Esercizio E3.6

Impostazione
Parte a)

Quali porte:

- Il processo `timer` ha due porte: una (`servizio` di tipo `int`) da cui riceve le richieste di attesa da parte di ciascuno degli `N` processi applicativi; l'intero che rappresenta un messaggio ricevuto tramite la porta, identifica il numero di quanti di tempo durante i quali il processo mittente richiede di rimanere sospeso. L'altra (`tempo` di tipo `signal`) utilizzata per ricevere messaggi da parte del driver dell'orologio in tempo reale. Ogni segnale ricevuto tramite questa porta identifica l'arrivo di una interruzione dall'orologio in tempo reale relativa alla fine di che un quanto di tempo.
- Ogni processo applicativo ha una porta (`wake_up` di tipo `signal`) dalla quale attende di ricevere un segnale da parte del processo `timer` dopo che ha inviato allo stesso un messaggio di richiesta di attesa. L'arrivo di tale segnale indica la fine del periodo di attesa.

Quali scambi di messaggi:

- Fra un processo applicativo e il processo `timer` (azioni relative alla richiesta di un'attesa di `x` quanti di tempo):

```
send (x) to timer.servizio;
proc=receive(s) from wake_up;
```

- Fra il driver delle interruzioni in tempo reale e `timer`:

```
send(s) to timer.tempo;
```


Soluzione

```
process timer {
  port int servizio;
  port signal tempo;

  int quanti;
  signal s;
  process proc;
  process proc_appl[N]; /*array di nomi di processo utilizzato per registrare gli identificatiri dei processi
                                clienti*/
  int attesa[N] /*array di interi utilizzato per registrare il numero di quanti di attesa richiesti da ogni cliente*/

  { for(int i=0; i<N; i++) attesa[i]=0;
    proc_appl[0]= P0; proc_appl[1]= P1;.....; proc_appl[N-1]= PN-1;
  } /*inizializzazione. Si suppone che P0, P1,… siano gli identificatori degli N processi clienti */

  do
    [] proc=receive(quanti) from servizio; -> /*arriva una richiesta di attesa*/
            int  j=0;  while(proc_appl[j]!=  proc)  j++;  /*  a  questo  punto proc_appl[j]
                                                          contiene il nome del processo applicativo richiedente*/
```

```
            attesa[j]=quanti; /*in attesa[j] si registra il numero di quanti di attesa
richiesti*/
   [] proc=receive(s) from tempo; -> /*arriva il segnale di terminazione di un quanto di
tempo */
            for (int k=0; k<N; k++) {
                  if(attesa[k]!=0) { /*se questo è vero, il processo proc_appl[k] è
ancora in
                                          attesa che siano trascorsi tutti i quanti di tempo
richiesti*/
                        attesa[k]--; /*si registra che è passato un ulteriore quanto */
                        if(attesa[k]==0) /*se è vero il processo può essere svegliato */
                              send(s) to proc_appl[k]).wake_up;
                  }
            }
  od
}
```

Parte b)

La soluzione rimane inalterata. Il fatto che le primitive siano sincronizzate non comporta nessun vantaggio in questo caso. Infatti la ricezione della richiesta di attesa da parte del `timer` non abilita il processo applicativo a proseguire. Infatti, il `timer` deve prima ricevere la richiesta di attesa in modo tale da memorizzarla e, successivamente, rispondere con un segnale di sveglia dopo che sono trascorsi i quanti di tempo richiesti.

Parte c)

Adesso al posto delle porte vanno dichiarate le `entry`.
La porta `tempo` viene sostituita da una corrispondente `entry` senza parametri. Per il `timer` è infatti sufficiente ricevere una chiamata di questa `entry` per capire che un quanto di tempo è passato.
Al posto della porta `servizio` possiamo dichiarare una corrispondente `entry`. Un processo cliente, chiamando la entry `servizio` passa al `timer` il numero dei quanti richiesti. Ma, dopo che il `timer` ha ricevuto questa chiamata, il cliente deve rimanere bloccato fino a quando il `timer` non decide di svegliarlo. Per mantenere bloccato il clente possiamo supporre che lo stesso, dopo aver chiamato la entry `servizio`, si sospenda chiamando una ulteriore `entry` (`wake_up`), entry senza parametri e che il processo `timer` dovrà accettare solo dopo che sono trascorsi i quanti di tempo richiesti. Affinchè il timer riesca però a svegliere, selettivamente proprio uno specifico processo cliente, è necessario che a ciascun cliente sia associata nel `timer` una propria entry `wake_up` privata. Per questo motivo, nel `timer` verrà dichiarata non una ma un array di entry `wake_up[N]`, una per ciscun cliente, in modo tale che `timer` possa accettare la entry di uno specifico cliente quando proprio lui deve essere svegliato.
A differenza delle primitive di comunicazione, dove la `receive` restituisce il nome del processo mittente, adesso l'istruzione `accept` non fornisce nessuna informazione sull'identità del processo chiamante. È quindi necessario prevedere un diverso criterio per stabilire su quale entry dell'array `wake_up` un cliente deve bloccarsi. Per risolvere questo problema è sufficiente supporre che il `timer`, quando riceve una chiamata di `servizio`, restituisca al cliente l'indice dell'elemento dell'array `wake_up` su cui il cliente dovrà sospendersi.
Per questo motivo, la entry `servizio` ha due parametri: un parametro intero (`q`) di modo `in` che rappresenta il numero di quanti di attesa e un parametro intero (`ind`) di modo `out` il cui valore compreso tra 0 ed `N`-1 viene restituito dal `timer` al processo applicativo chiamante.
Con le precedenti indicazioni, possiamo indicare la sequenza di azioni che un processo applicativo esegue per richiedere il servizio di attesa programmata:

```
timer.servizio(quanti, ind);
timer.wake_up[ind]();
```

Soluzione

```
process timer {
  entry servizio(in int q, out int ind);
  entry tempo();
  entry wake_up[N]();

  int quanti;
  int indice;
  int attesa[N]; /*ogni elemento di questo array viene utilizzato per registrare il numero di quanti di attesa
     richiesti da un particolare cliente. Se in certo istante attesa[i]==0 ciò significa che l'iesimo elemento
     dell'array non è utilizzato per registrare una qualche richiesta di attesa mentre se attesa[i]!=0 allora
     attesa[i]contiene il numero di quanti che ancora devono trascorrere prima che uno specifico processo cliente
     possa eseere svegliato. Il particolare processo cliente è colui che è sospeso sulla chiamata di wake_up[i] */

  { for(int i=0; i<N; i++) attesa[i]=0;
  } //inizializzazione

  do
    [] accept servizio(in int quanti, out int ind) {
             quanti=q;
             indice=0; while(attesa[indice]!=0] indice++;
             ind=indice; /*restituendo, tramite il parametro ind, il valore di indice, siglifica che il
                                 processo chiamante si sospenderà sulla chiamata di wake_up[indice] */
             } -> attesa[indice]=quanti;

    [] accept tempo(); ->
             for (int k=0; k<N; k++) {
                  if(attesa[k]!=0) {
                       attesa[k]--,
                       if(attesa[k]==0) accept wake_up[k]();
                  }
             }
  od
}
```