Sprawozdanie nr 2

Data ćwiczeń: 26.03.2015

Data oddania sprawozdania: 23.04.2015

* Tematyka

Na drugich zajęciach laboratoryjnych należało skorzystać z narzędzia zwanego semaforem binarnym, aby wyeliminować wyścig z pierwszego programu zamieszczonego w pierwszym sprawozdaniu.

Semafor binarny można porównać do semafora kolejowego. Jeśli szlaban jest opuszczony, to nikt nie może przejechać. Jeżeli podniesiony, to wszyscy mogą. W podobny sposób działają semafory binarne, które mają właśnie te dwa stany. Stanowią one klasyczną metodą kontroli dostępu do wspólnego zasobu. Po raz pierwszy zostały opisane przez wybitnego badacza Edsgera Djikstrę, który rozwinął w ten sposób algorytm Dekkera.

Semafory binarne nie są idealnym rozwiązaniem, ale pomagają zrozumieć istotę problemu jakim jest współdzielenie zasobu do wielu procesów i problemy z zadbaniem o to, aby ten zasób został przez nie wykorzystany prawidłowo

* Zadanie

Jak już wspomniano we wstępie, na zajęciach należało zmodyfikować kod z zajęć poprzednich, tak aby wyeliminować sytuację wyścigu.

* Kod

/\*\*

\* Created by Patryk on 2015-03-26.

\*/

public class Race {

public static void main(String[] args) {

Semaphore semaphore = new Semaphore();

Counter cnt = new Counter(0, semaphore);

IThread it = new IThread(cnt, semaphore);

DThread dt = new DThread(cnt, semaphore);

it.start();

dt.start();

try {

it.join();

dt.join();

} catch(InterruptedException ie) {

System.out.println(ie.getMessage());

}

System.out.println("value = " + cnt.value());

}

}

public class IThread extends Thread {

private Counter \_cnt;

private Semaphore semaphore;

public IThread(Counter c, Semaphore semaphore) {

\_cnt = c;

this.semaphore = semaphore;

}

public void run() {

for (int i = 0; i < 1000000; ++i) {

try {

this.semaphore.P();

\_cnt.inc();

} catch (InterruptedException e) {

e.printStackTrace();

} finally {

this.semaphore.V();

}

}

}

}

public class DThread extends Thread {

private Counter \_cnt;

private Semaphore semaphore;

public DThread(Counter c, Semaphore semaphore) {

\_cnt = c;

this.semaphore = semaphore;

}

public void run() {

for (int i = 0; i < 1000000; ++i) {

try {

this.semaphore.P();

\_cnt.dec();

} catch (InterruptedException e) {

e.printStackTrace();

} finally {

this.semaphore.V();

}

}

}

}

public class Counter {

private int \_val;

private Semaphore semaphore;

public Counter(int n, Semaphore semaphore) {

\_val = n;

this.semaphore = semaphore;

}

public void inc() {

\_val++;

}

public void dec() {

\_val--;

}

public int value() {

return \_val;

}

}

public class Semaphore {

private boolean \_stan;

private int \_czeka;

public Semaphore() {

this.\_stan = true;

this.\_czeka = 0;

}

public synchronized void P() throws InterruptedException { //opuszczenie semafora - acquire

\_czeka++;

while(!\_stan) {

try {

wait();

} catch (InterruptedException ie) {

System.out.println(ie.getMessage());

}

}

\_czeka--;

\_stan = false;

}

public synchronized void V() {

//podniesienie semafora - release

if(\_czeka > 0) {

this.notify();

}

\_stan = true;

}

}