Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації та управління

Організація обчислювальних процесів

Комп’ютерний практикум №2

«Розробка простої моделі масового обслуговування»

Виконав:

студент групи ІС-73

Коноплянка Д. С.

Перевірила:

Новікова П.А.

Київ 2020 р.

**Завдання до практичної роботи**

1. Реалізувати алгоритм імітації простої моделі обслуговування одним пристроєм (рис 2.1). **10 балів.**

PROCESS 1

DESPOSE

CREATE

Рисунок 2.1 – Схема моделі.

1. Модифікувати алгоритм, додавши обчислення середнього завантаження пристрою. **30 балів.**
2. Модифікувати алгоритм, додавши обчислення середнього очікування в черзі. **30 балів.**
3. Виконати верифікацію моделі, змінюючи значення вхідних змінних та параметрів моделі. Навести результати верифікації у таблиці. **30 балів.**

**Лістинг**

public class SimSimple {

    public static void main(String[] args) {

        System.out.println("Час моделювання | Обмеження довжини черги | Середня довжина черги | " +

        "Середній час очікування | Ймовірність відмови | Кількість вимог | Кількість не обслугованих | Час створення | Час обробки");

        Model model = new Model(2,1,5);

        model.simulate(1000);

        model = new Model(4,4,6);

        model.simulate(1000);

        model = new Model(4,8,5);

        model.simulate(1000);

        model = new Model(10,5,10);

        model.simulate(1000);

    }

}

public class Model {

    private double tnext;  // след знач

    private double tcurr; // тек знач

    private double tprev; // пред знач

    private double t0, t1; // т0 момент входа, т1 момент выхода

    private double delayCreate, delayProcess; // час створення, час обробки

    private int numCreate, numProcess, failure; // колво требований,

    private int state, maxqueue, queue; // состояние, макс очередь, длина очереди

    private int nextEvent; // для свича, если есть очередь

    private double avarage = 0.0; // среднее

    public Model(double delay0, double delay1) { // конструктор инициализации полей

        delayCreate = delay0;

        delayProcess = delay1;

        tnext = 0.0;

        tcurr = tnext;

        tprev = tnext;

        t0 = tcurr;

        t1 = Double.MAX\_VALUE;

        maxqueue = 0;

    }

    public Model(double delay0, double delay1, int maxQ) { // конструктор инициализации полей

        delayCreate = delay0;

        delayProcess = delay1;

        tnext = 0.0;

        tcurr = tnext;

        tprev = tnext;

        t0 = tcurr;

        t1 = Double.MAX\_VALUE;

        maxqueue = maxQ;

    }

    public void simulate(double timeModeling) { // функция симуляции

        while (tcurr < timeModeling) { // пока текущее значение меньше заданого

            tnext = t0; // приравниваем сгенерированное число к след

            nextEvent = 0;

            if (t1 < tnext) { // если нету элементов в очереди, то пропуск условия, если есть заходим и обрабатываем ближайший по времени элемент

                tnext = t1; // обслуживание ближайшего в очереди

                nextEvent = 1;

            }

            tprev = tcurr; // запоминаем текущее значение

            tcurr = tnext; // приравниваем след знач к настоящему

            switch (nextEvent) { //  в зависимости от очереди выполняется нужный свич

                case 0:

                    event0();

                    break;

                case 1:

                    event1();

            }

            avarage += (tcurr - tprev) \* queue; // для средней длины очереди

            // printInfo();

        }

        printStatistic();

        // System.out.println(avarage / timeModeling);

        // System.out.println(avarage / numProcess);

        printAll(timeModeling);

    }

    public void printStatistic() {

        System.out.println(" numCreate= " + numCreate + " numProcess = " + numProcess + " failure = " + failure);

    }

    public void printInfo() {

        System.out.println(" t= " + tcurr + " state = " + state + " queue = " + queue);

    }

    public void printAll(double time) {

        System.out.println(time + "\t \t \t" + maxqueue + "\t \t \t" + String.format("%.3f",(avarage / time)) + "\t \t \t" +

        String.format("%.3f",(avarage / numProcess)) + "\t \t \t \t" + String.format("%.3f", (Double.valueOf(failure) / Double.valueOf(numCreate))) + "\t \t \t" +

        numCreate + "\t \t" + failure + "\t \t \t" +  delayCreate + "\t \t" + delayProcess);

    }

    public void event0() {

        t0 = tcurr + getDelayOfCreate(); // время обслуживание плюс текущее

        numCreate++; // новое обслуживание

        if (state == 0) { // если до этого ничего не обслуживалось принимаем элемент

            state = 1; // ставим состояние "занятости"

            t1 = tcurr + getDelayOfProcess(); // время обслуживание плюс текущее

        } else {

            if (queue < maxqueue) // если максимальная очередь больше текущей тогда обслуживание провалено, если нет, добавляем в очередь

                queue++;

            else

                failure++;

        }

    }

    public void event1() {

        t1 = Double.MAX\_VALUE; // максимально большое число

        state = 0;

        if (queue > 0) { // если очередь не пуста берем на обслуживание  элемент с очереди

            queue--;

            state = 1; // ставим состояние "занятости"

            t1 = tcurr + getDelayOfProcess(); // текущее время обслуживание плюс сколько понадобиться на обслуживание, запоманием это число

        }

        numProcess++; // + удавшееся обслуживание

    }

    private double getDelayOfCreate() {

        return FunRand.Exp(delayCreate); // генерация случайного числа экспоненциально

    }

    private double getDelayOfProcess() {

        return FunRand.Exp(delayProcess); // генерация случайного числа экспоненциально

    }

}

import java.util.Random;

public class FunRand {

    /\*\*

     \* Generates a random value according to an exponential distribution

     \*

     \* @param timeMean mean value

     \* @return a random value according to an exponential distribution

     \*/

    public static double Exp(double timeMean) {

        double a = 0;

        while (a == 0) {

            a = Math.random();

        }

        a = -timeMean \* Math.log(a);

        return a;

    }

    /\*\*

     \* Generates a random value according to a uniform distribution

     \*

     \* @param timeMin

     \* @param timeMax

     \* @return a random value according to a uniform distribution

     \*/

    public static double Unif(double timeMin, double timeMax) {

        double a = 0;

        while (a == 0) {

            a = Math.random();

        }

        a = timeMin + a \* (timeMax - timeMin);

        return a;

    }

    /\*\*

     \* Generates a random value according to a normal (Gauss) distribution

     \*

     \* @param timeMean

     \* @param timeDeviation

     \* @return a random value according to a normal (Gauss) distribution

     \*/

    public static double Norm(double timeMean, double timeDeviation) {

        double a;

        Random r = new Random();

        a = timeMean + timeDeviation \* r.nextGaussian();

        return a;

    }

}

Результат виконання коду

