— COMPITO CON SOLUZIONE —

Sistemi Operativi (modulo II / B)

11 Settembre 2012

Esercizio 1

Si consideri il generico thread P(i) che effettua una stampa sincronizzata tramite il monitor `io` come segue:

```
thread P(i) {
    io.stampa(i,"Saluti da " + i)
}

monitor io {
    int current=1;
    condition c;
    void stampa(int i, String s) {
        while (i != current)
            c.wait
        print s;
        current=current+1;
        c.notifyAll;
    }
}
```

Considerare un programma che manda in esecuzione, uno dopo l'altro, i seguenti thread:

1. P(1),P(2),P(3)
2. P(3),P(2),P(1)
3. P(3),P(4),P(1)
4. P(3),P(4),P(2)

Per ogni caso indicare l'output spiegando brevemente (risposte senza spiegazione non verranno valutate).

SOLUZIONE: La condizione 'while (i != current)' nel monitor controlla che l'id del thread corrisponda con quello corrente (inizialmente 1). In caso contrario il thread invoca una wait sulla condition c e attende (lasciando libera la sezione critica del monitor). Se l'id corrisponde il thread stampa i saluti, incrementa current e esce. L'effetto è che i saluti vengono stampati partendo da 1 in progressione. Se un id manca ci sarà uno stallo. In dettaglio:

1. P(1),P(2),P(3). Stampa 'Saluti da 1', 'Saluti da 2', 'Saluti da 3';
2. P(3),P(2),P(1). Stampa 'Saluti da 1', 'Saluti da 2', 'Saluti da 3';
3. P(3),P(4),P(1). Stampa 'Saluti da 1' e va in stallo perché manca il thread 2;
4. P(3),P(4),P(2). Va in stallo perché manca il thread 1.

Esercizio 2

Considerare la seguente soluzione `software` (insoddisfacente) al problema della sezione critica per 2 processi `i` e `j`:

```
while (turno!=i) {}          while (turno!=j) {}

< sezione critica >          < sezione critica >

turno = j;                   turno = i;
```

in cui la variabile condivisa `turno` è inizializzata a `i`. Mostrare una esecuzione con 2 processi in cui NON viene garantito il *progresso*, in cui, cioè, la sezione critica è libera ma viene proibito l'accesso a uno dei due processi.

SOLUZIONE: L'idea è di imporre un'alternanza nell'accesso alla sezione critica da parte dei due processi. Potrà quindi accedere i poi j poi di nuovo i e così via. Il problema insorge nel momento in cui uno dei due processi vuole accedere due volte consecutive alla sezione critica. Se ad esempio i accede alla sezione critica, alla sua uscita la variabile turno sarà impostata al valore j. A questo punto se i tenta nuovamente di accedere si bloccherà sul ciclo 'while (turno!=i) '. L'accesso è proibito anche se j sta facendo altro. Solo nel momento in cui j utlizzerà la sezione critica e la rilascerà anche i potrà finalmente accedere. Per questa ragione la soluzione proposta non è soddisfacente.