Processi – parte V

Processi – parte V

- Sincronizzazione dei processi mediante monitor:
  - Sintassi
  - Funzionamento
  - Implementazione

Monitor

- Il costrutto monitor permette di definire un tipo di dato astratto e di sincronizzare processi che devono accedere in modo concorrente ad una stessa struttura dati di tipo monitor. Nel seguito è riportata la sintassi per definire un tipo di dato monitor e per dichiarare variabili di tipo monitor.

Monitor

```
type =monitor;
procedure entry (...);
  begin...end;
procedure entry (...);
  begin...end;
procedure (...);
  begin...end;
begin  end;
var :
```

Monitor

- Il costrutto monitor permette di definire un nuovo tipo di dato astratto, a cui saranno associate le operazioni (definite come procedure entry nella definizione) che costituiscono l'unica interfaccia attraverso cui si può agire sullo stato delle variabili di quel tipo.
- Per esempio si potrebbe definire un tipo buffer dotato di operazioni inserimento ed estrazione. L'implementazione del buffer (che potrà per esempio essere realizzato come vettore circolare o come lista linkata) non sarà visibile all'esterno.

Monitor

- In aggiunta il monitor permette di realizzare la sincronizzazione tra processi che condividono strutture dati di questo tipo in due modi:
  - assicurando che ogni operazione eseguibile su una data struttura dati x di tipo monitor avverrà in mutua esclusione rispetto a qualsiasi altra operazione sulla stessa struttura
  - permettendo ai processi che eseguono operazioni su una struttura dati di tipo monitor di sospendersi eseguendo una istruzione cond.wait (o wait(cond)) se non è verificata una data condizione cond.

Monitor

- Inoltre, tali processi possono essere risvegliati grazie all'esecuzione di una istruzione cond.signal (o signal(cond)) da parte di un altro processo che abbia richiesto l'esecuzione di una delle operazioni associate allo stesso monitor. Esistono varianti dell'operazione wait in cui è previsto che possa essere passato un parametro p il quale viene interpretato come priorità e viene usato per decidere in che ordine risvegliare i processi in attesa di una data condizione al momento dell'esecuzione di una cond.signal.

Monitor: funzionamento

- Il monitor è un costrutto che permette di risolvere problemi di sincronizzazione fra processi. Rispetto ai semafori è un costrutto di più alto livello, ma la potenza espressiva è equivalente. Infatti è possibile realizzare il monitor utilizzando i semafori e viceversa.
- Il primo tipo di sincronizzazione messo a disposizione dal monitor è la mutua esclusione nell'esecuzione di procedure definite all'interno del monitor stesso:
  - Se due processi richiedono contemporaneamente di eseguire una procedura su una stessa struttura dati di tipo monitor, le richieste verranno soddisfatte una alla volta, in mutua esclusione. Questo vale sia nel caso in cui la procedura richiesta sia la stessa sia nel caso i due processi vogliano eseguire procedure diverse.

Monitor: funzionamento

- Per esempio se la struttura dati in questione si chiama ProdCons, è di tipo buffer, ed è dotata delle operazioni inserimento ed estrazione, l'esecuzione di ProdCons.inserimento da parte di un processo produttore e di ProdCons.estrazione da parte di un processo consumatore avverranno in mutua esclusione.
- Il secondo tipo di sincronizzazione è quello realizzabile attraverso le operazioni cond.wait e cond.signal, dove cond è una variabile di tipo condition locale al monitor. La cond.wait permette ad un processo di sospendersi in attesa che si verifichi la condizione cond, mentre la cond.signal permette di segnalare che la condizione cond è vera, risvegliando eventuali processi in attesa.

Monitor: funzionamento

- Possiamo immaginare una struttura dati di tipo monitor come un oggetto capace di ospitare un solo processo alla volta per l'esecuzione di una delle sue procedure, e dotato di alcune code di attesa:
  - una per tenere traccia dei processi che hanno fatto richiesta di eseguire una procedura ma devono attendere perché qualche altro processo è già all'interno del monitor
  - una per ogni variabile cond di tipo condition, che serve a tenere traccia di tutti i processi che sono stati sospesi a causa dell'esecuzione di una operazione cond.wait mentre si trovavano dentro al monitor
  - infine vi è una coda che tiene traccia dei processi che hanno eseguito una cond.signal mentre esisteva qualche processo nella coda d'attesa della condizione cond

Monitor: esempio

- Consideriamo il seguente esempio: si tratta di un buffer utilizzabile per realizzare un sistema produttore-consumatore. È una variante rispetto all'esempio classico in quanto la procedura inserzione comprende un parametro aggiuntivo n tramite il quale è possibile richiedere di inserire nel buffer n copie di un messaggio

Monitor: esempio

```
type buffer = monitor;
var non_vuoto, non_pieno:condition;
var count:integer;
progedure entry inserzione(n, msg)
begin
    for i:=1 to n
        begin
            if count = N then non_pieno.wait;
            INSERIMENTO DI msg NEL BUFFER;
            count := count+1;
            if count=1 then non_vuoto.signal
        end;
end;
progedure entry estrazione(var msg)
begin
    if count = 0 then non_vuoto.wait;
    COPIA IN msg IL PRIMO ELEMENTO DEL BUFFER;
    count := count-1;
    if count= N-1 then non_pieno.signal
end;
begin;
    count:=0;
end;
var ProdCons:buffer;
```

Monitor: come classe

- Come la classe:
  - È un tipo di dato astratto che incapsula sia i dati che le funzioni che li manipolano.
  - Le funzioni non visibili dall'esterno della classe sono precedute dalla parola chiave private.
  - Implicitamente fornisce la mutua esclusione nell'esecuzione delle sue procedure.
- Al suo interno possono essere dichiarate variabili di tipo condition utili per la sincronizzazione.

Condition wait e signal

- Operazioni su variabili di tipo condition:
  - Condition wait è sempre bloccante.
  - Condition signal sveglia il primo processo bloccato su una coda associata alla condition.
  - Un segnale su una coda di condizione vuota non ha alcun effetto.

Produttore - Consumatore

```
const int MAX = 100;
monitor BoundedBuffer {
private:
  Message buffer [MAX];
  int p, c, count;
  condition nonempty, nonfull;
public:
  void send(Message m);
  Message receive(void);
  void init(void);
};
```

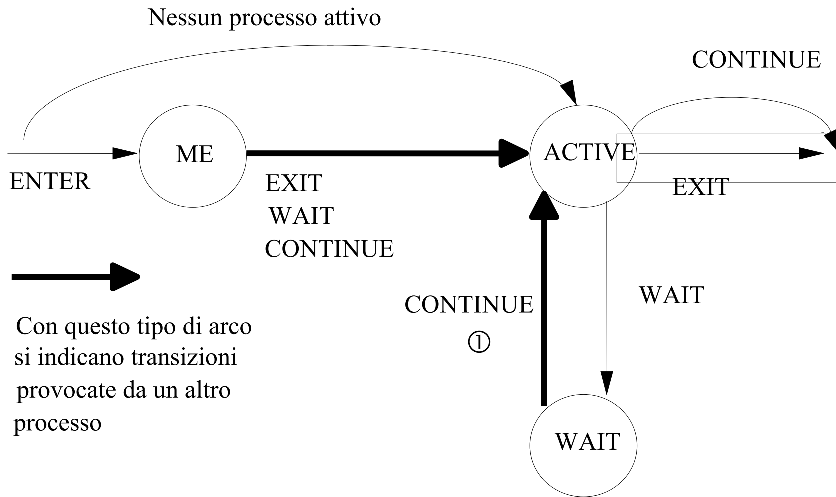
Produttore - Consumatore

```
void send(Message m){
  if (count == MAX) wait(nonfull);
  buffer[p] = m;
  p = (p + 1) % MAX;
  count++;
  signal(nonempty);
}

Message receive (void){
  if (count == 0) wait(nonempty);
  m =buffer[c];
  c = (c + 1) % MAX;
  count--;
  signal(nonfull);
}
```

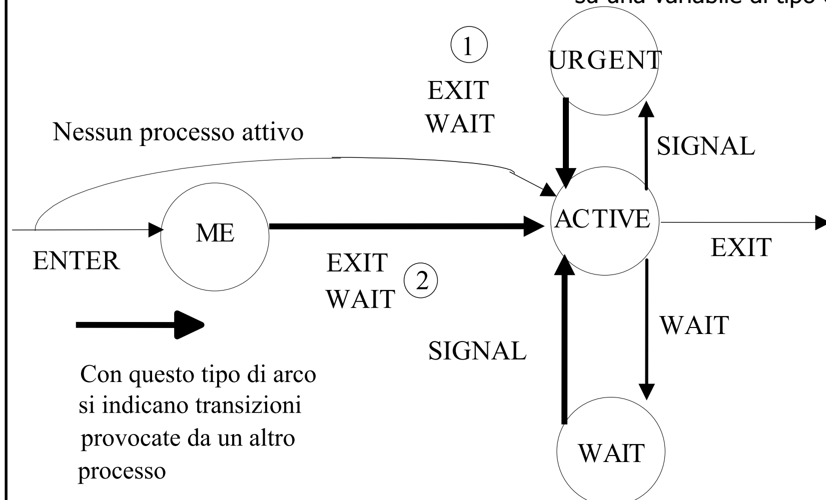
```
void init(void)
{
  p=c=count=0
}
```

Monitor di Brinch-Hansen



Monitor di Hoare

Lo stato URGENT rappresenta
uno stato in cui un processo finisce
quando ne sblocca un altro in attesa
su una variabile di tipo condition



Monitor di Hoare

- p.wait sospende il processo che l'ha invocata su p rilasciando il controllo esclusivo del monitor
- p.signal:
  - il processo che l'ha invocata continua l'esecuzione se non c'è alcun processo bloccato su p
  - altrimenti il processo viene temporaneamente sospeso mentre uno dei processi bloccati su p viene riattivato. Il processo che si sospende in seguito all'operazione p.signal riprende l'esecuzione solamente quando non c'è nessun altro processo in esecuzione nel monitor. Inoltre, tale processo ha una priorità maggiore su ogni processo che tenta di iniziare l'esecuzione di procedura del monitor.

Implementazione monitor di Hoare

```
enter : WAIT(me)

exit :  if (urgent.cnt < 0) SIGNAL(urg)
         else SIGNAL(ME)

cond_wait :
   if (urgent.cnt < 0) {WAIT (cond); SIGNAL (urg)}
   else {WAIT (cond); SIGNAL(me)}

cond_signal :
   if (cond.cnt < 0) {WAIT (urg); SIGNAL (cond)}
```

Semaforo binario: con monitor

```
monitor SemaforoBinario {
private:
  char busy;
  condition risorsa_libera;
public:
  void wait(void);
  void signal(void);
  void init(void);
};
void init(void){
  busy = FALSE;
}
```

Semaforo binario

```
void wait(void){
  if (busy) wait(risorsa_libera);
  busy = TRUE;
}
void signal(void){
  busy = FALSE;
  signal(risorsa_libera);
}
```

Uso di un semaforo binario

```
main(void){
  SemaforoBinario me;
  me.init();
  me.wait();
  // Regione Critica;
  me.signal();
}
```

Semaforo generale

```
monitor Semaforo {
private:
  int ar, r, s;
  condition cond;
public:
  void wait(void);
  void signal(void);
  void init(void);
};
void init(int I){
  ar = r = 0; s = I;
}
```

Semaforo generale

Invariante semaforico
CNT = I + s - ar

```
void wait(void){
  ar++;
  if (s < ar) wait(cond);
  r++;
}
void signal(void){
  s++;
  if (s <= ar)signal(cond); // ar > r
}
```

Uso di un semaforo generale

```
main(void){
 SemaforoGenerale me;
me.init(1);
me.wait();
  // Regione Critica;
me.signal();
}
```

R & W con precedenza ai Readers

```
monitor RW_precedenza_ai_Readers {
private:
  int nr;
  char busy;
  condition readers, writers;
public:
  void start_read(void);
  void start_write(void);
  void end_read(void);
  void end_write(void);
  void init(void);
};
```

R & W con precedenza ai Readers

```
void start_read(void) {
 if (busy) wait(readers);
 nr++;
 signal(readers);
}
```

```
void init (void){
  busy = FALSE;
  nr = 0;
}
```

```
void end_read(void) {
  nr--;
  if (nr == 0)signal(writers);
}
```

R & W con precedenza ai Readers

```
void start_write(void) {
 if (nr > 0 || busy) wait(writers);
 busy = TRUE;
}
```

```
void end_write(void) {
  busy = FALSE;
  if (queue(readers)signal(readers);
  else               signal(writers);
}
```

R & W con precedenza ai Readers

```
void start_write(void) {
 if (nr > 0 || busy) wait(writers);
 busy = TRUE;
}
```

```
void end_write(void) {
  busy = FALSE;
  if (queue(writers)signal(writers);
  else               signal(readers);
}
```

Variabili di condizione con priorità

- Molte soluzioni risultano più ovvie e semplici se si utilizza per l’accodamento un ulteriore parametro associato alla procedura WAIT.
- Questo parametro serve per stabilire la posizione nella coda nella quale deve essere inserito l'elemento.

Monitor shortest job next

```
monitor ShortestJobNext{
private:
  char busy;
  condition cond;
public:
  void acquire(int time);
  void release(void);
  void init(void);
};
void init(void){
  busy = FALSE;
}
```

Monitor shortest job next

```
void acquire(int time){
  if (busy) wait(cond, time);
  busy = TRUE;
}
void release(void){
  busy = FALSE;
}
```

Monitor Alarm Clock

```
monitor AlarmClock {
private:
  int now;
  condition wakeup;
public:
  void tick(void);
  void wakeme(int time);
  void init(void);
};
void init(void){
  now = 0;
}
```

Monitor Alarm Clock

```
void wakeme (int time){
  int alarmtime;
  alarmtime = now + time;
   while (now < alarmtime)
     wait(wakeup, alarmtime);
   signal(wakeup);
}
void tick ()
  now++;
  signal(wakeup);
}
```

Monitor Alarm Clock seconda versione

```
monitor AlarmClock {
private:
  int now;
  Queue alarm;
  condition wakeup;
public:
  void tick(void);
  void wakeme(int time);
  void init(void);
};
void init(void){ now = 0;}
```

Monitor Alarm Clock

```
void wakeme (int time){
  int alarmtime;
  enqueue(alarm, now+time);
  wait(wakeup, alarmtime);
  if (now > first(alarm)) {
    dequeue(alarm);
    signal(wakeup);
  }
}
```

```
void tick ()
  now++;
  if (now >= first(alarm)){
     dequeue(alarm);
     signal(wakeup);
  }
}
```

Monitor per schedulazione disco CSCAN

- Tempi di accesso ad un settore del disco:
  - Tempo di seek
  - Tempo di rotazione
  - Tempo di trasferimento in memoria
- Nella strategia CSCAN la testina si muove in una direzione dalla traccia 0 a quella finale.
- Serve le richieste accodate per le tracce successive a quella attuale
- Ritorna alla traccia 0 quando ha servito la richiesta accodata per la traccia più lontana da 0.

Monitor per schedulazione disco CSCAN

```
monitor CircularScan {
private:
  char busy;
  int headpos, count;
  condition up;
public:
  void release(void);
  void acquire(int destination);
  void init(void);
};
void init(void){headpos = 0; count = 0; busy = FALSE}
```

Monitor per schedulazione disco CSCAN

```
void acquire(int dest) {
  if (busy) {
    if (headpos <= dest)
      wait (up, dest + count);
    else
      wait (up, dest + count + MAX);
  }
  busy = TRUE;
  if (dest < headpos) count += MAX;
  headpos = dest;
}
```

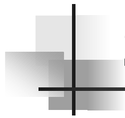
```
void
release(void){
  busy = FALSE;
  signal(up);
}
```

Esercizio

- Si risolva con il costrutto monitor il problema del barbiere assonnato che si può schematizzare nel modo seguente:
  - La bottega di un barbiere comprende una stanza di attesa con N sedie e l'angolo di taglio in cui è sistemata l'apposita poltrona. Si consideri l'attività del barbiere così schematizzata:
    - Se non ci sono clienti il barbiere va a riposare
    - Se un cliente entra nella bottega e trova tutte le sedie occupate allora rinuncia al taglio di capelli
    - Se il barbiere è impegnato con un altro cliente allora attende il proprio turno su una delle sedie disponibili
    - Se il barbiere sta riposando allora il cliente lo sveglia

Soluzione

```
type BarberShop  monitor;
var Chair:  0…N+1;
        Seats:  integer;
        Bell:  condition;
        Turn:  condition;
function entry CutRequest:  boolean
var Sitting:  boolean;
begin
        Sitting:=  Chair <= Seats;
        if Sitting then
                begin
                        Chair :=  Chair + 1;
                        if  (Chair >1)    then Turn.wait;
                end
        CutRequest:=  Sitting;
end
```



Soluzione

```
procedure entry ExitShop
begin
    if  (Chair ==1)    then Bell.signal;
end
procedure entry CutService
begin
    Chair :=  Chair  -1;
    if  (Chair == 0) then Bell.wait;
end
procedure entry Continue
begin
    if  (Chair >2)  then Turn.signal;
end
begin
    Chair  := 0;
end
```