Esercizio E3.4

Parte a)

Impostazione

Dall'esame del codice delle due funzioni `fun1` e `fun2` del monitor possiamo ricavare le seguenti specifiche per il server `S` da realizzare:

`S` deve offrire due servizi a due insiemi diversi di processi:

- il servizio relativo all'esecuzione del corpo della funzione `fun1` (statement `S1`). Tale servizio viene richiesto da ciascuno dei tre processi P_1, P_2 e P_3. Queste richieste di servizio devono essere bloccanti per i richiedenti se all'atto della richiesta la condizione `B1` non è verificata. Inoltre, in questo caso, quando `B1` sarà verificata (e ciò accade alla fine dell'esecuzione del corpo `S2` di `fun2`, se più processi sono sospesi, vanno risvegliati privilegiando P_1 su P_2 e P_2 su P_3, come indicato dalla wait con priorità presente nella funzione `fun1` del monitor.
- Il servizio relativo all'esecuzione del corpo della funzione `fun2` (statement `S2`). Tale servizio viene richiesto da altri processi clienti diversa dai tre processi P_1, P_2 e P_3. Queste richieste di servizio devono essere bloccanti per i richiedenti se all'atto della richiesta la condizione `B2` non è verificata. Inoltre, in questo caso, quando `B2` sarà verificata (e ciò accade alla fine dell'esecuzione del corpo `S1` di `fun1`, se più processi sono sospesi, non viene specificata nessuna particolare strategia, come indicato dalla wait ordinaria, senza priorità, presente nella funzione `fun2` del monitor.

Per fornire questi due tipi di servizi, `S` possiede due porte (`pfun1` e `pfun2`) da cui ricevere le corrispondenti richieste di servizio. Poiché `fun1` del monitor ha un parametro `i` di tipo `int` passato per valore, la porta `pfun1` è di tipo `int`. I messaggi che `S` riceve tramite questa porta hanno un valore intero, l'indice `i` del processo cliente che ha chiesto il servizio. Viceversa, la funzione `fun2` del monitor non ha parametri passati per valore. Quindi la porta `pfun2` è di tipo `signal`. Infine, poiché entrambe le funzioni del monitor hanno un parametro (`ris`) di tipo intero passato per riferimento e tramite il quale restituiscono un valore al processo chiamante, ogni cliente dovrà possedere una porta (`risultato`) di tipo `int`, attraverso la quale ricevere il valore del risultato del servizio richiesto, qualunque esso sia.

Operazioni eseguite dal cliente P_i (`i`=1,2,3) per richiedere al server il servizio relativo alla funzione `fun1` (supponendo che `r` sia una variabile locale di tipo `int` alla quale assegnare il valore del risultato del servizio richiesto):


```
send(i) to S.pfun1;
proc=receive(r)from risultato;
```

Operazioni eseguite da qualunque cliente per richiedere al server il servizio relativo alla funzione `fun2` (supponendo che `s` sia un variabile locale di tipo `signal` e `r` sia una variabile locale di tipo `int` alla quale assegnare il valore del risultato del servizio richiesto):

```
send(s) to S.pfun2;
proc=receive(r)from risultato;
```

Soluzione

Alla chiamata `M.fun1(i, r)` da parte di P_i (`i`=1,2,3) corrisponde:

```
send (i) to S.pfun1;
proc = receive(r) from risultato;
```

dove `r` è una variabile locale di tipo `int` alla quale assegnare il valore del risultato del servizio richiesto.

Alla chiamata `M.fun2(r)` da parte di un qualunque altro processo corrisponde:

```
send (s) to S.pfun2;
proc = receive(r) from risultato;
```

dove `:s` è un variabile locale di tipo `signal` e `r` una variabile locale di tipo `int` alla quale assegnare il valore del risultato del servizio richiesto.

```
process S{
   port signal pfun2;
   port int pfun1;

   process proc;
   signal s;
   int ris;
   int i;
   int bloccati; /* contatore dei processi bloccati che hanno richiesto il servizio fun1 */
   boolean sospeso[3]; /* indicatori relativi ai processi P1, P2 e P3. Se
                          sospeso[j]==true allora Pi(con j=i-1)
                          è sospeso sulla richiesta di fun1*/
   process p[3]; /* array di tre variabili di tipo process utilizzate per memorizzare i nomi dei
                    processi (P1, P2 e P3) sospesi */

   { for(int k=0; k<3; k++) sospeso[k]=false;
     lloccati=0;
     p[0]= P1; p[1]= P2; p[2]= P3;
   } /*inizializzazione */

   do
      [] proc=receive(i)from pfun1; -> /* Pi chiede fun1*/
            if(B1){ /* in questo caso il servizio può essere erogato subito*/
               S1; /* S1 assegna alla variabile locale ris il valore da restituire */
               send(ris)to proc.risultato;
               }
            else { /* in questo caso la condizione B1 è falsa e Pi deve attendere */
               bloccati++;
               sospeso[i-1]=true
               }
      [] (B2); proc=receive(s)from pfun2; ->
            /* il servizio fun2 può essere svolto */
            S2;
            send(ris)to proc.risultato;
            /* a questo punto B1 è vera e fun1 può essere eseguita */
            if(bloccati>0){ /* se ci sono clienti sospesi, si sveglia quello a priorità più
                               alta, si esegue S1 e si restituisce il risultato */
               int j=0;
               while(!sospeso[j]) j++;
               bloccati--;
               sospeso[j]=false;
               S1;
               send(ris)to p[j].risultato;
               }
   od
}
```

Parte b)

Impostazione

Col meccanismo del rendez-vous, al posto delle porte vengono utilizzate le entry. Alla funzione `fun2` del monitor facciamo corrispondere la entry `fun2` del server. Questa ha la stessa specifica della funzione `fun2` del monitor. In questo caso il parametro `ris` viene definito di modo `out` dovendo restituire un

valore al chiamate. Inoltre alla chiamata della corrispondente funzione del monitor (`M.fun2(r)`) corrisponde la chiamata alla stessa entry del server. Ovviamente il server accetterà tali chiamate in un ramo di un comando ripetitivo caratterizzato da una guardia logica corrispondente alla condizione `B2` che deve essere vera per eseguire `fun2`.
Per quanto riguarda l'altra funzione del monitor (`fun1`) non possiamo seguire lo stesso semplice schema. Infatti, poiché i processi clienti P_1, P_2 e P_3 che invocano questa funzione devono essere gestiti su base prioritaria, è necessario che il server accetti sempre le richieste di `fun1` anche quando la corrispondente condizione logica `B1` è falsa, in modo tale da sapere sempre chi fra i processi clienti è sospeso e da riattivarli in base alla loro priorità quando `B1` diventerà vera. Per questo motivo è necessario prevedere anche un array di tre entry (`vai[3]`), una per ciascun cliente. Il generico processo P_i invoca la entry `fun1` passandogli il proprio indice `i`. Quindi, si sospende chiamando `vai[i]` in attesa che il server accetti tale chiamata. Queste entry restituiscono valore `ris` del servizio richiesto.
Alla chiamata `M.fun1 (i, r)` da parte di P_i corrisponde quindi:

```
call S.fun1(i);
call S.vai[i-1](r) ;
```

Soluzione:

```
process S{
   entry fun1(int in i);
   entry vai[3]( int out r);
   entry fun2(int out r);

   int val, ris;
   int ris;
   int i;
   int bloccati; /* contatore dei processi bloccati che hanno richiesto il servizio fun1 */
   boolean sospeso[3]; /* indicatori relativi ai processi P1, P2 e P3. Se
                          sospeso[j]==true allora Pi(con j=i-1)
                          è sospeso sulla richiesta di fun1 */

   { for(int k=0; k<3; k++) sospeso[k]=false;
     bloccati=0;
   } /*inizializzazione */

   do
      [] accept fun1(int in i) {val=i; } -> /* Pi chiede fun1*/
            if(B1) /* in questo caso il servizio può essere erogato subito*/
               accept vai[val-1]( int out r) {S1;}; /* S1 assegna alla
                                    variabile locale ris il valore da restituire */
            else { /* in questo caso la condizione B1 è falsa e Pi deve attendere */
               bloccati++;
               sospeso[i-1]=true
               }
      [] (B2); accept fun2(int out r) {
                           /* il servizio fun2 può essere svolto */
                           S2;
                           /* S2 assegna alla variabile locale ris il valore
                              da restituire */
                           }/* a questo punto B1 è vera e fun1 può essere eseguita */
            -> if(bloccati>0){ /* se ci sono clienti sospesi, si sveglia quello a priorità
                                    più alta, si esegue S1 e si restituisce il risultato */
                     int j=0;
                     while(!sospeso[j]) j++;
```

```
                    bloccati--;
                    sospeso[j]=false;
                    accept accept vai[j]( int out r) {S1;}
                }
    od
}

port int risultato
```