Esercizio E1.8

Parte a)
Impostazione

1. Quali tipi di dati?

   - la struttura `busta` utilizzata come supporto per l'invio di messaggi. La struttura contiene due campi uno di tipo `T` per contenere l'informazione relativa al messaggio l'altro di tipo `int` destinato a contenere l'indice della busta successiva quando una busta viene concatenata nella lista FIFO costituente la mailbx;
   - un tipo astratto, `gestore`, tipo dell'oggetto `gestore_buste` che, condiviso tra tutti i processi, alloca a ciascuno degli `M` mittenti una busta mediante la funzione `richiesta` e a cui viene restituita una busta da parte di ciascuno degli `R` processi riceventi mediante la funzione `rilascio`;
   - un tipo astratto `coda_FIFO`, realizzata mediante una lista concatenata di buste e gestita in ordine FIFO che rappresenta il tipo dell'oggetto condivio `mailbox`.

2 Quali stutture dati?

   - l'array `buste` di `N` istanze del tipo `busta` che costituiscono le risorse allocate tramite il `gestore_buste`;
   - l'oggetto `gestore_buste`, istanza del tipo astratto `gestore`
   - l'oggetto `mailbox`, istanza del tipo astratto `coda_FIFO`.

3 Le funzioni

```
int richiesta(int priorità);
```
/*funzione del `gestore_buste` che alloca al richiedente, se disponibile, una `busta`, elemento dell'array `buste` restitendone l'indice, o blocca il processo se non ci sono buste disponibili*/

```
void rilascio(int b);
```
/*funzione del `gestore_buste` con cui un processo ricevente restituisce al gestore una busta*/

```
void inserimento(int b);
```
/*funzione della `mailbox` con la quale si inserisce una busta nella coda FIFO che la rappresenta*/

```
int estrazione();
```
/*funzione della `mailbox` con la quale si estrae una busta, se disponibile, dalla `coda_FIFO` che la rappresenta. Se in coda non ci sono buste il processo si sospende*/

```
void invio(T messaggio, int prio);
```
/*funzione invocata da ognuno degli `M` processi mittenti per inviare il `messaggio` indicando il valore `prio` della priorità con cui inviarlo/

```
T ricezione();
```
/*funzione invocata da ognuno degli `R` riceventi per ricevere un messaggio che la funzione restituisce*/

Soluzione

```
/* Definizione della struttura busta*/
typedef struct {
   T mes;
   int successiva;
```

```
} busta
```

/* Dichiarazione del `pool` di `buste`*/

```
busta buste[N];
```

/* Definizione del tipo astratto `gestore`*/

/* tipico gestore di un pool di risorse equivalenti che alloca in base alla priorità, passata come parametro della richiesta da parte dei richiedenti. Per cui verranno usati i semafori privati (uno per ogni livello di priorità). Dovendo restituire, in fase di richiesta, l'indice della busta allocata, conviene fare riferimento al secondo schema che usa la tecnica del *passaggio del restimone*/

```
class gestore {
   boolean libera[N]={true, …, true};
   int disponibili=N; /* intero che indica il numero di buste ancora disponibili */
   semaphore mutex=1; /* semaforo di mutua esclusione*/
   semaphore priv[3]={0,0,0}; /*semafori privati, uno per ogni livello di priorità*/
   int sospesi[3]= ]={0,0,0};/*contatori dei processi sospesi su ciascun livello di priorità*/

   public int richiesta(int priorità){
      int i;
      P(mutex);
      if(disponbili==0) {
         sospesi[priorità]++;
         V(mutex);
         P(priv[priorità]);
         sospesi[priorità]--;
         }
      i=0;
      while(!libera[i]) i++;
      libera[i]=false;
      disponibili--;
      V(mutex);
      return i;
   }

   public void rilascio(int b){

      P(mutex);

      disponibili++;

      libera[b]=true;

      if(sospesi[0]>0) V(priv[0]);

      else if(sospesi[1]>0) V(priv[1]);

           else if(sospesi[2]>0) V(priv[2]);

                else V(mutex);

   }
```

/* Dichiarazione dell'oggetto `gestore_buste`*/

```
gestore gestore_buste;
```

/* Definizione del tipo astratto `coda_FIFO`*/

/*Dovendo definire una coda di buste realizzata mediante una lista concatenata sono sufficienti due sole variabili: l'indice (`prima`) della prima busta della lista e l'indice (`ultima`) di quella finale, ultima inserita. Le buste contenute nella lista sono concatenate mediante il campo `successiva` di ogni busta

destinato a contenere l'indice della busta successiva nella lista. Gli indici `prima` e `ultima` contengono il valore –1 quando la lista è vuota e, analogamente, è -1 il campo succesivo dell'ultima busta della lista*/

```
class coda_FIFO {
  semaphore mutex=1; /* semaforo di mutua esclusione*/
  semaphore elementi=0; /* semaforo risorsa che conta il numero di elementi della lista*/
  int prima=-1;
  int ultima=-1;
  public void inserimento(int b){
     P(mutex);
     if(prima==-1) prima=b;
     else ultima.successiva=b;
     ultima=b;
     ultima.successiva=-1;
     V(mutex);
     V(elementi);
  }
  public int estrazione(){
     int da_restituire;
     P(elementi);
     P(mutex);
     da_restituire=prima;
     prima=prima.successiva;
     if(prima==-1) ultima=-1;
     V(mutex);
     return da_restituire;
  }
/* Dichiarazione dell'oggetto mailbox*/
  coda_FIFO mailbox;
/* funzione invio*/
   void invio(T messaggio, int prio){
      int b;
      b=gestore_buste.richiesta(prio);
      buste[b].mes=messaggio;
      mailbox.inserimento(b);
   }
/* funzione ricezione*/
   T ricezione(){
      int b;
      T messaggio;
      b=mailbox.estrazione();
```

```
        messaggio=buste[b].mes;
        gestore_buste.rilascio(b);
        return messaggio;
    }
```

Parte b)
Impostazione

La soluzione è simile alla precedente. Si tratta di utilizzare i monitor al posto dei semafori. Dovremo ancora definire gli stessi tipi. Il tipo `busta`, che rimane del tutto identico a prima; il tipo `gestore` che adesso è un `monitor` e il tipo `coda_FIFO` che di nuovo è un `monitor`.

Soluzione

/* Definizione della struttura `busta`*/

```
typedef struct {
   T mes;
   int successiva;
} busta
```

/* Dichiarazione del `pool` di `buste`*/

```
busta buste[N];
```

/* Definizione del tipo astratto `gestore`*/

/* Al posto dei semafori privati adesso vengono usate variabili `condition` private. Supponiamo di utilizzare la primitiva `signal` con la semantica *signal_and_urgent wait* */

```
monitor gestore {
   boolean libera[N]={true, …, true};
   int disponibili=N; /* intero che indica il numero di buste ancora disponibili */
   condition priv[3]; /*semafori privati, uno per ogni livello di priorità*/
   int sospesi[3]={0,0,0};/*contatori dei processi sospesi su ciascun livello di priorità*/
   public int richiesta(int priorità){
      int i;
      if(disponbili==0) { /*se fosse stata usata la semantica signal_and_continue al posto
                           dell'if avremmo dovuto utilizzare un while*/
         sospesi[priorità]++;
         wait(priv[priorità]);
         sospesi[priorità]--;
         }
      i=0;
      while(!libera[i]) i++;
      libera[i]=false;
      disponibili--;
      return i;
   }
   public void rilascio(int b){
      disponibili++;
      libera[b]=true;
      if(sospesi[0]>0) signal(priv[0]);
      else if(sospesi[1]>0) signal (priv[1]);
```

```
            else signal (priv[2]);
    }
```

/* usando la primitiva `empty` potremmo fare a meno dell'array di contatori `sospesi`. Infatti nel `rilascio` per verificare se ci sono processi sospesi sulle singole `condition` potremmo usare questa primitiva. Ad esempio, il primo `if` diventerebbe: `if(!empty(priv[0]))` e così via*/

/* avendo a disposizione anche la primitiva `wait` con priorità, la soluzione sarebbe stata ancora più semplice potendo dichiarare una sola variabile condition `priv`. Di seguito riscriviamo le due funzioni in questa ipotesi: */

```
    public int richiesta(int priorità){
        int i;
        if(disponbili==0) wait(priv,priorità);
        i=0;
        while(!libera[i]) i++;
        libera[i]=false;
        disponibili--;
        return i;
    }
    public void rilascio(int b){
        disponibili++;
        libera[b]=true;
        signal(priv);
    }
```

/* Dichiarazione dell'oggetto `gestore_buste`*/

```
  gestore gestore_buste;
```

/* Definizione del tipo astratto `coda_FIFO`*/

/*Supponiamo ancora di utilizzare la primitiva `signal` con la semantica *signal_and_urgent wait* */

```
  monitor coda_FIFO {
    int elementi=0; /* intero che conta il numero di elementi della lista*/
    condition cond; /*variabile su cui si blocca chi tenta di estrarre dalla coda vuota*/
    int prima=-1;
    int ultima=-1;
    public void inserimento(int b){
        if(prima==-1) prima=b;
        else ultima.successiva=b;
        ultima=b;
        ultima.successiva=-1;
        elementi++;
        signal(cond);
    }
    public int estrazione(){
```

```
      int da_restituire;
      if(elementi==0) wait(cond);;
      da_restituire=prima;
      prima=prima.successiva;
      if(prima==-1) ultima=-1;
      elementi--;
      return da_restituire;
   }
```

/* Dichiarazione dell'oggetto `mailbox`*/

```
  coda_FIFO mailbox;
```

/* Le due funzioni `invio` e `ricezione` ovviamente non cambiano*/

```
   void invio(T messaggio, int prio){
      int b;
      b=gestore_buste.richiesta(prio);
      buste[b].mes=messaggio;
      mailbox.inserimento(b);
   }

   T ricezione(){
      int b;
      T messaggio;
      b=mailbox.estrazione();
      messaggio=buste[b].mes;
      gestore_buste.rilascio(b);
      return messaggio;
   }
```