Національний технічний університет України

«Київський політехнічний інститут»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

**Лабораторна робота № 2**

з дисципліни «Комп’ютерне моделювання»

Виконав студент групи ІО-01

*Редько Олександр*

# Завдання

Побудувати і змоделювати комп’ютерну систему за допомогою мережі Петрі.

# Програма

## Лістинг коду

**public** **class** Main {

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

Simulation sim = **new** Simulation();

sim.getMh();

**double** [] procent = **new** **double**[MatrixHolder.*getAveragetime*().length];

sim.showTopology();

System.*out*.println();

sim.simulate();

**for**(**int** i = 0; i < sim.getMh().getRealTime().length; i++) {

procent[i] = sim.getMh().getStatistics()[i]/sim.getTime();

}

System.*out*.println("Процент роботи пристроїв (по часу): ");

sim.showArray(procent, "procent");

}

}

**public** **class** MatrixHolder {

/\*\*

\* Коефіцієнт масштабу

\*/

**private** **final** **static** **double** *scale* = 1;

**private** **final** **static** String [] *namesOfDevices* = {"ЦП ", "ПнМ", "ПдМ", "ОП ", "ГП ", "ВП ", "USB", "КД "};

/\*\*

\* Середній час спрацювання

\*

\* ЦП, ПнМ, Пдм, ОП, ГП, ВП, USB, КД

\*/

**private** **final** **static** **double** [] *averageTime* = {

*scale* \* 1, *scale* \* 1.875, *scale* \* 3.468, *scale* \* 2.81,

*scale* \* 3.89, *scale* \* 93.75, *scale* \* 140, *scale* \* 190

};

/\*\*

\* Ймовірності

\*/

**private** **final** **static** **double** [][] *p* = {

{0.6, 1 - 0.6},

{0.40, 0.30, 0.21, 1 - 0.40 - 0.30 - 0.21},

{0.05, 0.2, 0.35, 1 - 0.2 - 0.35 - 0.05}

};

/\*\*

\* Топологія системи

\*

\* ЦП, ПнМ, Пдм, ОП, ГП, ВП, USB, КД

\*/

**private** **final** **static** **double** [][] *topology* = {

// ЦП ПнМ ПдМ ОП ГП ВП USB КД

/\*ЦП \*/ { *p*[0][0], 1, 1, 1, 1, 1, 1, 1 },

/\*ПнМ\*/ { *p*[1][1], 0, *p*[1][1]+*p*[1][3], *p*[1][1]+*p*[1][3]+*p*[1][0], 1, 1, 1, 1 },

/\*ПдМ\*/ { 0, *p*[2][3], 0, 0, 0, *p*[2][3]+*p*[2][1], *p*[2][3]+*p*[2][1]+*p*[2][0], 1 },

/\*ОП \*/ { 0, 1, 1, 1, 1, 1, 1, 1 },

/\*ГП \*/ { 0, 1, 1, 1, 1, 1, 1, 1 },

/\*ВП \*/ { 0, 0, 1, 1, 1, 1, 1, 1 },

/\*USB\*/ { 0, 0, 1, 1, 1, 1, 1, 1 },

/\*КД \*/ { 0, 0, 1, 1, 1, 1, 1, 1 }

};

/\*\*

\* Черга

\*/

**private** **int** [] queue;

/\*\*

\* Стан роботи

\*/

**private** **boolean** [] state;

/\*\*

\* Вільно/зайнято

\* false - вільно

\* true - зайнято

\*/

**private** **boolean** [] freely;

/\*\*

\* Реальний час, коли спрацює пристрій

\*/

**private** **double** [] realTime;

/\*\*

\* Статистика роботи пристрою

\*/

**private** **double** [] statistics;

**public** **int**[] getQueue() {

**return** queue;

}

**public** **void** setQueue(**int**[] queue) {

**this**.queue = queue;

}

**public** **boolean**[] getFreely() {

**return** freely;

}

**public** **void** setFreely(**boolean**[] freely) {

**this**.freely = freely;

}

**public** **double**[] getRealTime() {

**return** realTime;

}

**public** **void** setRealTime(**double**[] realTime) {

**this**.realTime = realTime;

}

}

**public** **class** Simulation {

/\*\*

\* Кількість задач

\*/

**private** **static** **final** **int** *tasks* = 6;

/\*\*

\* Початковий стан черги

\*/

**private** **static** **final** **int** [] *initStatusQueue* = {5, 0, 0, 0, 0, 0, 0, 0};

/\*\*

\* Нульовий початковий стан

\*/

**private** **static** **final** **boolean** [] *initStatusState* = {**true**, **false**, **false**, **false**, **false**, **false**, **false**, **false**};

**private** **static** **final** **double** [] *initNullDouble* = {0, 0, 0, 0, 0, 0, 0, 0};

/\*\*

\* Початковий стан флагу "Вільно/зайнято"

\*/

**private** **static** **final** **boolean** [] *initStateFreely* = {**false**, **true**, **true**, **true**, **true**, **true**, **true**, **true**};

**private** MatrixHolder mh = **new** MatrixHolder();

**private** **double** time;

/\*\*

\* Встановлюємо в початковий стан

\*/

**public** Simulation () {

setTime(0.0);

mh.setQueue(*initStatusQueue*);

mh.setState(*initStatusState*);

mh.setFreely(*initStateFreely*);

mh.setRealTime(*initNullDouble*);

mh.setStatistics(*initNullDouble*);

}

/\*\*

\* Симуляція роботи комп'ютерної системи

\*/

**public** **void** simulate() {

**while**(getTime() < 10000) {

loadTasks(getTime());

statistics(minRealTime(), getTime());

changeTokens(minRealTime());

}

}

/\*\*

\* Метод завантаження задач

\* **@param** time поточний час

\* **@return** час роботи пристроїв

\*/

**private** **void** loadTasks (**double** time) {

**double** [] tmp = **new** **double**[MatrixHolder.*getAveragetime*().length];

**for**(**int** i = 0; i < tmp.length; i++) {

tmp[i] = time + getPoisson(MatrixHolder.*getAveragetime*()[i]);

}

mh.setRealTime(tmp);

}

/\*\*

\* Шукаємо мінімальне значення T і номер пристрою

\* **@param** time поточний час

\* **@return** масив, в першому елементі зберігається min(T), в другому - номер пристрою

\*/

**public** **double**[] minRealTime() {

**boolean** [] state = mh.getState();

**double** [] realTime = **new** **double**[mh.getRealTime().length];

**int**[] numberOfDevice = **new** **int**[state.length];

**for**(**int** i = 0; i < state.length; i++) {

numberOfDevice[i] = state[i] ? 1 : 0;

**if**(state[i] == **true**) {

realTime[i] = mh.getRealTime()[i];

}

**else** {

realTime[i] = Double.*NaN*;

}

}

searchMin(realTime);

**return** searchMin(realTime);

}

/\*\*

\* Збір статистики. Від реального часу віднімаємо поточний і додаємо до матриці statistics

\* **@param** searchMin масив, в першому елементі зберігається min(T), в другому - номер пристрою

\* **@param** time поточний час

\*/

**public** **void** statistics(**double**[] searchMin, **double** time) {

**double**[] statistics = mh.getStatistics();

statistics[(**int**) searchMin[1]] -= getTime() - searchMin[0];

mh.setStatistics(statistics);

}

/\*\*

\* Перестановка фішок

\* **@param** searchMin масив, в першому елементі зберігається min(T), в другому - номер пристрою

\*/

**public** **void** changeTokens(**double**[] searchMin) {

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

**int** currentDevice = (**int**) searchMin[1];

**int**[] queue = mh.getQueue();

**boolean**[] state = mh.getState();

**boolean** [] freely = mh.getFreely();

goDevice:

**for**(**int** i = 0; i < MatrixHolder.*getTopology*().length; i++) {

**if**(r < MatrixHolder.*getTopology*()[currentDevice][i]) {

queue[currentDevice] = --queue[currentDevice];

state[currentDevice] = !state[currentDevice];

freely[currentDevice] = ! freely[currentDevice];

queue[i] = ++queue[i];

state[i] = !state[i];

freely[i] = ! freely[i];

**break** goDevice;

}

}

setTime(mh.getRealTime()[currentDevice]);

}

/\*\*

\* Пошук мінімума. Не числа (NaN) пропускаються

\* **@param** array

\* **@return**

\*/

**private** **double**[] searchMin(**double** [] array) {

**double** min = 0;

**for**(**int** j = 0; j < array.length; j++) {

**if**(!Double.*isNaN*(array[j])) {

min = array[j];

**break**;

}

}

**double**[] minDevice = **new** **double**[2];

**for**(**int** i = 0; i < array.length; i++) {

**if**(!Double.*isNaN*(array[i]) && (array[i] <= min)) {

min = array[i];

minDevice[0] = min;

minDevice[1] = i;

}

}

**return** minDevice;

}

/\*\*

\*

\* **@param** lambda інтенсивність потоку

\* **@return** число, розподілене за законом Пуассона

\*/

**public** **double** getPoisson(**double** lambda) {

**return** -Math.*log*(Math.*random*())/lambda;

}

**public** **void** showArray(**double**[] array, String procent) {

String[] namesOfDevices = MatrixHolder.*getNamesofdevices*();

**double**[] averageTime = MatrixHolder.*getAveragetime*();

**for**(**int** i = 0; i < array.length; i++){

System.*out*.printf("%2.6s",namesOfDevices[i]);

System.*out*.printf(" (%2.4g) ", averageTime[i]);

System.*out*.printf("%2.3f%%\n", 100 \* array[i]);

}

System.*out*.println();

}

**public** **void** showArray(**boolean** [] array) {

**for**(**int** i = 0; i < array.length; i++){

System.*out*.printf("%b ", array[i]);

}

System.*out*.println();

}

/\*\*

\* Друкує на екран матрицю топології

\*/

**public** **void** showTopology() {

**double**[][] topology = MatrixHolder.*getTopology*();

String[] namesOfDevices = MatrixHolder.*getNamesofdevices*();

System.*out*.println("Топологія:");

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

**for**(**int** i = 0; i < namesOfDevices.length; i++) {

System.*out*.printf("%4.3s ", namesOfDevices[i]);

}

System.*out*.println();

**for**(**int** i = 0; i < topology.length; i++) {

**for**(**int** j = 0; j < topology[i].length; j++) {

**if**(j == 0) {

System.*out*.printf("%4.3s",namesOfDevices[i]);

}

**if**(topology[i][j] == 0 || topology[i][j] == 1) {

System.*out*.printf("%6.0f", topology[i][j]);

}

}

System.*out*.println();

}

}

**public** MatrixHolder getMh() {

**return** mh;

}

**public** **double** getTime() {

**return** time;

}

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

**this**.time = time;

}

}