Белорусский государственный университет информатики и радиоэлектроники

Кафедра «Программного обеспечения информационных технологий»

Группа 251001

Построение аналитической и имитационной непрерывно-стахостической модели и сравнить результаты исследования

Отчет о лабораторной работе №4

по дисциплине «САиММод»

Студент: П.А.Верховцов

Преподаватель: Н.И. Мельник

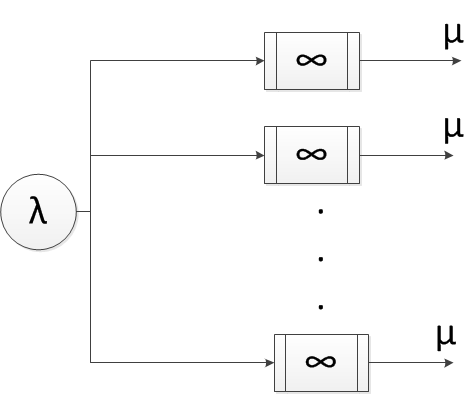
Минск 2015

**Условие (Вариант 32)**

Система представляет собой совокупность из *n* одноканальных СМО с неограниченной очередью. Входной поток простейший. Заявки из входного потока последовательно распределяются между СМО: 1-я заявка попадает в СМО №1, 2-я заявка попадает в СМО №2,…, *n*-я заявка попадает в СМО № *n*, (*n+1)*-я заявка попадает в СМО № *1* и т.д. Найти значения показателей эффективности Lоч , Lc , Wоч , Wс.

*n*=3, λ = 12, μ =4,5.

Отдельную СМО можно представить как систему Er/M/1/∞



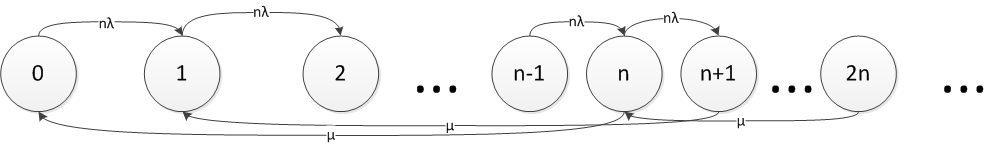
**(Схема СМО)**

**Кодирование**

Будем определять состояния в данный момент времени общим числом этапов поступления в этой системе.

Если в системе уже находится k заявок, а очередная находится на i-м этапе поступления, то *j = rk+i-1*.

**ДИП системы**



**Решение**

При составление системы воспользуемся классическим положением:

∑вход. потоков = ∑ выход. потоков

Тогда вероятность нахождения в системе k – заявок равна:

**Расчет параметров**

*По формуле Полячена-Хинчина:*

*По формуле Литтла:*

ω = λ/(n\*µ)

Из условия *n*=3, λ = 12, μ =4,5

ω= 12/(3\*4.5) = 0,88

**Loч** = (0,88^2)\*(1/3+1)/(2\*(1-0,88)) = 1.0325/0,24=**4,302**

**Lc =** 4,302+0.88 **= 5,19**

**Wоч** = 4,302 \*3 / 12 = **1,0755**

**Wc** = 5,19 \*3 / 12 = **1,2975**

Листинг программы

public class Main {

private static Processor[] querySystems;

private static TimePeriodGenerator taskInterval;

private static int countOfQuerySystems = 3;

private static long GeneratedTaskCounter;

private static double intensityOfService = 4.5;

private static double intensityOfTasks = 12;

private static int workHours = 10000;

public static void main(String[] args) {

long workTime = workHours \* 3600;

taskInterval = new TimePeriodGenerator(intensityOfTasks);

createQuerySystems();

int nextTaskInterval = taskInterval.getInterval();

startWork(nextTaskInterval, workTime);

statictic(workTime);

}

private static void statictic(long workTime) {

System.out.println("Work time> " +workTime/3600+" hours");

System.out.println("Generated task> " + GeneratedTaskCounter);

System.out.println("Processed task> " + Processor.getProcessedTask());

System.out.println("Absolutely bandwidth> "

+ (double) Processor.getProcessedTask() / workHours

+ " task/hours");

System.out.println("Average queue length> "

+ (float) Processor.getCountTasksInQueue() / (workTime \* countOfQuerySystems));

System.out.println("Average tasks in system> "

+ (float) Processor.getCountTasksInSystem() / (workTime \* countOfQuerySystems));

System.out.println("Average time in queue> "

+ Processor.getAverageTimeInQueue() + " hours");

System.out.println("Average time in system> "

+ Processor.getAverageTimeInSystem() + " hours");

}

private static void createQuerySystems() {

querySystems = new Processor[countOfQuerySystems];

for (int i = 0; i < countOfQuerySystems; i++) {

querySystems[i] = new Processor(intensityOfService);

}

}

private static void startWork(int nextTaskInterval, long workTime) {

for (int i = 0; i < workTime; i++) {

if (nextTaskInterval == 0) {

querySystems[(int) (GeneratedTaskCounter % countOfQuerySystems)]

.setTask(GeneratedTaskCounter);

nextTaskInterval = taskInterval.getInterval();

GeneratedTaskCounter++;

}

nextTaskInterval--;

decrimentQuerySystem();

}

}

private static void decrimentQuerySystem() {

for (Processor proc : querySystems) {

proc.decrimentTimer();

}

}

}

public class Processor {

private Queue<Object> taskQueue;

private TimePeriodGenerator procesedTimeGenerator;

private boolean isFree;

private int timeToProcess;

private static int timeTaskInQueue;

private static int timeTaskInSystem;

private static int processedTasks;

private static int taskInQueue;

private static int taskInSystem;

public Processor(double intensity) {

this.taskQueue = new LinkedList<Object>();

this.procesedTimeGenerator = new TimePeriodGenerator(intensity);

this.isFree = true;

}

public void setTask(Object task) {

if (isFree) {

timeToProcess = procesedTimeGenerator.getInterval();

timeTaskInSystem += timeToProcess;

isFree = false;

} else {

taskQueue.add(task);

}

}

public static int getProcessedTask() {

return processedTasks;

}

public static int getCountTasksInQueue() {

return taskInQueue;

}

public static int getCountTasksInSystem() {

return taskInSystem;

}

public static float getAverageTimeInQueue() {

return (float) timeTaskInQueue / processedTasks / 3600;

}

public static float getAverageTimeInSystem() {

return (float) timeTaskInSystem / processedTasks / 3600;

}

public void decrimentTimer() {

taskInQueue += taskQueue.size();

taskInSystem += taskQueue.size();

if (!isFree) {

taskInSystem += 1;

}

timeTaskInQueue += taskQueue.size();

timeTaskInSystem += taskQueue.size();

if (!isFree && timeToProcess == 0) {

isFree = true;

processedTasks++;

if (taskQueue.size() > 0) {

setTask(taskQueue.poll());

timeToProcess = procesedTimeGenerator.getInterval();

}

} else if (timeToProcess == 0) {

} else {

timeToProcess--;

}

}

}

public class TimePeriodGenerator {

private double intens;

private Random generator;

public TimePeriodGenerator(double intens) {

this.intens = intens;

this.generator = new Random();

}

public int getInterval() {

return (int) (-1/(double)this.intens\*Math.log(1-generator.nextDouble())\*3600)+1;

}

}