Reti e sistemi operativi
A.A. 2013/2014

24 Aprile 2014

Homework 2 (facoltativo)

- Consegna: entro **9 Maggio 2014 ore 24**
- Formato: **archivio (.zip oppure .tar.gz)** che contiene *una directory* al cui interno devono trovarsi: tutti i file sorgenti (.c e .h) ed un file .txt con le istruzioni su come compilare ed eseguire il software (specificare qui nome, cognome e numero di matricola).
- Si richiede: **codice funzionante e ben commentato**.

Esercizio 1

Implementare in C una versione semplificata dell'algoritmo del banchiere discusso a lezione, utilizzando le librerie pthread. Creare n threads che tentano periodicamente di allocare un certo numero di risorse, ovvero ogni thread dovrà ripetere *ciclicamente* la sequenza riportata sotto:

- Il thread richiede (chiamando funzione **int request(...)**) un certo numero di istanze per ogni risorsa disponibile. Il numero di istanze sarà casuale (usare ad esempio la funzione rand() dichiarata in stdlib.h), da 0 al massimo numero di istanze ancora necessarie (ad esempio il thread i tenterà di allocare per ogni risorsa j un numero casuale di risorse che può andare ogni volta da 0 a $need[i][j]$).
- La funzione **request()** dovrà verificare con l'algoritmo del banchiere che la richiesta può essere soddisfatta e che il sistema rimarrà in *safe-state* dopo l'eventuale allocazione. Se l'allocazione è possibile, sarà necessario aggiornare di conseguenza le varie strutture $available[]$, $need[][], \ldots$. **request()** dovrà ritornare 1 in caso di allocazione possibile, 0 altrimenti.
- Se l'allocazione è possibile (**request()** ha restituito 1), il thread che ha ottenuto le risorse ne simulerà l'utilizzo mettendosi in pausa (ad esempio, con la funzione usleep()) per un intervallo casuale (ad esempio, da 100 a 1000 msec). Al termine di tale intervallo, dovrà rilasciare tutte le risorse da esso allocate con la funzione **void release(...)**. La funzione **release()** dovrà aggiornare di conseguenza le varie strutture $available[]$, $need[][], \ldots$. Dopo aver chiamato la funzione **release()**, il thread si metterà in pausa per un intervallo casuale (ad esemio, da 100 a 1000 msec).

- Se l'allocazione non è possibile (**request()** ha restituito 0), il thread accetterà il fatto non facendo nulla, mettendosi in pausa per un intervallo casuale (ad esempio, da 100 a 1000 msec).
- Ripetere dal primo punto

Attenzione: le risorse condivise andranno protette con un **mutex lock**, in pthread sono oggetti di tipo **pthread_mutex_t**:

```
#include <pthread.h>
/* Tipo mutex lock in pthread, l'istanza dovrà essere
   visibile da tutti i thread che vogliono
   accedere alle risorse condivise */
pthread_mutex_t mutex;

/* Il mutex lock dovrà essere
inizializzato (una sola volta!) */
pthread_mutex_init(&mutex,NULL);

/* Per proteggere una regione critica,
ogni thread dovrà acquisire il mutex lock */

/* Acquisisce il mutex lock */
pthread_mutex_lock(&mutex);

/*** Sezione critica ***/

/* Rilascia il mutex lock */
pthread_mutex_unlock(&mutex);
```

Per semplicità si imponga un numero fisso di processi (ovvero di thread) e risorse a disposizione, ad esempio $n = 5$ thread e $m = 3$ tipologie di risorsa, ognuna delle quali con un numero fisso di istanze.

Se pensate che manchino delle specifiche al problema, o se avete dei dubbi, contattatemi via mail: pretto@dis.uniroma1.it