Міністерство освіти і науки України

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Кафедра АСОІУ

Спеціальність 126 Інформаційні системи та технології

КУРСОВА РОБОТА

з дисципліни

"Моделювання систем"

на тему

"Задача про створення

програмного керування верстатами"

Виконав

студент групи ІС-73

Турко М. В.

№ зал. кн. ІС-7328

_____ _____

(підпис) (оцінка)

Прийняли

Проф. каф. АСОІУ, к.т.н.

СТЕЦЕНКО І. В.

_____

(підпис)

Стар. викл. каф. АСОІУ, к.т.н.

НОВІКОВА П. А.

_____

(підпис)

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

Кафедра автоматизованих систем обробки інформації та управління
Дисципліна «Моделювання систем»
Спеціальність Інформаційні управляючі системи та технології
Курс 4 Група ІС - 73 Семестр 7

ЗАВДАННЯ

на курсову роботу студента

Турко Микола Васильович
(прізвище, ім'я, по батькові)

1.Тема роботи: Задача про створення програмного керування верстатами

2. Термін здачі студентом закінченої роботи "24" грудня 2020 р.

3. Вихідні дані до проекту

Варіант № 18

З файлу варіантів завдань на курсову роботу, у таблиці 3.20 обирається третій варіант вихідних даних відповідно до номеру группи з потоку

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 "10" вересня 2020 р.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів курсового проекту (роботи)	Термін виконання етапів роботи	Примітка
1	Отримання завдання	22.09.2020	
2	Аналіз можливих методів вирішення поставленого завдання	28.09.2020	
3	Розробка концептуальної моделі	10.10.2020	
4	Перший контроль за процесом виконання курсового проекту (роботи), консультація у викладача	15.10.2020	
5	Опис імітаційної моделі	25.10.2020	
6	Опис програмної реалізації імітаційної моделі	05.11.2020	
7	Другий контроль за процесом виконання курсового проекту (роботи), консультація у викладача	20.11.2020	
8	Аналіз та оцінка результатів	07.12.2020	
9	Оформлення пояснювальної записки	12.12.2020	
10	Здача пояснювальної записки	24.12.2020	
11	Захист курсового проекту (роботи)	25.12.2020	

Студент_____ *Турко М. В.*
(підпис) (прізвище, ім'я, по батькові)
Керівник _____ *Новікова П. А.*
(підпис) (прізвище, ім'я, по батькові)

«22» вересня 2020 р.

РЕФЕРАТ

Курсова робота: 35 с., 2 Рисунок, 4 табл., 2 додатки, 5 джерел літератури.

Об'єкт дослідження – система розробки ПЗ для керування металорізальними верстатами.

Мета роботи – обґрунтування вибору замовлення з черг обслуговування.

Метод дослідження – імітаційне моделювання системи створення програмного керування для металорізальних верстатів.

Проведено дослідження із різнотипного вибору завдань з черги для обслуговування, розроблена програмне забезпечення для реалізації імітаційного моделювання створення та встановлення програмного керування на верстати. Результати моделювання використані для прийняття управлінських рішень щодо оптимального вибору замовлень з черг процесів.

Модель програми розроблена вперше.

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

ЗМІСТ

ПОСТАНОВКА ЗАВДАННЯ

Відділ з обслуговування ЕОМ готує носіїв з програмами для металорізальних верстатів з 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 ($T_1+T_2+T_3$) 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 витрачається найменше загального часу обробки;

2) спочатку виконуються ті замовлення, на виконання яких витрачається найбільше загального часу обробки;

3) спочатку виконуються ті замовлення, на виконання яких витрачається найменший час обробки, що залишився;

4) спочатку виконуються ті замовлення, які мають найближчий директивний термін.

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

ВСТУП

Задача про створення програмного керування для металорізальних верстатів являє собою актуальну диллему вибору завдання для обслуговування. Перед власниками підприємств виникає класична проблема упорядкування завдань для виконання аби поліпшити виробництво. У діапазоні задач моделювання, поставлене завдання класифікується задачею управління.

Завдяки розробленій програмі імітаційного моделювання такої системи, з’являється можливість дослідити різні підходи у вирішенні цієї задачі та обрати оптимальний спосіб обслуговування відносно бажаних критеріїв оцінювання.

1 АНАЛІЗ МОЖЛИВИХ МЕТОДІВ ТА ЗАСОБІВ ВИРІШЕННЯ ПОСТАВЛЕНОГО ЗАВДАННЯ

Вирішення поставленого завдання, з умови задачі, полягає в дослідженні впливу вибору завдань з черги на запропоновані критерії об’єкту. З вхідних даних зрозуміло: інтенсивність надходження завдань, часові затримки обробки процесів та послідовність виконання завдання. Таким чином, поставлена задача надає загальне представлення системи, по якому можна скласти її модель за загальною структурою (див. рисунок 1): X – множина вхідних змінних системи, Y – множина вихідних змінних системи, P – множина параметрів, F – алгоритм функціонування.

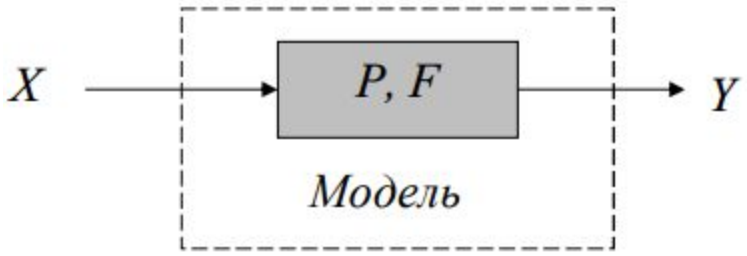

Рисунок 1.1 – загальна структура моделі

Так як з постановки задачі зрозуміло, що параметри моделі в цьому випадку пов’язані з реальними процесами і мають фізичну інтерпретацію. Тому систему створення програмного керування для металорізальних верстатів класифікується, як фізична модель.

1.1 Можливі методи вирішення

Визначившись, що для вирішення потрібно моделювати систему, розглядаються різні методи моделювання. Серед великої кількості методів моделювання, що існують, найпопулярнішими є такі: аналітичне моделювання, математичне моделювання, імітаційне моделювання.

Моделювання аналітичне, якщо представлення залежності F вихідних змінних Y від вхідних її змінних X має аналітичний вигляд, тобто представлений у вигляді відомих аналітичних функцій. В поставленому завданні неможливо описати залежність вихідних змінних від вхідних за допомогою аналітичних функцій, тому цей метод моделювання одразу відкидається.

Моделювання математичне, якщо відомий алгоритм відшукання точного розв'язку задачі, але сам розв'язок не може бути записаний в аналітичній формі. Модель системи поставленої задачі являється стохастичною, тому неможливо гарантувати точний розв'язок. Цей метод теж відкидається.

Моделювання імітаційне, якщо 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. З розглянутих методів, сама імітаційне моделювання найкраще підходить для дослідження поставленої задачі.

1.2 Засоби вирішення поставленої задачі

Визначивши найкращий метод моделювання (див. підрозділ 1.1) обирається засіб для його реалізації. Імітаційне моделювання системи передбачає, що процес функціонування системи відтворюється за допомогою алгоритму, який реалізується за допомогою комп'ютера.

Розрізняють два типи програм, що реалізуються імітаційне моделювання:

- написанні на спеціалізованих мовах імітаційного моделювання (SIMULA, GPSS і т.д.);

- написані на універсальних мовах програмування (C++, Java, Python);

Головною відмінністю між цими програмами полягає у гнучкості программ – хоч використання процесно-орієнтованих мов моделювання має перевагу у простоті складання, але набір операторів обмежений. Можливості створення імітаційної модель сильно обмежені вибором пропонованих операторів. Користування універсальними мовами програмування надає необмежені можливості при побудові імітаційного алгоритму.

Реалізовуючи програму для імітаційного моделювання поставленої задачі необхідно реалізовувати наступні методи:

- засоби генерації випадкових чисел і змінних;
- організацію збирання статистичних даних про роботу моделі;
- можливість програмного забезпечення відображати стан моделі в моменти часу;
- можливість верифікувати алгоритм імітації;

У рамках курсової роботи, програма імітаційного моделювання допускається у вигляді статистичних пакетів.

2 РОЗРОБКА КОНЦЕПТУАЛЬНОЇ МОДЕЛІ

Опис системи разом із вказанням цілі та задачі дослідження складає сутність концептуальної моделі системи. Назва „концептуальна” походить від латинського слова *conceptio*, що означає „сприйняття”.

Ціль моделювання – визначити оптимальних вибір замовлень до виконання з черг обслуговувань.

Структурна схема моделі – об’єкт моделювання являє собою систему створення програмного керування для металорізальних верстатів. Замовлення надходять з заданою інтенсивність від відділу креслень. Після надходження, завдання розглядають всі відділи обслуговувань, визначаючи часові затримки

для виконання завдання. Проміжок часу для цих підрахунків визначається як додатковий. Коли всі відділи розрахували часові затримки, то для завдання визначається директивний термін – момент у часі, до якого завдання повинно бути виконано, це сумарний час виконання всіх процесів обслуговування, часу надходження замовлення та додаткового часу. Після визначення директивного терміну, завдання надходить до команди програмістів, що пишуть програмне керування за час визначений під час обговорення відділів. Після того, як була розроблена програма, вона записується на носій. Ця дія теж виконується за розрахований, на попередніх етапах, час. І в кінці шляху системи, програма тестується на верстатах. Якщо завдання надійшло до процесу, коли відділ вже зайнятий обслуговуванням, то замовлення чекає у черзі.

Вхідні змінні:

- ➔ інтенсивність надходження замовлень з відділу креслень;
- ➔ часові затримки оброблень замовлень у різних процесах;
- ➔ послідовність повного процесу створення та встановлення програми на верстат;
- ➔ час моделювання;
- ➔ пріоритет вибору завдання з черги;

Вихідні змінні:

- ➔ кількість замовлень, що надійшли до системи;
- ➔ кількість оброблених замовлень у кожному відділі;
- ➔ коефіцієнт відношення між замовленнями, що надійшли до системи за час моделювання та загальною кількістю оброблених;
- ➔ максимальна кількість замовлень, що очікували в чергах процесів за час моделювання;
- ➔ завантаженість відділів за час моделювання;

➔ середня кількість замовлень в чергах для кожного відділу;

➔ Загальне середнє значення замовлень, що очікували в чергах за час моделювання;

➔ Загальна середня завантаженість відділів за час моделювання;

Параметри моделі:

➔ пріоритету вибору завдання з черги;

Формули для вихідних змінних:

Для підрахунку коефіцієнта оброблених замовлень за час моделювання використовується формула (2.1):

$$K_{оброб} = \frac{N_{оброб}}{N_{надійшли}}, \qquad (2.1)$$

де *Nоброб* – це загальна кількість оброблених замовлень за час моделювання; *Nнадійшли* – загальна кількість замовлень, що надійшли до системи для обслуговування.

Для отримання середньої завантаженості відділів за час моделювання використовується формула (2.2):

$$R_{сер} = \frac{\sum_i R_i \cdot \Delta t_i}{T_{мод}}, \qquad (2.2)$$

де *Ri* – спостережуване значення зайнятості процесу; *Δti* – проміжок часу, протягом якого спостерігалось значення завантаженості; *Tмод* – час моделювання.

Для отримання середнього значення черги за час моделювання використовується формула (2.3):

$$L_{сер} = \frac{\sum_i L_i \cdot \Delta t_i}{T_{мод}}, \qquad (2.3)$$

де L_i – спостережуване значення довжини черги; Δt_i – проміжок часу, протягом якого спостерігалось значення довжини черги; $T_{мод}$ – час моделювання.

Функціональна залежність – зміна пріоритетів вибору завдання з черги впливає на такі вихідні змінні: кількість оброблених замовлень у кожному відділі, максимальна кількість замовлень, що очікували в чергах процесів за час моделювання, завантаженість відділів за час моделювання, середня кількість замовлень в чергах для кожного відділу.

Приклад:

Є три замовлення, що очікують у черзі на виконання в процесі обслуговування. Час загального виконання робіт: №1 - 10 од. часу, №2 - 6 од. часу, №4 - 4 од. часу. Час моделювання – 10 од.

На рисунку 2.1 зображено взяття замовлення до виконання відповідно до пріоритету найбільшого загального часу.

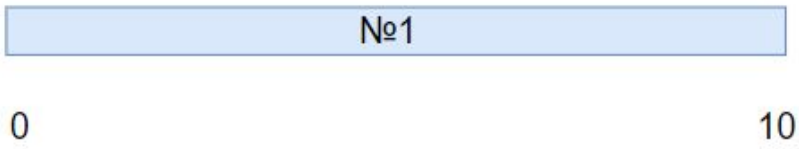

Рисунок 2.1 – діаграма Ганта для виконання завдання за пріоритетністю найбільшого загального часу

У випадку такої пріоритетності середня довжина черги (див. формулу 2.3) залишиться рівна 2 замовленням, а за встановлений проміж модельованого часу було оброблено лише одне завдання.

На рисунку 2.2 зображено взяття замовлення до виконання відповідно до пріоритету найменшого загального часу.

Рисунок 2.2 – діаграма Ганта для виконання завдання за пріоритетністю найменшого загального часу

У випадку такої пріоритетності середня довжина черги рівна 1.4 замовленням, а за встановлений проміж модельованого часу було оброблено два завдання. Таким чином доведено вплив параметру на вихідні змінні.

Цільова функція системи – щодо управління компанією, яка створює програмне керування для металорізальних верстатів, бажаним результатом є максимальна середня завантаженість відділів при мінімальній середній кількості замовлень в чергах.

Обмеження – з вхідних змінних можна варіювати лише пріоритетність вибору замовлення з черги. Для різних процесів обслуговування можна вказувати різні пріоритети вибору завдання з черги, але лише один варіант для одного процесу.

Схема концептуальної моделі

Схему концептуальної моделі відтворено за вербальним описом на рисунку 2.4.

Рисунок 2.3 – схема концептуальної моделі

3 РОЗРОБКА ІМІТАЦІЙНОЇ МОДЕЛІ ТА ОПИС ЇЇ ФУНКЦІОНУВАННЯ

Сформувавши концептуальну модель(див. роз. 2), необхідно визначити теоретичною базу для побудови моделі. Найпопулярнішими способами формалізувати систему є мережі масового обслуговування та мережі Петрі. Мережі Петрі надають перевагу у випадках, коли потрібно формалізувати систему, в якій використовуються спільні ресурси для багатьох процесів обслуговування та пристосованість до представлення виконання паралельних процесів. Відповідно до сформованої концептуальної моделі поставленої задачі, формалізація мережею масового обслуговування достатньо для імітаційного моделювання системи. На рисунку 3.1 зображено елементи формалізації мережі масового обслуговування.

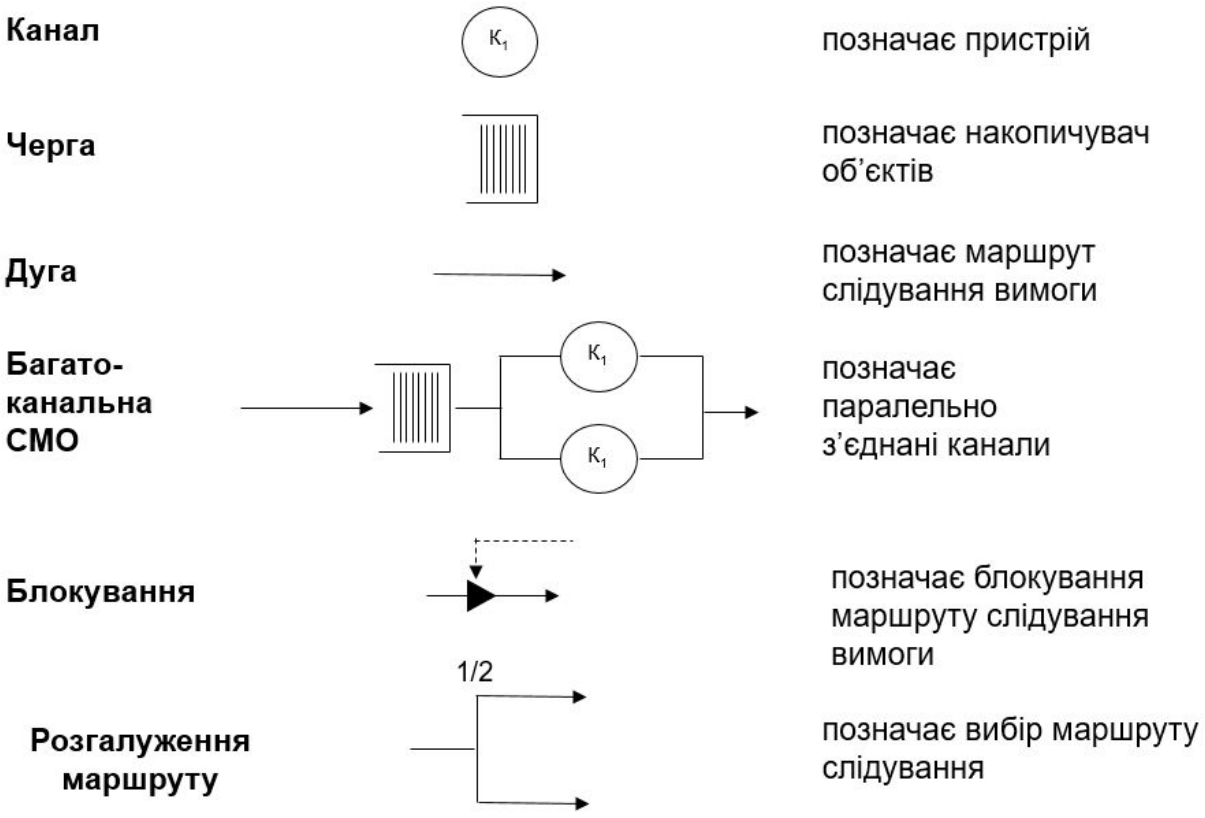

Рисунок 3.1 – елементи формалізації мережі масового обслуговування

3.1 Опис імітаційної моделі

Використовуючи елементи мереж масового обслуговування (див. Рисунок3.1) на рисунку 3.2 зображено формалізовану мережу поставленого завдання.

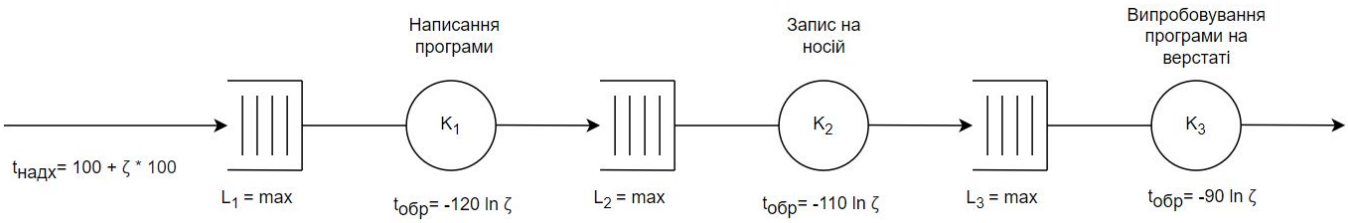

Рисунок 3.2 – мережа масового обслуговування поставленого завдання

Мережа масового обслуговування складається з наступних елементів:

- Надходження замовлень від відділу креслень;
- три пристрої, що обслуговують замовлення з різними часовими затримками;
- необмежені черги перед кожним пристроєм;

3.2 Опис програмної реалізації імітаційної моделі

Програмна реалізація імітаційного моделювання написана на універсальній мові програмування Python. Структура файлі ПЗ зображена на рисунку 3.2.1.

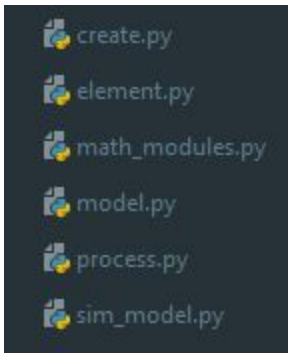

Рисунок 3.2.1

У таблиці 3.1 пояснюється сутність файлу до проекту.

Таблиця 3.1

Назва файлу	Пояснення
element.py	Містить в собі загальні характеристики елементів мережі масового обслуговування.
create.py	Опис надходження замовлень.
process.py	Опис системи масового обслуговування.
model.py	Алгоритм імітації моделювання.
sim_model.py	Задання імітаційної моделі
Math_modules.py	Закони розподілу.

У розробці програмної реалізації було обрано об’єктно-орієнтований підхід. На рисунку 3.3 зображено зв’язок між сутностями ПЗ.

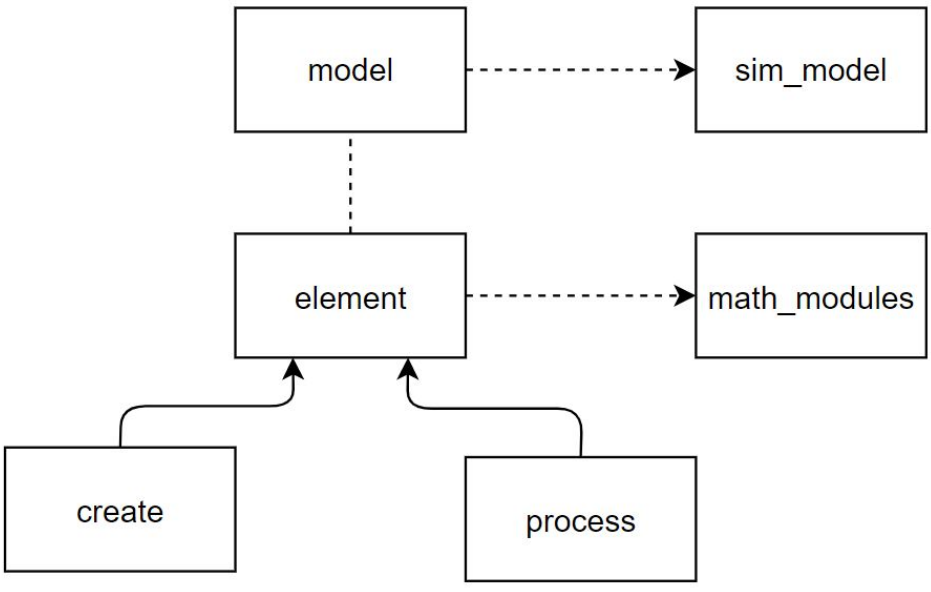

Рисунок 3.3 – поєднання сутностей файлів

Алгоритм імітації знаходиться у *model.py* тому це головний файл у ієрархії. Для моделювання використовуються моделі з файлу sim_model. Побудовані імітаційні моделі використовують елементи формалізації *create.py* та *process.py,* які наслідуються від файлу *element.py.* Так як програмне забезпечення написане на універсальній мові програмування, то просування стану моделі в часі відбувається за принципом орієнтованим на події.

В розробленій програмній реалізації імітаційного моделювання графічний спосіб відтворення відсутній, тому задання моделі відбувається напряму через написання програмного коду. На рисунку 3.4 зображено скіншот програмної реалізації створення імітаційної моделі поставленого завдання.

```
18      c1 = Create([100.0, 200.0], 120.0, 110.0, 90.0, [20.0, 60.0])
19
20      p1 = Process(task_priority="highest general time")
21      p2 = Process(task_priority="highest general time")
22      p3 = Process(task_priority="highest general time")
23
24      c1.name = 'INCOME TASK'
25      p1.name = 'PROGRAMMING PROCESS'
26      p2.name = 'WRITE ON HARDWARE'
27      p3.name = 'TESTING'
28
29      c1.next_element = [p1]
30      p1.next_element = [p2]
31      p2.next_element = [p3]
32
33      element_list = [c1, p1, p2, p3]
34      model = Model(element_list)
35      model.simulate(10000.0, True)
```

Рисунок 3.4 – мережа масового обслуговування поставленого завдання

Відповідно до наведеного скріншоту (див. Рисунок 3.4), створюється: об'єкт *c1* типу *Create.* У мережі масового обслуговування – елемент надходження. Цей об'єкт приймає параметри інтервалів часових затримок для обслуговування замовлення у різних відділах. Далі створення об'єктів *p1, p2, p3* типу *Process* – пристроїв обслуговування. Ці об'єкт приймають параметром тип пріоритету для замовлення у черзі. *Об'єкт*.name присвоює іменування для елементу. *Об'єкт*.next_element вказує маршрут слідування від одного елементу обслуговування до іншого. *Element_list* вказує на загальний список елементів системи. *Model(element_list)* сконструйована система приймає статус імітаційної моделі і моделюється за допомогою методу *model.simulate,* що приймає параметром час моделювання.

У таблиці 3.2 наводиться опис методів, що застосовується у класах.

Таблиця 3.2

Назва методу	**Опис**	**Вхідні змінні**	**Вихідні змінні**
Класс Element			
def init()	Ініціалізація об'єкту елементу для мереж масового обслуговування	name_element - ім'я елементу, delay - часова затримка	-
def get_delay()	Генерування часової затримки за законом розподілу	distribution - закон розподілу, інтервали часової затримки /математичне сподівання /середнє значення затримки та відхилення	Часова затримка отримана за вказаним законом розподілу

Продовження таблиці 3.1.2

Назва методу	Опис	Вхідні змінні	Вихідні змінні
Класс Element			
def out_act()	Вихід замовлення з елементу	-	Збільшення кількості оброблених замовлень
def print_res()	Вивід статистики для елементу за час моделювання	-	Статистика оброблених замовлень для елементу
def print_info()	Показ інформації про стан елементу на кожному кроці просування моделі	-	Інформація про ім’я елементу, стан зайнятості, кількість оброблених замовлень елементом
Класс Create			
def init()	Ініціалізація об’єкту надходження замовлень	interval1 - інтервали часової затримки для надходження, назва, smo1(2,3)_delay - математичне сподівання часової затримки СМО, interval2 - інтервал часової затримки часу визначення директивного терміну	-

Продовження таблиці 3.1.2

Назва методу	Опис	Вхідні змінні	Вихідні змінні
Класс Create			
def out_act()	Надходження замовлення до системи	-	Згенеровані характеристики часових затримок та директивного терміну завдання, збільшення кількості замовлень, що надійшли
def print_create_info()	Показ характеристик згенерованих часових затримок обробки замовлення у СМО	-	Виведення інформації про характеристику нового замовлення у системі
Класс Process			
def init()	Ініціалізація об’єкту системи масового обслуговування	taks_priority - тип пріоритетності замовлення для вибору з черги	-
def in_act()	Надходження нового замовлення до системи масового обслуговування	task_properties - характеристика замовлення з часовою затримкою для обслуговування	-
def out_act()	Завершення обслуговування	-	Збільшення кількості

	замовлення		замовлень, що були оброблені

Продовження таблиці 3.1.2

Назва методу	Опис	Вхідні змінні	Вихідні змінні
Класс Process			
def get_free_ channes()	Для багатоканальних СМО визначається список вільних пристроїв(каналів)	-	Вільні канали СМО у вигляді массиву
def get_current_ channel()	Визначення поточного каналу у якому ведеться обробка замовлення	-	Повертається індекс каналу, в якому ведеться обробка
def print_info()	Інформація про стан пристрою та черг	-	Виведення інформації у момент часу
def statistics()	Збирання статистичних даних про функціонування моделі	delta - проміжок часу спостережуваних величин	Значення черги та завантаження за спостере-жуваний інтервал часу
Класс Model			
def init()	Ініціалізація об’єкту імітаційної моделі	element_list - набір елементів, що складають модель	-
def simulate()	Алгоритм імітації моделювання системи	time - час моделювання, flag - вказівка дослідника до виконання певної	Стан моделі на крока, загальні результати моделювання, статистичні дані

		дії, epsilon - різниця відгуків	

4 ОЦІНКА АДЕКВАТНОСТІ МОДЕЛІ

Так як цільову функцію та обмеження для змінних неможливо описати аналітичним способом, то для оцінки адекватності спостерігається покрокова робота алгоритму імітації. Часові затримки для надходження та обслуговування замовлень взято з постановки завдання. На рисунку 4.1 зображено інформація про стан моделі на початку моделювання.

Рисунок 4.1 – поточна подія у моделі

У момент часу 0 надходить замовлення від відділу креслення. Для завдання формуються часові затримки обромлень у СМО та директивний термін. На рисунку 4.2 зображена інформація про стан моделі у даний момент часу.

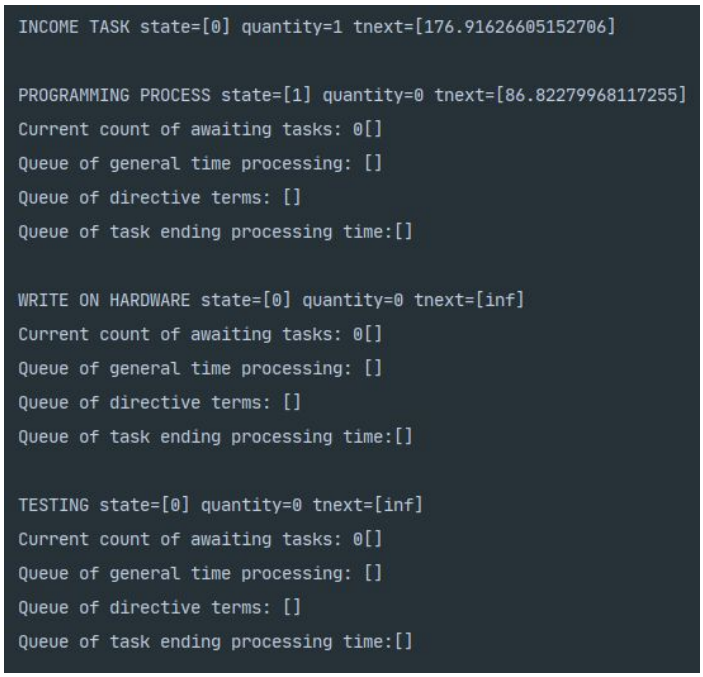

Рисунок 4.1 – стан моделі у момент часу

Для алгоритму імітації розраховується наступний момент часу для елементів моделі, у кожного ця характеристика наводиться значенням *t_next*. Вибір наступної події відбувається за визначенням мінімального найближчого часу наступної події елементів моделі.

У пристроях обслуговування наведені різні черги щодо характеристик замовлень. Це зумовлюється поставленим завданням та можливістю змінювати пріоритетність вибору завдання з черги за характеристикою. За маршрутом слідування, замовлення має надійти до відділу програмування. Це перше замовлення, тому воно одразу ж виконується відділом. На рисунку 4.3 зображено наступну подію, що була найближча у проміжку модельованого часу.

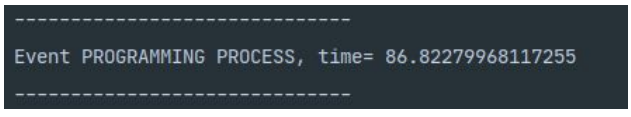

Рисунок 4.3 – наступна подія

Відповідно до встановленого часу обробки замовлення, процесс обслуговування завершився у момент часу 86.823~. На рисунку 4.4 зображено стан моделі у цей момент часу.

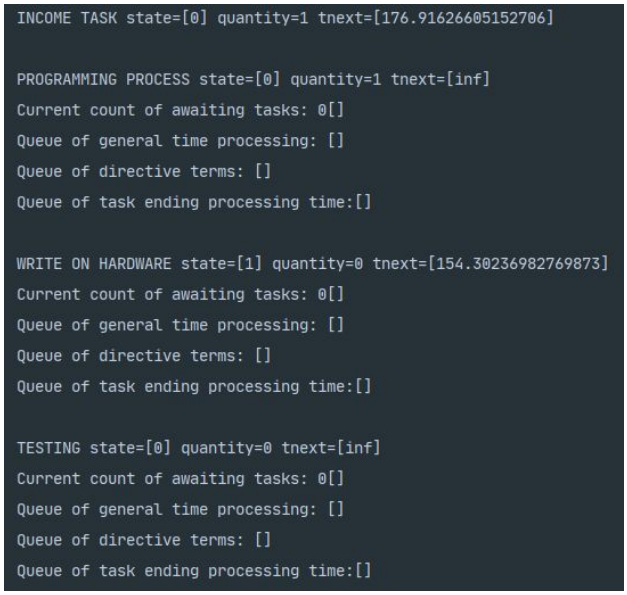

Рисунок 4.4

Правильність маршруту слідування слідування та адекватність алгоритму імітації доведена. Тепер покажемо, що завдання обирається з черги за вказаним пріоритетом. Для цього прогону були вказано пріоритет найбільшого загального часу для всіх систем обслуговування, для перевірки правильності знайдемо стан моделі в якому знаходяться замовлення в черзі для обслуговування. На рисунку 4.5 зображено стан відділу програмування, де знаходиться два замовлення у черзі.

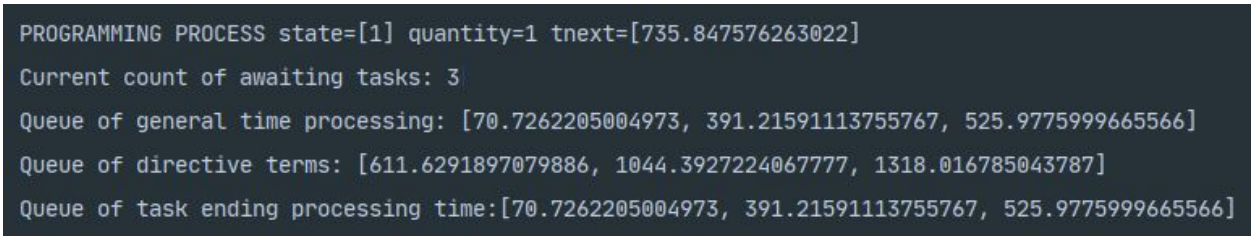

Рисунок 4.5

Завершення оброблення поточного замовлення відбудеться у моменті 735.8475, за встановленою логікою, після завершення оброблення цієї деталі,

повинно почати обробляти замовлення з загальним часом 525.977. На рисунку 4.6 зображено подію закінчення оброблення в відділі програмування.

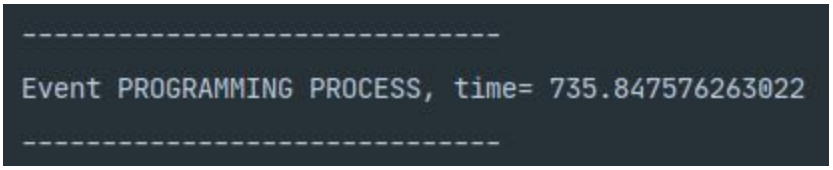

Рисунок 4.6

На рисунку 4.7 зображено стан елементу відділення програмування у цей момент часу.

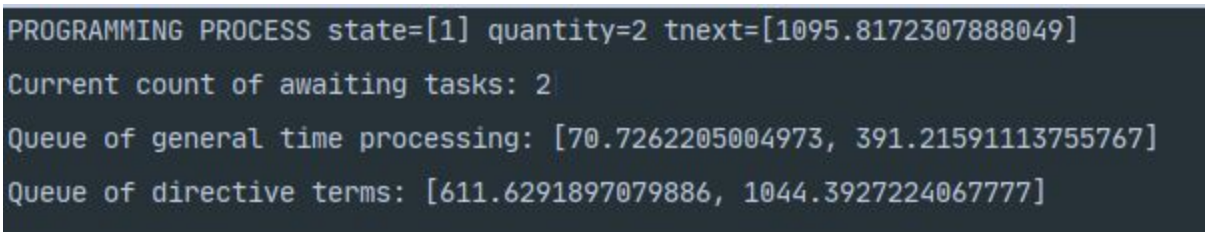

Рисунок 4.7

За інформаційним описом стану відділу програмування можна спостерігати, що збільшилась кількість оброблених деталей, а черги зменшились. Щодо правильності взяття з черги, то в обробку пішло завдання з найбільшим загальним часов виконання, це прослідковується зникненням з черги "*загальний час обробки*" значення 525.977, що було присутньо на попередньо стані СМО (див.рис.4.6).

На рисунку 4.8 зображено загальні результати імітаційного моделювання за час 1000 од.

```
---- RESULTS ---

INCOME TASK quantity=8

PROGRAMMING PROCESS quantity=7
Mean length of queue=0.29284968465153893  |  Max queue=1
Mean load=0.8609000392055394

WRITE ON HARDWARE quantity=5
Mean length of queue=0.09201994155729391  |  Max queue=1
Mean load=0.4726047448654155

TESTING quantity=5
Mean length of queue=0.022578970472206084  |  Max queue=1
Mean load=0.47683760835612116


Count of tasks that failed directive term: 1
Rate of completed tasks:0.625
```

Рисунок 4.7

За вказаний час моделювання надійшло 8 замовлень, з них повність оброблених лише 5, коефіцієнт виконання становить 0.625. Наведені максимальні та середні значення черг до СМО, та середню завантаженість за час моделювання. Кількість замовлень, що не були виконанні у встановлений директивний термін дорівнюють одиниці.

4.1 Верифікація алгоритму імітації

Для верифікації алгоритму імітації здійснюється порівняння результатів моделювання за вказаний час зі зміною вхідних параметрів

моделі. Для поставленого завдання збільшується інтенсивність надходження замовлень до системи. На рисунку 4.9 зображена зміна інтервалу розподілу часової затримки поступлення замовлень.

```
c1 = Create([50.0, 70.0], 120.0, 110.0, 90.0, [20.0, 60.0])
```

Рисунок 4.8 - інтервал затримок надходження прийняв значення [50;70]

Усі інші вхідні змінні не змінили свого значення відповідно до попередньої моделі (див. рис. 3.4). Від результатів моделювання очікується збільшення кількості замовлень, що надійшли. Збільшення: максимального та середнього значень черги до СМО, а також середнього значення завантаженості і кількості замовлень, що не були виконанні у дерективний термін. І безперечно зменшення коефіцієнту виконаних замовлень. На рисунку 4.9 зображено результати моделювання за зміненою моделлю(див. рис. 4.8). Гіпотези до зміни статистики підтвердились, а отже алгоритм імітації теж верифіковано.

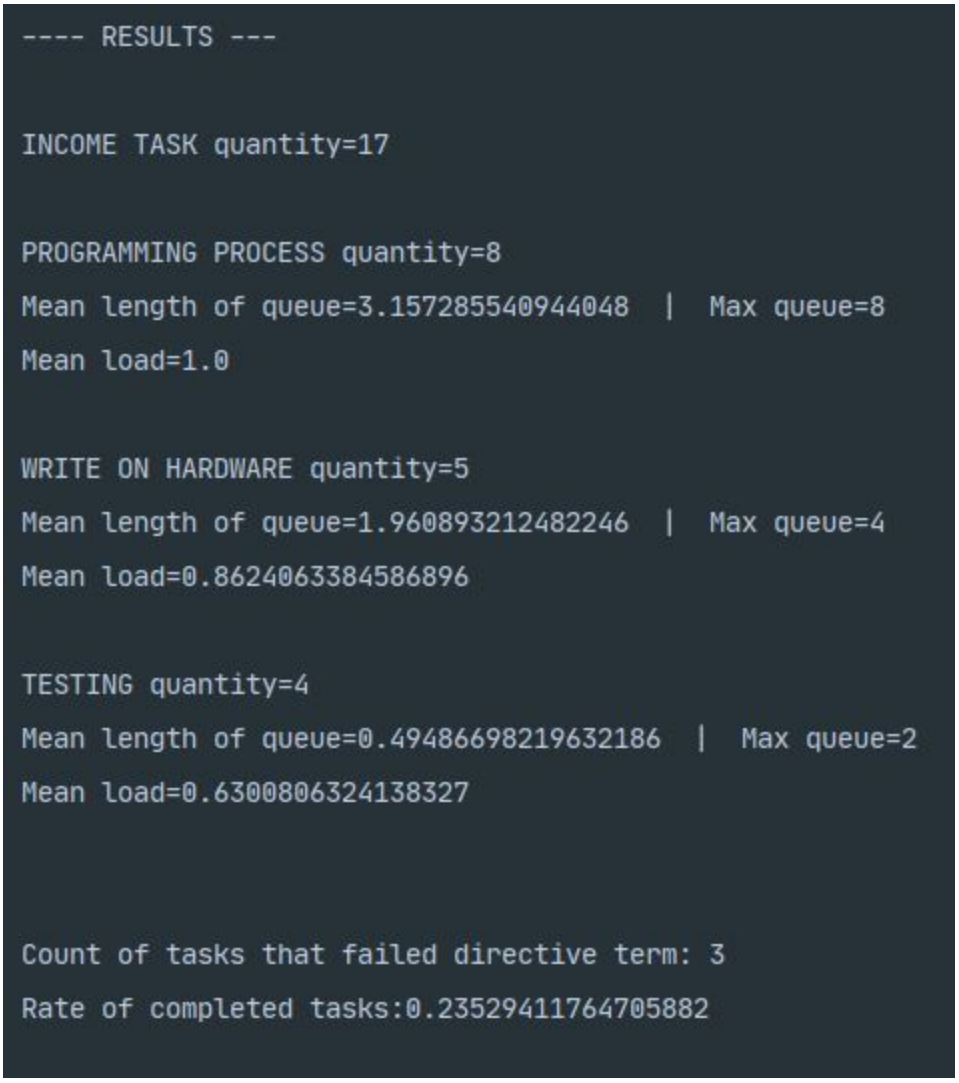

Рисунок 4.8 - результати моделювання зміненою імітаційної моделі

5 ОРГАНІЗАЦІЯ ЕКСПЕРИМЕНТІВ З МОДЕЛЛЮ

5.1 План організації експериментів

У якості дослідження функціонування моделі виконується факторний аналіз щодо впливу пріоритетності черги на загальне значення середньої завантаженості системи та загальне середнє значення черг. Так як фактор вибору черги може приймати має 4 рівні за постановкою завдання, то час моделювання визначається в середньому від отриманих значих варіантів.

5.1.1 Визначення часу моделювання

У якості відгуку моделі береться значення загальної середньої завантаженості системи. За закінчення перехідного періоду приймається час, коли різниця між поточним та попереднім загальним завантаженням не перевищуватиме значення 0.01 п’ять разів підряд.

Виконується 10 пробних прогонів, з типом пріоритетності вибору замовлення “найменший загальний час оброблення”. На рис. 5.1–5.10 представлено фрагменти виведення результатів прогону моделі та виділено значення перехідних періодів.

```
Current time:1236.1578807360772 | Respond:0.5000830398303425 | Delta:0.0015103067911034773
Current time:1248.095043389837 | Respond:0.5016762965775191 | Delta:0.0015932567471765857
Current time:1325.867153146279 | Respond:0.5113542263315519 | Delta:0.00967792975403281
Current time:1346.0118433373864 | Respond:0.5136786642693475 | Delta:0.002324437937795576
Current time:1352.9780102504747 | Respond:0.5161826189326377 | Delta:0.002503954663290231
End termination period
```

Рисунок 5.1

```
Current time:1287.733198781243 | Respond:0.5629258517258454 | Delta:0.004287547114121937
End termination period
```

Рисунок 5.2

```
Current time:1243.8691988234343 | Respond:0.5158402038607298 | Delta:0.00018943315769515134
End termination period
```

Рисунок 5.3

```
Current time:580.8058919067679 | Respond:0.5009092833882322 | Delta:0.0026758018541032658
End termination period
```

Рисунок 5.4

```
Current time:1673.8745552132764 | Respond:0.5204247987595823 | Delta:0.0041237003587721865
End termination period
```

Рисунок 5.5

```
Current time:646.6972269330405 | Respond:0.6403799961591635 | Delta:0.006910451863468392
End termination period
```

Рисунок 5.7

```
Current time:1070.1007078852663 | Respond:0.4759419673032726 | Delta:0.0013618625376710658
End termination period
```

Рисунок 5.8

```
Current time:1229.4834875258448 | Respond:0.3716081719810107 | Delta:0.006164717741694747
End termination period
```

Рисунок 5.9

```
Current time:1912.1101175961755 | Respond:0.5558937093556456 | Delta:0.00985799975866175
End termination period
```

Рисунок 5.10

За наведеними результатами здійснюється висновок, що перехідний період моделі складає в середньому 1000-1200 одиниць часу для пріоритетності “найменший загальний час”. На рисунках 5.11-5.20 представлено результати 10 пробних прогонів, з типом пріоритетності вибору замовлення “найбільший загальний час оброблення”.

```
Current time:652.3647714682326 | Respond:0.44893733188603746 | Delta:0.004448665718551348
End termination period
```

Рисунок 5.11

```
Current time:1821.2505215816775 | Respond:0.5794415422986301 | Delta:0.004789326879159339
End termination period
```

Рисунок 5.12

```
Current time:813.069111947576 | Respond:0.6053941491161986 | Delta:0.0036783004278172493
End termination period
```

Рисунок 5.13

```
Current time:1661.0801780525292 | Respond:0.6695721380710175 | Delta:0.001188615339367205
End termination period
```

Рисунок 5.14

```
Current time:1262.6321624815369 | Respond:0.5496940985929313 | Delta:0.007429745455653003
End termination period
```

Рисунок 5.15

```
Current time:733.0363296441359 | Respond:0.471552163038023 | Delta:0.002265946487845949
End termination period
```

Рисунок 5.16

```
Current time:1260.5173283291415 | Respond:0.5726916569430643 | Delta:0.0016848753461030652
End termination period
```

Рисунок 5.17

```
Current time:1322.1480637189188 | Respond:0.6153595112701787 | Delta:0.0022876694935528574
End termination period
```

Рисунок 5.18

```
Current time:1032.3090631766477 | Respond:0.5843964004018399 | Delta:0.009183893760254414
End termination period
```

Рисунок 5.19

```
Current time:858.37306517322 | Respond:0.6273339704523987 | Delta:0.0009031317120745674
End termination period
```

Рисунок 5.20

За наведеними результатами здійснюється висновок, що перехідний період моделі складає в середньому 1100-1300 одиниць часу для пріоритетності “найбільший загальний час”.

Такі ж розрахунки проводяться для:

- ❏ моделі з типом пріоритетності вибору замовлення “найменший залишившийся час оброблення”. Отриманий результат перехідного режиму складає 900-1100 одиниць часу.
- ❏ моделі з типом пріоритетності вибору замовлення “найближчий момент часу директивного терміну”. Отриманий результат перехідного режиму складає 1300-1500 одиниць часу.

Так як у поставленому завданні вказано, що модель повинна працювати у перехідному режимі, то час моделювання для експериментів буде становити

1200 одиниць часу для визначення впливу пріоритетності вибору завдань з черги на загальне середнє значення черги.

5.2 Експериментування

Так як модель працює у перехідному режимі, то експериментування відбувається за допомогою методу повторення. Кількість повторень прогонів становить 5 разів. У таблиці 5.1 наведено отримані вихідні значення загального середнього значення завантаженості системи.

Таблиця 5.1 - значення середнього завантаження системи

№	*Тип пріоритету для замовлень з черги*			
	Найменший загальний час	Найбільший загальний час	Найменший час обробки, що залишився	Найближчий директивний термін
1	0.468	0.583	0.5696	0.5750
2	0.5649	0.5217	0.5844	0.59429
3	0.463	0.4146	0.4827	0.54241
4	0.523	0.5019	0.4904	0.60468
5	0.7149	0.5187	0.5026	0.5913407
Avg	***0.54676***	***0.50798***	***0.52594***	***0.58154***

Серед отриманих значень альтернатив середньої завантаженості, найкраще завантаженість при пріоритеті взяття завдання з найближчим директивним терміном.

У таблиці 5.2 наведено отримані вихідні значення загального середнього значення черг замовлень у системі.

Таблиця 5.2

№	*Тип пріоритету для замовлень з черги*			
	Найменший загальний час	Найбільший загальний час	Найменший час обробки, що залишився	Найближчий директивний термін
1	0.397	0.5333	0.2652	0.3207
2	0.383	0.10067	0.24678	0.337
3	0.444	0.4823	0.21393	0.0546
4	0.141	0.2283	0.2288	0.0930

5	0.0736	1.07271	0.13382	0.2427

Продовження таблиці 5.2

№	*Тип пріоритету для замовлень з черги*			
	Найменший загальний час	Найбільший загальний час	Найменший час обробки, що залишився	Найближчий директивний термін
Avg	***0.28772***	***0.48346***	***0.21771***	***0.2096***

Серед отриманих значень альтернатив середньої черги у системі, найкраще значення отримано при пріоритеті взяття завдання з найближчим директивним терміном.

У таблиці 5.4 наведено значення коефіцієнту виконаних завдань за час моделювання.

Таблиця 5.4

№	*Тип пріоритету для замовлень з черги*			
	Найменший загальний час	Найбільший загальний час	Найменший час обробки, що залишився	Найближчий директивний термін
1	0.667	0.666	0.7	0.6
2	0.444	0.5	0.778	0.667
3	0.75	0.625	0.6	0.778
4	0.8	0.375	0.778	0.667
5	0.75	0.625	0.666	0.556
Avg	***0.6822***	***0.5582***	***0.7044***	***0.6536***

Серед отриманих значень коефіцієнтів виконання завдань найкраще значення отримано при пріоритеті взяття завдання з найменшим часом

обробки, що залишився, але різниця між середніми значеннями інших альтернатив не настільки велика.

6 АНАЛІЗ ТА ОЦІНКА РЕЗУЛЬТАТІВ

Відповідно до проведених експериментів(див. роз. 5.2), у трьох з чотирьох критеріїв найкращі середні значення отримані при пріоритеті “найближчий директивний термін”. У критерії “коефіцієнт оброблених замовлень” найкраща значення у пріоритету “найменший час обробки, що залишився”.

Мінімальна різниця середніх значень у альтернатив по критеріях вибору становить 0.05, а це занадто мале значення, аби гарантувати найкращий вибір пріоритету.

7 ВИСНОВКИ І РЕКОМЕНДАЦІЇ ЩОДО ВИКОРИСТАННЯ МОДЕЛІ

Поставлене завдання було вирішено за допомогою розробленого ПЗ та побудованої імітаційної моделі. Для власників підприємств, що створюють програмне керування для металорізальних верстатів, повинно бути зручно користуватись моделлю. Простота зміни вхідних змінних та результати моделювання надають можливість досліджувати роботу системи та налагоджувати процеси виробництва. Керуючись цим документ як технічною документацією, можна змінювати програмне забезпечення.

ПЕРЕЛІК ПОСИЛАНЬ

1. Стеценко І. В. Моделювання систем. Навчальний посібник / Інна Вячеславівна Стеценко., 2011. – 407 с.
2. Операционный анализ [Електронний ресурс] – Режим доступу до ресурсу:https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D1%8B%D0%B9_%D0%B0%D0%BD%D0%B0%D0%BB%D0%B8%D0%B7.

ДОДАТОК 1 - ЛІСТИНГ

Файл sim_model.py

```
def sim_model():

    # lowest general time
    # highest general time
    # lowest time of left processing
    # closest directive term

    c1 = Create([100.0, 200.0], 120.0, 110.0, 90.0, [20.0, 60.0])

    p1 = Process(task_priority="closest directive term")
    p2 = Process(task_priority="closest directive term")
    p3 = Process(task_priority="closest directive term")

    c1.name = 'INCOME TASK'
    p1.name = 'PROGRAMMING PROCESS'
    p2.name = 'WRITE ON HARDWARE'
    p3.name = 'TESTING'

    c1.next_element = [p1]
    p1.next_element = [p2]
    p2.next_element = [p3]

    element_list = [c1, p1, p2, p3]
    model = Model(element_list)
    model.simulate(1200.0, True)



sim_model()
```

Файл model.py

```
class Model(object):
    def __init__(self, elements_list):
        self.element_list = elements_list
        self.t_next = 0
        self.event = 0
        self.t_curr = 0
        self.stable = []
        self.general_mean_load, self.general_mean_queue = 0.0, 0.0

    def simulate(self, time, flag, epsilon=None):
        while self.t_curr < time:
            if self.t_curr > 0.0:
                p1 = self.element_list[1].mean_load / self.t_curr
                p2 = self.element_list[2].mean_load / self.t_curr
                p3 = self.element_list[3].mean_load / self.t_curr

            else:
                p1 = self.element_list[1].mean_load
                p2 = self.element_list[2].mean_load
                p3 = self.element_list[3].mean_load

            avg_load_start = (p1 + p2 + p3) / 3

            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 {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

        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 epsilon is not None:
            if self.t_curr > 0.0:
                p1 = self.element_list[1].mean_load / self.t_curr
                p2 = self.element_list[2].mean_load / self.t_curr
                p3 = self.element_list[3].mean_load / self.t_curr
                avg_load_end = (p1 + p2 + p3) / 3
                delta = math.fabs(avg_load_end - avg_load_start)
                                print(f"Current  time:{self.t_curr}  |  Respond:{avg_load_end}  |
Delta:{delta}")
                if 0.0 < delta < epsilon:
                    self.stable.append(1)
                    if len(self.stable) >= 5:
                        i = -1
                        while i > -6:
                            if self.stable[i] == 1:
                                i -= 1
                                if i == -6:
```

```
                        print("End termination period")
                        exit()

                else:
                    break
            else:
                self.stable.append(0)

        if self.element_list[self.event].id_element == 0:
            self.element_list[self.event].print_create_info(flag)
        self.print_info(flag)

    self.print_result(flag=True)

def print_info(self, flag):
    if flag is True:
        for element in self.element_list:
            element.print_info()

def print_result(self, flag):
    if flag is True:
        print("\n---- RESULTS ---")
        for element in self.element_list:
            element.print_res()
            if isinstance(element, Create):
                income_task = element.quantity
            elif isinstance(element, Process):
                p = element
                self.general_mean_load += p.mean_load / self.t_curr
                self.general_mean_queue += p.mean_queue / self.t_curr
                    print(f"Mean length of queue: {p.mean_queue/self.t_curr}  |  Max queue:
{p.count_max_queue}"
```

```
                f"\nMean load: {p.mean_load / self.t_next}")

            if p.directive_fail >= 0:
                print(f"\n")
                print(f"Count of tasks that failed directive term: {p.directive_fail}")
                print(f"Rate of completed tasks:{p.quantity / income_task}")

        print(f"\nAverage load of system processors: {self.general_mean_load/3}")
        print(f"Average queue of system processors: {self.general_mean_queue / 3}")
```

Файл element.py

```
class Element(object):
    next_id = 0

    def __init__(self, name_element=None, delay=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 = ('Element:', self.id_element)
        self.quantity = int()
        self.delay_dev = float()
        self.delay_mean = float()
        self.distribution = str()
        self.probability = [1]
        if delay is None and name_element is None:
            self.delay_mean = 1.0
            self.distribution = 'exp'
        elif delay is not None and name_element is None:
```

```
            self.delay_mean = delay
            self.distribution = str('')
        elif delay is not None and name_element is not None:
            self.delay_mean = delay
            self.distribution = 'exp'

    @property
    def delay_mean(self):
        return self.__delay_mean

    @delay_mean.setter
    def delay_mean(self, delay_mean):
        self.__delay_mean = delay_mean

    @property
    def delay_dev(self):
        return self.__delay_dev

    @delay_dev.setter
    def delay_dev(self, delay_dev):
        self.__delay_dev = delay_dev

    @property
    def quantity(self):
        return self.__quantity

    @quantity.setter
    def quantity(self, quantity):
        self.__quantity = quantity

    @property
    def distribution(self):
```

```
    return self.__distribution

@distribution.setter
def distribution(self, distribution):
    self.__distribution = distribution

@property
def state(self):
    return self.__state

@state.setter
def state(self, state):
    self.__state = state

@property
def next_element(self):
    return self.__next_element

@next_element.setter
def next_element(self, next_element):
    self.__next_element = next_element

@property
def t_curr(self):
    return self.__t_curr

@t_curr.setter
def t_curr(self, t_curr):
    self.__t_curr = t_curr

@property
def t_next(self):
```

```
    return self.__t_next

@t_next.setter
def t_next(self, t_next):
    self.__t_next = t_next

@property
def id_element(self):
    return self.__id_element

@id_element.setter
def id_element(self, id_element):
    self.__id_element = id_element

@property
def name(self):
    return self.__name

@name.setter
def name(self, name):
    self.__name = name

@property
def probability(self):
    return self.__probability

@probability.setter
def probability(self, value):
    self.__probability = value

def get_delay(self):
    if self.distribution == 'expo':
```

```
            delay = exp(self.delay_mean)
        elif self.distribution == 'normal':
            delay = norm(self.delay_mean, self.delay_dev)
        elif self.distribution == 'uniform':
            delay = unif(self.delay_mean, self.delay_dev)
        else:
            delay = self.delay_mean
        return delay

    def out_act(self):
        self.quantity += 1

    def statistics(self, delta):
        pass

    def print_res(self):
        print(f"\n{self.name} quantity={self.quantity}")

    def print_info(self):
                            print(f"\n{self.name}    state={self.state}    quantity={self.quantity}
tnext={self.t_next}")
```

Файл create.py

```
class Create(Element):
    def __init__(self, interval1, smo1_delay, smo2_delay, smo3_delay,  interval2):
        super().__init__(name_element=None, delay=interval1[0])
        self.distribution = 'unif'
        self.delay_dev = interval1[1]
        self.delay1, self.delay2, self.delay3 = smo1_delay, smo2_delay, smo3_delay
        self.smo1_delay, self.smo2_delay, self.smo3_delay = 0.0, 0.0, 0.0
        self.add_time_min = interval2[0]
```

```
        self.add_time_max = interval2[1]
        self.general_time, self.techno_time, self.add_time = 0.0, 0.0, 0.0
        self.properties = []

    def out_act(self):
        super().out_act()
        self.smo1_delay = exp(self.delay1)
        self.smo2_delay = exp(self.delay2)
        self.smo3_delay = exp(self.delay3)
        self.add_time = unif(self.add_time_min, self.add_time_max)
        self.general_time = self.smo1_delay + self.smo2_delay + self.smo3_delay
        self.techno_time = self.general_time + self.add_time
        delay = unif(self.delay_mean, self.delay_dev)
        self.t_next[0] = self.t_curr + delay
        directive_term = self.t_curr + self.techno_time + delay
        self.properties = [self.smo1_delay, self.smo2_delay, self.smo3_delay, self.add_time,
self.general_time, directive_term]
        self.next_element[0].in_act(self.properties)

    def print_create_info(self, flag):
        if flag is True:
                        print(f"Programming delay:{self.properties[0]} | Hardware write
delay:{self.properties[1]} | Testing delay:{self.properties[2]}")
            print(f"Technical delay:{self.techno_time} | Directive term {self.properties[5]}")
```

Файл process.py

```
class Process(Element):
    def __init__(self, channels=1, task_priority=None):
        super().__init__(name_element=None, 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 == "closest directive term":
                                                                    task_index =
self.directive_term_queue.index(min(self.directive_term_queue))
                elif self.task_priority == "lowest time of left processing":
                    task_index = self.left_time_queue.index(min(self.left_time_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

    @property
    def channel(self):
        return self.__channel

    @channel.setter
    def channel(self, value):
        self.__channel = value

    @property
```

```
def max_queue(self):
    return self._max_queue

@max_queue.setter
def max_queue(self, value):
    if value >= 0:
        self._max_queue = value
    else:
        print("Whoops, max queue doesn't support this type")

def print_info(self):
    super().print_info()
    print(f"Current count of awaiting tasks: {self.queue_count}"
          f"\nQueue of general time processing: {self.general_time_queue}"
          f"\nQueue of directive terms: {self.directive_term_queue}"
          f"\nQueue of task ending processing time:{self.left_time_queue}")

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
```