Esercizio E1.7

Impostazione

La soluzione di questo problema consiste nello scrivere il codice del gestore di un pool di risorse equivalenti con una specifica di priorità fra i processi clienti. Per cui è opportuno fare ricorso ai semafori privati. Poiché la funzione `richiesta` deve restituire un valore, conviene seguire il secondo schema dei semafori privati che utilizza la tecnica del *passaggio del testimone*. Questa tecnica prevede che non si possa svegliare più di un processo alla volta. Poiché nella funzione `tick` può accadere che più processi bloccati debbano essere risvegliati in seguito ad un *time-out*, se ne sveglierà comunque uno solo, quello a più alta priorità. Sarà poi compito suo, al termine delle sua funzione `richiesta`, verificare se altri debbono essere svegliati e, in questi caso, sveglare il secondo e così via.

Soluzione

```
class tipo_gestore {
   semaphore mutex=1; /*semaforo di mutua esclusione*/
   semaphore priv[N]={0,0,…0}; /*semafori privati dei processi clienti*/
   int sospesi=0; /*tiene traccia del numero dei processi clienti sospesi*/
   int disponibili=9; /*tiene traccia del numero di risorse disponibili*/
   boolean libera[9]={true, true,…, true}; /*libera[j] assume il
                     valore true se la risorsa j è libera; false diversamente*/
   int tempo[N]={0,0,…0}; /*se tempo[i]!=0, tale valore indica il numero di quanti di
                     tempo che Pi deve ancora attendere in attesa che termini la sua
                     richiesta prima che scada il time-out da lui specificato all'atto della
                     chamata di richiesta */
   boolean tout[N]={false,…..,false}; /*un flag per processo che indica se lo stesso ha
                     subito time-out in fase di richiesta*/
   /* semplici relazioni di consistenza sulla struttura dati sono:
   ((sospesi>0) => (disponibili==0))
   (sospesi==<numero di elementi del vettore tempo che hanno valori diversi da zero>) */
   public int richiesta(int t, int p) {
      int i;
      P(mutex);
      if(disponibili==0) { /*il richiedente si sospende*/
         sospesi++;
         tempo[p]=t;
         V(mutex);
         P(priv[p]);
         tempo[p]=0;
         sospesi++;
      }
      /* il processo arriva a questo punto o perché disponibili>0 oppure dopo essere stato
         risvegliato in seguito a un time-out*/
      if(tout[p]){ /* il processo ha subito time-out. La funzione termina restituendo il valore
            0. Però prima di terminare deve verificare se qualche altro cliente, meno importante
            di lui, abbiasubito a sua volta time-out ma non è stato svegliato nella funzione tick
            perché con la tecnica del passaggio del testimone si può svegliare un solo processo
            alla volta. Se questo è vero anche il successivo processo che ha subito time-out va
            svegliato, e così via*/
         do {p++; if(tout[p])break;}
```

```
        while(p<N);
        if(p<N) V(priv[p];{/* un altro processo che ha subito time-out viene svegliato.*/
        else V(mutex); /* altrimenti si rilascia la ME (passaggio del testimone)*/
        return 0;

        else { /*la funzione può terminare restituendo l'indice della risorsa allocata*/
        i=0;
        while(!libera[i]) i++;
        libera[i]=false;
        disponibili--;
        V(mutex);
        return (i+1);
    }

    public void rilascio(int r) {
       int p;
       P(mutex);
       libera[r-1]=true;
       disponibili++;
       if(sospesi>0) {
          p=0;
          while(tempo[p]==0) p++;
          V(priv[p]);
       else V(mutex);
    }

    public void tick {
       int p=0; boolean da_svegliare=false;
       P(mutex);'
       for(p=0; p<N; p++) { /*si cercano tutti i possibili processi bloccati che hanno
                                    subito time-out*/
          if(tempo[p]!=0) { /*processo bloccato*/
             tempo[p]--;
             if(tempo[p]==0) { /*time-out*/
                tout[p]=true;
                da_svegliare=true;
             }
          }
       }
       if(da_svegliare) { /*sono stati trovati processi che hanno subito time-out*/
          p=0;
          while(!tout[p]) p++;
          V(priv[p]); /*si sveglia il più importante*/
       }
       else V(mutex);
    }
}
```