Esercizio E3.2

Parte1)

Impostazione

Si suppone che ogni processo cliente chieda al `server` l'uso esclusivo del servizio offerto da `Ra` inviando un segnale sulla porta `ricRa` del `server` e rilasci tale servizio inviando un segnale sulla porta `rilRa` del `server`. Analogamente per richiedere e rilasciare il servizio offerto da `Rb` invia il segnale rispettivamente sulle porte `ricRb` e `rilRb` del `server`. Infine, per richiedere contemporaneamente i servizi di `Ra` e `Rb` invia il segnale rispettivamente sulle porta `ricRaRb` e `rilRaRb` del `server`. Poiche le primitive di comunicazione sono sincrone, per mantenere bloccato un processo quando la risorsa, o le risorse, richieste non sono disponibili, è sufficiente fare in modo che il `server` non riceva la richiesta. Quindi, quando una richiesta da parte di un cliente viene ricevuta dal `server`, il cliente termina l'esecuzione della corrispondente primitiva `send` con cui ha invisto la richiesta e può proseguire certo che la risorsa, o le risorse, richieste sono state allocate.

Richiesta della risorsa Ra da parte di un processo cliente: `send(s)to server.ricRa`

Richiesta della risorsa Rb da parte di un processo cliente: `send(s)to server.ricRb`

Richiesta di Ra e Rb da parte di un processo cliente: `send(s)to server.ricRaRb`

Rilascio della risorsa Ra da parte di un processo cliente: `send(s)to server.rilRa`

Rilascio della risorsa Rb da parte di un processo cliente: `send(s)to server.rilRb`

Rilascio di Ra e Rb da parte di un processo cliente: `send(s)to server.rilRaRb`

Soluzione

```
process server{
  port signal ricRa, rilRa, ricRb, rilRb, ricRaRb, rilRaRb;

  process proc;
  signal s;
  boolean liberoA=true, liberoB=true;

  do
   [] (liberoA); proc=receive(s)from ricRa; ->
         liberoA=false;
   []  proc=receive(s)from rilRa; ->
         liberoA=true;
   [] (liberoB); proc=receive(s)from ricRb; ->
         liberoB=false;
   []  proc=receive(s)from rilRb; ->
         liberoB=true;
```

```
    [] (liberoA && liberoB); proc=receive(s)from ricRaRb; ->
          liberoA=false; liberoB=false;
    []  proc=receive(s)from rilRaRb; ->
          liberoA=true; liberoB=true;
  od
}
```

Parte 2)
Impostazione

Dovendo implementare una strategia di allocazione, il `server` deve essere messo in condizione di sapere, nel momento in cui una risorsa viene rilasciata, quali richieste sono pendenti in modo tale da effettuare le proprie scelte sulla base della strategia da implementare. Ciò implica che il `server` deve ricevere una richiesta anche quando non e in grado di servirla subito con l'ottica di registrare tale richiesta come richiesta pendente da servire in un secondo momento. Quindi, la ricezione di una richiesta da parte di un cliente non deve abilitare il cliente stesso ad operare sulla risorsa richiesta (o sulle risorse richieste) come avviene nella soluzione precedente tenendo conto che le primitive di comunicazione sono sincrone. È quindi necessario ricorrere ad uno schema analogo a quello che avremmo dovuto implementare se le primitive fossero state asincrone, aggiungendo nel `server` ulteriori porte (ancora di tipo `signal`) e modificando il protocollo mediante il quale un cliente chiede la, o le, risporse al gestore:

Richiesta della risorsa Ra da parte di un processo cliente:

```
send(s)to server.ricRa;
send(s)to server.risorsaA;
```

Richiesta della risorsa Rb da parte di un processo cliente:

```
send(s)to server.ricRb;
send(s)to server.risorsaB;
```

Richiesta di Ra e Rb da parte di un processo cliente:

```
send(s)to server.ricRaRb;
send(s)to server.risorseAeB;
```

Rilascio della risorsa Ra da parte di un processo cliente:

```
send(s)to server.rilRa;
```

Rilascio della risorsa Rb da parte di un processo cliente:

```
send(s)to server.rilRb;
```

Rilascio di Ra e Rb da parte di un processo cliente:

```
send(s)to server.rilRaRb;
```

Soluzione

```
process server{
  port signal ricRa, rilRa, ricRb, rilRb, ricRaRb, rilRaRb;
  port signal risorsaA, risorsaB, risorseAeB;

  process proc;
  signal s;
  boolean liberoA=true, liberoB=true;
```

```
  int sospesiA=0, sospesiB=0, sospesiAB=0; /* contatori dei processi sospesi
                                              sulle relative richieste*/
  do
   []  proc=receive(s)from ricRa; ->
         if(liberoA) {
            liberoA=false;
            proc=receive(s)from risorsaA; }

         else {sospesiA++;}
   []  proc=receive(s)from rilRa; ->
         if(sospesiA>0) proc=receive(s)from risorsaA;
         else if(sospesiAB>0 && liberoB) {
                  liberoB=false;
                  proc=receive(s)from risorseAeB; }
              else liberoA=true;
   []  proc=receive(s)from ricRb; ->
         if(liberoB) {
            liberoB=false;
            proc=receive(s)from risorsaB;
            }
         else {sospesiB++;}
   []  proc=receive(s)from rilRb; ->
         if(sospesiB>0) proc=receive(s)from risorsaB;
         else if(sospesiAB>0 && liberoA) {
                  liberoA=false;
                  proc=receive(s)from risorseAeB; }
              else liberoB=true;
   []  proc=receive(s)from ricRaRb; ->
         if(liberoA && liberoB) {
            liberoA=false; liberoB=false;
            proc=receive(s)from risorseAeB;
            }
         else {sospesiAB++;}
   []  proc=receive(s)from rilRaRb; ->
         if(sospesiA>0) {
            proc=receive(s)from risorsaA;
            if((sospesiB>0) proc=receive(s)from risorsaB;
            else liberoB=true; }
         else if((sospesiB>0) {
                  proc=receive(s)from risorsaB;
                  liberoA=true; }
              else if(sospesiAB>0) proc=receive(s)from risorseAeB;
                   else { liberoA=true; liberoB=true; }
  od
}
```