— COMPITO CON SOLUZIONE —

Sistemi Operativi (modulo II / B)

20 maggio 2014

Esercizio 1

Si consideri il seguente processo che stampa due volte il numero `i` passato come parametro:

```
process number(i) {

    print(i)


    print(i)

}
```

Vengono eseguiti 10 processi `number(i)` con `i` $= 0, \ldots, 9$. Sincronizzare i processi tramite semafori in modo che vengano stampati i numeri nell'ordine 0 1 2 ... 8 9 9 8... 2 1 0. Oltre ad aggiungere le P e le V indicare il valore di inizializzazione dei semafori e spiegare brevemente la soluzione proposta.

SOLUZIONE: Utilizziamo 2 array di semafori, `crescente[10]` per la stampa in ordine crescente e `calante[10]` per quella in ordine calante. I semafori sono tutti inizializzati a 0 (rosso) tranne `crescente[0]` e `calante[9]` che sono inizializzati a 1 (verde). Ogni processo attende sul propio semaforo e sblocca il processo `i+1` in caso di ordine crescente e `i-1` per l'ordine calante. I due if evitano overflow sugli array. Gli unici due semafori verdi corrispondono ai casi in cui il processo non deve attendere: il processo 0 quando stampa i primo 0 e il processo 9 quando stampa il secondo 9.

```
process number(i) {
    P(crescente[i])
    print(i)
    if (i < 9) V(crescente[i+1])

    P(calante[i])
    print(i)
    if (i > 0) V(calante[i-1])
}
```

Esercizio 2

Progettare un Monitor `Stack` che implementi uno stack di interi di dimensione finita `MAX`. Le due funzioni di accesso `push` e `pop` devono aggiungere e togliere un elemento dallo stack ma nel caso lo stack sia, rispettivamente, pieno o vuoto diventano bloccanti. Inoltre, si richiede l'assenza di interferenze tra i thread che utilizzano lo `Stack`. Scrivere la soluzione in pseudo-codice spiegando opportunamente.

SOLUZIONE: Utilizziamo `wait` e `notify` e due condition `piene` e `vuote` per i due casi di attesa: in `push`, quando `posizione == MAX` attendiamo che si liberi spazio (`vuote`); in `pop`, quando `posizione == 0` attendiamo che ci sia almeno un elemento (`piene`). La `notify()` su `vuote` viene eseguita quando si crea spazio mentre quella su `piene` quando si aggiunge un elemento. Il monitor previene interferenze grazie alla mutua esclusione implicita.

```
monitor Stack {
    condition piene,vuote;
    int stack[MAX];  // stack di dimensione MAX
    int posizione=0; // la posizione sullo stack

    push(int d) {
        while(posizione == MAX)
            vuote.wait()
        stack[posizione] = d;
        posizione = posizione + 1;
        piene.notify();
    }

    int pop() {
        while(posizione == 0)
            piene.wait()
        posizione = posizione - 1;
        d = stack[posizione];
        vuote.notify();
        return d
    }
}
```