

МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

КАФЕДРА ІНФОРМАТИКИ ТА ПРОГРАМНОЇ ІНЖЕНЕРІЇ

Курсова робота з дисципліни «Моделювання систем»

Тема: Задача про створення програмного керування верстатами

|  |  |
| --- | --- |
| **Керівник**:  ас. Бернатович А.О.  «Допущено до захисту»  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (особистий підпис керівника)  «\_\_\_» \_\_\_\_\_\_\_\_\_\_ 20\_\_ р.  Захищено з оцінкою  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (оцінка)  Члени комісії:  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (особистий підпис)  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (особистий підпис) | **Виконавець**:  Бондаренко Д.С.  студент ІV курсу  групи ІТ-92  залікова книжка  № ІТ-9204  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (особистий підпис виконавця)  «13» листопада 2022 р.  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (розшифровка підпису)  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (розшифровка підпису) |

**Київ 2022**

Національний технічний університет України «КПІ ім. І. Сікорського»

Кафедра інформатики та програмної інженерії

Дисципліна «Моделювання систем»

Спеціальність Інженерія програмного забезпечення

Курс 4 Група ІТ – 92 Семестр 7

ЗАВДАННЯ

на курсову роботу студента

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Бондаренко Дмитро Сергійович\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(прізвище, ім'я, по батькові)

1.Тема роботи: Задача про створення програмного керування верстатами

2. Термін здачі студентом закінченої роботи "\_\_" \_\_\_\_\_\_ 2022 р.

3. Вихідні дані до проекту

Варіант № 18

З файлу варіантів завдань у методичних вказівках на курсову роботу, у таблиці 3.20 обирається третій варіант вихідних даних

4. Зміст розрахунково-пояснювальної записки (перелік питань, що розробляються)

*1. Аналіз існуючих методів вирішення завдання 2. Розробка концептуальної моделі 3. Вибір засобів моделювання 4. Розробка структурної схеми імітаційної моделі та опис її функціонування 4.1 Опис імітаційної моделі 4.2 Опис програмної реалізації імітаційної моделі 4.3 Оцінка адекватності моделі 5. Результати експериментів на моделі 5.1. План експериментів 5.2. Аналіз і оцінка результатів. Висновки.*

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

Графічного матеріалу не має.

6. Дата видачі завдання "4" жовтня 2022 р.

КАЛЕНДАРНИЙ ПЛАН

|  |  |  |  |
| --- | --- | --- | --- |
| № | Назва етапів курсового проекту (роботи) | Термін  виконання  етапів роботи | Примітка |
| 1 | Отримання завдання | 04.10.2022 |  |
| 2 | Аналіз можливих методів вирішення поставленого завдання | 13.11.2022 |  |
| 3 | Розробка концептуальної моделі | 14.11.2022 |  |
| 4 | Перший контроль за процесом виконання курсового проекту (роботи), консультація у викладача |  |  |
| 5 | Опис імітаційної моделі | 20.11.2022 |  |
| 6 | Опис програмної реалізації імітаційної моделі | 26.11.2022 |  |
| 7 | Другий контроль за процесом виконання курсового проекту (роботи), консультація у викладача |  |  |
| 8 | Аналіз та оцінка результатів | 29.11.2022 |  |
| 9 | Оформлення пояснювальної записки | 02.12.2022 |  |
| 10 | Здача пояснювальної записки |  |  |
| 11 | Захист курсового проекту (роботи) |  |  |

Студент\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ *Бондаренко Д.С.*

(підпис) (прізвище, ім'я, по батькові)

Керівник \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ *Бернатович А.О.*

(підпис) (прізвище, ім'я, по батькові)

«4» жовтня 2022 р.

РЕФЕРАТ

Курсова робота: 81 с., 32 рисунки, 9 табл., 6 додатки, 5 джерел літератури.

*Об'єкт дослідження* – система розробки програмного забезпечення для керування металорізальними верстатами з числовим програмним керуванням.

*Мета роботи* – обґрунтування типу вибору замовлення з черг обслуговування.

*Метод дослідження* – імітаційне моделювання системи створення програмного забезпечення для керування металорізальних верстатів з числовим програмним керуванням.

Проведено дослідження та тестування із різними типами вибору завдань з черги для обслуговування, розроблене програмне забезпечення для імітаційного моделювання створення та встановлення програмного забезпечення для керування на верстати. Результати моделювання використані для прийняття рішень щодо оптимального типу вибору замовлень з черг для обслуговування.

Модель програми розроблена вперше.

ІМІТАЦІЙНА МОДЕЛЬ, МЕРЕЖА МАСОВОГО ОБСЛУГОВУВАННЯ, ПРОЦЕС, МОВА МОДЕЛЮВАННЯ, ЕОМ, НОСІЙ, ВЕРСТАТ

**ЗМІСТ**

[ПОСТАНОВКА ЗАВДАННЯ 6](#_Toc120901875)

[ВСТУП 8](#_Toc120901876)

[РОЗДІЛ 1. РОЗРОБКА КОНЦЕПТУАЛЬНОЇ МОДЕЛІ 11](#_Toc120901877)

[РОЗДІЛ 2. РОЗРОБКА ФОРМАЛІЗОВАНОЇ МОДЕЛІ 16](#_Toc120901878)

[2.1 Опис імітаційної моделі 17](#_Toc120901879)

[РОЗДІЛ 3. ВИБІР ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ РЕАЛІЗАЦІЇ ЗАДАЧІ 18](#_Toc120901880)

[РОЗДІЛ 4. РЕАЛІЗАЦІЯ МОДЕЛІ ІЗ ВИКОРИСТАННЯМ УНІВЕРСАЛЬНОЇ МОВИ ПРОГРАМУВАННЯ 19](#_Toc120901881)

[4.1 Верифікація алгоритму імітації 31](#_Toc120901882)

[РОЗДІЛ 5. ПРОВЕДЕННЯ ЕКСПЕРИМЕНТІВ НА МОДЕЛІ 41](#_Toc120901883)

[5.1 Тактичне планування факторного експерименту 41](#_Toc120901884)

[5.2 Стратегічне планування факторного експерименту 46](#_Toc120901885)

[5.3 Експериментування 47](#_Toc120901886)

[ВИСНОВКИ 52](#_Toc120901887)

[СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 53](#_Toc120901888)

[ДОДАТКИ 54](#_Toc120901889)

[Додаток А. Лістинг коду файлу «create.py» 54](#_Toc120901890)

[Додаток Б. Лістинг коду файлу «element.py» 56](#_Toc120901891)

[Додаток В. Лістинг коду файлу «fun\_rand.py» 59](#_Toc120901892)

[Додаток Г. Лістинг коду файлу «process.py» 60](#_Toc120901893)

[Додаток Д. Лістинг коду файлу «model.py» 65](#_Toc120901894)

[Додаток Д. Лістинг коду файлу «main.py» 70](#_Toc120901895)

# **ПОСТАНОВКА ЗАВДАННЯ**

Відділ з обслуговування ЕОМ готує носіїв з програмами для металорізальних верстатів із числовим програмним керуванням. Креслення деталей поступають з конструкторсько-технологічного відділу. Програміст вивчає креслення і пише програму керування верстатом при обробці заготовки деталі. Програмування займає інтервал часу, розподілений за експоненціальним законом із середнім часом 120 хвилин. Програма налагоджується на ЕОМ і записується на носій (тривалість операції – експоненціально розподілена величина із середнім часом 110 хвилин). Потім носій з програмою ставиться на відповідний верстат для випробувань та редагування. Цей процес займає проміжок часу, розподілений експоненціально із середнім часом 90 хвилин.

Замовлення на підготовку носіїв з програмами поступають через проміжки часу, розподілені рівномірно в інтервалі 150 ± 50 хвилин. У момент надходження замовлення для нього визначають директивний термін - термін, до якого повинно бути виконане замовлення. Директивний термін визначається сумою часу надходження замовлення і технологічного часу виконання роботи. Технологічний час виконання робіт - це загальний час обробки (Т1+Т2+Т3) плюс додатковий час, рівномірно розподілений в інтервалі 40 ± 20 хвилин.

Керівництво відділу перевіряє декілька способів черговості обробки замовлень з метою визначення найкращого з них. Запропоновано чотири можливих порядки виконання чекаючих у кожній із черг робіт:

1. спочатку виконуються ті замовлення, на виконання яких затрачується найменше загального часу обробки;
2. спочатку виконуються ті замовлення, на виконання яких затрачується найбільше загального часу обробки;
3. спочатку виконуються ті замовлення, на виконання яких затрачується найменший час обробки, що залишився;
4. спочатку виконуються ті замовлення, які мають найближчий директивний термін.

***Запропонуйт*е** декілька критеріїв оцінки і ***оцініть*** запропоновані дисципліни вибору із черги. Час моделювання необхідно вибирати так, щоб модель працювала у перехідному режимі.

# **ВСТУП**

Задача вибору пріоритетної задачі для виконання тієї чи іншої дії являє собою актуальну дилему на виробництвах, де є черги перед безпосереднім виконанням завданням. Адже від цього вибору залежить швидкість та ефективність виробництва, або, як у нашому випадку, кількість успішно запрограмованих верстатів. У діапазоні задач моделювання, поставлене завдання класифікується задачею управління.

За допомогою розробленої у цій роботі програмі імітаційного моделювання ми можемо дати відповідь на поставлені питання, а саме який варіант черговості обробки замовлень ми маємо обрати, щоб оптимізувати виробництво. У нас з’являється можливість дослідити різні підходи у вирішенні цієї задачі та обрати оптимальний спосіб обслуговування відносно бажаних критеріїв оцінювання, які ми також запропонуємо у цій роботі.

Вирішення поставленого завдання, з наведеної умови задачі, полягає у тому, щоб дослідити, як тип вибору завдання із черги буде впливати на визначені критерії об’єкту. З вихідних даних ми розуміємо, що це буде інтенсивність надходження завдань, послідовність виконання завдання та часові затримки обробки завдань. Це означає, що поставлену задачу можна вирішити побудувавши алгоритм за наступною схемою [1] [c. 10]:



*Рисунок 1* – загальна структура моделі

Де X – множина вхідних змінних системи, Y – множина вихідних змінних системи, P – множина параметрів, F – алгоритм функціонування. Так, із поставленої задачі зрозуміло, що параметри моделі із схеми мають зв’язок із реальними процесами та мають фізичний варіант. Тому, систему створення програмам для металорізальних верстатів із числовим програмним керуванням можна класифікувати, як фізичну модель.

Ми визначились, що для вирішення поставленої задачі нам потрібно моделювати систему. Для цього існують декілька методів. Найпопулярнішими із великої кількості методів моделювання виділяють аналітичне моделювання, математичне моделювання та імітаційне моделювання.

*Моделювання аналітичне* – представлення залежності F вихідних змінних Y від вхідних її змінних X має аналітичний вигляд, тобто представлений у вигляді відомих аналітичних функцій [1] [c. 13]. В поставленому завданні ми не маємо можливості описати залежність вихідних змінних від вхідних за допомогою аналітичних функцій, тому цей метод моделювання нам не підходить.

*Моделювання математичне* – відомий алгоритм відшукання точного розв’язку задачі, але сам розв’язок не може бути записаний в аналітичній формі [1] [c. 13]. Модель поставленої задачі є стохастичною, а тому неможливо гарантувати точний розв’язок. Цей метод нам також не підходить.

*Моделювання імітаційне* – за допомогою імітації функціонування системи в часі багатократними прогонами імітаційної моделі дослідник отримує інформацію про властивості реальної системи. Імітація відбувається за допомогою комп’ютерної програми з алгоритмом відтворення [1] [c. 13]. Оскільки ми маємо систему, яка не піддається опису аналітичними функціями і для визначення найбільш оптимального варіанту доведеться робити багато прогонів системи для того, щоб отримати середні значення критеріїв, цей варіант нам підходить найбільше.

Ми зрозуміли, що для наших потреб варто обрати імітаційне моделювання. Тепер нам треба обрати засіб для реалізації цього способу. Імітаційне моделювання передбачає за собою, що процес функціонування системи відтворюється за допомогою алгоритму, який ми реалізуємо за допомогою комп’ютера.

Розрізняють декілька варіантів програм, які реалізують імітаційне моделювання:

* Написані на спеціальних мовах для імітаційного моделювання (GPSS, SIMULA).
* Написані на універсальних мовах програмування (Python, C#, Java).

Критичною відмінність між цими двома програмами полягає у гнучкості. Так, ми із легкістю зможемо змоделювати систему із використанням інструментів, які були зроблені лише для цього, але ми втрачаємо велику кількість можливостей для аналізу. Набір операторів також обмежений, а можливості створення імітаційної модель сильно обмежені вибором запропонованих операторів. Користуючись універсальними мовами програмування ми отримаємо необмежені можливості будуючи імітаційні алгоритми, оскільки ми самі ці можливості будемо програмувати.

Реалізовуючи програму для імітації ми маємо декілька нюансів. А саме, нам потрібно вручну написати наступні методи:

* Засоби генерації випадкових чисел та змінних.
* Організацію збирання статистичних даних під час роботи моделі.
* Відображення стану моделі у деякий момент часу.
* Можливість провести верифікацію алгоритму імітації.

# **РОЗДІЛ 1. РОЗРОБКА КОНЦЕПТУАЛЬНОЇ МОДЕЛІ**

Вказання цілі задачі дослідження разом із описом системи складає концептуальну модель системи.

*Ціль моделювання* – визначитись із оптимальним способом вибору замовлень із черг обслуговувань.

*Структурна схема моделі* – об’єкт моделювання являє собою систему для створення програмного забезпечення для керування металорізальних верстатів із числовим програмним керуванням. Замовлення надходять з деякою інтенсивністю від відділу креслень. Після надходження, для завдання визначаються його часові затримки для виконання. Проміжок часу для цих підрахунків розглядається як додатковий. Коли були розраховані часові затримки, то для завдання має бути визначений директивний термін – момент у часі, до якого завдання повинно бути виконано. Це сума часу виконання всіх процесів обслуговування, часу надходження замовлення та додаткового часу. Після цього завдання надходить до програміста, що пише програмне забезпечення для керування за деякий визначений час. Як тільки програма була розроблена, вона записується на носій. Ця дія також виконується за час, який було визначено на попередніх етапах. В кінці системи, програма ставиться та тестується на верстатах. Якщо завдання надійшло до процесу, а він зайнятий, то замовлення буде чекати у черзі.

*Вхідні змінні:*

* Час моделювання системи.
* Тип пріоритету вибору завдання із черги.
* Інтенсивність надходження замовлень із відділу креслень.
* Послідовність повного процесу створення та встановлення програми на верстат.
* Часові затримки оброблень замовлень у різних процесах.

*Вихідні змінні:*

* Максимальна кількість замовлень, що очікували в чергах процесів за час моделювання.
* Завантаженість відділів за час моделювання.
* Коефіцієнт відношення між замовленнями, що надійшли до системи за час моделювання та загальною кількістю оброблених.
* Кількість оброблених замовлень на кожному етапі.
* Кількість замовлень, які надійшли у систему.
* Загальна середня завантаженість відділів за час моделювання.
* Середня кількість замовлень в чергах для кожного відділу.
* Загальне середнє значення замовлень, що очікували в чергах за час моделювання.

*Формули для вихідних змінних:*

Для підрахунку коефіцієнту оброблених замовлень за визначений час моделювання ми будемо використовувати формулу (1.1):

, (1.1)

Де – це загальна кількість успішно оброблених замовлень; – загальна кількість замовлень, що надійшли до системи.

Для отримання середньої завантаженості відділів за визначений час моделювання ми будемо використовувати наступну формулу (1.2):

, (1.2)

Де – спостережуване значення зайнятості процесу; – проміжок часу, протягом якого ми спостерігали значення завантаженості; – загальний час моделювання системи.

Для отримання середнього значення черги за визначений час моделювання ми використаємо наступну формулу (1.3):

, (1.3)

Де – спостережуване значення довжини черги; – проміжок часу, протягом якого ми спостерігали значення довжини черги; – загальний час моделювання системи.

*Функціональна залежність* – зміна пріоритетів вибору завдання із черги впливає на завантаженість кожного етапу за деякий час моделювання, на максимальну кількість замовлень які очікували в чергах процесів за деякий час моделювання, на кількість оброблених замовлень на кожному етапі, а також на середню кількість замовлень в чергах на кожному етапі.

Приклад:

У нас є три замовлення, що очікують у черзі на виконання в процесі обслуговування. Загальний час виконання кожної роботи наступні:

* Перше замовлення – 10 часових одиниць.
* Друге замовлення – 6 часових одиниць.
* Третє замовлення – 4 часових одиниць.

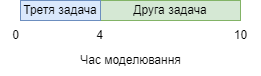
При цьому загальний час моделювання у нас визначений, як 10 часових одиниць. На наступному малюнку зображено взяття замовлення за умови пріоритетності вибору задач із найбільшим загальним часом виконання:



*Рисунок 1.1* – діаграма Ганта для виконання завдання із пріоритетністю найбільшого загального часу

Якщо ми виберемо такий тип розподілу завдання, то середня довжина черги за формулою 1.3 залишиться рівною двом задачам, а за встановлений час моделювання буде оброблена лише одна задача.

На наступному малюнку зображено взяття замовлення за умови пріоритетності вибору задач із найменшим загальним часом виконання:



*Рисунок 1.2* – діаграма Ганта для виконання завдання із пріоритетністю найменшого загального часу

Якщо ми виберемо такий тип розподілу завдання, то середня довжина черги за формулою 1.3 буде 1.4, а за встановлений час моделювання буде оброблено вже дві задачі. Таким чином ми довели вплив параметру на вихідні змінні.

*Цільова функція системи* – для компанії, яка створює програмне забезпечення для керування металорізальних верстатів, бажаним результатом є максимальна середня завантаженість кожного етапу при мінімальній середній кількості замовлень у чергах.

*Обмеження* – з вихідних змінних можна варіювати лише типом пріоритетності вибору замовлення із черги. Для різних процесів обслуговування можна вказувати різні типи пріоритету вибору завдання із черги, але лише один варіант для кожного процесу.

*Схема концептуальної моделі*

Схему було відтворено за вербальним описом на наступному рисунку:

  
*Рисунок 1.3* – схема концептуальної моделі

# **РОЗДІЛ 2. РОЗРОБКА ФОРМАЛІЗОВАНОЇ МОДЕЛІ**

Після того, як ми сформували концептуальну модель, ми маємо визначити теоретичну базу для побудови формалізованої моделі. Найбільш поширеними та відомими способами формалізації процесів функціонування дискретних систем серед спеціалістів є мережі Петрі та мережі масового обслуговування.

Мережі масового обслуговування призначаються для описання процесів обробки, таких як виробничі процеси, бізнес-процеси, і представляють великий, але чітко обмежений клас систем. Мережі Петрі являються більш потужнішим засобом формалізації дискретних процесів і описують системи із найскладнішими зв’язками, в тому числі управляючими [1] [c. 62].

Мережі Петрі надають перевагу у випадках, коли потрібно формалізувати систему, яка має спільні ресурси для багатьох процесів обслуговування та коли є потреба виконувати паралельні процеси. При цьому програмувати мережі Петрі набагато складніше, ніж мережі масового обслуговування. У нашому випадку, відповідно до сформованої концептуальної моделі розділом вище, модель є доволі простою, і для того, щоб пришвидшити аналіз а також через те, що переваги використання мережі Петрі будь мінімальними, ми будемо використовувати мережу масового обслуговування для нашої задачі.

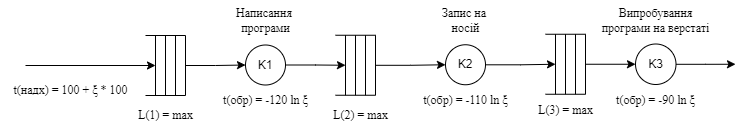
На наступному рисунку зображено елементи формалізації мережі масового обслуговування [1] [c. 64]:



*Рисунок 2.1* – елементи формалізації мережі масового обслуговування

## **2.1 Опис імітаційної моделі**

Використовуючи елементи для побудови мережі масового обслуговування наведені вище, зобразимо наступну формалізовану мережу масового обслуговування виходячи із поставленого завдання:



*Рисунок 2.2* – мережа масового обслуговування для поставленого завдання

Побудована мережа масового обслуговування складається із наступних елементів:

* Момент надходження замовлень від відділу креслень.
* Три пристрої, що будуть обслуговувати замовлення із різними часовими затримками, які наведені на малюнку.
* Необмежені черги перед кожним із пристроїв.

# **РОЗДІЛ 3. ВИБІР ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ РЕАЛІЗАЦІЇ ЗАДАЧІ**

Нам потрібно обрати на якій із універсальних мов програмування ми будемо проводити моделювання. Для цієї задачі нам найкращим чином будуть підходити мови, які реалізують можливість написання алгоритмів використовуючи об’єктно орієнтовний підхід, оскільки ми будемо мати повторення коду для елементів моделі, яке можна уникнути способом наслідування. Одним із найпопулярніших варіантів є Java, C# та Python. Із цих трьох мов програмування ми виберемо Python.

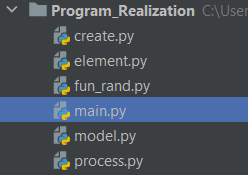
Даний вибір був здійснений через те, що дана мова надає безліч різноманітних та легкий у використанні бібліотек для роботи із графіками, таблицями та математичними функціями, що робить її дуже зручною для виконання завдань даної роботи.

Python – інтерпретована об'єктно-орієнтована мова програмування високого рівня зі строгою динамічною типізацією [2]. Вона була розроблена в 1990 році Гвідо ван Россумом. Динамічна семантика та динамічне зв'язування роблять її привабливою для швидкої розробки програм, а також як засіб поєднування вже існуючих компонентів. Python підтримує різні модулі, що сприяє її модульності та повторному використанню коду.

# **РОЗДІЛ 4. РЕАЛІЗАЦІЯ МОДЕЛІ ІЗ ВИКОРИСТАННЯМ УНІВЕРСАЛЬНОЇ МОВИ ПРОГРАМУВАННЯ**

Для цієї роботи нам знадобиться бібліотека під назвою Pandas [3] для запису результатів дослідження у Excel таблиці щоб мати більш зручний доступу до даних, бібліотека Numpy [4] для зручної роботи із математичними задачами цієї курсової роботи та Matplotlib [5] для зручної роботи із графіками.

Структура файлів програмного забезпечення для моделювання зображені на наступному малюнку:



*Рисунок 4.1* – структура файлів розробленого ПЗ

У таблиці 4.1 пояснюється сутність кожного файлу до проекту.

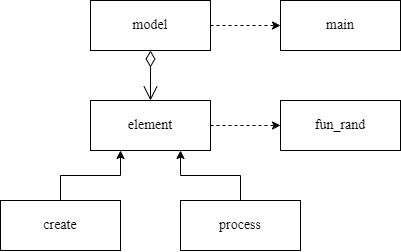
**Таблиця 4.1 – сутності кожного файлу проекту**

|  |  |
| --- | --- |
| **Назва файлу** | **Пояснення** |
| *element.py* | Містить у собі загальну характеристику елемента мережі масового обслуговування. Використовується для наслідування. |
| *create.py* | Опис надходження замовлення. |
| *fun\_rand.py* | Містить у собі функції, які описують закони розподілу. |

**Продовження таблиці 4.1**

|  |  |
| --- | --- |
| **Назва файлу** | **Пояснення** |
| *main.py* | Точка входу у програму. Задання імітаційної моделі. |
| *model.py* | Алгоритм імітації моделювання. |
| *process.py* | Опис системи масового обслуговування. |

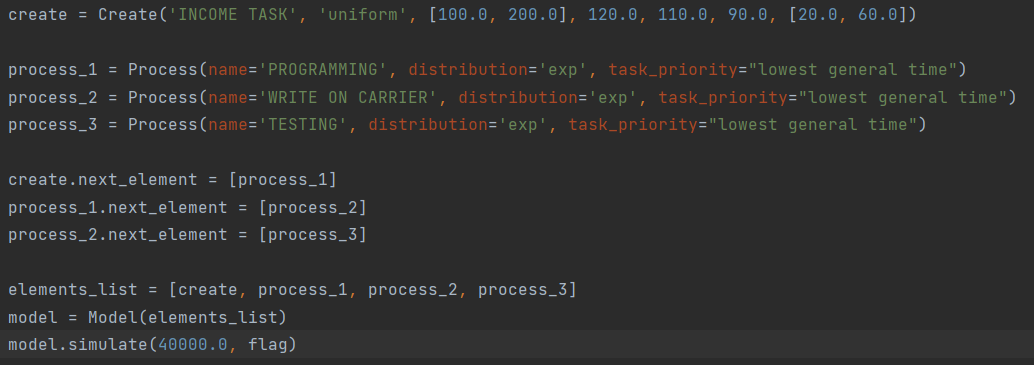
Для розробки програмної реалізації ми використали об’єктно-орієнтовний підхід. На наступному рисунку зображено зв’язок між сутностями написаного програмного забезпечення:



*Рисунок 4.2* – поєднання сутностей файлів

Написаний алгоритм імітації знаходиться у файлі *model.py*, а тому це є головний файл у ієрархії наведеній вище. Точкою входу у програму служить файл *main.py*. У ньому ж ми і описали нашу модель. Побудовані імітаційні моделі використовують елементи формалізації *create.py* та *process.py*, які унаслідуються від файлу *element.py*. Оскільки програмне забезпечення написане на універсальній мові програмування, то просування стану моделі в часі відбувається за принципом орієнтованим на події.

У розробленій програмній реалізації відсутній графічний спосіб задання моделі, а тому це відбувається напряму – через написання програмного коду. На наступному рисунку зображений скріншот програмної реалізації алгоритму імітації моделі поставленого завдання:



*Рисунок 4.3* – мережа масового обслуговування поставленого завдання

Відповідно до цього скріншоту ми створюємо декілька об’єктів:

* *create* типу Create – як елемент надходження
* *process\_1* типу Process – пристрій обслуговування (етап програмування)
* *process\_2* типу Process – пристрій обслуговування (етап запису на носій)
* *process\_3* типу Process – пристрій обслуговування (етап тестування)
* *element\_list* типу list – список із усіма пристроями обслуговування
* *model* типу Model – безпосередньо сама модель для процесу моделювання

Об’єкт *create* приймає свій іменний ідентифікатор, закон розподілу та параметри інтервалів затримок для обслуговування замовлення у різних відділах. Об’єкти *process\_1, process\_2, process\_3* приймають параметрами свій іменний ідентифікатор, закон розподілу та тип пріоритету для замовлення у черзі. Об’єкт *.next\_element* вказує маршрут слідування від одного елементу обслуговування до іншого. Об’єкт *model* приймає список об’єктів імітаційної моделі і моделюється за допомогою методу *.simulate*, що приймає параметром час моделювання та спеціальний флаг, який говорить чи будемо ми проводити вивід інформації про стан мережі масового обслуговування.

У таблиці 4.2 наведемо опис методів, які застосовуються у класах.

**Таблиця 4.2 – опис методів**

|  |  |  |  |
| --- | --- | --- | --- |
| **Назва методу** | **Опис** | **Вхідні змінні** | **Вихідні змінні** |
| *Клас Element* | | | |
| *def \_\_init\_\_()* | Метод для ініціалізації об’єкту елементу мережі масового обслуговування | name\_element – ім’я створюваного елементу, delay – часова затримка,  distribution – тип закону розподілу | Відсутні |
| *def get\_delay()* | Генерування  часової затримки  за деяким заданим законом  розподілу | Відсутні | Часова затримка отримана за вказаним законом розподілу |
| *def in\_act()* | Вхід замовлення до елементу | priority – тип пріоритету | Відсутні (реалізація цього методу покладена на унаслідуванні класи) |
| *def out\_act()* | Вихід замовлення із елементу | Відсутні | Збільшення  кількості  оброблених  замовлень |
| *def statistics()* | Збирання  статистичних  даних про  функціонування  моделі | delta – проміжок  часу  спостережуваних  величин | Відсутні (реалізація цього методу покладена на унаслідуванні класи) |
| *def result()* | Вивід статистики  для елементу за деякий  час моделювання | Відсутні | Статистика  оброблених  замовлень для  елементу та статус зайнятості елементу |

**Продовження таблиці 4.2**

|  |  |  |  |
| --- | --- | --- | --- |
| **Назва методу** | **Опис** | **Вхідні змінні** | **Вихідні змінні** |
| *Клас Element* | | | |
| *def print\_info()* | Показ інформації  про стан елементу  на кожному кроці  просування моделі |  | Інформація про  ім’я елементу,  стан зайнятості,  кількість  оброблених  замовлень  елементом та момент настання наступної події. |
| *Клас Create* | | | |
| *def \_\_init\_\_()* | Метод для ініціалізації об’єкту елементу мережі масового обслуговування | name – ім’я створюваного елементу,  distribution – тип закону розподілу,  interval1 – інтервал часу надходження замовлення на підготовку носіїв з програмами, elem1\_delay – середній час програмування,  elem2\_delay - середній час запису на носій,  elem3\_delay – середній час тестування,  interval2 – інтервал додаткового часу для розрахунку технологічного часу виконання | Відсутні |

**Продовження таблиці 4.2**

|  |  |  |  |
| --- | --- | --- | --- |
| **Назва методу** | **Опис** | **Вхідні змінні** | **Вихідні змінні** |
| *Клас Create* | | | |
| *def out\_act()* | Надходження  замовлення до  системи | Відсутні | Згенеровані  характеристики  часових  затримок та  директивного  терміну  завдання,  збільшення  кількості  замовлень, що  надійшли |
| *def print\_event\_info()* | Показ  характеристик  згенерованих  часових затримок  обробки  замовлення у СМО | flag – спеціальний індикатор того, чи треба виводити цю інформацію | Виведення  інформації про  характеристики  нового  замовлення у  системі |
| *Клас Process* | | | |
| *def \_\_init\_\_()* | Ініціалізація  об’єкту системи  масового  обслуговування | taks\_priority – тип  пріоритетності  замовлення для  вибору з черги, name – ім’я створюваного елементу,  distribution – тип закону розподілу,  channels – кількість каналів у СМО | Відсутні |

**Продовження таблиці 4.2**

|  |  |  |  |
| --- | --- | --- | --- |
| **Назва методу** | **Опис** | **Вхідні змінні** | **Вихідні змінні** |
| *Клас Process* | | | |
| *def in\_act()* | Надходження  нового замовлення  до СМО | task\_properties – характеристика  замовлення з  часовою  затримкою для  обслуговування | Відсутні |
| *def out\_act()* | Вихід замовлення  зі СМО | Відсутні | Збільшення кількості замовлень, що  були оброблені, розрахований момент наступної події |
| *def get\_free\_channels()* | Знаходження вільних каналів | Відсутні | Список вільних каналів обслуговування |
| *def get\_current\_channel()* | Визначення поточного каналу у якому ведеться обробка замовлення | Відсутні | Список поточних каналів обслуговування |
| *def print\_info()* | Інформація про  стан пристрою та  черг | Відсутні | Виведення  інформації у  момент часу |
| *def statistics()* | Збирання  статистичних  даних про  функціонування  моделі | delta – проміжок  часу  спостережуваних  величин | Значення черги та завантаження за спостережуваний інтервал часу |

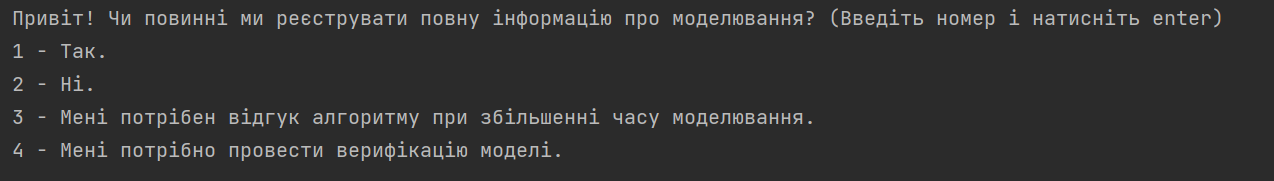
**Продовження таблиці 4.2**

|  |  |  |  |
| --- | --- | --- | --- |
| **Назва методу** | **Опис** | **Вхідні змінні** | **Вихідні змінні** |
| *Клас Model* | | | |
| *def \_\_init\_\_()* | Ініціалізація об’єкту імітаційної моделі | element\_list – набір елементів, з яких побудована модель | Відсутні |
| *def simulate()* | Алгоритм імітації моделювання системи | time – загальний час моделювання, flag – спеціальний індикатор того, чи треба виводити додаткову інформацію | Стан моделі на деякому кроці, загальні  результати  моделювання,  статистичні дані |
| *def print\_info()* | Вивід інформації про деякий елемент моделі | flag – спеціальний індикатор того, чи треба виводити цю інформацію | Інформація про ім’я елементів,  стан зайнятості,  кількість  оброблених  замовлень  елементами та момент настання наступної події. |
| *def print\_result()* | Вивід інформації про результат моделювання | flag – спеціальний індикатор того, чи треба виводити цю інформацію | Інформація про  середню довжину черги, максимальне значення черги, середнє завантаження, задачі які провалили директивний термін, відношення успішних задач до задач які надійшли |

**Продовження таблиці 4.2**

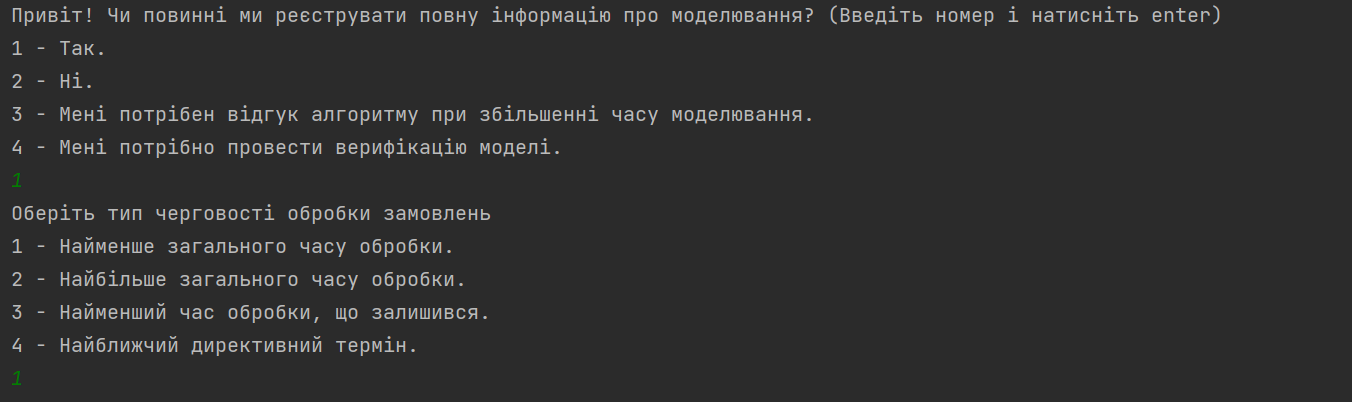
|  |  |  |  |
| --- | --- | --- | --- |
| **Назва методу** | **Опис** | **Вхідні змінні** | **Вихідні змінні** |
| *Клас Model* | | | |
| *def experiments()* | Збір інформації про експерименти у таблицю | Відсутні | Таблиця, що має у собі результати експерименту, а саме середнє значення черг, середнє завантаження та коефіцієнт виконаних завдань |

Так, як цільову функцію та її обмеження для змінних не можна описати аналітичним способом, то для оцінки працездатності та адекватності алгоритму імітації ми будемо спостерігати його покрокову роботу. Часові затримки при цьому ми візьмемо із постановки завдання. Запустимо програму. При першому запуску ми побачимо наступне текстове меню:

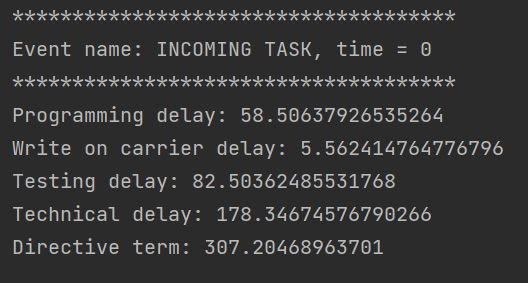


*Рисунок 4.4* – меню додатку

Для нашого випадку, нам потрібно ввести «1» для того, щоб нам повністю показувало, як працює алгоритм. Тепер, просто для перевірки, ми будемо використовувати найменший загальний час як тип вибору пріоритетності задачі. Для цього у меню знову введемо «1»:

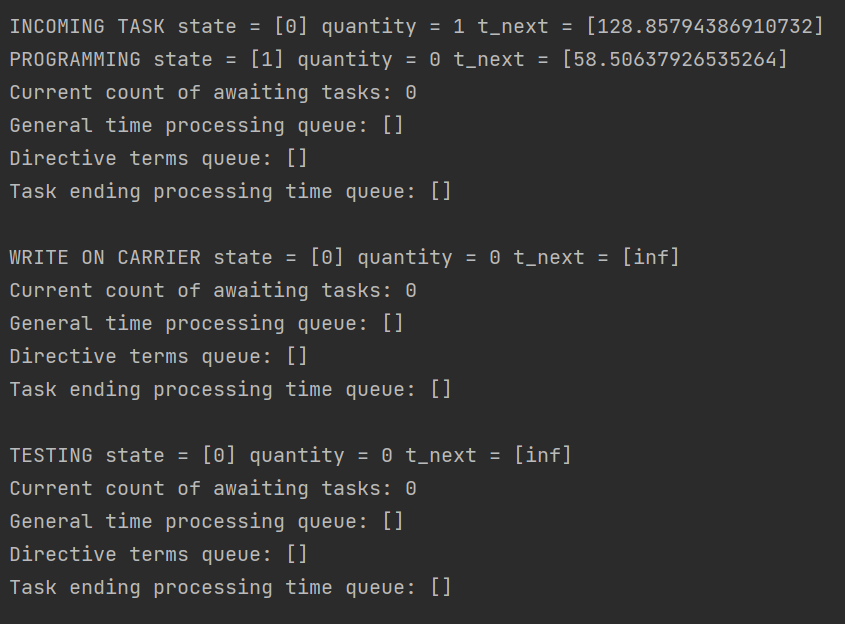


*Рисунок 4.5* – вибір типу черговості обробки



*Рисунок 4.6* – поточна подія у моделі

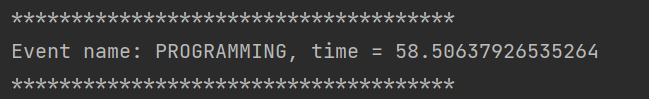
У часовий момент 0 надходить замовлення від відділу креслення. Для кожного завдання формуються часові затримки оброблень у СМО та директивний термін. На наступному рисунку зображена інформація про стан моделі у даний момент часу:



*Рисунок 4.7* – стан моделі у момент часу

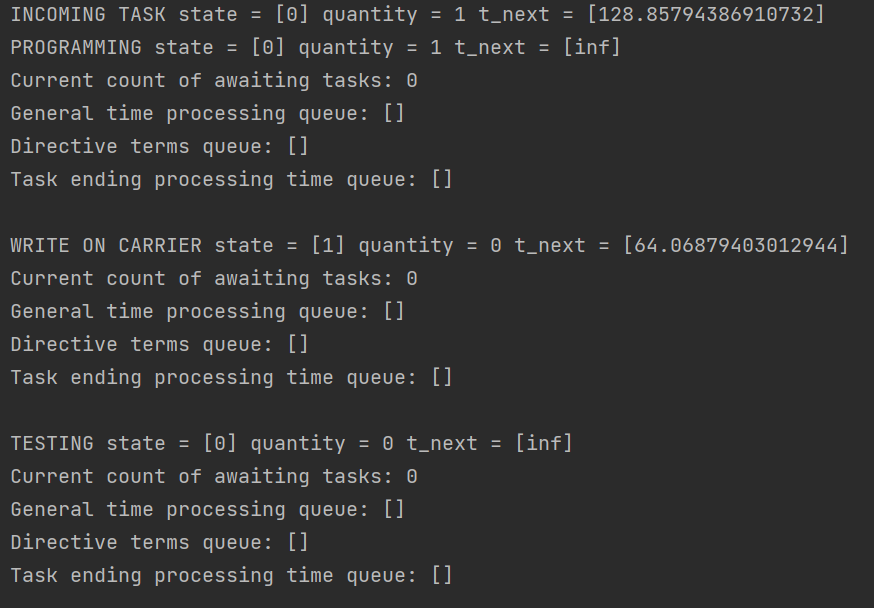
Для алгоритму імітації розраховується наступний момент часу для елементів моделі, у кожного елементу ця характеристика представлена значенням *t\_next*. Вибір наступної події відбувається за визначенням мінімального найближчого часу наступної події елементів моделі.

У пристроях обслуговування зазначені різні черги щодо характеристик замовлень. Це було зумовлено поставленим завдання та можливістю змінювати пріоритетність вибору завдання з черги за характеристикою. За визначеним маршрутом слідування, замовлення має надійти до відділу програмування. Це перше замовлення, а тому відділ його виконує одразу, тобто воно не стає у чергу. На наступному рисунку зображена подія, що була найближчою у проміжку модельованого часу:



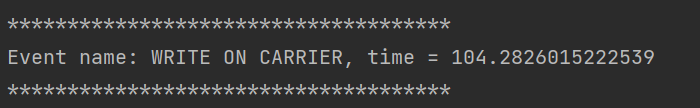
*Рисунок 4.8* – наступна подія

Відповідно до встановленого часу обробки замовлення, процес завершився у момент часу рівним ~58.8 часових одиниць. На наступному рисунку зображено стан моделі у цей момент часу:

**

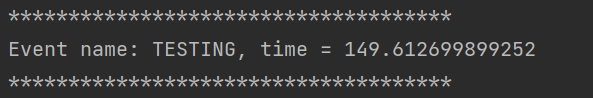
*Рисунок 4.9* – стан моделі у момент часу

За тією ж самою логікою, замовлення, із плином часу, перейде у фазу запису на пристрій:



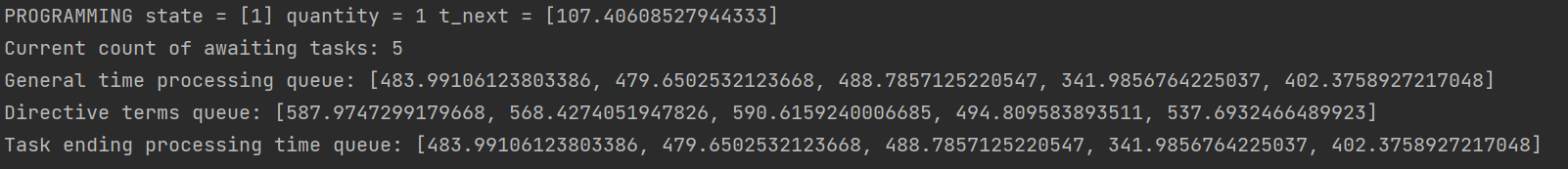
*Рисунок 4.10* – наступна подія

А потім і тестування:



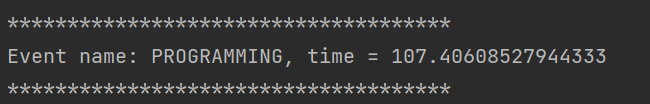
*Рисунок 4.11* – наступна подія

Правильність маршруту слідування та адекватність алгоритму імітації доведена. Тепер ми маємо показати, що завдання обирається з черги за вказаним пріоритетом. Для цього прогону ми обрали пріоритет із найменшим загальним часом обробки для всіх СМО. Для перевірки правильності знайдемо стан моделі в якому знаходяться замовлення в черзі. На наступному рисунку зображено стан відділу програмування, де знаходяться п’ять замовлень у черзі:



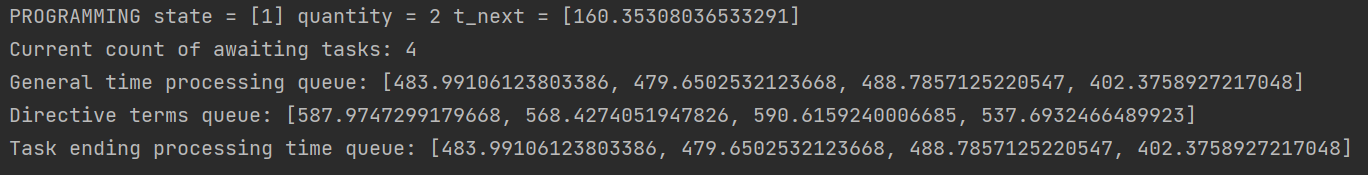
*Рисунок 4.12*  – черга замовлень

Завершення оброблення поточного замовлення відбудеться у моменті 107.4060, за встановленим типом пріоритетності, після завершення оброблення цієї деталі, повинно почати обробляти замовлення з загальним часом 341.985. На наступному рисунку зображено подію закінчення оброблення в відділі програмування:



*Рисунок 4.13* – наступна подія

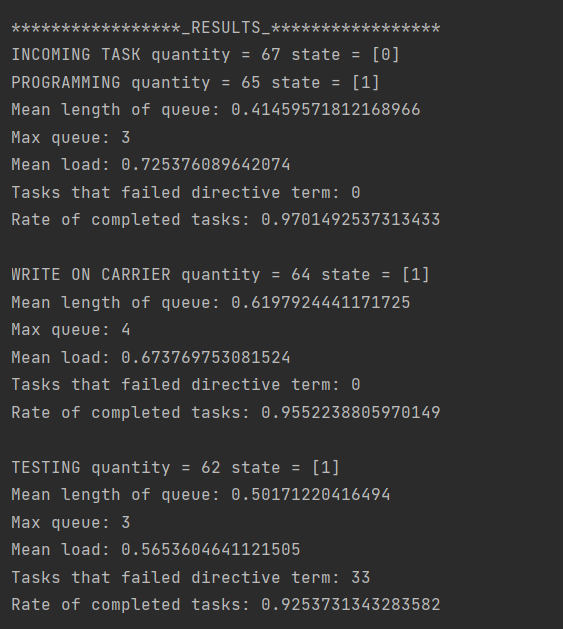
На наступному рисунку зображено стан відділу програмування у цей момент часу:

****

*Рисунок 4.14* – стан відділу програмування

За текстовим описом програми можна спостерігати, що у нас відбулось збільшення кількості оброблених елементів, а черга зменшилась. Щодо правильності взяття із черги, ми можемо побачити, що завдання, яке мало найменший час обробки (~341.98, як наведено на рисунку 4.9) пішло в обробку і зникло із черги. Тобто алгоритм вибору за заданим пріоритетом працює.

На наступному рисунку зображені загальні результати імітаційного моделювання за час 10000 од:

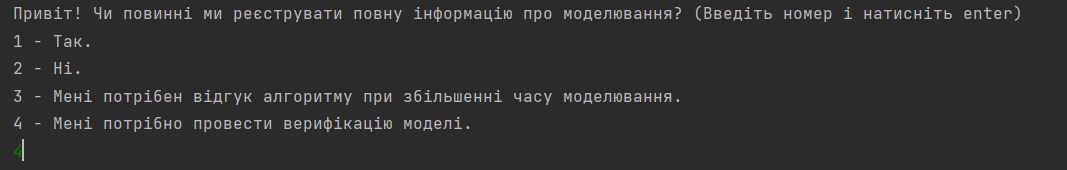


*Рисунок 4.15* – результат моделювання

За вказаний час моделювання ми отримали 67 замовлень, з них повністю обробилися 62. Коефіцієнт виконання становить 0.925. Також наведені максимальні та середні значення черг до СМО, та середню завантаженість за час моделювання. Кількість замовлень, які не були виконані у встановлений директивний термін дорівнює 33.

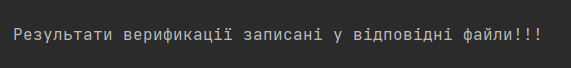
## **4.1 Верифікація алгоритму імітації**

Для верифікації алгоритму імітації здійснюється порівняння вихідних результатів моделювання за вказаний час зі зміною вхідних параметрів моделі. Для цього ми будемо проводити верифікацію для кожного типу вибору пріоритетності із черг. Для цього ми запустимо програму у відповідному режимі. Виберемо пункт «4» коли будемо запускати програму:



*Рисунок 4.16* – вибір режиму «верифікації»

Результатом виконання програми буде 4 файли із таблицями, де будуть записані результати верифікації для кожного типу вибору пріоритетності.



*Рисунок 4.17* – результат роботи програми

У таблиці 4.3 наведено результати верифікації для типу пріоритетності, коли затрачується найменше загального часу обробки:

**Таблиця 4.3 – верифікація найменше загального часу**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Номер прогону** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** |
| Мінімальні затримки надходження | 100 | **0** | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 |
| Максимальні затримки надходження | 200 | 200 | **1000** | 200 | 200 | 200 | 200 | 200 | 200 | 200 | 200 |
| Затримка програмування | 120 | 120 | 120 | **0** | **500** | 120 | 120 | 120 | 120 | 120 | 120 |
| Затримка запису | 110 | 110 | 110 | 110 | 110 | **0** | **500** | 110 | 110 | 110 | 110 |
| Затримка тестування | 90 | 90 | 90 | 90 | 90 | 90 | 90 | **0** | **500** | 90 | 90 |

**Продовження таблиці 4.3**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Номер прогону** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** |
| Мінімальні затримки дод. часу | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | **0** | 20 |
| Максимальні затримки дод. часу | 60 | 60 | 60 | 60 | 60 | 60 | 60 | 60 | 60 | 60 | **1000** |
| Середня черга на програмування | 0,83 | 6,26 | 0,008 | 0 | 12,8 | 0,84 | 0,84 | 0,79 | 1,34 | 0,84 | 0,86 |
| Середня черга на запис | 0,64 | 1,96 | 0,02 | 0,71 | 0,12 | 0 | 12,7 | 0,75 | 1,1 | 0,64 | 0,68 |
| Середня черга на тестування | 0,4 | 0,81 | 0,01 | 0,32 | 0,12 | 0,37 | 0,09 | 0 | 11,5 | 0,37 | 0,47 |
| Середня завантаженість на програмування | 0,8 | 0,97 | 0,21 | 0 | 0,99 | 0,79 | 0,75 | 0,78 | 0,8 | 0,79 | 0,78 |
| Середня завантаженість на запис | 0,67 | 0,81 | 0,19 | 0,76 | 0,39 | 0 | 0,98 | 0,71 | 0,69 | 0,68 | 0,67 |
| Середня завантаженість на тестування | 0,54 | 0,67 | 0,16 | 0,56 | 0,35 | 0,57 | 0,31 | 0 | 0,96 | 0,55 | 0,57 |
| Коефіцієнт виконання на програмуванні | 0,98 | 0,89 | 0,98 | 1 | 0,62 | 0,97 | 0,97 | 0,98 | 0,97 | 0,97 | 0,97 |

**Продовження таблиці 4.3**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Номер прогону** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** |
| Коефіцієнт виконання на записі | 0,95 | 0,84 | 0,96 | 0,98 | 0,61 | 0,97 | 0,60 | 0,96 | 0,93 | 0,95 | 0,95 |
| Коефіцієнт виконання на тестуванні | 0,93 | 0,83 | 0,95 | 0,96 | 0,61 | 0,95 | 0,6 | 0,96 | 0,58 | 0,93 | 0,93 |

У таблиці 4.4 наведено результати верифікації для типу пріоритетності, коли затрачується найбільше загального часу обробки:

**Таблиця 4.4 – верифікація найбільше загального часу**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Номер прогону** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** |
| Мінімальні затримки надходження | 100 | **0** | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 |
| Максимальне затримки надходження | 200 | 200 | **1000** | 200 | 200 | 200 | 200 | 200 | 200 | 200 | 200 |
| Затримка програмування | 120 | 120 | 120 | **0** | **500** | 120 | 120 | 120 | 120 | 120 | 120 |
| Затримка запису | 110 | 110 | 110 | 110 | 110 | **0** | **500** | 110 | 110 | 110 | 110 |
| Затримка тестування | 90 | 90 | 90 | 90 | 90 | 90 | 90 | **0** | **500** | 90 | 90 |

**Продовження таблиці 4.4**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Номер прогону** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** |
| Мінімальні затримки дод. часу | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | **0** | 20 |
| Максимальні затримки дод. часу | 60 | 60 | 60 | 60 | 60 | 60 | 60 | 60 | 60 | 60 | **1000** |
| Середня черга на програмування | 1,75 | 17,6 | 0,01 | 0 | 28,5 | 1,25 | 1,49 | 1,98 | 1,72 | 1,55 | 1,10 |
| Середня черга на запис | 1,78 | 3,71 | 0,03 | 1,2 | 0,007 | 0 | 26,2 | 1,62 | 1,23 | 1,18 | 2,21 |
| Середня черга на тестування | 0,71 | 0,90 | 0,01 | 0,80 | 0,002 | 0,7 | 0,001 | 0 | 24 | 0,8 | 1,08 |
| Середня завантаженість на програмування | 0,81 | 0,97 | 0,22 | 0 | 0,99 | 0,77 | 0,75 | 0,80 | 0,78 | 0,77 | 0,75 |
| Середня завантаженість на запис | 0,71 | 0,8 | 0,19 | 0,76 | 0,10 | 0 | 0,97 | 0,66 | 0,65 | 0,67 | 0,72 |
| Середня завантаженість на тестування | 0,53 | 0,58 | 0,15 | 0,58 | 0,08 | 0,56 | 0,09 | 0 | 0,96 | 0,54 | 0,55 |
| Коефіцієнт виконання на програмуванні | 0,93 | 0,64 | 0,97 | 1 | 0,12 | 0,97 | 0,97 | 0,92 | 0,96 | 0,94 | 0,96 |

**Продовження таблиці 4.4**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Номер прогону** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** |
| Коефіцієнт виконання на записі | 0,89 | 0,56 | 0,96 | 0,97 | 0,12 | 0,97 | 0,12 | 0,88 | 0,93 | 0,91 | 0,91 |
| Коефіцієнт виконання на тестуванні | 0,87 | 0,54 | 0,94 | 0,94 | 0,12 | 0,95 | 0,12 | 0,88 | 0,13 | 0,89 | 0,87 |

У таблиці 4.5 наведено результати верифікації для типу пріоритетності, при якому на виконання задач затрачується найменший час обробки, що залишився:

**Таблиця 4.5 – верифікація найменший час обробки, що залишився**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Номер прогону** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** |
| Мінімальні затримки надходження | 100 | **0** | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 |
| Максимальні затримки надходження | 200 | 200 | **1000** | 200 | 200 | 200 | 200 | 200 | 200 | 200 | 200 |
| Затримка програмування | 120 | 120 | 120 | **0** | **500** | 120 | 120 | 120 | 120 | 120 | 120 |
| Затримка запису | 110 | 110 | 110 | 110 | 110 | **0** | **500** | 110 | 110 | 110 | 110 |
| Затримка тестування | 90 | 90 | 90 | 90 | 90 | 90 | 90 | **0** | **500** | 90 | 90 |

**Продовження таблиці 4.5**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Номер прогону** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** |
| Мінімальні затримки дод. часу | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | **0** | 20 |
| Максимальні затримки дод. часу | 60 | 60 | 60 | 60 | 60 | 60 | 60 | 60 | 60 | 60 | **1000** |
| Середня черга на програмування | 1,09 | 5,75 | 0,01 | 0 | 13,6 | 0,75 | 1,01 | 0,86 | 1,15 | 0,93 | 0,81 |
| Середня черга на запис | 0,69 | 1,88 | 0,01 | 0,51 | 0,11 | 0 | 11,5 | 0,62 | 1,13 | 0,71 | 0,73 |
| Середня черга на тестування | 0,38 | 0,57 | 0,01 | 0,3 | 0,06 | 0,34 | 0,09 | 0 | 10,24 | 0,33 | 0,36 |
| Середня завантаженість на програмування | 0,82 | 0,97 | 0,20 | 0 | 1 | 0,77 | 0,79 | 0,79 | 0,78 | 0,78 | 0,78 |
| Середня завантаженість на запис | 0,7 | 0,8 | 0,19 | 0,69 | 0,39 | 0 | 0,97 | 0,69 | 0,70 | 0,67 | 0,71 |
| Середня завантаженість на тестування | 0,55 | 0,66 | 0,15 | 0,55 | 0,31 | 0,55 | 0,33 | 0 | 0,97 | 0,55 | 0,57 |
| Коефіцієнт виконання на програмуванні | 0,98 | 0,89 | 0,97 | 1 | 0,6 | 0,97 | 0,97 | 0,97 | 0,96 | 0,98 | 0,97 |

**Продовження таблиці 4.5**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Номер прогону** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** |
| Коефіцієнт виконання на записі | 0,94 | 0,85 | 0,95 | 0,98 | 0,59 | 0,97 | 0,62 | 0,94 | 0,92 | 0,95 | 0,95 |
| Коефіцієнт виконання на тестуванні | 0,93 | 0,84 | 0,95 | 0,97 | 0,58 | 0,95 | 0,61 | 0,94 | 0,60 | 0,93 | 0,93 |

У таблиці 4.6 наведено результати верифікації для типу пріоритетності, при якому спочатку виконуються ті замовлення, які мають найближчий директивний термін.

**Таблиця 4.6 – верифікація найближчий директивний термін**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Номер прогону** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** |
| Мінімальні затримки надходження | 100 | **0** | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 |
| Максимальні затримки надходження | 200 | 200 | **1000** | 200 | 200 | 200 | 200 | 200 | 200 | 200 | 200 |
| Затримка програмування | 120 | 120 | 120 | **0** | **500** | 120 | 120 | 120 | 120 | 120 | 120 |
| Затримка запису | 110 | 110 | 110 | 110 | 110 | **0** | **500** | 110 | 110 | 110 | 110 |
| Затримка тестування | 90 | 90 | 90 | 90 | 90 | 90 | 90 | **0** | **500** | 90 | 90 |

**Продовження таблиці 4.6**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Номер прогону** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** |
| Мінімальні затримки дод. часу | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | **0** | 20 |
| Максимальні затримки дод. часу | 60 | 60 | 60 | 60 | 60 | 60 | 60 | 60 | 60 | 60 | **1000** |
| Середня черга на програмування | 1,09 | 11 | 0,01 | 0 | 20,9 | 1,26 | 1,18 | 0,86 | 0,87 | 1,59 | 1,03 |
| Середня черга на запис | 0,90 | 2,7 | 0,01 | 0,57 | 0,06 | 0 | 19,1 | 1,03 | 1,32 | 1,06 | 1,01 |
| Середня черга на тестування | 0,48 | 1 | 0,01 | 0,43 | 0,03 | 0,49 | 0,05 | 0 | 18 | 0,59 | 0,49 |
| Середня завантаженість на програмування | 0,78 | 0,97 | 0,21 | 0 | 1 | 0,79 | 0,81 | 0,78 | 0,76 | 0,83 | 0,76 |
| Середня завантаженість на запис | 0,69 | 0,79 | 0,21 | 0,71 | 0,23 | 0 | 0,99 | 0,69 | 0,73 | 0,68 | 0,71 |
| Середня завантаженість на тестування | 0,54 | 0,63 | 0,15 | 0,57 | 0,18 | 0,57 | 0,2 | 0 | 0,96 | 0,55 | 0,55 |
| Коефіцієнт виконання на програмуванні | 0,96 | 0,77 | 0,98 | 1 | 0,34 | 0,97 | 0,96 | 0,98 | 0,97 | 0,96 | 0,97 |

**Продовження таблиці 4.6**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Номер прогону** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** |
| Коефіцієнт виконання на записі | 0,94 | 0,72 | 0,97 | 0,98 | 0,32 | 0,97 | 0,34 | 0,94 | 0,93 | 0,93 | 0,94 |
| Коефіцієнт виконання на тестуванні | 0,92 | 0,69 | 0,95 | 0,96 | 0,33 | 0,95 | 0,33 | 0,94 | 0,32 | 0,9 | 0,92 |

Як ми можемо спостерігати із чотирьох наведених таблиць, модель поводить себе майже однаково при зміні вхідних параметрів. Так, якщо зменшити мінімальні затримки надходження, то значення завантаженості пристроїв та їх черг почнуть рости, а коефіцієнт опрацьованих задач падати. І навпаки, якщо ми збільшимо максимальні затримки надходження, то значення черг і завантаженості будуть малими, а коефіцієнт опрацьованих задач – високим. Так само, якщо ми зменшимо затримки на кожному СМО, то черга і завантаженість цього девайсу будуть малими, а коефіцієнт опрацьованих задач – високим. І навпаки, при збільшені – черга і завантаженість будуть високими, а коефіцієнт буде малим. Гіпотези до зміни статистики підтвердились, а отже верифікація алгоритму імітації була успішною.

Варто також наголосити, що ми отримали такі результати під час прогону системи за 10000 часових одиниць. Кожне значення отримане середнім значенням за 20 прогонів.

# **РОЗДІЛ 5. ПРОВЕДЕННЯ ЕКСПЕРИМЕНТІВ НА МОДЕЛІ**

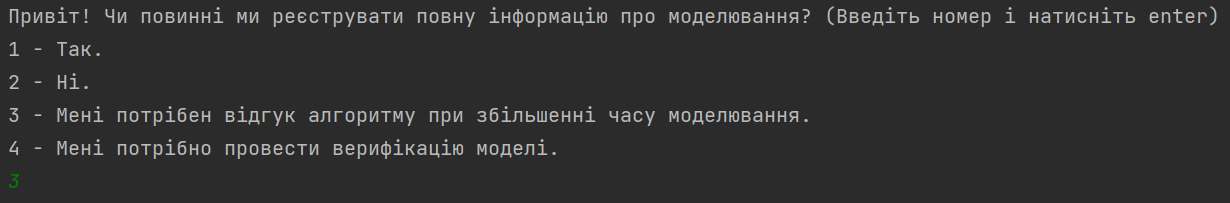
У якості дослідження функціонування моделі ми будемо використовувати факторний експеримент щодо впливу типу пріоритетності черги на загальне значення середньої завантаженості системи та загальне середнє значення черг.

## **5.1 Тактичне планування факторного експерименту**

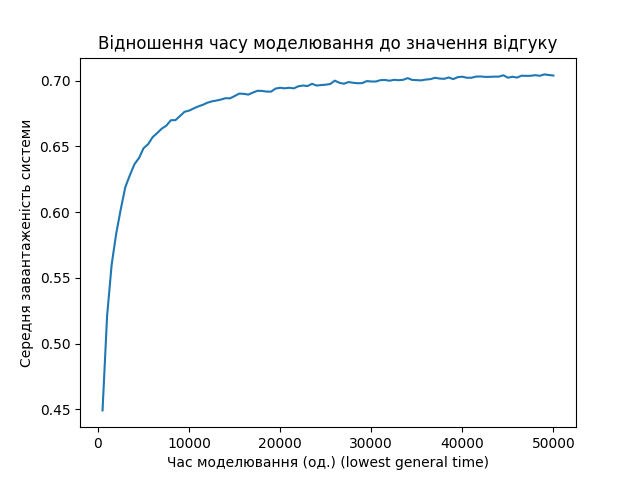
Оскільки про закон розподілу відгуку моделі нічого не відомо, кількість прогонів була визначена як 2000. Це найбільш оптимальне значення за нерівністю Чебишева для отримання точних та очікуваних результатів моделювання.

Визначимо тривалість одного прогону. Щоб це зробити ми для кожного типу пріоритетності черги зробимо декілька прогонів із різним значенням часу моделювання та будемо дивитися, коли графік відгуків почне стабілізуватися та приймати рівний горизонтальний вид. Почнемо із 500 часових одиниць і закінчимо 50000 часовими одиницями. При цьому кожний прогін буде збільшуватись на 500 одиниць.

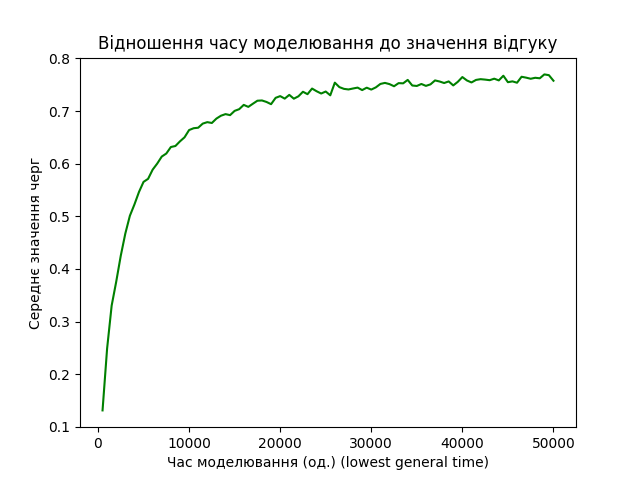
Для того, щоб відгук алгоритму при збільшенні часу моделювання, ми маємо при запуску програми вибрати 3 пункт меню:



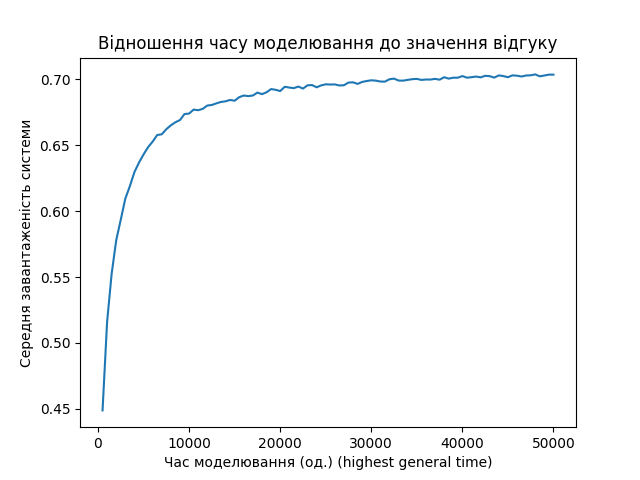
*Рисунок 5.1* – вибір третього пункту із меню програми



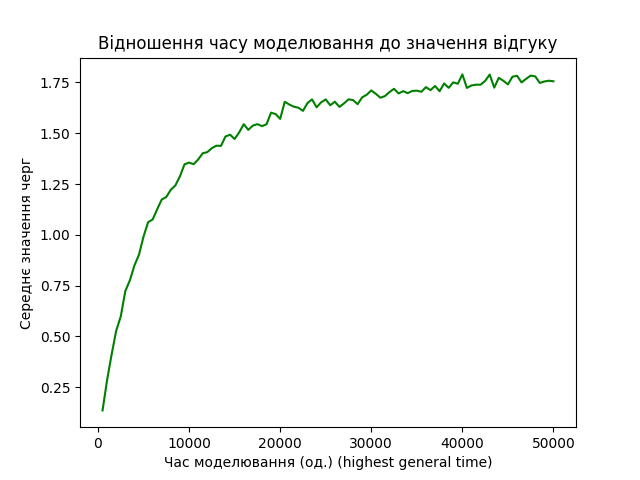
*Рисунок 5.2* – відношення часу моделювання до середньої завантаженості системи (найменше загального часу обробки)



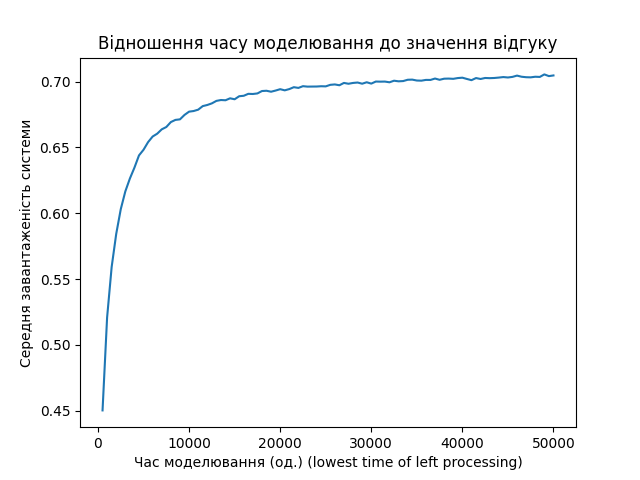
*Рисунок 5.3* – відношення часу моделювання до середнього значення черг (найменше загального часу обробки)



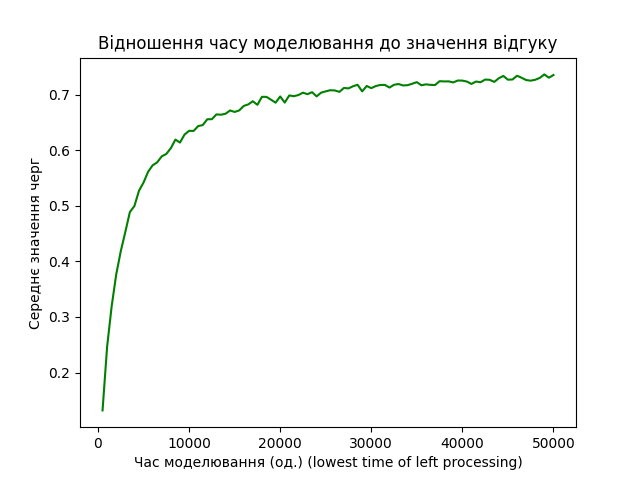
*Рисунок 5.4* – відношення часу моделювання до середньої завантаженості системи (найбільше загального часу обробки)



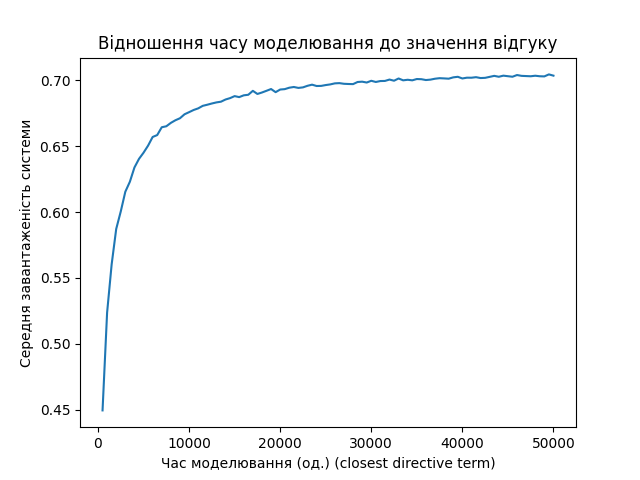
*Рисунок 5.5* – відношення часу моделювання до середнього значення черг (найбільше загального часу обробки)



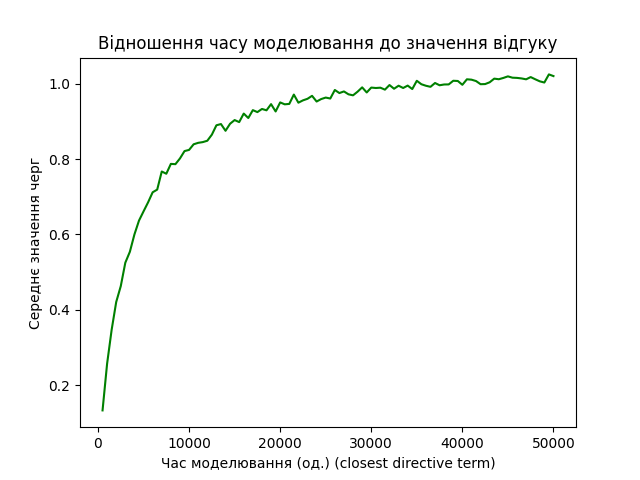
*Рисунок 5.6* – відношення часу моделювання до середньої завантаженості системи (найменший час обробки, що залишився)



*Рисунок 5.7* – відношення часу моделювання до середнього значення черг (найменший час обробки, що залишився)



*Рисунок 5.8* – відношення часу моделювання до середньої завантаженості системи (найближчий директивний термін)



*Рисунок 5.9* – відношення часу моделювання до середнього значення черг (найближчий директивний термін)

Як ми можемо побачити із наведених графіків, майже кожен із них починає приймати найбільш рівну форму починаючи із значення 40000. Тому для забезпечення найбільш точних результатів, за час моделювання буде взяте значення 100000 часових одиниць, щоб точно гарантувати, що система працює у перехідному режимі.

## **5.2 Стратегічне планування факторного експерименту**

Типи пріоритетності черги являються якісними факторами. При дослідженні впливу якісних факторів метою експерименту є якісна оцінка впливу факторів, тобто відповідь на запитання «впливає чи не впливає значення фактору на відгук моделі». Для якісної оцінки впливу факторів використовують ***дисперсійний*** аналіз.

Для проведення експериментів будуть використані наступні формули та правила:

, (5.1)

, (5.2)

, (5.3)

Де p – кількість прогонів у кожному експерименті, – i = 1,..p, j = 1,2,3,4 – результат j-ого експерименту в i-тому спостереженні, – середні значення j-ого експерименту, – середнє значення всіх експериментів. Кількість повторень прогонів, як уже було зазначено, становить 2000 разів.

Критичне значення критерію Фішера знаходять на основі рівня значимості , ступенів вільності числівника та ступенів вільності знаменника . Для всіх наших експериментів . Якщо , то середні , та розрізняються значимо і вплив фактору є значним. В протилежному випадку, вплив фактору визнається не значним, а різниця у середніх значеннях пояснюється випадковими причинами.

## **5.3 Експериментування**

Так як модель працює у перехідному режимі, то експериментування відбувається за допомогою методу повторення. У таблиці 5.1 наведено отримані вихідні значення загального середньої завантаженості системи.

**Таблиця 5.1 – загальні значення середньої завантаженості системи**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | *Тип пріоритету для замовлень з черги* | | | |
| Найменший  загальний час | Найбільший  загальний час | Найменший час  обробки, що  залишився | Найближчий  директивний термін |
| Серед. | 0,708150908 | 0,707575895 | 0,706454589 | 0,707109789 |

Серед отриманих значень альтернатив середньої завантаженості, явного фаворита ми не спостерігаємо. Якщо середнє значення і відрізняється, то дуже не набагато. Найкраще себе показав тип із найменшим загальним часом. Проведемо дисперсійний аналіз впливу факторів із використанням формул 5.1, 5.2 та 5.3. Так, , , .

Отже, , а тому вплив фактору визнаємо значним. Це значить, що тип вибору пріоритету буде впливати на завантаженість системи.

У таблиці 5.2 наведено отримані вихідні значення середніх черг системи.

**Таблиця 5.2 – вихідні значення середніх черг системи**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | *Тип пріоритету для замовлень з черги* | | | |
| Найменший  загальний час | Найбільший  загальний час | Найменший час  обробки, що  залишився | Найближчий  директивний термін |
| Серед. | 0,778959 | 1,82017 | 0,739693 | 1,040202 |

Серед отриманих значень, із незначним відривом, найкраще себе показав тип пріоритетності із найменшим часом обробки, що залишився. На другому місці – найменший загальний час. Проведемо дисперсійний аналіз впливу факторів із використанням формул 5.1, 5.2 та 5.3. Так, , , .

Отже, , а тому вплив фактору визнаємо значним. Це значить, що тип вибору пріоритету буде впливати на середні значення черг.

У таблиці 5.3 наведено отримані вихідні значення коефіцієнту виконаних завдань за час моделювання.

**Таблиця 5.3 – значення коефіцієнту виконаних завдань**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | *Тип пріоритету для замовлень з черги* | | | |
| Найменший  загальний час | Найбільший  загальний час | Найменший час  обробки, що  залишився | Найближчий  директивний термін |
| Серед. | 0,993049581 | 0,988221848 | 0,993267648 | 0,991844994 |

Серед отриманих значень, із дуже мінімальним відривом, найкраще себе показав тип пріоритетності із найменшим часом обробки, що залишився. На другому місці знову найменший загальний час. Проведемо дисперсійний аналіз впливу факторів із використанням формул 5.1, 5.2 та 5.3. Так, , , .

Отже, , а тому вплив фактору визнаємо значним. Це значить, що тип вибору пріоритету буде впливати на значення коефіцієнту виконаних завдань.

# **ВИСНОВКИ**

Відповідно до проведених експериментів, ми можемо сказати, що найкращими типом вибору задач із черг за двома із трьох критеріїв оцінки буде найменший час обробки, що залишився. Варто зазначити, що від нього із незначним відривом відстав тип вибору найменший загальний час. При цьому, якщо оцінювати отримані їхні середні значення по критеріям, то різниця отримана при тестуванні середніх значень черг – 0.039266. Це значення є достатньо малим, щоб гарантувати точно, який тип із цих двох буде самим найкращим. А при тестуванні коефіцієнту виконаних завдань різниця між ними становить взагалі 0,000218067, що також є дуже малим значенням. Така сама ситуація і при середній завантаженості системи, де виграшним вийшов найменший загальний час – різниця 0,001696319

Поставлене завдання ми вирішили за допомогою розробленого програмного забезпечення та розробленої імітаційної моделі. Для власників підприємств, що створюють програмне керування для металорізальних верстатів, дане програмне забезпечення має бути корисним та зручним у використанні. Простота зміни вхідних даних та результати моделювання надають можливість дослідити роботу системи та налагодити процеси виробництва. Повний код алгоритмів та програми можна знайти у додатках.

# **СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ**

1. Стеценко І. В. Моделювання систем / Інна Вячеславівна Стеценко.. – 407 с.
2. Guido van Rossum. Python Reference Manual / Guido van Rossum., 2006.
3. Pandas [Електронний ресурс] – Режим доступу до ресурсу: https://pandas.pydata.org/.
4. NumPy [Електронний ресурс] – Режим доступу до ресурсу: https://numpy.org/.
5. Matplotlib: Visualization with Python [Електронний ресурс] – Режим доступу до ресурсу: https://matplotlib.org/.

# **ДОДАТКИ**

## **Додаток А. Лістинг коду файлу «create.py»**

from element import \*

class Create(Element):

def \_\_init\_\_(self, name, distribution, interval1, elem1\_delay, elem2\_delay, elem3\_delay, interval2):

super().\_\_init\_\_(name\_element=name, distribution=distribution, delay=interval1[0])

self.delay\_dev = interval1[1]

self.delay1 = elem1\_delay

self.delay2 = elem2\_delay

self.delay3 = elem3\_delay

self.elem1\_delay = 0.0

self.elem2\_delay = 0.0

self.elem3\_delay = 0.0

self.add\_time\_min = interval2[0]

self.add\_time\_max = interval2[1]

self.general\_time = 0.0

self.technical\_time = 0.0

self.add\_time = 0.0

self.properties = []

def out\_act(self):

super().out\_act()

self.elem1\_delay = exp(self.delay1)

self.elem2\_delay = exp(self.delay2)

self.elem3\_delay = exp(self.delay3)

self.add\_time = uniform(self.add\_time\_min, self.add\_time\_max)

self.general\_time = self.elem1\_delay + self.elem2\_delay + self.elem3\_delay

self.technical\_time = self.general\_time + self.add\_time

delay = uniform(self.delay\_mean, self.delay\_dev)

self.t\_next[0] = self.t\_curr + delay

directive\_term = self.t\_curr + self.technical\_time + delay

self.properties = [

self.elem1\_delay,

self.elem2\_delay,

self.elem3\_delay,

self.add\_time,

self.general\_time,

directive\_term]

self.next\_element[0].in\_act(self.properties)

def print\_event\_info(self, flag):

if flag is True:

print(f"Programming delay: {self.properties[0]}")

print(f"Write on carrier delay: {self.properties[1]}")

print(f"Testing delay: {self.properties[2]}")

print(f"Technical delay: {self.technical\_time}")

print(f"Directive term: {self.properties[5]}")

print()

## **Додаток Б. Лістинг коду файлу «element.py»**

from fun\_rand import \*

class Element:

next\_id = 0

def \_\_init\_\_(self, name\_element=None, delay=None, distribution=None):

self.next\_element = None

self.t\_next = [0]

self.t\_curr = self.t\_next

self.state = [0]

self.id\_element = Element.next\_id

Element.next\_id += 1

self.name = name\_element

self.quantity = int()

self.delay\_dev = float()

self.delay\_mean = float()

self.distribution = str()

self.probability = [1]

self.delay\_mean = delay

self.distribution = distribution

def get\_delay(self):

if self.distribution == 'exp':

delay = exp(self.delay\_mean)

elif self.distribution == 'uniform':

delay = uniform(self.delay\_mean, self.delay\_dev)

else:

delay = self.delay\_mean

return delay

def in\_act(self, priority):

pass

def out\_act(self):

self.quantity += 1

def statistics(self, delta):

pass

def result(self):

print(f'{self.name} quantity = {str(self.quantity)} state = {self.state}')

def print\_info(self):

print(f'{self.name} state = {self.state} quantity = {self.quantity} t\_next = {self.t\_next}')

## **Додаток В. Лістинг коду файлу «fun\_rand.py»**

import random

import numpy as np

def exp(time\_mean):

a = 0.0

while a == 0:

a = random.random()

a = -time\_mean \* np.log(a)

return a

def uniform(time\_min, time\_max):

a = 0.0

while a == 0:

a = random.random()

a = time\_min + a \* (time\_max - time\_min)

return a

## **Додаток Г. Лістинг коду файлу «process.py»**

from element import \*

import numpy as np

class Process(Element):

def \_\_init\_\_(self, name, distribution, channels=1, task\_priority=None):

super().\_\_init\_\_(name\_element=name, distribution=distribution, delay=None)

self.queue = []

self.general\_time\_queue = []

self.directive\_term\_queue = []

self.left\_time\_queue = []

self.queue\_count = 0

self.max\_queue = np.inf

self.mean\_queue, self.mean\_load = 0.0, 0.0

self.failure = 0

self.directive\_fail = 0

self.channel = channels

self.t\_next = [np.inf] \* self.channel

self.state = [0] \* self.channel

self.current\_task = []

self.probability = [1]

self.task\_priority = task\_priority

self.count\_max\_queue = 0

def in\_act(self, task\_properties):

free\_route = self.get\_free\_channels()

if len(free\_route) > 0:

for i in free\_route:

self.state[i] = 1

self.current\_task = task\_properties

self.delay\_mean = self.current\_task[self.id\_element-1]

self.t\_next[i] = self.t\_curr + self.delay\_mean

break

else:

if len(self.queue) < self.max\_queue:

self.queue.append(task\_properties)

self.general\_time\_queue.append(task\_properties[4])

self.directive\_term\_queue.append(task\_properties[5])

left\_time = 0.0

for i in range(self.id\_element-1, 3):

left\_time += task\_properties[i]

self.left\_time\_queue.append(left\_time)

self.queue\_count += 1

else:

self.failure += 1

def out\_act(self):

current\_channel = self.get\_current\_channel()

for i in current\_channel:

super().out\_act()

self.t\_next[i] = np.inf

self.state[i] = 0

if self.next\_element is not None:

next\_el = np.random.choice(a=self.next\_element, p=self.probability)

next\_el.in\_act(self.current\_task)

else:

if self.t\_curr > self.current\_task[5]:

self.directive\_fail += 1

self.current\_task = []

if len(self.queue) > 0:

if self.task\_priority == "lowest general time":

task\_index = self.general\_time\_queue.index(min(self.general\_time\_queue))

elif self.task\_priority == "highest general time":

task\_index = self.general\_time\_queue.index(max(self.general\_time\_queue))

elif self.task\_priority == "lowest time of left processing":

task\_index = self.left\_time\_queue.index(min(self.left\_time\_queue))

elif self.task\_priority == "closest directive term":

task\_index = self.directive\_term\_queue.index(min(self.directive\_term\_queue))

else:

task\_index = -1

self.queue\_count -= 1

self.state[i] = 1

self.current\_task = self.queue.pop(task\_index)

self.general\_time\_queue.pop(task\_index)

self.directive\_term\_queue.pop(task\_index)

self.left\_time\_queue.pop(task\_index)

self.delay\_mean = self.current\_task[self.id\_element - 1]

self.t\_next[i] = self.t\_curr + self.delay\_mean

def get\_free\_channels(self):

free\_channels = []

for i in range(self.channel):

if self.state[i] == 0:

free\_channels.append(i)

return free\_channels

def get\_current\_channel(self):

current\_channels = []

for i in range(self.channel):

if self.t\_next[i] == self.t\_curr:

current\_channels.append(i)

return current\_channels

def print\_info(self):

super().print\_info()

print(f"Current count of awaiting tasks: {self.queue\_count}")

print(f"General time processing queue: {self.general\_time\_queue}")

print(f"Directive terms queue: {self.directive\_term\_queue}")

print(f"Task ending processing time queue: {self.left\_time\_queue}")

print()

def statistics(self, delta):

self.mean\_queue += len(self.queue) \* delta

if self.queue\_count > self.count\_max\_queue:

self.count\_max\_queue = self.queue\_count

for i in range(self.channel):

self.mean\_load += self.state[i] \* delta

## **Додаток Д. Лістинг коду файлу «model.py»**

from process import \*

from create import \*

import warnings

import pandas as pd

warnings.simplefilter(action='ignore', category=FutureWarning)

column\_names = ['Average queue', 'Average load', 'Average task rate']

class Model:

general\_mean\_load = 0

general\_mean\_queue = 0

Result\_table = pd.DataFrame(columns=column\_names)

def \_\_init\_\_(self, elements\_list):

self.element\_list = elements\_list

self.t\_next = 0

self.event = 0

self.t\_curr = 0

self.stable = []

self.mean\_length\_of\_queue\_list = list()

self.mean\_load\_list = list()

self.rate\_of\_completed\_tasks\_list = list()

self.directive\_fail\_list = list()

self.general\_mean\_load, self.general\_mean\_queue = 0.0, 0.0

def simulate(self, time, flag):

while self.t\_curr < time:

self.t\_next = np.inf

for element in self.element\_list:

t\_next\_val = np.min(element.t\_next)

if t\_next\_val < self.t\_next:

self.t\_next = t\_next\_val

self.event = element.id\_element

if flag is True:

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print(f"Event name: {self.element\_list[self.event].name}, time = {self.t\_next}")

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

for element in self.element\_list:

element.statistics(self.t\_next - self.t\_curr)

self.t\_curr = self.t\_next

for element in self.element\_list:

element.t\_curr = self.t\_curr

if len(self.element\_list) > self.event:

self.element\_list[self.event].out\_act()

for element in self.element\_list:

if self.t\_curr in element.t\_next:

element.out\_act()

if len(self.element\_list) > self.event and self.element\_list[self.event].id\_element == 0:

self.element\_list[self.event].print\_event\_info(flag)

self.print\_info(flag)

self.print\_result(flag)

self.experiments()

def print\_info(self, flag):

if flag is True:

for element in self.element\_list:

element.print\_info()

def print\_result(self, flag):

income\_task = 0

if flag is True:

print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\_RESULTS\_\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

for element in self.element\_list:

element.result()

if isinstance(element, Create):

income\_task = element.quantity

elif isinstance(element, Process):

e = element

self.general\_mean\_load += e.mean\_load / self.t\_curr

self.general\_mean\_queue += e.mean\_queue / self.t\_curr

print(f"Mean length of queue: {e.mean\_queue / self.t\_curr}")

self.mean\_length\_of\_queue\_list.append(e.mean\_queue / self.t\_curr)

print(f"Max queue: {e.count\_max\_queue}")

print(f"Mean load: {e.mean\_load / self.t\_next}")

self.mean\_load\_list.append(e.mean\_load / self.t\_next)

print(f"Tasks that failed directive term: {e.directive\_fail}")

self.directive\_fail\_list.append(e.directive\_fail)

print(f"Rate of completed tasks: {e.quantity / income\_task}")

self.rate\_of\_completed\_tasks\_list.append(e.quantity / income\_task)

print()

print()

def experiments(self):

avg\_mean\_load, avg\_mean\_queue = 0.0, 0.0

income, done = 0, 0

for e in self.element\_list:

if isinstance(e, Create):

income = e.quantity

elif isinstance(e, Process):

avg\_mean\_load += e.mean\_load / self.t\_curr

avg\_mean\_queue += e.mean\_queue / self.t\_curr

if e.id\_element == 3:

done = e.quantity

Model.general\_mean\_load = avg\_mean\_load / 3

Model.general\_mean\_queue = avg\_mean\_queue / 3

Model.rate\_task = done / income

result = pd.DataFrame([[Model.general\_mean\_queue, Model.general\_mean\_load, Model.rate\_task]],

columns=column\_names)

Model.Result\_table = Model.Result\_table.append(result, ignore\_index=True)

## **Додаток Д. Лістинг коду файлу «main.py»**

from model import \*

from process import \*

import matplotlib.pyplot as plt

def sim\_response(modeling\_time, test\_times, queue\_type):

times = 0

while times < test\_times:

start\_model(False, modeling\_time, queue\_type, None)

times += 1

def sim\_verify(modeling\_time, test\_times):

n\_params = 11

order\_income\_min\_value = [100, 0, 100, 100, 100, 100, 100, 100, 100, 100, 100]

order\_income\_max\_value = [200, 200, 1000, 200, 200, 200, 200, 200, 200, 200, 200]

programming = [120, 120, 120, 0, 500, 120, 120, 120, 120, 120, 120]

write = [110, 110, 110, 110, 110, 0, 500, 110, 110, 110, 110]

testing = [90, 90, 90, 90, 90, 90, 90, 0, 500, 90, 90]

additional\_tech\_time\_min\_value = [20, 20, 20, 20, 20, 20, 20, 20, 20, 0, 20]

additional\_tech\_time\_max\_value = [60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 1000]

queue\_types = ['lowest general time',

'highest general time',

'lowest time of left processing',

'closest directive term']

for queue\_type in queue\_types:

df = pd.DataFrame()

rows = []

for i in range(n\_params):

times = 0

mean\_length\_of\_queue\_programming = list()

mean\_length\_of\_queue\_write = list()

mean\_length\_of\_queue\_testing = list()

mean\_load\_programming = list()

mean\_load\_write = list()

mean\_load\_testing = list()

rate\_of\_completed\_tasks\_programming = list()

rate\_of\_completed\_tasks\_write = list()

rate\_of\_completed\_tasks\_testing = list()

directive\_fail\_programming = list()

directive\_fail\_write = list()

directive\_fail\_testing = list()

while times < test\_times:

create = Create('INCOMING TASK',

'uniform',

[order\_income\_min\_value[i], order\_income\_max\_value[i]],

programming[i],

write[i],

testing[i],

[additional\_tech\_time\_min\_value[i], additional\_tech\_time\_max\_value[i]])

model = start\_model(False, modeling\_time, queue\_type, create)

model.print\_result(True)

mean\_length\_of\_queue\_programming.append(model.mean\_length\_of\_queue\_list[0])

mean\_length\_of\_queue\_write.append(model.mean\_length\_of\_queue\_list[1])

mean\_length\_of\_queue\_testing.append(model.mean\_length\_of\_queue\_list[2])

mean\_load\_programming.append(model.mean\_load\_list[0])

mean\_load\_write.append(model.mean\_load\_list[1])

mean\_load\_testing.append(model.mean\_load\_list[2])

rate\_of\_completed\_tasks\_programming.append(model.rate\_of\_completed\_tasks\_list[0])

rate\_of\_completed\_tasks\_write.append(model.rate\_of\_completed\_tasks\_list[1])

rate\_of\_completed\_tasks\_testing.append(model.rate\_of\_completed\_tasks\_list[2])

directive\_fail\_programming.append(model.directive\_fail\_list[0])

directive\_fail\_write.append(model.directive\_fail\_list[1])

directive\_fail\_testing.append(model.directive\_fail\_list[2])

times += 1

param = {'order\_income\_min\_value': order\_income\_min\_value[i],

'order\_income\_max\_value': order\_income\_max\_value[i],

'programming': programming[i],

'write': write[i],

'testing': testing[i],

'additional\_tech\_time\_min\_value': additional\_tech\_time\_min\_value[i],

'additional\_tech\_time\_max\_value': additional\_tech\_time\_max\_value[i],

'mean\_length\_of\_queue\_programming': sum(mean\_length\_of\_queue\_programming) / len(mean\_length\_of\_queue\_programming),

'mean\_length\_of\_queue\_write': sum(mean\_length\_of\_queue\_write) / len(mean\_length\_of\_queue\_write),

'mean\_length\_of\_queue\_testing': sum(mean\_length\_of\_queue\_testing) / len(mean\_length\_of\_queue\_testing),

'mean\_load\_programming': sum(mean\_load\_programming) / len(mean\_load\_programming),

'mean\_load\_writ': sum(mean\_load\_write) / len(mean\_load\_write),

'mean\_load\_testing': sum(mean\_load\_testing) / len(mean\_load\_testing),

'rate\_of\_completed\_tasks\_programming': sum(rate\_of\_completed\_tasks\_programming) / len(rate\_of\_completed\_tasks\_programming),

'rate\_of\_completed\_tasks\_write': sum(rate\_of\_completed\_tasks\_write) / len(rate\_of\_completed\_tasks\_write),

'rate\_of\_completed\_tasks\_testing': sum(rate\_of\_completed\_tasks\_testing) / len(rate\_of\_completed\_tasks\_testing),

'directive\_fail\_programming': sum(directive\_fail\_programming) / len(directive\_fail\_programming),

'directive\_fail\_write': sum(directive\_fail\_write) / len(directive\_fail\_write),

'directive\_fail\_testing': sum(directive\_fail\_testing) / len(directive\_fail\_testing)}

rows.append({\*\*param})

file\_name = f'{queue\_type}\_verification.xlsx'

df = df.append(rows)

df.to\_excel(file\_name)

print("Результати верификації записані у відповідні файли!!!")

def sim(flag, modeling\_time, test\_times, task\_priority):

times = 0

while times < test\_times:

start\_model(flag, modeling\_time, task\_priority, None)

times += 1

print(Model.Result\_table)

def start\_model(flag, modeling\_time, task\_priority, create\_val):

# lowest general time

# highest general time

# lowest time of left processing

# closest directive term

if create\_val is not None:

create = create\_val

else:

Element.next\_id = 0

create = Create('INCOMING TASK', 'uniform', [100.0, 200.0], 120.0, 110.0, 90.0, [20.0, 60.0])

process\_1 = Process(name='PROGRAMMING', distribution='exp', task\_priority=task\_priority)

process\_2 = Process(name='WRITE ON CARRIER', distribution='exp', task\_priority=task\_priority)

process\_3 = Process(name='TESTING', distribution='exp', task\_priority=task\_priority)

create.next\_element = [process\_1]

process\_1.next\_element = [process\_2]

process\_2.next\_element = [process\_3]

elements\_list = [create, process\_1, process\_2, process\_3]

model = Model(elements\_list)

model.simulate(modeling\_time, flag)

Element.next\_id = 0

return model

def select\_queuing\_type(flag, test\_times, modeling\_time):

print("Оберіть тип черговості обробки замовлень")

print("1 - Найменше загального часу обробки.")

print("2 - Найбільше загального часу обробки.")

print("3 - Найменший час обробки, що залишився.")

print("4 - Найближчий директивний термін.")

task = input()

if task == "1":

sim(flag, modeling\_time, test\_times, "lowest general time")

Model.Result\_table.to\_excel(r'lowest\_general\_time.xlsx', sheet\_name='1', index=False)

elif task == "2":

sim(flag, modeling\_time, test\_times, "highest general time")

Model.Result\_table.to\_excel(r'highest\_general\_time.xlsx', sheet\_name='1', index=False)

elif task == "3":

sim(flag, modeling\_time, test\_times, "lowest time of left processing")

Model.Result\_table.to\_excel(r'lowest\_time\_of\_left\_processing.xlsx', sheet\_name='1', index=False)

elif task == "4":

sim(flag, modeling\_time, test\_times, "closest directive term")

Model.Result\_table.to\_excel(r'closest\_directive\_term.xlsx', sheet\_name='1', index=False)

else:

print("Ви повинні вибрати відповідне завдання!")

main()

def main():

print("Привіт! Чи повинні ми реєструвати повну інформацію про моделювання? (Введіть номер і натисніть enter)")

print("1 - Так.")

print("2 - Ні.")

print("3 - Мені потрібен відгук алгоритму при збільшенні часу моделювання.")

print("4 - Мені потрібно провести верифікацію моделі.")

task = input()

# 100000

modeling\_time = 100000

test\_times = 2000

if task == "1":

select\_queuing\_type(True, test\_times, modeling\_time)

elif task == "2":

select\_queuing\_type(False, test\_times, modeling\_time)

elif task == "3":

queue\_types = ['lowest general time',

'highest general time',

'lowest time of left processing',

'closest directive term']

for queue\_type in queue\_types:

start\_time = 500

response\_list1 = list()

response\_list2 = list()

time\_list = list()

test = 1

while test < 101:

sim\_response(start\_time, test\_times, queue\_type)

response\_list1.append(Model.Result\_table['Average load'].mean())

response\_list2.append(Model.Result\_table['Average queue'].mean())

time\_list.append(start\_time)

Model.Result\_table = pd.DataFrame(columns=column\_names)

test += 1

start\_time += 500

plt.plot(time\_list, response\_list1)

plt.xlabel(f'Час моделювання (од.) ({queue\_type})')

plt.ylabel('Середня завантаженість системи')

plt.title('Відношення часу моделювання до значення відгуку')

plt.show()

plt.plot(time\_list, response\_list2, 'g')

plt.xlabel(f'Час моделювання (од.) ({queue\_type})')

plt.ylabel('Середнє значення черг')

plt.title('Відношення часу моделювання до значення відгуку')

plt.show()

print(queue\_type)

print(response\_list1)

print(response\_list2)

print(time\_list)

print(len(time\_list))

elif task == "4":

sim\_verify(modeling\_time, test\_times)

else:

print("Ви повинні вибрати відповідне завдання!")

main()

if \_\_name\_\_ == "\_\_main\_\_":

main()