Белорусский государственный университет информатики и радиоэлектроники

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Отчет по лабораторным работам № 5, 6

по дисциплине «Системный анализ и машинное моделирование»

Вариант 3б

Проверил:

Мельник Н. И.

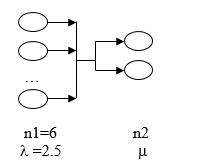
Выполнил:

студент группы 451003

Бобков И.С.

**1. Построение и исследование аналитической модели непрерывно-стохастической системы массового обслуживания**

СМО с ожиданием ответа



Определить среднее число ожидающих ответа источников, среднее время ожидания ответа и абсолютную пропускную способность (интенсивность на выходе системы) Входные потоки и потоки обслуживаний – простейшие.

в) n2=1, μ=18

Решение в Mathcad:





























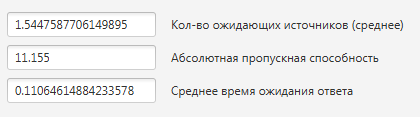




**2. Построение и исследование имитационной модели непрерывно-стохастической системы массового обслуживания**

* для заданной СМО простроить имитационную модель и исследовать ее:

Результаты работы программы:



**Вывод**:

В ходе лабораторной работы была аналитически смоделирована непрерывно-стохастическая СМО и разработана программа, имитирующая ее поведение. Построенная модель позволяет статистически подсчитать необходимые характеристики СМО. Полученные статистические значения искомых характеристик близки к теоретически рассчитанным.

**Листинг**:

**Main:**

**package** sample;  
  
**import** javafx.application.Application;  
**import** javafx.fxml.FXMLLoader;  
**import** javafx.scene.Parent;  
**import** javafx.scene.Scene;  
**import** javafx.stage.Stage;  
  
**public class** Main **extends** Application {  
  
 @Override  
 **public void** start(Stage primaryStage) **throws** Exception{  
 Parent root = FXMLLoader.*load*(getClass().getResource(**"sample.fxml"**));  
 primaryStage.setTitle(**"Mod 5"**);  
 primaryStage.setScene(**new** Scene(root, 500, 320));  
 primaryStage.show();  
 }  
  
  
 **public static void** main(String[] args) {  
 *launch*(args);  
 }  
}

**Controller:**

**package** sample;

**import** javafx.fxml.FXML;  
**import** javafx.scene.control.TextField;

**public class** Controller {  
  
 @FXML  
 TextField **λ\_input**, **μ\_input**, **NumberTextField**, **nk\_out**, **aveTime**, **ATF**;  
  
 @FXML  
 **public void** runButtonClick()  
 {  
 **double** λ = Double.*parseDouble*(**λ\_input**.getText());  
 **double** μ = Double.*parseDouble*(**μ\_input**.getText());  
 **int** time = Integer.*parseInt*(**NumberTextField**.getText());  
 Queue queue = **new** Queue(λ, μ, time);  
  
 queue.run(0);  
  
 **nk\_out**.setText(Double.*toString*(queue.**nk\_counter** / (**double**) time));  
 **ATF**.setText(Double.*toString*(queue.**outCount** /(**double**) (time)));  
 **aveTime**.setText(Double.*toString*(queue.**counter1** / queue.**outCount**));  
 }  
}

**Queue:**

**package** sample;

**import** org.jetbrains.annotations.Contract;

**import** java.util.Random;

**public class** Queue {  
 **public** Random **random** = **new** Random();  
 **private final double λ**, **μ**, **time**;  
 **private int state**;  
  
 **public double e** = 2.71828182846;  
  
 **public long outCount**;  
 **public double counter1**;  
 **public double nk\_counter**;  
  
 **public double current\_time** = 0;  
 **private boolean** [] **sourceFlags** = {**false**, **false**, **false**, **false**, **false**, **false**};  
 **private double** [] **sourceTime** = {0, 0, 0, 0, 0, 0};  
 **private int** [] **channelFlags** = {-1, -1};  
 **private double** [] **channelTime** = {0, 0};  
  
 **public** Queue(**double** λ, **double** μ, **int** time)  
 {  
 **this**.**λ** = λ;  
 **this**.**μ** = μ;  
 **this**.**time** = time;  
  
 **outCount** = 0;  
 **counter1** = 0;  
 **nk\_counter** = 0;  
  
 **state** = 0;  
 }  
  
 **public void** run(**double** current\_time) {  
 **this**.**current\_time** = current\_time;  
 generateNextRequests();  
 **if**(current\_time >= **time**) **return**;  
 updateChannels();  
 }  
  
 **private void** generateNextRequests() {  
 **for** (**int** i = 0; i < 6; i++) {  
 **if**(!**sourceFlags**[i]) nextRequest(i);  
 }  
 }  
  
 **private void** nextRequest(**int** i) {  
 **sourceFlags**[i] = **true**;  
 **double** tmpTime = 0;  
 **for** (**int** j = 0; j < 2; j++) {  
 **if**(i == **channelFlags**[j]) tmpTime = **channelTime**[j];  
 }  
 **sourceTime**[i] = tmpTime + -Math.*log*(**random**.nextDouble())/**λ**;*//время через которое возникнет новая заявка* }  
  
 **private void** updateChannels() {  
 **double** minTime = **time**;  
 **double** tmpTime = 0;  
 **for** (**int** i = 0; i < 2; i++) {  
 tmpTime = handleRequest(i);  
 **if**(tmpTime < minTime) minTime = tmpTime;  
 }  
 run(minTime);  
 }  
  
 **private double** handleRequest(**int** i) {*//время когда освоюодится канал* **int** index = findIndexOfMin();  
 **if**(**current\_time** >= **channelTime**[i]) {  
 **sourceFlags**[index] = **false**;  
 **outCount**++;*//j.hfn* **channelFlags**[i] = index;  
 **double** tmp = -Math.*log*(**random**.nextDouble()) / **μ**;  
 **counter1** += tmp; **if** (**sourceTime**[index] > **current\_time**)  
 **channelTime**[i] = **sourceTime**[index] + tmp;  
 **else channelTime**[i] = **current\_time** + tmp;  
 **nk\_counter** += **channelTime**[i] - **sourceTime**[index];  
 }  
 **return channelTime**[i];  
 }  
  
 @Contract(pure = **true**)  
 **private int** findIndexOfMin() {  
 **int** tmpIndex = 0;  
 **for** (**int** i = 0; i < 6; i++) {  
 **if**((**sourceTime**[tmpIndex] > **sourceTime**[i]) && **sourceFlags**[i]) tmpIndex = i;  
 }  
 **return** tmpIndex;  
 }  
}