# Rinnakkaisohjelmointi, Syksy 2006

17.12.2006

Opintopiiri WTF

Mika Holmström
Paula Kemppi
Janne Piippo
Lasse Lukkari

# Javan semaforit

## 1. Menetelmän käyttötarkoitus ja sovellusalue

Semaforin idea kehitettiin jo noin 40 vuotta sitten, kun mietittiin tehokkaampia ja parempia keinoja hoitaa rinnakkaisuudesta aiheutuvia synkronointiongelmia moniajojärjestelmissä (Dijkstra 1968). Yleisenä ongelmana on yhteisessä käytössä olevat resurssit, joiden samanaikainen käyttö saattaa aiheuttaa tiedon katoamista tai väärentymistä. Tällaiset tilanteet voidaan estää semaforeilla: päästetään vain yksi tai erikseen ilmoitettu määrä prosesseja, Javassa säikeitä, käsittelemään kriittistä resurssia, jolloin konflikteja ei synny.

Semaforien käyttöön ja siihen, että resursseja varataan ja estetään tiettyjen prosessien eteneminen, liittyy kuitenkin lukkiutumisongelma. Lukkiutuminen tapahtuu silloin, kun tietty prosessi ei pääsee koskaan takaisin suoritukseen, sillä sen haluama resurssi on jatkuvasti jollakin toisella prosessilla käytössä. Klassinen esimerkki tällaisesta tilanteesta on aterioivat filosofit: viisi haarukkaa ja viisi filosofia. Jokainen tarvitsee kaksi haarukkaa syödäkseen, mutta mitä jos naapuri ehtii ensin?

Semaforeiden heikkoutena on niiden herkkyys ohjelmointivirheille. Ohjelmoijan on helppo unohtaa vapauttaa semafori, jonka seurauksena koko järjestelmä saattaa lukkiutua. Tämän takia on kehitelty muita "ystävällisempiä" tapoja hallita rinnakkaisuutta, mmm. monitorit.

## 2. Menetelmän perusidea

Idea semaforiin saatiin rautateillä olevasta tekniikasta estää junien törmäykset: semafori ylläpitää kokonaislukumuuttujaa, joka kertoo kullakin hetkellä jäljellä olevien lupien (permits) määrän, eli siis sen kuinka monta säiettä voi toimia samanaikaisesti. Kun säie haluaa jollekin kriittiselle alueelle, se pyytää lupaa tietyltä semaforilta, joka myöntää luvan mikäli lupia on jäljellä. Muutoin säie joutuu odotamaan luvan vapautumista. Lupia vapautuu, kun mahdolliset muut säikeet lopettavat toimintansa kriittisellä alueella ja ilmoittavat siitä semaforille.

Semaforeita on kahta eri tyyppiä: binääri- ja yleissemafori. Binäärisemaforissa semaforin arvo vaihtelee ykkösen ja nollan välillä, joka tarkoittaa sitä, että tietyllä alueella voi olla vain yksi prosessi suorituksessa kerrallaan. Yleisessä semaforissa lupa-arvo voi puolestaan olla mikä tahansa kokonaisluku ja täten samalle alueelle voi päästä useampia prosesseja samaan aikaan.

## 3. Menetelmän yhteneväisyydet/eroavaisuudet kurssilla esitettyyn perustapaukseen verrattuna

Semaforin perusmääritelmän [BenA06] mukaan semaforille annettavan kokonaisluvun arvon tulee olla nolla tai sitä suurempi, mutta Javassa on kuitenkin mahdollista antaa semaforia alustaessa myös negatiivisa arvoja. Käytännössä tämä tarkoittaa sitä, että semaforin ohi pääsee vasta kun tietty määrä lupia on saatu takaisin (released) ja semaforin lupa-arvo on suurempi kuin nolla.

[BenA06] määritelmän mukaan perussemafori valitsee seuraavaksi suoritukseen pääsevän prosessin sattumanvaraisesti kaikista halukkaista prosesseista. Javassakin voi käyttää tällaista epäreilua tapaa, mutta on myös mahdollista luoda reiluja semaforeja (fair semaphores), jotka päästävät suoritukseen sen säikeen, joka on jonotanut kauoiten FIFO-jonossa (First In, First Out).

## 4. Esimerkit

### a) Aterioivat filosofit

"Viisi filosofia istuu pyöreän pöydän ympärillä. Jokaisella filosofilla on edessään lautasellinen spagettia. Spagetti on niin liukasta, että kukin filosofi tarvitsee kaksi haarukkaa syödäkseen. Ideaalisten filosofiemme elämä koostuu ainoastaan syömisestä ja ajattelemisesta. Kun filosofi tulee nälkäiseksi hän yrittää saada itselleen lautasen oikealla ja vasemmalla puolella olevan haarukan. Jos haarukoiden saanti onnistuu, hän syö hetken, laittaa haarukat takaisin pöydälle ja jatkaa miettimistään." [Wikipedia]

Katso liite 3.

### b) Nukkuvan parturin ongelma

Tässä skenaariossa on parturi, parturin tuoli ja tuoleja odotaville asiakkaille. Kun asiakkaita ei ole paikalla, niin parturi istuu tuoliinsa ja alkaa nukkua. Heti kun asiakas saapuu, hän joko herättää parturin tai jos parturi on leikkaamassa jonkun toisen hiuksia, asiakas istuu odotushuoneen tuolille. Jos kaikki tuolit ovat varattuja, niin juuri saapunut asiakas poistuu liikkeestä ja yrittää ehkä myöhemmin uudestaan.

Katso liite 4.

## 5. Menetelmän käyttöohje "rautalangasta väännettynä"

### Tarvittavat pakkaukset

Jotta Semaphore luokkaa päästäisiin käyttämään on luokka java.util.concurrent.Semaphore tuotava käyttöön (import) lauseella:

import java.util.concurrent.Semaphore;

tai vaihtoehtoisesti, jos halutaan käyttää muitakin pakkauksen luokkia, voidaan kaikki pakkauksen luokat tuoda lauseella:

import java.util.concurrent.*;

### Semaphore-luokan konstruktorit

Semaphore -luokan konstruktori on kuormitettu ja siitä on kaksi versiota
Ensimmäinen konstruktoreista on muotoa:

Semaphore(int lupia)

Kontruktorille annetaan parametrina lupien määrän alkuarvo kokonaislukuna. Lupien määrä voi olla myös negatiivinen. Luotu semafori on epäreilu (unfair).

Toinen konstruktoreista on taas muotoa:
Semaphore(int lupia, boolean reilu)

Konstruktorille annetaan parametreina lupien määrän alkuarvo kokonaislukuna kuten ensimmäisessäkin konstruktorissa. Lisäksi annetaan totuusarvo, jolla semafori voidaan määrittää reiluksi. Asettamalla arvoksi true semaphroista tulee reilu, jos taas flase semaforista tulee epäreilu, kuten ensimmäiselläkin konstruktorilla

**Semaphore luokan keskeiset metodit**

void acquire()
Pyytää semaforilta lupaa. Jää odottamaan kunnes lupa edetä on saatu tai säie keskeytetään

void acquire(int lupia)
Pyytää semaforilta parametrina annetun kokonaisluvun määrän lupia. Jää odottamaan kunnes kaikki luvat on saatu tai kunnes säie keskeytetään

int availablePermits()
Palauttaa kokonaislukuna jäljellä olevien lupien määrän.

protected  void reducePermits(int vähennys)
Vähentää lupien määrää parametrina annetun kokonaisluvun verran.

void release()
Vapauttaa semaforin lukituksen ja kasvattaa semaforin lupien määrää yhdellä

void release(int lupia)
Vapauttaa semaforin lukituksen ja kasvattaa semaforin lupien määrää parametrina annetun kokonaisluvun verran.

boolean tryAcquire()
Pyytää lupaa semaforilta, mutta ei jää odottamaan, jos lupia ei ole jäljellä

boolean tryAcquire(int lupia)
Pyytää kokonaislukuna annetun parametrin verran lupia semaforilta, mutta ei jää odottamaan, jos lupia ei ole jäljellä koko tarvittavaa määrää

loput metodit löytyvät osoitteesta:
http://java.sun.com/j2se/1.5.0/docs/api/java/util/concurrent/Semaphore.html

**Pelkistetty koodiesimerkki**

```
import java.util.concurrent.Semaphore;

public class MinunLuokka {

    Semaphore sema = new Semaphore(1, true);

    public MinunLuokka() {
        //luokan normaali konstruktori
    }

    public void suojattuMetodi() {
        try {
            // Pyydetään semaforilta lupaa. Jos lupaa ei ole heti saatavilla,
            // jäädään odottamaan sitä.

        this.sema.acquire();

        // Kriittinen alue
        // Kun semaforilta on saatu lupa, voidaan nyt olla varmoja etteivät
```

```
        // muut säikeet ole samaan aikaan suorittamassa koodia tällä
        // semaforilla suojatulla alueella.

        } catch (InterruptedException e) {
            //poikkeuksen käsittely
        }

        // Kun tarvittavat semaforin suojissa tehtävät operaatiot on suorittetu
        // palautetaan "lupalappu" jotta muut säikeet pääsivät vuorollaan
        // suorittamaan.

        this.sema.release();
    }
}
```

***LIITE 2***

## Ohjelmat

## Aterioivat Filosofit
Ks. Liite 3

Sisältää luokat:
**`AterioivatFilosofit.java`**
**`Filosofi.java`**
**`Poyta.java`**

## Sleeping Barber
ks. Liite 4

Sisältää luokat:
**`Barber.java`**
**`Customer.java`**
**`SleepingBarber.java`**
**`Waitingroom.java`**

## Kertauskysymys 1
ks Liite 5

## Kertauskysymys 2
ks. Liite 6

## Kertauskysymykset html-sivuna:
http://cs.helsinki.fi/u/prautio/RIO/pract.html

## *LIITE 3*

## Aterioivat filosofit

```
public class AterioivatFilosofit {
	public static void main(String[] args) {
		int filosofeja = 5;
		Poyta poyta = new Poyta(filosofeja);
		Filosofi[] filosofit = new Filosofi[filosofeja];

		for (int i = 0; i < filosofeja; i++){
			filosofit[i] = new Filosofi(poyta,i);
		}

		for (int i = 0; i < filosofeja; i++){
			filosofit[i].aloita();
		}
	}
}


public class Filosofi implements Runnable {
	static int kertojaYhteensa;
	private int kertoja;
	private Thread t = new Thread(this);
	private Poyta poyta;
	private int numero;

	public Filosofi(Poyta poyta, int numero){
		this.poyta = poyta;
		this.numero = numero;
		System.out.println("Filosofi numero " + numero + " luotu");
	}

	private void syo(){

		if (poyta.otaHaarukat(numero)){
			this.kertoja++;
			kertojaYhteensa++;
			System.out.println("Filosofi numero " + numero + " on syönyt " +
kertoja +" kertaa (" + kertojaYhteensa + ")" );
		mieti();
			poyta.palautaHaarukat(numero);
			System.out.println("Filosofi numero " + numero + " palautti
haarukat");
		}
	}

	private void mieti(){
		try{
		t.sleep((int) ( Math.random()*100));
		}
		catch (InterruptedException e){
		}
	}

	public void aloita(){
		t.start();
	}

	public void run(){
		while(true){
		syo();
		mieti();
		}

	}

	public static void main(String[] args) {
	}
}
```

```
import java.util.concurrent.Semaphore;

public class Poyta {

	private int koko;
	private Semaphore[] haarukat;
	private boolean[] kaytossa;
	private boolean[] syomassa;

	public Poyta(int koko) {
		this.haarukat = new Semaphore[koko];
		this.kaytossa = new boolean[koko];
		this.syomassa = new boolean[koko];
		for (int i = 0; i < koko; i++) {
			this.haarukat[i] = new Semaphore(1, true);
			this.kaytossa[i] = false;
			this.syomassa[i] = false;
		}
		this.koko = koko;
	}

	public boolean otaHaarukat(int numero) {

		int arpaluku = (int) (Math.random() + 0.5);

		if (arpaluku == 0) {
			try {
				haarukat[numero].acquire();
				this.kaytossa[numero] = true;

				if (!kaytossa[(numero + 1) % koko]) {
					haarukat[(numero + 1) % koko].acquire();
					this.kaytossa[(numero + 1) % koko] = true;
					this.syomassa[numero] = true;
					return true;
				}
				else {
					kaytossa[numero] = false;
					haarukat[numero].release();
					return false;
				}
			}
			catch (Exception e) {
			}
		}

		else {
			try {
				haarukat[(numero + 1) % koko].acquire();
				this.kaytossa[(numero + 1) % koko] = true;

				if (!kaytossa[numero]) {
					haarukat[numero].acquire();
					this.kaytossa[numero] = true;
					this.syomassa[numero] = true;
					return true;
				}
				else {
					kaytossa[(numero + 1) % koko] = false;
					haarukat[(numero + 1) % koko].release();
					return false;
				}
			}
			catch (Exception e) {
			}

		}
		return false;
	}

	public void palautaHaarukat(int numero) {
		this.syomassa[numero] = false;
		kaytossa[numero] = false;
		haarukat[numero].release();
		kaytossa[(numero + 1) % koko] = false;
		haarukat[(numero + 1) % koko].release();
```

```
	}

	public String toString() {
		String palautettava = "";
		for (int i = 0; i < this.koko; i++) {
			palautettava = palautettava + "[" + this.syomassa[i] + "]";

		}
		if ((palautettava + "[" + this.syomassa[0] + "]")
				.indexOf("[true][true]") != -1)
			return "Virhe! Vierekkäiset filosofit syömässä!" + palautettava;
		return palautettava;
	}

	public static void main(String[] args) {
	}
}
```

## *LIITE 4*

## Sleeping Barber

```
public class Barber extends Thread {

    Semaphore customers, barber, accessSeats;
    Waitingroom room;

    /** Parametrinä saadaan kaikki tarvittavat
     * semaforit sekä viittaus odotushuone-olioon */
    public Barber(Semaphore customers, Semaphore barber,
                  Semaphore accessSeats, Waitingroom room) {
        this.customers = customers;
        this.barber = barber;
        this.accessSeats = accessSeats;
        this.room = room;
    }

    public void run() {
        while(true) {
            try {
                customers.acquire();        // Nukutaan kunnes asiakas saapuu
                    accessSeats.acquire();  // Parturi hereillä ja asiakas paikalla,
tuolien tilaa muutettava
                        room.releaseSeat(); // Asiakas poistui jonotustilan tuolilta
                        barber.release();   // Parturi on valmiina leikkaamaan
                    accessSeats.release();  // Annetaan uuden asiakkaan istua
odotushuoneeseen

                // Parturi leikkaa hiuksia, jonka jälkeen
                // hän palaa odottaamaan uutta asiakasta

            } catch (InterruptedException ie) {}
        }
    }
}

public class Waitingroom {

    private int freeSeats;

    /** Luodaan odotushuone ja sinne tietty määrä tuoleja */
    public Waitingroom(int freeSeats) {
        this.freeSeats = freeSeats;
    }

    /** Onko vapaita paikkoja */
    public boolean gotFreeSeats() {
        if (freeSeats > 0)
            return true;
        return false;
    }

    /** Yksi tuoli vapautui käytöstä */
    public void releaseSeat() {
        freeSeats++;
    }

    /** Varataan tuoli, muttei tarkisteta onko
     * vapaita, se tarkastetaan muualla */
    public void takeSeat() {
        freeSeats--;
    }
}
```

```
import java.util.concurrent.Semaphore;

public class Customer extends Thread {

    private static int nextCustomerId = 0;  // Asiakkaiden numerointiin
    private Semaphore customers, barber, accessSeats;
    private Waitingroom room;
    private boolean notCut; // Onko hiukset leikkaamatta
    private int customerId; // Jokaisella on ID tulostusta varten

    /** Parametrinä saadaan kaikki tarvittavat
     * semaforit sekä viittaus odotushuone-olioon */
    public Customer(Semaphore customers, Semaphore barber,
                    Semaphore accessSeats, Waitingroom room) {
        this.customers = customers;
        this.barber = barber;
        this.accessSeats = accessSeats;
        this.room = room;
        notCut = true;
        customerId = nextCustomerId++;
    }

    public void run() {
        try {
            // Asiakas yrittää niin kauan, että saa tyhjän
            // tuolin ja pääsee viimein parturoitavaksi
            while(notCut) {
                accessSeats.acquire();  // asiakas yrittää päästä käsiksi tuoleihin
                    if (room.gotFreeSeats()) {  // jos löytyi tyhjä tuoli
                        room.takeSeat();        // istutaan tuolille
                        customers.release();    // kerrotaan, että ollaan valmiina
parturoitavaksi
                        accessSeats.release();  // enää ei tarvita tuoleja
                        barber.acquire();       // odotetaan jos parturi on kiireinen
                        notCut = false;

                        System.out.println("Asiakas["+ customerId +"]: Tack så mycket,
parturi!");
                    }
                    else {    // ei tyhjiä tuoleja
                        accessSeats.release();  // annetaan muidenkin yrittää istumista
                    }
            }
        } catch (InterruptedException ie) {}
    }
}
```

```
import java.util.concurrent.Semaphore;

public class SleepingBarber {

    // Montako tuolio odotushuoneessa on
    private static final int MAX_CUSTOMERS = 3;
    // Kertoo onko parturi vapaa vai ei
    static Semaphore barber = new Semaphore(1,true);
    // Kertoo odotushuoneessa olevien asiakkaiden määrän
    static Semaphore customers = new Semaphore(0, true);
    // Poissulkee sen, ettei yhdelle tuolille voi tulla kahta asiakasta.
    static Semaphore accessSeats = new Semaphore(1, true);
    // Huone, johon asiakkaat menevät odottamaan (tuolit ovat täällä)
    static Waitingroom room = new Waitingroom(MAX_CUSTOMERS);

    public static void main(String[] args) {
        // Luodaan parturi ja muutama asiakas testausta varten
        Barber sleepingBarber = new Barber(barber, customers, accessSeats, room);
        Thread[] clients = {
            new Customer(barber, customers, accessSeats, room),
            new Customer(barber, customers, accessSeats, room),
            new Customer(barber, customers, accessSeats, room),
            new Customer(barber, customers, accessSeats, room),
            new Customer(barber, customers, accessSeats, room),
            new Customer(barber, customers, accessSeats, room),
            new Customer(barber, customers, accessSeats, room),
            new Customer(barber, customers, accessSeats, room),
            new Customer(barber, customers, accessSeats, room),
            new Customer(barber, customers, accessSeats, room),
            new Customer(barber, customers, accessSeats, room),
            new Customer(barber, customers, accessSeats, room),
            new Customer(barber, customers, accessSeats, room),
            new Customer(barber, customers, accessSeats, room),
            new Customer(barber, customers, accessSeats, room)};

        // Käynnistettään parturi- ja asiakassäikeet
        sleepingBarber.start();
        for (int i=0; i<clients.length; i++)
            clients[i].start();
    }

}
```