— COMPITO CON SOLUZIONE —

Sistemi Operativi (modulo II / B)

15 gennaio 2015

Esercizio 1

La funzione `transfer(i,j,c)` esegue un trasferimento bancario di `c` euro da `account`[`i`] a `account`[`j`], nel caso ci sia disponibilità, altrimenti ritorna `false`.

```
boolean transfer(int i, int j, int c);
    P(mutex)
    if (account[i] >= c) then {
        account[i] = account[i]-c;
        account[j] = account[j]+c;
       V(mutex)
        return(true);
    } else {
       V(mutex)
        return(false);
    }
```

1. Illustrare possibili problemi di interferenza causati dall'esecuzione simultanea di più bonifici;

   Problema 1: consideriamo due trasferimenti `transfer(0,1,100)` `simultanei con account[0]=110`. Se la condizione `account[0] >= 100` viene verificata dal secondo bonifico prima del decremento da parte del primo il conto andrà in negativo (-90 Euro).

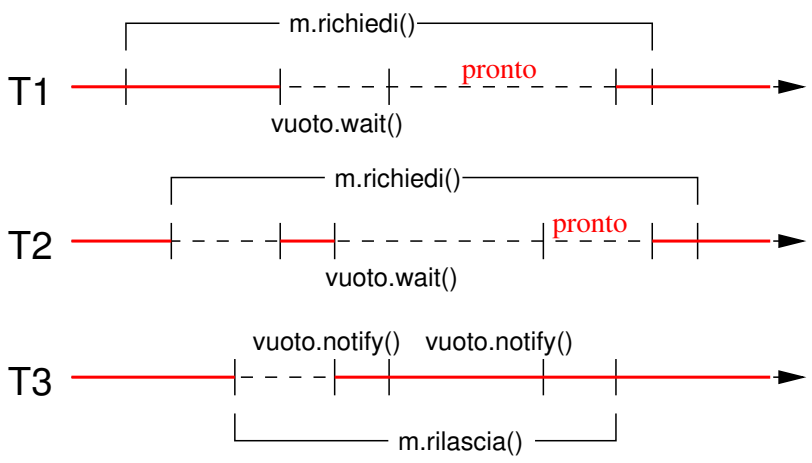
   Problema 2: consideriamo due trasferimenti `transfer(0,1,100)` `simultanei con account[0]=200`. Può succedere che `account[0] = account[0]-100;` venga eseguito allo stesso momento e che entrambi i thread leggano il valore 200 per poi decrementarlo in memoria e scrivere 100. Uno dei decrementi andrebbe perduto e il conto conterrebbe 100 euro invece che 0.

2. Eliminare le interferenze discusse al punto precedente tramite semafori aggiungendo opportune operazioni P e V a fianco del codice e indicando l'inizializzazione dei semafori. Discutere brevemente la soluzione proposta.

   Per proteggere il codice dalle interferenze è necessario racchiudere tutte le operazioni che possono interferire in una *sezione critica*. Utilizziamo quindi un semaforo mutex inizializzato a 1 e aggiungiamo una `P(mutex)` prima del `if (account[i] >= c) then`. La relativa `V(mutex)` va inserita prima del `return` in entrambi i rami del if-then-else. Le interferenze discusse precedentemente vengono evitate in quanto solo un thread alla volta potrà verificare la condizione `account[i] >= c` ed eseguire le operazioni sui conti. Solo dopo l'aggiornamento dei conti il mutex viene rilasciato e il thread successivo può eseguire il proprio trasferimento.

Esercizio 2

I thread $T1, T2, T3$, in esecuzione su tre processori, eseguono le procedure `richiedi()` e `rilascia()` di un monitor `m` con una variabile condition `vuoto`.
Indicare, nello schema sottostante, i thread in esecuzione tramite una linea continua; lasciare invece la linea tratteggiata nel caso il thread sia in attesa. Assumere che le code sulle condition e sul mutex siano FIFO.



Spiegare:

I thread procedono in parallelo (sulle 3 CPU) fino all'ingresso del monitor. $T1$ acquisisce il mutex, $T2$ e $T3$ attendono, nell'ordine. $T1$ si blocca sulla wait e $T2$ (il primo in coda) prende il mutex per poi bloccarsi sulla wait e lasciare il mutex a $T3$. Quest'ultimo fa le due notify sbloccando $T1$ e $T2$ (la condition è FIFO) e rimane in esecuzione. Quando rilascia il mutex viene eseguito $T1$ (il primo dei pronti) e, quando anche $T1$ esce dal monitor, viene eseguito $T2$. Fuori dal monitor i tre thread procedono in parallelo.