НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. ІГОРЯ СІКОРСЬКОГО”

КАФЕДРА АВТОМАТИЗОВАНИХ СИСТЕМ ОБРОБКИ ІНФОРМАЦІЇ І УПРАВЛІННЯ

Комп’ютерний практикум № 4

з дисципліни

“Моделювання систем”

Виконав:

студент групи ІТ-04

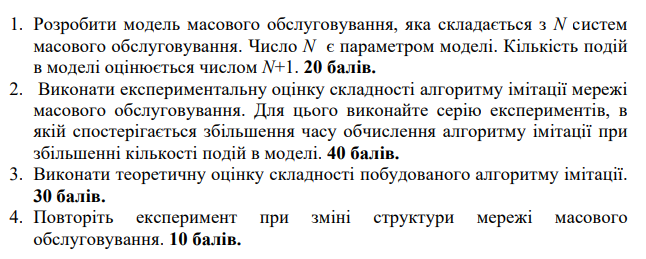
Чапча Святослав

Перевірив:

Дифучин А.Ю.

Київ-2023

**Завдання**

****

Повну версію коду можна знайти за посиланням:   
<https://github.com/whitetark/modelling-systems>

**Програмний код**

**Main.java**

import java.util.\*;  
  
public class Main {  
  
 public static void main(String[] args) {  
 Map<Integer, Double> results = new HashMap<>();  
 int k = 5;  
  
 for (int N = 10; N <= 300; N += 10) {  
 double[] Nresults = new double[k];  
  
 for (int i = 0; i < k; i++) {  
 Model model = *createOneChainModel*(N);  
  
 long startTime = System.*currentTimeMillis*();  
  
 model.simulate(50000);  
  
 long endTime = System.*currentTimeMillis*();  
 Nresults[i] = endTime - startTime;  
 }  
  
 double averageTime = *calculateAverage*(Nresults);  
 results.put(N, averageTime);  
 }  
  
 *printResults*(results);  
 }  
  
 public static Model createTwoChainModel(int N) {  
 Create create = new Create(1, "Create");  
 create.setDistributionType(DistributionType.*EXPONENTIAL*);  
 List<NextEvent> nextEventList = new ArrayList<>();  
 List<MultiTaskProcessor> multiTaskProcessors1 = new ArrayList<>();  
 List<MultiTaskProcessor> multiTaskProcessors2 = new ArrayList<>();  
 Process Process1 = new Process(1, "Process1");  
 Process1.setDistributionType(DistributionType.*EXPONENTIAL*);  
  
 for (int i = 0; i < N; i++) {  
 Process process = new Process(1, "Process" + i);  
 process.setDistributionType(DistributionType.*EXPONENTIAL*);  
  
 MultiTaskProcessor multiTaskProcessor = new MultiTaskProcessor(List.*of*(process), "MultiTaskProcessor" + i, Integer.*MAX\_VALUE*);  
  
 NextEvent hostsNext = new NextEvent(multiTaskProcessor, 1);  
 nextEventList.add(hostsNext);  
  
 multiTaskProcessors1.add(multiTaskProcessor);  
 }  
 for (int i = 0; i < N; i++) {  
 Process process = new Process(1, "Process" + i);  
 process.setDistributionType(DistributionType.*EXPONENTIAL*);  
  
 MultiTaskProcessor multiTaskProcessor = new MultiTaskProcessor(List.*of*(process), "MultiTaskProcessor" + i, Integer.*MAX\_VALUE*);  
  
 NextEvent hostsNext = new NextEvent(multiTaskProcessor, 1);  
 nextEventList.add(hostsNext);  
  
 multiTaskProcessors2.add(multiTaskProcessor);  
 }  
 create.setNextEvent(new NextEvents(nextEventList, NextEventsType.*PRIORITY*));  
  
 for (int i = 0; i < N; i++) {  
 Process1.setNextEvent(new NextEvents(List.*of*(new NextEvent(multiTaskProcessors1.get(i), 1)), NextEventsType.*PRIORITY*));  
 multiTaskProcessors1.get(i).setNextEvent(new NextEvents(List.*of*(new NextEvent(multiTaskProcessors2.get(i), 1)), NextEventsType.*PRIORITY*));  
 }  
  
 multiTaskProcessors1.addAll(multiTaskProcessors2);  
 return new Model(create, multiTaskProcessors1);  
 }  
  
 public static Model createOneChainModel(int N) {  
 Create create = new Create(1, "Create");  
 create.setDistributionType(DistributionType.*EXPONENTIAL*);  
 List<NextEvent> nextEventList = new ArrayList<>();  
 List<MultiTaskProcessor> multiTaskProcessors = new ArrayList<>();  
 for (int i = 0; i < N; i++) {  
 Process process = new Process(1, "Process" + i);  
 process.setDistributionType(DistributionType.*EXPONENTIAL*);  
  
 MultiTaskProcessor multiTaskProcessor = new MultiTaskProcessor(List.*of*(process), "MultiTaskProcessor" + i, Integer.*MAX\_VALUE*);  
  
 NextEvent hostsNext = new NextEvent(multiTaskProcessor, 1);  
 nextEventList.add(hostsNext);  
  
 multiTaskProcessors.add(multiTaskProcessor);  
 }  
 create.setNextEvent(new NextEvents(nextEventList, NextEventsType.*PRIORITY*));  
 return new Model(create, multiTaskProcessors);  
 }

**MultiTaskProcessor.java**

import java.util.\*;  
  
public class MultiTaskProcessor extends Event {  
 private List<Process> processes;  
 protected int totalCustomersExited = 0;  
 protected double totalExitTime = 0.0;  
 protected double lastExitTime = 0.0; // Останній час виходу клієнта  
 protected double totalEnterTimeStart = 0.0;  
 protected double totalEnterTimeEnd = 0.0;  
 protected int queue, maxQueue;  
 public MultiTaskProcessor(List<Process> processes, String name) {  
 super(name);  
 this.processes = processes;  
 setEventTime();  
 }  
 public MultiTaskProcessor(List<Process> processes, String name, int maxQueue) {  
 super(name);  
 this.processes = processes;  
 this.maxQueue = maxQueue;  
 setEventTime();  
 }  
 @Override  
 public void inAct(double currentTime) {  
 super.inAct(currentTime);  
 Process process = getFreeProcess();  
 if (process != null) {  
 totalEnterTimeStart += currentTime;  
 process.outAct(currentTime);  
 setEventTime();  
 } else {  
 if(this.queue < this.maxQueue) {  
 this.queue += 1;  
 }  
 else {  
 this.failure++;  
 }  
 }  
 }  
 @Override  
 public void outAct(double currentTime) {  
 super.outAct(currentTime);  
 Process process = getThisProcess(currentTime);  
 if (process != null) {  
 process.setState(0);  
 process.setEventTime(Double.*MAX\_VALUE*);  
 setEventTime();  
 if (this.queue > 0) {  
 process.outAct(currentTime);  
 this.queue -= 1;  
 setEventTime();  
 }  
  
 double exitTime = currentTime; // Час виходу клієнта  
 totalCustomersExited++;  
 totalExitTime += exitTime - lastExitTime; // Різниця між поточним та попереднім часом виходу  
 lastExitTime = exitTime; // Оновлення останнього часу виходу  
 totalEnterTimeEnd += currentTime;  
 }  
 }  
  
  
 private Process getFreeProcess() {  
 for (Process process : processes) {  
 if (process.state == 0) {  
 return process;  
 }  
 }  
 return null;  
 }  
  
 private Process getThisProcess(double currentTime) {  
 for (Process process : processes) {  
 if (process.eventTime == currentTime) {  
 return process;  
 }  
 }  
 return null;  
 }  
  
 private void setEventTime() {  
 this.eventTime = Collections.*min*(processes.stream().map(process -> process.eventTime).toList());  
 }  
  
 @Override  
 public void setNextEvent(NextEvents element) {  
 for (Process process : processes) {  
 process.setNextEvent(element);  
 }  
 }

**Результати виконання програми**

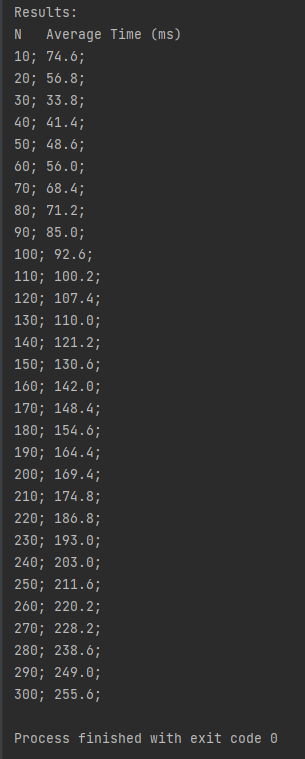
****

Рис. 1 – Результат виконання програми

**Порівняльна таблиця**

|  |  |  |
| --- | --- | --- |
| **N** | One Chain | Two Chain |
| **10** | 119.8 | 74.6 |
| **20** | 64.4 | 56.8 |
| **30** | 83.8 | 33.8 |
| **40** | 108.8 | 41.4 |
| **50** | 133.0 | 48.6 |
| **60** | 153.6 | 56.0 |
| **70** | 174.8 | 68.4 |
| **80** | 205.0 | 71.2 |
| **90** | 254.0 | 85.0 |
| **100** | 277.6 | 92.6 |
| **110** | 279.0 | 100.2 |
| **120** | 315.2 | 107.4 |
| **130** | 334.8 | 110.0 |
| **140** | 373.8 | 121.2 |
| **150** | 384.6 | 130.6 |
| **160** | 415.0 | 142.0 |
| **170** | 475.2 | 148.2 |
| **180** | 492.0 | 154.6 |
| **190** | 532.8 | 164.4 |
| **200** | 553.0 | 169.4 |
| **210** | 551.4 | 174.8 |
| **220** | 615.4 | 186.8 |
| **230** | 601.6 | 193.0 |
| **240** | 626.2 | 203.0 |
| **250** | 638.2 | 211.6 |
| **260** | 675.6 | 220.2 |
| **270** | 717.0 | 228.2 |
| **280** | 720.0 | 238.6 |
| **290** | 733.0 | 249.0 |
| **300** | 805.0 | 255.6 |

Як можемо побачити, при зміні структури мережі масового обслуговування, результати покращились. Це ми можемо побачити і на рисунку.

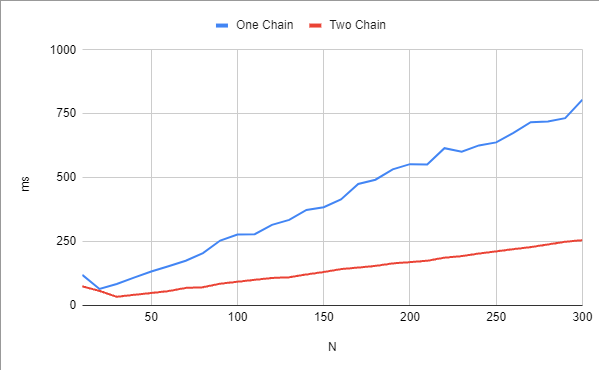
****

Рис. 2 – Діаграма результатів

**ВИСНОВОК**

Модель з послідовно з’єднаними СМО наближається до теоретичних очікувань, а модель з паралельно з’єднаними СМО затрачає менше часу при великій кількості смо, оскільки кожна заявка вимагає тільки обробки однією смо, що зменшує кількість подій виходу з обробки смо.