Esercitazione #7 -- Corso di Sistemi Operativi

Sincronizzazione in Java

Luca Gherardi e Patrizia Scandurra – a.a. 2012-13

Usare il meccanismo dei semafori in 1, un meccansimo a piacere (modificatore synchronized, o semafori, o lock con variabili condizione) in 2.

1. **Party.** Si progetti una applicazione Java che simuli una festa (la risorsa condivisa) con un numero di posti liberi limitato e un numero di invitati maggiore. Supporre che alcuni invitati attendono all'ingresso per un certo periodo di tempo (attesa con timeout), dopodiché lasciano perdere. L'applicazione dovrà essere organizzata nelle seguenti classi:

- Party: La classe Party simula la festa, quindi ha un *numero di posti disponibile* e un *semaforo* per l'accesso concorrente. La classe dispone inoltre dei metodi goIn() e goIn(millis) (passato il timeout millis, si risolve senza l'accesso) con cui gli invitati provano ad entrare alla festa, del metodo goOut() per uscire dalla festa liberando un posto, e del metodo goOutWithStamp() per uscire dalla festa con un timbro sul braccio senza liberare un posto e per poter poi rientrare con il metodo goInWithStamp() senza competere per l'accesso.
- Invited: La classe per i thread che accedono alla festa. Il task di un invitato è quello di provare ad entrare nella festa (acquisendo il diritto ad accedere), divertirsi un po' (simulato da uno sleep random), uscire (rilasciando il diritto di accesso ad altri) o uscire col timbro (senza rilasciare il diritto di accesso ad altri), riposarsi un po' (sleep random) e provare a ritornare dentro.
- LazyInvited Classe che estende la classe Invited per rappresentare un invitato "lazy" (pigro). Un invitato pigro ha lo stesso ciclo di vita di un invitato normale, con l'eccezione che dopo 5 secondi di fila per entrare alla festa si annoia ed esce dalla coda. In questo caso, il metodo goIn(millis) effettua un tryAcquire() che, passato il timeout, si risolve senza l'accesso lasciando libero però l'invitato di fare altro.
- MainPR. Classe contenente un metodo main() per la simulazione della festa. Ad esempio:

```
public static void main(String[] args) {
  //Definiamo il party di 20 persone concorrenti
   Party party = new Party(20);

   //Creiamo 40 invitati (con un esubero di 20)
   Invited inLista[] = new Invited[40];

  //metà saranno lazy, gli altri no
   for(int i=0;i<inLista.length;i++){
     Invited tmp;
     if (i%2==0)
         tmp = new Invited("NotLazy#"+i, party);
     else
         tmp = new LazyInvited("Lazy#"+i, party);
     inLista[i]=tmp;
   }

   //avviamo i 40 invitati (in un dato istante, al più 20 persone possono essere presenti alla festa)
   for(int i=0;i<inLista.length;i++)
      inLista[i].start();
 }
```

*[**Suggerimento**:* Per implementare il metodo goIn(millis) della classe Party, usare il metodo semaphore.tryAcquire(time, TimeUnit.SECONDS) sul semaforo. *]*

2. **Single Lane Bridge** Un ponte (vedi Figura 1.) lungo un fiume è talmente stretto da permettere una sola carreggiata di transito da usare in modo *bidirezionale* dalle autovetture provenienti da entrambe le direzioni (auto rosse da sinistra e auto blu da destra). Uno scontro (violazione della proprietà di *safety*) avviene se due autovetture (vedi Figura 1) provenienti da direzioni opposte (una rossa ed una blu) entrano nel ponte nello stesso tempo.

Figura 1

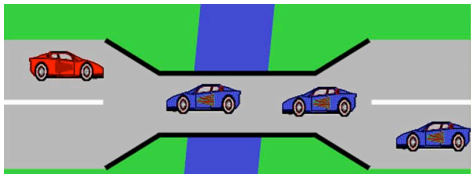

Figura 2

La Figura 2 suggerisce una soluzione al problema del transito attraverso una turnazione stretta tra macchine blu e rosse. Si fornisca tale soluzione in Java sfruttando il meccanismo di sincronizzazione dei semafori. A tale scopo, si definisca una classe `SafeBridge` che implementi l'interfaccia `Bridge` di seguito riportata contenente i metodi eseguiti dai thread (le auto rosse e blu) per l'ingresso (*enter*) e l'uscita (*exit*) dal ponte.

```
interface Bridge {
   abstract void redEnter() throws InterruptedException;
   abstract void redExit();
   abstract void blueEnter() throws InterruptedException;
   abstract void blueExit();
}
```

ATTENZIONE Proporre una soluzione che non soffre del problema della *starvation* delle auto, oltre che garantire la mutua esclusione. Se ad es. è presente una fila lunga di macchine rosse e le "rosse" hanno avuto l'opportunità di passare, assicurarsi che le eventuali macchine blu non aspettino un tempo *indefinitamente lungo* (devono prima passare tutte le "rosse") per poter passare.

*[**Suggerimento**:* Definire nella classe `SafeBridge` attributi per mantenere il conteggio delle auto blu e rosse in transito sul ponte ed altri contatori per raffinare il protocollo di accesso concorrente al ponte.*]*