PROGETTO JAVA - A.A 2011/12

Il progetto è composto da 4 parti da svolgere un stretta sequenza:

Parte prima

- Implementare la classe **`FairSemaphore`** che definisce un semaforo generale assicurando che la gestione dei thread sospesi sul semaforo sia rigorosamente First-In-First-Out.
- Per l'implementazione della classe **`FairSemaphore`** devono essere utilizzati esclusivamente i meccanismi di sincronizzazione offerti da Java versione 4.0 (blocchi e/o metodi `synchronized`, metodi `wait()`, `notify()` e `notifyll()`).

Parte seconda

- Utilizzando come meccanismo di sincronizzazione esclusivamente i semafori (come implementati nella parte prima), implementare la classe **`SynchPort`** che simula una porta sincronizzata.
- La porta deve essere generica rispetto al tipo T di informazione scambiata (`class SynchPort<T>`)
- Ogni porta deve essere dichiarata come oggetto `public` locale al thraed (processo) ricevente. La porta deve essere infatti visibile a ciascuno dei processi mittenti che dovranno invocare il metodo `send` sulla specifica porta.
- Il numero di mittenti deve essere passato coma parametro al costruttore della porta.
- Il metodo `receive` deve restituire al processo ricevente l'informazione di tipo `<T>`contenuta nel messaggio ricevuto e l'identificazione del processo mittente.

Parte terza

- Implementare la classe **`PortVector`** che definisce un array di porte sincronizzate da usare poi, nella parte quarta, come array di porte attraverso le quali un processo server riceve richieste di servizio. Per questo, locale ad un processo server verrà dichiarato un oggetto (`public`) di tipo **`PortVector`**
- Ogni processo mittente (cliente) invia i propri messaggi ad una delle porte dell'array identificata, oltre che dall'identificatore dell'oggetto di tipo **`PortVector`** anche dal suo indice nell'array stesso.
- Il numero `N` delle porte componenti il vettore è passato come parametro al costruttore della classe.
- Per semplicità, tutte le porte del vettore vengono definite con lo stesso numero di processi mittenti (il numero dei clienti del server).
- Il processo ricevente (server) utilizza, per ricevere messaggi, il metodo `Receive` a cui vengono passati come parametri, fra l'altro, un vettore di interi `vet` e un intero `n`.
  `n` rappresenta la dimensione del vettore `vet` (`1>=n>=N`). Gli elementi di `vet` sono `n` interi che rappresentano gli indici delle porte (indici di elementi del vettore **`PortVector`**) da cui si vuol ricevere. La receive è bloccante se su nessuna delle porte indicate esiste un mittente pronto a inviare un messaggio. In questo caso il processo ricevente verrà svegliato quando sarà possibile ricevere da una qualunque delle porte indicate. Se, viceversa, da una o da più di una delle porte è possibile ricevere, ne viene scelta una a piacere e viene ricevuto il messaggio da tale porta. Di nuovo, la receive deve restituire al processo ricevente l'informazione di tipo <T> contenuta nel messaggio ricevuto e l'identificazione del processo mittente, ma in questo caso, deve essere restituito anche l'indice della porta dell'array dalla quale il messaggio è stato ricevuto.

Parte quarta

Utilizzando esclusivamente il meccanismo di comunicazione sincronizzata implementato nella parte terza, implementare:

- Un processo server che costituisce una **`Mailbox`** a cui inviano informazioni di tipo integer 5 processi mittenti (`mit1`, …,`mit5`) e da cui riceve esclusivamente un processo ricevente `ric`.
- Il processo **`Mailbox`** fornisce un buffer (array circolare) di 3 elementi.
- I due servizi offerti dal server sono costituti dalla ricezione ed inserimento nel buffer (quando c'è spazio disponibile) dei valori inviati dai mittenti e l'estrazione di un valore dal buffer (quando disponibile) e suo invio al processo ricevente.
- La **`Mailbox`** serve i processi mittenti in base alla loro priorità (priorità di `mit1`> priorità di `mit2`, priorità di `mit2`> priorità di `mit3` e così via).
- Ogni processo mittente esegue un ciclo di 4 iterazioni e in ogni iterazione invia alla **`Mailbox`** un diverso intero.
- Il processo ricevente esegue un ciclo di 20 iterazioni per ricevere tutti i valori inviati. In ogni iterazione riceve uno dei valori inviati dai mittenti e stampa a video tale valore e il nome del processo mittente.