Esercizio E1.3

Parte 1)
Impostazione

La soluzione corrisponde alla realizzazione di tre mailbox sovrapposte nel senso che le tre mailbox condividono lo stesso vettore circolare (`buffer`) dove, essendoci un solo mittente è sufficiente un solo indice (`ultimo`) che indichi dove inserire il prossimo messaggio, ma tre diversi indici (`primo1`, `primo2`, `primo3`) per indicare a ciascun ricevente da dove prelevare il prossimo messaggio. Analogamente è sufficiente un solo semaforo risorsa (`vuoto`) che indica il numero di posizioni vuote nel vettore, ma sono necessari tre diversi semafori risorsa (`pieno1`, `pieno2`, `pieno3`) per indicare rispettivamente a ciascun ricevente il numero di elementi del vettore contenenti messaggi che devono essere ancora prelevati. Sono anche utili tre contatori (`c1`, `c2`, `c3`) ciascuno dei quali ha un valore che coincide col valore del corrispondente semaforo. Quindi il numero di elementi vuoti del vettore coincide con (10-max) se max è il massimo tra i valori di `c1`, `c2` e `c3`. Paerciò l'estrazione di un dato dal vettore da parte del processo `Ri` non significa necessariamente che si generi un elemento vuoto in più. Questo accade solo se il dato estratto dal processo è già stato estratto anche dagli altri e cioè se dopo l'esrazione `ci` è ancora maggiore o uguale agli altri due contatori.

Possiamo quindi concludere che per la mailbox valgono le seguenti relazioni di consistenza:

- (Nº elementi vuoti di `buffer`) == val_{vuoto} == 10−max(c1, c2, c3)
- ci == val_{pienoi} (per i=1,2,3)
- (ultimo − primoi)%10 == ci == val_{pienoi} (per i=1,2,3)

Soluzione

```
class mailbox {
   T buffer[10]; /* vettore circolare*/
   int  ultimo=0;   /* rappresenta l'indice dell'elemento di buffer in cui inserire il prossimo
                       messaggio*/
   int  primo1=0; /* rappresenta l'indice dell'elemento di buffer da cui R1 preleva il prossimo
                     messaggio*/
   int primo2=0; /* analogamente per R2*/
   int primo3=0; /* analogamente per R3*/
   semaphore mutex=1;  /*semaforo di mutua esclusione*/
   semaphore vuoto=10;  /* semaforo risorsa che indica il numero di elementi vuoti nel buffer*/
   semaphore pieno1=0;  /*semaforo risorsa che indica gli elementi pieni per R1*/
   semaphore pieno2=0;  /* analogamente pre R2*/
   semaphore pieno3=0;  /* analogamente pre R3*/
   int  c1=0;  /* numero di messaggi presenti nel buffer e che R1 deve ancora ricevere (coincide col
                  valore di pieno1) */
   int c2=0;  /* analogamente per R2*/
   int c3=0;  /* analogamente per R3*/

   public void send (T messaggio) {
       P(vuoto);
       buffer[ultimo]=messaggio;
       ultimo=(ultimo+1)%10;
       P(mutex)
         c1++;  c2++;  c3++;
       V(mutex);
       V(pieno1);  V(pieno2);  V(pieno3);
   }
```

```
   public T receive1() {
       T messaggio;
       P(pieno1);
       P(mutex);
       messaggio=buffer[primo1];
       primo1=(primo1+1)%10;
       c1--;
       if (c1≥c2 && c1≥c3) /*se questo è vero significa che prima del decremento c1 era
                    sicuramente il massimo tra i tre contatori e quindi, dopo il suo decremento c'è un
                    elemento vuoto in più nel vettore */
            V(vuoto);
       V(mutex);
       return messaggio;
   }
   // la receive2 si può facilmente ottenere da receive1 cambiando 1 con 2, 2 con 3 e 3 con 1.
     Analogamente per la receive3
}
```

Parte 2)
Impostazione

La struttura dati della classe resta inalterata salvo i tre semafori `pieno1`, `pieno2` e `pieno3` che vengono sostituiti da tre semafori privati (`priv1`, `priv2`, `priv3`) e da tre indicatori booleani (`bloccato1`, `bloccato2`, `bloccato3`). Per garantire la priorità richiesta viene utilizzato lo schema con *passaggio di testimone*

```
class mailbox {
   T buffer[10]; /* vettore circolare*/
   int ultimo=0;  /* rappresenta l'indice dell'elemento di buffer in cui inserire il prossimo
                     messaggio*/
   int primo1=0; /* rappresenta l'indice dell'elemento di buffer da cui R1 preleva il prossimo
                    messaggio*/
   int primo2=0; /* analogamente per R2*/
   int primo3=0; /* analogamente per R3*/
   semaphore mutex=1; /*semaforo di mutua esclusione*/
   semaphore vuoto=10; /* semaforo risorsa che indica il numero di elementi vuoti nel buffer*/
   int c1=0; /* numero di messaggi presenti nel buffer e che R1 deve ancora ricevere */
   int c2=0; /* analogamente per R2*/
   int c3=0; /* analogamente per R3*/
   semaphore priv1=0; //semaforo pivato di R1
   semaphore priv2=0; // analogamente pre R2
   semaphore priv3=0; // analogamente pre R3
   boolean bloccato1=false; //indica quando R1 è sospeso
   boolean bloccato2=false; // analogamente pre R2
   boolean bloccato3=false; // analogamente pre R3

   public void send (T messaggio) {
       P(vuoto);
       buffer[ultimo]=messaggio;
       ultimo=(ultimo+1)%10;
       P(mutex)
       c1++;  c2++;  c3++;
       if (bloccato1) V(priv1);
```

```
    else if (bloccato2) V(priv2);
         else if (bloccato3) V(priv3);
              else V(mutex):
}
public T receive1() {
    T messaggio;
    P(mutex);
    if (c1 == 0) {
         bloccato1=true;
         V(mutex);
         P(priv1);
         bloccato1=false;
         }
    messaggio=buffer[primo1]
    primo1=(primo1+1)%10;
    c1--;
    if (c1≥c2 && c1≥c3) V(vuoto);
    if (bloccato2 && c2>0) V(priv2);
    else if ((bloccato3 && c3>0) V(priv3);
         else  V(mutex);
    return messaggio;
}
```

// la `receive2` e la `receive3` si ottengono in modo analogo