Міністерство освіти і науки України

Національний технічний університет України

«Київський політехнічний інститут»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Лабораторна робота №2

з дисципліни «Комп’ютерне моделювання»

Виконав:

Ст. гр. ІО-42

Слюсаренко Олександр

Київ – 2016

Код програми:

**package** laba2;

**import** java.util.\*;

**public** **class** Unit {

**private** String name;

**private** **double** averageSolutionTime;

**private** ArrayList<Core> cores;

**private** ArrayList<Task> tasks;

**private** LinkedHashMap<Unit, Double> linkedUnits;

**private** **int** n;

**private** **int** queueLength;

**public** Unit(String name, **double** averageSolutionTime) {

**this**(name, averageSolutionTime, 1);

}

**public** Unit(String name, **double** averageSolutionTime, **int** coreCount) {

**this**.name = name;

**this**.averageSolutionTime = averageSolutionTime;

cores = **new** ArrayList<>();

tasks = **new** ArrayList<>();

linkedUnits = **new** LinkedHashMap<>();

**for** (**int** i = 0; i < coreCount; i++) {

cores.add(**new** Core("CPU" + i));

}

}

**public** **void** addLinkedUnit(Unit unit, **double** probability) {

linkedUnits.put(unit, probability);

}

**public** **void** addTask(Task task, **double** currentTime) {

**if** (isFree()) {

Core core = getFreeCore();

core.setTask(task, currentTime);

} **else** {

tasks.add(task);

}

}

**public** **void** nextStep(**double** currentTime) {

Core core = getNextCore();

Task solvedTask = core.getTask();

getNextUnit().addTask(solvedTask, currentTime);

**if** (!tasks.isEmpty()) {

core.setTask(tasks.remove(0), currentTime);

}

n++;

queueLength += tasks.size();

}

**public** **boolean** isFree() {

**boolean** result = **false**;

**for** (Core core: cores) {

**if** (core.isFree()) {

result = **true**;

**break**;

}

}

**return** result;

}

**public** **double** getSolveTime() {

**double** solveTime = Double.***POSITIVE\_INFINITY***;

**for** (Core core: cores) {

**if** (!core.isFree()) {

solveTime = Math.*min*(solveTime, core.getSolveTime());

}

}

**return** solveTime;

}

**public** **void** printStatistic(**double** simulationTime) {

System.***out***.println(name);

System.***out***.println("Average queue length: " + ((**double**) queueLength / n));

**if** (cores.size() == 1) {

System.***out***.println("Functioning capacity: " + (cores.get(0).getWorkingTime() / simulationTime));

} **else** {

**for** (Core core : cores) {

System.***out***.println(core.getName());

System.***out***.println("Functioning capacity: " + (core.getWorkingTime() / simulationTime));

}

}

System.***out***.println();

}

**private** Core getFreeCore() {

Core freeCore = **null**;

**for** (Core core: cores) {

**if** (core.isFree()) {

freeCore = core;

**break**;

}

}

**return** freeCore;

}

**private** Unit getNextUnit() {

Unit nextUnit = **null**;

**double** currentProbability = 0.0;

**double** generatedProbability = Math.*random*();

**for** (Map.Entry<Unit, Double> entry: linkedUnits.entrySet()) {

currentProbability += entry.getValue();

**if** (generatedProbability < currentProbability) {

nextUnit = entry.getKey();

**break**;

}

}

**return** nextUnit;

}

**private** Core getNextCore() {

Core nextCore = **null**;

**double** solveTime = Double.***POSITIVE\_INFINITY***;

**for** (Core core: cores) {

**if** (!core.isFree() && core.getSolveTime() < solveTime) {

nextCore = core;

solveTime = core.getSolveTime();

}

}

**return** nextCore;

}

**private** **class** Core {

**private** String name;

**private** Task task;

**private** **double** solveTime;

**private** **double** workingTime;

**public** Core(String name) {

**this**.name = name;

}

**public** Task getTask() {

Task solvedTask = task;

task = **null**;

**return** solvedTask;

}

**public** **void** setTask(Task task, **double** currentTime) {

**this**.task = task;

**double** solutionTime = TimeGenerator.*nextSolutionTime*(averageSolutionTime);

solveTime = currentTime + solutionTime;

workingTime += solutionTime;

}

**public** **double** getSolveTime() {

**return** solveTime;

}

**public** **boolean** isFree() {

**return** (task == **null**);

}

**public** String getName() {

**return** name;

}

**public** **double** getWorkingTime() {

**return** workingTime;

}

}

}

**package** laba2;

**import** java.util.\*;

**public** **class** UnitController {

**private** **int** taskCount;

**private** **double** simulationTime;

**private** ArrayList<Unit> units;

**public** UnitController(**int** taskCount, **double** simulationTime) {

**this**.taskCount = taskCount;

**this**.simulationTime = simulationTime;

units = **new** ArrayList<>();

initLinks();

}

**public** **void** initLinks() {

Unit cpu = **new** Unit("CPU", 1.0, 2);

Unit nb = **new** Unit("NB", 2.0);

Unit ram = **new** Unit("RAM", 20.0);

Unit vp = **new** Unit("VP", 100.0);

Unit sb = **new** Unit("SB", 5.0);

Unit wa = **new** Unit("WA", 100.0);

Unit mdc = **new** Unit("MDC", 200.0);

Unit odc = **new** Unit("ODC", 400.0);

Unit isa = **new** Unit("ISA", 20.0);

Unit lpt = **new** Unit("LPT", 500.0);

cpu.addLinkedUnit(nb, 1.0);

nb.addLinkedUnit(cpu, 0.45);

nb.addLinkedUnit(vp, 0.05);

nb.addLinkedUnit(sb, 0.1);

nb.addLinkedUnit(ram, 0.4);

ram.addLinkedUnit(nb, 1.0);

vp.addLinkedUnit(cpu, 1.0);

sb.addLinkedUnit(nb, 0.5);

sb.addLinkedUnit(isa, 0.15);

sb.addLinkedUnit(wa, 0.1);

sb.addLinkedUnit(mdc, 0.2);

sb.addLinkedUnit(odc, 0.05);

wa.addLinkedUnit(sb, 1.0);

mdc.addLinkedUnit(sb, 1.0);

odc.addLinkedUnit(sb, 1.0);

isa.addLinkedUnit(sb, 0.9);

isa.addLinkedUnit(lpt, 0.1);

lpt.addLinkedUnit(cpu, 1.0);

units.add(cpu);

units.add(nb);

units.add(ram);

units.add(vp);

units.add(sb);

units.add(wa);

units.add(mdc);

units.add(odc);

units.add(isa);

units.add(lpt);

**for** (**int** i = 0; i < taskCount; i++) {

cpu.addTask(**new** Task(), 0.0);

}

}

**public** **void** start() {

Unit currentUnit = getNextUnit();

**double** currentTime = currentUnit.getSolveTime();

**while** (currentTime < simulationTime) {

currentUnit.nextStep(currentTime);

currentUnit = getNextUnit();

currentTime = currentUnit.getSolveTime();

}

printResult();

}

**private** Unit getNextUnit() {

Unit nextUnit = **null**;

**double** solveTime = Double.***POSITIVE\_INFINITY***;

**for** (Unit unit: units) {

**if** (!unit.isFree() && unit.getSolveTime() < solveTime) {

nextUnit = unit;

solveTime = unit.getSolveTime();

}

}

**return** nextUnit;

}

**public** **void** printResult() {

**for** (Unit unit: units) {

unit.printStatistic(simulationTime);

}

}

**public** **static** **void** main(String[] args) {

**new** UnitController(100, 100000.0).start();

}

}

**package** laba2;

**import** java.util.\*;

**public** **class** TimeGenerator {

**public** **final** **static** **double** nextSolutionTime(**double** averageSolutionTime) {

**return** -averageSolutionTime \* Math.*log*(Math.*random*());

}

}

**package** laba2;

**public** **class** Task {

}