Programmazione concorrente e distribuita
Paolo Ancilotti, Maurelio Boari

Esercizio E2.11

Parte a)

Prima soluzione errata

L'errore presente nella prima soluzione è il seguente:
come indicato nel testo, la classe `IPCM` è condivisa fra più processi mittenti. Per garantire la mutua esclusione tra due o più mittenti viene utilizzata l'istanza `mailbox` del monitor `buffer_circolare` per cui la funzione `invio`, in quanto funzione di monitor, viene eseguita in mutua esclusione anche se contemporaneamente invocata da mittenti diversi avendo, gli stessi, invocato contemporaneamente la funzione `send`. Può accadere, però, che tra due esecuzioni consecutive di `invio` da parte di un processo mittente, la stessa funzione possa essere eseguita da un altro mittente con la conseguenza che, all'interno del monitor `mailbox`, messaggi inviati da mittenti diversi possono mescolarsi.

Seconda soluzione errata

L'errore presente nella seconda soluzione è il seguente:
aver definito `IPCM` come monitor e non come classe garantisce che le due funzioni `send` e `receive` siano eseguite in mutua esclusione e ciò elimina l'inconveniente della precedente soluzione. In questo caso, però, il monitor `mailbox` viene dichiarato locale al monitor `IPCM`. Si possono quindi verificare gli inconvenienti legati alle chiamate innestate a funzioni di monitor. Ad esempio, se un mittente invoca la `send` passandole un messaggio composto da più di 5 caratteri, dopo un massimo di cinque esecuzioni consecutive della funzione `invio` la `mailbox` è piena, per cui all'inizio della sesta esecuzione il processo si sospende sulla variabile condizione `non_pieno`, liberando la mutua esclusione del monitor `mailbox` ma mantenendo quella del monitor `IPCM`. Di conseguenza si verifica una condizione di stallo poiché l'unico processo che può risvegliare il mittente sospeso è il processo ricevente che però, quando invoca la funzione `receive`, si sospende per mutua esclusione su `IPCM`.

Terza soluzione errata

L'errore presente nella terza soluzione è il seguente:
questa soluzione risolve i due precedenti inconvenienti. Infatti, siamo ora in presenza di un solo monitor che implementa `IPCM` senza usare un monitor nidificato al suo interno e ciò elimina l'inconveniente della seconda soluzione. Inoltre, essendo la procedura `send` definita come funzione di monitor, viene eliminato anche l'inconveniente della prima soluzione. Si presenta però un nuovo problema che produce errori analoghi a quelli prodotti dalla prima soluzione. In pratica può accadere che un mittente invochi la funzione `send` per spedire un messaggio di dimensione superiore a 10 caratteri. Quando il `buffer` e` pieno il processo mittente si blocca all'interno della funzione `send` e libera il monitor. Può allora accadere che un secondo processo mittente invochi a sua volta la `send` e, in tal caso, trovando il `buffer` pieno (`cont` == 5) il nuovo mittente si blocca a sua volta sulla condizione `non_pieno` liberando di nuovo il monitor. Successivamente il ricevente può invocare la procedura `receive`, svuotando il `buffer` e risvegiando il primo mittente che riprende a riempirlo. In questo caso, riempito il `buffer` per la seconda volta il primo mittente che non ha ancora completato la `send`, si blocca di nuovo inserendosi nella coda `non_pieno`, ma stavolta dietro al secondo mittente che sarà il prossimo processo ad essere risvegliato dal ricevente e che quindi inzierà ad inserire nel `buffer` i propri caratteri prima che sia terminato l'invio del messaggio da parte del primo mittente. In pratica avremmo di nuovo due messaggi i cui caratteri vengono mescolati all'interno del `buffer` e quindi ricevuti in modo erroneo.

Parte b)

Impostazione

Per descrivere una soluzione corretta possiamo partire dall'ultima vista eliminando però l'inconveniente in essa presente e cioè garantendo che, dal momento in cui un mittente, eseguendo la `send`, inizia ad inserire caratteri nel `buffer` e fino a quando non è terminato l'invio dell'intero messaggio, nessun altro mittenete possa iniziare ad inserire, a sua volta, caratteri nel `buffer`.
Per ottenere tale risultato è sufficiente mantenere aggiornato lo stato del monitor `IPCM` mediante un indicatore (`boolean occupato`) che viene posto al valore `true` quando un mittente inizia l'esecuzione di una `send` e resettato al termine di tale esecuzione. Inoltre, riservando una ulteriore variabile `condition` (`libero`) un mittente, all'inizio di una `send` deve verificare che non sia già stata iniziata un'altra `send` che non è ancora terminata (cioè verificando che il `buffer` non sia occupato da un altro mittente). Se ciò accade, il nuovo mittente deve sospendersi esplicitamente su questa condizione in attesa del termine del trasferimento in atto.

```
monitor IPCM {
   char buffer[5];
   int testa=0, coda=0, cont=0;
   condition non_pieno, non_vuoto;
   boolean occupato=false;
   condition libero;

   public void send (char mes[]){
      int i=0;
      if(occupato) wait(libero); /* è in corso una send non terminata*/
      occupato=true; /* si indica che inzia una send*/
      do  /* ciclo da eseguire fino all'ultimo carattere della stringa*/
         if (cont==5) wait(non_pieno);
         buffer[coda]=mes[i];
         coda = (coda+1)%5;
         cont++;
         if(cont==5||mes[i]=='/0') signal(non_vuoto);
         i++;
      while(mes[i-1]!='/0');
      occupato=false; /* si indica che termina una send*/
      signal(libero); /* una nuova send può iniziare*/

   }

   public void receive(char mes[]) {
      int i=0;
      do
         if (cont==0) wait(non_vuoto);
         mes[i]= buffer[testa];
         testa = (testa+1)%5;
         cont--;
         if(cont==0||mes[i]=='/0') signal(non_pieno);
         i++;
      while(mes[i-1]!='/0');
   }
}
```

La soluzione proposta utilizza la semantica *signal_and_urgent*. La stessa potrebbe però essere facilmente trasformata utilizzando la sematica *signal_and_continue* inserendo la `wait(libero)` presente all'inizio

della `send` non all'interno un `if` ma dentro un `while` e sostituendo la `signal(libero)` finale con una `signalAll(libero)`.