Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

Комп’ютерний практикум №2

Моделювання систем

Виконав студент групи ІП-15: Плугатирьов Д.В.		Перевірив: Стеценко І.В.
		Дата:
		Оцінка:

Київ 2025

Завдання

1. Реалізувати алгоритм імітації простої моделі обслуговування одним пристроєм з використанням об’єктно-орієнтованого підходу. **5 балів.**
2. Модифікувати алгоритм, додавши обчислення середнього завантаження пристрою. **5 балів.**
3. Створити модель за схемою, представленою на рисунку 1. **30 балів.**
4. Виконати верифікацію моделі, змінюючи значення вхідних змінних та параметрів моделі. Навести результати верифікації у таблиці. **10 балів.**

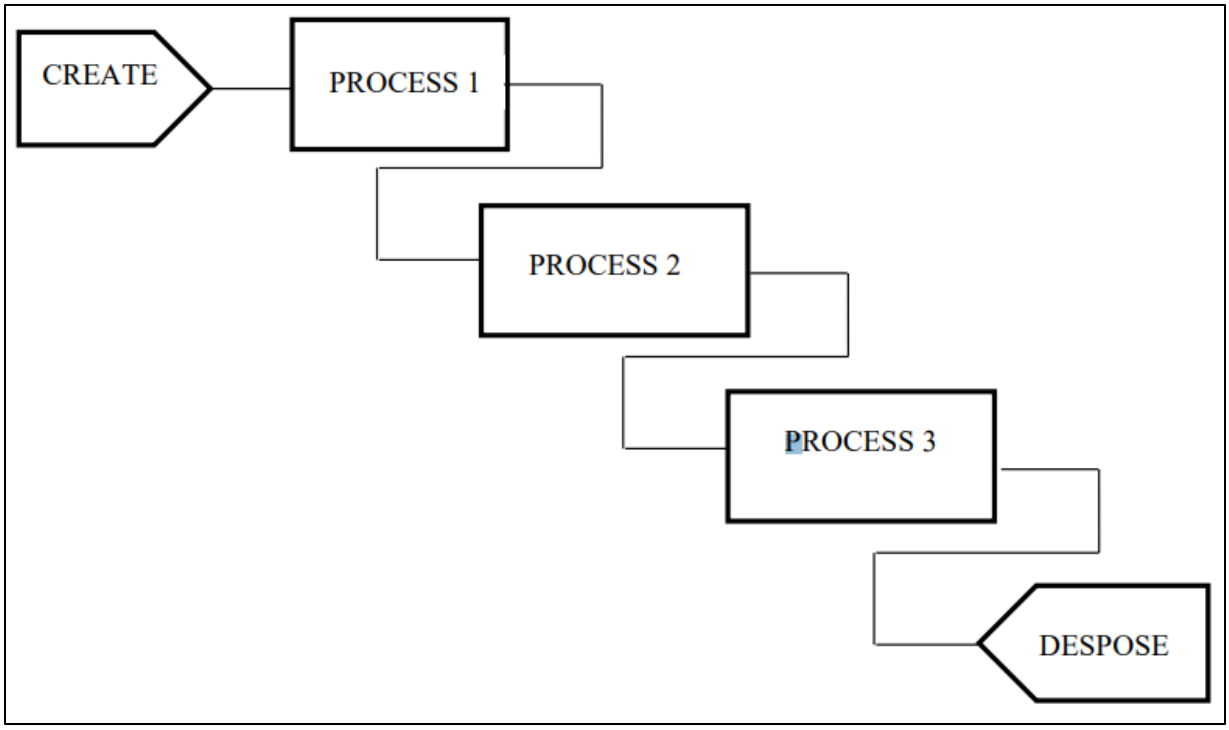

Рисунок 1 – Схема моделі

5. Модифікувати клас PROCESS, щоб можна було його використовувати для моделювання процесу обслуговування кількома ідентичними пристроями. **20 балів.**
6. Модифікувати клас PROCESS, щоб можна було організовувати вихід в два і більше наступних блоків, в тому числі з поверненням у

попередні блоки. **30 балів.**

Структура коду

Програма реалізована у шести файлах. Файл **utils.py** містить допоміжні функції та константи (зокрема, форматування часу, **INF_TIME**, **TIME_EPS**, а також декоратор для підрахунку створених екземплярів).

У файлі **base_element.py** оголошено базовий клас **Element**, який відповідає за основну логіку обробки подій: зберігає час поточної події, генерує час наступної, передає подію далі за заданими імовірностями переходу.

Клас **CreateElement**, що знаходиться у файлі **creator.py**, відповідає за генерацію заявок (подій) з обраним розподілом інтервалів (експоненційним у нашому прикладі). Його метод **end_action()** збільшує лічильник створених заявок та визначає час появи наступної події.

Файл **process.py** містить клас **ProcessElement**. Він реалізує чергу (з обмеженням на максимальний розмір) і підтримує роботу з декількома ідентичними “обробниками” (handlers). У випадку, коли всі обробники зайняті, заявка або потрапляє у чергу, або “втрачається” (збільшується кількість відмов), якщо черга переповнена. Після завершення обслуговування подія може бути скерована в один чи кілька наступних блоків із заданими імовірностями (у прикладі частина заявок повертається з третього процесу назад у перший).

Логікою запуску та “прогону” (simulation run) усієї мережі елементів керує клас **Model** із файлу **simulation_model.py**. Він відстежує глобальний час, знаходить найраніший момент завершення події серед усіх елементів, викликає для них **end_action()** і так далі, аж поки не буде досягнуто кінцевий час моделювання.

Нарешті, у файлі **simulation.py** створюються та з’єднуються екземпляри

CreateElement і кількох **ProcessElement**, встановлюються початкові параметри (наприклад, кількість обслуговуючих пристроїв, інтенсивність надходження заявок тощо), після чого запускається симуляція та формується підсумкова статистика.

Алгоритм роботи програми

На початку **CreateElement** з певним середнім інтервалом часу генерує нову заявку та негайно передає її на вхід першому **ProcessElement**. Процесори (**ProcessElement**) приймають заявку в чергу, якщо всі пристрої вже зайняті; якщо у черзі є вільне місце, заявка чекатиме, інакше система “відмовляє”. Коли обслуговування завершується, метод **end_action()** передає заявку далі за визначеними ймовірностями: у прикладі частина заявок може повертатися назад у попередній процес. Клас **Model** синхронізує події: знаходить найближчий час завершення обслуговування серед усіх процесорів та генерує виклик їхніх **end_action()**. По закінченні заданого періоду моделювання збирається статистика: кількість створених або оброблених заявок, середній розмір черги, середній час очікування, середнє завантаження обслуговуючих пристроїв і ймовірність відмови.

Підсумкові статистичні дані моделювання

На рисунку нижче наведено приклад підсумкового виводу для проміжку часу 1000 умовних одиниць, коли перший блок генерує заявки із середнім інтервалом 0.2.

Три процесори мають власні характеристики обслуговування:

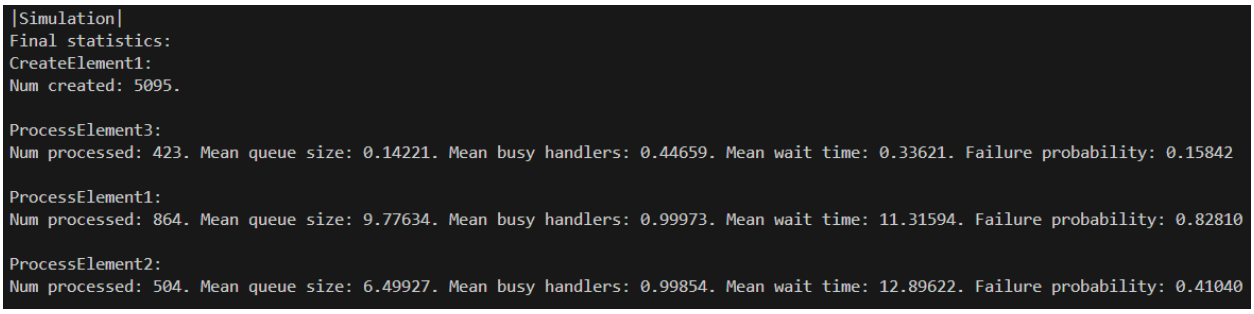

Рисунок 2 – Підсумкові статистичні дані моделювання

Таблиця 1 – Результати верифікації

Елемент	**Кількість оброблених (або створених) заявок**	**Середній розмір черги**	**Середнє завантаження пристроїв**	**Середній час очікування**	**Ймовірність відмови**
CreateElement1	5095	-	-	-	-
ProcessElement1	864	9.77634	0.99973	11.31594	0.82810
ProcessElement2	504	6.49927	0.99854	12.89622	0.41040
ProcessElement3	423	0.14221	0.44659	0.33621	0.15842

У початковому режимі (без модифікації) генератор заявок (CreateElement1) створив 5095 одиниць за 1000 умовних одиниць часу, що задає приблизно стабільне навантаження. Для процесних елементів спостерігається досить специфічна картина:

– **ProcessElement1:** Оброблено лише 864 заявки при середній довжині черги ≈9.78 та майже 100% завантаженні пристроїв (0.99973). Середній час очікування склав ≈11.32, а висока ймовірність відмов (0.82810) свідчить про критичне перевантаження через обмежену можливість

паралельного обслуговування.

- **ProcessElement2:** Аналогічно, обробка 504 заявок, середня черга ≈6.50 та час очікування ≈12.90 вказують на значні затримки і відмови (ймовірність 0.41040) через недостатню кількість обслуговуючих каналів.
- **ProcessElement3:** Тут отримано нижчі значення – 423 оброблені заявки, мінімальна середня довжина черги (≈0.14) та короткий час очікування (≈0.34), а ймовірність відмов (0.15842) залишається досить низькою. Частково це може свідчити про те, що навантаження на цей елемент було меншим або ж його внутрішня політика обслуговування дозволяла ефективно розвантажувати вхідний потік.

Модифіковані процесори для можливості обслуговування багатьма паралельними пристроями.

```
|Modified simulation|
Final statistics:
ProcessElement6:
Num processed: 1736. Mean queue size: 0.48721. Mean busy handlers: 1.73757. Mean wait time: 0.28067. Failure probability: 0.48611

ProcessElement4:
Num processed: 4086. Mean queue size: 6.98675. Mean busy handlers: 4.93770. Mean wait time: 1.71002. Failure probability: 0.21760

ProcessElement5:
Num processed: 3383. Mean queue size: 4.48783. Mean busy handlers: 6.78222. Mean wait time: 1.32666. Failure probability: 0.16981

CreateElement2:
Num created: 5044.
```

Рисунок 3 – Підсумкові статистичні дані модифікованого моделювання

Таблиця 2 – Результати верифікації після модифікації

Елемент	**Кількість оброблених (або створених) заявок**	**Середній розмір черги**	**Середнє завантаження пристроїв**	**Середній час очікування**	**Ймовірність відмови**
CreateElement1	5165	-	-	-	-

Продовження таблиці 2

ProcessElement1	4121	6.72649	4.94115	1.63236	0.22537
ProcessElement2	3371	4.50005	6.82404	1.33502	0.17831
ProcessElement3	1730	0.48625	1.71670	0.28109	0.48577

У модифікованій моделі, на відміну від звичайної послідовної схеми, реалізовано використання паралельних обслуговуючих механізмів через параметр **num_handlers** у кожному з процесних блоків. Це дозволило збільшити пропускну здатність системи за рахунок одночасної обробки більшої кількості заявок. У вихідній (звичайній) моделі кожен процесний елемент працював із єдиним потоком (відсутність багатопоточності), що призводило до накопичення заявок у чергах, зростання часу очікування та підвищеної ймовірності відмов.

У новій конфігурації:

- **ProcessElement1** отримав 5 паралельних каналів обслуговування, що значно зменшило середню довжину черги (з 9.78 до 6.73) та час очікування (з ≈11.3 до ≈1.63). Це свідчить про успішне розвантаження елемента завдяки більшій кількості обробників, які дозволяють обробляти заявок набагато більше за одиницю часу.
- **ProcessElement2** при 7 паралельних каналах демонструє подібну тенденцію із зменшенням черги та часу очікування (черга скорочується до ≈4.50, час до ≈1.34), що також підтверджує покращення пропускної здатності.

- **ProcessElement3** – хоча кількість оброблених заявок значно зросла (з 423 до 1730), тут впроваджено додатковий механізм зворотного зв’язку: 10% заявок повертаються до ProcessElement1. Результатом є дещо підвищена середня довжина черги і збільшення ймовірності відмов (з ≈0.16 до ≈0.49). Це вказує на те, що для цього елемента, зокрема через невеликий розмір черги (максимум 1) та обмежену кількість обслуговуючих каналів (2), додаткове навантаження із зворотного переходу потребує подальшої оптимізації.

Посилання на репозиторій з кодом: https://github.com/I-delver-I/system-modelling.

Висновок

Проведене моделювання системи обробки заявок у двох варіантах — стандартному (послідовному) та модифікованому (з використанням паралельного обслуговування) — дозволило глибоко проаналізувати вплив розширення кількості обслуговуючих пристроїв на продуктивність системи. У стандартній моделі кожен блок обслуговування працював послідовно, що призводило до збільшення середньої довжини черг та значного часу очікування при високому навантаженні. Це було явно відображено у характеристиках ProcessElement1 та ProcessElement2, де обмежені можливості паралельної обробки спричиняли накопичення заявок і високий рівень відмов.

У модифікованій моделі впровадження багатоканальності через параметри num_handlers значно покращило ситуацію: завдяки одночасній обробці заявок зменшилися як середня довжина черги, так і час очікування, що дозволило підвищити загальну пропускну здатність системи. Конкретніше, для ProcessElement1 та ProcessElement2 спостерігається різке зниження показників часу очікування та ймовірності відмов, що свідчить про ефективне

використання додаткових ресурсів.

Однак аналіз також виявив, що навіть за умов паралельного обслуговування оптимізація не є універсальною. Зокрема, у ProcessElement3, який має обмежений розмір буфера і лише два обслуговуючих канали, додаткове навантаження від повернення 10% заявок до попереднього елемента призводить до зростання ймовірності відмов. Це підкреслює необхідність ретельного балансування параметрів: збільшення кількості пристроїв може знизити затримки, але без оптимального розподілу ресурсів і налаштування маршрутів обробки навіть паралельне обслуговування може спричиняти неочікувані проблеми.

Загалом, отримані результати демонструють, що модифікація моделі із використанням паралельного обслуговування є потужним інструментом для підвищення ефективності системи, знижуючи час очікування та середню довжину черг. Проте для досягнення оптимальної продуктивності необхідно враховувати специфіку кожного блокового елемента, особливо при організації зворотних маршрутів обробки заяв.