Sistemi Operativi (modulo II / B)

7 giugno 2017

Esercizio 1

Si consideri una variante del problema dei filosofi in cui i filosofi raccolgono due qualsiasi tra le 5 bacchette. I filosofi raccolgono le bacchette invocando **due volte** di seguito il metodo `raccogli()` del monitor `Tavola`:

```
Monitor Tavola {
        n_bacchette = 5;

        void raccogli() {
                while (n_bacchette == 0)
                        wait();
                n_bacchette--;
        }
}
```

```
void deposita() {
        n_bacchette++;
        notifyAll();
}
```

1. Discutere la possibilità di stallo mostrando una possibile esecuzione problematica
   Se tutti i filosofi decidono di raccogliere le bacchette contemporaneamente ed eseguono con successo la prima invocazione di `raccogli()`, avremo che `n_bacchette` varrà 0 e la successiva invocazione sarà bloccante mandando tutti e 5 i filosofi in stallo: si forma una attesa circolare in cui ogni filosofo attende che un altro filosofo rilasci una bacchetta.

2. Realizzare un metodo `raccogliAtomico()` che raccoglie le bacchette solo se sono entrambe disponibili o, in alternativa, attende. I filosofi invocheranno una sola volta tale metodo. Spiegare perché questa soluzione previene lo stallo.

```
raccogliAtomico() {
        while (n_bacchette < 2)
                wait();
        n_bacchette = n_bacchette - 2;
}
```

   In questo modo non si può formare una attesa circolare perché si va a negare il “possesso e attesa”: nessun filosofo in attesa possiede bacchette che potrebbero essere usate da altri filosofi.

Esercizio 2

Il thread `main` deve attendere l’esecuzione di tutti i thread `task(i)`, con `i` = `0`, … , `N_TASK-1`. Realizzare la sincronizzazione aggiungendo opportuni semafori e operazioni P e V:

```
thread main {
  int N_TASK = 100;
  // dichiarare qui i semafori:
  semaphore sem[N_TASK] = {0, … , 0};

  < lancia i thread task(i) e fa il proprio lavoro >

  // Attende i task:
  for (i=0; i < N_TASK; i++) {
    // attende il task i
    P(sem[i]);
  }
}
```

```
thread task(i) {
  < Fa il proprio lavoro >

  // Notifica la terminazione:
  V(sem[i]);
}
```

Spiegare brevemente il funzionamento della soluzione proposta: E’ sufficiente definire un array di semafori inizializzati a 0 (rosso). Il thread main esegue le P su tutti i N_TASK semafori, mentre il task i-esimo esegue la corrispettiva V sbloccante sul “proprio” semaforo. Poiché i semafori sono rossi il thread main terminerà solo dopo che tutti i task hanno terminato. E’ importante osservare che se un task termina prima che il main abbia eseguito la P sul semaforo, il semaforo corrispondente diventerà verde (1) e la P non sarà più bloccante.