Copyright © 2007 – The McGraw-Hill Companies srl

Esercizio E2.2

Parte a)

Impostazione

Il monitor da realizzare, indichiamolo con l'identificatore di `scambiatore`, offre la sola funzione `scambia`. Possiamo supporre che contenga due variabili intere (`buffer1` e `buffer2`) destinate a contenere rispettivamente i due interi che il primo e il secondo processo di ogni coppia passano come parametro alla funzione `scambia` e che, quindi, devono essere restituiti, il primo valore al secondo processo che ha invocato `scambia` e il secondo valore al primo dei due processi. E' quindi fondamentale che ogni processo che invoca `scambia` possa determinare se è il primo di una coppia oppure il secondo. Per questo è sufficiente tenere aggiornata una varabile `boolean` (indichiamola con l'identificatore `primo`). In questo modo la funzione `scambia` può essere strutturata come un semplice `if else`. Se un processo si accorge di essere il primo, inserisce il proprio dato nel `buffer1`, commuta la variabile boolean e si sospende su una variabile `condition` in attesa che arrivi il secondo processo della coppia (variabile `attesa_secondo`). Altrimenti (se è il secondo) inserisce il proprio dato nel `buffer2` e, a questo punto il primo processo può essere risvegliato e i due processi, uno alla volta, possono terminare la funzione `scambia`, il primo restituendo il valore contenuto in `buffer2` e il secondo restituendo il valore contenuto in `buffer1`. Ovviamente la variabile boolean deve essere riportata al valore iniziale per predisporla per la prossima coppia.
Adottando la semantica *signal_and-urgent_wait*, poiché il primo processo è svegliato dal secondo, è il primo processo a terminare per primo la fuzione mentreil secondo (il segnalante) termina per ultimo.

Soluzione

McGraw-Hill
Tutti i diritti riservati

```
monitor scambiatore {
   int buffer1, buffer2;
   boolean primo=true; /*il valore iniziale di primo denota che che il prossimo processo a
                          invocare scambia è il primo di una coppia*/
   condition attesa_secondo;

   public int scambia(int x) {
      if(primo) {
         primo=false;
         buffer1=x;
         wait(attesa_secondo);
         primo=true;
         return buffer2;
         }
      else {
         buffer2=x;
         signal(attesa_secondo);
         return buffer1;
      }
   }
```

Questa soluzione, che struttura la funzione `scambia` in due rami alternativi, presuppone che chi esegue `scambia` possa essere o il primo processo di una coppia (colui che esegue il ramo `then` dell'`if`, o il secondo , che viceversa esegue il ramo `else`. In altri termini, fra l'istante in cui il primo processo di una coppia inizia ad eseguire il corpo della funzione e l'istante in cui il secondo termina l'esecuzione della

Copyright © 2007 – The McGraw-Hill Companies srl

stessa, nessun altro processo deve iniziare a sua volta l'esecuzione del corpo della funzione. Infatti, se ciò accadesse, un terzo processo, non essendo né il primo né il secondo di una coppia, produrrebbe un errore eseguendo la funzione, qualunque sia il ramo eseguito.
Avendo utilizzato la semantica *signal_and_urgent_wait*, la soluzione è corretta in quanto nessun processo terzo potrà mai inizare l'esecuzioine di `scambia` prima che sia terminata l'esecuzione della funzione da parte dei primi due processi. Infatti, un processo può iniziare l'esecuzione della funzione soltanto quando la mutua esclusione del monitor lo consente e cioè quando nessuno opera sul monitor. All'inizio il monitor è libero e quindi il primo processo che invoca `scambia` è abilitato ad iniziare la sua esecuzione occupando il monitor fino a quando non si sospende sulla condizione `attesa_secondo`. A questo punto il monitor viene liberato e un secondo processo può iniziare l'esecuzione della funzione. Altri processi che abbiano invocato la funzione, o che la invocano da ora in poi, restano sospesi sulla *entry_queue* del monitor fino a quando la mutua esclusione non viene rilasciata. Il secondo processo, eseguendo il ramo `else` della funzione, sveglia il primo e gli cede il controllo senza rilascare la mutua esclusione (*passaggio del testimone*). Quindi il primo termina e, in base alla semantica utilizzata, questo cede di nuovo il testimone al secondo senza rilasciare la mutua esclusione. Quindi anche il secondo termina riportando il monitor nello stato iniziale prima di rilasciare definitivamente la mutua esclusione. Solo a questo punto, quindi, un nuovo processo può iniziare a sua volta l'esecuzione della funzione, comportandosi come primo processo di una nuova coppia.

Parte b)

Impostazione

Utilizzando la semantica *signal_and_continue*, la garanzia che un processo terzo non inizi l'esecuzione di `scambia` prima che i primi due abbiano terminato la loro esecuzione della funzione, viene meno. Infatti, supponiamo che avvenga la seguente sequenza di eventi indicando con P e Q rispettivamente il primo e il secondo processo di una coppia: se durante l'esecuzione della funzione da parte di Q (il secondo processo di una coppia), uno o più processi (ad esempio R e W) invocano la funzione, questi ultimi si sopendono nella *entry_queue* del monitor per mutua esclusione. Quando il processo in esecuzione (Q) sveglia il primo processo (P) che è sospeso sulla condizione `attesa_secondo`, in base alla semantica *signal_and_continue*, il processo svegliato viene accodato nella *entry_queue* del monitor e il processo segnalante continua terminando l'esecuzione di `scambia` e rilasciando la mutua esclusione. Dopo questo evento viene abilitato a entrare nel monitor il primo dei processi sospesi nella *entry_queue* che, in questo caso è il processo R e non il processo P, che essendo il primo della coppia quando riprende la sua esecuzione si aspetta di trovare in `buffer2` il valore da restituire. In realtà il processo R che lo precede nell'esecuzione della funzione, trova lo stato del monitor tale per cui si comporta come se fosse il secondo processo di una coppia (variabile `primo==false`) e quindi cambia il valore di `buffer2` e restituisce il valore di `buffer1` già restituito anche al processo Q.
Per evitare questo tipo di errori dovuti a corse critiche nel rientrare in mutua esclusione da parte del primo processo di una coppia, dobbiamo inserire nel monitor una ulteriore variabile boolean (`secondo`) che consenta di discriminare, non solo fra chi è primo o il secondo di una coppia, ma anche chi non è né primo né secondo, ed inserire una ulteriore variabile condition (`fine_primo`) su cui bloccare questi processi terzi. Le due variabili boolean `primo` e `secondo` vengono utilizzate con questi significati:

a) se `primo` ha valore `true`: lo stato interno del monitor è quello corrispondente allo stato iniziale in cui si attende che il primo processo di una coppia invochi `scambia`;
b) se `primo` ha valore `false` ma `secondo` ha valore `true`: lo stato interno del monitor è quello in cui il primo si è sospeso su `attesa_secondo` e si attende che il secondo processo di una coppia invochi `scambia`;
c) se `primo` ha valore `false` e anche `secondo` ha valore `false`: lo stato interno del monitor è quello intermedio in cui il secondo ha terminato e si attende che il primo processo riprenda l'esecuzione e termini `scambia`.

Copyright © 2007 – The McGraw-Hill Companies srl

Soluzione

```
monitor scambiatore {
   int buffer1, buffer2;
   boolean primo=true;
   boolean secondo=true;
   condition attesa_secondo;
   condition fine_primo;

   public int scambia(int x) {
      while(!primo && !secondo) /* il processo che inizia l'esecuzione della funzione
                                   non è né il primo né il secondo e quindi si deve
                                   sospendere in attesa che il primo termini
                                   l'esecuzione riportando il monitor in stato iniziale*/
         wait(fine_primo);
      if(primo) { /*il processo che inizia l'esecuzione della funzione è il primo*/
         primo=false;
         buffer1=x;
         wait(attesa_secondo);
         primo=true; /* il monitor viene riportato in stato iniziale*/
         secondo=true;
         signalAll(fine_primo); /*tutti gli eventuali processi terzi, sospesi su
                                  fine_primo, vengono riattivati e, inseriti nella
                                  entry_queue del monitor, possono rientrarci uno alla
                                  volta e iniziare l'esecuzione di una nuova coppia*/

         return buffer2;
         }
      else {
         buffer2=x;
         secondo=false;
         signal(attesa_secondo);
         return buffer1;
      }
   }
```

McGraw-Hill
Tutti i diritti riservati