Міністерство освіти і науки України

Національний технічний університет України

«Київський політехнічний інститут»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

**Лабораторна робота №3**

з дисципліни «[Організація обчислювальних процесів](http://wiki.kpi.ua/index.php/%D0%9E%D1%80%D0%B3%D0%B0%D0%BD%D1%96%D0%B7%D0%B0%D1%86%D1%96%D1%8F%20%D0%BE%D0%B1%D1%87%D0%B8%D1%81%D0%BB%D1%8E%D0%B2%D0%B0%D0%BB%D1%8C%D0%BD%D0%B8%D1%85%20%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%96%D0%B2_(20402040))»

на тему «Дослідження дисциплін обслуговування заявок при обмежених ресурсах»

Виконав:

студент ІІI курсу

групи ІО-53

Лісовий В. О.

Перевірив:

[ст.вик. Сімоненко А. В.](http://rozklad.kpi.ua/Schedules/ViewSchedule.aspx?v=7507d508-d0a2-4c01-a858-a063f8d724cc)

Київ – 2018

**Сирцевий код**

**SFQueue**

**package** lab3.model;  
  
**import** java.util.Comparator;  
**import** java.util.LinkedList;  
  
**public class** SFQueue **implements** Queue<Task> {  
  
 **private** LinkedList<Task> tasks;  
  
 **public** SFQueue() {  
 **this**.tasks = **new** LinkedList<>();  
 }  
  
 @Override  
 **public void** add(Task task) {  
 tasks.add(task);  
 tasks.sort(Comparator.comparingDouble(Task::getSolveTime));  
 }  
  
 @Override  
 **public** Task get() {  
 **return** tasks.pop();  
 }  
  
 @Override  
 **public boolean** isEmpty() {  
 **return** tasks.isEmpty();  
 }  
  
 @Override  
 **public** LinkedList<Task> getTasks() {  
 **return** tasks;  
 }  
}

**Task**

**package** lab3.model;  
  
**class** Task {  
 **private double inTime**;  
 **private double solveTime**;  
 **private double onSolving**;  
 **private double outTime**;  
  
 **public** Task(**double** enterTime, **double** solveTime) {  
 **this**.**inTime** = enterTime;  
 **this**.**solveTime** = solveTime;  
 }  
  
 **public double** getInTime() {  
 **return inTime**;  
 }  
  
 **public void** setInTime(**double** inTime) {  
 **this**.**inTime** = inTime;  
 }  
  
 **public double** getSolveTime() {  
 **return solveTime**;  
 }  
  
 **public void** setSolveTime(**double** solveTime) {  
 **this**.**solveTime** = solveTime;  
 }  
  
 **public double** getOnSolving() {  
 **return onSolving**;  
 }  
  
 **public void** setOnSolving(**double** onSolving) {  
 **this**.**onSolving** = onSolving;  
 }  
  
 **public double** getOutTime() {  
 **return outTime**;  
 }  
  
 **public void** setOutTime(**double** outTime) {  
 **this**.**outTime** = outTime;  
 }  
  
}

**Model**

**package** lab3.model;  
  
**import** org.springframework.beans.factory.annotation.Value;  
**import** org.springframework.stereotype.Component;  
  
**import** java.util.ArrayList;  
**import** java.util.Random;  
  
@Component  
**public class** Model {  
  
 **private double** LENGTH;  
 **private double** LAMBDA;  
 **private double** MU;  
  
 **private double** T;  
 **private** ArrayList<Task> solvedTasks;  
 **private** Queue<Task> queue;  
  
 **private boolean** isFree = **true**;  
 **private double** t1 = 0;  
 **private double** t2 = Double.MAX\_VALUE;  
 **private double** tp;  
 **private int** k = 1;  
 **private** Task currTask = **null**;  
  
 **public** Model(@Value(**"${model.LENGTH}"**) **double** LENGTH, @Value(**"${model.LAMBDA}"**) **double** LAMBDA, @Value(**"${model.MU}"**) **double** MU) {  
 **this**.LENGTH = LENGTH;  
 **this**.LAMBDA = LAMBDA;  
 **this**.MU = MU;  
 **this**.queue = **new** SFQueue();  
 **this**.T = 0;  
 **this**.solvedTasks = **new** ArrayList<>();  
 }  
  
 **public void** step() {  
 tp = getSolveTime();  
 **if** (t1 < t2) {  
 T = t1;  
 **if** (isFree) {  
 currTask = **new** Task(t1, tp);  
 currTask.setOnSolving(T);  
 t2 = T + tp;  
 currTask.setOutTime(t2);  
 isFree = **false**;  
 } **else** {  
 queue.add(**new** Task(t1, tp));  
 }  
 t1 = genNextEnterTime();  
 k++;  
 } **else** {  
 T = t2;  
 solvedTasks.add(currTask);  
 **if** (queue.isEmpty()) {  
 t2 = Double.MAX\_VALUE;  
 isFree = **true**;  
 } **else** {  
 currTask = queue.get();  
 currTask.setOnSolving(T);  
 t2 = T + currTask.getSolveTime();  
 currTask.setOutTime(t2);  
 isFree = **false**;  
 }  
 }  
 }  
  
 **public void** simulate() {  
 **while** (**k** < **LENGTH**) {  
 step();  
 }  
 }  
  
  
 **public double** getEvTime() {  
 **return solvedTasks**.stream().mapToDouble(task -> task.getOutTime() - task.getInTime()).sum() / **solvedTasks**.size();  
 }  
  
 **public double** getReactionTime() {  
 **return** solvedTasks.stream().mapToDouble(task -> task.getOnSolving() - task.getInTime()).sum() / solvedTasks.size();  
 }  
  
 **public double** getRatio() {  
 **return** (**double**) solvedTasks.size() / (solvedTasks.size() + queue.getTasks().size());  
 }  
  
  
 *//Count solve time for task* **private double** getSolveTime() {  
 Random r = **new** Random();  
 **return** - 1 / MU \* Math.log(r.nextDouble());  
 }  
  
  
 *//Count the time when the next task comes into the system* **private double** genNextEnterTime() {  
 Random r = **new** Random();  
 **return** T - 1 / LAMBDA \* Math.log(r.nextDouble());  
 }  
  
 **public double** getLENGTH() {  
 **return** LENGTH;  
 }  
  
 **public double** getLAMBDA() {  
 **return** LAMBDA;  
 }  
  
 **public double** getMU() {  
 **return** MU;  
 }  
  
 **public double** getT() {  
 **return** T;  
 }  
  
 **public** ArrayList<Task> getSolvedTasks() {  
 **return** solvedTasks;  
 }  
  
 **public** Queue<Task> getQueue() {  
 **return** queue;  
 }  
  
 **public boolean** isFree() {  
 **return** isFree;  
 }  
  
 **public** Task getCurrTask() {  
 **return** currTask;  
 }  
  
 **public int** getK() {  
 **return** k;  
 }  
}

**Результати тестування програми**



