MONITOR

Idea: garantire l'integrita` di una struttura dati condivisa permettendo l'accesso soltanto attraverso le procedure del monitor ("tipi di dato astratto").

Definizione: un monitor e` un tipo di dato che e` costituito da:

- una struttura dati condivisa: contiene le informazioni utilizzate dai processsi concorrenti e le variabili di stato per garantire la sincronizzazione dei processi che usano il monitor;

- una operazione di inizializzazione della struttura;

- un insieme di procedure di monitor dette entry che sono l'unico modo per accedere dall'esterno alla struttura condivisa; esse utilizzano le variabili di stato ed eventuali altre variabili e procedure locali. Le entry sono eseguite in modo mutuamente esclusivo;

- variabili di condizione, condition variables, per garantire la sincronizzazione.

Dichiarazione: (Concurrent Pascal)

```
type monitor_name = monitor
     var "strutture dati del monitor";
procedure entry proc_name1("parametri");
     body_1;
     ...
procedure entry proc_nameN("parametri");
     body_N;
begin
     "inizializzazione";
end;
```

Istanziazione:

```
var M: monitor_name; "viene eseguita l'inizializzazione"
```

Utilizzo delle procedure entry:

```
M.proc_namei("valori dei parametri");
```

Sincronizzazione:

I processi possono sincronizzarsi (quindi venire messi in coda in attesa di un evento) attraverso una condition variable dichiarata:
`var x: condition;`

all'interno delle procedure di monitor posso usare:

- `wait(x);` **sospende il processo che aveva eseguito la procedura entry e rilascia la mutua esclusione;**
- `signal(x);` **riattiva un processo sospeso su** `x`**.**

ATTENZIONE:
`wait` **e** `signal` **sono diverse dalle corrispondenti operazioni su semafori:**

- `signal(x)` non viene memorizzata se nessun processo e' in coda sulla condition variable `x` (quindi "va persa");
- `signal(s)` su semaforo incrementa il "contatore" `cnt[s]` di 1 (quindi il semaforo puo' diventare "verde" per eventuali processi che arriveranno in futuro);

Quando un processo esegue `wait(x)`**, esso viene messo in una coda associata ad** `x` **e rilascia la mutua esclusione.**

Vi e` anche la mutua esclusione sulle procedure entry, quindi una ulteriore coda di ingresso al monitor.

In generale le code di accesso al monitor saranno "FIFO", ma potrebbe non essere sempre vero.

In certe implementazioni del monitor e` presente anche una funzione** `queue(x);` **restituisce** `TRUE` **se ci sono processi in coda sulla condition variable** `x`**, altrimenti** `FALSE`.**

Monitor "mutua esclusione sulla risorsa"

```
program mut_ex;
type resource= monitor
  var busy: boolean;
  nonbusy: condition;
  "altre variabili tipiche della risorsa"
procedure entry acquire( )
  begin
     if busy then wait(nonbusy);
     busy:=true;
     "uso della risorsa"
end;
procedure entry release ( )
  begin
     busy:=false;
     signal(nonbusy);
end;
begin “inizial.”
     busy:=false; nonbusy:=empty_queue;
end;
var R1:resource;

process user(i)
  begin
     while "..." do
        r1.acquire();
        R1.release()
     endwhile
  end;

main()
begin
     spawn(user,10);
end.
```

Produttore e consumatore con buffer limitato

```
type bounded_buffer = monitor
 var buffer: array[0..n-1] of item;
     in,out: 0..n-1;
     size: 0..n;
     nonempty,nonfull: condition;
procedure entry put (x: item);
begin
     if size=n then wait(nonfull);
     buffer[in]:=x;
     in:=(in+1) mod n;
     size:=size+1;
     signal(nonempty);
end;
procedure entry get (var x:item);
begin
     if size=0 then wait(nonempty);
     x:= buffer[out];
     out:=(out+1) mod n;
     size:=size-1;
     signal(nonfull);
end;
begin "inizializzazione"
     size:=0; in:=0; out:=0;
end "della dich. Monitor"
```

Nota:
in questo esempio dopo la signal non ci sono altre istruzioni. Questo pero' potrebbe non essere sempre vero, per altri monitor!

```
program producer_consumer;
type bounded_buffer= monitor " vedi lucido precedente"
var buf: bounded_buffer;
       num:integer;

process producer;
begin
     while true do
          "produce  x"
          buf.put(x);
     endwhile
end;

process consumer;
begin
     while true do
         buf.get(x);
         "consuma x"
     endwhile
end;

begin
     num:=2;
     fork L1;
     producer;
     goto L2;
L1:  consumer;
L2:  join num;
end.
```

Implementazione di un monitor (Hoare)

L'accesso al monitor e la sincronizzazione al suo interno sono regolate da semafori.

E' necessario un primo semaforo per regolare l'accesso al monitor in mutua esclusione, sia esso `mutex`, inizializzato a 1.

La coda associata a `mutex` e' di tipo FIFO e si accede effettivamente al monitor eseguendo

`P(mutex);`

(usiamo `P` e `V` per non confonderci con le `wait` e `signal` dei monitor)

Per ogni condition variable occorre implementare una coppia di `wait/signal`. Esse richiedono:

- **un semaforo `condsem`, inizializzato a 0, che corrisponde alla condizione su cui il processo intende sospendersi;**
- **una variabile intera `condcount`, inizializzata a 0, che conta quanti processi sono in attesa sulla coda della condizione.**

Implementazione di `wait` e `signal` su una condizione:

```
wait ("condition")
begin
     condcount:=condcount+1;
     V(mutex);
     P(condsem);
     condcount:=condcount-1;
end;

signal("condition")
begin
     if condcount>0 then
          V(condsem)
     else
          V(mutex);
end;
```

Nota:

Questa `signal` presuppone che il processo che la esegue intenda anche uscire dal monitor.
Infatti ne risveglia un altro, sospeso sulla condition variable o all'ingresso, su `mutex`.

Questa implementazione e' detta monitor "di tipo monitor".

In molti casi (forse tutti quelli che esamineremo!) le `signal` sono effettivamente le ultime operazioni delle procedure di monitor.

Potrebbero pero' esistere problemi in cui dopo una `signal` il processo deve eseguire altre operazioni, quindi l'implementazione sopra vista non puo' essere sempre valida.

Esistono altre implementazioni del monitor che risolvono questo problema, sono pero' piu' complesse del monitor di tipo monitor.

Ad es. il monitor "di tipo mediatore" richiede una coda ulteriore per i processi sospesi dopo una `signal` ma che devono ancora eseguire operazioni prima di uscire dal monitor.

I monitor di tipo mediatore avranno quindi bisogno di un semaforo ed un contatore ad esso associato per regolare la fase di "uscita".

Problema dei lettori e scrittori

- Un monitor conferisce permessi di accesso al data base condiviso
- Lettori e scrittori chiedono il permesso di accedere e poi accedono direttamente
- occorre definire quattro procedure entry cioe` i protocolli di ingresso ed uscita rispettivamente per lettori e scrittori:
  startread, endread,
  startwrite, endwrite

Uso del monitor da parte dei processi lettori e scrittori:

```
process reader1;
var M: monitor ...; "vedi avanti"
    DB: data_base;
begin
    M.startread;
    DB.read(...);
    M.endread;
end;

process writer1;
var M: monitor ...; "vedi avanti"
    DB: data_base;
begin
    M.startwrite;
    DB.write (...);
    M.endwrite;
end;
```

Lettori e scrittori: monitor "senza priorita`"

Le variabili:
- **busy** indica se esiste uno scrittore operante sul database;
- **readercount** il numero dei lettori operanti sul database;

Le code di attesa:
- **users:** quando il database e` occupato da uno scrittore, contiene tutti i lettori e gli scrittori;
- **writers:** contiene gli scrittori in attesa quando il database e` occupato da un lettore

```
Type WritersExcludeOthers = monitor
var readercount: integer;
    busy: boolean;
    users, writers: condition;

proc entry startread()
begin
    if busy then wait (users);
    readercount:=readercount+1;
    signal (users)
end; "in questo caso possono esserci in attesa solo
      lettori"

proc entry endread()
begin
    readercount:=readercount-1;
    if readercount=0 then
      if queue(writers) then signal(writers)
                        else signal(users)
end; "puo' esserci uno scrittore in users"

proc entry startwrite()
begin
    if readercount>0 or busy then wait(users);
    "quando lo scrittore e' svegliato possono esserci
      lettori sulla risorsa"
    if readercount>0 then wait(writers);
    busy:=true
end; "se la risorsa e' occupata da uno scrittore si
      attende in users"

proc entry endwrite()
begin
    busy:=false;
    if queue(writers) then signal(writers)
                      else signal(users)
end; "si forza precedenza agli scrittori"

begin "inizializzazione"
  readercount:=0;
  busy:=false
end
```

Note

- Si usa la primitiva **`queue`**

- E` indispensabile che i processi lettori e scrittori rispettino i protocolli di ingresso ed uscita. Se non lo facessero, potrebbe succedere che.... (provare per esercizio)

- Gli scrittori potrebbero attendere “per sempre” se.... (costruire un esempio in cui cio’ accade). Questo algoritmo quindi puo` presentare *starvation*

- nella **`startwrite`** quando un lettore riesce ad accedere alla risorsa, uno scrittore che era bloccato nella coda **`users`** puo` passare nella coda **`writers`**

- L’algoritmo e` nondeterministico e non assegna priorita` eccetto che in un caso: la **`endwrite`** privilegia il risveglio degli scrittori. In questo caso dovendo esplicitare su quale condition si fa la **`signal`** si finisce “comunque” per violare il nondeterminismo.

- La **`signal`** in **`startread`** serve ad aumentare il parallelismo; potrebbe risvegliare un lettore o uno scrittore

- nella **`endread`** se non ci sono lettori in attesa viene risvegliato uno scrittore; puo` trovarsi in una delle due code.

Lettori e scrittori con precedenza ai lettori

- Introduciamo due distinte code di attesa:
  readers per i lettori
  writers per gli scrittori
- Al termine di una scrittura si risveglia con priorita` un lettore

```
type readers_priority = monitor
var readercount: integer;
    busy: boolean;
    readers, writers: condition;
procedure entry startread();
begin
    if busy then wait(readers);
    readercount:= readercount+1;
    signal(readers);
end;
procedure entry endread();
begin
    readercount:=readercount-1;
    if readercount=0 then signal(writers);
end;
procedure entry startwrite();
begin
    if readercount>0 or busy then wait(writers);
    busy:=true;
end;
procedure entry endwrite();
begin
    busy:=false;
    if queue(readers) then signal(readers)
                      else signal(writers)
end; "se c'e' almeno un lettore in attesa, viene
     svegliato"

begin "inizial."
    Readercount:=0; busy:=false
end
```

Nota
La soluzione e` piu` semplice della precedente!!!
I Monitor si prestano particolarmente a realizzare algoritmi con priorita` e scelte di scheduling deterministiche.

Lettori e scrittori con precedenza ai lettori e senza attesa infinita

Modifichiamo la procedure entry **startread** in modo da controllare ancheil tipo di richieste in coda.
Il resto e` invariato.

```
type readers_priority = monitor
var  readercount: integer;
     busy: boolean;
     readers, writers: condition;

procedure entry startread();
begin
     if busy or queue(writers) then wait(readers);
     readercount:= readercount+1;
     signal(readers);
end; "un lettore attende se vi sono scrittori gia` in
      attesa"

procedure entry endread();
begin
     readercount:=readercount-1;
     if readercount=0 then signal(writers);
end;
procedure entry startwrite();
begin
     if readercount>0 or busy then wait(writers);
     busy:=true;
end;
procedure entry endwrite();
begin
     busy:=false;
     if queue(readers) then signal(readers)
                       else signal(writers)
end; "se c'e' almeno un lettore in attesa, viene
      svegliato"
begin "inizial."
     Readercount:=0; busy:=false
end
```

Monitor con funzione di sveglia

E` utilizzato da processi che vogliono sospendersi e quindi essere risvegliati dopo un certo periodo di tempo.
Strutture:
coda con priorita` **wakeup**:
la priorita` e` data dal tempo a cui risvegliare il processo, specificato da **alarmtime** (il secondo parametro della **wait**!)
procedure entry **wakeme**:
chiamata dai processi che vogliono sospendersi;
procedure entry **tick**:
e' chiamata dal clock del sistema (non dai processi); un processo risvegliato viene rimesso in attesa se il tempo di risveglio non e` ancora arrivato.

```
Type alarm_clock= monitor
var now, alarmtime: integer;
    wakeup: priority_condition;

procedure entry wakeme(var time: integer);
begin
    alarmtime:=now + time;
    while now < alarmtime do wait(wakeup,alarmtime);
end;

procedure entry tick();
begin
    now:=now+1;
    signal(wakeup)
end;

begin "inizial."
    now:=0;
end
```

E` possibile evitare il risveglio prima del termine usando una coda ordinata dei tempi di risveglio, **alarm,** ed apposite funzioni per accedervi:

```
Type alarm_clock= monitor

var now: integer;
    alarmtime: ordered_queue of integer;
    wakeup: priority_condition;

procedure entry wakeme(var time: integer);
begin
    enqueue(alarm, now + time);
    wait(wakeup,now + time);
    if now > first(alarm) then
      begin dequeue(alarm); signal(wakeup) end
end;

procedure entry tick();
begin
    now:=now+1;
    if now >= first(alarm) then
      begin dequeue(alarm); signal(wakeup) end
end;
begin "inizial."
    now:=0; alarm := (); "coda vuota"
end
```