Esercizio E4.2

Impostazione

Il processo `server` offre ai processi mittenti la entry `invio(in int i)` dove l'intero `i` rappresenta l'indice del processo chiamante (valore compreso tra zero e 4) Chiamando tale entry il generico mittente specifica il proprio indice al `server`, informazione che serve a quest'ultimo per gestire la priorità fra i processi clienti. Infatti, subito dopo aver chiamato la precedente entry `invio`, il mittente i_{esimo} chiama una seconda entry del `server`, la i_{esma} componente dell'array di entry `inserisci[5](in T mes)` a cui passa il valore `mes` di tipo `T` da inserire nel buffer circolare. Quando il server accetta la chiamata di `invio`, se c'è spazio nel buffer circolare accetta subito anche la entry `inserisci` corrispondente al processo chiamante, per inserire il messaggio nel buffer; altrimenti il chiamante resta sospeso sulla chiamata di `inserisci`. In questo caso, quando successivamente un messaggio sarà stato ricevuto da uno qualunque dei processi riceventi e quindi sarà stato fatto spazio nel buffer, il `server` potrà selezionare, fra i vari processi mittenti bloccati, quello di indice più basso, e cioè quello a priorità più alta, svegliandolo mediante l'accettazione della corrispondente chiamata di `inserisci`.
Con analoghi criteri vengoni gestite le chiamate dei processi riceventi offrendo sia la entry `ricezione(in int i)` e l'array di entry `estrai[5](out T mes)`.

Per inviare un messaggio, il processo P_i $(0 \leq i \leq 4)$ segue il seguente schema:

```
server.invio(i);
server.inserisci[i](messaggio);
```

dove `messaggio` denota il valore di tipo `T` da inviare.

Per ricevere un messaggio, il processo P_j $(5 \leq j \leq 9)$ segue il seguente schema:

```
server.ricezione(j);
server.estrai[j](mes);
```

dove `mes` è la variabile di tipo `T` in cui ricevere il messaggio.

Soluzione

```
process server {
  entry invio(in int i);
  entry ricezione(in int i);
  entry inserisci[5](in T mes);
  entry estrai[5](out T mes);

  T buffer[N];
  int testa=0; int coda=0;
  int contatore=0;
  int cliente;
  boolean proc_bloccato[10] /*indicatori necessari per sapere quale processo è bloccato e quale no*/
  int sospesi_mit, sospesi_ric; /*contatori dei processi mittenti bloccati e dei riceventi bloccati*/

  { for(int i=0; i<10; i++) {proc_bloccato[i]=false;}
    sospesi_mit=0;
    sospesi_ric=0;
  } /*inizializzazione*/

  do
     [] accept invio(in int i) {cliente=i;} ->
          if(contatore<N) {  /*c'è spazio nel buffer, il server può accettare il messaggio */
             accept inserisci[cliente](in T mes) {buffer[coda]=mes;}
             coda=(coda+1)%N;
             contatore++;
             if(sospesi_ric>0 { /*ci sono riceventi sospesi e quindi uno (quello di indice più basso) può
                                  essere svegliato accettando la sua chiamata di estrai*/
               int k=5;
               while(! proc_bloccato[k]) k++;
```

```
            proc_bloccato[k]=false;
            sospesi_ric--;
            accept estrai[k](out T mes) {mes=buffer[testa];}
            testa=(testa +1)%N;
            contatore--;
        }
        else { /*non  c'è spazio nel buffer,  il mittente deve rimanere sospeso */
          proc_bloccato[cliente]=true;
          sospesi_mit++;
        }
   [] accept ricezione(in int i) {cliente=i;} ->
        if(contatore>0) {  /*ci sono messaggi nel buffer,  il server può accettare la richiesta */
           accept estrai[cliente](out T mes) {mes=buffer[testa];}
           testa=(testa +1)%N;
           contatore--;
           if(sospesi_mit>0 { /*ci sono mittenti sospesi e quindi uno (quello di indice più basso) può
                                essere svegliato accettando la sua chiamata di inserisci*/
            int h=0;
            while(! proc_bloccato[h]) h++;
            proc_bloccato[h]=false;
            sospesi_mit--;
            accept inserisci[h](in T mes)     {buffer[coda]=mes;}
            coda=(coda +1)%N;
            contatore++;
        }
        else { /*non  ci sono messaggi nel buffer,  il ricevente deve rimanere sospeso */
          proc_bloccato[cliente]=true;
          sospesi_ric++;
        }
  od
}
```