Esercizio di Sincronizzazione Tra Processi: Locale con Corridoi:

Testo:

Un **locale** possiede due accessi o **corridoi**, per consentire l'ingresso e l'uscita degli utenti. La protezione civile ha previsto un numero massimo di persone presenti nel locale (CAP).
Ogni accesso può, in un certo istante, permettere il passaggio o in ingresso o in uscita ad al massimo un certo numero di utenti (MAX).
In un dato istante, i gruppi che stanno passando attraverso un corridoio sono tutti nello stesso verso. Inoltre, se il corridoio e' pieno, si ritarda ogni accesso a richieste ulteriori.

Gli utenti possono anche presentarsi in gruppi (di consistenza numerica inferiore a MAX). Si noti che il gruppo entra tutto o aspetta: non e' possibile quindi un passaggio frazionato di un gruppo.

Si descriva la politica di sincronizzazione per i corridoi ed il locale, usando almeno uno dei seguenti costrutti (facoltativamente più di uno):
* **monitor, regione critica condizionale**.

La politica di sincronizzazione e la conseguente soluzione devono essere adeguatamente commentata. Inoltre si descriva la politica scelta discutendo adeguatamente riguardo:
- alle possibilità di starvation e
- alla possibilità di lasciare sottoutilizzate le risorse.

1° Soluzione con l'uso del costrutto Monitor:

Si considera un unico monitor per la gestione della sala.
Il monitor unico consente una gestione integrata dei due corridoi.
- due batterie di code di accesso ai corridoi:
  codain [corridoio, gruppo], codaout[corridoio, gruppo];
- una coda di sospensione per raggiunta capacita' massima della sala: attsala;
- variabili di stato opportune;

La soluzione prevede di massimizzare l'uso della risorsa sala, corridoi, e di favorire quindi l'uscita dalla sala, rispetto all'ingresso.

```
program SalaConDueCorridoi;

const MAX = ... ;
      CAPAC = ...;
type  dir  = ( in, out );
      corridoio = (1, 2);
      gruppo = 1 .. MAX;

type utente (c: corridoio ;  n: gruppo) = process ;
begin
sala. INcorrAccesso (n, c);
< transita >
sala. INcorrRilascio (n, c);
< in sala >
sala. OUTcorrAccesso (n,c);
< transita >
sala .OUTcorrRilascio  (n, c);
end;

type GestoreSala = monitor ;

var  direz : array [corridoio] of dir ;
     nutenti : array [corridoio] of integer;
     cap   : 0.. CAP;
     attsala : condition;
contatt : integer;
     codain, codaout: array [corridoio, gruppo] of condition;
```

```
procedure entry INcorrAccesso (c: corridoio; n: gruppo);
begin
     while  (cap + n) > CAP do  begin
        contatt + := 1;
        attsala. wait;
        contatt - := 1;
                        end;
     cap + := n;

     while ((direz [c] <> in)  and (nutenti [c] <> 0 )       or
          ((direz [c] = in)  and ((nutenti[c] +  n) > MAX)) or
          ((direz [c] = in)  and ((codaout [c,n]. queue)       do
   {sospensione anche in caso di utenti sospesi in attesa di uscire attraverso lo stesso corridoio }
                                codain [c, n]. wait;
     nutenti [c] + :=   n;
     direz [c] := in ;

end;

 procedure segnalaout (c: corridoio);
var i : integer;
begin
    for i = MAX downto 1 do
    while (codaout [c, i].queue  and
              (nutenti [c] + i <= MAX) do   codaout [c, i]. signal;
 end;

 procedure segnalain (c: corridoio);
var i : integer;
begin
    for i = MAX downto 1 do
    while (codain [c, i].queue  and
              (nutenti [c] + i <= MAX) and not codaout[c,i].queue  do
           codain [c, i]. signal;
 end;

procedure entry INcorrRilascio (c: corridoio; n: gruppo);
begin
     nutenti [c] - := n;
if nutenti [c] = 0 then segnalaout (c) else segnalain (c)
end;
```

```
procedure entry OUTcorrAccesso (c: corridoio; n: gruppo);
var i : integer;
begin
    cap - := n;
 < segnalazione sull'altro corridoio se e' il caso >
for i := 1 to contatt do attsala.signal;
    { tutti i processi in coda sono segnalati: l'ordine relativo       tra i processi e' preservato }

    while ((direz [c] <> out)  and (nutenti [c] <> 0 )       or
        ((direz [c] =  out)  and ((nutenti[c] +  n) > MAX))  do
                    codaout [c, i]. wait
nutenti [c] + :=   n;
    direz [c] := out ;
end;

procedure entry OUTcorrRilascio (c: corridoio; n: gruppo);
begin
    nutenti [c] - := n;
segnalaout (c);
segnalain (c);
end;


var  sala : GestoreSala;
    utente1 ( 2, n1  ) : utente; ...
    utente2 ( 1, n2  ) : utente, ...

begin end.
```

2° Soluzione con l'uso del costrutto Monitor:

Si noti che, nella precedente soluzione, la sala è impegnata anche da utenti che non sono autorizzati ad entrare (a causa della direzione corrente del corridoio) e che impegnano la capacità del locale. Una soluzione diversa può verificare le condizioni di accesso in un unico passo

```
program SalaConDueCorridoiUnicoPasso;
const MAX = ... ;
      CAPAC = ...;
type  dir  = ( in, out );
      corridoio = (1, 2);
      gruppo = 1 .. MAX;

type utente (c: corridoio ;  n: gruppo) = process ;
 ...{come prima soluzione }
end;

type GestoreSala = monitor ;
var  direz : array [corridoio] of dir ;       nutenti : array [corridoio] of integer;
     cap   : 0.. CAP;
     codain, codaout: array [corridoio, gruppo] of condition;

procedure entry INcorrAccesso (c: corridoio; n: gruppo);
begin
     while ((direz [c] <> in)  and (nutenti [c] <> 0 )       or
          ((direz [c] = in)  and
            ((nutenti[c] +  n) > MAX) or
              codaout [c,n]. queue    or
              (cap + n >   CAP )        )                do
     { si noti la unica condizione di sospensione e di riattivazione }
                                codain [c, n]. wait;
     cap + := n;
     nutenti [c] + :=   n;
     direz [c] := in ;

end;

procedure segnalaout (c : corridoio);
var i : integer;
begin
    for i = MAX downto 1 do
    while (codaout [c, i].queue    and
             nutenti [c] + i <= MAX)
                               do   codaout [c, i]. signal;
 end; { notare le condizioni verificate prima della riattivazione }
```

```
procedure segnalain (c : corridoio);
var i : integer;
begin
   for i = MAX downto 1 do
   while (codain [c, i].queue    and
            (nutenti[c] + i <= MAX) and
               (cap + i <= CAP) and not codaout[c,i].queue)
                        do   codain [c, i]. signal;
 end;

procedure entry INcorrRilascio (c: corridoio; n: gruppo);
begin
    nutenti [c] - := n;
if nutenti [c] = 0 then segnalaout (c)
else segnalain (c);
end;

procedure entry OUTcorrAccesso (c: corridoio; n: gruppo);
begin
    cap - := n;
    segnalain (altro (c));
    while ((direz [c] <> out)  and (nutenti [c] <> 0 )       or
         ((direz [c] =  out)  and ((nutenti[c] +  n) > MAX))  do
                        codaout [c, n]. wait
nutenti [c] + :=   n;
    direz [c] := out ;
end;

procedure entry OUTcorrRilascio (c: corridoio; n: gruppo);
begin
    nutenti [c] - := n; segnalaout(c);
segnalain(c);
end;
var  sala : gestoreSala;
    utente1 ( 2, n1  ) : utente; ...  utente2 ( 1, n2  ) : utente, ...
begin end.
```

Soluzione con l'uso delle regioni critiche condizionali:

Introduciamo una versione che consente di registrare le richieste e di tenere conto di richieste gia' ottenute (per la priorita' in uscita)

```
const MAX = ... ;
      CAPAC = ...;

type  dir  = ( in, out );
      corridoio = (1, 2);
      gruppo = 1 .. MAX;

type utente (c: corridoio ;  n: gruppo) = process ;
begin  sala. INcorrAccesso (n, c); < transita >
       sala. INcorrRilascio (n, c);
         < in sala >
       sala. OUTcorrAccesso (n,c); < transita >
       sala .OUTcorrRilascio  (n, c);
end;

var sala: shared record
          req   : array [corridoio, dir] of integer;
          direz : array [corridoio] of dir ;
          nutenti : array [corridoio] of integer;
          cap   : 0.. CAP;
         end record;


procedure entry INcorrAccesso (c: corridoio; n: gruppo);
begin
    region sala do
      req [c, in] + := 1;
    end region;
    region sala
    when   ((direz [c] <> in)  and (nutenti [c] = 0 )       or
           ((direz [c] = in)  and (nutenti[c] +  n <= MAX) and
            (cap + n <= CAP) and (req[c, out] = 0) )  do
      req [c, in] - := 1;
      cap + := n;
      nutenti [c] + :=   n;
      direz [c] := in ;
    end region;
end;

procedure entry INcorrRilascio (c: corridoio; n: gruppo);
begin
```

```
  region sala do
     nutenti [c] - := n;   end region;
end;


procedure entry OUTcorrRilascio (c: corridoio; n: gruppo);
begin
  region sala do
     nutenti [c] - := n;
  end region;
end;


procedure entry OUTcorrAccesso (c: corridoio; n: gruppo);
begin
    region sala do
      req [c, out] + := 1;
              end region;
    region sala
    when   ((direz [c] <> out)  and (nutenti [c] = 0 )       or
          ((direz [c] =  out)  and (nutenti[c] +  n <= MAX) do
        cap - := n;
        req [c, out] - := 1;
        nutenti [c] + :=   n;
        direz [c] := out ;
    end region;
end;

procedure init
begin
region sala do
  contatt := 0; cap := 0;
  nutenti [1] := 0; nutenti [2] := 0;
  direz [1] := in; direz [2] := in;
end region;
end;

end.
```

Soluzione con l'uso di ADA

In ADA, possiamo scrivere una soluzione del problema considerando per le quattro funzionalita' entry separate. Si e' aggiunta una entry per la prenotazione di un ingresso dall'esterno (Request).
Il rilascio del corridoio e' qui lo stesso sia in ingresso dal locale sia in uscita.

I rilasci possono essere parametrici, mentre gli accessi dovrebbero essere gestiti uno per uno.
Si puo' ricorrere alle famiglie di entry: il cliente indica la selezione all'interno della famiglia; il servitore deve specificare separatamente le entry della famiglia.

```
package ApplicazioneSalaConCorridoi is

const MAX = ... ;
      CAPAC = ...;
type  dir  = ( in, out );
      corridoio = (1, 2);
      gruppo = 1 .. MAX;

task type utente is
end;

task body utente is
  g : gruppo;
  c : corridoio;
begin
 sala . Request (c, in);
 sala . INcorrAccesso (g) (c);  < passaggio in>
 sala . corrRilascio (g, c);  < accesso alla sala >
 sala . Request (c, out);
 sala . INcorrAccesso (g) (c); < passaggio in out)
 sala . corrRilascio  (g, c);
end utente;
-- le famiglie sono in grassetto

task sala is
 entry INcorrAccesso (gruppo) (corridoio);
 entry OUTcorrAccesso (gruppo) (corridoio);
 entry corrRilascio  (g : in gruppo; c: in corridoio);
 entry Request (c: in corridoio; d: in dir);
end sala;



task body sala is
```

```
     direz : array [corridoio] of dir ;
     nutenti : array [corridoio] of integer;
     cap   : 0.. CAP;

begin
direz [1] := in; direz [2]:= in;
cap := 0; nutenti [1] :+ 0; nutenti [2] := 0;

loop
select
    accept Request (c: in corridoio; d : in dir) do
         req [c, d]
    end Request;
or
-- e' necessaria una alternativa per ogni entry della famiglia:
-- quindi g=1, c=1; g=1, c=2; g=2, c=1; g=2, c=2; ...; g=MAX, c=2
-- when (nutenti [c] = 0) or
--     ((direz[c]= in) and (nutenti + g <= MAX)
--      and (cap + g <= CAP) and (req [c, out] = 0) ==>
--   accept INcorrAccesso (gruppo) (corridoio) do
--        req [c, in] - := 1;
--        cap + := g;
--        nutenti [c] + :=   g;
--        direz [c] := in ;
--   end INcorrAccesso (gruppo) (corridoio);

  -- g = 1, c = 1
  when (nutenti [1] = 0) or
       ((direz[1]= in) and (nutenti + 1 <= MAX)
        and (cap + 1 <= CAP) and (req [1, out] = 0) ==>
     accept INcorrAccesso (1) (1) do
          req [1, in] - := 1;
          cap + := 1;
          nutenti [1] + :=   1;
          direz [1] := in ;
     end INcorrAccesso (1) (1);
or   ...
     ...
or
-- lo stesso  per l'accesso in verso opposto
-- when (nutenti [c] = 0) or
--     ((direz[c]= out) and (nutenti + g<= MAX) ==>
--   accept OUTcorrAccesso (gruppo) (corridoio) do
--        req [c, in] - := 1;
--        cap - := g
--        nutenti [c] + :=   g
```

```
--        direz [c] := out ;
--   end OUTcorrAccesso (gruppo) (corridoio);

-- p.e. g = MAX, c = 2
   when (nutenti [2] = 0) or
       ((direz[2]= out) and (nutenti + MAX <= MAX) ==>
     accept OUTcorrAccesso (MAX) (2) do
          req [2, in] - := 1;
          cap - := MAX;
          nutenti [2] + :=   MAX;
          direz [2] := out ;
     end OUTcorrAccesso (MAX) (2);
or
    accept corrRilascio (g: in gruppo; c: in corridoio) do
        nutenti [c] - := g;
  end corrRilascio;
end select; end loop;

end sala;

u1, u2, ...   : utente;
end ApplicazioneSalaConCorridoi;
```