

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 №2

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**

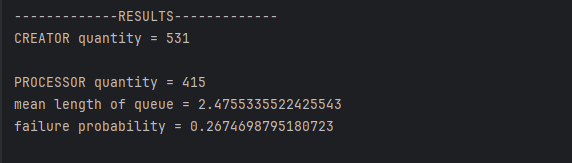
1. Реалізувати алгоритм імітації простої моделі обслуговування одним пристроєм з використанням об’єктно-орієнтованого підходу. **5 балів.**
2. Модифікувати алгоритм, додавши обчислення середнього завантаження пристрою.**5 балів.**
3. Створити модель за схемою, представленою на рисунку. **30 балів.**Зображення, що містить схема, ряд, План, знімок екрана

   Автоматично згенерований опис
4. Виконати верифікацію моделі, змінюючи значення вхідних змінних та параметрів моделі. Навести результати верифікації у таблиці.**10 балів.**
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.**

**Виконання**

**Завдання 1**

Для цього завдання використовується код з лістингу, просто запустимо його і переглянемо результати.



З результатів можна отримати середню чергу і шанс відмови.

**Завдання 2**

Для модифікації необхідно додати нове приватне поле у Process з часом роботи:

private double meanQueue, totalBusyTime;

Також модифікувати функцію doStatistics() для додавання часу затримки, коли процес зайнятий.

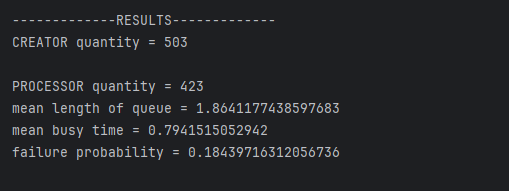
@Override  
public void doStatistics(double delta) {  
 meanQueue = getMeanQueue() + queue \* delta;  
 if (getState() == 1) {  
 totalBusyTime += delta;  
 }  
}

Також для відображення у Model додамо функцію для отримання середнього часу затримок додамо функцію

public double getMeanBusyTime(double totalTime) {  
 return totalBusyTime / totalTime;  
}

І відобразимо її у Model  
  
public void printResult() {  
 System.*out*.println("\n-------------RESULTS-------------");  
 for (Element e : list) {  
 e.printResult();  
 if (e instanceof Process p) {  
 System.*out*.println("mean length of queue = " +  
 p.getMeanQueue() / tcurr  
 + "\nmean busy time = " +  
 p.getMeanBusyTime(tcurr)  
 + "\nfailure probability = " +  
 p.getFailure() / (double) p.getQuantity());  
 }  
 System.*out*.println();  
 }  
}

В результаті отримаємо результати з середнім показником навантаженності.



**Завдання 3**

Трохи модифікуємо inAct у процесі, щоб можна було переходити від процеса до процеса.

if (super.getNextElement() != null) {  
 super.getNextElement().inAct();  
}

Також модифікуємо Main додатковими двома процесами.

public static void main(String[] args) {  
 Create c = new Create(2.0);  
 Process p1 = new Process(2.0);  
 Process p2 = new Process(2.0);  
 Process p3 = new Process(2.0);  
 c.setNextElement(p1);  
 p1.setNextElement(p2);  
 p2.setNextElement(p3);  
 p1.setMaxqueue(5);  
 p2.setMaxqueue(5);  
 p3.setMaxqueue(5);  
 c.setName("CREATOR");  
 p1.setName("PROCESSOR1");  
 p2.setName("PROCESSOR2");  
 p3.setName("PROCESSOR3");  
 c.setDistribution("exp");  
 p1.setDistribution("exp");  
 p2.setDistribution("exp");  
 p3.setDistribution("exp");  
 ArrayList<Element> list = new ArrayList<>();  
 list.add(c);  
 list.add(p1);  
 list.add(p2);  
 list.add(p3);  
 Model model = new Model(list);  
 model.simulate(1000.0);

**Завдання 4**

Для верифікації моделі побудуємо таблицю з різними варіантами вхідних параметрів для елементів ММО.

**Завдання 5 - 6**

Для виконання цих завдань необхідно практично повністю переписати процес.

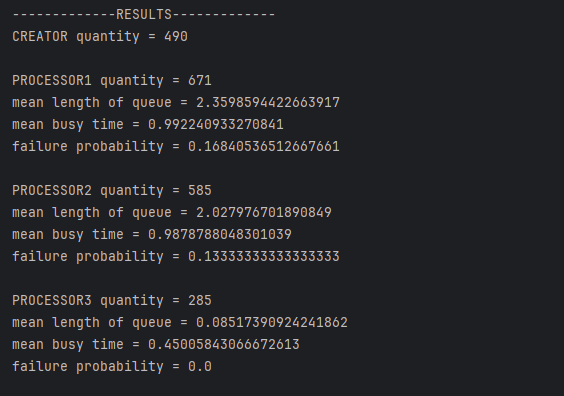
import java.util.ArrayList;  
import java.util.Random;  
public class Process extends Element {  
 private int queue, maxqueue, failure;  
 private double meanQueue, busyTime;  
 private final int machinesNum;  
 private final double[] nextMachines;  
 private ArrayList<Element> nextElements;  
 public Process(double delay, int machinesNum) {  
 super(delay);  
 queue = 0;  
 maxqueue = Integer.*MAX\_VALUE*;  
 meanQueue = 0.0;  
 busyTime = 0.0;  
 nextElements = new ArrayList<>();  
 this.machinesNum = machinesNum;  
 this.nextMachines = new double[machinesNum];  
 for (int i = 0; i < machinesNum; i++) {  
 nextMachines[i] = Double.*MAX\_VALUE*;  
 }  
 }  
 @Override  
 public void inAct() {  
 int freeMachine = getFreeMachine();  
 if (freeMachine != -1) {  
 nextMachines[freeMachine] = super.getTcurr() + super.getDelay();  
 super.setState(1);  
 } else {  
 if (getQueue() < getMaxqueue()) {  
 setQueue(getQueue() + 1);  
 } else {  
 failure++;  
 }  
 }  
 double minTnext = Double.*MAX\_VALUE*;  
 for (double tnext : nextMachines) {  
 if (tnext < minTnext) {  
 minTnext = tnext;  
 }  
 }  
 super.setTnext(minTnext);  
 }  
 @Override  
 public void outAct() {  
 super.outAct();  
 double minValue = Double.*MAX\_VALUE*;  
 int nextMachine = -1;  
 for (int i = 0; i < machinesNum; i++) {  
 if (nextMachines[i] < minValue) {  
 minValue = nextMachines[i];  
 nextMachine = i;  
 }  
 }  
 if (nextMachine != -1) {  
 nextMachines[nextMachine] = Double.*MAX\_VALUE*;  
 if (getQueue() > 0) {  
 setQueue(getQueue() - 1);  
 nextMachines[nextMachine] = super.getTcurr() + super.getDelay();  
 } else {  
 boolean isAllFree = true;  
 for (double tnext : nextMachines) {  
 if (tnext != Double.*MAX\_VALUE*) {  
 isAllFree = false;  
 break;  
 }  
 }  
 if (isAllFree) {  
 super.setState(0);  
 }  
 }  
 if (!nextElements.isEmpty()) {  
 Element nextElement = getNextRandomElement();  
 if (nextElement != null) {  
 nextElement.inAct();  
 }  
 }  
 }  
 double minTnext = Double.*MAX\_VALUE*;  
 for (double tnext : nextMachines) {  
 if (tnext < minTnext) {  
 minTnext = tnext;  
 }  
 }  
 super.setTnext(minTnext);  
 }  
 public int getFailure() {  
 return failure;  
 }  
 public int getQueue() {  
 return queue;  
 }  
 public void setQueue(int queue) {  
 this.queue = queue;  
 }  
 public int getMaxqueue() {  
 return maxqueue;  
 }  
 public void setMaxqueue(int maxqueue) {  
 this.maxqueue = maxqueue;  
 }  
 @Override  
 public void printInfo() {  
 super.printInfo();  
 System.*out*.println("failure = " + this.getFailure());  
 }  
 @Override  
 public void doStatistics(double delta) {  
 meanQueue = getMeanQueue() + queue \* delta;  
 if (getState() == 1) {  
 busyTime += delta;  
 }  
 }  
 public double getMeanQueue() {  
 return meanQueue;  
 }  
 public double getMeanBusyTime(double totalTime) {  
 return busyTime / totalTime;  
 }  
 private int getFreeMachine() {  
 int freeI = -1;  
 for (int i = 0; i < machinesNum; i++) {  
 if (nextMachines[i] == Double.*MAX\_VALUE*) {  
 return i;  
 }  
 }  
 return freeI;  
 }  
 private Element getNextRandomElement() {  
 return nextElements.get(new Random().nextInt(nextElements.size()));  
 }  
 public void addNextElement(Element element) {  
 nextElements.add(element);  
 }  
}

Було додано переходи між процесами від p2 назад p1 або p3 за допомогою addNextElement() і ArrayList<Element> nextElements. Процес на який відбудеться переход випадково обирається за допомогою getNextRandomElement(). Також для створення декількох машин всередині одного процесу було модифіковано inAct() і outAct(), а також додано поле machines, яке заповнюється при ініціалізації. Також було модифіковано Main.

public static void main(String[] args) {  
 Create c = new Create(2.0);  
 Process p1 = new Process(2.0, 2);  
 Process p2 = new Process(2.0, 2);  
 Process p3 = new Process(2.0, 2);  
 c.setNextElement(p1);  
 p1.addNextElement(p2);  
 p2.addNextElement(p1);  
 p2.addNextElement(p3);  
 p1.setMaxqueue(5);  
 p2.setMaxqueue(5);  
 p3.setMaxqueue(5);  
 c.setName("CREATOR");  
 p1.setName("PROCESSOR1");  
 p2.setName("PROCESSOR2");  
 p3.setName("PROCESSOR3");  
 c.setDistribution("exp");  
 p1.setDistribution("exp");  
 p2.setDistribution("exp");  
 p3.setDistribution("exp");  
 ArrayList<Element> list = new ArrayList<>();  
 list.add(c);  
 list.add(p1);  
 list.add(p2);  
 list.add(p3);  
 Model model = new Model(list);  
 model.simulate(1000);  
}

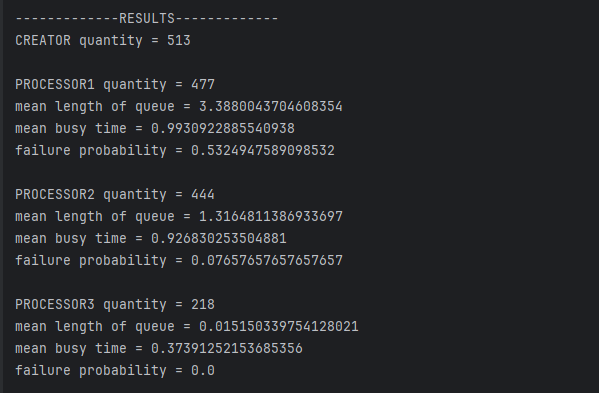
Результати при

Create c = new Create(2.0);  
Process p1 = new Process(4.0, 3);  
Process p2 = new Process(6.0, 4);  
Process p3 = new Process(2.0, 2);



Результати при

Create c = new Create(2.0);  
Process p1 = new Process(6.0, 3);  
Process p2 = new Process(5.0, 3);  
Process p3 = new Process(2.0, 2);



**Висновок**

У ході виконання комп'ютерного практикуму була створена модель обслуговування за допомогою об’єктно-орієнтованого підходу. Модель було вдосконалено для виведення більш детальної статистики та додано додаткові елементи із різними типами зв’язків між ними. Модель також пройшла перевірку на різних наборах вхідних даних, і результати цієї перевірки були зведені в таблицю.