SISTEMI OPERATIVI IIN/IEL/IDT

INFORMATICA INDUSTRIALE E SISTEMI OPERATIVI IDI

SISTEMI DI ELABORAZIONE P.O.

prova scritta del 12.11.2004

Nome: _____ Cognome: _____

In un sistema, *N* istanze di due tipi di processo produttore P_A, e P_B, depositano rispettivamente prodotti di tipo A e *B,* in due distinti vettori di dimensione D=2. Se possibile, i processi produttore generano e inseriscono il prodotto nei vettori, altrimenti si sospendono.

Nel sistema esistono anche *M* istanze di due tipi di processo consumatore C_A e C_{AB}, che per svolgere il proprio compito necessitano, rispettivamente, di una risorsa di tipo *A* (C_A), di una risorsa di tipo A ed una di tipo B (C_{AB}). Un processo consumatore, se possibile, preleva i prodotti necessari alla propria elaborazione ed esegue, altrimenti rimane in attesa.

La politica di servizio dei processi consumatore, dovrebbe soddisfare il vincolo per cui nessun processo C_A deve mai aspettare a causa di processi C_{AB} sospesi con una risorsa *A* ed in attesa di una risorsa *B*.

Si scriva la soluzione Java che consenta ai processi produttore e consumatore di coordinarsi secondo la politica indicata.

Soluzione

```
public class Principale
{
	public static void main(String args[]) {
		int N, M;		// costanti o assegnati come argomenti da linea di comando
		Vettore va = new Vettore();
		Vettore vb = new Vettore();
		for (int i=0;i<N;i++) {
			Produttore Pa = new Produttore(va);
			Produttore Pb = new Produttore(vb);
			Pa.start();
			Pb.start();
		}

		for (int j=0;j<M;j++) {
			Consumatore Ca = new Consumatore(va);
			Consumatore Cab = new Consumatore(va, vb);
			Ca.start();
			Cab.start();
		}
	}
}


public class Vettore
{
	private Object v[];
	private int count;

	public Vettore() {
		v = new Object[2];
		count = 0;
	}

	public synchronized void inserisci(Object item) {
		while (count==2) {
			try {
				wait();
			}
			catch (InterruptedException e) {}
		}
		v[count] = item;
		count++;
		notifyAll();
	}

	public synchronized Object preleva() {
		while (count==0) {
			try {
				wait();
			}
			catch (InterruptedException e) {}
		}
		Object item = v[count];
		count--;
		notifyAll();
		return item;
	}
```

```
	public static void napping() {
		sleeptime = (int) 1000*Math.random();
		try {
			Thread.sleep(sleeptime);
		}
		catch (InterruptedException e) {}
	}

}

public class Produttore extends Thread
{
	private Vettore v;

	public Produttore(Vettore vect) {
		v = vect;
	}

	public void run() {
		Date message;
		while (true) {
			message = new Date();
			v.inserisci(message);
			Vettore.napping();
		}
	}
}

public class ConsumatoreA extends Thread
{
	private Vettore v;

	public ConsumatoreA(Vettore vect) {
		v = vect;
	}

	public void run() {
		while (true) {
			Object item = v.preleva();
			Vettore.napping();
		}
	}
}

public class ConsumatoreAB extends Thread
{
	private Vettore va;
	private Vettore vb;

	public ConsumatoreAB(Vettore vecta, Vettore vectb) {
		va = vecta;
		vb = vectb;
	}

	public void run() {
		while (true) { // per soddisfare la politica è sufficiente prendere prima B e poi A
			Object itemb = vb.preleva();
			Object itema = va.preleva();
			Vettore.napping();
		}
	}
}
```