

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ  
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ  
СІКОРСЬКОГО»  
КАФЕДРА ІНФОРМАТИКИ ТА ПРОГРАМНОЇ ІНЖЕНЕРІЇ

Курсова робота з освітнього компоненту

«Технології паралельних обчислень. Курсова робота»

**Тема: Імітаційна модель системи процесорів на основі  
формального опису мережею Петрі**

**Керівник:**

ст.викл. Дифучина Олександра Юріївна

«Допущено до захисту»

---

«\_\_\_» \_\_\_\_\_ 2024 р.

Захищено з оцінкою

---

Члени комісії:

---

---

**Виконавець:**

Панченко Сергій Віталійович

студент групи ІП-11

залікова книжка № ІП-1123

«5» грудня 2024 р.

Інна СТЕЦЕНКО

Олександра ДИФУЧИНА

**Київ – 2024**

Національний технічний університет України  
“Київський політехнічний інститут імені Ігоря Сікорського”

Кафедра інформатики та програмної інженерії

Дисципліна «Моделювання систем»

Спеціальність 121 Інженерія програмного забезпечення

Курс 4    Група ІП-11    Семестр 1

## ЗАВДАННЯ

на курсову роботу студента

Панченка Сергія Віталійовича  
(прізвище, ім'я, по батькові)

---

1. Тема роботи «Імітаційна модель системи процесорів на основі формального опису мережею Петрі»

---

2. Термін здачі студентом закінченої роботи "5" грудня 2024р.

3. Зміст розрахунково-пояснювальної записки

1. Опис 2. Псевдокод 3. Реалізація 4. Реалізація 5. Реалізація 6. Проведення експериментів над моделями. Висновки.

4. Дата видачі завдання "29" жовтня 2024 року

---

## КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів виконання курсової роботи	Термін виконання етапів роботи	Примітка
1	Отримання індивідуального завдання на курсову роботу	29.10.2024	
2	Розробка концептуальної моделі системи	29.11.2024	
3	Розробка формалізованої моделі системи	03.11.2024	
4	Алгоритмізація моделі системи та її програмна реалізація	08.11.2024	
5	Експериментальне дослідження моделі системи	13.11.2024	
6	Інтерпритація результатів моделювання, формулювання висновків та пропозицій	15.11.2024	
7	Оформлення пояснювальної записки	17.11.2024	
8	Захист КР	5.12.2024	

Студент \_\_\_\_\_ Панченко С. В.  
(підпис)

Керівник \_\_\_\_\_ Дифучина О.Ю.  
(підпис)

## АНОТАЦІЯ

Структура та обсяг роботи. Пояснювальна записка курсової роботи складається з 5 розділів, містить 2 рисунки, 2 таблиці, 1 додаток, 2 джерела.

Мета: визначення статистичних характеристик роботи багатопроцесорної обчислювальної системи, зокрема середнього значення, середнього квадратичного відхилення та закону розподілу для таких вихідних значень, як: часу виконання завдання в системі; завантаження чотирьох дисків, каналу передачі даних і процесорів; використання пам'яті; кількості завдань, які очікують виділення ресурсу, та часу їхнього очікування.

У розділі розробки концептуальної моделі було окреслено концептуальну модель та визначено необхідні властивості імітаційних алгоритмів.

У розділі аналіз розробки формалізованої моделі було описано моделі у рамках формалізму мереж Петрі.

У розділі алгоритмізації моделі та її реалізації було описано усі деталі щодо алгоритму та його реалізації.

У розділі експериментів на моделі було проведено верифікацію моделі та факторний експеримент.

У розділі інтерпретації результатів моделювання та експериментів було описано отримані результати в рамках проведеного моделювання та експериментування.

**КЛЮЧОВІ СЛОВА:** МОДЕЛЮВАННЯ СИСТЕМ, МЕРЕЖІ ПЕТРІ, БАГАТОПРОЦЕСОРНА ОБЧИСЛЮВАЛЬНА СИСТЕМА.

## ЗМІСТ

Постановка завдання.....	7
Вступ.....	9
1 Розробка концептуальної моделі.....	11
1.1 Опис задачі.....	11
1.2 Структурна схема моделі.....	11
1.3 Опис процесу.....	11
1.4 Вхідні змінні.....	12
1.5 Вихідні змінні.....	12
1.6 BPNM-діаграма.....	13
1.7 Висновки до розділу.....	15
2 Розробка формалізованої моделі.....	16
2.1 Вхідні параметри.....	17
2.2 Вихідні параметри.....	19
2.3 Діаграма.....	20
2.4 Висновки до розділу.....	23
3 Алгоритмізація моделі та її реалізація.....	24
3.1 Опис алгоритму імітації мережі Петрі.....	24
3.2 Модифікації алгоритму.....	27
3.2.1 Додавання розподілу Пуассона для генерації подій.....	27
3.2.2 Вдосконалення роботи з NetLibrary.....	27
3.2.3 Покращення графічного інтерфейсу.....	28
3.3 Клас CourseWorkNet. Протокол подій.....	28
3.3.1 Опис атрибутів.....	28
3.3.2 Опис TaskObject.....	30
3.3.3 Опис методів.....	31
3.4 Обчислення вихідних характеристик.....	32
3.5 Верифікація.....	35
4 Експерименти на моделі.....	36

5 Інтерпретація результатів моделювання та експериментів.....	37
Висновки.....	38
Список використаних джерел.....	39
Додатки.....	40

## ПОСТАНОВКА ЗАВДАННЯ

Багатопроцесорна обчислювальна система складається з двох процесорів із загальною оперативною пам'яттю обсягом 131 сторінка, чотирьох накопичувачів на дисках, кожний із яких доступний обом процесорам, і одного каналу передачі даних. Завдання надходять у систему із середньою інтенсивністю, рівною 12 завданням у хвилину відповідно до розподілу Пуассона. Загальний час, необхідний процесору на обробку завдання, розподілено нормально з математичним сподіванням 10 секунд та середнім квадратичним відхиленням 3 секунди. Час обробки процесором включає переривання, необхідні для здійснення обміну по каналу вводу-виводу. Інтервали між перериваннями розподілені за негативний експоненціальний розподілом з математичним сподіванням, що дорівнює оберненій величині середньої інтенсивності операцій вводу-виводу завдання. Середня інтенсивність операцій введення-виведення розподілена рівномірно на інтервалі від 2 секунд до 10 секунд. Операції введення-виведення призначаються конкретному диску. Завданню, що надходить у систему, призначається пріоритет, що є величиною, оберненою до потреби в пам'яті. Потреба завдання в пам'яті розподілена рівномірно в інтервалі від 20 до 60 сторінок. Як тільки пам'ять виділена для завдання, один з вільних процесорів починає його обробку. При видачі запиту на здійснення введення-виведення завдання може продовжувати використання процесора доти, доки в черзі залишиться тільки один запит. Таким чином, якщо зроблений запит на здійснення введення-виведення і один запит вже очікує в черзі, то процесор звільняється, а запит на введення-виведення розміщується в черзі. Після виконання поточного запиту введення-виведення процесор може відновити обробку завдання в тому випадку, якщо вона вільна. Після переривання процесора автоматично виконується запит введення-виведення з призначеним завданню диском. Таким чином, здійснюється прямий доступ до диска з процесора. Передбачається, що час позиціонування диска розподілено рівномірно на інтервалі від 0,0 до 0,075 секунд. Одночасно може здійснюватися

тільки одна операція позиціонування диска. Після позиціонування здійснюється обмін даними по каналу передачі даних. Час обміну дорівнює  $0,001 \times (2,5 + h)$ , де  $h$  - рівномірно розподілена на інтервалі від 0 до 25 величина. Після здійснення обміну запит введення-виведення вважається виконаним. Визначити загальний час виконання завдання в системі, а також статистичні оцінки завантаження усіх чотирьох дисків, каналу передачі даних та обох процесорів. Крім того, необхідно одержати оцінку середнього використання пам'яті, статистику щодо кількості завдань, які очікують виділення ресурсу, та щодо часу очікування.



## ВСТУП

У сучасному світі багатопроцесорні обчислювальні системи відіграють ключову роль у вирішенні складних обчислювальних задач. Ефективність їх роботи залежить від багатьох факторів, включаючи управління пам'яттю, розподіл процесорного часу та організацію введення-виведення. Тому дослідження характеристик таких систем є актуальним завданням для оптимізації їх роботи та підвищення продуктивності.

Метою даної курсової роботи є визначення статистичних характеристик роботи багатопроцесорної обчислювальної системи, зокрема середнього значення, середнього квадратичного відхилення та закону розподілу для таких вихідних значень, як-от: часу виконання завдання в системі; завантаження чотирьох дисків, каналу передачі даних і процесорів; використання пам'яті; кількості завдань, які очікують виділення ресурсу, та часу їхнього очікування.

Для досягнення поставленої мети використовується апарат мереж Петрі, який є потужним математичним інструментом для моделювання та аналізу паралельних та розподілених систем. Мережі Петрі дозволяють ефективно описувати та досліджувати асинхронні, паралельні процеси в системі, враховуючи стохастичну природу процесів, що відбуваються в ній. У роботі застосовуються різні типи ймовірнісних розподілів для моделювання надходження завдань (розподіл Пуассона), часу обробки завдань (нормальний розподіл), інтервалів між перериваннями (експоненційний розподіл) та інших характеристик системи.

Для реалізації моделі використовується спеціалізоване програмне забезпечення PetriObjModelPaint, розроблене для моделювання та аналізу мереж Петрі. Цей інструмент надає можливості для створення, візуалізації та дослідження мережевих моделей, а також дозволяє отримувати статистичні оцінки завантаження всіх компонентів системи, включаючи процесори, диски, канал передачі даних, аналізувати використання пам'яті та характеристики черг завдань.

Результати дослідження дозволять оцінити ефективність роботи системи та виявити потенційні "вузькі місця" в її функціонуванні, що може бути використано для подальшої оптимізації роботи багатопроцесорних обчислювальних систем.

## **1 РОЗРОБКА КОНЦЕПТУАЛЬНОЇ МОДЕЛІ**

У цьому розділі представлено концептуальну модель багатопроцесорної обчислювальної системи, яка включає опис її складових, основних процесів, правил взаємодії елементів та параметрів, необхідних для подальшого моделювання і аналізу.

### **1.1 Опис задачі**

Мета моделювання – визначити показники роботи багатопроцесорної обчислювальної системи, зокрема:

- загальний час виконання завдань;
- завантаження дисків, каналу передачі даних і процесорів;
- середнє використання пам'яті;
- кількість завдань, що очікують на виділення ресурсу;
- час очікування виділення пам'яті.

### **1.2 Структурна схема моделі**

Об'єктом моделювання є багатопроцесорна система, яка включає:

- два процесори, що використовують спільну оперативну пам'ять на 131 сторінку;
- чотири диски, доступні обом процесорам;
- один канал передачі даних, який використовується для обміну між процесорами та дисками.

### **1.3 Опис процесу**

Процес складається з таких елементів:

- завдання надходять у систему з інтенсивністю 12 завдань/хв за розподілом Пуассона;
- кожному завданню призначається пріоритет, обернений до його потреби в пам'яті;

- потреба завдань у пам'яті розподілена рівномірно в межах 20–60 сторінок;
- якщо пам'ять доступна, одразу призначається один із вільних процесорів;
- час виконання завдання розподілений нормально із середнім 10 секунд та відхиленням 3 секунди;
- у процесі виконання завдань виникають переривання для обміну даними; інтервали між перериваннями розподілені експоненційно з параметром, оберненим до інтенсивності операцій введення-виведення (2–10 секунд);
- операції введення-виведення включають;
- позиціонування диска ( $0-0,075$  с);
- передачу даних ( $0,001 \times (2,5+h)$ ), де  $h$  розподілено рівномірно в межах 0–25);
- завдання звільняє процесор, якщо черга операцій введення-виведення складається з максимум одного завдання; в іншому випадку, воно продовжує займати процесор.

#### 1.4 Вхідні змінні

Модель має такі вхідні змінні, як-от:

- інтенсивність надходження завдань (12 завдань/хв);
- потреба завдань у пам'яті (20–60 сторінок);
- час виконання завдання процесором (нормальний розподіл: середнє 10 с, відхилення 3 с);
- інтенсивність операцій введення-виведення (2–10 с);
- час позиціонування диска ( $0-0,075$  с);
- час передачі даних ( $0,001 \times (2,5+h)$ ),  $h$  у межах 0–25).

#### 1.5 Вихідні змінні

Модель має такі вихідні змінні, як-от:

- загальний час виконання завдань;
- завантаження процесорів, дисків, каналу передачі даних;
- середнє використання пам'яті;
- кількість завдань, що очікують виділення пам'яті чи ресурсів;
- час очікування виділення пам'яті.

## 1.6 BPMN-діаграма

BPMN-діаграма (Business Process Model and Notation[1]) – це графічне представлення процесів, яке дозволяє детально описати їхню послідовність, взаємодію елементів системи та потік даних між ними. Вона використовується для візуалізації концептуальної моделі, допомагаючи зрозуміти логіку роботи системи, виявити можливі вузькі місця та полегшити подальшу формалізацію і моделювання.

На рисунку 1.1 представлено BPMN-діаграму моделі, яка відображає ключові етапи обробки завдань і взаємодію різних компонентів системи через обмін повідомленнями. Діаграма ілюструє потік даних і послідовність дій від генерації завдання до його завершення.

Розглянемо блок генерації завдань. Через визначений проміжок часу, вказаний у коментарі, відбувається подія створення нового завдання. Після цього завданню призначається обсяг пам'яті (кількість сторінок), необхідний для його обробки, і воно передається в оперативну пам'ять.

Оперативна пам'ять отримує повідомлення про нове завдання, додає його в чергу й перевіряє наявність вільних сторінок для виконання хоча б одного завдання з черги. Щойно вільна пам'ять стає доступною, вона виділяється, і завдання надсилається до процесорів.

Процесори приймають повідомлення про нове завдання і додають його в чергу. Як тільки один із процесорів стає вільним, він починає обробку завдання. По завершенню обробки перевіряються три умови: наявність вільного диска, невиконаний запит введення-виведення та отримання відповідних повідомлень. Коли ці умови виконано, процесор звільняється, а система надсилає кілька

повідомлень: генератору запитів про виконання запиту, оперативній пам'яті про звільнення сторінок і диску про готовність прийняти завдання.

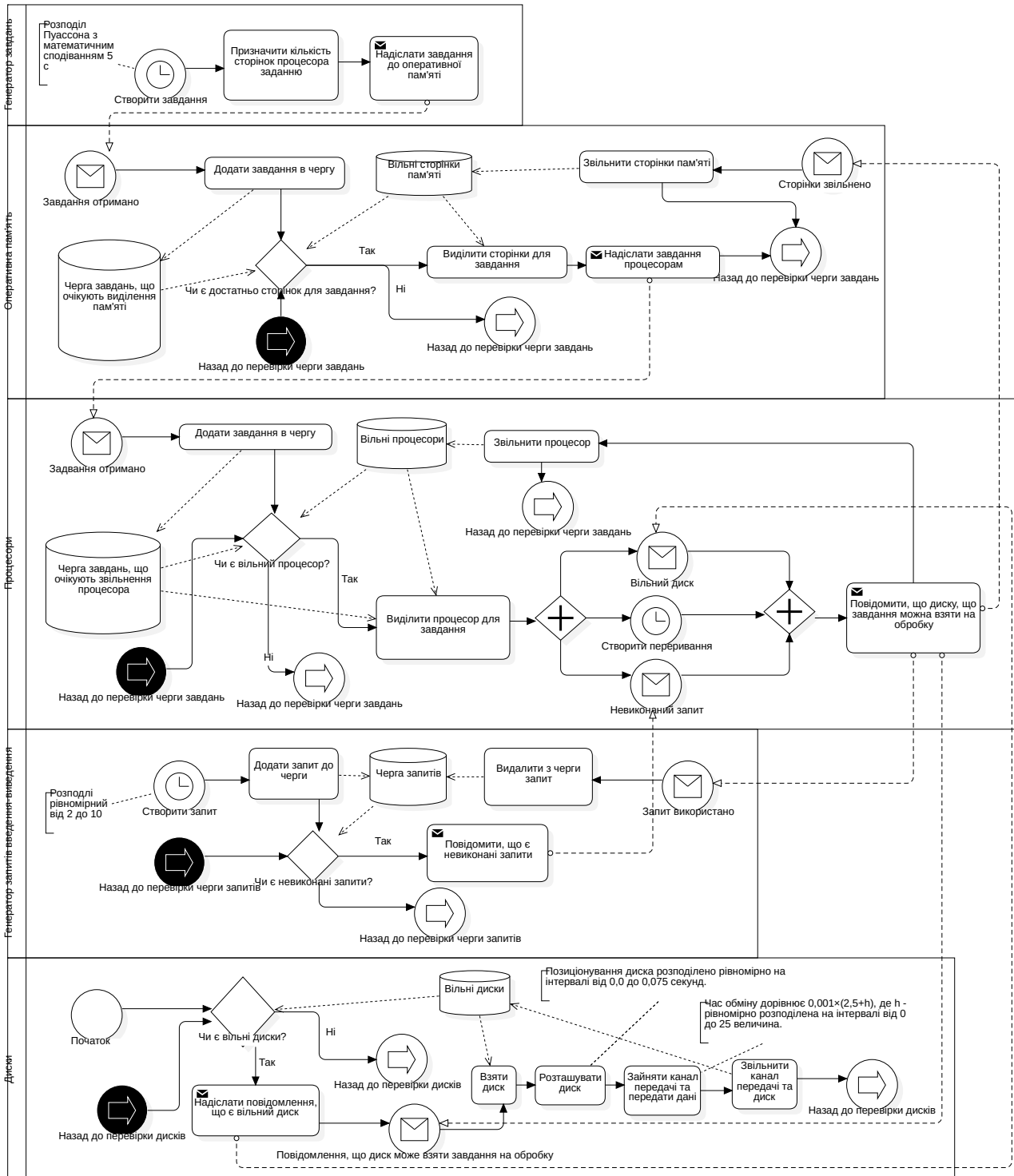


Рисунок 1.1 — BPMN-діаграма

Розглянемо блок генератора запитів. Через заданий час відбувається подія створення нового запиту, який додається в чергу. Далі постійно перевіряється наявність хоча б одного невиконаного запиту в черзі. Якщо такий запит є, повідомлення надсилається процесорам.

Коли диск отримує повідомлення про готовність захопити завдання, він розпочинає позиціонування своєї головки зчитування, після чого виконується передача даних через канал введення-виведення. Завдання вважається завершеним після успішного виконання цих операцій.

### **1.7 Висновки до розділу**

У висновку можна зазначити, що розроблена концептуальна модель багатопроцесорної обчислювальної системи забезпечує базу для формалізації процесів, побудови алгоритму імітації та проведення експериментального дослідження. Визначені вхідні змінні, параметри та вихідні характеристики дозволяють ефективно аналізувати роботу системи та оптимізувати її продуктивність.

## 2 РОЗРОБКА ФОРМАЛІЗОВАНОЇ МОДЕЛІ


У цьому розділі представлено формалізовану модель багатопроцесорної обчислювальної системи, розроблену з використанням мережі Петрі. Модель описує елементи системи, такі як пам'ять, процесори, диски та канал передачі даних, а також події, що визначають їх взаємодію. Вказано числові параметри для кожного компонента, правила спрацьовування переходів, а також описано генерацію випадкових величин для вхідних змінних. У розділі також подано формули для обчислення вихідних характеристик системи, які дозволяють аналізувати її продуктивність, завантаженість та ефективність роботи. Створена модель є основою для подальшого алгоритму імітації та експериментального дослідження.

Мережа Петрі – це математичний апарат для моделювання дискретних систем, який складається з позицій, переходів та дуг. Позиції представляють стани системи, переходи – події або дії, що змінюють ці стани, а дуги визначають зв'язки між позиціями і переходами. Стан мережі визначається кількістю маркерів у позиціях, які називаються токенами [2][с. 67].


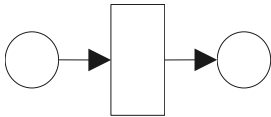



В основі роботи мережі лежить правило спрацьовування переходу: якщо у всіх вхідних позиціях переходу кількість маркерів не менша за кратність відповідних дуг, перехід активується. Під час спрацьовування маркери переміщуються з вхідних позицій до вихідних відповідно до кратності дуг. Мережа Петрі дозволяє описувати процеси з конфліктними переходами, багатоканальні операції та стохастичні події, що забезпечує її високу гнучкість для моделювання складних систем.

У таблиці 2.1 наведено елементи формалізації мережі Петрі [2][с. 69].

Таблиця 2.1 Мережа Петрі

Назва	Позначення	Опис
Перехід		Позначає подію



Назва	Позначення	Опис
Позиція		Позначає умову
Дуга		Позначає зв'язки між подіями та умовами
Маркер(один)		Позначає виконання (або не виконання) умови
Багато фішок		Позначає багатократне виконання умови
Багато дуг		Позначає велику кількість зв'язків

## 2.1 Вхідні параметри

Розглянемо параметри переходів моделі у таблиці 2.2. Нехай маємо відрізок  $[M; N]$ , що позначає діапазон можливої кількості сторінок, що завдання може займати. Нехай  $K \in [M; N]$ , де  $K$  — кількість сторінок, що займає довільне завдання.

Таблиця 2.2 Параметри переходів

Назва переходу	Часова затримка	Значення пріоритету	Значення ймовірності запуску
generate_task	Poisson(5.0)	0	1

Назва переходу	Часова затримка	Значення пріоритету	Значення ймовірності запуску
generate_task_K_pages	0	0	$\frac{1}{M-N}$
fail_allocate_task_K_pages	0	-1	1
try_allocate_task_K_pages	0	0	1
wait_allocate_task_20_pages	0	0	1
process_task_K_pages	Norm(10, 3)	$N-K$	1
generate_io_request	Unif(2, 10)	0	1
create_io_task_20_pages	0	$N-K$	1
take_up_disk_task_K_pages	0	$N-K$	1
generate_interrupt_K_pages	Exp(6)	0	1
drop_interrupt	0	-1	1
place_disk_K_pages	Unif(0, 0.075)	$N-K$	1
io_channel_transfer_task_K_pages	Unif(0.0025, 0.0275)	$N-K$	1

Розглянемо параметри позицій у таблиці 2.3.

Таблиця 2.3 Параметри позицій

Назва позиції	Початкове значення
generator_task	1
generated_task	0
task_K_pages	0

Назва позиції	Початкове значення
fail_allocate_token_task_K_pages	0
allocated_task_K_pages	0
total_wait_allocate_task	0
pages	131
generator_io_request	1
processors	2
generated_request	0
io_task_K_pages	0
generator_interrupt	1
generated_interrupt	0
busy_disk_task_K_pages	0
free_disks	4
disk_placed_task_K_pages	0
is_disk_placement_available	1
finished_tasks_task_K_pages	0
finished_tasks	0

## 2.2 Вихідні параметри

Для початку визначимо загальний час виконання завдання в системі. Для цього зафіксуємо моменти часу, коли відбувається вихід з переходів `generate_task_K_pages` та `finish_task_K_pages`. Тоді ми, маючи різницю цих двох моментів часу, можемо визначити час, що провело певне завдання у системі. Далі можна знайти суму усіх відрізків часу, розділити її на кількість виконаних завдань та отримати середній витрачений завданням час в системі.

$$\Delta T_{\text{average time in system}} = \frac{\sum_{K=M}^N \left[ \sum_{i=0}^{I_{K \text{ pages}}} [T_{K \text{ pages, finish, } i} - T_{K \text{ pages, start, } i}] \right]}{\sum_{K=M}^N I_{K \text{ pages}}},$$

де  $N$  - максимальна кількість сторінок,  $M$  - мінімальна кількість сторінок,  $I$  - кількість завершених з кількістю сторінок  $K$ .

Час очікування на виділення пам'яті розраховується аналогічно:

$$\Delta T_{\text{average wait allocation}} = \frac{\sum_{K=M}^N \left[ \sum_{i=0}^{I_{K \text{ pages}}} [T_{K \text{ pages, wait alloc, } i} - T_{K \text{ pages, fail alloc, } i}] \right]}{\sum_{K=M}^N I_{K \text{ pages}}}.$$

Завантаження є результат ділення часу, що елемент працював, до всього часу моделювання.

$$L_{\text{procs}} = \frac{\sum_{K=M}^N \left[ \sum_{i=0}^{I_{\text{procs task, } K, \Delta T}} \Delta T_{\text{procs task, } K, i} \right]}{\sum_{K=M}^N I_{\text{process task, } K, \Delta T}}, L_{\text{disks}} = \frac{\sum_{K=M}^N \left[ \sum_{i=0}^{I_{\text{place disk, } K, \Delta T}} \Delta T_{\text{place disk, } K, i} \right]}{\sum_{K=M}^N I_{\text{place disk, } K, \Delta T}},$$

$$L_{\text{io channel}} = \frac{\sum_{K=M}^N \left[ \sum_{i=0}^{I_{\text{io channel transfer, } K, \Delta T}} \Delta T_{\text{io channel transfer, } K, i} \right]}{\sum_{K=M}^N I_{\text{io channel transfer, } K, \Delta T}},$$

де  $I_{\text{io channel transfer}}$ ,  $I_{\text{place disk}}$ ,  $I_{\text{procs task}}$  - це кількість виходів з відповідних переходів.

## 2.3 Діаграма

На рисунку 2.1 можна розглянути діаграму мережі Петрі. Для зручності вона побудована лише з двома типами завдань. Це робить її менш нагромадженою, оскільки частини схеми для кожного завдання є однаковими.

Основними компонентами мережі є генератор завдань, оперативна пам'ять, процесори, генератор запитів введення-виведення, диски, канал передачі даних та блок завершення роботи.



відповідають їхньому обсягу пам'яті, наприклад, `task_20_pages` для завдань із потребою у 20 сторінок.

Оперативна пам'ять представлена позицією `pages`, яка відображає загальну кількість доступних сторінок (131 сторінка). Переходи `try_allocate` і `wait_allocate` перевіряють наявність вільної пам'яті та виділяють її для завдань відповідного обсягу. У разі успішного виділення токени передаються до позиції `allocated`, а у разі невдачі – у `fail_allocate_token`.

Процесори моделюються позицією `processors`, яка визначає кількість доступних процесорів (2 процесори). Переходи `process` відповідають за обробку завдань із використанням нормального розподілу часу.

Генератор запитів введення-виведення моделюється позицією `generated_io_request`, яка активує перехід `create_io`. Цей перехід генерує запити введення-виведення для завдань, які були оброблені процесорами.

Диски представлені позицією `free_disks`, яка відображає кількість доступних дисків (4 диски). Перехід `place_disk` імітує позиціонування головки зчитування з використанням рівномірного розподілу часу. Після виконання цієї операції токени передаються через канал введення-виведення.

Канал введення-виведення моделюється позицією `is_disk_placement_available`, яка визначає готовність каналу до передачі даних. Перехід `io_channel_transfer` відповідає за передачу даних із використанням рівномірного розподілу часу. Завершення роботи моделюється позицією `finished_tasks`, яка накопичує виконані завдання. Це дозволяє аналізувати продуктивність системи та обчислювати необхідні показники.

Логіка роботи мережі ґрунтується на потоках tokenів між позиціями через переходи. Завдання генеруються з певним пріоритетом і обсягом пам'яті, який визначається рівномірним розподілом (від 20 до 60 сторінок). Якщо завдання не отримало пам'ять, воно повертається до черги. Виділені завдання передаються до процесорів, які здійснюють їх обробку із врахуванням стохастичних затримок. Після обробки завдання надсилають запити на диски. Якщо диск доступний, він виконує операцію зчитування та передачу даних через канал

введення-виведення. Завдання вважається завершеним після успішної передачі даних і повернення ресурсів системі, включаючи пам'ять, процесори та диски.

## **2.4 Висновки до розділу**

У цьому розділі було створено формалізовану модель багатопроцесорної обчислювальної системи, яка дозволяє математично описати її компоненти та взаємодії. Модель побудована з використанням мережі Петрі, що забезпечує можливість детального моделювання процесів генерації, обробки завдань, запитів введення-виведення та управління ресурсами. Було визначено всі необхідні вхідні параметри, їх числові характеристики, а також формули для обчислення вихідних характеристик системи. Створена модель є основою для розробки алгоритму імітації та проведення експериментального дослідження ефективності роботи системи.

### **3 АЛГОРИТМІЗАЦІЯ МОДЕЛІ ТА ЇЇ РЕАЛІЗАЦІЯ**

Цей розділ присвячений алгоритмізації та програмній реалізації імітаційної моделі багатопроцесорної обчислювальної системи, створеної за допомогою мережі Петрі. Метою є опис роботи стандартного алгоритму моделювання, наданого бібліотекою PetriObjModelPaint [3], а також внесених модифікацій, які забезпечують відповідність алгоритму вимогам поставленої задачі. Розділ містить опис основних етапів алгоритму, реалізацію додаткового функціоналу, оптимізацію коду під конкретну задачу, а також протокол подій, що фіксуються під час моделювання. Наприкінці представлено результати верифікації моделі, що підтверджують коректність її реалізації.

#### **3.1 Опис алгоритму імітації мережі Петрі**

Алгоритм імітації мережі Петрі, реалізований у програмному застосунку PetriObjModelPaint, виконує симуляцію роботи системи, забезпечуючи просування часу, обробку подій, розв'язання конфліктів між переходами та збір статистики. Основні етапи алгоритму можна розділити на кілька частин: ініціалізація моделі, основний цикл симуляції та завершення роботи. Усі ці кроки враховують як правила функціонування мережі Петрі, так і специфічні особливості моделі, що досліджується.

На етапі ініціалізації модель готується до роботи: задається час моделювання, об'єкти сортуються за пріоритетом, а маркери обробляються методами `input()`. Після цього виводяться початкові маркування позицій, якщо протокол увімкнено.

Основний цикл симуляції є ключовим компонентом алгоритму. На кожному кроці визначається найближчий час події (мінімальний час серед усіх переходів), оновлюється поточний час симуляції, збирається статистика, а також розв'язуються конфлікти між переходами, що можуть активуватись одночасно. Розв'язання конфліктів базується на пріоритетах переходів; якщо пріоритети збігаються, вибір здійснюється випадковим чином.



Після розв'язання конфлікту виконується подія вибраного переходу за допомогою методу `doT()`. Далі оновлюється маркування позицій, і викликається метод `input()` для обробки нових маркерів у переходах. Якщо увімкнено протокол, відображаються результати цих дій.

На етапі завершення симуляції модель повертається до початкового порядку об'єктів, виводяться зібрані статистичні дані, а результати симуляції підсумовуються. Наведений нижче псевдокод описує основні етапи алгоритму:

---

```

1 # Initialization of the model
2 set simulation_time = timeModeling # Set the total simulation time
3 set current_time = 0.0 # Start simulation at time 0
4 sort list_of_objects by priority # Order objects by priority for initial processing
5 # Initialize Petri objects
6 for obj in list_of_objects:
7     obj.input() # Process initial markings
8 # Print initial markings if protocol is enabled
9 if protocol_enabled:
10     for obj in list_of_objects:
11         obj.print_markings() # Output the current marking of each object
12 # Main simulation loop
13 while current_time < simulation_time:
14     # Clear the list of conflicting objects for this step
15     clear conflict_list
16     min_time = list_of_objects[0].get_time_min() # Find the minimum event time
17     # Find the nearest event time across all objects
18     for obj in list_of_objects:
19         if obj.get_time_min() < min_time:
20             min_time = obj.get_time_min()
21     # Update statistics for each object if enabled
22     if statistics_enabled:
23         for obj in list_of_objects:
24             # Calculate the time delta and update statistics
25             delta_time = min(min_time - current_time, simulation_time - current_time)
26             obj.update_statistics(delta_time / min_time)

```

---

---

```

27  # Update the current simulation time
28  current_time = min_time # Move simulation time to the next event
29  # Print time progress if protocol is enabled
30  if protocol_enabled:
31      print(f"Time progress: current_time = {current_time}")
32  # Collect conflicting objects
33  for obj in list_of_objects:
34      if obj.get_time_min() == current_time:
35          conflict_list.append(obj) # Add to the conflict list
36  # Resolve conflicts by selecting an object to process
37  if len(conflict_list) > 1:
38      # Sort conflicting objects by priority and resolve ties randomly
39      conflict_list.sort(by_priority)
40      max_priority_index = find_highest_priority_index(conflict_list)
41      selected_obj = conflict_list[random_choice(max_priority_index)]
42  else:
43      selected_obj = conflict_list[0] # No conflict, process the only object
44  # Print the selected object for this event if protocol is enabled
45  if protocol_enabled:
46      print(f"Selected object: {selected_obj.get_name()}")
47  # Process the selected object's event
48  selected_obj.process_event()
49  selected_obj.output_markings() # Update markings after event
50  # Shuffle the list of objects to ensure fair processing of shared resources
51  shuffle(list_of_objects)
52  sort list_of_objects by priority # Re-sort the objects by priority
53  # Update input markings for all objects
54  for obj in list_of_objects:
55      obj.input() # Process new input markings
56  # Print current markings if protocol is enabled
57  if protocol_enabled:
58      for obj in list_of_objects:
59          obj.print_markings() # Output updated markings for all objects
60  # Finalize simulation

```

---

---

```

61 sort list_of_objects by initial_order # Return to the initial object order
62 if statistics_enabled:
63     for obj in list_of_objects:
64         obj.print_statistics() # Output collected statistics for each object

```

---

## 3.2 Модифікації алгоритму

У процесі реалізації моделі були внесені зміни до базового алгоритму, щоб адаптувати його до специфіки поставленої задачі. Основні модифікації стосувалися покращення роботи мережі Петрі та збору додаткової інформації для аналізу. Зміни можна поділити на два напрямки: функціональні вдосконалення алгоритму та зручність роботи з мережами.

### 3.2.1 Додавання розподілу Пуассона для генерації подій

Для точного моделювання вхідних потоків завдань у систему було реалізовано підтримку розподілу Пуассона. Це дозволило описати інтенсивність надходження завдань у вигляді стохастичного процесу, що відповідає заданій середній інтенсивності. Реалізація виконана шляхом додавання відповідної функції до генератора подій у мережі.

---

```

1 public static double poisson(final double timeMean) {
2     PoissonDistribution poisson = new PoissonDistribution(timeMean);
3     return poisson.sample();
4 }

```

---

### 3.2.2 Вдосконалення роботи з NetLibrary

Для зручності роботи з мережами Петрі було додано клас NetLibraryManager, який дозволяє динамічно додавати методи в бібліотеку NetLibrary. Завдяки цьому можна легко інтегрувати нові мережі в графічний інтерфейс, використовувати рефлексію для виклику методів і забезпечувати їх коректність під час компіляції. Також додано перевірку методів генерації мережі за допомогою анотацій NetLibraryMethod та NetLibraryMethodProcessor.

---

```

1 @NetLibraryMethod

```

---

---

```

2 public static PetriNet createNet() throws ExceptionInvalidTimeDelay {
3     final CourseWorkNet net = new CourseWorkNet();
4     return net.net;
5 }

```

---

### 3.2.3 Покращення графічного інтерфейсу

Був виправлений метод `generateGraphNetBySimpleNet`, що забезпечує коректне графічне відображення мережі Петрі. Помилка полягала у тому, що оскільки не було перевірки на включення переходів та позицій в контейнери `inTrans` та `inPlaces`, то дублікати додавалися у графічне відображення мережі.

---

```

1 if (
2     !inTrans.contains(tran) // Нова перевірка
3     && tran.getNumber() == outArc.getNumT()
4 ) {
5     inTrans.add(tran);
6 }
7 // ...
8 if (
9     !inPlaces.contains(place) // Нова перевірка
10    && place.getNumber() == inArc.getNumP()
11 ) {
12    inPlaces.add(place);
13 }

```

---

## 3.3 Клас `CourseWorkNet`. Протокол подій

Клас `CourseWorkNet` розроблений для аналізу та моделювання роботи багатопроцесорної обчислювальної системи за допомогою мережі Петрі. Він є обгорткою для створення та налаштування компонентів мережі, забезпечуючи зручний інтерфейс для опису та управління моделлю.

### 3.3.1 Опис атрибутів

Позиція `generated_task` зберігає кількість згенерованих завдань у системі. Ця позиція використовується для активізації переходів, що створюють нові

завдання. Її основне призначення полягає у фіксації вхідного потоку завдань до системи.

Позиція `generated_io_request` відповідає за генерацію запитів введення-виведення для завдань після їх обробки. Її призначення полягає в забезпеченні інтеграції завдань із підсистемою дисків.

Позиція `generated_interrupt` використовується для моделювання переривань, які запускають запити до дисків. Її призначення - імітувати реальні події введення-виведення у багатозадачній системі.

Позиція `processors` відображає кількість вільних процесорів у системі, яка початково становить два. Вона використовується для визначення доступності процесорів для обробки завдань.

Позиція `pages` відображає кількість доступних сторінок пам'яті в системі, початково маючи 131 сторінку. Її призначення полягає у визначенні можливості виділення пам'яті для завдань.

Позиція `free_disks` відображає кількість доступних дисків, яких початково чотири. Вона забезпечує розподіл завдань між доступними дисками.

Позиція `total_wait_allocate_task` підраховує загальний час очікування завдань на виділення пам'яті. Вона використовується для аналізу затримок у системі.

Позиція `finished_tasks` зберігає кількість завершених завдань. Її призначення полягає у відображенні загальної продуктивності системи.

Позиція `is_disk_placement_available` вказує на доступність каналу введення-виведення для передачі даних. Вона контролює одночасний доступ до каналу.

Перехід `generate` генерує нові завдання у системі із заданими параметрами, такими як обсяг пам'яті. Його призначення полягає у забезпеченні створення потоку завдань.

Перехід `try_allocate` перевіряє наявність вільної пам'яті для завдань. У разі успіху пам'ять виділяється. Його призначення - запускати обробку завдань, які отримали доступ до пам'яті.

Перехід `fail_allocate` відображає спробу виділення пам'яті, яка завершилась невдачею. Його призначення полягає у поверненні завдання у чергу для подальших спроб.

Перехід `wait_allocate` переводить завдання з черги у стан обробки після виділення пам'яті. Цей перехід забезпечує повторне надання пам'яті для завдань.

Перехід `process` імітує обробку завдань процесорами із використанням нормального розподілу часу. Це основний перехід для виконання завдань.

Перехід `create_io` створює запити введення-виведення для завдань, які були оброблені. Його призначення полягає в інтеграції із підсистемою дисків.

Перехід `take_up_disks` захоплює доступний диск для обробки запиту. Він контролює завантаження дисків.

Перехід `place_disk` імітує позиціонування головки диска перед операцією. Його призначення - затримувати виконання запиту для моделювання реального часу доступу.

Перехід `io_channel_transfer` імітує передачу даних через канал введення-виведення. Цей перехід завершує обробку запитів введення-виведення.

### 3.3.2 Опис TaskObject

Об'єкт `TaskObject` представляє завдання з конкретним обсягом пам'яті (від 20 до 50 сторінок) та визначеним життєвим циклом у системі. Цей об'єкт містить низку важливих полів, що керують його функціонуванням у системі.

Поле `generate` є генератором завдання цього типу. Поле `task` являє собою позицію, що зберігає завдання. `Try_allocate` є переходом для перевірки доступності пам'яті, тоді як поле `allocated` - це позиція, яка фіксує завдання, що отримали пам'ять.

`Fail_allocate` є переходом, що фіксує невдалі спроби виділення пам'яті. Позиція `fail_allocate_token` призначена для невдалих завдань, які очікують повторного виділення пам'яті. `Wait_allocate` представляє собою перехід для повторного виділення пам'яті.

Process є переходом, що імітує обробку завдання, а create\_io - це перехід, який створює запити введення-виведення. Take\_up\_disks виступає переходом для захоплення дисків завданням.

Поле busy\_disk є позицією, яка показує, що диск зайнятий завданням. Place\_disk представляє перехід, що імітує позиціонування диска, а disk\_placed - це позиція, яка фіксує успішне позиціонування диска.

Io\_channel\_transfer є переходом, що забезпечує передачу даних, а finish - це позиція, яка зберігає завершені завдання.

Об'єкти TaskObject використовуються для відстеження стану кожного типу завдань окремо, включаючи їх генерацію, обробку, введення-виведення та завершення. Завдяки цьому можна проводити детальний аналіз роботи системи та оптимізувати її параметри.

### **3.3.3 Опис методів**

Метод create\_task\_generator створює генератор завдань, який додає нові завдання до системи відповідно до розподілу Пуассона. Він приймає параметри d\_P (список позицій), d\_T (список переходів), d\_In (список вхідних дуг) та d\_Out (список вихідних дуг). В результаті повертає позицію generated\_task, яка накопичує створені завдання.

Метод create\_processors створює позицію processors, яка відображає кількість доступних процесорів. Він приймає єдиний параметр d\_P (список позицій) та повертає позицію processors.

Метод create\_pages створює позицію pages, що відображає кількість доступних сторінок пам'яті. Приймає параметр d\_P (список позицій) та повертає позицію pages.

Метод create\_free\_disks створює позицію free\_disks, яка відповідає кількості доступних дисків. Приймає параметр d\_P (список позицій) та повертає позицію free\_disks.

Метод create\_io\_request\_generator створює генератор запитів введення-виведення з рівномірним розподілом часу. Він приймає параметри d\_P (список

позицій), d\_T (список переходів), d\_In (список вхідних дуг) та d\_Out (список вихідних дуг). В результаті повертає позицію generated\_io\_request.

Метод create\_interrupt\_generator створює генератор переривань із експоненційним розподілом часу. Він приймає параметри d\_P (список позицій), d\_T (список переходів), d\_In (список вхідних дуг) та d\_Out (список вихідних дуг). В результаті повертає позицію generated\_interrupt.

Конструктор CourseWorkNet є основним конструктором класу, який створює та ініціалізує всі компоненти мережі Петрі. Він створює генератори завдань, запитів введення-виведення та переривань, позиції для обробки завдань, пам'яті, процесорів, дисків, а також переходи для роботи з ресурсами. Додаткова логіка включає генерацію завдань з різними параметрами (обсяг пам'яті від 20 до 60 сторінок), розрахунок ймовірностей для кожного типу завдання та додавання дуг між позиціями та переходами.

Метод generate\_task\_objects створює об'єкти завдань TaskObject для кожного обсягу пам'яті між pages\_start та pages\_end із заданою ймовірністю. Він приймає параметри pages\_start (мінімальний обсяг пам'яті завдання), pages\_end (максимальний обсяг пам'яті завдання) та probability (ймовірність створення завдання кожного типу). В результаті додає об'єкти TaskObject до списку taskObjects.

### 3.4 Обчислення вихідних характеристик

Код у методі main реалізує симуляцію роботи моделі мережі Петрі та обчислює ключові характеристики продуктивності системи. Нижче наведено пояснення кожного блоку обчислень.

Перший етап - ініціалізація моделі. На цьому етапі створюється об'єкт класу CourseWorkNet, який ініціалізує всю мережу Петрі. Також створюється об'єкт PetriObjModel, який об'єднує всі PetriSim у єдину модель для симуляції. Це налаштовує модель для подальшого запуску симуляції.

---

```
1 final ArrayList<PetriSim> list = new ArrayList<>();
```

```
2 final NetLibrary.CourseWorkNet courseWorkNet = new NetLibrary.CourseWorkNet();
```

---



---

```
3 list.add(new PetriSim(courseWorkNet.net));
4 final PetriObjModel model = new PetriObjModel(list);
```

---

Другий етап - запуск симуляції. Тут симуляція триває протягом timeModeling секунд. Протокол подій вимикається через setIsProtokol(false), щоб зменшити обсяг виводу під час симуляції. Цей етап безпосередньо запускає імітацію моделі.

---

```
1 model.setIsProtokol(false);
2 double timeModeling = 10000;
3 model.go(timeModeling);
```

---

Третій етап включає ініціалізацію змінних для збору статистики. Створюються змінні для збереження підсумкових характеристик, таких як загальний час у системі, кількість завершених завдань, завантаження дисків, каналів, процесорів та середній час очікування на виділення пам'яті.

---

```
1 double totalTimeInSystem = 0;
2 int totalFinishedTasks = 0;
3 double totalPlaceDiskWorkTime = 0;
4 double totalIoChannelWorkTime = 0;
5 double totalProcessorsWorkTime = 0;
6 double totalWaitAllocateTime = 0;
7 int totalWaitAllocatedTasks = 0;
```

---

На четвертому етапі відбувається обчислення ключових характеристик. Час у системі розраховується як різниця між часом виходу завдання з переходу io\_channel\_transfer та часом генерації завдання. Час очікування на виділення пам'яті визначається як різниця між часом невдалої спроби виділення пам'яті та успішним виділенням. Робочий час дисків - це загальний час роботи переходу place\_disk. Робочий час каналу введення-виведення обчислюється як загальний час роботи переходу io\_channel\_transfer. Робочий час процесорів - це загальний час роботи переходу process. Загальна кількість завершених завдань розраховується як кількість завершених транзакцій у переході io\_channel\_transfer.

---

```

1 for(final NetLibrary.CourseWorkNet.TaskObject taskObject : courseWorkNet.taskObjects) {
2     for(int i = 0; i < taskObject.io_channel_transfer.getOutMoments().size(); i++) {
3         final double timeInSystem = taskObject.io_channel_transfer.getOutMoments().get(i)
4             - taskObject.generate.getOutMoments().get(i);
5         totalTimeInSystem += timeInSystem;
6     }
7     for(int i = 0; i < taskObject.wait_allocate.getOutMoments().size(); i++) {
8         final double waitTime = taskObject.wait_allocate.getOutMoments().get(i)
9             - taskObject.fail_allocate.getOutMoments().get(i);
10        totalWaitAllocateTime += waitTime;
11    }
12    totalWaitAllocatedTasks += taskObject.wait_allocate.getOutMoments().size();
13    totalPlaceDiskWorkTime += taskObject.place_disk.getTotalTimeServ();
14    totalFinishedTasks += taskObject.io_channel_transfer.getOutMoments().size();
15    totalIoChannelWorkTime += taskObject.io_channel_transfer.getTotalTimeServ();
16    totalProcessorsWorkTime += taskObject.process.getTotalTimeServ();
17 }

```

---

П'ятий етап присвячений обчисленню середніх характеристик. Середній час у системі - це середній час, який завдання проводить у системі від генерації до завершення. Завантаження дисків розраховується як відношення загального часу роботи дисків до часу моделювання. Завантаження каналу введення-виведення - це відношення загального часу роботи каналу до часу моделювання. Завантаження процесорів визначається як відношення загального часу роботи процесорів до часу моделювання. Середнє використання пам'яті обчислюється як різниця між загальною кількістю сторінок та середнім значенням кількості вільних сторінок. Середній час очікування виділення пам'яті розраховується як загальний час очікування, поділений на кількість завдань, що чекали на виділення пам'яті.

---

```

1 final double averageTimeInSystem = totalTimeInSystem / (double)totalFinishedTasks;
2 System.out.println("Average time in system: ".concat(Double.toString(averageTimeInSystem)));
3 System.out.println("Disk load: ".concat(Double.toString(totalPlaceDiskWorkTime /
timeModeling)));

```

---

---

```
4 System.out.println("Io channel load: ".concat(Double.toString(totalIoChannelWorkTime /
timeModeling)));
5 System.out.println("Processors load: ".concat(Double.toString(totalProcessorsWorkTime /
timeModeling)));
6 System.out.println("Average use of pages:
".concat(Double.toString(NetLibrary.CourseWorkNet.TOTAL_PAGES -
courseWorkNet.pages.getMean())));
7 System.out.println("Total wait allocate task:
".concat(Double.toString(courseWorkNet.total_wait_allocate_task.getMean())));
8 final double averageWaitAllocate = totalWaitAllocateTime / (double)totalWaitAllocatedTasks;
9 System.out.println("Average wait allocate time:
".concat(Double.toString(averageWaitAllocate)));
```

---

### 3.5 Верифікація

## **4      ЕКСПЕРИМЕНТИ НА МОДЕЛИ**

## **5      ІНТЕРПРЕТАЦІЯ РЕЗУЛЬТАТІВ МОДЕЛЮВАННЯ ТА ЕКСПЕРИМЕНТІВ**

## **ВИСНОВКИ**

## **СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ**

- 1., Business Process Model and Notation,
2. І. В. Стеценко, Моделювання систем, 2011
- 3: , PetriObjModelPaint,

## ДОДАТКИ