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

Комп’ютерне моделювання дискретно-подійних систем

Виконав студент групи ІП-21: Плугатирьов Д.В.		Перевірив: Стеценко І.В.
		Дата:
		Оцінка:

Київ 2026

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.**

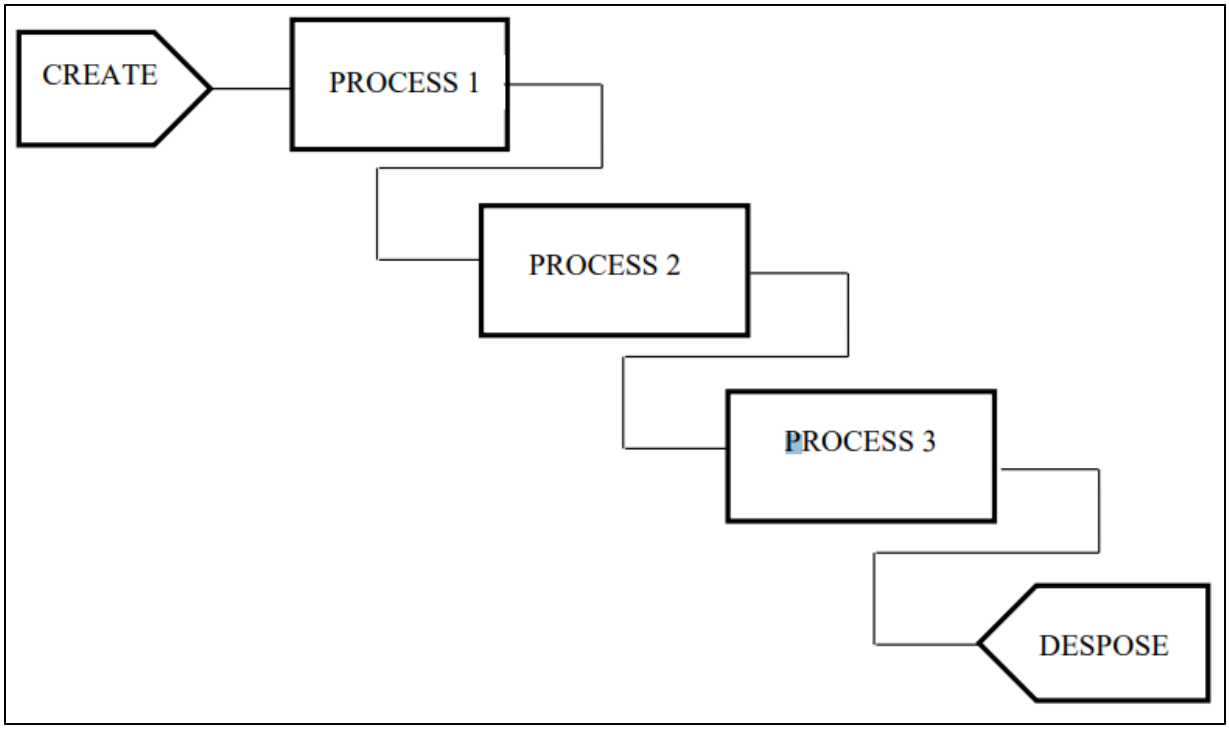

Рисунок 1 – Схема моделі

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.

Завдання 1, 2: проста модель із обчисленням середнього завантаження пристрою

На рисунку нижче наведено приклад підсумкового виводу для проміжку часу 1000 умовних одиниць, коли перший блок генерує заявки із середнім інтервалом 2.

```
=================== Завдання 1 та 2: Проста модель ===================
PROCESSOR:
Num processed: 497. Mean queue size: 0.52996. Mean busy handlers: 0.53729. Mean wait time: 1.06653. Failure p
robability: 0.00794
CREATE:
Num created: 503.
DISPOSE:
Num disposed: 497.
```

Рисунок 1 – Підсумкові статистичні дані моделювання простої моделі

Таблиця 1 – Результати верифікації

Елемент	**Кількість оброблених (або створених) заявок**	**Середній розмір черги**	**Середнє завантаження пристроїв**	**Середній час очікування**	**Ймовірність відмови**
CREATE	503	-	-	-	-
PROCESSOR	497	0.52996	0.53729	1.06653	0.00794
DISPOSE	497	-	-	-	-

Ця статистика відображає роботу **простої одноканальної системи** масового обслуговування (СМО), що складається з джерела заявок (CREATE) та одного обслуговуючого пристрою (PROCESSOR).

1. Джерело заявок (CREATE)
   - Num created: 503 (кількість створених). За час симуляції (1000 одиниць часу) у систему надійшло **503 заявки**.
2. Обслуговуючий пристрій (PROCESSOR)
   - Num processed: 497 (кількість оброблених). Пристрій успішно обслужив **497 заявок**. *(Різниця між створеними та обробленими заявками — це ті, що були втрачені через переповнення черги, або ті, що залишилися в системі на момент закінчення часу).*
   - Mean queue size: 0.52996 (середня довжина черги). У середньому протягом усього часу роботи в черзі очікувало **0.53 заявки**. Це свідчить про те, що черга не накопичувалась значно.
   - Mean busy handlers: 0.53729 (середнє завантаження). Пристрій був зайнятий **53.7% часу** (або, інакше кажучи, коефіцієнт використання пристрою приблизно 0.54). Оскільки пристрій один, це означає, що майже половину часу він простоював.
   - Mean wait time: 1.06653 (середній час очікування). У середньому

кожна заявка чекала в черзі перед обслуговуванням **1.07 одиниці часу**.

- Failure probability: 0.00794 (ймовірність відмови). **0.01%** заявок отримали відмову і покинули систему необслуговуваними. Це сталося тому, що в певні моменти черга досягала свого максимуму (у коді встановлено обмеження max_queue_size=5), і нові заявки не могли туди потрапити.

Система працює **стабільно** з помірним навантаженням (~54%). Більшість заявок успішно обробляються, черги невеликі, а втрати заявок мінімальні (~0.01%), що є гарним показником для простої системи з обмеженою чергою.

Завдання 3: лінійна модель

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:

```
==================== Завдання 3: Лінійна модель (Рис. 2.1) ====================
Final statistics:
PROCESS_3:
Num processed: 418. Mean queue size: 0.14629. Mean busy handlers: 0.43825. Mean wait time: 0.34998. Failure pr
obability: 0.13786

PROCESS_2:
Num processed: 485. Mean queue size: 6.44044. Mean busy handlers: 0.99502. Mean wait time: 13.27943. Failure p
robability: 0.40931

CREATE:
Num created: 4979.

PROCESS_1:
Num processed: 837. Mean queue size: 9.79877. Mean busy handlers: 0.99983. Mean wait time: 11.70717. Failure p
robability: 0.82952

DISPOSE:
Num disposed: 418.
```

Рисунок 2 – Підсумкові статистичні дані моделювання лінійної моделі

Таблиця 2 – Результати верифікації

Елемент	**Кількість оброблених (або створених) заявок**	**Середній розмір черги**	**Середнє завантаження пристроїв**	**Середній час очікування**	**Ймовірність відмови**
CREATE	4979	-	-	-	-
PROCESS_1	837	9.79877	0.99983	11.70717	0.82952
PROCESS_2	485	6.44044	0.99502	13.27943	0.40931
PROCESS_3	418	0.14629	0.43825	0.34998	0.13786
DISPOSE	418	-	-	-	-

У початковому режимі (без модифікації) генератор заявок (CREATE) створив 4979 одиниць за 1000 умовних одиниць часу, що задає приблизно стабільне навантаження. Для процесних елементів спостерігається досить специфічна картина:

- **PROCESS_1:** Оброблено лише 837 заявок при середній довжині черги ≈9.8 та майже 100% завантаженні пристроїв (0.99983). Середній час очікування склав ≈11.7, а висока ймовірність відмов (0.82952) свідчить про критичне перевантаження через обмежену можливість паралельного обслуговування.
- **PROCESS_2:** Аналогічно, обробка 485 заявок, середня черга ≈6.44 та час очікування ≈13.27 вказують на значні затримки і відмови (ймовірність 0.40931) через недостатню кількість обслуговуючих каналів.
- **PROCESS_3:** Тут отримано нижчі значення – 418 оброблені заявки, мінімальна середня довжина черги (≈0.15) та короткий час очікування (≈0.44), а ймовірність відмов (0.13786) 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.

Завдання 4: верифікація моделі із завдання 3

Для верифікації моделі, побудованої у Завданні 3, було проведено серію експериментів зі змінною інтенсивністю вхідного потоку заявок.

В якості змінного параметра було обрано **середній інтервал надходження заявок (Delay)** елемента CREATE. Середній час обслуговування на всіх пристроях залишався фіксованим і дорівнював **1.0 од. часу**.

Було досліджено три стани системи:

1. **Перевантаження** (Delay < 1.0): заявки надходять частіше, ніж пристрої встигають їх обробляти.
2. **Граничний стан** (Delay ≈ 1.0): інтенсивність надходження дорівнює інтенсивності обслуговування.
3. **Недовантаження** (Delay > 1.0): пристрої простоюють, черги мінімальні.

На рисунку нижче наведено приклад підсумкового виводу для проміжку часу 1000 умовних одиниць, коли перший блок генерує заявки із середніми інтервалами 0.5, 0.8, 0.9, 1, 1.1, 1.5, 2, 5:

```
==================== Завдання 4: Верифікація моделі (Завдання 3) ====================
Delay    | P1 Fail %  | P1 Queue   | P2 Queue   | P3 Queue   | Disposed
-----------------------------------------------------------------------
0.5      | 49.62      | 4.0790     | 2.3221     | 1.6155     | 782
0.8      | 27.98      | 3.0109     | 2.0192     | 1.2842     | 775
0.9      | 19.61      | 2.5572     | 1.8727     | 1.3179     | 737
1.0      | 15.11      | 2.2097     | 1.3815     | 1.4708     | 735
1.1      | 9.97       | 1.7585     | 1.1855     | 1.2565     | 717
1.5      | 2.32       | 0.8849     | 0.7336     | 0.6432     | 600
2.0      | 0.00       | 0.2853     | 0.4152     | 0.3444     | 465
5.0      | 0.00       | 0.0394     | 0.0470     | 0.0239     | 184
```

Рисунок 3 – Дані верифікації лінійної моделі

Аналіз результатів:

1. При високому навантаженні (**Delay = 0.5**) спостерігається значне зростання черги на першому пристрої (PROCESS_1), що призводить до високої ймовірності відмов (**~53%**). Це підтверджує, що перший блок стає "вузьким місцем" системи. Наступні блоки (P2, P3) мають менші черги, оскільки P1 фільтрує потік заявок.
2. При зменшенні навантаження (**Delay = 2.0...5.0**) черги зникають, а ймовірність відмови стає нульовою. Система працює стабільно, більшу частину часу пристрої простоюють.
3. Кількість оброблених заявок (Disposed) пропорційно зменшується зі збільшенням інтервалу надходження, що є логічним.

Завдання 5, 6: складна модель

Модифіковані процесори для можливості обслуговування багатьма паралельними пристроями та утворення розгалужень.

```
==================== Завдання 5 та 6: Складна модель ====================
Final statistics:
DISPOSE:
Num disposed: 1544.

PROCESS_3:
Num processed: 1715. Mean queue size: 0.48200. Mean busy handlers: 1.73308. Mean wait time: 0.28105. Failure p
robability: 0.47783

PROCESS_1:
Num processed: 4086. Mean queue size: 6.68985. Mean busy handlers: 4.94182. Mean wait time: 1.63728. Failure p
robability: 0.19620

CREATE:
Num created: 4930.

PROCESS_2:
Num processed: 3291. Mean queue size: 4.79748. Mean busy handlers: 6.80512. Mean wait time: 1.45778. Failure p
robability: 0.19109
```

Рисунок 4 – Підсумкові статистичні дані складної моделі

Таблиця 3 – Результати верифікації

Елемент	**Кількість оброблених (або створених) заявок**	**Середній розмір черги**	**Середнє завантаження пристроїв**	**Середній час очікування**	**Ймовірність відмови**
CREATE	4930	-	-	-	-
PROCESS_1	4086	6.68985	4.94182	1.63728	0.19620
PROCESS_2	3291	4.79748	6.80512	1.45778	0.19109
PROCESS_3	1715	0.48200	1.73308	0.28105	0.47783
DISPOSE	1544	-	-	-	-

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:

- **PROCESS_1** отримав 5 паралельних каналів обслуговування, що значно зменшило середню довжину черги (з 9.8 до 6.69) та час очікування (з ≈11.7 до ≈1.63). Це свідчить про успішне розвантаження елемента завдяки більшій кількості обробників, які дозволяють обробляти заявок набагато більше за одиницю часу.

- **PROCESS_2** при 7 паралельних каналах демонструє подібну тенденцію із зменшенням черги та часу очікування (черга скорочується до ≈4.8, час до ≈1.46), що також підтверджує покращення пропускної здатності.
- **PROCESS_3** – хоча кількість оброблених заявок значно зросла (з 418 до 1715), тут впроваджено додатковий механізм зворотного зв'язку: 10% заявок повертаються до PROCESS_1. Результатом є дещо підвищена середня довжина черги і збільшення ймовірності відмов (з ≈0.14 до ≈0.48). 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

У ході виконання комп'ютерного практикуму було розроблено програмну реалізацію системи імітаційного моделювання з використанням об'єктно-орієнтованого підходу. Реалізація базових класів (CreateElement, ProcessElement, DisposeElement та Model) дозволила гнучко конструювати моделі різної складності та проводити їх детальний аналіз.

Робота складалася з двох ключових етапів, результати яких наведено нижче:

1. **Побудова та верифікація лінійної моделі (Завдання 1–4):**

   Були реалізовані $Create \to P1$ та $Create \to P1 \to P2 \to P3 \to Dispose$ лінійні схеми обслуговування. Проведення верифікації (Завдання 4) шляхом варіювання інтенсивності вхідного потоку підтвердило

адекватність роботи програмного коду. Експериментально встановлено, що система втрачає стабільність при наближенні інтервалу надходження заявок до часу обслуговування (Delay ≈ 1), що призводить до експоненційного зростання черг та різкого збільшення ймовірності відмов (до 53% за Delay = 0.5). Це довело коректність роботи механізмів черг та збору статистики.

2. **Модифікація системи: багатоканальність та зворотні зв'язки (Завдання 5–6):**

Перехід до модифікованої моделі продемонстрував ефективність введення паралельної обробки:

- **Вплив багатоканальності (Task 5):** Впровадження параметру num_handlers (5 каналів для P1 та 7 для P2) дозволило ліквідувати "вузькі місця", які спостерігалися в лінійній моделі. Навіть при високому навантаженні черги на P1 та P2 суттєво зменшилися, а час очікування скоротився, що підвищило загальну пропускну здатність системи.
- **Маршрутизація та зворотний зв'язок (Task 6):** Реалізація ймовірнісної передачі заявок (повернення 10% від P3 до P1) ускладнила динаміку системи. Аналіз показав, що PROCESS_3, маючи лише два канали обслуговування та обмежену чергу (L=1), став найбільш чутливим елементом системи, де спостерігалося зростання відмов.

Отримані результати свідчать, що просте збільшення кількості обслуговуючих пристроїв є потужним, але не завжди достатнім методом оптимізації. Введення зворотних зв'язків створює додаткове навантаження на систему (ефект рециркуляції заявок), що вимагає ретельного балансування параметрів черг та кількості каналів для уникнення втрат заявок. Розроблена

об’єктно-орієнтована архітектура довела свою ефективність, дозволяючи легко масштабувати модель та імплементувати складну логіку маршрутизації без зміни ядра симулятора.