Esercitazione #6 -- Corso di Sistemi Operativi

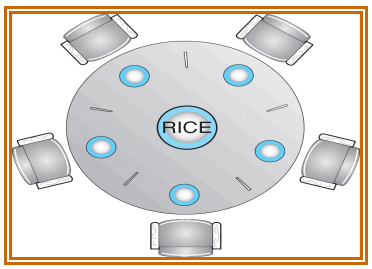
Sincronizzazione in Java

Semafori e barriere

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

1. **Il problema dei filosofi**. Si progetti un'applicazione Java per la soluzione del *problema dei cinque filosofi a cena* tramite l'impiego dei semafori, ovvero rappresentando ciascuna bacchetta con un semaforo: `Semaphore chopstick[] = new Semaphore[5];`

   dove tutti gli elementi di `chopstick` sono inizializzati a 1 (semafori binari). Il comportamento dell'i-esimo filosofo è mostrato in Figura 1:




```
while (true) {
    // prendi la bacchetta di sinistra
    chopStick[i].acquire();
    // prendi la bacchetta di destra
    chopStick[(i + 1) % 5].acquire();
    eating();
    // rendi la bachetta di sinistra
    chopStick[i].release();
    // rendi la bacchetta di destra
    chopStick[(i + 1) % 5].release();
    thinking();
}
```

Figura 1 -- Filoso "i"

   Questa soluzione garantisce che due filosofi adiacenti non mangino simultaneamente (sincronizzazione garantita) ma può generare deadlock quando tutti i filosofi hanno fame e prendono contemporaneamente la forchetta di sinistra.

2. **Il problema dei filosofi (senza deadlock)** Provare a modificare l'applicazione di cui al punto 1, implementando una *soluzione asimmetrica* per risolvere il problema del deadlock come segue. Un filosofo di posto *dispari* raccoglie per prima la bacchetta alla sua sinistra, e solo dopo quella alla sua destra, mentre un filosofo di posto *pari* raccoglie prima quella di destra e poi quella di sinistra.

3. **Somma casuale.** Si progetti una applicazione Java che usa il meccanismo della barriera per sincronizzare n thread che si comportano nel modo seguente. Ciascun thread dorme per un numero casuale di secondi, poi pensa ad un numero casuale ed infine aspetta che anche gli altri n-1 thread abbiano pensato il loro numero. Il programma deve calcolare la somma degli n numeri pensati dai thread.

4. **CokeMachine**. Si illustri, con il linguaggio Java, una soluzione al problema di prelevare lattine di coca-cola da una macchinetta e di rifornirla nel caso in cui rimanga vuota. Si utilizzi a tale scopo i *semafori* del package `java.util.concurrent` come meccanismo di sincronizzazione.
   Si assume che inizialmente la macchinetta è piena, e che un utente (a scelta: il primo a trovare la macchinetta vuota o l'utente che preleva l'ultima lattina) può segnalare al fornitore che la macchinetta è vuota.