

Міністерство освіти і науки України

Національний технічний університет України „КПІ імені Ігоря Сікорського ”

Факультет інформатики та обчислювальної техніки

Кафедра інформаційних систем та технологій

Звіт до комп’ютерного практикуму № 4

З дисципліни «Моделювання систем»

Прийняв: Виконав:

ст. викл. Дифучин А.Ю. Студент 4 курсу,гр. ІП-13

Петров Ігор Ярославович

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_2024 р.

2024 р.

**Завдання**

1. Розробити модель масового обслуговування, яка складається з N систем масового обслуговування. Число N є параметром моделі. Кількість подій в моделі оцінюється числом N+1. 20 балів.
2. Виконати експериментальну оцінку складності алгоритму імітації мережі масового обслуговування. Для цього виконайте серію експериментів, в якій спостерігається збільшення часу обчислення алгоритму імітації при збільшенні кількості подій в моделі. 40 балів.
3. Виконати теоретичну оцінку складності побудованого алгоритму імітації. 30 балів.
4. Повторіть експеримент при зміні структури мережі масового обслуговування. 10 балів.

**Виконання**

**Завдання 1**

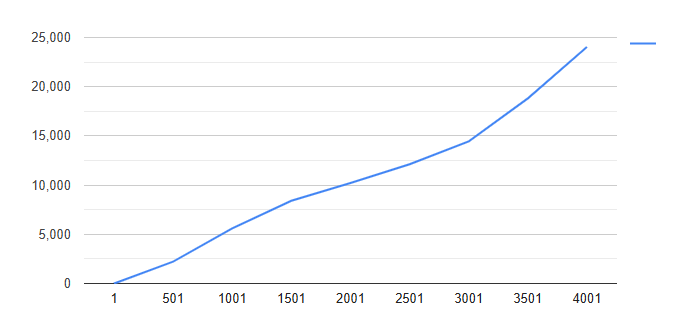
За основу було взято модифікований код з другої лабораторної роботи. Було розроблено модель масового обслуговування, яка складається з N систем масового обслуговування.

import java.util.ArrayList;  
  
public class Main {  
 public static void main(String[] args) {  
 *seq*();  
 }  
  
 public static void seq(){  
 final int N = 4001;  
 final int iterations = 3;  
  
 for (int processNum = 1; processNum <= N; processNum += 500) {  
 long totalTime = 0;  
  
 for (int currIter = 0; currIter < iterations; currIter++) {  
  
 Element.*resetNextId*();  
 ArrayList<Element> elements = new ArrayList<>();  
  
  
  
 Create creator = new Create(5.0);  
 creator.setName("CREATOR");  
 creator.setDistribution("exp");  
 elements.add(creator);  
  
  
  
 Process lastProcess = null;  
 for (int i = 0; i < processNum; i++) {  
 Process newProcess = new Process(1.0, 1);  
 newProcess.setName("PROCESSOR " + (i + 1));  
 newProcess.setDistribution("exp");  
 elements.add(newProcess);  
 if (lastProcess != null) {  
 lastProcess.addNextElement(newProcess);  
 } else {  
 creator.addNextElement(newProcess);  
 }  
 lastProcess = newProcess;  
 }  
  
  
  
 Model model = new Model(elements);  
 long start = System.*currentTimeMillis*();  
 model.simulate(1000);  
 long end = System.*currentTimeMillis*();  
 totalTime += (end - start);  
 }  
 System.*out*.printf("time with %d processes: %d ms%n", processNum, totalTime / iterations);  
 }  
 }  
}

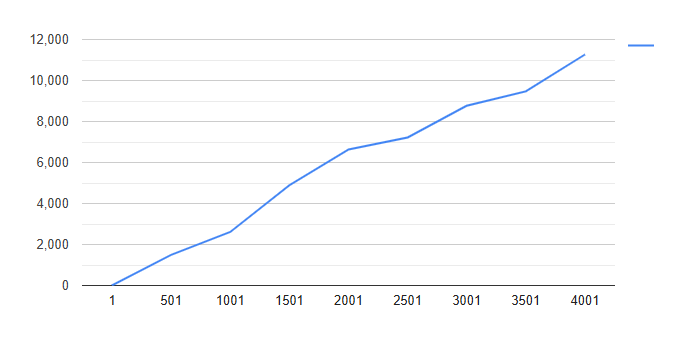
**Завдання 2**

Буде проведено по 3 заміри та взято середнє значення швидкості симуляції. Максимальна кількість процесів 4001 із кроком в 500 процесів.

Послідовна модель з часовою затримкою генерації запитів 2 та часовою затримкою обробки запитів 1:



Послідовна модель з часовою затримкою генерації запитів 5 та часовою затримкою обробки запитів 1:



При більш розвантаженії моделі за рахунок параметрів з часовою затримкою генерації запитів 5 та часовою затримкою обробки запитів 1, час симуляції значно знижений. Це пояснюється меншою кількістю запитів, що проходять через систему під час симуляції, а звідси і меншу навантаженість на систему.

**Завдання 3**

**Генерація запитів:**

Складність: O(1) часу.

**Обробка запитів:**

**Послідовна структура:**

Складність обробки запиту для одного процесу — O(1).

Загальна складність обробки одного запиту в системі — O(N).

**Паралельна структура:**

Процеси розділені на дві підгрупи (N/2).

Запити обробляються паралельно, але сумарна кількість обробок залишається такою самою.

**Симуляція:**

**Послідовна:**

Зі збільшенням кількості процесів N, загальна складність симуляції зростає лінійно: O(T × N) де T — кількість запитів, N — кількість процесів.

**Паралельна:**

Обробка запитів розподіляється між двома гілками, але сумарна кількість обробок залишається такою самою.

Загальна складність також залишається: O(T × N) де T — кількість запитів, N — кількість процесів.

**Завдання 4**

Змінено структуру мережі масового обслуговування наступним чином. Обробки запитів будуть здійснюватися двома гілками процесів.

import java.util.ArrayList;  
  
public class Main {  
 public static void main(String[] args) {  
 *paralell*();  
 }  
 public static void paralell(){  
 final int N = 4001;  
 final int iterations = 3;  
 for (int processNum = 1; processNum <= N; processNum += 500) {  
 long totalTime = 0;  
  
 for (int currIter = 0; currIter < iterations; currIter++) {  
 Element.*resetNextId*();  
 ArrayList<Element> elements = new ArrayList<>();

Create creator = new Create(5.0);  
 creator.setName("CREATOR");  
 creator.setDistribution("exp");  
 elements.add(creator);

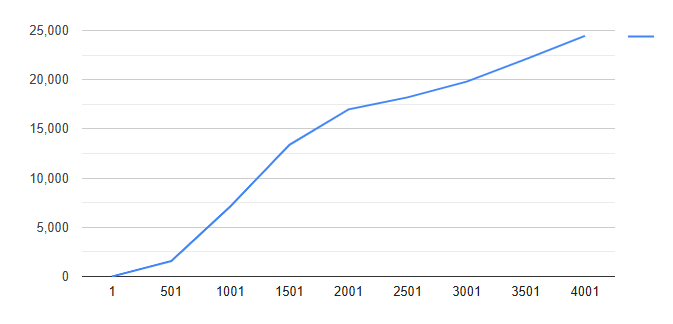
ArrayList<Process> branch1 = new ArrayList<>();  
 ArrayList<Process> branch2 = new ArrayList<>();

for (int i = 0; i < processNum; i++) {  
 Process newProcess = new Process(1.0, 1);  
 newProcess.setName("PROCESS " + (i + 1));  
 newProcess.setDistribution("exp");  
 elements.add(newProcess);  
  
 if (i % 2 == 0) {  
 branch1.add(newProcess);  
 } else {  
 branch2.add(newProcess);  
 }  
 }

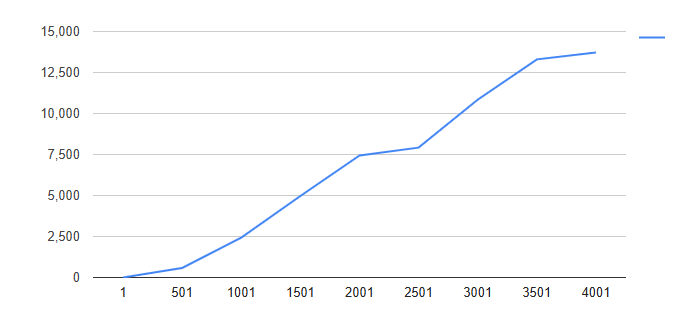
//branch1  
 for (int i = 0; i < branch1.size(); i++) {  
 Process currentProcess = branch1.get(i);  
 if (i == 0) {  
 creator.addNextElement(currentProcess);  
 } else {  
 branch1.get(i - 1).addNextElement(currentProcess);  
 }  
 }  
 //branch2  
 for (int i = 0; i < branch2.size(); i++) {  
 Process currentProcess = branch2.get(i);  
 if (i == 0) {  
 creator.addNextElement(currentProcess);  
 } else {  
 branch2.get(i - 1).addNextElement(currentProcess);  
 }  
 }

Model model = new Model(elements);  
 long start = System.*currentTimeMillis*();  
 model.simulate(1000);  
 long end = System.*currentTimeMillis*();  
 totalTime += (end - start);  
 }  
 System.*out*.printf("time with %d processes: %d ms%n", processNum, totalTime / iterations);  
 }  
 }  
}

Паралельна модель з часовою затримкою генерації запитів 2 та часовою затримкою обробки запитів 1:



Паралельна модель з часовою затримкою генерації запитів 5 та часовою затримкою обробки запитів 1:



При більш розвантаженії моделі за рахунок параметрів з часовою затримкою генерації запитів 5 та часовою затримкою обробки запитів 1, час симуляції значно знижений. Це пояснюється меншою кількістю запитів, що проходять через систему під час симуляції, а звідси і меншу навантаженість на систему, аналогічно із послідовною структурою моделі.

Паралельна структура не має значною переваги в часу симуляції із послідовною структурою, як і було передбачено в теоретичних розрахунках, це пояснюється тим, що обробка запитів розподіляється між двома гілками, але сумарна кількість обробок залишається такою самою. Загальна складність також залишається: O(T × N). Тобто маємо лінійну залежність, так само як і в послідовній структурі.

**Виконання**

Виконана робота дозволила розробити модель масового обслуговування, яка демонструє лінійну залежність часу симуляції від кількості процесів у системі. Зміна топології мережі не призвела до суттєвих покращень продуктивності. Експериментальні результати переконливо підтверджують теоретичні розрахунки.