НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

КАФЕДРА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

Лабораторна робота №3

з дисципліни **«**Комп’ютерне моделювання**»**

Виконав:

студент 3 курсу

ФІОТ гр. ІО-21

Кузьменко Володимир

Перевірив:

Марковський О.П.

Київ – 2014 р.

Варіант завдання: задач 6, ядер процесора 2, конфігурація системи – ВП

Лістинг коду програми

package cm\_lab3;

public class Test {

public static void main(String[] args) {

Simulation s = new Simulation();

s.buildSystem();

s.simulate();

}

}

package cm\_lab3;

import java.util.Random;

public class Simulation {

public static EventList<Event> eventList = new EventList<Event>();

public static Random r = new Random();

public static double time = 0;

public static double timeSolution(double lambda) {

return (-1 / lambda) \* Math.log(r.nextDouble());

}

/\*\*

\* метод для побудови та налаштування системи

\*/

public void buildSystem() {

Device processor = new Device(

1,//середній час розв\*язання

2,//кількість ядер

6,//початкова кількість задач

"Процесор");

Device north\_bridge = new Device(35, 1, 0, "Північний міст");

Device south\_bridge = new Device(50, 1, 0, "Південний міст");

Device ram = new Device(10, 1, 0, "Оперативна пам\*ять");

Device video\_processor = new Device(33, 1, 0, "Відео процесор");

//налаштуванння зв\*язків між пристроями та встановлення імовірностей

processor.linkDevices(north\_bridge, processor);

processor.linkOutProbability(0.15, 0.85);

processor.calcNextTime();

north\_bridge.linkDevices(processor, south\_bridge, ram);

north\_bridge.linkOutProbability(0.1, 0.4, 0.5);

ram.linkDevices(north\_bridge);

ram.linkOutProbability(1);

south\_bridge.linkDevices(video\_processor);

south\_bridge.linkOutProbability(1);

video\_processor.linkDevices(processor);

video\_processor.linkOutProbability(1);

}

//симуляція

public void simulate() {

while (time < 10000) {

Device dev = minTimeDevice();

dev.transaction();

}

eventList.statistic();

}

// пошук пристрою з мінімальним часом спрацювання

private Device minTimeDevice() {

double minTime = Double.MAX\_VALUE;

Device device = eventList.get(0).device;

int i = 0;

int index = 0;

for (Event evnt : eventList) {

if (minTime > evnt.timeEvent) {

minTime = evnt.timeEvent;

device = evnt.device;

index = i;

}

i++;

}

eventList.remove(index);

return device;

}

}

package cm\_lab3;

import java.util.ArrayList;

import static cm\_lab3.Simulation.\*;

@SuppressWarnings("serial")

public class EventList <T> extends ArrayList<Event> {

private double timeRam=0;

private double timeNorthBridge=0;

private double timeSorthBridge=0;

private double timeVideoProcessor=0;

private double timeOneCore=0;

private double timeTwoCore=0;

public EventList() {

super();

}

/\*\*

\* метод додавання подій

\* @param e подія

\* @param time\_solution час розв\*язання

\* @param n\_cores кількість ядер

\* @return

\*/

public boolean add(Event e, double time\_solution, int n\_cores) {

switch (e.device.name) {

case "Процесор":

if (n\_cores==1)

timeTwoCore+=time\_solution;

else timeOneCore+=time\_solution;

break;

case "Північний міст":

timeNorthBridge+=time\_solution;

break;

case "Південний міст":

timeSorthBridge+=time\_solution;

break;

case "Оперативна пам\*ять":

timeRam+=time\_solution;

break;

case "Відео процесор":

timeVideoProcessor+=time\_solution;

break;

default:

System.err.println("Error!");

break;

}

return super.add(e);

}

/\*\*

\* статистика

\*/

public void statistic(){

timeNorthBridge=(timeNorthBridge/time)\*100;

timeOneCore=(timeOneCore/time)\*100;

timeRam = (timeRam/time)\*100;

timeSorthBridge = (timeSorthBridge/time)/100;

timeTwoCore=(timeTwoCore/time)\*100;

timeVideoProcessor = (timeVideoProcessor/time)\*100;

System.out.print("Час моделювання: ");

System.out.printf("%6.5f\n",time);

System.out.print("\nЗагрузка Одного ядра процесора: ");

System.out.printf("%6.5f",timeOneCore);

System.out.print("%\nЗагрузка Двох ядер процесора: ");

System.out.printf("%6.5f",timeTwoCore);

System.out.print("%\nЗагрузка Оперативної пам'яті: ");

System.out.printf("%6.5f",timeRam);

System.out.print("%\nЗагрузка Північного моста: ");

System.out.printf("%6.5f",timeNorthBridge);

System.out.print("%\nЗагрузка Південного моста: ");

System.out.printf("%6.5f",timeSorthBridge);

System.out.print("%\nЗагрузка Відео процесора ");

System.out.printf("%6.5f",timeVideoProcessor);

System.out.print("%");

}

}

package cm\_lab3;

public class Event {

//з якого пристрою подія

Device device;

//коли спрацює

double timeEvent;

public Event(Device device, double time) {

this.device = device;

this.timeEvent = time;

}

}

package cm\_lab3;

import java.util.ArrayList;

import static cm\_lab3.Simulation.\*;

public class Device {

//середній час задачі в пристрої

private double mean\_time\_solution;

//модельний час, коли закінчиться розв\*язуватись задача

private double time\_solution;

//час роботи пристрою

public double time\_work=0;

//час розв\*язання певної задачі

public double time\_task\_solution=0;

// черга

public ArrayList<Integer> queue = new ArrayList<>();

// ресурс

private ArrayList<Integer> resource = new ArrayList<>();

// вільні ядра

private ArrayList<Integer> free\_cores = new ArrayList<>();

// пристрої на які буде виходити задача з даного пристрою

private ArrayList<Device> out\_Devices = new ArrayList<Device>();

// ймовірність переходу задачі вихідні пристрої

private ArrayList<Double> out\_probability = new ArrayList<>();

//ім\*я пристрою

public String name;

public Device(double mean\_time\_solution, int number\_cores, int start\_task\_queue, String name) {

this.mean\_time\_solution = mean\_time\_solution;

for (int i = 0; i < number\_cores; i++) {

free\_cores.add(1);

}

for (int i = 0; i < start\_task\_queue; i++) {

queue.add(1);

}

this.name = name;

}

/\*\*

\* метод встановлює пристрої на які буде виходити задача з даного пристрою

\* @param devices пристрої

\*/

public void linkDevices(Device... devices) {

for (Device device : devices) {

out\_Devices.add(device);

}

}

/\*\*

\* метод встановлює імовірності попадання задачі на вихідні пристрої

\* список має бути зростаючим

\* @param probability

\*/

public void linkOutProbability(double... probability) {

for (double p : probability) {

out\_probability.add(p);

}

}

/\*\*

\* дoдає задачу в чергу ресурсу

\*/

private void taskToDevice(){

this.queue.add(1);

}

/\*\*

\* відповідає за "часовий" перехід

\*/

public void transaction() {

time\_work+=time\_task\_solution;

time=time\_solution;

resource.remove(0);

free\_cores.add(1);

Device dev = out\_Devices.get(findNextDevice());

dev.taskToDevice();

dev.calcNextTime();

}

/\*\*

\* шукає куда піде задача

\* @return ідекс у масиві out\_Devices (вихідні пристрої) куди піде задача

\*/

private int findNextDevice(){

double p = r.nextDouble();

int i;

Double [] pi = new Double [out\_probability.size()];

pi = out\_probability.toArray(pi);

double s=0;

for (i = 0; i < pi.length; i++) {

s+=pi[i];

if (s>p)break;

}

return i;

}

/\*\*

\* створює нову подію, решення задачі

\*/

public void calcNextTime(){

while (free\_cores.size()!=0 && queue.size()!=0){

free\_cores.remove(0);

resource.add(1);

queue.remove(0);

if (resource.size()!=0){

time\_task\_solution =timeSolution(mean\_time\_solution);

time\_solution = time\_task\_solution+time;

}

Event newEvent = new Event(this, time\_solution);

eventList.add(newEvent, time\_task\_solution, free\_cores.size());

}

}

}

package cm\_lab3;

public class Event {

//на якому пристрої відбудеться подія

Device device;

//коли спрацює

double timeStart;

double timeFinish;

public Event(Device device, double timeStart, double timeFinish) {

this.device = device;

this.timeStart = timeStart;

this.timeFinish = timeFinish;

}

@Override

public boolean equals(Object obj) {

Event e = (Event)obj;

return e.device.equals(device) && e.timeStart== timeStart && e.timeFinish == timeFinish;

}

}

package cm\_lab3;

import java.util.ArrayList;

import static cm\_lab3.Simulation.\*;

public class Device {

//середній час задачі в пристрої

public double mean\_time\_solution;

// черга

public ArrayList<String> queue = new ArrayList<String>();

// ресурс

public ArrayList<String> resource = new ArrayList<String>();

// вільні ядра

public ArrayList<String> free\_cores = new ArrayList<String>();

// пристрої на які буде виходити задача з даного пристрою

public ArrayList<Device> out\_Devices = new ArrayList<Device>();

// ймовірність переходу задачі вихідні пристрої

public ArrayList<Double> out\_probability = new ArrayList<>();

public Resource resourceProc = new Resource();

//ім\*я пристрою

public String name;

public int id;

public Device(double mean\_time\_solution, int number\_cores, int start\_task\_queue, String name, int id) {

this.mean\_time\_solution = mean\_time\_solution;

for (int i = 0; i < number\_cores; i++) {

free\_cores.add("1");

}

for (int i = 0; i < start\_task\_queue; i++) {

queue.add("1");

}

this.name = name;

this.id = id;

}

/\*\*

\* метод встановлює пристрої на які буде виходити задача з даного пристрою

\* @param devices пристрої

\*/

public void linkDevices(Device... devices) {

for (Device device : devices) {

out\_Devices.add(device);

}

}

/\*\*

\* метод встановлює імовірності попадання задачі на вихідні пристрої

\* список має бути зростаючим

\* @param probability

\*/

public void linkOutProbability(double... probability) {

for (double p : probability) {

out\_probability.add(p);

}

}

/\*\*

\* дoдає задачу в чергу ресурсу

\*/

public void taskToDevice(){

this.queue.add("1");

}

/\*\*

\* відповідає за "часовий" перехід

\*/

public void transaction(Event event) {

eventList.remove(event);

free\_cores.add("1");

Device dev = out\_Devices.get(findNextDevice());

time=event.timeFinish;

if (id ==1){

resourceProc.remove(event);

}else{

resource.remove(0);

}

if (this.queue.size()!=0){

this.calcNextTime(time);

}

dev.taskToDevice();

dev.calcNextTime(time);

}

/\*\*

\* шукає куда піде задача

\* @return ідекс у масиві out\_Devices (вихідні пристрої) куди піде задача

\*/

public int findNextDevice(){

double p = r.nextDouble();

int i;

Double [] pi = new Double [out\_probability.size()];

pi = out\_probability.toArray(pi);

double s=0;

for (i = 0; i < pi.length; i++) {

s+=pi[i];

if (s>p)break;

}

return i;

}

/\*\*

\* створює нову подію, решення задачі

\*/

public void calcNextTime(double time){

while (free\_cores.size()!=0 && queue.size()!=0){

free\_cores.remove(0);

queue.remove(0);

Event newEvent = new Event(this, time, time + mean\_time\_solution);

if (id ==1){

resourceProc.add(newEvent);

}else{

resource.add("1");

}

eventList.add(newEvent);

}

}

public void pr (){

System.out.print("\n"+name+"| " +"Черга "+ queue.size()+"| Ресурс "+resource.size()+"| Вільні ядра "+free\_cores.size());

}

}

package cm\_lab3;

import java.util.ArrayList;

import java.util.Iterator;

public class EventList implements Iterable<Event> {

private ArrayList<Event> list = new ArrayList<Event>();

void add(Event e) {

list.add(e);

}

void remove(Event e) {

list.remove(e);

}

Event get (int index ){

return list.get(index);

}

@Override

public Iterator<Event> iterator() {

return list.iterator();

}

}

package cm\_lab3;

import static cm\_lab3.Simulation.\*;

public class Resource {

private Event oneCore = null;

private Event twoCore = null;

public double allTime = 0;

public double paralelTime = 0;

private double startWork = 0;

public void add(Event e) {

if (oneCore == null && twoCore == null) {

startWork = time;

}

if (oneCore == null && twoCore == null) {

// було 0 задач, прийшла одна задача

} else {

// була 1 задача прийшла ще одна

if (e.timeStart < oneCore.timeFinish) {

paralelTime += oneCore.timeFinish - e.timeStart;

}

}

if (oneCore == null) {

oneCore = e;

} else {

twoCore = e;

}

}

public void remove(Event e) {

if (oneCore != null && twoCore != null) {

// було 2 задачі лищається одна задача

} else {

// була одно задача, лишаєється 0 задач

}

if (oneCore.equals(e)) {

oneCore = twoCore;

twoCore = null;

} else {

twoCore = null;

}

if (oneCore == null && twoCore == null) {

allTime += time - startWork;

}

}

public int size() {

if (oneCore == null && twoCore == null) {

return 0;

} else {

if ((oneCore == null && twoCore != null)

|| (oneCore != null && twoCore == null)) {

return 1;

} else {

return 2;

}

}

}

}

package cm\_lab3;

public class Test {

public static void main(String[] args) {

Simulation s = new Simulation();

s.buildSystem();

s.simulate();

}

}