Esercizio E3.5

Impostazione

Lo schema da realizzare nel server corrisponde a quello visto nell'esempio N.3 del paragrafo 5.7.1 con la differenza che, invece di fornire le funzioni del gestore dobbiamo fornire le entry del `server`. Inoltre, dovendo implementare una strategia di allocazione, come previsto dai requisiti imposti per evitare la *starvation*, le richieste in lettura e in scrittura (corrispondenti alle chiamate delle entry `richiestalet` e `richiestascrit` rispettivamente) vengono sempre accettate dal `server` e il processo cliente che ha effettuato le chiamate sulle corrispondenti entry si pone immediatamente in attesa invocando altre entry (la `Oklettura` per le richieste in lettura e `Okscrittura` per quelle in scrittura). Il processo `server` accetterà queste ulteriori chiamate, abilitando quindi il cliente a leggere o scrivere rispettivamente, soltanto quando le corrispondenti condizioni logiche sono verificate.
La condizione logica per abilitare una lettura è che: *non sia attiva una scrittura e che non ci siano processi scrittori in attesa*. Analogamante, la condizione logica per abilitare una scrittura è che: *non sia attiva una scrittura e che non siano attive delle letture*. Il `server` fornisce infine, le entry per indicare le terminazioni sia della lettura (`rilasciolet`) che della scrittura (`rilascioscrit`).
Tutte le entry del `server` servono esclusivamente come meccanismi di sincronizzazione e quindi non hanno parametri. Inoltre, per lo stesso motivo, gli statement `accept` di queste entry sono privi di corpo.

Ogni processo lettore segue il seguente schema:

```
server.richiestalet();
server.Oklettura();
   <lettura>
sever.rilasciolet();
```

Ogni processo scrittore segue il seguente schema:

```
server.richiestascrit();
server.Okscrittura();
   <scrittura>
sever.rilascioscrit();
```


Soluzione

```
process server {
   entry richiestalet();
   entry Oklettura();
   entry rilasciolet();
   entry richiestascrit();
   entry Okscrittura();
   entry rilascioscrit();

   int lettori_attivi=0; /* contatore dei processi che stanno leggendo */
   boolean scrittore_attivo=false; /* indicatore di scrittura attiva */
   int lettori_bloccati=0; /* contatore dei processi lettori bloccati */
   int scrittori-bloccati=0; /* contatore dei processi scrittori bloccati */

   do
      [] accept richiestalet() {} ->
            if(!scrittore_attivo && scrittori_bloccati==0) {
               lettori_attivi++;
               accept Oklettura(){}
            }
```

```
            else lettori_bloccati++;
      [] accept rilasciolet(){} ->
            lettori_attivi--;
            if(lettori_attivi==0 && scrittori_bloccati>0){
               scrittore_attivo=true;
               scrittori_bloccati--;
               accept Okscrittura(){}
            }
      [] accept richiestascrit() {} ->
            if(!scrittore_attivo && lettori_attivi==0) {
               scrittore_attivo=true;
               accept Okscrittura(){}
            }
            else scrittori_bloccati++;
      [] accept rilascioscrit(){} ->
            scrittore_attivo=false;
            if(lettori_bloccati>0) /* alla fine di una scrittura si privilegiano i lettori */
               while(lettori_boccati>0){
                  lettori_attivi++;
                  lettori_bloccati--;
                  accept Okslettura(){}
               }
            else if(scrittori_bloccati>0){
                  scrittore_attivo=true;
                  scrittori_bloccati--;
                  accept Okscrittura(){}
                  }
   od
}
```