Міністерство освіти і науки України

Національний технічний університет України

«Київський політехнічний інститут»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Лабораторна робота №1

Виконав:

Ст. гр. ІО-42

Слюсаренко О. Є.

Київ – 2016

Тип СМО: Rand, FB, SF

Код програми:

**package** lab1;

**import** java.util.ArrayList;

**import** java.util.Collections;

**import** java.util.LinkedList;

**import** java.util.List;

**import** java.util.Random;

**import** java.util.concurrent.atomic.AtomicInteger;

**public** **class** Lab1 {

**public** **static** **final** **int** ***listSize*** = 10;

**public** **static** **final** **long** ***TAU*** = 10; //milliseconds

**public** **static** **final** Double ***LAMBDA*** = 0.5;

**public** **static** **final** Double ***MU*** = 0.6;

**public** **static** **final** **int** ***MILLISECONDS\_CONST*** = 10;

**public** **static** **final** **long** ***T1*** = ***MILLISECONDS\_CONST*** \* 4;

**public** **static** **final** **long** ***T2*** = ***MILLISECONDS\_CONST*** \* 6;

**public** **static** **final** **int** ***MAX\_COUNT*** = 1000;

**static** List<List<Task>> *tasksStack* = **new** LinkedList<List<Task>>();

**static** List<Task> *tasksListShort* = **new** LinkedList<Task>();

**static** List<Task> *tasksListRand* = **new** LinkedList<Task>();

**static** {

*tasksStack*.add(**new** LinkedList<Task>());

}

**public** **static** **boolean** *shouldFinish* = **false**;

**public** **static** **boolean** *shouldFinishRand* = **false**;

**public** **static** **boolean** *shouldFinishShort* = **false**;

**public** **static** AtomicInteger *wasted* = **new** AtomicInteger(0);

**public** **static** AtomicInteger *wastedRand* = **new** AtomicInteger(0);

**public** **static** AtomicInteger *wastedShort* = **new** AtomicInteger(0);

**public** **static** AtomicInteger *allCreatedTasks* = **new** AtomicInteger(0);

**public** **static** AtomicInteger *allCreatedTasksRand* = **new** AtomicInteger(0);

**public** **static** AtomicInteger *allCreatedTasksShort* = **new** AtomicInteger(0);

**public** **static** **final** List<Task> ***compleatedTasks*** = **new** ArrayList<>();

**public** **static** **final** List<Task> ***compleatedTasksShort*** = Collections.*synchronizedList*(**new** ArrayList<>());

**public** **static** **final** List<Task> ***compleatedTasksRand*** = Collections.*synchronizedList*(**new** ArrayList<>());

**public** **static** Random *randomGenerator* = **new** Random();

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

TaskCreator creator = **new** TaskCreator();

creator.start();

TaskExecuter executer = **new** TaskExecuter();

executer.start();

RandomTaskExecuter randExecuter = **new** RandomTaskExecuter();

randExecuter.start();

ShortestTaskExecuter shortExecuter = **new** ShortestTaskExecuter();

shortExecuter.start();

executer.join();

randExecuter.join();

shortExecuter.join();

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

System.***out***.println("compleated tasks: " + ***compleatedTasks***.size());

System.***out***.println("all tasks: " + *allCreatedTasks*.get());

**final** Double tasksRatio = *getTasksRatio*(***compleatedTasks***.size(), *allCreatedTasks*.get());

System.***out***.println("Ratio compleated to all: " + String.*format*("%.2f", tasksRatio) + " %");

**final** Double avarageTime = *getAvarageTimeInSystem*(***compleatedTasks***);

System.***out***.println("Avarage task's time in system: " + String.*format*("%.2f", avarageTime) + " ms");

**final** Double reactionTime = *getSystemReaction*(***compleatedTasks***);

System.***out***.println("Avarage system reaction time: " + String.*format*("%.2f", reactionTime) + " ms");

**final** Double disp = *getDispersion*(avarageTime, ***compleatedTasks***);

System.***out***.println("Dispersion: " + String.*format*("%.2f", disp));

**final** Double actual = *getTotalActuality*(***compleatedTasks***);

System.***out***.println("Average actuality: " + String.*format*("%.2f", actual) + " %");

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

System.***out***.println("RANDOM:::::::::: ");

System.***out***.println("compleated tasks: " + ***compleatedTasksRand***.size());

System.***out***.println("all tasks: " + *allCreatedTasksRand*.get());

**final** Double tasksRatioRand = *getTasksRatio*(***compleatedTasksRand***.size(), *allCreatedTasksRand*.get());

System.***out***.println("Ratio compleated to all: " + String.*format*("%.2f", tasksRatioRand) + " %");

**final** Double avarageTimeRand = *getAvarageTimeInSystem*(***compleatedTasksRand***);

System.***out***.println("Avarage task's time in system: " + String.*format*("%.2f", avarageTimeRand) + " ms");

**final** Double reactionTimeRand = *getSystemReaction*(***compleatedTasksRand***);

System.***out***.println("Avarage system reaction time: " + String.*format*("%.2f", reactionTimeRand) + " ms");

**final** Double dispRand = *getDispersion*(avarageTimeRand, ***compleatedTasksRand***);

System.***out***.println("Dispersion: " + String.*format*("%.2f", dispRand));

**final** Double actualRand = *getTotalActuality*(***compleatedTasksRand***);

System.***out***.println("Average actuality: " + String.*format*("%.2f", actualRand) + " %");

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

System.***out***.println("SHORT!:::::::::::: ");

System.***out***.println("compleated tasks: " + ***compleatedTasksShort***.size());

System.***out***.println("all tasks: " + *allCreatedTasksShort*.get());

**final** Double tasksRatioShort = *getTasksRatio*(***compleatedTasksShort***.size(), *allCreatedTasksShort*.get());

System.***out***.println("Ratio compleated to all: " + String.*format*("%.2f", tasksRatioShort) + " %");

**final** Double avarageTimeShort = *getAvarageTimeInSystem*(***compleatedTasksShort***);

System.***out***.println("Avarage task's time in system: " + String.*format*("%.2f", avarageTimeShort) + " ms");

**final** Double reactionTimeShort = *getSystemReaction*(***compleatedTasksShort***);

System.***out***.println("Avarage system reaction time: " + String.*format*("%.2f", reactionTimeShort) + " ms");

**final** Double dispShort = *getDispersion*(avarageTimeShort, ***compleatedTasksShort***);

System.***out***.println("Dispersion: " + String.*format*("%.2f", dispShort));

**final** Double actualShort = *getTotalActuality*(***compleatedTasksShort***);

System.***out***.println("Average actuality: " + String.*format*("%.2f", actualShort) + " %");

}

**public** **static** **void** putTaskToStack(**final** Task newTask){

**synchronized** (*tasksStack*) {

//System.out.println("==== task's time: " + newTask.getTime());

**if**(*tasksStack*.size() <= newTask.getStackNumber()){

*tasksStack*.add(**new** LinkedList<>());

}

List<Task> currentTasksStack = *tasksStack*.get(newTask.getStackNumber());

**if** (currentTasksStack.size()<***listSize***) {

currentTasksStack.add(newTask);

} **else** {

*wasted*.incrementAndGet();

}

}

}

**public** **static** **void** putTaskToListRand(**final** Task newTask){

**synchronized** (*tasksListRand*) {

**if** (*tasksListRand*.size()<***listSize***) {

*tasksListRand*.add(newTask);

} **else** {

*wastedRand*.incrementAndGet();

//System.out.println("Wasted tasks: " + wasted.get());

}

}

}

**public** **static** **void** putTaskToListShort(**final** Task newTask){

**if** (*tasksListShort*.size()<***listSize***) {

*tasksListShort*.add(newTask);

} **else** {

*wastedShort*.incrementAndGet();

//System.out.println("Wasted tasks: " + wasted.get());

}

}

**public** **static** Task getFirstAvailableTask() {

**synchronized** (*tasksStack*) {

**for** (List<Task> stack : *tasksStack*) {

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

Task task = stack.get(0);

stack.remove(0);

**return** task;

}

}

**return** **null**;

}

}

**public** **static** Task getRandomTask() {

**synchronized** (*tasksListRand*) {

**if**(*tasksListRand*.size() > 0) {

**int** index = *randomGenerator*.nextInt(*tasksListRand*.size());

Task randomTask = *tasksListRand*.get(index);

*tasksListRand*.remove(index);

**return** randomTask;

}

**return** **null**;

}

}

**public** **static** Task getShitestTask() {

**synchronized** (*tasksListShort*) {

**if**(*tasksListShort*.size() > 0) {

**int** shortestIndex = 0;

Task shortTask = *tasksListShort*.get(0);

**for**(**int** i = 0; i< *tasksListShort*.size(); i++){

Task another = *tasksListShort*.get(i);

**if**(shortTask.getTime() > another.getTime()) {

shortTask = another;

shortestIndex = i;

}

}

Task shortestTask = *tasksListShort*.get(shortestIndex);

*tasksListShort*.remove(shortestIndex);

**return** shortestTask;

}

**return** **null**;

}

}

**public** **static** Double getTasksRatio(**int** createdTasks, **int** allTasks){

**return** ((**double**)createdTasks/(**double**)allTasks) \* 100;

}

**private** **static** Double getDispersion(**final** Double avarageTime, List<Task> compleatedTasks) {

**double** dispersion = 0;

**for**(Task task: compleatedTasks){

dispersion += Math.*pow*((task.getTimeInSystem() - avarageTime), 2);

}

**return** dispersion;

}

**private** **static** Double getAvarageTimeInSystem(List<Task> compleatedTasks){

**long** allTasksTime = 0;

**for**(Task task: compleatedTasks){

allTasksTime += task.getTimeInSystem();

}

**return** (**double**)allTasksTime/(**double**)compleatedTasks.size();

}

**private** **static** Double getSystemReaction(List<Task> compleatedTasks){

**long** systemRactionTime = 0;

**for**(Task task: compleatedTasks){

systemRactionTime += (task.getTimeInSystem() - task.getNumberOfExecutions() \* ***TAU***);

}

**return** (**double**)systemRactionTime/(**double**)compleatedTasks.size();

}

**private** **static** Double getTotalActuality(List<Task> compleatedTasks) {

**double** act = 0;

**for**(Task task: compleatedTasks) {

act += *getActuality*(task);

}

**return** act/(**double**)compleatedTasks.size() \* 100;

}

**private** **static** Double getActuality(Task task) {

**if**(task.getTimeInSystem() <= ***T1***) {

**return** (**double**) 1;

} **else** **if**(task.getTimeInSystem() > ***T2***) {

**return** (**double**) 0;

} **else** {

**return** (**double**) ((***T2*** - ***T1***) - (task.getTimeInSystem() - ***T1***)) / (***T2*** - ***T1***);

}

}

}

**package** lab1;

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

**public** Task(**final** **long** milliseconds){

**this**.time = milliseconds;

**this**.setCreationTime(System.*currentTimeMillis*());

}

**private** **long** time;

**private** **int** stackNumber = 0;

**private** **long** creationTime = 0;

**private** **long** endTime = 0;

**public** **long** getTime() {

**return** time;

}

**public** **void** setTime(**long** time) {

**this**.time = time;

}

**public** **int** getStackNumber() {

**return** stackNumber;

}

**public** **void** setStackNumber(**int** stackNumber) {

**this**.stackNumber = stackNumber;

}

**public** **long** getCreationTime() {

**return** creationTime;

}

**public** **void** setCreationTime(**long** creationTime) {

**this**.creationTime = creationTime;

}

**public** **long** getEndTime() {

**return** endTime;

}

**public** **void** setEndTime(**long** endTime) {

**this**.endTime = endTime;

}

**public** **long** getTimeInSystem(){

**return** endTime - creationTime;

}

**public** **long** getNumberOfExecutions(){

**return** stackNumber + 1;

}

}

**package** lab1;

**public** **class** TaskCreator **extends** Thread {

@Override

**public** **void** run(){

System.***out***.println("Starting to add tasks!");

**while**(!Lab1.*shouldFinish* || !Lab1.*shouldFinishShort* || !Lab1.*shouldFinishRand*){

//System.out.println("should finish: " + Lab1.shouldFinish);

//System.out.println("should finish rand: " + Lab1.shouldFinishRand);

waitForNextTask();

**final** Task newTask = **new** Task(generateTimeToExecute());

**if**(!Lab1.*shouldFinish*) {

Lab1.*putTaskToStack*(newTask);

Lab1.*allCreatedTasks*.incrementAndGet();

}

**if**(!Lab1.*shouldFinishRand*) {

Lab1.*putTaskToListRand*(newTask);

Lab1.*allCreatedTasksRand*.incrementAndGet();

}

**if**(!Lab1.*shouldFinishShort*) {

Lab1.*putTaskToListShort*(newTask);

Lab1.*allCreatedTasksShort*.incrementAndGet();

}

}

}

**private** **long** generateTimeToSleep(){

Double rand = Math.*random*();

**if**(rand == 0){

rand += 0.1;

}

**long** timeToSleep = (**long**) ((-1/Lab1.***LAMBDA*** \* Math.*log*(rand)) \* Lab1.***MILLISECONDS\_CONST***);

System.***out***.println("TaskCreator: Sleep for next Task creation: " + timeToSleep);

**return** timeToSleep;

}

**private** **long** generateTimeToExecute(){

Double rand = Math.*random*();

**if**(rand == 0){

rand += 0.1;

}

**return** (**long**) ((-1/Lab1.***MU*** \* Math.*log*(rand)) \* Lab1.***MILLISECONDS\_CONST***);

}

**private** **void** waitForNextTask() {

**try** {

Thread.*sleep*(generateTimeToSleep());

} **catch** (InterruptedException e) {

System.***out***.println("TaskCreator: Exeption happened while Thread.sleep");

}

}

}

**package** lab1;

**public** **class** TaskExecuter **extends** Thread {

@Override

**public** **void** run() {

**while**(!Lab1.*shouldFinish*) {

Task task = Lab1.*getFirstAvailableTask*();

*waitForTaskExecution*(task);

**if**(task != **null**) {

**if**(task.getTime() > Lab1.***TAU***) {

task.setTime(task.getTime() - Lab1.***TAU***);

task.setStackNumber(task.getStackNumber() + 1);

Lab1.*putTaskToStack*(task);

} **else** {

task.setEndTime(System.*currentTimeMillis*());

Lab1.***compleatedTasks***.add(task);

//System.out.println("------ TaskExecuter: task finished!");

**if**(Lab1.***compleatedTasks***.size() == Lab1.***MAX\_COUNT***){

Lab1.*shouldFinish* = **true**;

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

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

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

}

}

}

}

}

**public** **static** **void** waitForTaskExecution(Task task) {

**try** {

Thread.*sleep*(Lab1.TAU);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

}

**package** lab1;

**public** **class** RandomTaskExecuter **extends** Thread {

@Override

**public** **void** run() {

**while**(!Lab1.*shouldFinishRand*) {

Task task = Lab1.*getRandomTask*();

**if**(task != **null**) {

*waitForTaskExecution*(task);

task.setEndTime(System.*currentTimeMillis*());

Lab1.***compleatedTasksRand***.add(task);

**if**(Lab1.***compleatedTasksRand***.size() == Lab1.***MAX\_COUNT***){

Lab1.*shouldFinishRand* = **true**;

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

System.***out***.println("FINISHED RAND");

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

}

}

}

}

**public** **static** **void** waitForTaskExecution(Task task) {

**try** {

Thread.*sleep*(task.getTime());

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

}

**package** lab1;

**public** **class** ShortestTaskExecuter **extends** Thread {

@Override

**public** **void** run() {

**while**(!Lab1.*shouldFinishShort*) {

Task task = Lab1.*getShitestTask*();

**if**(task != **null**) {

*waitForTaskExecution*(task);

task.setEndTime(System.*currentTimeMillis*());

Lab1.***compleatedTasksShort***.add(task);

**if**(Lab1.***compleatedTasksShort***.size() == Lab1.***MAX\_COUNT***){

Lab1.*shouldFinishShort* = **true**;

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

System.***out***.println("FINISHED SHORT");

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

}

}

}

}

**public** **static** **void** waitForTaskExecution(Task task) {

**try** {

Thread.*sleep*(task.getTime());

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

}