Esercizio E1.6

Parte 1)
Impostazione

Se l'evento fosse semplicemente testato da chi deve attendere che sia stato segnalato e semplicemente segnalato da chi è preposto a farlo, sarebbe sufficiente utilizzare un solo *semaforo evento*. Il fatto, però, che chi lo segnala si debba sospendere se l'evento precedentemente segnalato non è stato ancora consumato e che alla sua segnalazione debbano essere svegliati tutti coloro che lo hanno richiesto, implica una soluzione diversa.
Per registrare la presenza dell'evento (*eventoA* già segnalato e non ancora consumato) possiamo riservare in `R` un indicatore di presenza (il boolean `eventoA`). La condizione per eseguire e terminare `testa_A` senza bloccarsi è che l'*eventoA* sia presente (cioè già segnalato ma non ancora consumato), mentre la condizione duale è necessaria per eseguire `segnala_A` senza bloccarsi. Possiamo così specificare le due funzioni:

```
public void testaA():
         region R <<
                    if(<evento disponibile>;) {
                      <"consuma" l'evento>;
                      <if(ci sono processi in attesa di segnalare l'evento>) <attivane uno)>);
                     }
                    else when(<evento disponibile>);
                   >>

public void segnalaA():
         region R <<
                    when(<evento non disponibile>)
                    <"segnala" l'evento>;
                    <if(ci sono processi in attesa dell'evento, attivali tutti)>);
                   >>
```

Riserviamo, a questo scopo, due semafori condizione (`attesaA` e `attesa_nonA`) oltre a due interi destinati a registrare il numero dei processi sospesi sui corrispondenti semafori (`sospesiA` e `sospesi_nonA`).

Soluzione

La risorsa condivisa `R` è un'istanza del seguente tipo di dati astratto:

```
class tipodiR {
   boolean eventoA=false; /* indicatore di disponibilità dell'evento*/
   semaphore mutex=1; /* semaforo di mutua esclusione*/
   semaphore attesaA=0; /* semaforo condizione associato alla presenza dell'evento*/
   semaphore attesa_nonA=0;/*semaforo condizione associato alla non presenza
                                    dell'evento*/
   int sospesiA=0; /* contatore dei processi sospesi in attesa che l'evento si verifichi*/
   int sospesi_nonA=0; /* contatore dei processi sospesi in attesa che l'evento venga
                                 consumato*/

   public void testaA() {
      P(mutex);
      if(eventoA){
         eventoA=false;
         if(sospesi_nonA>0) V(attesa_nonA);
         else V(mutex); /*se ci sono processi in attesa di segnalare l'evento ne
```

```
                                    attiviamo uno con la tecnica del passaggio del testimone*/
    }
    else { /*evento non disponibile. Il processo si sospende in attesa dell'evento. Quando
              viene risvegliato termina la funzione senza dover rientrare in mutua esclusione*/
       sospesiA++;
       V(mutex);
       P(attesaA);
    }
}

public void segnalaA() {
    P(mutex);
    if(eventoA){ /*evento già disponibile. Il processo si sospende in attesa che l'evento sia
                    consumato*/
       sospesi_nonA++;
       V(mutex);
       P(attesa_nonA);
       sospesi_nonA--;
    }
    /* in questo punto l'evento non è disponibile (eventoA==false) e quindi può essere
       segnalato*/
    if((sospesiA>0) /*ci sono processi in attesa che l'evento venga segnalato. Vanno
              svegliati tutti senza che l'indicatore eventoA sia posto al valore true in
              quanto l'evento viene immediatamente consumato dai processi svegliati. Si
              possono svegliare tutti senza problemi perché i processi riprendono la loro
              esecuzione avendo già terminato la funzione testaA senza necessità di
              rientrare in mutua esclusione*/
       while(sospesiA>0) {
          sospesiA--;
          V(attesaA);
       }
    else /*non ci sono processi in attesa dell'evento e quindi è sufficiente segnalarlo*/
       eventoA=true;
    V(mutex);
}
```


Parte 1)
Impostazione

La soluzione al secondo quesito prevede la specifica di una priorità fra i processi segnalanti. Per risolvere il problema è sufficiente sostituire il solo semaforo condizione (`attesa_nonA`) con un vettore di semafori privati, uno per ciascuno degli `N` processi segnalanti.

```
class tipodiR {
   boolean eventoA=false; /* indicatore di disponibilità dell'evento*/
   semaphore mutex=1; /* semaforo di mutua esclusione*/
   semaphore attesaA=0; /* semaforo condizione associato alla presenza dell'evento*/
   semaphore attesa_nonA[N]={0,…..,0};/*array di semafori privati, uno per ciascun
                                        processo segnalante */
   int sospesiA=0; /* contatore dei processi sospesi in attesa che l'evento si verifichi*/
   int sospesi_nonA=0; /* contatore dei processi sospesi in attesa che l'evento venga
                          consumato*/
   boolean bloccato_nonA[N]={false,…..,false};/* array di indicatori, uno per
              processo segnalante. Ciascuno indica se il corrispondente processo è sospeso */
```

```
public void testaA() {
   P(mutex);
   if(eventoA){
      eventoA=false;
      if(sospesi_nonA>0) {/* ci sono processi in attesa di segnalare l'evento attiviamo
                    quello a più alta priorità  con la tecnica del passaggio del testimone*/
         int i=0;
         while(!bloccato_nonA[i]) i++;
         /*il processo segnalante di indice  i  è quello sospeso a più alta priorità e viene
           risvegliato*/
         V(attesa_nonA[i]);
      }
      else V(mutex);
   }
   else { /*evento non disponibile. Il processo si sospende in attesa dell'evento. Quando
             viene risvegliato termina la funzione senza dover rientrare in mutua esclusione*/
      sospesiA++;
      V(mutex);
      P(attesaA);
   }
}
public void segnalaA(int i) {
   P(mutex);
   if(eventoA){ /*evento già disponibile. Il processo si sospende in attesa che l'evento sia
                  consumato*/
      sospesi_nonA++;
      bloccato[i}=true;
      V(mutex);
      P(attesa_nonA[i]);
      bloccato[i}=false;
      sospesi_nonA--;
   }
   /* in questo punto l'evento non è disponibile  (eventoA==false) e quindi può essere
     segnalato*/
   if((sospesiA>0) /*ci sono processi in attesa che l'evento venga segnalato. Vanno
            svegliati tutti senza che l'indicatore eventoA sia posto al valore true in
            quanto l'evento viene immediatamente consumato dai processi svegliati. Si
            possono svegliare tutti senza problemi perché i processi riprendono la loro
            esecuzione avendo già terminato la funzione testaA senza necessità di
            rientrare in mutua esclusione*/
      while(sospesiA>0) {
         sospesiA--;
         V(attesaA);
      }
   else /*non ci sono processi in attesa dell'evento e quindi è sufficiente segnalarlo*/
      eventoA=true;
   V(mutex);
}
```