НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

КАФЕДРА СИСТЕМНОГО ПРОГРАМУВАННЯ І
СПЕЦІАЛІЗОВАНИХ КОМП'ЮТЕРНИХ СИСТЕМ

«На правах рукопису»
УДК 004.021

«До захисту допущено»

Завідувач кафедри СПСКС

_____Віталій РОМАНКЕВИЧ
(підпис) (ім'я, прізвище)

"___"_____2020р.

Магістерська дисертація

на здобуття ступеня магістра

зі спеціальності 123 Комп'ютерна інженерія

на тему: Програмна модель однопроцесорної системи реального часу ____

Виконав: студент ІІ курсу, групи КВ-92мп
Болгов Іван Михайлович _____ _____
(прізвище, ім'я, по батькові) (підпис)

Науковий керівник д.т.н., професор Володимир ЗАЙЦЕВ ___ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент _____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант з нормоконтролю доцент, с.н.с.,к.т.н. Юлія БОЯРІНОВА _____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2020 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра системного програмування і спеціалізованих комп'ютерних систем

Рівень вищої освіти – другий (магістерський)
за освітньо-професійною програмою
Спеціальність 123 Комп'ютерна інженерія (Системне програмування)

ЗАТВЕРДЖУЮ
Завідувач кафедри СПСКС
_____ Віталій РОМАНКЕВИЧ
(підпис)
«___»_____2020 р.

ЗАВДАННЯ
на магістерську дисертацію студенту
Болгову Івана Михайловичу

1. Тема дисертації «Програмна модель однопроцесорної системи реального часу», науковий керівник дисертації д.т.н., професор Володимир ЗАЙЦЕВ, затверджені наказом по університету від «___» _____ 2020 р. № _____

2. Термін подання студентом дисертації 10 грудня 2020 р._____

3. Об'єкт дослідження: виконання задач системи реального часу _____

4. Предмет дослідження: програмна модель виконання задач системи реального часу
_____

5. Перелік завдань, які потрібно розробити: реалізувати програмні моделі сітки Петрі, задачі реального часу, створити модель виконання задач для декількох задач, що мають часові характеристики, навести приклад роботи моделі та обґрунтувати результати. _____
_____

6. Перелік ілюстративного матеріалу: презентація _____
_____
_____

_____

_____

7. Перелік публікацій: _____

_____

_____

_____

8. Дата видачі завдання 5 вересня 2019 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1.	Вивчення літератури за тематикою дисертації	11.10.2019	
2.	Підготовка матеріалів першого розділу магістерської дисертації	16.09.2020	
3.	Підготовка матеріалів другого розділу магістерської дисертації	22.09.2020	
4.	Підготовка матеріалів третього розділу магістерської дисертації	26.09.2020	
5.	Підготовка матеріалів четвертого розділу магістерської дисертації	30.09.2020	
6.	Підготовка матеріалів п’ятого розділу магістерської дисертації	05.10.2020	
7.	Розробка моделі сітки Петрі	09.10.2020	
8.	Розробка моделі задачі реального часу	15.10.2020	
9.	Оформлення документації магістерської дисертації	21.11.2020	
10.	Попередній розгляд магістерської дисертації на кафедрі	26.11.2020	

Студент _____ Іван БОЛГОВ
(підпис)

Науковий керівник дисертації _____ Володимир ЗАЙЦЕВ
(підпис)

РЕФЕРАТ

Актуальність теми.

Часові характеристики систем реального часу є визначними показниками для проектування цієї системи. Існуючі методи оцінки часових характеристик систем реального часу не є досить універсальними та не гарантують отримання вірного результату. А отже, неможна засвідчити, що система буде працездатна.

Запропонований метод визначення часових характеристик за допомогою моделювання розподілу процесорного часу між задачами згідно обраних алгоритмів планувальників з використанням моделі сіток Петрі гарантує отримання достовірних показників для конкретного типу процесора.

Об’єктом дослідження є гарантоване визначення часових характеристик системи реального часу.

Предметом дослідження є метод визначення часових характеристик за допомогою моделювання розподілу процесорного часу між задачами згідно обраних алгоритмів планувальників з використанням моделі сіток Петрі.

Мета роботи є розробка алгоритму визначення часових характеристик за допомогою моделювання розподілу процесорного часу між задачами з використанням моделі сіток Петрі.

Методи дослідження. Моделювання системи реального часу з різними параметрами задач, запропонованих для виконання. Перевірка роботи системи за допомогою обраного планувальника задач.

Наукова новизна полягає у створенні моделі системи задач за допомогою сіток Петрі, що дозволяє моделювати систему з великою кількістю задач.

Практична цінність полягає у пришвидшені процесу розробки систем реального часу загалом та мінімізації випадків, коли помилковий розподіл процесорного часу має критичне значення, наприклад, коли вихід системи з ладу веде до серйозних наслідків таких як грошові втрати на відновлення або

непередбачену помилкову роботу системи, яка може привести до аварійних або небезпечних для здоров'я подій.

Структура та обсяг роботи. Магістерська дисертація складається з вступу, п'ятьох розділів, висновків та додатків.

У вступі представлена загальна характеристика роботи, проведена оцінка існуючих рішень та їх недоліків, обґрунтована необхідність розробки нового алгоритму, сформульована задача роботи.

У першому розділі дається визначення систем реального часу; описана теоретична частина проектування систем реального часу, види планувальників та проблеми моделювання.

У другому розділі описується метод визначення часових характеристик за допомогою моделювання розподілу процесорного часу між задачами згідно обраних алгоритмів планувальників з використанням моделі сіток Петрі, наводяться приклади роботи обраного методу та аналіз отриманих результатів.

У третьому розділі визначається програмне середовище для розробки алгоритму, вказуються переваги обраного програмного засобу, програмні компоненти, які використовуються у роботі алгоритму, але не мають прямого відношення до нього – модель сіток Петрі, способи швидкого помноження лінійних матриць, які використовуються в алгоритмі, система виводу результатів.

У четвертому розділі наведено алгоритм визначення часових характеристик за допомогою моделювання розподілу процесорного часу між задачами з використанням моделі сіток Петрі.

У п'ятому розділі наведено приклад роботи алгоритму на різних наборах даних та обробка результатів.

У висновках результати аналізуються, щоб затвердити чи є алгоритм ефективним при роботі системи реального часу.

Ключові слова системи реального часу, сітка Петрі, планувальник задач.

Abstract

Actuality of theme.

The time characteristics of real-time systems are outstanding indicators for the design of this system. Existing methods for estimating the time characteristics of real-time systems are not universal enough and do not guarantee a correct result. Therefore, it is not possible to certify that the system will be operational.

The proposed method of determining the time characteristics by modeling the distribution of CPU time between tasks according to the selected algorithms of schedulers using the model of Petri nets ensures reliable performance for a particular type of processor.

Object of study is a guaranteed determination of the time characteristics of the real-time system.

Subject of research is a method of determining the time characteristics by modeling the distribution of processor time between tasks according to the selected algorithms of schedulers using the model of Petri nets.

Purpose is the development of an algorithm for determining the time characteristics by modeling the distribution of CPU time between tasks using the model of Petri nets.

Research objective. Simulation of a real-time system with different parameters of the tasks proposed for execution. Check the operation of the system using the selected task scheduler.

Scientific novelty is to create a model of the problem system using Petri nets, which allows you to model a system with a large number of problems.

The practical value is to speed up the development of real-time systems in general and minimize cases where erroneous allocation of CPU time is critical, such as when system failure leads to serious consequences such as loss of money on recovery or unforeseen malfunction of the system, which can lead to disasters or unhealthy events.

Structure and scope of work. The master's thesis consists of an introduction, five chapters, conclusions and appendices.

The introduction presents the general characteristics of the work, evaluates the existing solutions and their shortcomings, substantiates the need to develop a new algorithm, and formulates the task.

The first section defines real-time systems; describes the theoretical part of real-time systems design, types of planners and modeling problems.

The second section describes the method of determining the time characteristics by modeling the distribution of processor time between tasks according to the selected algorithms of schedulers using the model of Petri nets, gives examples of the selected method and analysis of the results.

The third section defines the software environment for algorithm development, indicates the advantages of the selected software, software components used in the algorithm, but not directly related to it - the model of Petri nets, methods of rapid multiplication of linear matrices used in the algorithm, output system results.

The fourth section presents an algorithm for determining time characteristics by modeling the distribution of CPU time between tasks using a model of Petri nets.

The fifth section gives an example of how the algorithm works on different data sets and processing the results.

In the conclusions, the results are analyzed to determine whether the algorithm is effective in the real-time system.

Keywords real-time system, petri grid, task scheduler.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

АП – алгоритм планування.

ЛБ– лінійний блок.

ОС – операційна система.

ОС РЧ – операційна система реального часу.

СМО – система масового обслуговування.

СРЧ – система реального часу.

ASMP (asymmetric multiprocessing) – асиметрична багатопроцесорна обробка.

CSV (Comma-separated values) – формат файлів для представлення табличних даних, у якому значення відокремлюються символом коми та символом переходу на наступний рядок.

DMA (Deadline Monotonic Algorithm) – монотонний алгоритм термінів.

EDF (earliest deadline first) – алгоритм планування по найближчому терміну завершення.

FBS (Foreground-Background Scheduler) – планувальник переднього та фонового виконання, керований пріоритетами.

FIFO (first in, first out) – тип планування, при якому задачі виконуються у порядку надходження.

MQS (Multi Queue Scheduler) – планувальник декількох черг.

PI (process identificator) – ідентифікатор процесу.

RMA (Rate Monotonic Analysis) – монотонний планувальник частоти використання.

SMP (symmetric multiprocessing) – симетрична багатопроцесорна обробка.

TCB (Task Control Block) – дескриптор процесу.

ВСТУП

Згідно визначенню, системи реального часу (СРЧ) – це системи, для яких специфіковані вимоги щодо часу виконання задач. У залежності від особливостей цих вимог розрізняють м’які та жорсткі СРЧ. СРЧ - належать до класу жорстких (hard), якщо порушення встановленого часу виконання задач неприпустимо. СРЧ називають м’якою (light), якщо отримання результату після наперед встановленого строку призводить лише до зниження якості функціонування системи.

Системи реального часу використовуються у різноманітних сферах людської діяльності, наприклад, у системах керування технічними процесами на виробництві, системах зчитування, обробки та архівування інформації та робототехніці. Також системи реального часу відіграють роль у пристроях друку, таких як принтери і сканери, побутовій техніці, мережевому обладнанні: маршрутизатори, модеми, системи відеоконференцій, бортових системах автомобілів та літаків. Окремо слід зазначити вклад СРЧ у розвиток медичного обладнання. Наприклад, томографічні прилади та приліжкове моніторування стану пацієнта.

Зі збільшенням продуктивності процесорів системи реального часу виконують важливу роль в індустрії комп’ютерних ігор орієнтованих на мережевий зв'язок між декількома гравцями. В цьому випадку латентність СРЧ визначає кількість графічних помилок при синхронізації подій. Наприклад, визначення позиції одного гравця відносно іншого.

Основною проблемою в системах реального часу є таке планування задач, яке б забезпечувало прогнозовану поведінку системи за всіх можливих обставин.

При створенні СРЧ вважається, що розробка планувальника в СРЧ має дві складові: планувальник періоду розробки (off-line) та планувальник періоду виконання. Зупинимось на планувальнику періоду розробки. На етапі розробки СРЧ у розпорядженні розробника має бути інформація про прикладні задачі та особливості їх взаємодії. Створення планувальника періоду розробки базується на обробці проектної інформації до початку роботи системи. Планувальник періоду

виконання є складовою системи і починає діяти при її запуску. Алгоритм його роботи базується на інформації, яка отримана у результаті періоду розробки.

Планування періоду розробки включає аналіз можливості виконання вимог до СРЧ необхідність в якому виникає, коли обмеження на час виконання задач системи чітко сформульовані і порушення цього часу може призвести до виходу системи з ладу.

Успішне завершення аналізу дає гарантію того, що за різних обставин виконання всіх задач буде завершено у заданий час.

Дана робота відображає проблему визначення часових характеристик задач систем реального часу, успішність роботи яких залежить від часу, за який вони отримують результат. Визначення часових характеристик СРЧ на стадії проектування є складною задачею.

На даний момент існує два способи визначення цих характеристик:

1. Теоретичні розрахунки, направлені на отримання критеріїв здійсненності.

2. Моделювання роботи системи на статичних моделях масового обслуговування.

Але ні перший, ні другий метод не дає можливість отримати гарантований результат, що суттєво ускладнює процес проектування.

У цій роботі запропоновано метод оцінки часових характеристик задач в системах реального часу шляхом аналізу даних, отриманих моделюванням розподілу процесорного часу між задачами згідно обраних алгоритмів планувальника з використанням моделі сіток Петрі.

Метод гарантує отримання часових характеристик задач в разі обрання конкретного типу процесора і планувальника, що потрібно для початку технічного проектування системи реального часу.

1. ОСНОВНІ ОСОБЛИВОСТІ СИСТЕМ РЕАЛЬНОГО ЧАСУ

1.1 Визначення систем реального часу

Існує декілька визначень систем реального часу (СРЧ). Не можна вважати те чи інше визначення правильним або помилковим, так як визначну роль грає контекст в якому йдеться мова про систему реального часу.

Наведемо найпопулярніші визначення:

1. Тлумачний словник по обчислювальним системам дає наступне визначення. Система реального часу – це будь-яка система, в якій істотну роль грає час генерації вихідного сигналу. Це зазвичай пов'язано з тим, що вхідний сигнал відповідає якимось змінам у фізичному процесі, і вихідний сигнал повинен бути пов'язаний з цими ж змінами. Тимчасова затримка від отримання вхідного сигналу до видачі вихідного сигналу повинна бути невеликою, щоб забезпечити прийнятне час реакції.

2. Тлумачний словник з інформатики дає інше визначення. Режим реального часу режим обробки даних, при якому забезпечується взаємодія обчислювальної системи з зовнішніми по відношенню до неї процесами в темпі, порівнянній зі швидкістю протікання цих процесів.

3. Канонічним є визначення Дональда Гиллиєса з якого системою реального часу називається система, в якій успішність роботи будь-якої програми залежить не тільки від її логічної правильності, а й від часу, за який вона отримала результат. Якщо часові обмеження не задоволені, то фіксується збій в роботі систем.

4. Системами реального часу з маркетингових міркувань називають системи постійної готовності. Якщо програму називають працюючою в реальному часі, то це означає, що вона встигає обробляти запити від людини, для якої затримка непомітна.

5. Системи реального часу зв’язують з поняттям «швидка система». Але це не точна аналогія, тому що час затримки реакції СРЧ на подію уже не так і

важливий. Головне, щоб цього часу було досить для розглянутого додатку і гарантовано. Приналежність системи до класу СРЧ ніяк не пов'язана з її швидкодією.

Отже, найбільш доцільним для розглядання є канонічне визначення систем реального часу. Далі саме воно використовується в контексті цієї роботи.

1.2 Вимоги до систем реального часу

Існують два режими роботи системи реального часу:

- сильні (hard);
- слабкі (soft).

Для розглядання відмінностей цих режимів доцільно навести приклад. Нехай маємо конвеєр, який постійно рухається з певною періодичністю. У момент затримки оператор повинен виконати певну дію з об’єктом на конвеєрі.

Якщо така система працює в режимі сильного реального часу, то невиконання операції у відповідний момент часу призведе до критичного порушення роботи системи. Після цього неможливо відновити СРЧ та продовжити виконання.

Якщо йдеться мова про слабкий режим реального часу, помилка призведе до втрати продуктивності, але система відреагує на це та продовжить роботу.

Більшість програмного забезпечення (ПЗ) працює в режимі слабого реального часу при цьому СРЧ має додатковий рівень безпеки або корекції поведінки роботи програми на той випадок, коли обчислення не завершились у відведений час.

Для визначення режиму роботи системи треба перелічити ознаки жорсткого і м'якого реального часу.

Ознаки жорсткого реального часу:

- затримки призводять до виходу системи з ладу;
- результати роботи не мають значення, якщо виникла помилка.

Ознаки м'якого реального часу:

- невиконання часових обмежень призводить до виникнення помилок, які негативно впливають на роботу системи;

- помилки призводять до зниження продуктивності системи.

Для того щоб система могла задовольнити вимогам, що пред'являються до СРЧ, апаратні, програмні засоби та алгоритми роботи системи повинні відповідати часовим параметрам реакції системи.

Вимоги до часу реакції системи і інших часових параметрів визначаються або технічним завданням, або логікою функціонування системи.

1.3 Операційні системи реального часу

Операційна система (ОС) – це комплекс програм, який забезпечує керування апаратними пристроями комп’ютера та виконанням прикладних програм.

Операційна система реального часу (ОС РЧ) – інструмент для побудови конкретної СРЧ.

Усі ОС РЧ можна розділити по різниці внутрішній архітектури:

- монолітні ОС;
- ОС на основі мікроядра;
- об’єктно-орієнтовані ОС.

Графічно архітектура різних ОС РЧ зображена на рис. 1.1, рис. 1.2, рис. 1.3.

Рисунок 1.1 – ОС РЧ з монолітною архітектурою

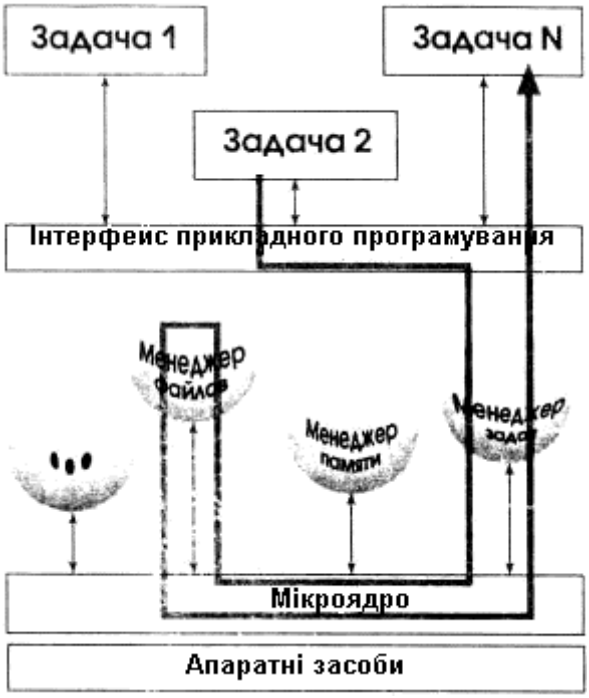

Рисунок 1.2 – ОС РЧ на основі мікроядра

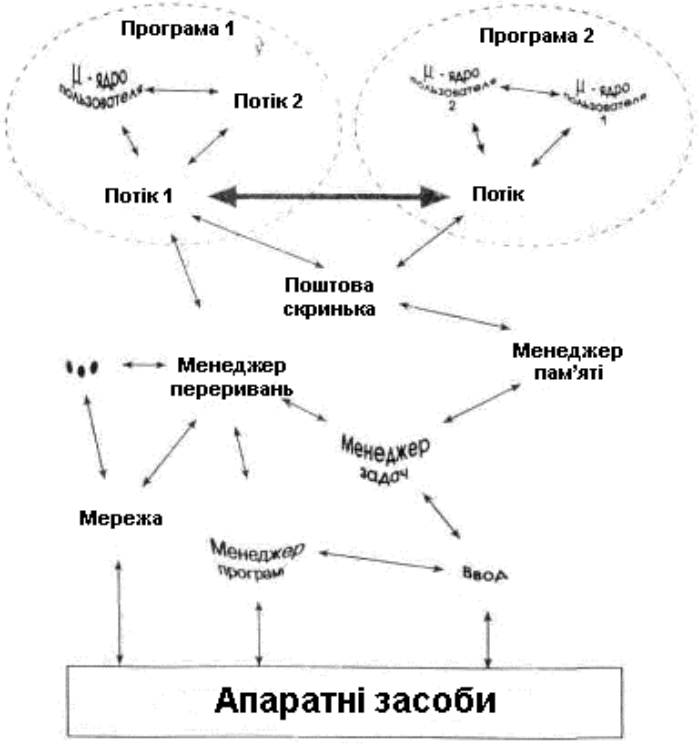

Рисунок 1.3 – Об'єктно-орієнтована ОС РЧ

Існує декілька вимог до операційних систем реального часу.

1. ОС РЧ має підтримувати диспетчеризацію з витісненням однієї задачі іншою. Планувальник повинен мати можливість призупиняти виконання (витісняти) будь-яку задачу і починати виконувати іншу задачу, яка є більш пріоритетною.

2. Кожна задача повинна мати пріоритет виконання для передачі управління тій задачі, термін реакції на подію є мінімальним. Для цього потрібно прогнозувати час завершення задач для того, щоб вони встигли до критичних термінів.

3. ОС повинна мати можливість синхронізувати задачі.

4. Повинен існувати механізм успадкування пріоритетів та захист від інверсії пріоритетів. Для створення інверсії пріоритетів повинно бути задіяно як мінімум три задачі. Якщо задача з найнижчим пріоритетом заблокувала ресурс, який вона ділить з самою пріоритетною задачею, в той час як працює задача з проміжним пріоритетом, виникає наступний ефект: задача з найвищим пріоритетом очікує звільнення ресурсу; нитка з проміжним пріоритетом витісняє фонову задачу і працює, поки не завершиться; управління отримує фонова задача, яка звільняє ресурс, і тільки після цього задача з високим пріоритетом може продовжити свою роботу. В цьому випадку час, необхідний для завершення задачи з найвищим пріоритетом, залежить від часу роботи задачі з більш низьким пріоритетом - це і є інверсія пріоритетів. Щоб уникнути таких ситуацій, ОС РЧ повинна бути забезпечена механізмом успадкування пріоритетів, тобто блокуюча задача повинна наслідувати пріоритет задачі, яку вона блокує, у випадку, якщо заблокована задача має більш високий пріоритет. Спадкування означає, що блокуючий ресурс успадковує пріоритет ресурсу, який він блокує.

1.4 Багатозадачність

Багатозадачність – це властивість ОС забезпечувати паралельне (або псевдопаралельне у випадку однопроцесорних систем) виконання декількох задач.

Так як тема роботи стосується однопроцесорних систем, то багатозадачність реалізується за допомогою перемикань між завданнями, які відбуваються настільки часто, що у користувача створюється враження, що задачі виконуються одночасно.

Багатозадачність ОС реалізує наступний механізм:

1. Задача може виконується, поки вона не буде витиснена більш пріоритетною задачею.
2. Зберігається контекст задачі для подальшого подовження виконання після повернення до перерваної задачі.
3. Інша задача починає виконуватись.
4. Цикл 1-3 повторюється, поки в системі присутні завдання, які чекають на виконання.

Багатозадачність зображена на рис. 1.4. Переключення задач збільшує загальний обсяг роботи, що виконується системою, однак у цілому ефективність роботи зростає.

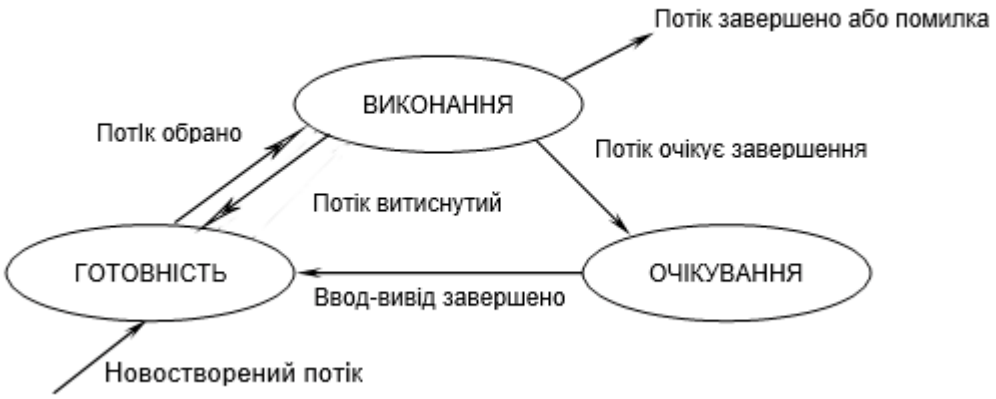

Рисунок 1.4 – Схематичне зображення багатозадачності

1.5 Основні поняття систем реального часу

Існують різні визначення терміна «задача» для ОС РЧ. Будемо вважати задачею набір операцій, призначений для виконання логічно закінченої функції. Отже фактично, завданням – це одиниця роботи, для виконання якої

використовуються ресурси процесору. Обчислювальний процес може включати в себе декілька завдань.

Прийнято розрізняти два різновиди завдань:

- процеси;
- потоки.

Процес – програмний модуль, який завантажується окремо.

Процес має:

- області пам'яті, в яких здійснюється зберігання коду та даних;
- стек;
- стан.

Процес може перебувати у таких станах:

- зупинений: у цьому стані процес не потребує ресурсів процесору, тому що фактично був тільки що створений;
- термінований: процес також не потребує ресурсів процесору, тому що він закінчився;
- чекає: процес чекає на появу деякої події (зазвичай переривання);
- готовий процес не зупинений, не термінований, не очікує, не видалений, але і не працює; процес може не отримувати доступу до процесора, якщо в даний момент виконується більш пріоритетний процес;
- виконується: стан, у якому процесор виконує процес.

Графічно так званий «цикл життя» процесу зображений на рис. 1.5.

Рисунок 1.5 – Схема «циклу життя процесу»

Завдання поділяють на три категорії:

- циклічні – використовуються декілька разів підряд з однаковими умовами, мають місто у інтерактивних процесах, де певні дії повторюються;
- періодичні – часто використовуються для проведення синхронізації за певним планом у відповідні моменти часу;
- імпульсні – можуть використовуватись для асинхронних технологічних процесів, які з’являються несподівано при певних обставинах.

Потоки можуть користуватися загальними ділянками коду і даних в рамках одного програмного модуля.

Переваги потоків:

1. Так як безліч потоків здатне розміщуватися всередині одного EXE-модуля, це дозволяє економити ресурси як зовнішньої, так і внутрішньої пам'яті.

2. Процеси не мають загальної області пам'яті, тому ОС повинна або цілком скопіювати повідомлення з області пам'яті одного завдання в область пам'яті інший (що для великих повідомлень досить складно), або передбачити спеціальні механізми, які дозволили б одній задачі отримати доступ до повідомлення з області пам'яті іншої задачі.

3. Як правило, контекст потоків менше, ніж контекст процесів, а значить, час перемикання між завданнями-потоками менше, ніж між завданнями-процесами.

4. Так як всі потоки розміщуються в одному EXE-модулі, значно спрощується використання дебагерів.

Недоліки потоків:

1. Як правило, потоки не можуть бути довантажені динамічно. Щоб додати новий потік, необхідно змінити додаток та перекомпілювати його знов. Процеси, на відміну від потоків, підгружаються динамічно, що дозволяє змінювати функції системи в процесі її роботи. Крім того, так як процесам відповідають окремі програмні модулі, вони можуть бути розроблені різними компаніями, чим досягається додаткова гнучкість і можливість використання раніше напрацьованого ПО.

2. Те, що потоки мають доступ до областям даних один одного, може привести до ситуації, коли некоректно працюючий потік здатний зіпсувати дані іншого потоку. На відміну від цього процеси захищені від взаємного впливу, а спроба запису у закриту область пам'яті призводить до виникнення переривання по обробці виняткових ситуацій.

Також необхідно ввести поняття ресурсу. Ресурс – це об'єкт, який використовується завданням для виконання. Ресурси поділяють на розподілені та неподільні.

Ресурс виділяється задачі, якщо вона звернулась до системи та:

- запитуваний ресурс вільний і в системі немає задач, які чекають на виконання та мають більший пріоритет;
- ресурс використовується задачею нижчого пріоритету, яка може бути призупинена для передачі ресурсу іншій задачі.

Якщо система отримує запит від задачі на надання ресурсу, який зайнятий у цей момент часу, ця задача додається у чергу на надання необхідного ресурсу.

Стек – це спеціальна область пам'яті для зберігання локальних змінних.

Віртуальна пам'ять – це пам'ять, яка використовується для створення адресного простору певного процесу. Така пам'ять збільшує обсяг оперативної пам'яті за рахунок використання постійної дискової пам'яті;

Подія – це оповіщення процесу операційною системою про взаємодію між процесами. Прикладом події є переривання.

1.6 Основні властивості задач

Як правило, вся важлива, з точки зору операційної системи, інформація про процеси зберігається в уніфікованої структурі даних – керуючому блоці (Task Control Block, TCB), який також називають дескриптором процесу.

Дескриптор процесу містить наступну інформацію:

- ідентифікатор процесу (ІД);
- тип процесу;
- пріоритет процесу, згідно якого планувальник визначає порядок виконання задач;
- стан процесу;
- приватну область пам'яті, в якій зберігаються поточні значення регістрів процесу, якщо процес переривається, не закінчивши роботи;
- інформацію про ресурси, якими процес володіє, наприклад, інформація про незавершені операції введення / виводу тощо;

- інформація для організації спілкування з іншими процесами;
- параметри часу запуску, які складаються з момент часу у який процес повинен запросити процесорний час і періодичність виконання, якщо процес є періодичним;

Пріоритет – це певне ціле число, що привласнюється задачі і характеризує її важливість в порівнянні з іншими задачами в системі. Пріоритет використовується планувальником завдань для визначення того, яка з готових до роботи задач повинна отримати управління. Розрізняють системи з динамічною і статичною пріоритетністю. У першому випадку пріоритет завдань може змінюватися в процесі виконання, в той час як у другому пріоритет завдань жорстко задається на етапі розробки або під час початкової конфігурації системи.

Контекст завдання – це набір даних, що містить всю необхідну інформацію для відновлення виконання завдання з того місця, де вона була раніше перервана. Часто контекст зберігається в TCB і включає в себе такі дані, як лічильник команд, покажчик стека, регістри CPU і FPU тощо. Планувальник завдань в разі необхідності зберігає контекст поточної активної задачі і повертає контекст задачі, призначеної до виконання . Таке переключення контекстів і є, по суті, основним механізмом ОС РЧ при переході від виконання однієї задачі до іншої.

З точки зору операційної системи, задача може перебувати в кількох станах. Назви та ідентифікатори цих станів відрізняються від однієї ОС до іншої. Проте, в будь-який ОС РЧ задача може перебувати, принаймні, в трьох станах.

- активна задача – це задача, яка виконується системою у поточний момент часу;
- готова задача – це задача, готова до виконання, яка очікує у планувальника своєї «черги»;
- блокована задача – задача, виконання якого призупинено.

Порожня задача (Idle Task) – це завдання, що запускається операційною системою в момент ініціалізації і виконується тільки тоді, коли в системі немає

інших готових для виконання завдань. Порожня задача запускається з найнижчим пріоритетом і, як правило, представляє собою нескінчений цикл.

Багатозадачні ОС дозволяють запускати кілька копій однієї і тієї ж задачі. При цьому для кожної такої копії створюється свій ТСВ і виділяється своя область пам'яті. З метою економії пам'яті може бути обране спільне використання одного і того ж коду, що виконується для всіх запущених копій. У цьому випадку програма повинна забезпечувати реентерабельність. Крім того, програма не повинна використовувати тимчасові файли з фіксованими іменами і повинна здійснювати доступ до спільних ресурсів.

Реентерабельність означає можливість без негативних наслідків тимчасово перервати виконання будь-якої функції або підпрограми, а потім викликати цю функцію або підпрограму знову. Прикладом реентерабельності є рекурсія, коли тіло підпрограми містить виклик самої себе. Прикладом нереентерабельності системи є DOS, а типовою причиною нереентерабельності служить не використання глобальних змінних.

Припустимо, що у нас є функція, що реалізує нізкоривневий запис на диск, і нехай вона використовує глобальну змінну write_sector, яка встановлюється відповідно до параметра, що передаються цій функції при виклику. Припустимо тепер, що Задача А викликає цю функцію з параметром 3, тобто хоче записати дані в сектор номер 3. Допустимо, що коли змінна write_sector вже дорівнює 3, але сам запис ще не проведений, виконання Завдання А переривається і починає виконання Задача В, яка викликає ту ж функцію, але з аргументом 10. Після того як запис в сектор номер 10 буде виконаний, управління повернеться до Задачі А, яка продовжить роботу з того ж місця. Однак, так як змінна write_sector має тепер значення 10, дані Завдання А, призначені для сектора номер 3, будуть замість цього записані в сектор номер 10. З наведеного прикладу видно, що помилки, пов'язані з нереентерабельністю, важко виявити, а наслідки вони можуть викликати катастрофічні.

1.7 Основні параметри задач

Кожна робота завдання характеризується наступними часовими параметрами:

- r (Release Time) – момент часу, коли задача запитує систему для отримання процесорного часу;
- d (Absolute Deadline) – абсолютний крайній термін часу, до якого задача повинна завершитись без виникнення помилок;
- s (Start Time) – момент часу, коли задача фактично починає виконання;
- c (Completion Time) – момент часу, коли задача фактично завершує виконання;
- D (Relative Deadline) – відносний крайній термін, який обчислюється як D = d – r, та може використовуватись для періодичних задач;
- e (Execution Time) – фактичний час виконання задачі, який обчислюється за наступною формулою: e = c -s;
- R (Response Time) – час відгуку, який дорівнює R = c – r та визначає кількість часу, затраченого на обробку задачі.

Приклад визначення часових характеристик наведений на рис. 1.6 та містить параметри: r = 2, d = 11, s = 5, c = 9, D = 11 − 2 = 9, e = 9 − 5 = 4, R = 9 − 2 = 7.

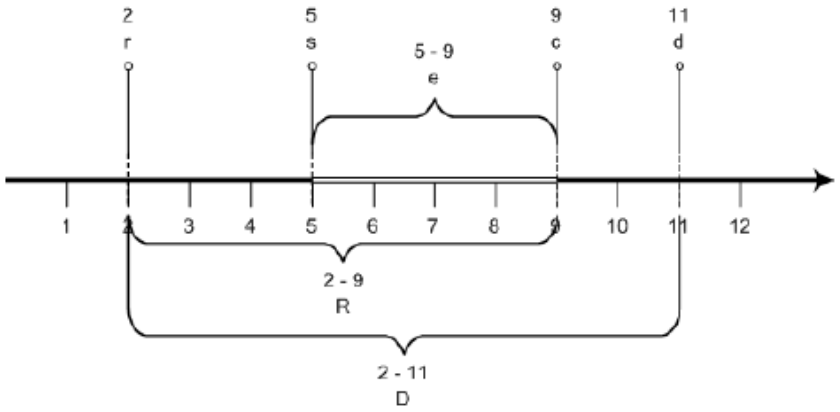

Рисунок 1.6 – Часові параметри задачі реального часу

Періодичні задачі мають деякі особливі параметри, де j – номер періоду:

- $\{T_j\}$ – моменти часу, коли j-та задача потребує виконання;
- φ_j – фаза періодичного j-го завдання;

Періодична задача може бути задана у наступній формі:

$$T_j\ [\varphi_j, p_j, e_j, D_j] \tag{1.1}$$

Коефіцієнт використання ресурсів процесора періодичною задачею визначається як:

$$\mu_j = e_j\ /\ p_j, \tag{1.2}$$

де e_j – час виконання задачі, а p_j – період виконання.

1.8 Класифікація часових обмежень

Правильність виконання задач у режимі реального часу залежить як від логічної правильності результату, так і від задоволення відповідних обмежень часу. Обмеження часу фактично стосуються певних подій у системі. Ці події можуть генеруватися самими завданнями або середовищем системи.

Класифікація часових обмежень крім кращого розуміння поведінки системи, також може дозволити точно опрацювати специфікацію системи в режимі реального часу.

Класифікація часових обмежень представлена на рис. 1.7

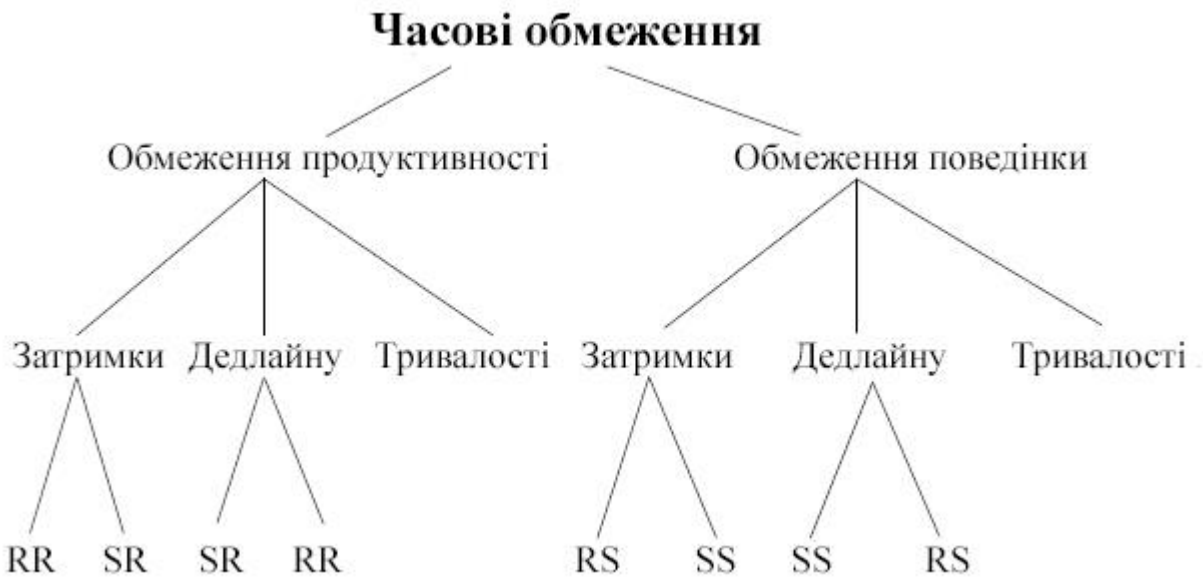

Рисунок 1.7 – Класифікація часових обмежень

Подія може генеруватися або системою, або середовищем системи. На основі цього можна розділити обмеження на два типи:

- стимульні події: генеруються середовищем і діють на систему. Ці події можуть виникати асинхронно (аперіодично). Наприклад, користувач, що натискає кнопку на телефоні, генерує подію стимулу для дії на телефонну систему. Події стимулу також можуть генеруватися періодично. Як приклад, можна розглянути періодичне вимірювання температури;
- події відповіді: зазвичай генеруються системою у відповідь на деякі події стимулу. Події відповіді діють на навколишнє середовище. Наприклад, розглянемо хімічний завод. Як тільки температура перевищує 100 °C, система реагує вимкненням нагрівача. Тут подія температури, що перевищує 100 °C, є стимулом, а вимкнення нагрівача – реакцією. Події відповіді можуть бути як періодичними, так і аперіодичними.

Подія може бути миттєвою або мати певну тривалість. Наприклад, подія натискання кнопки описується тривалістю, протягом якої кнопку утримували натиснутою.

Деякі автори стверджують, що тривалі події насправді не є базовим типом подій, але їх можна виразити за допомогою інших подій. Насправді можна розглядати подію тривалості як поєднання двох подій: початкової та кінцевої. Наприклад, подія натискання кнопки може бути описана комбінацією подій „натискання кнопки старту" та „натискання кнопки закінчення". Однак часто зручно зберігати поняття тривалої події. У цьому тексті ми розглядаємо тривалі події як особливий клас подій.

Загалом часові обмеження можна розділити на дві категорії:

- обмеження поведінки забезпечують правильну поведінку середовища системи;
- обмеження продуктивності забезпечують задовільну роботу самої СРЧ.

Кожне обмеження продуктивності та поведінки можна додатково розділити на три типи:

- обмеження затримки;
- обмеження крайнього терміну;
- обмеження тривалості.

Обмеження затримки фіксує мінімальний час, який повинен пройти між виникненням двох довільних подій e_1 та e_2. Після того, як перша подія настає, якщо друга подія настає раніше мінімальної затримки, тоді фіксується порушення затримки. Нехай, $t(e_1)$ та $t(e_2)$ є позначками часу настання подій e_1 та e_2 відповідно, а d – мінімальна допустима затримка, як показано на рис. 1.8. Отже, обмеження можна сформулювати як:

$$t(e_2) - t(e_1) \geq d \tag{1.3}$$

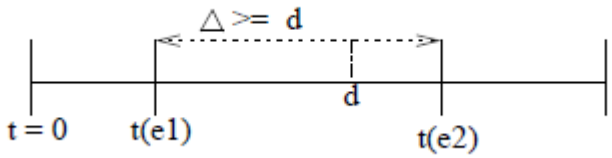

Рисунок 1.8 – Обмеження затримки між двома подіями

Обмеження крайнього терміну фіксує допустиму максимальну затримку між будь-якими двома довільними подіями e_1 та e_2. Іншими словами, друга подія повинна слідувати за першою у межах допустимого максимального часу очікування. Нехай, $t(e_1)$ та $t(e_2)$ є позначками часу настання подій e_1 та e_2 відповідно, а d - кінцевий термін, як показано на рис. 1.9. Отже, обмеження можна сформулювати як:

$$t(e_2) - t(e_1) \leq d \tag{1.4}$$

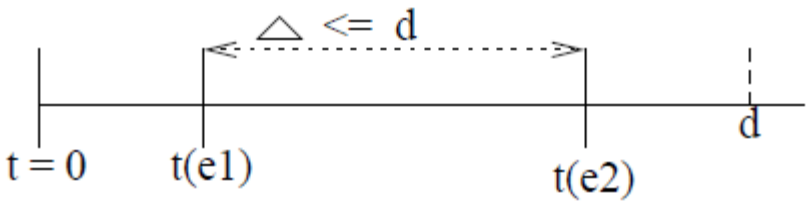

Рисунок 1.9 – Обмеження крайнього терміну між двома задачами

Обмеження тривалості події визначає період, протягом якого подія виконується. Це обмеження може бути як мінімальним, так і максимальним типом. Обмеження мінімальної тривалості вимагає, щоб після початку події, вона не закінчилась до певної мінімальної тривалості. Тоді як обмеження на максимальну тривалість вимагає, щоб подія закінчилась до того, як пройде певна максимальна тривалість.

Ми проілюструємо різні класи обмежень часу, використовуючи приклади телефонної системи. Принципова схема телефонної системи наведена на рис. 1.10.

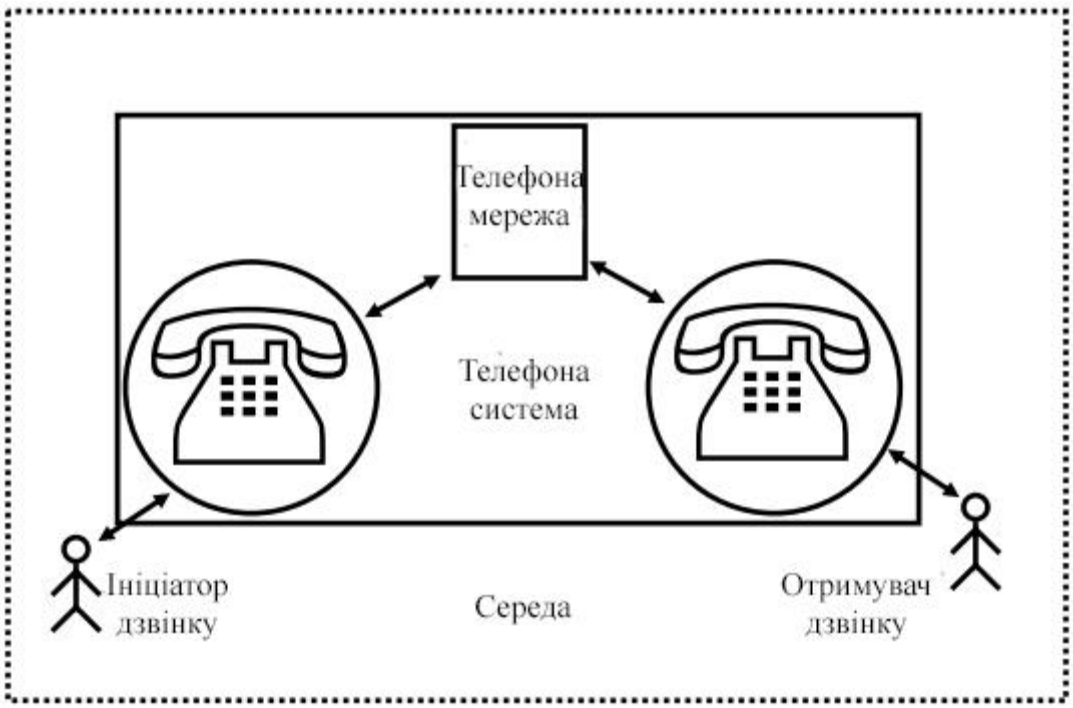

Рисунок 1.10 – Схема телефонної системи

Наведемо кілька простих прикладів операцій телефонної системи, щоб проілюструвати різні типи обмежень часу:

1.8.1 Обмеження стимул-стимул

Стимул-стимул (SS – Stimulus-Stimulus): У цьому випадку термін визначається між двома стимулами. Це поведінкове обмеження, оскільки обмеження накладається на другу подію, яка є стимулом.

Прикладом обмеження терміну дії SS є наступне:

Як тільки користувач завершить набір цифри, він повинен набрати наступну цифру протягом наступних 5 секунд; інакше видається сигнал простою. У цьому прикладі набір двох послідовних цифр представляє два стимули для телефонної системи.

1.8.2 Обмеження стимул-відповідь

Стимул-відповідь (SR – Stimulus-Response): У цьому випадку термін визначається для події відповіді, вимірюється від виникнення відповідної події стимулу. Це обмеження продуктивності, оскільки обмеження накладається на подію відповіді. Прикладом обмеження терміну дії SR є наступне:

Після того, як приймач ручного набору піднімає трубку, система повинна видавати сигнал набору протягом 2 секунд, інакше видається звуковий сигнал. У цьому прикладі підйом трубки приймача являє собою стимул для телефонної системи, а відповідь викликає сигнал дзвінка.

1.8.3 Обмеження відповідь-стимул

Відповідь-стимул (RS – Response-Stimulus): Тут кінцевий термін передбачає отримання відповіді, відрахованої від відповідного стимулу. Це обмеження поведінки, оскільки обмеження накладається на подію стимулу. Прикладом обмеження терміну дії типу RS є наступне.

Після того, як з’явиться сигнал набору, першу цифру потрібно набрати протягом 30 секунд, інакше система переходить у режим очікування і створюється сигнал простою.

1.8.4 Обмеження відповідь-відповідь

Відповідь-відповідь (RR – Response-Response): Тип обмеження терміну дії RR визначається для двох подій відповіді. У цьому випадку, як тільки відбувається перша подія відповіді, друга подія відповіді повинна відбутися до певного терміну. Це обмеження продуктивності, оскільки обмеження часу було визначено для події відповіді. Прикладом обмеження терміну дії RR є наступне:

Після того, як дзвінок надається абоненту, відповідний тон дзвінка повинен бути наданий абоненту протягом двох секунд, інакше дзвінок припиняється.

1.10 Планування задач

Важливою частиною будь-якої ОС РЧ є планувальник задач. Надивлячись на те, що в різних джерелах він може вказуватись по-різному (диспетчер задач, супервізор), його функції залишаються однаковими: визначити, яка з задач повинна виконуватися в системі в кожен конкретний момент часу.

1.10.1 Термінологія планування

Перш ніж зосередитись на різних класах планувальників, необхідно ввести кілька важливих понять, які будуть використані далі.

Дійсний розклад для набору завдань – це той, де одночасно призначається процесору не більше одного завдання, жодне завдання не планується до часу його прибуття і переваги та обмеження ресурсів усіх завдань задовольняються.

Дійсний розклад називається можливим розкладом, лише якщо всі завдання відповідають відповідним часовим обмеженням.

Кажуть, що планувальник завдань П1 є більш досвідченим, ніж інший планувальник П2, якщо П1 може запланувати всі набори завдань, які П2 може здійснити, але не навпаки. Тобто, П1 може доцільно запланувати всі набори завдань, які може П2, але існує принаймні один набір завдань, який П2 не може здійснити, тоді як П1 може. Якщо П1 може реально запланувати всі набори завдань, які П2 може реально запланувати, і навпаки, то П1 і П2 називаються однаково досвідченими планувальниками.

Планувальник завдань у режимі реального часу називається оптимальним, якщо неможливо знайти більш досвідчений алгоритм планування, ніж оптимальний планувальник. Якщо оптимальний планувальник не може запланувати якийсь набір завдань, тоді жоден інший планувальник не повинен мати можливості створити розклад для цього завдання.

Попереджувальний планувальник - це той, який при надходженні завдання з вищим пріоритетом призупиняє будь-яке завдання з нижчим пріоритетом, яке може

виконуватися, і бере на себе завдання з вищим пріоритетом. Таким чином, у превентивному планувальнику не може бути так, що завдання з вищим пріоритетом готове і чекає виконання, а завдання з нижчим пріоритетом виконується.

1.10.2 Класифікація алгоритмів планування

Існує багато схем класифікації алгоритмів планування. Популярна схема класифікує алгоритми планування завдань у реальному часі на основі того, як визначаються точки планування. Загалом планувальники поділяють на три категорії:

1. Керовані часом.
    - керовані таблицями;
    - циклічні планувальники.
2. Керовані подіями.
3. Гібридні.

Розглянемо окремо планувальники керовані часом. Для них точки планування визначаються таймерами переривань. Такі планувальники також називають offline планувальниками, тому що вони фіксують розклад виконання до запуску системи. Планувальник попередньо визначає, яке завдання коли буде виконуватись. Тому ці планувальники несуть дуже мало накладні витрати часу. Однак великим недоліком цього класу планувальників є те, що вони не можуть впоратися з періодичними та спорадичними завданнями, оскільки точний час виникнення цих завдань не можна передбачити. З цієї причини такий тип планувальників називають статичними планувальниками.

Планувальники, керовані таблицями, зазвичай заздалегідь обчислюють, які завдання треба виконувати на кожному такті, і зберігають цей графік у таблиці під час проектування або налаштування системи. Замість автоматичного обчислення розкладу, програміст може отримати свободу вибору власного розкладу для набору завдань.

Приклад табличного розкладу наведений у таблиці 1.1.

Задача	Час початку виконання
T_1	0
T_2	5
T_3	8
T_4	12
T_5	19

Таблиця 1.1 – Розклад планувальника, керованого таблицею

Циклічні планувальники дуже популярні і широко використовуються в СРЧ. Переважна більшість усіх невеликих вбудованих додатків базуються на циклічних планувальниках. Циклічні планувальники прості, ефективні та легко програмуються. Прикладом програми, де зазвичай використовується циклічний планувальник, є регулятор температури. Терморегулятор періодично перевіряє температуру в приміщенні і підтримує її на заданому рівні. Такі регулятори температури вбудовані в типові кондиціонери, керовані комп'ютером.

Циклічний планувальник повторює попередньо розрахований графік. Попередньо розрахований графік потрібно зберігати лише протягом одного основного циклу. Кожне завдання повторюється однаково у кожному великому циклі. Основний цикл поділяється на один або кілька другорядних циклів. Кожен другорядний цикл також іноді називають кадром.

Межі кадру визначаються через переривання, генеровані періодичним таймером. Кожне завдання призначене для запуску в одному або декількох кадрах.

Циклічні планувальники дуже ефективні. Однак яскравим недоліком циклічних планувальників є те, що дуже складно визначати відповідний розмір кадру, а також здійснити графік, коли кількість завдань збільшується. Далі, майже в кожному кадрі витрачається деякий час на обробку (оскільки розмір кадру більший

за всі часи виконання завдання), що призводить до отримання неоптимальних графіків.

Планувальники, керовані подіями, долають ці недоліки. Крім того, керовані подіями планувальники можуть більш досконало обробляти періодичні та епізодичні завдання. З іншого боку, керовані подіями планувальники менш ефективні, оскільки вони застосовують більш складні алгоритми планування. Тому планувальники, керовані подіями, менш придатні для вбудованих програм.

У керованому подіями плануванні точки планування визначаються подіями завершення завдання та прибуття завдання. Цей клас планувальників зазвичай є переважним, тобто, коли завдання з вищим пріоритетом стає готовим, воно випереджує будь-яке завдання з нижчим пріоритетом, яке може бути запущене.

Слід визначити три важливі типи планувальників, керованих подіями:

- Foreground-Background Scheduler;
- Rate Monotonic Analysis (RMA);
- Earliest Deadline First (EDF).

Планувальник Foreground-Background (FB) – це, можливо, найпростіший планувальник керований пріоритетами. При плануванні задачі реального часу запускаються як задачі переднього плану. Спорадичні та аперіодичні завдання виконуються як фонові завдання. Серед задач першого плану у кожній точці планування вибирається завдання найвищого пріоритету. Фонове завдання може запускатися, коли жодне із завдань переднього плану не виконується. Іншими словами, фонові завдання виконуються з найменшим пріоритетом.

У плануванні найменшого крайнього терміну (EDF) у кожній точки планування вибирається до виконання завдання, яке має найкоротший термін. Основні принципи цього алгоритму дуже інтуїтивні та прості для розуміння.

Тест на планування для EDF також простий. Набір завдань можна планувати за допомогою EDF, лише тоді, коли він задовольняє умові, що загальна завантаженість

процесора менше 1. Для набору періодичних задач {T1, T2,…, Tn}, Критерій здійсненності планування за допомогою EDF може бути виражений як:

$$\sum_{i=1}^{n} e_i/T_i, \tag{1.5}$$

де e_i – час виконання задачі, а T_i – період.

Доведено, що EDF є оптимальним алгоритмом планування для однопроцесорних систем. Це означає, що якщо набір завдань не можна спланувати за допомогою EDF, то жоден інший алгоритм планування не може здійснити планування цього набору завдань. У простому тесті планування для EDF (вираз 3.2) припустимо, що період виконання кожного завдання збігається із кінцевим терміном. Однак у практичних завданнях час виконання завдання може часом відрізнятися від терміну його виконання. У таких випадках тест на планування потрібно змінити. Якщо $p_i > d_i$, то для кожного завдання потрібно e_i кількість обчислювального часу кожну min(pi, di) тривалості часу.

Однак, якщо pi > di, можливо, що набір завдань можна планувати за допомогою EDF.

Якщо EDF слід розглядати як динамічний алгоритм планування, ми повинні мати можливість визначити точне значення пріоритету завдання в будь-який момент часу, а також мати можливість показати, як воно змінюється з часом. Планування EDF не вимагає обчислення будь-якого значення пріоритету для будь-якого завдання в будь-який час. Насправді EDF не має поняття пріоритетного значення для завдання. Завдання плануються виключно на основі крайнього терміну завершення. Однак, чим довше завдання чекає в готовій черзі, тим більша ймовірність бути прийнятим на планування. Отже, ми можемо уявити, що віртуальне значення пріоритету, пов’язане із завданням, збільшується з часом, поки завдання не буде прийнято для планування. Однак важливо розуміти, що в EDF завдання не мають жодного пріоритетного значення, пов'язаного з ними, планувальник не виконує жодних пріоритетних обчислень, щоб визначити пріоритет завдання під час виконання або компіляції.

Наївна реалізація EDF полягала б у підтримці всіх завдань, які готові до виконання, у черзі. Будь-яке щойно отримане завдання буде встановлено в кінець черги. Кожен вузол у черзі містив би абсолютний термін виконання завдання. У кожному пункті попередження вся черга буде відсканована з самого початку, щоб визначити завдання, що має найкоротший термін. Однак таке впровадження було б дуже неефективним. Проаналізуємо складність цієї схеми. Кожне додавання завдання буде здійснено за постійний час, але для вибору завдання знадобиться час O(n), де n – кількість завдань у черзі.

Більш ефективне впровадження EDF було б таким. EDF можна реалізувати, підтримуючи всі готові завдання в відсортованій черзі пріоритетів. Сортовану чергу пріоритетів можна ефективно реалізувати за допомогою структури даних купи. У черзі пріоритетів завдання завжди сортуються відповідно до близькості їх терміну. Коли завдання надходить, його можна вставити в купу за час O (log2 n), де n - загальна кількість завдань у черзі пріоритетів. У кожній точці планування наступне завдання, яке потрібно виконати, можна знайти у верхній частині купи. Коли завдання приймається для планування, його потрібно вилучити з черги пріоритетів. Цього можна досягти за час O (1).

Ще ефективнішого впровадження EDF можна досягти наступним чином, припускаючи, що кількість крайніх термінів завершення, які можуть мати завдання в заявці, обмежена. При такому підході, щоразу, коли надходить завдання, його абсолютний термін обчислюється з часу випуску та відносного терміну. Для кожного окремого відносного терміну, який може мати завдання, ведеться окрема черга FIFO. Планувальник вставляє нещодавно надійшло завдання в кінці відповідної відносної черги терміну. Зрозуміло, завдання в кожній черзі впорядковуються відповідно до їх абсолютних термінів.

Щоб знайти завдання з найменшим абсолютним терміном, планувальнику потрібно лише здійснити пошук серед усіх черг FIFO. Якщо кількість черг

пріоритетів, які підтримує планувальник, дорівнює Q, тоді порядок пошуку буде O(1). Час для вставки завдання також буде O(1).

Слід також зазначити декілька проблемних моментів в реалізації EDF.

Перехідне перевантаження означає перевантаження системи протягом дуже короткого часу. Воно виникає, коли виконання деякого завдання займає більше часу, ніж це було заплановано під час проектування. Виконання завдання може тривати довше з багатьох причин. Наприклад, завдання може увійти в нескінченний цикл або зіткнутися з незвичним станом. Коли EDF використовується для планування набору періодичних завдань у режимі реального часу, перевитрата часу завершення завдання може призвести до того, що деякі інші завдання втратять свої терміни. Зазвичай під час розробки програми дуже важко передбачити, яке завдання може пропустити свій кінцевий термін, коли в системі виникає перехідне перевантаження через невисокий пріоритет завдання, що перевищує його термін. Єдиний прогноз, який можна зробити, – це те, що завдання, яке запускатиметься відразу після завдання, що спричиняє перехідне перевантаження, затримується і може пропустити відповідні терміни. Навіть найважливіше завдання може пропустити свій термін через дуже низький пріоритет завдання, що перевищує запланований час завершення. Отже, повинно бути зрозуміло, що при використанні EDF будь-яка ретельна розробка не гарантує того, що найважливіше завдання не пропустить свій термін при перехідних перевантаженнях. Це серйозний недолік алгоритму планування EDF.

RMA є важливим алгоритмом планування, керованим подіями. Це алгоритм статичного пріоритету, який широко використовується в практичних програмах. RMA призначає пріоритети завданням, виходячи з рівня їх виконання. Чим менша частота виконання завдання, тим нижчим є призначений пріоритет. Завдання, що мають найвищу частоту виникнення (найнижчий період), отримує найвищий пріоритет. RMA було доведено оптимальним статичним алгоритмом планування завдань у режимі реального часу.

У RMA пріоритет завдання прямо пропорційний його швидкості (або, обернено пропорційний його періоду). Тобто пріоритет будь-якого завдання Ti обчислюється як: $priority = k \,/\, p_i$, де pi – період виконання завдання Ti, а k – константа. Використовуючи цей простий вираз, можна легко отримати графіки значень пріоритету завдань за RMA. Ці ділянки показані на рис. 1.11 (а) та рис. 1.11 (б). З цих цифр можна помітити, що пріоритет завдання лінійно зростає з приходом завдання та навпаки з періодом.

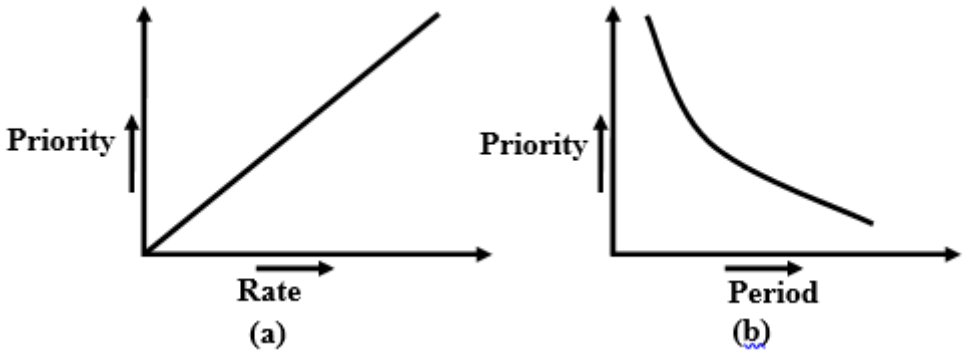

Рисунок 1.11 – Визначення пріоритету задачі

Важливою проблемою, яка вирішується під час проектування однопроцесорної системи реального часу, є перевірка того, чи можливо здійснити запланований набір періодичних завдань реального часу згідно RMA. Можливість планування завдання, встановленого згідно RMA, може бути визначена на основі знань про найгірший час виконання та періоди виконання завдань. На цьому етапі актуальним є питання про те, як розробник системи може визначити найгірший час виконання завдання ще до того, як система буде розроблена. Час виконання найгіршого випадку зазвичай визначається експериментально або шляхом імітаційних досліджень.

Нижче наведено деякі важливі критерії, за допомогою яких можна перевірити можливість планування набору завдань, встановлених згідно RMA.

Набір періодичних завдань у реальному часі не можна планувати RMA, якщо вони не задовольняють наступній необхідній умові:

$$\sum_{i=1}^{n} {}^{e_i}/_{T_i} \leq 1, \quad (1.6)$$

де e_i – найгірший час виконання, а T_i – період виконання завдання Ti, n - кількість запланованих завдань. Цей тест виражає той факт, що загальна завантаженість центрального процесора внаслідок усіх завдань у наборі завдань повинна бути менше 1.

RMA дуже часто використовується для планування завдань реального часу в практичних програмах. Базова підтримка доступна майже у всіх комерційних операційних системах реального часу для розробки програм, що використовують RMA, який також є оптимальним статичним алгоритмом планування пріоритетних завдань. На відміну від EDF, він вимагає дуже мало спеціальних структур даних. Більшість комерційних операційних систем реального часу підтримують (статичні) рівні пріоритету в задачах. Завдання, що мають рівні пріоритету в режимі реального часу, розташовані у багаторівневих чергах зворотного зв'язку. Серед завдань на одному рівні, ці комерційні операційні системи в режимі реального часу, як правило, надають можливість або зрізування часу, і кругового планування, або планування FIFO.

RMA має можливості обробки перехідних перевантажень. Хороша здатність до перехідних перевантажень означає, що коли завдання з нижчим пріоритетом не виконується протягом запланованого часу завершення, воно не може зробити жодне із завдань із вищим пріоритетом, щоб пропустити свій термін. Давайте тепер розберемося, як перехідне перевантаження вплине на набір завдань, запланованих згідно RMA. Чи вплине затримка з виконанням завдання з нижчим пріоритетом на завдання з вищим пріоритетом? Завдання з нижчим пріоритетом, навіть коли воно перевищує запланований час виконання, не може змусити завдання з вищим пріоритетом чекати відповідно до основних принципів RMA

Щоразу, коли завдання із вищим пріоритетом готове, воно попереджує будь-яке виконання завдання з нижчим пріоритетом. Таким чином, RMA стабільний при перехідних перевантаженнях, і завдання з нижчим пріоритетом, що перевищує час його завершення, не може зробити завдання з більш високим пріоритетом, щоб пропустити термін.

До недоліків RMA можна віднести наступне. В рамках RMA дуже важко підтримувати аперіодичні та спорадичні завдання. Крім того, RMA не є оптимальним, коли періоди та терміни виконання завдань різняться.

RMA більше не залишається оптимальним алгоритмом планування для періодичних завдань, коли терміни виконання завдань і періоди різняться для деяких завдань із набору запланованих завдань. Для таких наборів завдань монотонний алгоритм термінів (Deadline Monotonic Algorithm – DMA) виявляється більш досвідченим, ніж RMA. DMA по суті є різновидом RMA і призначає пріоритети завданням на основі їх термінів, а не присвоєння пріоритетів на основі періодів завдань, як це робиться в RMA. DMA призначає вищі пріоритети завданням із меншими термінами. Коли відносний термін виконання кожного завдання пропорційний його періоду, RMA та DMA виробляють однакові рішення. Коли відносні терміни є довільними, DMA є більш досвідченим, ніж RMA, в тому сенсі, що він іноді може створити реальний графік, коли RMA не вдається.

1.10 Самостійне призупинення задач

Задача може спричинити своє самостійне призупинення, коли вона виконує операції введення / виведення або чекає на якусь подію / умову. Коли завдання самостійно призупиняє себе, операційна система видаляє його з готової черги, розміщує в заблокованій черзі та бере наступне прийняте завдання для планування. Таким чином, самостійне призупинення вводить додаткову точку планування.

Визначимо вплив самопризупинення на планування набору завдань. Розглянемо набір періодичних завдань {T1, T2,…, Tn}, які розташовані у порядку збільшення їх пріоритетів (або порядку зменшення періодів). Нехай у найгіршому

випадку час самопризупинення завдання Ti є bi. Тоді bti можна виразити як:

$$bt_i = b_i + \sum_{k=1}^{i-1} \min(e_k, b_k) \quad (1.7)$$

Самопризупинення завдання Tk з вищим пріоритетом може вплинути на час відгуку завдання Ti з нижчим пріоритетом на стільки, скільки час його виконання ek, якщо ek < bk. Ця найгірша затримка може виникнути, коли завдання з вищим пріоритетом після самостійного призупинення починає своє виконання саме в той момент, коли завдання з нижчим пріоритетом виконувалося б інакше. Тобто після самостійного призупинення виконання завдання з вищим пріоритетом перекривається із завданням із нижчим пріоритетом, з яким воно інакше не перетиналося б. Однак, якщо ek> bk, то самостійне призупинення завдання з вищим пріоритетом може затримати завдання з нижчим пріоритетом щонайбільше до bk, оскільки максимальний період перекриття виконання завдання з вищим пріоритетом через самопризупинення обмежено bk.

ВИСНОВКИ ДО РОЗДІЛУ 1

Даний розділ наводить визначення, які необхідні для розуміння подальшого контексту та детально розкриває особливості планувальників, які використовуються в порівнянні результатів.

2. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ПРОБЛЕМИ ВИЗНАЧЕННЯ ЧАСУ ВИКОНАННЯ ЗАДАЧ

2.1 Методи визначення часу виконання програм

Визначення часу виконання програми можна розбити на декілька етапів:

- дослідження методів визначення часу виконання програми у монопольному режимі;
- дослідження методів визначення часу виконання програм у СРЧ, що використовує алгоритми планування, засновані на пріоритетах задач.

2.2 Прогнозування часу виконання програми

Вважається, що час виконання певної дії є деякою константою. Наприклад, оператори мови програмування, такі як: додавання, ділення, присвоєння.

Відповідно, знаючи час виконання однієї дії, можна обчислити час виконання послідовності цих дій.

Однак, гіпотеза про постійний час виконання не витримує критики. Час може змінюватись у залежності від типу адресації, поточного стану процесора, типу організації пам’яті процесора тощо.

Іноді підраховують мінімальний та максимальний час виконання, що дещо покращує ймовірність отримати вірний прогноз.

Такі обчислення називають аналітичними. Подальше покращення аналітичних обчислень можливе, якщо додатково промоделювати роботу різних пристроїв процесора або навіть побудувати емулятор обчислювача. Частіше використовують лише статичну імітацію, яка зводиться до побудови таблиць зайнятості пристроїв процесора при виконанні програми.

У подальшому будуть розглядатися тільки спеціалізовані системи, у яких програми представлені мовою високого рівня, а тип процесора визначений наперед. Це дозволяє абстрагуватись від особливостей архітектури процесорів.

Часто використовують термін «лінійний блок програми». Лінійним блоком

(ЛБ) програми будемо вважати фрагмент, що створений послідовними діями програми, якому притаманні наступні властивості:

- виконання фрагменту починається з його першої дії;
- виконуються послідовно усі зазначені дії до останньої.

Загальна лінійна структура програми - досить рідкісний випадок. Як правило, вона складається, як з лінійних блоків, так і з циклів, умовних, безумовних переходів та викликів процедур і функцій. Така структура програми визначає досить складну структуру графа передачі керування між базовими блоками. Цей граф фактично визначає множину всіх шляхів виконання від початку до завершення програм.[2]

Задача прогнозування часу виконання окремої програми пов’язана з вирішенням таких задач:

- вимір часу виконання прикладної програми;
- вимір часу виконання обраного фрагменту програми;
- прогнозування часу виконання програми на моделі випадкового марківського процесу з дискретними станами та дискретним часом.

2.3 Методика виміру часу виконання програми

Впливання на час виконання мають такі фактори:

- особливості реалізації програми;
- процесорна архітектура;
- операційна система;
- наявність багатопроцесорної системи;
- стан системи на момент початку виконання програми;
- вплив вимірювача часу виконання.

Зазначимо методи усунення стороннього впливу:

- програму слід запускати циклічно та поділити загально отриманий час на число ітерацій циклу;
- виключити з виміру часу виконання процедури ініціалізації та

деініціалізації фрагменту коду;

- зменшити вплив частини коду, яка реалізує вимір часу, цей код має викликатись виключно на початку та при завершенні фрагменту коду, час виконання якого треба визначити;
- виконати прив’язку поточного потоку до одного з ядер процесора (якщо він багатоядерний), для цього встановити для поточного процесу найвищий пріоритет;
- для виміру часу доцільно використовувати функції, що отримують поточне значення високочастотного лічильника та перетворюють його в одиниці часу (кількість тиків у секунду).

Результат підрахунку є прогнозованим, тому він не може гарантувати завершення програми за отриманий час.

2.4 Методика виміру часу лінійних блоків прикладної програми

З метою аналізу часу виконання ЛБ прикладної програми розглянемо схему програми, що відповідає алгоритму сортування Шелла, яка зображена на рис. 2.1.

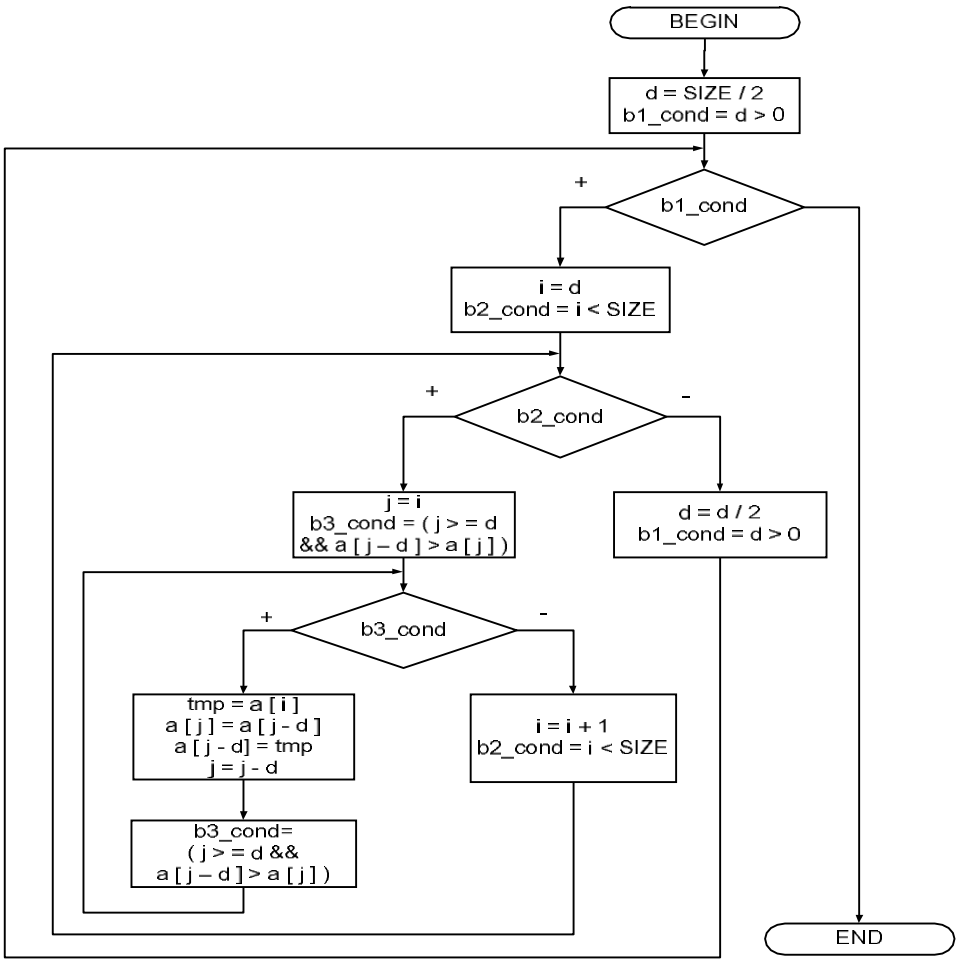

Рисунок 2.1. – Схема алгоритму сортування

За отриманою схемою можна побудувати граф переходів програми, де окремі блоки програми є вершинами графу. Лінійні блоки, час виконання яких буде у подальшому виміряно, позначаються цифрами, починаючи з нуля. Вершини графу, що відповідають переходам, позначено символом «с» та номером відповідного переходу. Всього на графі зображено 7 лінійних блоків з номерами від 0 до 6 та три умовні переходи C_0, C_1, C_2. Граф переходів представлений на рис. 2.2.

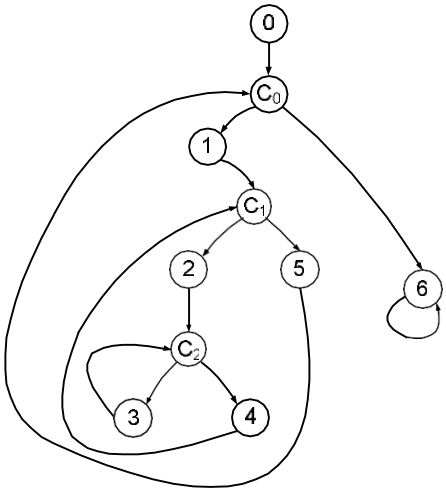

Рисунок 2.2 – Граф переходів програми

Результати роботи програми наведено у таблиці. 2.1. Часи виконання блоків розраховано в мілісекундах.

Таблиця 2.1 – Результат виміру часу

Block #0
Average time: 0.000514992
Block #1
Average time: 0.000303495
Block #2
Average time: 0.000403693
Block #3
Average time: 0.000306495
Block #4
Average time: 0.000307495
Block #5
Average time: 0.000303995

2.5 Метод прогнозування часу виконання програм за допомогою марківського ланцюга

Марківський ланцюг – це граф, у якому вершини відповідають дискретним станам програми, а ребра графа відображають перехід з одного дискретного стану у інший. Ребра мають вагу, яка дорівнює ймовірності цього переходу.

Марківський граф містить службові вершини:

- вершина початковому стану програми;
- вершина поглинаючого стану – стан завершення випадкового.

Стан випадкового процесу – це один з лінійних блоків програми. Отже, кількість станів дорівнює кількості ЛБ. При цьому перехід на графі фактично є переходом програми, який може бути як умовним, так і безумовним.

Такий граф дає змогу отримати кількість запусків різних лінійних блоків. Обчислення завершується при переході у поглинаючий стан, а отже, при завершені виконання програми. Такий метод надає результат у вигляді середнього часу виконання, оскільки спрацювання лінійних блоків можуть відрізнятись.

Для обчислення часу виконання програми методом марківського ланцюга використовують методи побудови фундаментальної матриці, яка об’єднує вершини та переходи та моделюють роботу переходів при виконанні ланцюга. Слід зазначити, що при кількості станів більше тридцяти, обчислення значно уклинюються, а отже, збільшується вплив на ресурси процесору. Для сучасних програм, де кількість станів дуже велика, такий підхід не є доцільним.

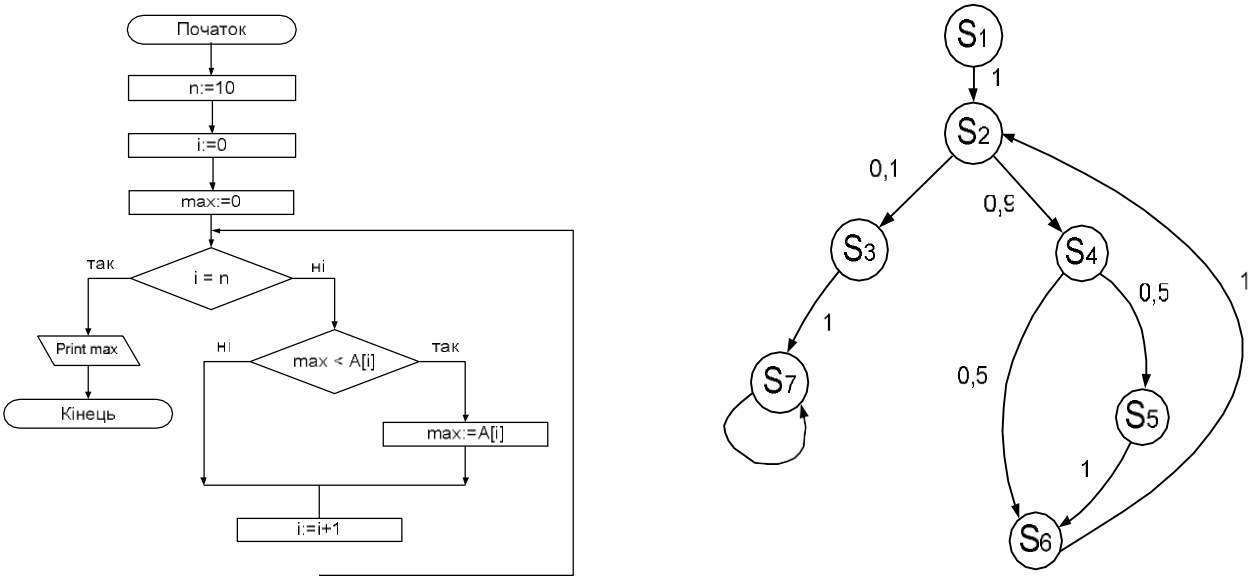

Рисунок 2.3 – Схема алгоритму (а) та представлення у вигляді графу (б)

2.6 Методи дослідження часових характеристик виконання комплексу програм

Система масового обслуговування (СМО) відповідає програмній системі, що не використовує пріоритети задач. Така СМО є скінченою.

Граф випадкового процесу в СМО представлений на рис. 2.4.

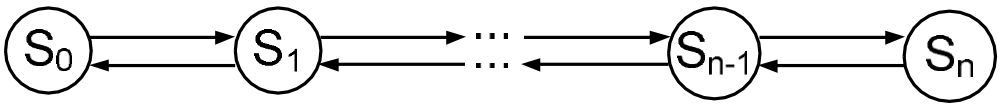

Рисунок 2.4 – Граф випадкового процесу *S*

Стан випадкового процесу відображає кількість задач, які доступні для виконання. Задачі виконуються у порядку надходження для виконання у системі.

Із присутніх у системі задач, будь-яка може запросити процесорний час. Якщо процесор має вільні ресурси, вони надаються цій задачі. У іншому випадку, коли процесор вже виконує певну роботу, задача записується в чергу виконання.

Формування черги потребує додаткового часу виконання та ресурсів процесору. Задача не може запросити ресурси процесора, якщо передній запит ще не був оброблений.

Абсолютні пріоритети виконання задач мають певні закономірності.

1. Якщо маємо тільки два рівні пріоритетів, формується ізольована пріоритетна черга. Це дозволяє обробляти лише запити з більш високим пріоритетом та економити ресурси процесору.

2. Якщо маємо k різних пріоритетів, розмір черги виконання не залежить від порядку надходження задач.

Якщо система є багатозадачною, слід використовувати марківську модель СМО.

Розглянемо граф $G\{x(t)\}$ на рис. 2.5. Інтенсивність виконання задач напряму залежить від наповненості черги задач. Найбільш пріоритетні задачі не залежать від задач, які мають менший пріоритет.

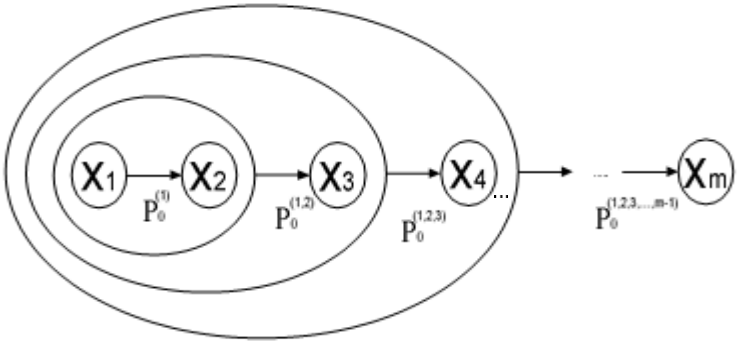

Рисунок 2.5 – Графу $G\{x(t)\}$

2.7 Розрахунок критерію здійсненності для циклічних задач СРЧ

Розрахунок часових характеристик відбувається за допомогою двох моделей СМО:

- базової моделі СМО;
- об’єднаної базової моделі СМО.

Основною характеристику задачі є її пріоритет.

Обчислення часових характеристик залежить від обраного алгоритму планування.

Отже можна зробити висновок, що даний інструмент дозволяє моделювати планувальники задач на алгоритмах MQS та FIFO. Але необхідно адаптувати алгоритм для кожного типу планувальника. Зі збільшенням кількості пріоритетів, складність розрахунків значно збільшується. Тому даний метод не можна назвати універсальним.

2.8 Моделювання часових обмежень

Моделювання часових обмежень дуже важливо, оскільки вона може служити формальною специфікацією системи. Крім того, якщо всі часові обмеження в системі змодельовані точно, то вона може навіть використовуватися для автоматичного генерування коду. Окрім того, що вони слугують специфікацією, моделювання часових обмежень може допомогти перевірити та зрозуміти систему реального часу.

2.8.1 Автомат скінченного стану

The Finite State Machine (FSM) – це потужний інструмент, який давно використовується для моделювання традиційних систем. У FSM стан визначається з точки зору значень, прийнятих деякими атрибутами. Наприклад, стани ліфта можна позначити через його напрямки руху. Тут напрямок – це атрибут, на основі якого визначаються стани вгору, вниз і канцелярські товари.

У моделі FSM у будь-який момент часу система може знаходитися в будь-якому стані. Стан представляється колом. Система змінює стан через події, що змінюють значення або відношення між змінними стану. Зміна стану також називається переходом держави. Подія, що спричиняє перехід, може бути або подією інтерфейсу, яка передається між середовищем та комп'ютерною системою, або може

бути внутрішньою подією, яка генерується та споживається виключно в системі. Перехід з одного стану в інший представляється малюванням спрямованої дуги від джерела до пункту призначення. Подія, що викликає перехід, коментується на дузі. Ми зводимо обговорення питань FSM до мінімуму, оскільки припускаємо, що читач знайомий з базовим моделюванням FSM традиційних систем.

2.8.2 Розширений кінцевий автомат

Ми використовуємо розширений кінцевий автомат (Extended Finite State Machine – EFSM) для моделювання часових обмежень. EFSM розширює традиційний FSM, включаючи дію встановлення таймера та подію закінчення таймера.

Приклад, показаний на рис. 2.6, описує, що якщо подія e1 настає, коли поточний стан системи дорівнює s1, тоді буде виконана дія, встановивши таймер, який закінчується протягом наступних 20 мілісекунд, і система переходить у стан s2.

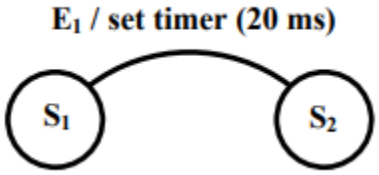

Рисунок 2.6 – Конвенції, що використовуються при складанні EFSM

2.8.3 Моделювання обмеження стимул-стимул

Розглянемо приклад обмеження терміну дії SS. Після того, як перша цифра буде набрана на телефонній трубці, наступна цифра повинна бути набрана протягом наступних 5 мілісекунд. Модель зображена на рис. 2.7. Можна спостерігати, що як тільки набирається перша цифра, система переходить у стан "Очікуйте другу цифру", а таймер встановлюється на 20 мілісекунд. Якщо наступна цифра не з'являється протягом 20 мілісекунд, тоді сигнал таймера закінчується, і система переходить у стан "Очікуйте абонента підключення" і видається звуковий сигнал.

Якщо друга цифра трапляється до 20 мілісекунд, система переходить до стану "Чекаю наступної цифри".

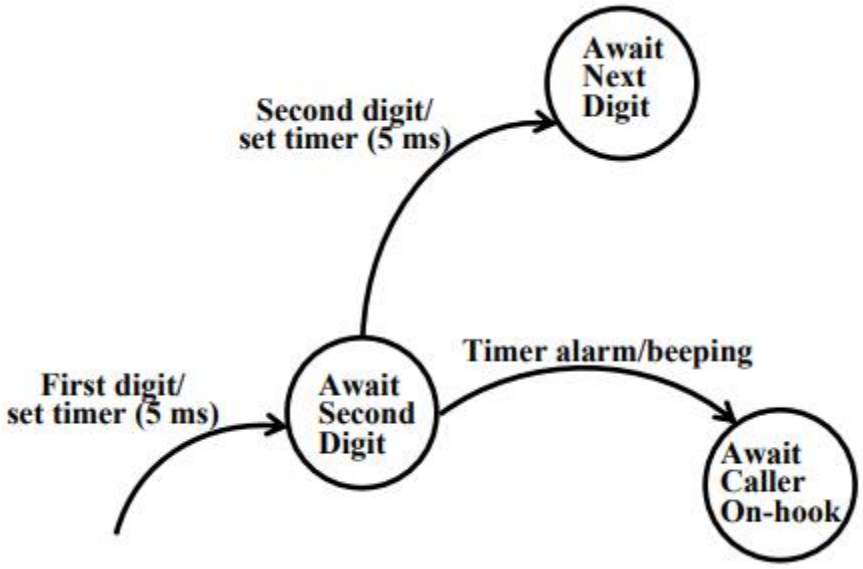

Рисунок 2.7 – Модель обмеження SS

2.8.4 Моделювання обмеження відповідь-відповідь

Модель EFSM для цього обмеження показана на рис. 2.8. Як тільки з’являється сигнал набору, таймер встановлюється так, щоб закінчитись через 30 секунд, і перевести систему у стан “Очікуйте першу цифру”. Якщо таймер закінчується до прибуття першої цифри, система переходить у режим очікування, де виробляється сигнал простою. В іншому випадку, якщо цифра з’являється першою, система переходить у стан “Очікуйте другу цифру”.

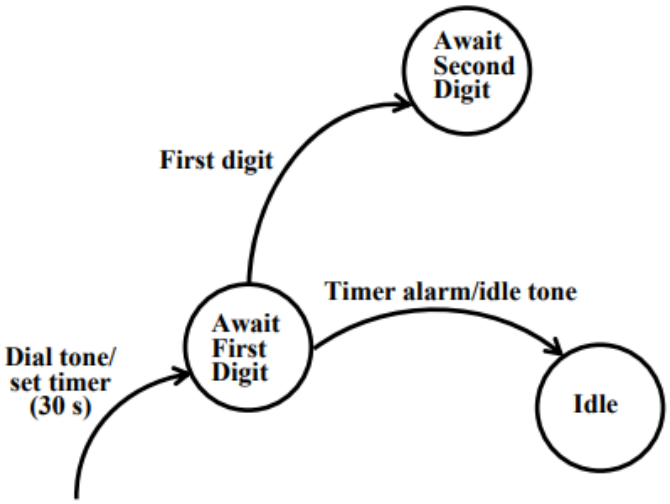

Рисунок 2.8 – Модель обмеження RR

2.8.5 Модель обмеження стимул-відповідь

Модель EFSM для цього обмеження показана на рис. 2.9. Як тільки слухавку піднімають, таймер встановлюється через 2 секунди, і система переходить у стан “Очікуйте сигнал набору”. Якщо спочатку з’являється сигнал набору, система переходить у стан “Очікуйте першу цифру”. В іншому випадку він переходить у стан “Очікуйте приймача підключення”.

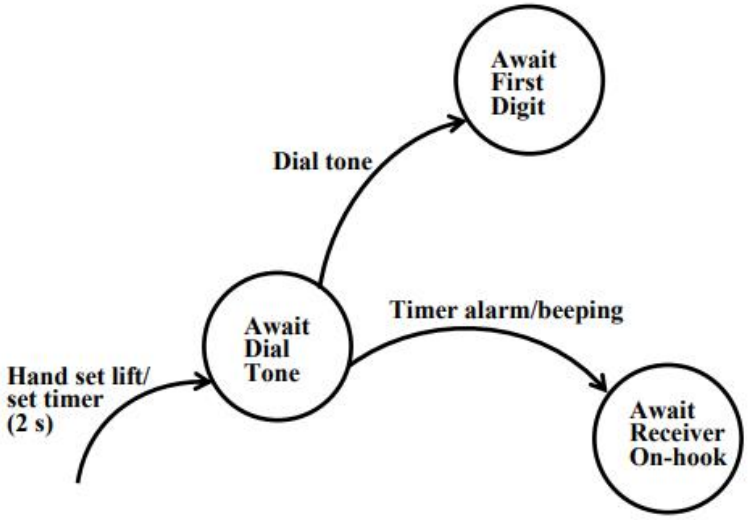

Рисунок 2.9 – Модель обмеження SR

2.8.6 Модель обмеження відповідь-відповідь

Модель EFSM для цього обмеження показана на рис. 2.10. Як тільки виробляється сигнал дзвінка, система переходить у стан “Очікуйте сигнал зворотного дзвінка”, і таймер встановлюється для закінчення за 2 секунди.

Якщо сигнал дзвінка з'являється першим, система переходить у стан "Очікуйте першу цифру", в іншому випадку вона переходить у стан "Очікуйте приймача підключено", і дзвінок припиняється.

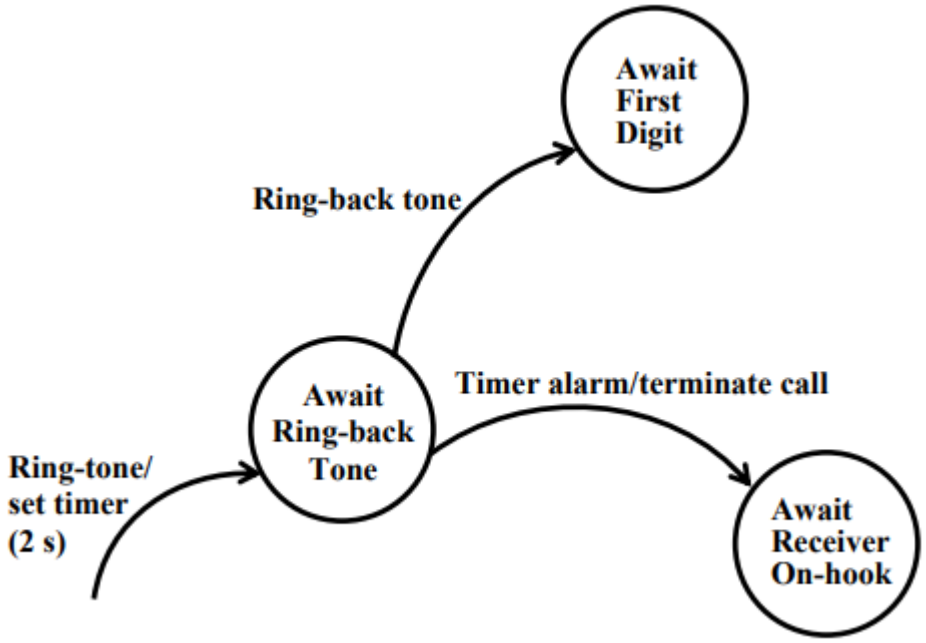

Рисунок 2.10 – Модель обмеження RR

ВИСНОВКИ ДО РОЗДІЛУ 2

Даний розділ наводить критерії оцінки існуючих методів визначення часових характеристик. Перелічуються та аналізуються способи прогнозування та визначення здатності систем реального часу виконувати необхідні задачі без помилок у роботі системи.

3. ВИЗНАЧЕННЯ ЧАСОВИХ ХАРАКТЕРИСТИК ЗА ДОПОМОГОЮ МОДЕЛЮВАННЯ РОЗПОДІЛУ ПРОЦЕСОРНОГО ЧАСУ МІЖ ЗАДАЧАМИ ЗГІДНО ОБРАНИХ АЛГОРИТМІВ ПЛАНУВАЛЬНИКІВ З ВИКОРИСТАННЯМ МОДЕЛІ СІТОК ПЕТРІ

3.1 Характеристика задачі реального часу

Кожна задача Ai характеризується трьома параметрами: часом обчислення, періодом та часом завершення обчислення.

Кожне виконання задачі потребує отримання процесорного часу. Програма складається з незалежних задач, які не мають взаємозалежності щодо доступу до спільних ресурсів.

Різні завдання, що породжуються за рахунок однієї тієї ж самої задачі, вважаються однотипними завданнями. Кожне створення такого завдання збільшує кількість задач, що претендують на отримання процесорного часу. Завершення кожного завдання зменшує кількість задач, що претендують на процесорний час.

3.2 Витискання задач

Витискання – це можливість переривання виконуваної задачі для виконання іншої задачі. Через деякий час процесор може знов взяти перервану задачу для виконання. Це призводить до виникнення інтерференції – взаємного впливу активних задач на час їх виконання.

Критичний сценарій для задачі A_i – це сукупність подій, при яких існування чергової задачі A_i збільшується до максимального значення R_i.

Часові обмеження на виконання задач типу A_i задаються відносним крайнім терміном виконання задачі $A_i – D_i$.

Отже, необхідно забезпечити виконання умови:

$$R_i <= D_i \quad (3.1)$$

Визначимо характеристику u_i – навантаження на процесор від задачі A_i. Значення навантаження визначається як відношення потрібного процесорного часу до періоду активізації задачі:

$$u_i = e_i / T_i \tag{3.2}$$

Сумарне навантаження задач:

$$U = \sum u_j = \sum e_j / T_j \tag{3.3}$$

3.3 Модель виконання однієї задачі

Схема дослідження представлена на рис. 1. Вона складається з генератора потоку задач, алгоритмів планування та моделі виконання задач.

Генератор потоку задач видає послідовності задач в межах періоду. Планувальник відповідно до обраного алгоритму планування визначає задачі, яким слід надати процесорний час і отримує від моделі виконання задач інформацію про моменти надання процесорного часу кожній задачі, проценти та моменти завершення виконання, а також про ступінь завантаженості процесора за період виконання T_i.

У моделі виконання задач попередньо має бути задана інформація про час виконання кожної із задач.

Схема моделювання зображена на рис. 3.1.

Рисунок 3.1 – Схема моделювання

Структура сіток Петрі складається з двох типів вузлів: позицій та переходів. Орієнтовані дуги поєднують позиції і переходи. Дуга, що направлена від P_i позиції до переходу τ_j визначає позицію, що являє собою вхід у перехід. Вихідна позиція

вказується дугою від переходу до позиції. Входи та виходи можуть бути кратними, що позначається на схемі або кількістю дуг, або цифрою k на дузі. Кожна позиція може мати певну кількість маркерів, які можуть рухатись з позиції на позицію при запуску переходів.

Сітка Петрі – це орієнтований дводольний мультиграф. Аналітичним способом задається як S = (P, Θ, F, H. μ_0), де додатково визначені вхідна функція F та вихідна H. Через F(τ_j) позначається множина вхідних позицій, а через H(τ_j) – множина вихідних позицій переходу, μ_0 – початкове маркування.

Перехід називають дозволеним, якщо кожна з його вхідних позицій має кількість маркерів не менше ніж кількість дуг із позицій в перехід. Кратні маркери необхідні для кратних вхідних дуг.

На рис. 3.2 наведена сітка Петрі, задану графічно.

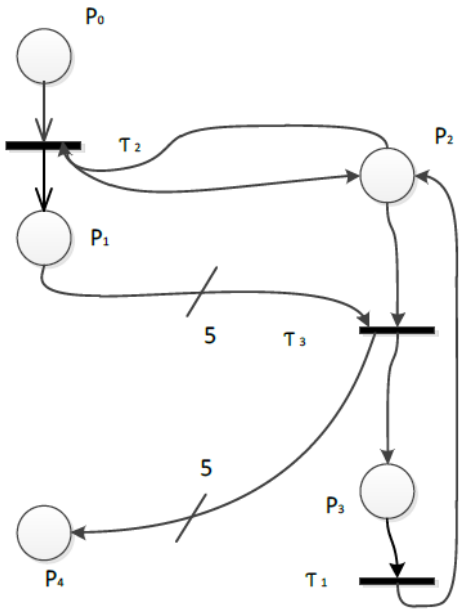

Рисунок 3.2 – Сітка Петрі

Представлення сітки у матричній формі:

де M^- та M^+ – матриці розміром (таб. 3.1, таб. 3.2) m x n, де m – кількість переходів, а n – кількість позицій.

Матриця інцедентності, зображена у таб. 3.3, сітки Петрі визначається як

Для наведеної на рис. 2 сітки початкове маркування M = [5 0 0 1 0], оскільки] = 0 дорівнюють кількісті маркерів в позиції P_i.

Таблиця 3.1 – Матриця M^-

τ_1					
τ_2					
τ_3					

Таблиця 3.2 – Матриця M^+

τ_1					
τ_2					
τ_3					

Таблиця 3.3 – Матриця M

τ_1					-
τ_2					
τ_3					

Перехід виконується тільки якщо він дозволений. Обчислення нового маркування здійснюється за формулою:

$$\mu_{i+1} = \mu_i + B_j \text{ x } M, \qquad (3.6)$$

де B_j – одиничний вектор переходу j, усі комірки якого нульові, крім тієї, що відповідає номеру j та дорівнює 1. Якщо перехід спрацьовує, нове маркування μ_{i+1} не буде мати від'ємних маркувань.

Наявність від'ємних чисел у позиціях при новому обчисленому маркуванні свідчить про те, що перехід не спрацьовує і маркування залишається незмінним.

Величина B_j x M є константою:

$$\tau_1 : B_1 \text{ x } M = [1\ 0\ 0] \text{x} M = [0\ 0\ 1\ -1\ 0];$$
$$\tau_2 : B_2 \text{ x } M = [0\ 1\ 0] \text{x} M = [-1\ 1\ 0\ 0\ 0];$$
$$\tau_3 : B_3 \text{ x } M = [0\ 0\ 1] \text{ x } M = [0\ -5\ -1\ 1\ 5].$$

Такт запуску сітки – це спроба запустити послідовність переходів τ_1, τ_2, τ_3. На кожному такті частина з кожних трьох переходів не спрацьовує. Якщо перехід не спрацьовує, він не змінює маркування сітки, і маркування залишається попереднім.

З аналізу сітки Петрі видно, що переходи маркерів з P_0 позиції у P_1 позицію і з P_1 позиції у P_4 виконуються за тактами, що складаються з послідовного збудження сукупності переходів.

Таблиця 3.4 – Виконання сітки Петрі

Номер такту	Перехід	Початкове маркування	Результат запуску	Спрацювання переходу	Нове маркування
1	τ_1	5 0 0 1 0	5 0 1 0 0	+	5 0 1 0 0
	τ_2	5 0 1 0 0	4 1 1 0 0	+	4 1 1 0 0
	τ_3	4 1 1 0 0	4 -4 0 1 5	-	4 1 1 0 0
2	τ_1	4 1 1 0 0	4 1 2 -1 0	-	4 1 1 0 0
	τ_2	4 1 1 0 0	3 2 1 0 0	+	3 2 1 0 0
	τ_3	3 2 1 0 0	3 -3 0 1 5	-	3 2 1 0 0
3	τ_1	3 2 1 0 0	3 2 2 -1 0	-	3 2 1 0 0
	τ_2	3 2 1 0 0	2 3 1 0 0	+	2 3 1 0 0
	τ_3	2 3 1 0 0	2 -2 0 1 5	-	2 3 1 0 0
4	τ_1	2 3 1 0 0	2 3 2 -1 0	-	2 3 1 0 0
	τ_2	2 3 1 0 0	1 4 1 0 0	+	1 4 1 0 0
	τ_3	1 4 1 0 0	1 -1 0 1 5	-	1 4 1 0 0
5	τ_1	1 4 1 0 0	1 4 2 -1 0	-	1 4 1 0 0
	τ_2	1 4 1 0 0	0 5 1 0 0	+	0 5 1 0 0
	τ_3	0 5 1 0 0	0 0 0 1 5	+	0 0 0 1 5

Створена модель виконання однієї задачі є початковою точкою для проектування моделі виконання декількох задач.

3.4 Модель виконання декількох задач

Головною відмінністю між розробкою моделі задачі та моделі декількох задач є необхідність реалізації переривань під час передачі управління від однієї задачі до

іншої та поновлення перерваної задачі. Змоделювати переривання можна за допомогою припинення видачі процесорного часу на відповідному такті виконання. Поновлення, навпаки, досягається за рахунок повернення процесорного часу для задачі на іншому такті.

Нехай маємо сукупність задач n, яка повинна бути представлена графічно за допомого сіток Петрі. Для цього необхідно виділити спільну позицію P_0 на графі для всіх задач. Коли задача отримує процесорний час для виконання, призводить до збудження переходів цієї задачі. І, навпаки, при припиненні виконання задачі, припиняються збудження переходів.

У запропонованій на рис. 2.3 моделі ведеться подвійна індексація, яка дозволяє розрізняти позиції та переходи однієї задачі від відповідних елементів іншої. Перший індекс подає номер задачі, а другий – нумерований елемент частини графу, що відповідає цій задачі.

Окремо постає питання виміру невикористаного процесорного часу. Для реалізації цього процесу слід включити до моделі додаткову задачу, час виконання якої дорівнює одиниці. Отже, кількість маркерів, накопичена у позиції P_4 відповідної псевдозадачі A_0, буде вказувати на кількість невикористаних тактів процесору за період виконання T.

3.5 Принцип виділення квантів

В процесі виконання задач велике значення відіграє точність моделювання. Для цього необхідно перейти від реального часу виконання в область квантів – поділити загальний час моделювання T на певну кількість квантів, кожен з яких відповідає певному часовому інтервалу.

Процесорний час також виділяється квантами.

Чим менше величина кванта, тим точніше задається виділений завданню час.

Якій задачі надати квантовий процесорній час визначає планувальник. Виділений час визначається з точністю до кванта. Чим на більшу кількість квантів

розбивається інтервал моделювання T, тим точніше результати моделювання. При збільшені точності, збільшується і кількість тактів моделювання, яка відповідає числу квантів і початковій кількості маркерів в позиції P_0.

3.6 Приклад моделювання

У прикладі представлена робота трьох задач реального часу: A_1, A_2, A_3. Задачі мають відповідні часові характеристики:

$$A_1: r_1, e_1, d_1 = 0, 3, 10$$
$$A_2: r_2, e_2, d_2 = 2, 6, 14$$
$$A_3: r_3, e_3, d_3 = 4, 4, 12$$

Час циклу виконання T=14.

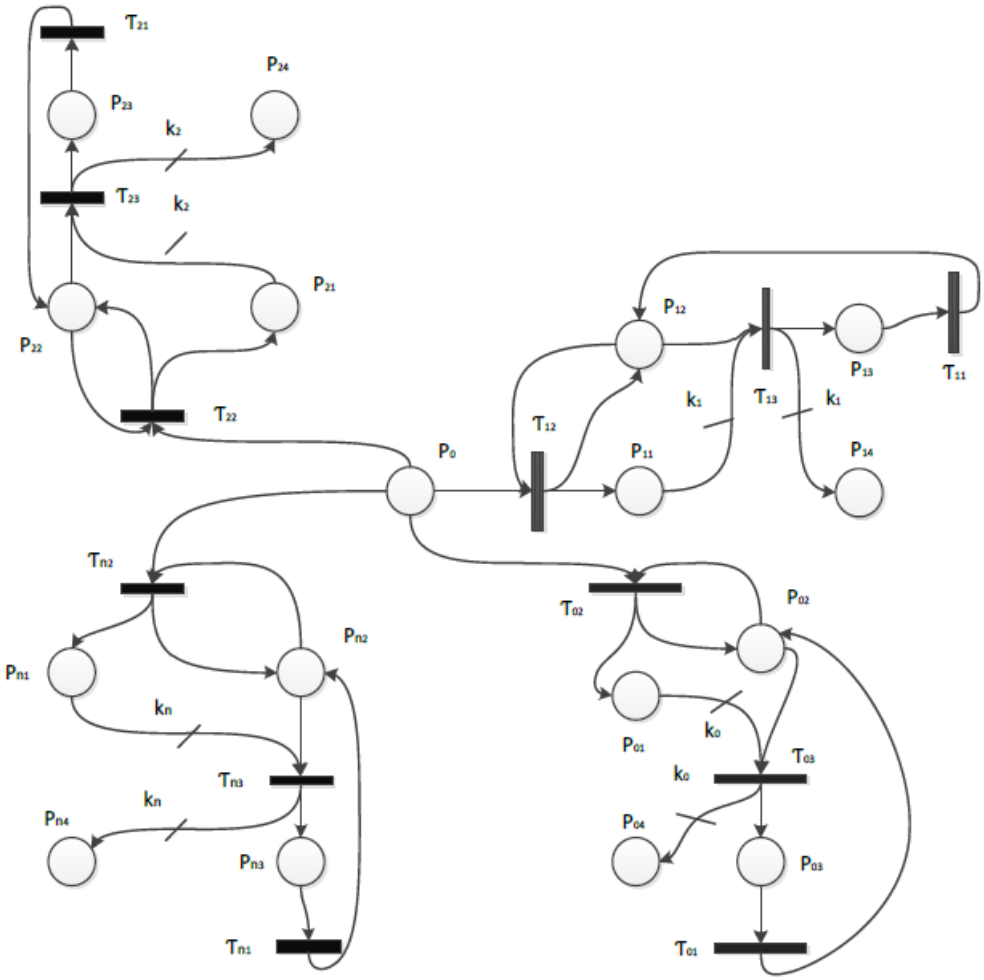

Рисунок 3.3 – Модель виконання n задач

Принциповим є вибір планувальника для керування потоком задач. Для цього прикладу виберемо планувальник із алгоритмом планування EDF з витисканням.

Матриця інцедентності для і задачі представлена у таблиці 3.5.

Початкові маркування задач за умови, що задача першої запускається на виконання:

$$A_1 = [14\ 0\ 0\ 1]$$

$$A_2 = [14\ 0\ 0\ 1]$$

$$A_3 = [14\ 0\ 0\ 1]$$

Умова першого виконання задачі при представлені маркувань пов'язана с тим, що позиція P_0 спільна для всіх задач, тому вона може змінюватись на будь-якому такті виконання.

Таблиця 3.5 – Загальна матриця

τ_1					-
τ_2					
τ_3					

Константи, які додаються до маркування сітки для отримані нового маркування:

$$\tau_{i1}: [0\ 0\ 1\ -1\ 0]$$

$$\tau_{i2}: [-1\ 1\ 0\ 0\ 0]$$

$$\tau_{i3}: [0\ -k_i\ -1\ 1\ k_i]$$

Результат роботи моделі виконання задач наведений у таблиці 3.6.

Першою починає виконуватись задача A_1 та виконується до свого завершення, яке відбувається на 3 такті. Далі на 4 такті починає виконуватись задача A_2, але вона переривається задачею A_3 на наступному такті. Задача A_3 виконується повність до завершення на 8 такті, після чого передає процесорний час задачі A_2, яка завершується на 13 такті. Оскільки період виконання T=14, то на 14 такті виконується тільки псевдозадача A_0, яка рахує час простою процесора.

Таблиця 3.6 – Результат роботи моделі

Номер такту	Перехід	Початкове маркування	Результат запуску	Спрацювання переходу	Нове маркування
1	τ_{11} τ_{12} τ_{13}	1 4 0 0 1 0 1 4 0 1 0 0 1 3 1 1 0 0	1 4 0 1 0 0 1 3 1 1 0 0 1 3 -2 0 1 3	+ + -	1 4 0 1 0 0 1 3 1 1 0 0 1 3 1 1 0 0
2	τ_{11} τ_{12} τ_{13}	1 3 1 1 0 0 1 3 1 1 0 0 1 2 2 1 0 0	1 2 1 2 -1 0 3 2 1 0 0 1 2 -1 0 1 3	- + -	1 3 1 1 0 0 1 2 2 1 0 0 1 2 2 1 0 0
3	τ_{11} τ_{12} τ_{13}	1 2 2 1 0 0 1 2 2 1 0 0 1 1 3 1 0 0	1 2 2 2 -1 0 1 1 3 1 0 0 1 1 0 0 1 3	- + +	1 2 2 1 0 0 1 1 3 1 0 0 1 1 0 0 1 3
4	τ_{21} τ_{22} τ_{23}	1 1 0 0 1 0 1 1 0 1 0 0 1 0 1 1 0 0	1 1 0 1 0 0 1 0 1 1 0 0 1 0 -5 0 1 6	+ + -	1 1 0 1 0 0 1 0 1 1 0 0 1 0 1 1 0 0
5	τ_{31} τ_{32} τ_{33}	1 0 0 0 1 0 1 0 0 1 0 0 9 1 1 0 0	1 4 2 -1 0 9 1 1 0 0 9 -3 0 1 4	+ + -	1 0 0 1 0 0 9 1 1 0 0 9 1 1 0 0
6	τ_{31} τ_{32} τ_{33}	9 1 1 0 0 9 1 1 0 0 8 2 1 0 0	9 1 2 -1 0 8 2 1 0 0 8 -2 0 1 4	- + -	9 1 1 0 0 8 2 1 0 0 8 2 1 0 0
7	τ_{31} τ_{32} τ_{33}	8 2 1 0 0 8 2 1 0 0 7 3 1 0 0	8 2 2 -1 0 7 3 1 0 0 7 -1 0 1 4	- + -	8 2 1 0 0 7 3 1 0 0 7 3 1 0 0
8	τ_{31} τ_{32} τ_{33}	7 3 1 0 0 7 3 1 0 0 6 4 1 0 0	7 3 2 -1 0 6 4 1 0 0 6 0 0 1 4	- + +	7 3 1 0 0 6 4 1 0 0 6 0 0 1 4
9	τ_{21} τ_{22} τ_{23}	6 1 1 0 0 6 1 1 0 0 5 2 1 0 0	6 1 2 -1 0 5 2 1 0 0 5 -4 0 1 6	- + -	6 1 1 0 0 5 2 1 0 0 5 2 1 0 0
10	τ_{21} τ_{22} τ_{23}	5 2 1 0 0 5 2 1 0 0 4 3 1 0 0	5 2 2 -1 0 4 3 1 0 0 4 -3 0 1 6	- + -	5 2 1 0 0 4 3 1 0 0 4 3 1 0 0
11	τ_{21} τ_{22} τ_{23}	4 3 1 0 0 4 3 1 0 0 3 4 1 0 0	4 3 2 -1 0 3 4 1 0 0 3 -2 0 1 6	- + -	4 3 1 0 0 3 4 1 0 0 3 4 1 0 0
12	τ_{21} τ_{22} τ_{23}	3 4 1 0 0 3 4 1 0 0 2 5 1 0 0	3 4 2 -1 0 2 5 1 0 0 2 -2 0 1 6	- + -	3 4 1 0 0 2 5 1 0 0 2 5 1 0 0
13	τ_{21} τ_{22} τ_{23}	2 5 1 0 0 2 5 1 0 0 1 6 1 0 0	2 5 2 -1 0 1 6 1 0 0 1 0 0 1 6	- + +	2 5 1 0 0 1 6 1 0 0 1 0 0 1 6
14	τ_{01} τ_{02} τ_{03}	1 0 0 1 0 1 0 1 0 0 0 1 1 0 0	1 0 1 0 0 0 1 1 0 0 0 0 1 0 1	+ + +	1 0 1 0 0 0 1 1 0 0 0 0 0 1 1

ВИСНОВКИ ДО РОЗДІЛУ 3

Даний розділ демонструє ідею моделі виконання задач, наводить особливості реалізації сітки Петрі. Теоретичний опис дозволяє скласти структуру для подальшої розробки програмної моделі виконання задач. Також наводиться приклад роботи трьох задач реального часу, який можна використовувати для попередньої перевірки програмної моделі. Під попередньою перевіркою розуміється процес усунення ймовірних недоліків, які можуть виникати під час розробки і не стосуються аналізу роботи моделі.

4. ПРОГРАМА МОДЕЛІ ВИКОНАННЯ ЗАДАЧ

Програмна реалізація моделі виконання задач є складовою задачею. Тому доцільно розглянути кожну частину програми та їх взаємодію. Мета розділення програми на компоненти – зробити систему більш гнучкою, що дозволить легко проводити модифікації. Також, аналізувати окремі компоненти більш доцільно в процесі розробки.

Програма поділяється на такі компоненти:

- модуль керування матрицями відповідає за створення та відображення матриць, математичні операції з матрицями;
- об’єкт задач створює нові задачі із вказаними параметрами;
- модуль-планувальник визначає яка задача повинна виконуватись, керує перериваннями та поновленнями задач;
- головна програма.

4.1 Реалізація модуля керування матрицями

Матриця представлена у вигляді структури, яка обмежена цілочисельним типом – це означає, що внутрішня реалізація однаково працює для різних типів даних, які мають спільну основу, в даному випадку, тип Int, який може мати різну кількість байтів для запису змінної. Обмеження типів дозволяє не дублювати код, якщо буде необхідність змінити тип даних, який використовує елемент матриці.

Елементи матриці зберігаються у векторі. Це дає змогу виконувати деякі операції набагато простіше, ніж у двовимірному масиві. Розмір матриці задається при її створені, при цьому, кількість рядків та стовпчиків є приватними константами, а дізнатись розмірність матриці можливо за допомогою іншої динамічної змінної (властивості) – вона не зберігає значення, а тільки повертає його, без можливості редагування.

Індексування відбувається за допомогою трьох функцій-індексаторів, які дозволяють отримувати елемент матриці за допомогою двох індексів, або окремий рядок чи стовпчик за єдиним індексом.

Порівняння матриць реалізовано визначенням протоколу Equitable (Порівняльник). Однаковими вважаються матриці однакової розмірності, в яких елементи на однакових позиціях рівні.

Арифметичні операції додавання та віднімання реалізовані статичними функціями зі знаками “+” та “-”, які отримують дві матриці як параметри та повертають нову матрицю як результат. Для цих функцій використовуються спеціальний операнд precondition, який перевіряє чи можна використовувати дану операцію для вибраних матриць. В даному випадку перевіряється однакова розмірність вхідних матриць.

Операція додавання представлена двома функціями, які повертають однаковий результат. Функція matrixMultiply(by: Matrix) реалізує класичне множення матриць, яке доцільне для матриць невеликої розмірності. Складність множення визначається як $O(n^3)$. Для збільшення швидкості підрахунків слід використовувати функцію strassenMatrixMultiply(by: Matrix), яка розбиває матрицю на вісім підматриць на рекурсивно виконує множення, зо дозволяє отримати результат $O(2^{2.82})$.

Також визначені дві функції для виводу матриць на екран. Перша реалізує властивість description та є текстовим відображенням класу. Ця функція лише перелічує елементи матриці. Друга функція виводу іменує матрицю для більш наглядного вигляду та відмінності однієї матриці від іншої. [8]

4.2 Об’єкт керування задачами

Задача створюється ініціалізатором з такими параметрами:

- id – ідентифікатор задачі дозволяє відрізняти одну задачу від іншої;
- r – момент часу, коли задача надходить для виконання;
- e – час виконання задачі;

- d – крайній термін, до якого задача повинна завершити роботу;
- T – період, з яким виконується задача.

Параметри r та d мають періодичність, тому змінні abs_r та abs_d вказують абсолютний час. При проходженні кожного періоду виконання, ці змінні обчислюються знову.

Змінна counter зберігає кількість тактів, на яких задача виконувалась.

У масиві, елементами якого є об'єкти задач, реалізована функція getHyperperiod(). Результатом роботи цієї функції є кількість тактів для виконання всіх задач.

4.3 Модуль-планувальник

Розроблена модель виконання задач повинна працювати з будь-яким планувальником, тому доцільно реалізувати протокол (інтерфейс) – абстрактний клас, який описує загальний функціонал планувальника, але не реалізацію. Такий підхід дозволяє абстрагуватись від реалізації при зверненні до планувальника.

Протокол Scheduler містить властивість tasks типу даних масив задач та функцію run(), яка запускає планувальник та повертає чергу тактів з відповідними запланованими задачами.

В даній моделі використовується планувальник Earliest Deadline First (EDF). Алгоритм планування по найближчому крайньому терміну є одним з динамічних алгоритмів.

Планувальник шукає серед готових до виконання задач ту, крайній термін завершення якої є мінімальним.

EDF має дві реалізації:

- з витискання – задача, що виконується може бути витіснена більш пріоритетною задачею
- без витискання – задача, яка почала виконуватись, не може бути перервана іншої задачею.

Використання обох типів планувальника EDF обумовлено необхідністю порівняння роботи моделі виконання задач з різними планувальниками.

Якщо існує можливість виконати набір незалежних завдань, кожна з яких характеризується часом настання, часом виконання і крайнім терміном завершення, так, щоб гарантувати виконання всіх завдань до крайнього терміну, цей алгоритм призначить завдання до виконання так, щоб всі задачі були завершені до їх крайнього терміну.

При призначенні періодичних процесів, які мають крайні терміни еквівалентні їх періодів, алгоритм планування по найближчому терміну завершення використовує завантаження на 100%. Отже, перевірка на можливість планування для даного алгоритму буде:

$$U = \sum_{i-1}^{n} \frac{C_i}{T_i} \leq 1, \quad (4.1)$$

де $\{C_i\}$ - найгірший час виконання n процесів, а $\{T_i\}$ - їх відповідний період між термінами настання.

Тобто, призначення по найближчому терміну завершення гарантує, що всі крайні терміни будуть дотримані, за умови що загальна завантаження процесора не перевищує 100%. У порівнянні з використанням фіксованих пріоритетів, планування по найближчому терміну завершення гарантує дотримання всіх крайніх термінів при більшому завантаженні.

Програмна реалізація планувальника поділяється на частини.

Перш за все, треба визначити можливість планування для наданого набору задач. Для цього виконується функція cpuUtil(), яка повертає число типу Float. Якщо результат менше 1, то планування можливе, в іншому випадку планування завершується з помилкою.

Далі розраховується гіпер-період виконання всіх задач. За розрахунок відповідає об’єкт задач, але результат передається планувальнику.

Для запуску будь-якого планувальника необхідно звернутись до функції run(). Раніше визначені параметри є приватними і не можуть бути змінені ззовні, отже,

планувальник завжди починає роботу з гарантовано правильним списком задач та значенням гіпер-періоду.

Планувальник EDF зберігає індекс поточної задачі, що виконується. Так як не на кожному такті задачі надається процесорний час, змінна nowExecutedIndex є опціональною та не має початкого значення.

Робота планувальника виконується, доки не буде визначений стан кожного такту процесора в межах обчисленого гіпер-періоду. Отже, планувальник обробляє кожен такт процесора за допомогою циклу for. Спочатку зі списку задач вибираються ті, які поступили для виконання на одному з попередніх тактів або на поточному та не є завершеними.

Якщо маємо хоча б одну таку задачу, вибираємо з доступних для виконання задачу із найменшим крайнім терміном. Вибрана для виконання задача збільшує лічильник тактів виконання на одиницю. Для перевірки задачі на завершення на початку кожного такту необхідно перевірити присутність задачі, що виконувалась. Якщо така задача існує та значення лічильника дорівнює часу виконання цієї задачі, така задача помічається як завершена.

Завершена задача не попадає у список доступних для виконання, доки не завершиться її період виконання. При завершені періоду, задача знов помічається як незавершена, лічильнику тактів виконання присвоюється 0, а значення абсолютних часових характеристик обчислюються знов для наступного періоду.

В разі використання планувальника EDF без витискання, пошук задачі для виконання на обраному такті виконується тільки, якщо попередня задача була завершена.

Для формування результатів роботи планувальника на кожному такті формується комірка у черзі. Комірка містить відповідний номер такту та опціональну задачу. Сформована комірка наприкінці кожному такту записується у масив, який відображає чергу виконання.

Результатом роботи планувальника є черга виконання задач.

4.4 Головна програма

Використання протоколів (інтерфейсів) для реалізації планувальника дозволяє створювати параметризовану програму. Параметром виступає тип планувальника.

Згідно схеми моделювання, головна програма містить 3 основні компоненти:

– генератор потоку задач;
– планувальник задач;
– модель виконання задач.

В якості генератору задач використовується масив задач. Це дозволяє швидко змінювати пакети задач для планування. Такий підхід є найкращім для тестування та може бути легко замінений реальними задачами.

Процес планування виконує окрема функція, яка приймає масив задач та передає його обраному при ініціалізації планувальнику. Результатом роботи цієї функції є черга виконання.

Для покращення читання коду черга представлена додатковим типом даним Queue, який по факту є масивом комірок виконання.

Після отримання черги виконання необхідно визначити константні вектори для кожного переходу кожної задачі. Для цього слід обчислити матриці інцедентності кожної задачі. Це досягається проходження циклом for по масиву задач. На кожній ітерації викликається функція getM(task: Task), яка приймає задачу в якості параметра і розраховує відповідну матрицю інцедентності. Слід зазначити, що результуючі матриці мають однакову структуру та відрізняються лише параметром третього переходу, де використовується час виконання задачі.

Маючи матрицю інцедентності можна розрахувати вектори переходів. Функція getVector(by: Matrix<Int>) вираховує масив векторів для кожного переходу обраної задачі.

Результатом роботи циклу є двовимірний масив константних векторів. Кожен рядок масиву відповідає окремій задачі.

Далі необхідно визначити початкові маркування для кожної задачі.

Для цього масив задач перетворюється за допомогою функції map, яка є стандартною для масивів, вона дозволяє виконувати однакові перетворення для кожного елементу масиву. Наприклад, масив цілих чисел [2, 5, 7, 6] можна перетворити на масив строк за допомогою виклику [2, 5, 7, 6].map { String($0) }.

Перетворення масиву задач до вектору початкових маркувань виконується як:

$$tasks.map\{\ Matrix < Int > (rows: [[[\$0.T, 0\ \ 0, 1, 0]]\ \}$$

Окремо створюється псевдо-задача з часом виконання e=1. Початкове маркування цієї задачі дорівнює [1, 0, 0, 1, 0]. Вектор констант визначається також за допомогою виклику getVector(by: getM(by: singletask)).

4.5 Модель виконання задач

Коли всі початкові значення визначені, модель виконання задач починає виконуватись. Оскільки черга виконання вже визначена планувальником, модель працює в циклі, послідовно обробляючи кожен елемент черги.

Спочатку поточний елемент черги записуються в окрему змінну для спрощення запису.

Всередині основного циклу виконання працює цикл переходів. Так як запропонована модель має по три переходи для кожної задачі, то шапка циклу має вигляд for(i in 1…3).

Всередині циклу переходів перевіряється чи поточна комірка черги має задачу для виконання.

Якщо на поточному такті ніяка задача не виконується, використовуємо псевдо-задачу для підрахунку тактів, на яких процесор не виконує роботу. Відповідно до поточного переходу вибирається константний вектор. Далі обчислюється нове маркування.

Для перевірки нового маркування на дозволеність використовується окрема функція. Оскільки маркування представлено вектором, а вектор матрицею з одним рядком, то перевірка відбувається за допомогою розширення функціоналу матриці.

Якщо матриця не містить від’ємних елементів, перехід є дозволеним. У іншому випадку – перехід заборонений.

Визначається результуюче маркування для поточного переходу. Якщо перехід дозволений, записується нове маркування, якщо ні – попереднє. Маркування псевдо-задачі змінюється на результуюче.

У випадку, коли черга має задачу для виконання на поточному такті, береться ідентифікатор цієї задачі. Так само як і для псевдо-задачі, вибирається константний вектор. Але в цей раз враховується номер задачі. Нове маркування також вибирається з масиву маркувань для виконуваної задачі. Якщо перехід дозволений, береться нове маркування. Якщо перехід заборонений, береться попереднє маркування для обраної задачі.

Так як позиція P_0 є спільною для всіх задач, слід оновити значення P_0 для маркувань всіх задач.

4.6 Вивід результатів

Запропонований алгоритм потребує впровадження системи виводу результатів.

Існує декілька способів реалізації виводу результатів:

1. Вивід результатів у консолі. Найпростіший спосіб виводу, але при кожному запуску алгоритму дані перезаписуються. Для збереження необхідно копіювати дані з консолі до окремого файлу. Також такий спосіб не структурує дані, а тільки записує їх у рядковому представлені.

2. Запис результатів у файл формату csv для подальшого аналізу. Цей спосіб також має просту реалізацію, але автоматично зберігає дані у структурований файл. Це дозволяє легко проводити аналіз після виконання.

3. Графічні представлення результатів. Створення графічного середовища для запису результатів и подальшого аналізу дає змогу реалізувати будь-яку структуру, але потребує додаткової розробки, яка не є метою роботи.

З аналізу варіантів реалізації виводу можна зробити висновок, що використання csv-файлів для логування є найкращім способом у даному випадку. Так як метою роботи є розробка програмної моделі, а не програми з графічним інтерфейсом, то цей спосіб є оптимальним.

За реалізацію виводу відповідає окремий клас Recorder, який є універсальним засобом логування. Цей клас працює з об'єктами, які реалізують протокол Record.

Наприклад, якщо необхідно записати результат роботи планувальника на 5 такті, то слід створити новий клас Scheduler: Record, який має дві властивості: номер задачі, що виконується, та такт виконання. Реалізація протоколу логування досягається за рахунок додавання властивості csvDescription типу String. Ця властивість відображає текстову інформацію про клас, яка буде записуватись у csv-файл.

Recorder має приватне сховище записів – масив типу Record. Для додавання записів використовується метод addRecord(:Record). Метод reset() видаляє всі елементи із сховища. Коли колекція записів сформована, для запису результуючої строки у файл викликається метод createCSV(), який створює унікально іменований файл формату csv та зберігає його у папці проекту.

ВИСНОВКИ ДО РОЗДІЛУ 4

Розділ 4 розкриває деталі розробки програмної моделі виконання задача. По перше, наводиться список програмних елементів, які беруть участь у функціонуванні моделі. Також, описується процес роботи моделі виконання задач та її компонентів, вказуються особливості реалізації, аргументуються вибрані підходи до розробки.

5. ТЕСТУВАННЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

5.1 Опис методів тестування

Для коректності тестування і аналізу результатів слід обрати методи, які будуть виконуватись для кожного тестового набору. Оскільки модель виконання задач перевіряє розклад планувальника, це означає, що слід використати декілька планувальників для однакового набору задач. В даному випадку, тестування виконується за допомогою EDF планувальника з витисканням та EDF планувальника без витискання.

Результатом роботи моделі виконання сітки Петрі є три результуючі таблиці:

1. Перша таблиця демонструє процес виконання сітки Петрі на кожному такті. Висновки щодо результатів роботи моделі формуються окремо в більш компактні таблиці.
2. Друга таблиця наводить розклади обох планувальників для побудови графіків виконання.
3. Третя таблиця містить відповідність задачі та часу її завершення. Аналіз її записів визначає, чи може пакет задач виконатись з відповідним планувальником.

5.2 Аналіз базового прикладу

Спочатку перевіримо роботу реалізованої моделі виконання задач із параметрами, які наведені у другому розділі, щоб переконатись у правильності побудови результуючої таблиці.

Пакет задач для виконання складається із задач $A_1(r\colon 0, e\colon 3, d\colon 10)$, $A_2(r\colon 2, e\colon 6, d\colon 14)$, $A_3(r\colon 4, e\colon 4, d\colon 12)$, період виконання T=14.

У таблиці 4.1 наводиться розклад алгоритму EDF з витисканням. У таблиці 4.2 – розклад EDF без витискання.

Таблиця 4.1 – Розклад EDF з витисканням

Такт	Номер задачі
0	1
1	1
2	1
3	1
3	2
4	2
4	3
5	3
6	3
7	3
8	3
8	2
9	2
10	2
11	2
12	2
13	2
14	0

Таблиця 4.2 – Розклад EDF без витискання

Такт	Номер задачі
0	1
1	1
2	1
3	1
3	2
4	2
4	3
5	3
6	3
7	3
8	3
8	2
9	2
10	2
11	2
12	2
13	2
14	0

Для покращення сприйняття наводяться відповідні графічні вигляди розкладів на рис. 4.1 та рис. 4.2.

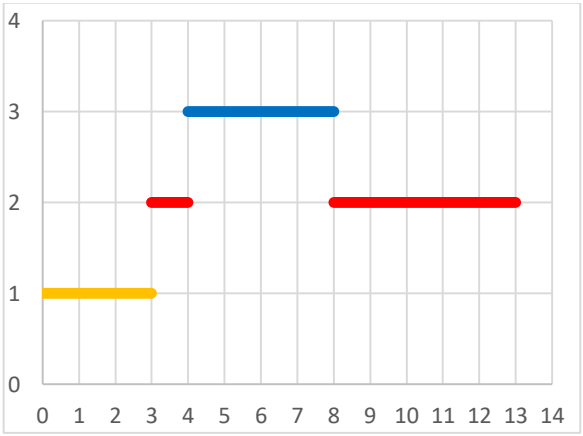

Рисунок 4.1 – Графік розклад EDF планувальника з витисненням

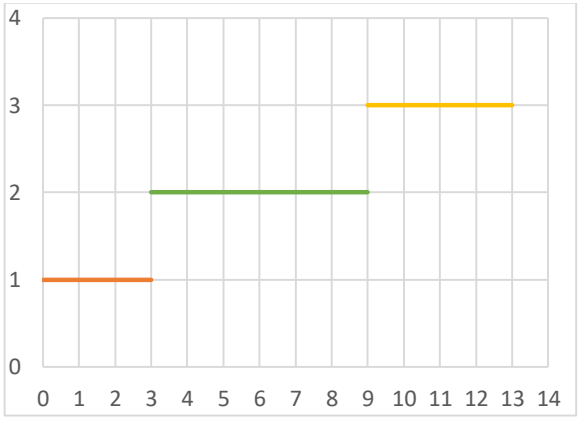

Рисунок 4.2 – Графік розклад EDF планувальника з витисненням

У таблиці 4.3 наведені результати роботи моделі виконання сітки Петрі.

Таблиця 4.3 – Виконання моделі сітки Петрі задач A_1, A_2, A_3 з алгоритмом планування EDF з витисканням

Номер такту	Перехід	Початкове маркування	Результат запуску	Спрацювання	Нове маркування
1	tau11	14 0 0 1 0	14 0 1 0 0	+	14 0 1 0 0
	tau12	14 0 1 0 0	13 1 1 0 0	+	13 1 1 0 0
	tau13	13 1 1 0 0	13 -2 0 1 3	-	13 1 1 0 0
2	tau11	13 1 1 0 0	13 1 2 -1 0	-	13 1 1 0 0
2	tau12	13 1 1 0 0	12 2 1 0 0	+	12 2 1 0 0
2	tau13	12 2 1 0 0	12 -1 0 1 3	-	12 2 1 0 0
3	tau11	12 2 1 0 0	12 2 2 -1 0	-	12 2 1 0 0
3	tau12	12 2 1 0 0	11 3 1 0 0	+	11 3 1 0 0
3	tau13	11 3 1 0 0	11 0 0 1 3	+	11 0 0 1 3
4	tau21	11 0 0 1 0	11 0 1 0 0	+	11 0 1 0 0
4	tau23	11 0 1 0 0	10 1 1 0 0	+	10 1 1 0 0
4	tau23	10 1 1 0 0	10 -5 0 1 6	-	10 1 1 0 0
5	tau31	10 0 0 1 0	10 0 1 0 0	+	10 0 1 0 0
5	tau32	10 0 1 0 0	9 1 1 0 0	+	9 1 1 0 0
5	tau33	9 1 1 0 0	9 -3 0 1 4	-	9 1 1 0 0
6	tau31	9 1 1 0 0	9 1 2 -1 0	-	9 1 1 0 0
6	tau32	9 1 1 0 0	8 2 1 0 0	+	8 2 1 0 0
6	tau33	8 2 1 0 0	8 -2 0 1 4	-	8 2 1 0 0
7	tau31	8 2 1 0 0	8 2 2 -1 0	-	8 2 1 0 0
7	tau32	8 2 1 0 0	7 3 1 0 0	+	7 3 1 0 0
7	tau33	7 3 1 0 0	7 -1 0 1 4	-	7 3 1 0 0
8	tau31	7 3 1 0 0	7 3 2 -1 0	-	7 3 1 0 0
8	tau32	7 3 1 0 0	6 4 1 0 0	+	6 4 1 0 0
8	tau33	6 4 1 0 0	6 0 0 1 4	+	6 0 0 1 4
9	tau21	6 1 1 0 0	6 1 2 -1 0	-	6 1 1 0 0
9	tau22	6 1 1 0 0	5 2 1 0 0	+	5 2 1 0 0
9	tau23	5 2 1 0 0	5 -4 0 1 6	-	5 2 1 0 0
10	tau21	5 2 1 0 0	5 2 2 -1 0	-	5 2 1 0 0
10	tau22	5 2 1 0 0	4 3 1 0 0	+	4 3 1 0 0
10	tau23	4 3 1 0 0	4 -3 0 1 6	-	4 3 1 0 0
11	tau21	4 3 1 0 0	4 3 2 -1 0	-	4 3 1 0 0
11	tau22	4 3 1 0 0	3 4 1 0 0	+	3 4 1 0 0
11	tau23	3 4 1 0 0	3 -2 0 1 6	-	3 4 1 0 0
12	tau21	3 4 1 0 0	3 4 2 -1 0	-	3 4 1 0 0
12	tau22	3 4 1 0 0	2 5 1 0 0	+	2 5 1 0 0
12	tau23	2 5 1 0 0	2 -1 0 1 6	-	2 5 1 0 0
13	tau21	2 5 1 0 0	2 5 2 -1 0	-	2 5 1 0 0
13	tau22	2 5 1 0 0	1 6 1 0 0	+	1 6 1 0 0
13	tau23	1 6 1 0 0	1 0 0 1 6	+	1 0 0 1 6
14	tau01	1 0 0 1 0	1 0 1 0 0	+	1 0 1 0 0
14	tau02	1 0 1 0 0	0 1 1 0 0	+	0 1 1 0 0
14	tau03	0 1 1 0 0	0 0 0 1 1	+	0 0 0 1 1

З аналізу процесу виконання бачимо, що результат відповідає тому, що наводиться у розділі 3.

Проаналізуємо моменти завершення кожної задачі відповідно кожного з наведених типів планувальників, наведені у таблиці 4.4 та таблиці 4.5.

Таблиця 4.4 – Завершення задач при розкладі EDF з витисканням

Номер задачі	Такт завершення
1	3
3	8
2	13

Таблиця 4.5 – Завершення задач при розкладі EDF без витискання

Номер задачі	Такт завершення
1	3
2	8
3	13

Порівняємо моменти завершення задач з їх крайніми термінами.

Для розкладу EDF з витисканням, результат наступний:

$$d_1 = 3 < 10$$

$$d_2 = 13 < 14$$

$$d_3 = 8 < 12$$

Для розкладу EDF без витискання:

$$d_1 = 3 < 10$$

$$d_2 = 9 < 14$$

$$d_3 = 13 < 12$$

В обох випадках задачі виконуються в межах періоду T, але при плануванні алгоритмом EDF без витискання задача A_3 не виконується до крайнього терміну. Отже система працює з помилкою.

Слід зазначити, що при використанні будь-якого планувальника, можна згенерувати точний результат.

5.3 Тестування на різних наборах даних

Для підтвердження факту функціонування моделі виконання задач, слід провести декілька запусків для різних наборів завдань. Розглянемо послідовно набори з 10 та 15 задач, період виконання T=100.

5.3.1 Тестування на наборі з десяти задач

Нехай заданий набір задач:

$$A_1(r:0, e:, d:)$$
$$A_2(r:2, e:, d:)$$
$$A_3(r:4, e:, d:)$$
$$A_4(r:34, e:, d:)$$
$$A_5(r:10, e:, d:)$$
$$A_6(r:4, e:, d:)$$
$$A_7(r:8, e:, d:)$$
$$A_8(r:45, e:, d:)$$
$$A_9(r:21, e:, d:)$$
$$A_{10}(r:21, e:, d:)$$

Представимо розклад планувальників EDF з витисканням та без витискання у додатку 1.

Відповідні графіки виконання зображені на рис. 4.3 і рис. 4.4.

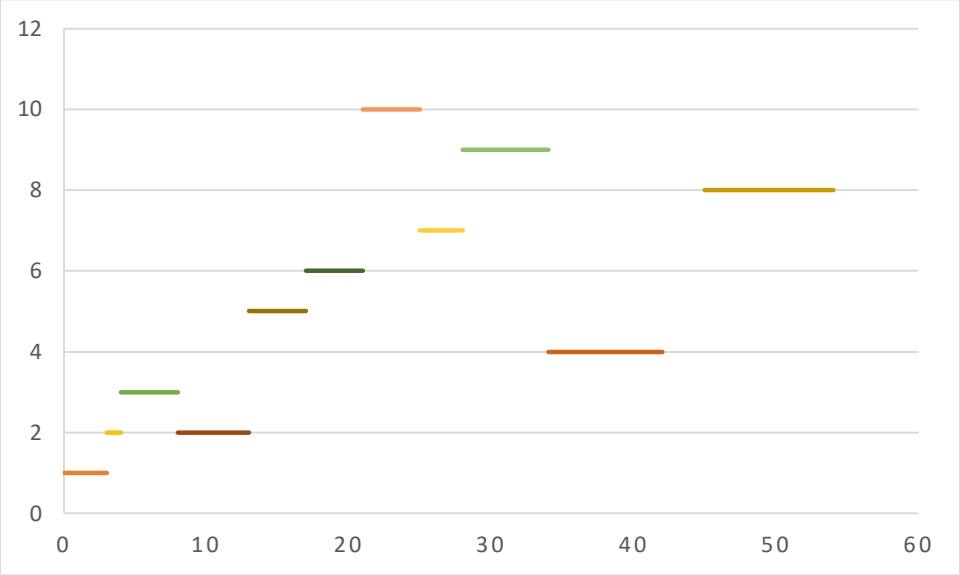

Рисунок 4.3 – Графік розкладу EDF з витисканням

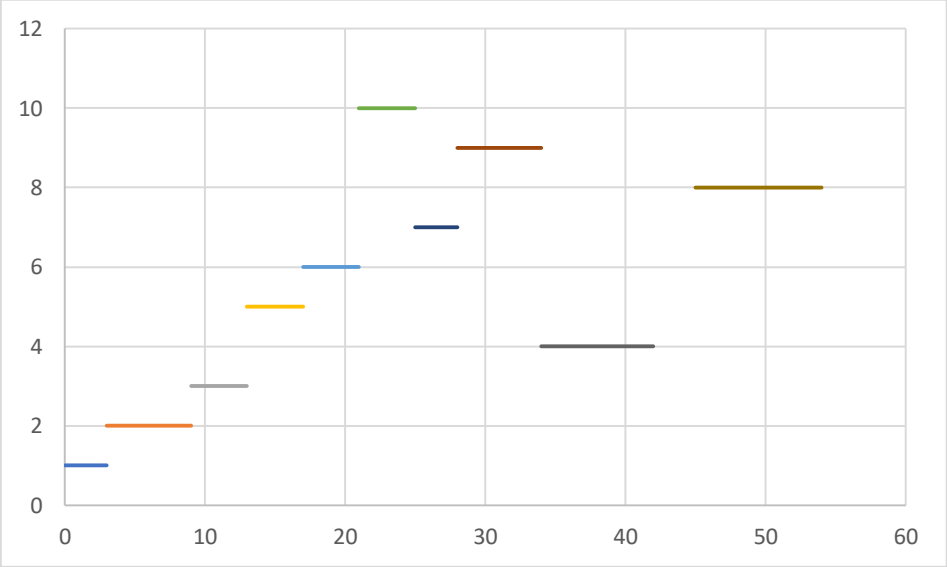

Рисунок 4.4 – Графік розкладу EDF без витискання

Як видно на графіках, задачі при розкладі за допомогою EDF з витисканням виконуються майже однаково, ніж при плануванні EDF без витискання і у обох випадках задачі завершують виконання в межах періоду T. Різниця у виконанні задачі A_2, яка переривається, щоб надати процесорний час задачі A_3.

Але це не свідчить про те, що кожна задача завершила виконання до свого крайнього терміну. Для цього проаналізуємо моменти закінчення задач.

З результатів, наведених у таблиці 4.7, видно, що при використанні планувальника EDF без витискання всі задачі у даному випадку завершують виконання до крайнього терміну. Це свідчить про те, що всі 10 задач можуть бути гарантовано виконані у межах періоду T з заданими часовими характеристиками.

Таблиця 4.7 – Таблиця завершення задач при плануванні EDF з витисканням

Task ID	Completion	Deadline	State
1	3	10	true
3	8	12	true
2	13	14	true
5	17	21	true
6	21	23	true
10	25	32	true
7	28	54	true
9	34	67	true
4	42	77	true
8	54	85	true

Проаналізуємо моменти завершення задач, вказані у таблиці 4.8. Результати дуже схожі за винятком того, що задача A_2 не призупинила виконання для передачі управління задачі A_3. При цьому A_2 завершила виконання до крайнього терміна в обох випадках з різними планувальниками, а A_3 не вистачило одного такту виконання, тому ця задача не встигає завершити виконання до крайнього терміну. Отже гарантовано планування EDF без витискання не дає змогу виконати набір задач без отримання помилки.

Таблиця 4.8 – Таблиця завершення задач при плануванні EDF без витискання

Task ID	Completion	Deadline	State
1	3	10	true
2	9	14	true
3	13	12	false
5	17	21	true
6	21	23	true
10	25	32	true
7	28	54	true
9	34	67	true
4	42	77	true
8	54	85	true

5.3.2 Тестування на наборі з 15 задач

Нехай заданий набір задач:

$$A_1(r:0, e:, d:)$$

$$A_2(r:2, e:, d:)$$

$$A_3(r:4, e:, d:)$$

$$A_4(r:34, e:, d:)$$

$$A_5(r:10, e:, d:)$$

$$A_6(r:4, e:, d:)$$

$$A_7(r:8, e:, d:)$$

$$A_8(r:45, e:, d:)$$

$$A_9(r:21, e:, d:)$$

$$A_{10}(r:21, e:, d:)$$

$$A_{11}(r:73, e:, d:)$$

$$A_{12}(r:52, e:, d:)$$

$$A_{13}(r:25, e:, d:)$$

$$A_{14}(r\colon 70, e\colon, d\colon)$$

$$A_{15}(r\colon 48, e\colon, d\colon)$$

Представимо розклад планувальників EDF з витисканням та без витискання у додатку 1.

Відповідні графіки виконання зображені на рис. 4.5 і рис. 4.6.

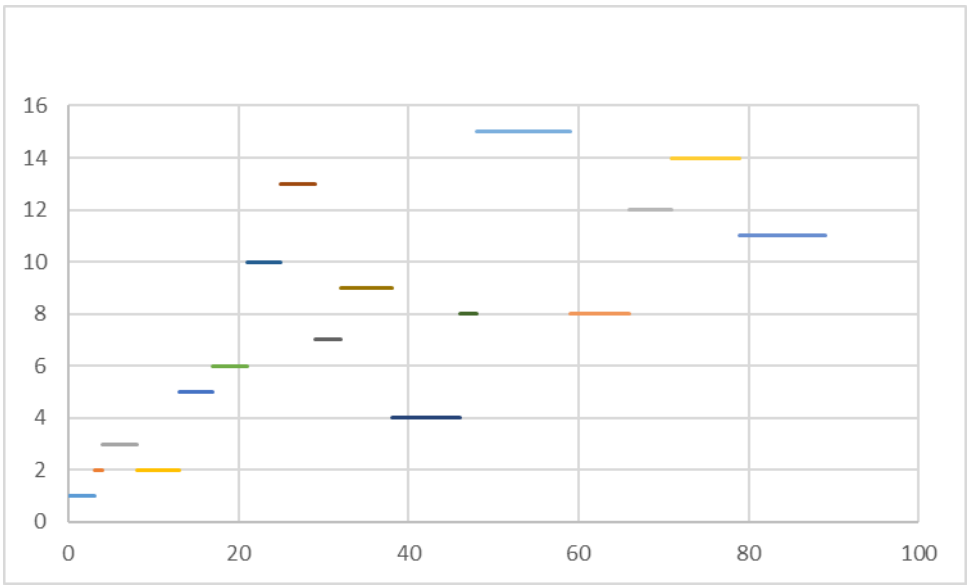

Рисунок 4.5 – Графік розкладу EDF з витисканням

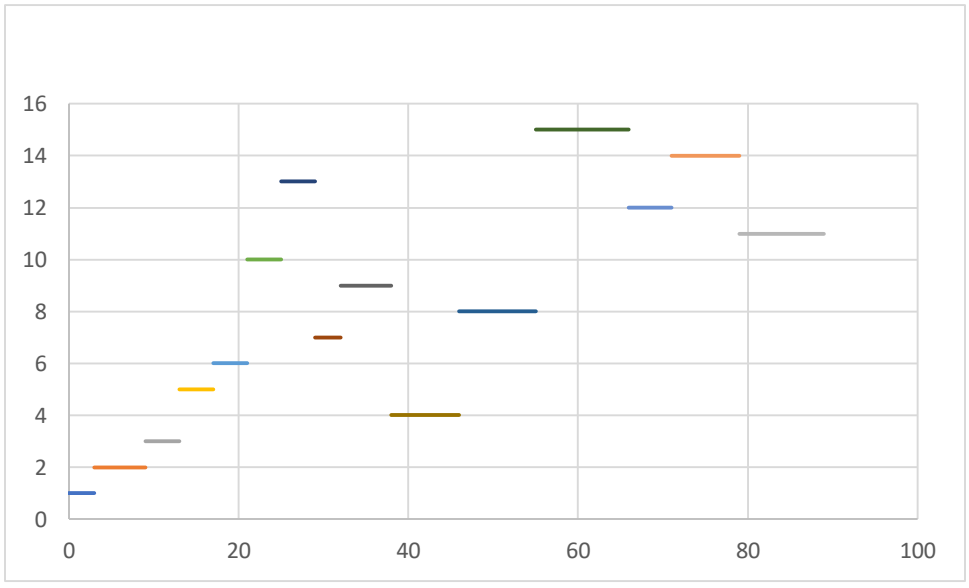

Рисунок 4.4 – Графік розкладу EDF без витискання

З аналізу наведених графічних результатів видно, що час виконання теж майже однаковий. У випадку використання EDF з витисканням задача A_2 віддає керування задачі A_3, а задача A_8 – задачі A_{15}. Попередній випадок підтвердив, що витискання можуть грати ключову роль у можливості виконання набору задач. Далі перевіримо відповідність моментів завершення задач та крайніх термінів.

З результатів, наведених у таблиці 4.9, задачі завершують виконання до своїх крайніх термінів. Отже, всі 15 задач можуть бути гарантовано виконані у межах періоду T з заданими часовими характеристиками.

Таблиця 4.9 – Таблиця завершення задач при плануванні EDF з витисканням

Task ID	Completion	Deadline	State
1	3	10	true
3	8	12	true
2	13	14	true
5	17	21	true
6	21	23	true
10	25	32	true
13	29	45	true
7	32	54	true
9	38	67	true
4	46	77	true
15	59	70	true
8	66	85	true
12	71	85	true
14	79	90	true
11	89	98	true

Проаналізуємо моменти завершення задач, вказані у таблиці 4.10. Не вдалось завершити задачу A_2 до її крайнього терміну. Ця проблема була у і попередньому тестуванні. Але слід зауважити, що виконання задачі A_8 до кінця не вплинуло на роботу наступної задачі A_{15} – не завжди відсутність переривання призводить до помилки в роботі системи.

Таблиця 4.10 – Таблиця завершення задач при плануванні EDF без витисканням

Task	Completion	Deadline	State
1	3	10	true
2	9	14	true
3	13	12	false
5	17	21	true
6	21	23	true
10	25	32	true
13	29	45	true
7	32	54	true
9	38	67	true
4	46	77	true
8	55	85	true
15	66	70	true
12	71	85	true
14	79	90	true
11	89	98	true

ВИСНОВКИ ДО РОЗДІЛУ 5

Даний розділ містить тестування моделі виконання задач на різних наборах вхідних даних при використанні планувальників EDF з витисканням та без витискання. Проведені дослідження доводять ефективність роботи моделі виконання задач, оскільки у кожному випадку можна отримати остаточну відповідь щодо можливості виконання набору задач для однопроцесорної системи. Також можна порівняти ефективність роботи планувальників у витисканні задач.

ВИСНОВКИ

Запропонована програмна модель виконання задач реального часу повинна гарантовано визначати можливість виконання вхідного набору задач за обраний період T.

Результати роботи моделі, показані у розділі 5, доводять доцільність розробки.

Для початку, результати програмної моделі були порівняні з результатами аналітичної моделі виконання задач, яка демонструється у розділі 3. Це дозволяє переконатись у відсутності помилок при написанні програми. Після отримання однакових результатів, можна підтвердити, що модель відповідає аналітичному поданню.

Доцільність використання моделі виконання задач доводиться за допомогою тестування на різних наборах даних. Для всіх тестів значення періоду T=100. При визначені періоду виконання відбувається перехід від привичного для нас часу до квантового часу. Квантування дозволяє визначити точність моделювання.

Послідовно модель запускається з різними типами планувальниками. Результатом кожного запуску є генерація трьох таблиць:

- таблиці розкладу планувальника;
- таблиці виконання задач;
- таблиці завершених задач.

Остання таблиця наводить квантові моменти завершення задач та їх крайні терміни. Порівняння двох моментів дозволяє отримати точну відповідь, чи може задача бути виконана у відведений їй термін.

Отже, можна зазначити, що модель виконання задач за допомогою сітки Петрі є доцільним інструментом при розробці систем реального часу. Гарантованість отримання результату моделювання була доведена за допомогою тестування та аналізу часів виконання задач.

Слід також зазначити, що модель є досить простою в реалізації та не містить складних циклічних розрахунків. Це є додатковою перевагою, оскільки моделювання теж потребує ресурсів процесору.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Комп’ютерні системи реального часу: навчальний посібник / Національний технічний університет України “Київський політехнічний інститут імені Ігоря Сікорського”. В.Г. Зайцев, Є.І. Цибаєв. – Київ, 2019. Електронний ресурс КПІ ім. Ігоря Сікорського: https://ela.kpi.ua/handle/123456789/29604.
2. Системы реального времени: конспект лекцій / Владим. гос. ун-т; сост. А. С. Голубев. – Владимир: Изд-во Владим. гос. ун-та, 2010 – 127 с.
3. Baker T. Multiprocessors EDF and Deadline Monotonic Shedulability Analysis // Proceeding of 24 IEEE Real – Time Systems Symposium, 2003, p. 120 – 129.
4. Сети Петри. Электронній ресурс: http://www.hpc-education.ru/files/lectures/2011/ershov/ershov_2011_lectures05.pdf
5. Питерсон Дж. Теория сетей Петри и моделирование систем. – М.: Мир, 1984. – 264 с.
6. Фальк В.Н. Введение в сети Петри и моделирование систем. Учебное пособие. Москва. 2009. Электронный ресурс: https://studfiles.net/preview/1529418
7. Стеценко І.В. Система імітаційного моделювання засобами сіток Петрі / І.В. Стеценко, О.В. Бойко / Математичні машини і системи, 2009 – № 1. – С.117 – 124.
8. Raywenderlich Swift Algorithm Club by Tutorials, 2018.
9. Karoly Nyisztor Design Patterns in Swift 5: Learn how to implement the Gang of Four Design Patterns using Swift 5.