Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Системный анализ и машинное моделирование (САиММод)

ОТЧЕТ

по лабораторной работе № 5-6

Тема: Построение аналитической и имитационной модели

Вариант 27

Выполнила

студентка: гр. 751006 Статкевич Н.М.

Проверил: Мельник Н.И.

Минск 2020

1. **Исходные данные**

В СМО вида М/М/1/1 поступают заявки двух видов. появляются на входе с вероятностью р, второго с вероятностью (1-р). Заявка первого вида имеет более высокий приоритет и может вытеснить заявку второго вида из канала в очередь, если место в очереди свободно или из системы, если место занято. В случае, когда заявка первого вида застает систему в состоянии обслуживания заявки первого вида, то она ставится в очередь, если место ожидания свободно или занято заявкой второго вида (менее приоритетная заявка теряется). Найти относительные пропускные способности Q1 и Q2. =0,5, =0,45, р=0,4.

Кодирование состояний:

ab, где:

**a** – заявка в очереди

{0 - отсутствие заявки в очереди, 1 – заявка первого вида, 2 – заявка второго вида};

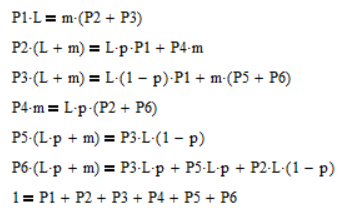
**b** – наличие заявки в канале

{0 – отсутствие заявки, 1 – заявка первого вида, 2 – заявка второго вида};

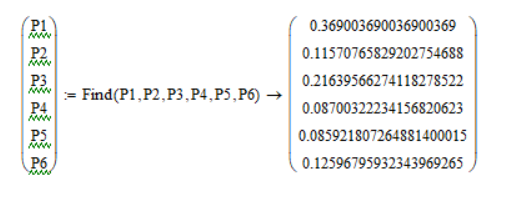
1. Диаграмма интенсивности переходов



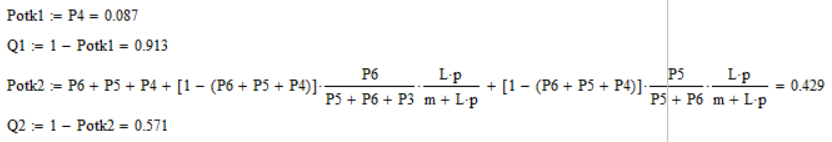
1. Система уравнений



1. Решение системы



1. Вычисление параметров



1. Код программы

**class App**

package samm;

import static samm.InputData.inputDouble;

import static samm.InputData.inputInt;

import samm.model.QueueingSystem;

public class App {

public static void main(String[] args) {

double m = inputDouble("m");

double l = inputDouble("l");

double p = inputDouble("p");

int simulationTime = inputInt("simulation time");

QueueingSystem queueingSystem = new QueueingSystem(m, l, p, simulationTime);

//QueueingSystem queueingSystem = new QueueingSystem(0.5, 0.45, 0.4, 100000);

int[] requests = queueingSystem.run();

double denied1 = queueingSystem.getDeniedRequests1();

double denied2 = queueingSystem.getDeniedRequests2();

System.out.println("P отказа 1 = " + denied1 / requests[0]);

System.out.println("Q 1 = " + (1 - denied1 / requests[0]));

System.out.println("P отказа 2 = " + denied2 / requests[1]);

System.out.println("Q 2 = " + (1 - denied2 / requests[1]));

}

}

**class InputData**

package samm;

import java.util.InputMismatchException;

import java.util.Scanner;

public class InputData {

public static double inputDouble(String strValue) {

while (true) {

Scanner scan = new Scanner(System.in);

System.out.print("Input " + strValue + ": ");

try {

return scan.nextDouble();

} catch (InputMismatchException e) {

System.out.println("Sorry! Value " + strValue + " must be Double (e.g. 1,8 or 2)");

}

}

}

public static int inputInt(String strValue) {

while (true) {

Scanner scan = new Scanner(System.in);

System.out.print("Input " + strValue + ": ");

try {

return scan.nextInt();

} catch (NumberFormatException e) {

System.out.println("Sorry! Value " + strValue + " must be Integer (e.g. 100000)");

}

}

}

}

**class QueueingSystem**

package samm.model;

import static java.lang.Math.log;

import java.util.ArrayList;

import java.util.List;

import java.util.Random;

public class QueueingSystem {

private static final int FIRST\_PRIORITY\_REQUEST = 1;

private static final int SECOND\_PRIORITY\_REQUEST = 2;

private static final int NO\_REQUEST = 0;

private final Random randRequest;

private final Random randTime;

private final double p;

private final double l;

private final double m;

private final int simulationTime;

private final double averageTimeBetweenRequests;

private final double averageProcessingTime;

private int deniedRequests1 = 0;

private int deniedRequests2 = 0;

private int totalRequests1 = 0;

private int totalRequests2 = 0;

private int proceedRequests1 = 0;

private int proceedRequests2 = 0;

private int d = 0;

public QueueingSystem(double m, double l, double p, int simulationTime) {

this.p = p;

this.m = m;

this.l = l;

this.simulationTime = simulationTime;

this.averageTimeBetweenRequests = 1 / l;

this.averageProcessingTime = 1 / m;

this.randRequest = new Random(100000);

this.randTime = new Random(100000);

}

public int getDeniedRequests1() {

return deniedRequests1;

}

public int getDeniedRequests2() {

return deniedRequests2;

}

private int getAppearedRequestPriority() {

return randRequest.nextDouble() < p

? FIRST\_PRIORITY\_REQUEST

: SECOND\_PRIORITY\_REQUEST;

}

private double getTimeBetweenRequests(double averageTimeBetweenRequests) {

return -averageTimeBetweenRequests \* log(randRequest.nextDouble());

}

private double getProcessingTime(double averageProcessingTime) {

return -averageProcessingTime \* log(randTime.nextDouble());

}

public int[] run() {

double currentTime = 0;

List<Request> requests = new ArrayList<Request>();

Channel channel1 = new Channel();

int request1count = 0;

Queue queue = new Queue();

while (currentTime < simulationTime) {

double timePassed = getTimeBetweenRequests(averageTimeBetweenRequests);

currentTime += timePassed;

if (currentTime > simulationTime) {

break;

}

Request request = new Request(

getAppearedRequestPriority(),

currentTime,

getProcessingTime(averageProcessingTime));

requests.add(request);

double timeLeft1 = 0;

if (channel1.isBusy()) {

channel1.getRequest().updateProcessingTime(timePassed);

if (channel1.getRequest().isCompleted()) {

timeLeft1 = channel1.getRequest().getOverTime();

channel1.stop();

}

}

if (queue.hasRequest()) {

if (channel1.isFree()) {

channel1.setRequest(queue.getRequest());

channel1.getRequest().updateProcessingTime(timeLeft1);

if (channel1.getRequest().isCompleted()) {

channel1.stop();

}

queue.clear();

}

}

if (request.getPriority() == FIRST\_PRIORITY\_REQUEST) {

request1count++;

if (channel1.isFree()) {

channel1.setRequest(request);

} else if (channel1.getRequest().getPriority() == SECOND\_PRIORITY\_REQUEST) {

if (queue.isEmpty()) {

queue.setRequest(channel1.getRequest());

} else {

deniedRequests2++;

channel1.getRequest().deny();

}

channel1.setRequest(request);

} else if (queue.isEmpty()) {

queue.setRequest(request);

} else if (queue.getRequest().getPriority() == SECOND\_PRIORITY\_REQUEST) {

deniedRequests2++;

queue.getRequest().deny();

queue.setRequest(request);

} else {

deniedRequests1++;

request.deny();

}

}

if (request.getPriority() == SECOND\_PRIORITY\_REQUEST) {

if (channel1.isFree()) {

channel1.setRequest(request);

} else if (queue.isEmpty()) {

queue.setRequest(request);

} else {

deniedRequests2++;

request.deny();

}

}

}

return new int[]{request1count, requests.size() - request1count};

}

}

**class Request**

package samm.model;

public class Request {

private final int priority;

private final double appearedAt;

private final double processingTime;

private double currentProcessingTime = 0;

private boolean completed = false;

private boolean denied = false;

public Request(int priority, double appearedAt, double processingTime) {

this.priority = priority;

this.appearedAt = appearedAt;

this.processingTime = processingTime;

}

public int getPriority() {

return priority;

}

public boolean isDenied() {

return denied;

}

public void deny() {

this.denied = true;

}

public void updateProcessingTime(double dt) {

currentProcessingTime += dt;

if (currentProcessingTime >= processingTime) {

completed = true;

}

}

public boolean isCompleted() {

return completed;

}

public double getOverTime() {

return currentProcessingTime - processingTime;

}

}

**class Channel**

package samm.model;

public class Channel {

private Request request = null;

public boolean isFree() {

return request == null;

}

public boolean isBusy() {

return request != null;

}

public Request getRequest() {

return request;

}

public void setRequest(Request request) {

this.request = request;

}

public void stop() {

request = null;

}

}

**class Queue**

package samm.model;

public class Queue {

private Request request = null;

public boolean isEmpty() {

return request == null;

}

public boolean hasRequest() {

return request != null;

}

public Request getRequest() {

return request;

}

public void setRequest(Request request) {

this.request = request;

}

public void clear() {

request = null;

}

}

**Вывод:**

В данной работе была исследована непрерывно-стохастическая модель. Для нее была построена диаграмма интенсивностей переходов, на основе которой были посчитаны вероятности состояний СМО. Были посчитаны необходимые характеристики системы и сравнены с результатами, получившимися при имитационном моделировании.