Sincro da ripetere

mercoledì 4 dicembre 2024 14:37

Problema dei Produttori consumatori

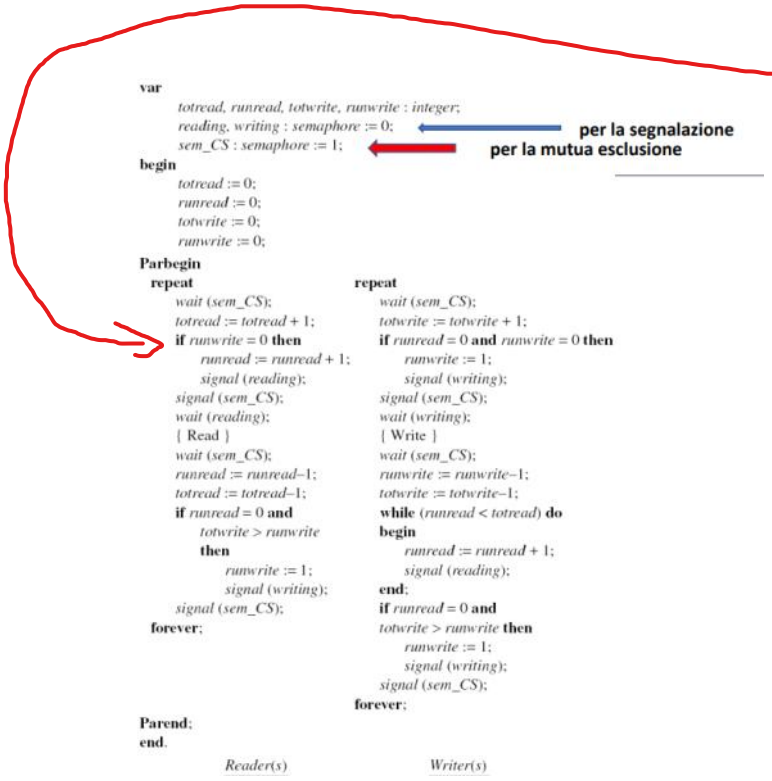
Produttori Consumatori con buffer limitato

```
type    item = . . .;
var
        full  : Semaphore := 0; { Initializations }
        empty  : Semaphore := 1;
        buffer  : array [0] of item;
begin
Parbegin
   repeat                                  repeat
        wait (empty);                           wait (full);
        buffer [0] := . . .;                    x := buffer [0];
            { i.e., produce }                       { i.e., consume }
        signal (full);                          signal (empty);
        { Remainder of the cycle }              { Remainder of the cycle }
   forever;                                forever;
Parend;
end.
            Producer                                Consumer
```

Produttori consumatori ad n buffer

```
const           n = . . .;
type            item = . . .;
var
                buffer  :  array [0..n – 1] of item;
                full  :  Semaphore := 0; { Initializations }
                empty  :  Semaphore := n;
                prod_ptr, cons_ptr  :  integer;
begin
                prod_ptr := 0;
                cons_ptr := 0;
Parbegin
   repeat                                       repeat
        wait (empty);                               wait (full);
        buffer [prod_ptr] := . . .;                 x := buffer [cons_ptr];
            { i.e. produce }                            { i.e. consume }
        prod_ptr := prod_ptr + 1 mod n;             cons_ptr := cons_ptr + 1 mod n;
        signal (full);                              signal (empty);
        { Remainder of the cycle }                  { Remainder of the cycle }
   forever;                                     forever;
Parend;
end.
            Producer                                    Consumer
```

Lettori-Scrittori:




```
var
    totread, runread, totwrite, runwrite : integer;
    reading, writing : semaphore := 0;
    sem_CS : semaphore := 1;
begin
    totread := 0;
    runread := 0;
    totwrite := 0;
    runwrite := 0;
Parbegin
 repeat
    wait (sem_CS);
    totread := totread + 1;
    if runwrite = 0 then
        runread := runread + 1;
        signal (reading);
    signal (sem_CS);
    wait (reading);
    { Read }
    wait (sem_CS);
    runread := runread–1;
    totread := totread–1;
    if runread = 0 and
        totwrite > runwrite
        then
            runwrite := 1;
            signal (writing);
    signal (sem_CS);
 forever;

                          repeat
                              wait (sem_CS);
                              totwrite := totwrite + 1;
                              if runread = 0 and runwrite = 0 then
                                  runwrite := 1;
                                  signal (writing);
                              signal (sem_CS);
                              wait (writing);
                              { Write }
                              wait (sem_CS);
                              runwrite := runwrite–1;
                              totwrite := totwrite–1;
                              while (runread < totread) do
                              begin
                                  runread := runread + 1;
                                  signal (reading);
                              end;
                              if runread = 0 and
                              totwrite > runwrite then
                                  runwrite := 1;
                                  signal (writing);
                              signal (sem_CS);
                          forever;
Parend;
end.
          Reader(s)                 Writer(s)
```

Se fosse prio scrittori sarebbe così nei lettori

```
if (runwrite = 0 and totwrite = 0) then begin
      runread := runread + 1;
      signal(reading);
   end;
```

Barbone addormentato

#Barb *SEMAFORI*

```
program barbiere_addormentato;
const sedie N;
var
        coda : integer;
        mutex : semaforo (:=1);
        cliente : semaforo (:=0);
        barbiere : semaforo (:=0):
procedure barbiere:
begin
        repeat
                wait(cliente);
                wait(mutex);
                coda--;
                signal(barbiere);
                signal(mutex);
                /*Taglia Capelli*/

        forever;
end;
```

```
procedure cliente:
begin
      repeat
            wait(mutex);
            if(coda<sedie){
                  coda++;
                  signal(mutex);
                  wait(barbiere);
                  /*Taglio Capelli*/
            }else{
                  /*Vai via*/
                  signal(mutex);
      forever
end;
```

ALGORITMO DI DEKKER

Algoritmo 6.3 Algoritmo di Dekker

```
var   turn : 1..2;
      c1, c2 : 0..1;
begin
      c1 := 1;
      c2 := 1;
      turn := 1;
Parbegin
   repeat
      c1 := 0;
      while c2 = 0 do
         if turn = 2 then
         begin
            c1 := 1;
            while turn = 2
               do { niente };
            c1 := 0;
         end;
      { Sezione critica }
      turn := 2;
      c1 := 1;
      { Resto del ciclo }
   forever;
Parend;
end.
```

Processo P_1

```
repeat
   c2 := 0;
   while c1 = 0 do
      if turn = 1 then
      begin
         c2 := 1;
         while turn = 1
            do { niente };
         c2 := 0;
      end;
   { Sezione critica }
   turn := 1;
   c2 := 1;
   { Resto del ciclo }
forever;
```

Processo P_2

ALGORITMO DI PETERSON

Algoritmo 6.4 Algoritmo di Peterson

```
var     flag : array [0..1] of boolean;
        turn : 0..1;
begin
        flag[0] : = false;
        flag[1] : = false;
Parbegin
   repeat                                   repeat
      flag[0] : = true;                        flag[1] : = true;
      turn : = 1;                              turn : = 0;
      while flag[1] and turn = 1               while flag[0] and turn = 0
         do { niente };                           do { niente };
      { Sezione critica }                      { Sezione critica }
      flag[0] = false;                         flag[1] = false;
      { Resto del ciclo }                      { Resto del ciclo }
   forever;                                 forever;
Parend;
end.
```

Processo P_0 *Processo* P_1

ALGORITMO DEL PANETTIERE / FORNAIO (BAKERY)

Algoritmo 6.6 Algoritmo del panettiere (Bakery) (Lamport [1974])

```
const n = ...;
var   choosing : array [0 .. n − 1] of boolean;
      number : array [0 .. n − 1] of integer;
begin
      for j : = 0 to n − 1 do
          choosing[j] : = false;
          number[j] : = 0;
Parbegin
   process Pi :

      repeat
          choosing[i] : = true;
          number[i] : = max (number[0],..,number[n − 1])+1;
          choosing[i] := false;
          for j := 0 to n − 1 do
          begin
              while choosing[j] do { nothing };
              while number[j] ≠ 0 and (number[j], j) < (number[i],i)
                  do { nothing };
          end;
          { Sezione critica }
          number[i] := 0;
          { Resto del ciclo }
       forever;
   processo Pj : ...
Parend;
end.
```