REALIZZAZIONE DI POLITICHE DI GESTIONE DELLE RISORSE. SEMAFORI PRIVATI

Condizione di sincronizzazione.

• Qualora si voglia realizzare una *determinata politica di gestione* delle risorse,la decisione se ad un dato processo sia consentito proseguire la esecuzione dipende dal verificarsi di una condizione detta *condizione di sincronizzazione.*

• La condizione è espressa in termini di variabili che rappresentano lo *stato della risorsa* e di *variabili locali ai singoli processi.*

• Più processi possono essere bloccati durante l'accesso ad una risorsa condivisa, ciascuno in attesa che la propria condizione di sincronizzazione sia verificata.

• In seguito alla modifica dello stato della risorsa da parte di un processo, le condizioni di sincronizzazione di alcuni processi bloccati possono essere *contemporaneamente* verificate.

• Problema di quale processo mettere in esecuzione (accesso alla risorsa mutuamente esclusivo). Definizione di una *politica per il risveglio dei processi bloccati*.

• Nei casi precedenti la condizione di sincronizzazione era particolarmente semplificata (vedi mutua esclusione) e la scelta di quale processo riattivare veniva effettuata tramite l'algoritmo implementato nella *signal.*

• Normalmente questo algoritmo dovendo essere sufficientemente generale ed il più possibile efficiente coincide con quello *FIFO.*

Esempio 1.
• Su un buffer da N celle di memoria più produttori possono depositare messaggi di *dimensione diversa*.

• Politica di gestione: tra più produttori ha *priorità di accesso* quello che fornisce il messaggio di *dimensione maggiore*.

• La politica di gestione comporta che finché un produttore, il cui messaggio ha dimensioni *maggiori* dello spazio disponibile nel buffer, rimane sospeso *nessun altro produttore* può depositare un messaggio anche se la sua dimensione potrebbe essere contenuta nello spazio libero del buffer.

•**Condizione di sincronizzazione** : il deposito può avvenire se c'è sufficiente spazio per memorizzare il messaggio e non ci sono produttori in attesa.

• Il prelievo di un messaggio da parte di un consumatore prevede la riattivazione tra i produttori sospesi, di quello il cui messaggio ha la *dimensione maggiore*, semprechè esista sufficiente spazio nel buffer.

• Se lo spazio disponibile non è sufficiente *nessun produttore viene riattivato.*

Esempio 2

• Un insieme di processi utilizza un insieme di risorse comuni R1,R2,..Rn.

• Ogni processo può utilizzare una qualunque delle risorse. La *condizione di sincronizzazione* si riduce quindi a valutare se esiste una risorsa libera.

•A ciascun processo è assegnata *una priorità*.

•In fase di riattivazione dei processi sospesi viene scelto quello cui corrisponde *la massima priorità*

Esempio 3

Con riferimento al problema dei *lettori scrittori*, si intende realizzare una politica di gestione che eviti condizioni di *attesa indefinita* per entrambe le classi di processi.

SEMAFORO PRIVATO

• Un semaforo si dice **privato** per un processo quando solo tale processo può eseguire sul semaforo la primitiva **wait.**

• La primitiva **signal** sul semaforo può essere invece eseguita **anche da altri processi.**

• Il semaforo privato viene inizializzato con il valore **zero**

Procedure di acquisizione e rilascio di una risorsa

```
var   mutex: semaphore initial (1);
      priv: array (1..num-proc) of semaphore initial (0);
procedure acquisizione (i:integer,…);
      ….
      begin
          wait (mutex);
             if< condizione di sincronizzazione> then begin
                                <allocazione della risorsa>
                                signal (priv(i))
                                                        end
             else < indicazione di sospensione del processo>
          signal (mutex)
          wait (priv(i))
      end;
```

```
procedure rilascio (..)
        var i:integer;
        begin
             wait (mutex);
             <rilascio della risorsa>;
             if <esiste almeno un processo sospeso per il quale la
                condizione di sincronizzazione è soddisfatta>
            then begin
                        <scelta del processo p(i) da attivare>;
                        <allocazione della risorsa a p(i)>;
                        <indicazione che p(i) non è più sospeso>;
                         signal (priv(i));
                    end
            signal (mutex);
end
```

Proprietà della soluzione

a) La sospensione del processo, nel caso in cui la condizione di sincronizzazione non sia soddisfatta, *non può avvenire entro la sezione critica* in quanto ciò impedirebbe ad un processo che rilascia la risorsa di accedere a sua volta alla sezione critica e di riattivare il processo sospeso.
La sospensione avviene *al di fuori della sezione critica*.

b) La specifica del particolare algoritmo di assegnazione della risorsa non è opportuno che sia realizzata *nella primitiva signal*. Nella soluzione proposta è possibile programmare esplicitamente tale algoritmo scegliendo in base ad esso il processo da attivare ed *eseguendo la signal sul suo semaforo privato.*

Lo schema presentato può, in certi casi, presentare degli inconvenienti.

a) l'operazione wait sul semaforo privato viene *sempre eseguita* anche quando il processo richiedente non deve essere bloccato.

b) Il codice relativo all'assegnazione della risorsa viene *duplicato* nelle procedure acquisizione e rilascio

Si può definire uno schema che non ha questi inconvenienti

```
var     mutex: semaphore initial (1);
        priv: array (1..num-proc) of semaphore initial (0);
 procedure acquisizione (i:integer,…);
        ….
        begin
            wait (mutex);
            if not< condizione di sincronizzazione> then
            begin
                <Indicazione della sospensione del processo>;
                 signal(mutex);
                 wait (priv(i));
                <indicazione processo non più
                 sospeso>;
             end
            <allocazione della risorsa>;
             signal (mutex);
          end;
```

```
procedure rilascio (….);
        var i:integer;
        begin
             wait (mutex);
             <rilascio della risorsa>;
             if <esiste almeno un processo sospeso per il quale la
                condizione di sincronizzazione è soddisfatta>
            then begin
                        <scelta del processo p(i) da attivare>;
                         signal (priv(i));
                    end
           else     signal (mutex);
end;
```

• A differenza della soluzione precedente, tuttavia, in questo caso risulta più complesso realizzare la riattivazione di più processi per i quali risulti vera contemporaneamente la condizione di sincronizzazione.

• Lo schema prevede infatti che il processo che rilascia la risorsa attivi *al più un processo sospeso*, il quale dovrà a sua volta provvedere alla riattivazione di eventuali altri processi.

Soluzione all'esempio 1

```
var     richiesta: array (1..num-proc)of integer initial (0);
        sospesi: integer initial (0);
        vuote:integer initial(m);
        mutex:semaphore initial (1);
        priv: array( 1..num-proc) of semaphore initial (0);
        Buffer : …/*dichiarazione del buffer come insieme di
celle*/
```

```
procedure acquisizione (i:integer,m:integer);
        begin
            wait (mutex);
                if sospesi=0 and vuote>=m
                then begin
                vuote:= vuote-m;
                <assegnazione al processo di m celle di buffer>;
                 signal (priv(i));
                 end;
                else begin
                sospesi:= sospesi+1; richiesta(i):= m;
                end
                signal (mutex);
                wait (priv(i));
        end;
```

```
procedure rilascio (m:integer);/*m specifica il numero di celle
rilasciate*/  var k:1..num-proc;
              begin
              wait (mutex); vuote:= vuote+m;
              while sospesi<>0 do begin<individuazione del
              processo Pk con la massima richiesta>
              if richiesta (k) <=vuote
              then begin
                        vuote:=vuote-richiesta(k);
                        <assegnazione a Pk delle celle richieste>
                         richiesta (k)=0; sospesi:=sospesi-1;
                         signal (priv(k));
                    end
              else exit/*termine istruzione while*/
                                   end;
              signal(mutex)
              end;
```

Esempio 2

PS(i)	variabile logica che assume il valore vero se il processo Pi è sospeso; il valore falso diversamente.
risorse(j)	variabile logica che assume il valore falso se la risorsa j-esima è occupata; il valore vero diversamente.
disponibile	il numero delle risorse non occupate
sospesi	il numero dei processi sospesi
mutex	un semaforo di mutua esclusione
priv(i)	il semaforo privato del processo Pi

```
var     risorse: array (1..num-ris)of boolean initial (true);
        PS: array (1..num-proc)of boolean initial (false);
        sospesi: 0..num-proc  initial (0);
        disponibile: 0..num-ris  initial(num..ris)
        mutex: semaphore initial (1);
        priv: array( 1..num-proc) of semaphore initial (0);
```

```
Procedure Richiesta (var x: 1..num-ris; i:1..num-proc);
/* x indica il numero di risorsa allocata al processo richiedente;i
il nome del processo*/ var k: 0..num-ris;
        begin
            wait (mutex);
                        if disponibile=0 then begin
                        sospesi:=sospesi+1;PS(i):=true;
                         signal (mutex);
                        wait (priv(i));
                        PS(i):=false; sospesi:=sospesi-1;
                                                end
            k:=0;
            repeat k:=k+1 until risorse(k);
            x:=k; disponibile:=disponibile+1;
            risorse(k):=false
            signal(mutex);
        end;
```

```
procedure rilascio (x:1..num-ris);/*x indica il numero di risorsa
rilasciate*/
 var j:1..num-proc;
        begin
             wait (mutex);
             disponibile:=disponibile+1;
             risorse(x):=true
             if sospesi>0
             then begin
                <seleziona il processo Pj a massima priorità tra
                 quelli sospesi utilizzando il vettore PS>
                 signal (priv(j));
                    end
             else signal(mutex));
        end;
```

Considerazioni sulle soluzioni presentate

- Ogni processo che nella fase di acquisizione della risorsa trova la condizione di sincronizzazione non soddisfatta, **deve lasciare traccia in modo esplicito** della sua sospensione entro la sezione critica.
- Il processo che libera la risorsa deve infatti eseguire la primitiva signal(priv(i)) **solo se** esistono processi sospesi. In tutte le soluzioni e' stata introdotta **un'apposita variabile** per indicare il numero dei processi sospesi.
- La fase di assegnazione di una risorsa ad un processo è **separata** dalla fase di uso della risorsa stessa.
- Occorre quindi **lasciare traccia** in modo esplicito entro la sezione critica della assegnazione e quindi della **non disponibilità della risorsa**. Nel primo caso viene decrementata la variabile vuote; nel secondo viene modificata la variabile R