Esercizio E1.5

Parte 1)
Impostazione

Il problema può essere considerato una generalizzazione di quello dei *lettori/scrittori*. Possiamo quindi utilizzare i semafori privati con il primo dei due schemi relativi al loro uso. Conviene il primo schema dovendo svegliare più processi contemporaneamente in fase di rilascio. Le condizioni di sincronizzazione per accettare le richieste sono le segenti;

per la `richiesta_A`: `(attivi_B==0)&&(attivi_C==0)`
per la `richiesta_B`: `(attivi_A==0)&&(attivi_C==0)`
per la `richiesta_C`: `(attivi_A==0)&&(attivi_B==0)`

Soluzione

```
Class gestore {
   semaphore  mutex=1; /*semaforo di mutua esclusione*/
   semaphore   priv_A=0; /*semaforo privato associato a chi richiede OpA*/
   semaphore   priv_B=0; /*semaforo privato associato a chi richiede OpB*/
   semaphore   priv_C=0; /*semaforo privato associato a chi richiede OpC*/
   int sospesi_A=0; /*contatore dei processi sospesi in attesa di eseguire OpA*/
   int sospesi_B=0; /*contatore dei processi sospesi in attesa di eseguire OpB*/
   int sospesi_C=0; /*contatore dei processi sospesi in attesa di eseguire OpC*/
   int attivi_A=0; /*contatore dei processi che stanno eseguendo OpA*/
   int attivi_B=0; /*contatore dei processi che stanno eseguendo OpB*/
   int attivi_C=0; /*contatore dei processi che stanno eseguendo OpC*/

   public void richiesta_A() {
      P(mutex);
      if((attivi_B==0)&&(attivi_C==0)){ /*la condizione per eseguire OpA è vera*/
        attivi_A++;
        V(priv_A); /* V preventiva su priv_A per non bloccarsi alla fine della funzione*/
      }
      else sospesi_A++;/*condizione di sincronizzazione non verificata, viene indicato che
                         il processo si blocca*/
      V(mutex);
      P(priv_A);
   }
```

La funzione `richiesta_B` è identica. Basta cambiare `A` con `B`, `B` con `C` e `C` con `A`. Analogamente per `richiesta_C` cambiando `A` con `C`, e `C` con `A`.

```
   public void rilascio_A() {
      P(mutex);
      attivi_A--;
      if(attivi_A==0){ /*non ci sono più OpA in esecuzione*/
         if(sospesi_B>0) /* si privilegiano eventuali richieste di opB rispetto a OpC*/
            while(sospesi_B>0){ /* si attivano tutte le richieste pendenti di opB*/
               sospesi_B--;
               attivi_B++;
               V(priv_B);
            }
         else if(sospesi_C>0)
                  while(sospesi_C>0){ /* si attivano tutte le richieste pendenti di opC*/
```

```
                    sospesi_C--;
                    attivi_C++;
                     V(priv_C);
                  }
      V(mutex);
```

La funzione `rilascio_B` è identica. Basta cambiare `A` con `B`, `B` con `A`. Analogamente per `richiesta_C` cambiando `A` con `C`, `B` con `A` e `C` con `B`.

La soluzione presentata è soggetta a possibili condizioni di starvation. Quando un gruppo acquisisce il controllo della risorsa può non rilasciarlo più se prima che l'ultimo del gruppo rilasci la risorsa altri processi del gruppo continuano a richiederla.

Parte 2)
Impostazione

In assenza di regole di priorità, per eliminare il pericolo della starvation possiamo imporre ulteriori requisiti simili a quelli visti per il problema *lettori/scrittori*. Ad esempio:

- una nuova richiesta non può essere accettata se ci sono pendenti richieste di altre operazioni.
- se sono attive esecuzioni di `OpA`, quando termina l'ultima di queste, vengono attivate, se ce ne sono, tutte le richieste pendenti di `OpB` altrimenti quelle, se ce ne sono, di `OpC`.
- se sono attive esecuzioni di `OpB`, quando termina l'ultima di queste, vengono attivate, se ce ne sono, tutte le richieste pendenti di `OpC` altrimenti quelle, se ce ne sono, di `OpA`.
- se sono attive esecuzioni di `OpC`, quando termina l'ultima di queste, vengono attivate, se ce ne sono, tutte le richieste pendenti di `OpA` altrimenti quelle, se ce ne sono, di `OpB`.

In questo modo nessun gruppo può generare starvation di altri. Infatti se sono attive esecuzioni di una operazione, altre richieste della stessa sono inibite appena si sospende il richiedente di un'altra operazione. Al termine delle operazioni di un gruppo la priorità con cui si attivano le altre richieste pendenti è circolare: alla fine di `OpA` si privilegia `OpB`, alla fine di `OpB` si privilegia `OpC` e alla fine di `OpC` si prioviegia `OpA`. In questo modo nessuno può essere escluso per sempre.
Con questi ulteriori requisiti, le condizioni di sincronizzazione per accettare le richieste sono le segenti;

per la `richiesta_A`:
`(attivi_B==0)&&(attivi_C==0)&&(sospesi_B==0)&&(sospesi_C==0)`
per la `richiesta_B`:
`(attivi_A==0)&&(attivi_C==0)&&(sospesi_A==0)&&(sospesi_C==0)`
per la `richiesta_C`:
`(attivi_A==0)&&(attivi_B==0)&&(sospesi_A==0)&&(sospesi_B==0)`

Soluzione

```
Class gestore {
   semaphore  mutex=1; /*semaforo di mutua esclusione*/
   semaphore   priv_A=0; /*semaforo privato associato a chi richiede OpA*/
   semaphore   priv_B=0; /*semaforo privato associato a chi richiede OpB*/
   semaphore   priv_C=0; /*semaforo privato associato a chi richiede OpC*/
   int sospesi_A=0; /*contatore dei processi sospesi in  attesa di eseguire  OpA*/
   int sospesi_B=0; /*contatore dei processi sospesi in  attesa di eseguire  OpB*/
   int sospesi_C=0; /*contatore dei processi sospesi in  attesa di eseguire  OpC*/
   int attivi_A=0; /*contatore dei processi che stanno eseguendo  OpA*/
   int attivi_B=0; /*contatore dei processi che stanno eseguendo  OpB*/
   int attivi_C=0; /*contatore dei processi che stanno eseguendo  OpC*/
```

```
public void richiesta_A() {
   P(mutex);
   if((attivi_B==0)&&(attivi_C==0)
                        &&(sospesi_B==0)&&(sospesi_C==0)){
      attivi_A++;
      V(priv_A); /* V preventiva su priv_A per non bloccarsi alla fine della funzione*/
   }
   else sospesi_A++;/*condizione di sincronizzazione non verificata, viene indicato che
                        il processo si blocca*/
   V(mutex);
   P(priv_A);
}
```

La funzione `richiesta_B` è identica. Basta cambiare `A` con `B`, `B` con `C` e `C` con `A`. Analogamente per `richiesta_C` cambiando `A` con `C`, e `C` con `A`.

```
public void rilascio_A() {
   P(mutex);
   attivi_A--;
   if(attivi_A==0){ /*non ci sono più OpA in esecuzione*/
      if(sospesi_B>0) /* si privilegiano eventuali richieste di opB rispetto a OpC*/
         while(sospesi_B>0){ /* si attivano tutte le richieste pendenti di opB*/
            V(priv_B);
            sospesi_B--;
            attivi_B++;
         }
      else if(sospesi_C>0)
               while(sospesi_C>0){ /* si attivano tutte le richieste pendenti di opC*/
                 V(priv_C);
                 sospesi_C--;
                 attivi_C++;
               }
   V(mutex);
```

La funzione `rilascio_B` è identica. Basta cambiare `A` con `B`, `B` con `C` e `C` con `A`. Analogamente per `richiesta_C` cambiando `A` con `C`, `B` con `A` e `C` con `B`.