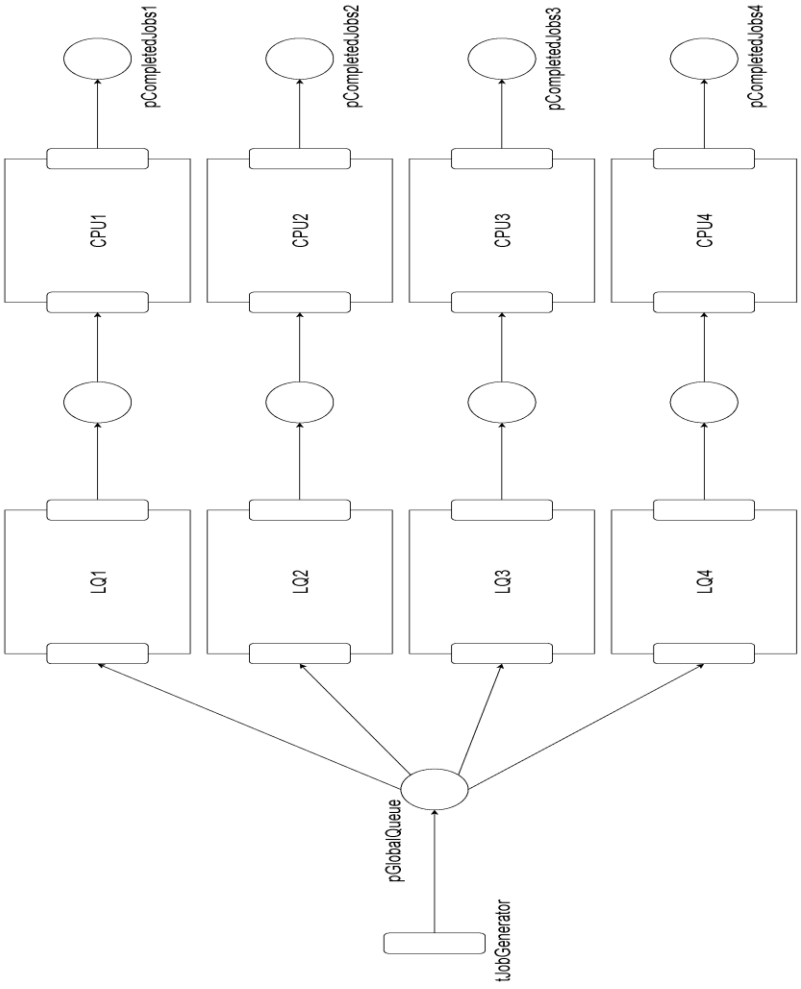

Рисунок 4.3 – Спрощений огляд чотирьох процесорів з використанням моделі з рисунка 4.2

Загальний потік багатопроцесорної моделі залишається узгодженим з однопроцесорною моделлю, з двома переходами, що потребують модифікації: tCpuQueue та tStealLQ. В середовищі з одним процесором tCpuQueue може завжди спрацьовувати, а tStealLQ може взагалі не спрацьовувати. Однак ці переходи є критично важливими для розподілу та перерозподілу завдань у багатопроцесорному середовищі. Перехід tCpuQueue для процесора повинен спрацьовувати лише тоді, коли його локальна черга має найменшу кількість завдань. Це забезпечує справедливий розподіл завдань, мінімізуючи час простою системи. У сценаріях, коли час виконання завдань змінюється, один процесор може накопичувати кілька завдань, тоді як інший залишається без завдань. У такому сценарії перехід tStealLQ повинен взяти завдання з pLQ та повернути його до pGlobalQueue , що дозволить перерозподіл на простоюючий процесор.

Моделювання написано в MATLAB з використанням платформи General-purpose Petri Net Simulator (GPenSIM). Кожен алгоритм планування реалізовано окремо, але має значну кількість спільного коду для оптимізації процесу розробки. У цьому розділі розглядається реалізація генерації завдань, як ми оновлюємо дані завдань під час моделювання, а також як кожен процесор запитує та вивільняє вбудовані ресурси GPenSIM.

4.3 Тестування та результати

Набори даних, що використовуються в симуляціях, складаються з двох різних наборів:

- набір даних коротких завдань, цей набір даних складається із завдань з відносно коротким часом виконання. Ці завдання призначені для імітації процесів, які швидко завершуються.

- змішаний набір даних завдань, навпаки, «змішаний набір даних завдань» забезпечує складніше та реалістичніше тестове середовище. Він моделює сценарії, де процесор повинен керувати поєднанням

короткострокових та довгострокових завдань, що дуже нагадує реальні ситуації, де процеси з різним часом виконання постійно конкурують за системні ресурси. Цей набір даних допомагає нам оцінити, наскільки добре наші алгоритми планування адаптуються до викликів, що виникають через різноманітний набір завдань.

Функція генерує короткі завдання, використовуючи нормальний розподіл, $N(\mu,\sigma)$, з μ як середнє значення та σ як стандартне відхилення. Для розрахунку загального часу виконання короткої роботи ми встановлюємо μ=25 і σ=10, зі значеннями, обмеженими [1,50] Довгі завдання створюються за допомогою різних μ і σ цінності, зокрема μ=100, σ=25, і обмежено [50, 150]. Кожен набір даних містить 250 завдань, причому короткий набір даних містить виключно короткі завдання. Змішаний набір даних складається з рівної кількості 50% короткий і 50% довгі завдання. У змішаних наборах даних короткі завдання представлені непарними ідентифікаторами завдань, тоді як парні ідентифікатори завдань відповідають довгим завданням. На рисунку 4.4 наведено візуалізацію розподілу часу виконання завдань для обох наборів даних.

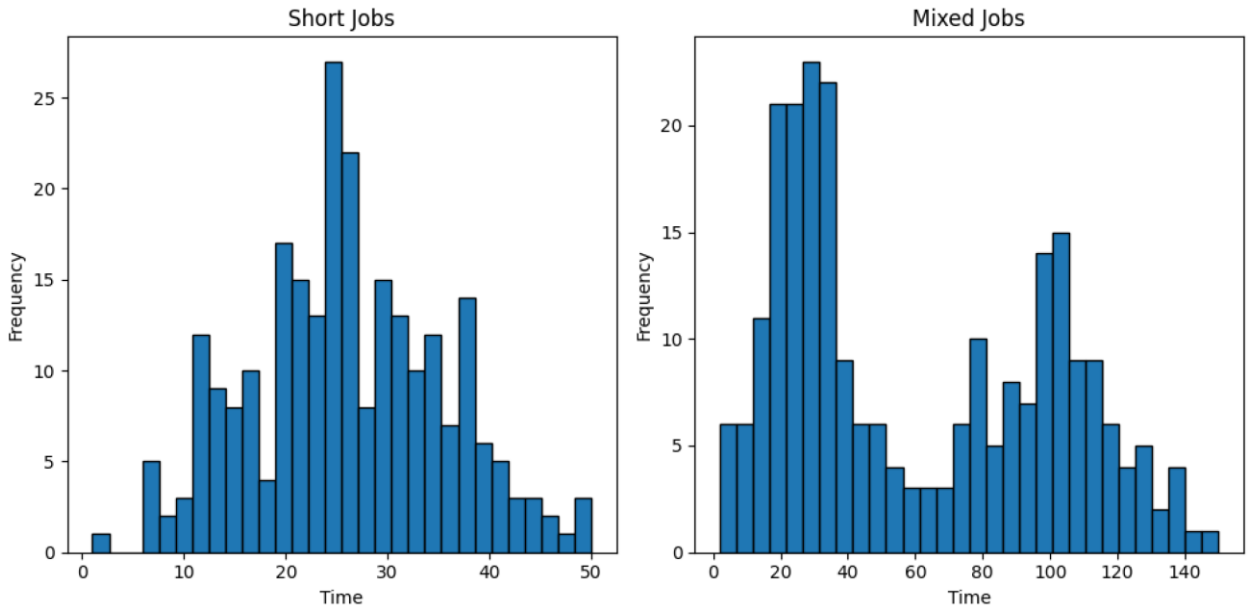

Рисунок 4.4 – Огляд часу виконання завдання: короткий набір даних проти змішаного набору даних.

Параметр якості для кожної роботи генерується за допомогою нормального розподілу з $N(0,6)$ і обмежений $[-20,19]$. Оскільки очікується, що більшість робіт матимуть коефіцієнт приємності 0, нормальний розподіл з μ=0 добре підходить. Стандартне відхилення σ=6 використовується для введення варіативності між завданнями, присвоєння вищого пріоритету одним завданням і нижчого пріоритету іншим.

Останній крок у створенні набору даних включає встановлення часу прибуття завдань. Для реалізації ми рівномірно призначили час прибуття в діапазоні$[1, (\sum^{n}_{i=1} T(i,total)/6]$, за винятком перших 10 завдань, час надходження яких дорівнює 0. Ці початкові завдання слугують фазою запуску. Поділ загального часу виконання на 6 враховує наявність чотирьох процесорів, що фактично зменшує загальний час приблизно на 4. Ми збільшили його на 2, щоб завдання надходили ближче одне до одного. Завдання сортуються за часом надходження, тобто для завдання з ідентифікатором і його час надходження задовольняє $i-1 \leq i \leq i+1$.

Ми використовуємо два набори даних для моделювання різноманітних робочих навантажень, всебічно оцінюючи, як алгоритми планування працюють за різних умов. Такий підхід дозволяє нам визначити, чи планувальник переважно працює з короткими завданнями, чи добре адаптується до змішаного навантаження, що включає як інтерактивні, так і ресурсомісткі завдання.

У цьому розділі оцінюються різні алгоритми планування з використанням метрик, описаних раніше. Ми починаємо з розгляду середніх метрик для кожного алгоритму як у коротких, так і в змішаних наборах даних. Щоб забезпечити повне уявлення про продуктивність, ми представляємо коробкові діаграми, що ілюструють медіану, стандартні відхилення та викиди. Після цього ми заглиблюємося в конкретні метрики, що охоплюють час обробки, час відгуку та пропускну здатність обох наборів даних. Нарешті, ми оцінюємо балансування навантаження процесора.

Середня продуктивність. В роботі представлено комплексну оцінку

різних алгоритмів планування з використанням метрик, визначених раніше в цьому дослідженні. Беручи до уваги як короткі, так і змішані набори даних, ми спочатку аналізуємо середні показники продуктивності для кожного алгоритму планування. Час виконання, час відгуку та пропускна здатність обчислюються як середні значення, тоді як індекс справедливості Джайна обчислюється за допомогою рівняння. Поле migrations представляє загальну кількість міграцій, що відбулися під час моделювання, тоді як scheduled – загальну кількість перемикань контексту.

Час виконання, середній час виконання значно нижчий у FCFS та SJF порівняно з іншими алгоритмами планування. Цей результат був очікуваним, оскільки FCFS та SJF виконували кожне завдання до завершення без будь-якої форми перемикання контексту. На прямокутних діаграмах, зображених на рисунках 4.5 та 4.6, SJF демонструє значні відхилення в часі виконання, що пояснюється його практикою пріоритезації коротких завдань та затримки виконання довгих. На противагу цьому, RR, MLFQ та CFS демонструють набагато ближчу відповідність середнього часу виконання, оскільки вони виділяють часовий інтервал для кожного завдання для виконання. Підвищене стандартне відхилення, що спостерігається для CFS на прямокутних діаграмах, можна пояснити рівнем якості, який призначає вищий пріоритет певним завданням. Навпаки, за винятком MLFQ, інші алгоритми планування призначають однаковий пріоритет кожному завданню. Варто підкреслити, що SJF може бути доведено оптимальним, коли всі завдання надходять одночасно [3]. Таким чином, SJF служить цінним орієнтиром при порівнянні інших алгоритмів планування з точки зору часу виконання.

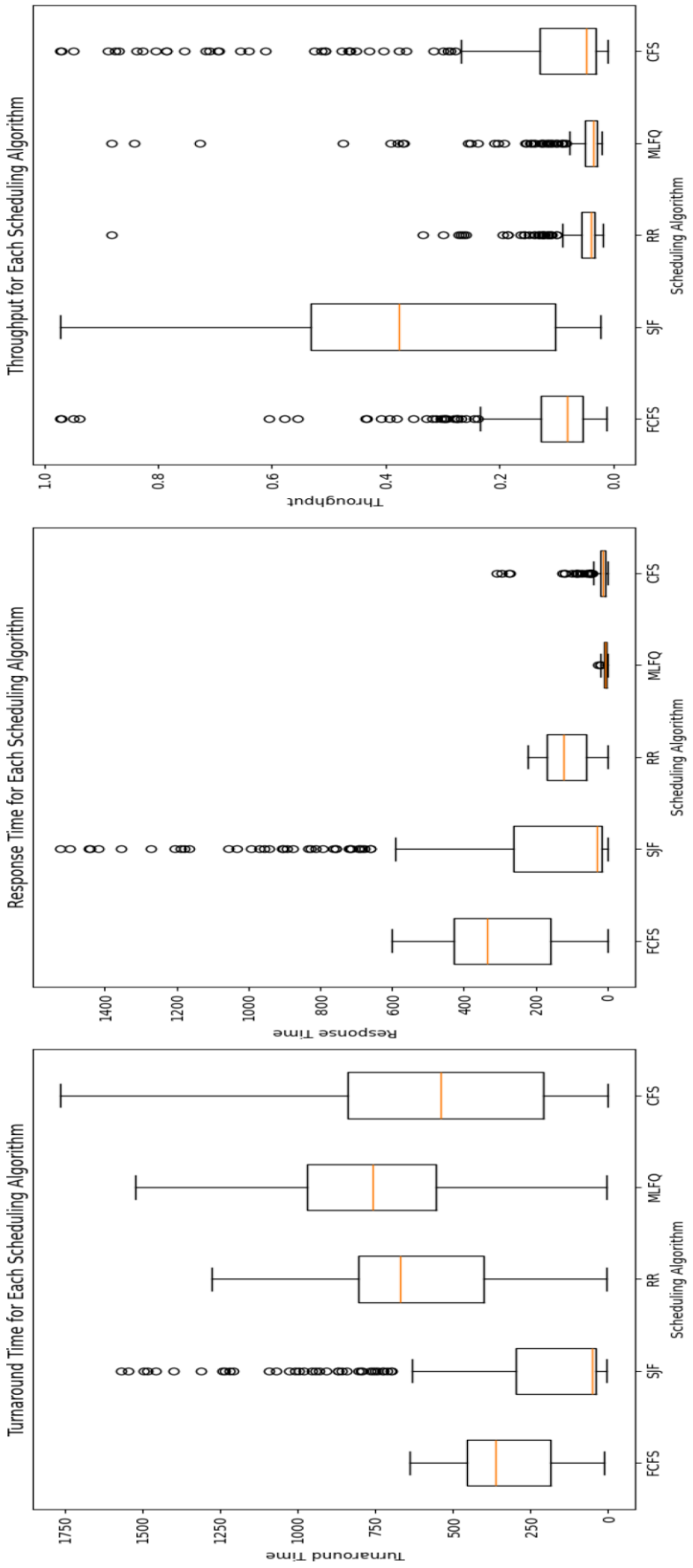

Рисунок 4.5 – Коробчаста діаграма показників з короткими завданнями.

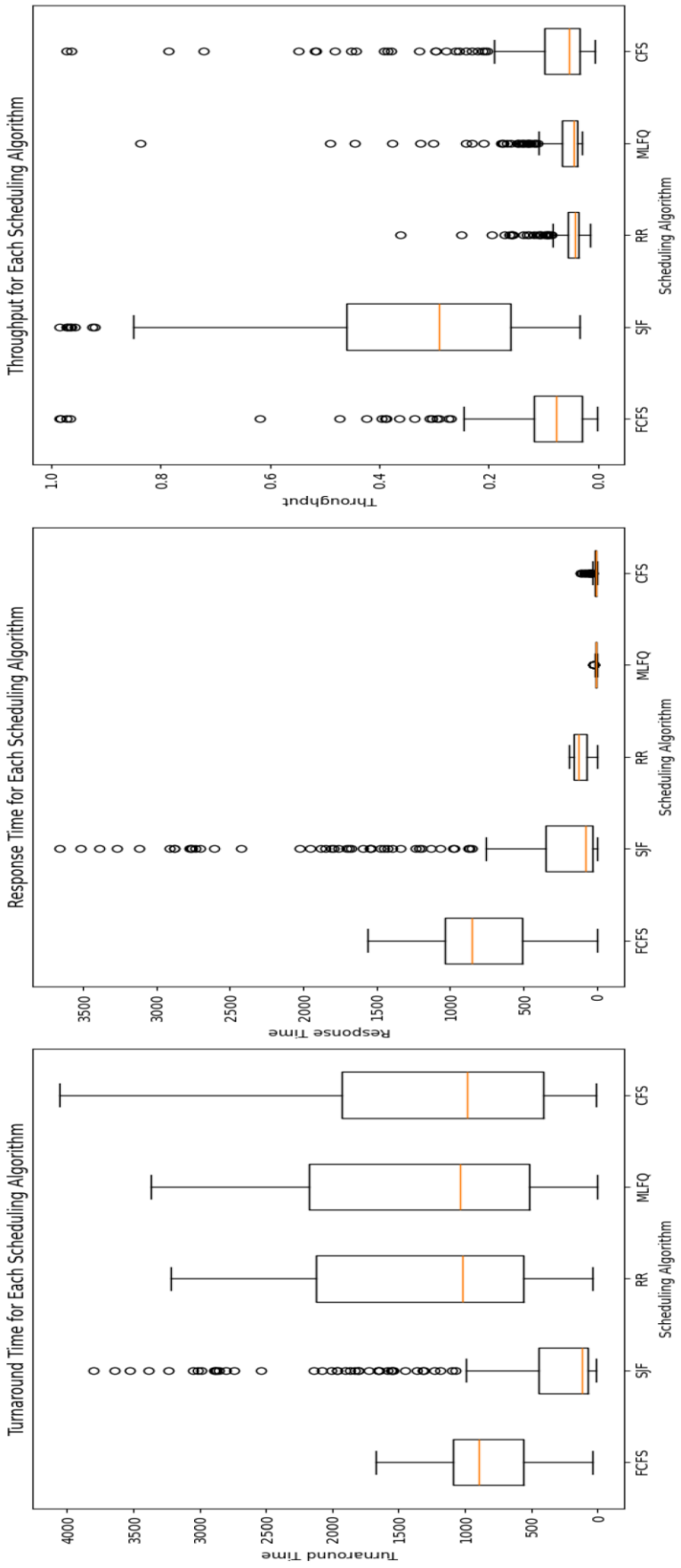

Рисунок 4.6 – Коробчаста діаграма показників зі змішаними завданнями.

Час відгуку, середній час відгуку для MLFQ та CFS перевищує середній час відгуку інших алгоритмів планування, зокрема FCFS та SJF. Хоча можна було б очікувати, що RR демонструватиме кращий час відгуку завдяки послідовному перемиканню контексту на заданому кванті часу, механізму черг та пріоритезації завдань у порядку «перший прийшов – перший вийшов», що призводить до того, що пізніші завдання розміщуються в кінці черги та виникають затримки, MLFQ та CFS, використовуючи певну форму пріоритету, часто дозволяють новонадійшовшим завданням виконуватися на більш ранній стадії, що сприяє загальному покращенню часу відгуку системи. Як SJF, так і CFS демонструють більше викидів у часі відгуку, як видно на блокових діаграмах на рисунках 4.5 та 4.6.

Пропускна здатність, оскільки пропускна здатність розраховується як час виконання, поділений на загальний час для даного завдання, бажаними є планувальники з високою пропускною здатністю. Однак важливо зазначити, що пропускна здатність, особливо для SJF, може бути дещо спотворена через те, що певні завдання переживають тривалі періоди голодування. Коробчасті діаграми надають корисне уявлення про те, як різні алгоритми планування працюють щодо пропускної здатності. Крім того, варто визнати, що значення, близькі до 1, можуть вказувати на перше завдання, що надійшло в локальній черзі, особливо для FCFS, SJF та CFS.

Індекс справедливості Джайна, за винятком SJF та RR зі змішаними завданнями, значення справедливості для більшості алгоритмів планування тісно пов'язані. Це свідчить про відносно рівномірний розподіл процесорів. Однак дещо несподіваним було те, що CFS мав найнижче значення справедливості. Озираючись назад, це спостереження узгоджується з очікуваннями, враховуючи, що рівень якості суттєво впливає на стандартне відхилення часу виконання, як обговорювалося раніше. Отже, цей вплив поширюється на пропускну здатність, фактор, який враховує індекс справедливості Джайна.

Міграції та заплановані міграції, у випадку FCFS та SJF немає

зареєстрованих міграцій, навіть якщо завдання могли переходити з одного процесора на інший. Розрахунок міграцій базується на виконанні завдань на процесорі; якщо завдання не виконується на двох різних процесорах, воно не вважається перенесеним. Хоча менша кількість міграцій загалом є кращою, планувальники, які розподіляють часові інтервали для завдань, таких як RR, MLFQ та CFS, повинні зіткнутися з міграціями. Важливо зазначити, що підрахунок кількості міграцій не враховує сценарії, коли завдання виконується на кількох процесорах послідовно, що може суттєво вплинути на продуктивність більше, ніж одна міграція.

Заплановані показники відображають кількість перемикань контексту під час симуляції, що відбуваються, коли завдання або завершено, або передано для планування нового завдання. Зі збільшенням кількості перемикань контексту час симуляції збільшується через необхідність переходів з часом спрацьовування 0,1 для кожного перемикання. Цей ефект стане більш помітним на наступних графіках. Отже, вимальовується помітна закономірність, що демонструє більший час виконання зі збільшенням перемикань контексту, як і слід було очікувати.

ВИСНОВКИ

В роботі було проведено тестування розробленленої платформу; після вивчення результатів ми можемо зробити висновок, що CFS та MLFQ виділяються як переважні алгоритми планування. Вони загалом добре працюють і не залежать від попередніх знань про час виконання завдань, що є нереалістичним і є недоліком для SJF. Хоча це справедливо, планувальник RR має притаманні проблеми, особливо при роботі з великою кількістю одночасних завдань у черзі. Це призводить до низької пропускної здатності та, в деяких випадках, низького часу відгуку.

Враховуючи, що CFS використовується в ядрі Linux, наші симуляції дають уявлення про його ефективність, демонструючи хороший час виконання та надзвичайно низький час відгуку. Крім того, він підтримує призначення пріоритету завданням, що дозволяє визначати пріоритети інтерактивних завдань.

Однак планувальник MLFQ стикається з такими труднощами, як визначення оптимальної кількості черг та часу їх виконання. Важливий механізм підвищення також створює труднощі з налаштуванням. Незважаючи на ці труднощі, як CFS, так і MLFQ пропонують цінну інформацію про різні сценарії, сприяючи всебічному розумінню їхніх сильних та обмежень.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Theis, T.N.; Wong, H.S.P. The End of Moore's Law: A New Beginning for Information Technology. Comput. Sci. Eng. 2017, 19, 41–50.

2. Arpaci-Dusseau, R.H.; Arpaci-Dusseau, A.C. Operating System; Three Easy Pieces. In Arpaci-Dusseau Books; CreateSpace Independent Publishing Platform: North Charleston, SC, USA, 2018.

3. Karger, D.R.; Stein, C.; Wein, J. Scheduling algorithms. In Algorithms and Theory of Computation Handbook; Chapman & Hall/CRC: Boca Raton, FL, USA, 1999; Volume 1, p. 20.

4. Mohammadi, A.; Akl, S.G. Scheduling Algorithms for Real-Time Systems; School of Computing Queens University: London, UK, 2005. Horn, W. Some simple scheduling algorithms. Nav. Res. Logist. Q. 1974, 21, 177–185. [Google Scholar] [CrossRef]

5. Hou, E.S.H.; Ansari, N. Genetic algorithm for multiprocessor scheduling. IEEE Trans. Parallel Distrib. Syst. 1994, 5, 113–120.

6. Wu, A.S.; Yu, H.; Jin, S.; Lin, K.C.; Schiavone, G. An Incremental Genetic Algorithm Approach to Multiprocessor Scheduling. IEEE Trans. Parallel Distrib. Syst. 2004, 15, 824–834.

7. Visalakshi, P. Multiprocessor Scheduling Using Hybrid Particle Swarm Optimization with Dynamically Varying Inertia. Int. J. Comput. Sci. Appl. 2007, 4, 95–106.

8. Stone, H.S. Multiprocessor Scheduling with the Aid of Network Flow Algorithms. IEEE Trans. Softw. Eng. 1977, SE-3, 85–93.

9. Coffman, E.G.; Garey, M.R.; Johnson, D.S. An Application of Bin-Packing to Multiprocessor Scheduling. Siam J. Comput. 1978, 7, 1–17.

10. Baruah, S.; Fisher, N. The partitioned multiprocessor scheduling of deadline-constrained sporadic task systems. IEEE Trans. Comput. 2006, 55, 918–923.

11. Leontyev, H.; Anderson, J.H. Generalized Tardiness Bounds for Global Multiprocessor Scheduling. In Proceedings of the IEEE International Real-Time Systems Symposium, Tucson, AZ, USA, 3–6 December 2007.

12. Guan, N.; Stigge, M.; Yi, W.; Yu, G. Fixed-Priority Multiprocessor Scheduling with Liu and Layland's Utilization Bound. In Proceedings of the 16th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2010, Stockholm, Sweden, 12–15 April 2010.

13. Kasahara, H.; Narita, S. Practical multiprocessor scheduling algorithms for efficient parallel processing. Syst. Comput. Jpn. 1999, 16, 11–duling with rejection.

14. Liu, S.; Quan, G.; Ren, S. On-line scheduling of real-time services with profit and penalty. In Proceedings of the 2011 ACM Symposium on Applied Computing, Tai Chung, Taiwan, 21–24 March 2011.

15. Langen, P.; Juurlink, B.; Juurlink, B. Leakage-Aware Multiprocessor Scheduling. J. Signal Process. Syst. 2009, 57, 73–88.

16. Squillante, M.S.; Lazowska, E.D. Using processor-cache affinity information in shared-memory multiprocessor scheduling. IEEE Trans. Parallel Distrib. Syst. 1993, 4, 131–143.

17. Veltman, B.B. Multiprocessor Scheduling with Communication Delays. Ph.D. Thesis, CWI, Eindhoven, The Netherlands, 1993.

18. Garcia, R.C.; Chung, J.M.; Jo, S.W.; Ha, T.; Kyong, T. Response time performance estimation in smartphones applying dynamic voltage & frequency scaling and completely fair scheduler. In Proceedings of the IEEE International Symposium on Consumer Electronics, Jeju, Republic of Korea, 22–25 June 2014.

19. Murata, T. Petri nets: Properties, analysis and applications. Proc. IEEE 1989, 77, 541–580.

20. Peterson, J.L. Petri Net Theory and The Modeling of Systems; Prentice Hall PTR: Upper Saddle River, NJ, USA, 1981.

21. Davidrajuh, R. Petri Nets for Modeling of Large Discrete Systems; Springer: Berlin/Heidelberg, Germany, 2021.

22. Davidrajuh, R. Modeling Discrete-Event Systems with GPenSIM; Springer International Publishing: Cham, Switzerland, 2018.

23. Davidrajuh, R. Colored Petri Nets for Modeling of Discrete Systems: A Practical Approach with GPenSIM; Springer Nature: Berlin/Heidelberg, Germany, 2023.

24. Jain, R. The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling; Wiley: Hoboken, NJ, USA, 1991.

25. Fan, L.; Cao, P. Summary cache: A scalable wide-area Web cache sharing protocol. IEEE/ACM Trans. Netw. 2000, 8, 281–293.