

SISTEMI OPERATIVI IDI: Seconda prova intermedia, 15 aprile 2005

Esercizio 1

Sia dato un sistema che prevede risorse di tipo R_A , R_B , R_C , e R_D , delle quali sono presenti rispettivamente 3, 2, 4, e 4 istanze. Sul sistema si trovano in esecuzione 4 processi (P_1 , P_2 , P_3 e P_4) le cui richieste massime per le suddette risorse sono indicate nella tabella sottostante:

	R_A	R_B	R_C	R_D
P_1	2	1	0	1
P_2	2	2	2	3
P_3	0	1	2	4
P_4				

All'istante t_0 alcune istanze delle suddette risorse sono allocate ai processi, come indicato di seguito:

	R_A	R_B	R_C	R_D
P_1	1	0	0	0
P_2	1	1	1	0
P_3	0	0	1	2
P_4	0	0	0	1

Applicando l'algoritmo del banchiere

- si determini se il sistema si trova in uno stato sicuro, ed eventualmente si individui una sequenza sicura;
- nell'ipotesi che il processo P_2 richieda un'ulteriore risorsa di tipo R_C , si verifichi se questa può essere accolta.

Esercizio 2

Sia data una classe Java `Distributore` così definita:

```
public class Distributore {  
    ...  
    public synchronized Object gettone() {...}  
}
```

che attraverso il metodo `gettone()` restituisce un gettone con un identificativo progressivo univoco. Sia inoltre data una classe Java `CodaIllimitataOrdinata` così definita:

```
public class CodaIllimitataOrdinata {  
    ...  
    public void chiediServizio( Object g ) {...}  
    public Object serviUtente() {...}  
    public boolean vuota() {...}  
}
```

Tramite un'istanza della classe `CodaIllimitataOrdinata` gli utenti di un servizio possono fare richiesta dello stesso depositando un gettone `g` tramite il metodo `chiediServizio(...)`; gli inservienti preposti al servizio ottengono il gettone relativo al prossimo utente da servire tramite il metodo `serviUtente()`; il metodo `vuota()` restituisce vero se la coda è vuota, ovvero se non ci sono al momento utenti da servire.

Si realizzino le seguenti classi che, tramite l'uso di threads, consentano la creazione di utenti e inservienti che operano come descritto:

- `Utente`, che preleva un gettone e lo deposita in una coda (di tipo `CodaIllimitataOrdinata`) e quindi attende di essere servito; una volta servito l'utente termina la propria esecuzione;
- `Inserviente`, che, ciclicamente, se ci sono utenti da servire, individua il prossimo utente da servire ottenendo il relativo gettone da una coda, e lo sblocca; se non ci sono utenti da servire, attende il prossimo utente.

Si realizzi infine un programma Java che istanzi un oggetto di tipo `Distributore`, un oggetto di tipo `CodaIllimitataOrdinata`, avvii `N` threads di tipo `Inserviente` e, indefinitamente, avvii threads di tipo `Utente`.