SISTEMI OPERATIVI IDT/IEL: seconda prova intermedia, 17 giugno 2005

Esercizio 1

Su un sistema sono disponibili 4 tipologie di risorse (A, B, C, e D), con una singola istanza per ciascuna tipologia. In un determinato periodo di tempo operano sul sistema 3 processi (P1, P2, e P3).

Le necessità dei processi in termini di risorse sono sintetizzate nella seguente tabella, dove un segno “X” indica che un processo può richiedere una particolare risorsa durante la propria esecuzione:

	A	B	C	D
P1	X		X	
P2	X	X		
P3		X	X	X

Ad un certo istante ai diversi processi risultano già allocate alcune risorse, come indicato nella seguente tabella:

	A	B	C	D
P1	X			
P2		X		
P3				

Nelle condizioni descritte sopra

a) si determini se il sistema si trova in uno stato sicuro;

b) qualora la risposta al punto a) sia affermativa, si indichi una sequenza sicura;

c) qualora la risposta al punto b) sia affermativa, si indichi se può essere accolta una richiesta di P3 per un'istanza della risorsa di tipo C;

d) sempre nell'ipotesi che la risposta al punto b) sia affermativa, ed in alternativa al punto c), si indichi se può essere accolta una richiesta di P3 per un'istanza della risorsa di tipo D.

Si giustifichino adeguatamente le risposte fornite.

Esercizio 2

All'interno di un programma Java viene creato un insieme di threads che operano secondo lo schema descritto qui di seguito:

```
public void run() {
 /* ... operazioni che non richiedono l'accesso alle risorse */
 try {
       /* ... richiesta delle istanza della risorsa necessarie
             allo svolgimento del compito assegnato a thread... */
       r.richiediIstanze( i );
       /* ... utilizzo delle istanze della risorsa ... */
       Thread.sleep( (int) ( 10000. * Math.random() ) );
 } catch ( InterruptedException ie ) { ... }
 /* ... rilascio delle istanze della risorsa ... */
 r.rilasciaIstanze( i );
 /* ... altre operazioni che non richiedono l'accesso alle risorse */
}
```

dove `r` rappresenta un oggetto di tipo `Risorsa`, utilizzato per accedere in maniera ordinata alle diverse istanze di una determinata tipologia di risorse.

Nell'ipotesi (non garantita nella realtà, ma supposta vera ai fini di questo esercizio) che in Java i threads che si sospendono su un determinato oggetto con il metodo `wait()` siano poi sbloccati seguendo una politica FIFO, si realizzi la classe `Risorsa`, in maniera tale che:

- se le istanze della risorsa sono disponibili al momento della richiesta, queste vengono immediatamente allocate al thread richiedente;
- se le istanze richieste non sono disponibili, la richiesta viene registrata in un'opportuna struttura dati ed il thread viene sospeso;
- quando un thread restituisce istanze della risorsa controllata, nel rispetto della politica specificata sopra per sbloccare i threads sospesi, si verifica iterativamente se è possibile soddisfare alcune richieste pendenti precedentemente registrate ricorrendo alle istanze che sono disponibili a seguito della restituzione; in caso affermativo si procede a sbloccare i relativi threads.

Soluzione

Esercizio 1

In questo caso particolare (un'istanza per ogni tipo di risorsa) il problema può essere risolto sia con l'algoritmo del grafo di allocazione, sia con l'algoritmo del banchiere.
a) sì
b) P1-P2-P3
c) no
d) sì

Esercizio 2

```
public class Risorsa {
	public Risorsa ( int istanzeDisponibili ) {
		this.istanzeDisponibili = istanzeDisponibili;
	}
	public synchronized void richiediIstanze( int istanze ) {
		if ( istanze > istanzeDisponibili )
		{
			pendenti.add( new Integer( istanze ) );
			try {
				wait();
			} catch ( InterruptedException ie ) {...}
		}
		else
		{
			istanzeDisponibili -= istanze;
		}
	}
	public synchronized void rilasciaIstanze( int istanze ) {
		istanzeDisponibili += istanze;
		while ( pendenti.size() > 0 ) {
			Integer i = (Integer) pendenti.getFirst();
			int richiesta = i.intValue();
			if ( richiesta <= istanzeDisponibili )
			{
				pendenti.removeFirst();
				istanzeDisponibili -= richiesta;
				notify();
			}
			else
				return;
		}
	}
	public int istanzeDisponibili() { return istanzeDisponibili; };
	private int istanzeDisponibili;
	private LinkedList pendenti = new LinkedList();
}
```