Ministerul Educaţiei al Republicii Moldova Universitatea Tehnică a Moldovei

Facultatea Calculatoare Informatica si Microelectronica Departamentul Ingineria Software si Automatica

**RAPORT**

Lucrarea de laborator nr.2 *Programarea in retea*

## *Tema:* Elemente ale procesarii concurente

A efectuat: st.gr. SI-171 , Buga Grigore

A verificat:

asist. univ. Cebotari Daria

Chisinau 2020

Scopul lucrarii:

Realizarea firelor de execuţie în Java/C#. Proprietăţile firelor. Stările unui fir de execuţie. Lansarea, suspendarea şi oprirea unui fir de execuţie. Grupuri de Thread-uri. Elemente pentru realizarea comunicării şi sincronizării.

Sarcina:



Studiind graful propus am stabilit dependenta dintre noduri.

Cu ajutorul semafoarelor am organizat executia thread-urilor.

Nodul 5 nu are nicio dependenta si lam setat simplu el fiind primul.

Nodurile 2,3,4 depind de nodul 5 la care le-am setat dependenta.

Nodul 1 este ultimul nod care depinde de celelalte noduri.

Codul sursa:

package prlab2var2;

import java.util.Random;

import java.util.concurrent.Semaphore;

public class pr2lab {

private static final Random random = new Random();

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

//crearea si initializarea semafoarelor

Semaphore sem1 = new Semaphore(0);

Semaphore sem2 = new Semaphore(0);

Semaphore sem3 = new Semaphore(0);

Semaphore sem4 = new Semaphore(0);

Semaphore sem5 = new Semaphore(2);

new Thread(new Runnable() {

public void run() {

Work("5",sem5);// 5 nu e dependent

}

}).start();

new Thread(new Runnable() {

//3 depinde de 5

public void run() {

Waitnwork("3",sem5,sem3);

}

}).start();

new Thread(new Runnable() {

//2 depinde de 5

public void run() {

Waitnwork("2",sem5,sem2);

}

}).start();

new Thread(new Runnable() {

//4 depinde de 5

public void run() {

Waitnwork("4",sem5,sem4);

}

}).start();

new Thread(new Runnable() {

public void run() {

try {

//1 depinde de 2,3 si 4

sem2.acquire();

sem3.acquire();

sem4.acquire();

Work("1",sem1);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}).start();

}

public static void Waitnwork(String id,Semaphore semToAquire,Semaphore semToRelease){

try {

semToAquire.acquire();

Work(id,semToRelease);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

public static void Work(String id, Semaphore semToRelease){

int millis = random.nextInt(10);

try {

Thread.sleep(millis); // sleep for time < 2000

System.out.printf("Thread nr. %s\n",id);

semToRelease.release(); // semaphore releases next thread

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

Executia:

