ESERCIZI DI SINCRONIZZAZIONE TRA PROCESSI IN AMBIENTE LOCALE

ESERCIZIO SincrProcAmbLoc-1

In un sistema con processi ad ambiente locale sono presenti i processi P1 (con priorità 1), P2 (con priorità 2), e P3 (con priorità 3), I processi comunicano con le primitive *send(destinatario, &mess)* e *receive(mittente, &buffer)*, che specificano esplicitamente mittente e destinatario (modello *uno-a-uno)*. La primitiva *send* è asincrona; la primitiva *receive* è bloccante.
Al tempo *t* il processo P3 è in esecuzione e la coda pronti contiene (nell'ordine) i processi P2 e P1. Nel canale di comunicazione non vi sono messaggi giacenti.
Lo scheduling del processore avviene con una politica a priorità (va in esecuzione il processo con priorità più elevata).
Si chiede come si modifica lo stato dei processi se si verificano (in alternativa) le seguenti sequenze di eventi:

a) P3 esegue *receive(P1, &buffer);* quindi il processo in esecuzione esegue *receive(P3, &buff);* quindi il processo in esecuzione esegue *send(P3, &mess).*
b) P3 esegue *send(P2, &mess));* quindi il processo in esecuzione esegue *receive(P2, &buffer);* quindi il processo in esecuzione esegue *receive(P3, &buffer).*

SOLUZIONE

	Sequenze di eventi	In Esecuzione	Coda Pronti
a-1	P3 esegue *receive(P1, &buffer)*	P2	P1
a-2	Il processo in esecuzione esegue *receive(P3, &buff)*	P1	∅
a-3	Il processo in esecuzione esegue *send(P3, &mess)*	P3	P1
b-1	P3 esegue *send(P2, &mess)*	P3	P2 → P1
b-2	Il processo in esecuzione esegue *receive(P2, &buffer)*	P2	P1
b-3	Il processo in esecuzione esegue *receive(P3, &buff)*	P2	P1

ESERCIZIO SincrProcAmbLoc-2

In un sistema operativo ad ambiente locale, i processi A1, A2, ... , Am (clienti) e il processo B (servente) comunicano mediante primitive di comunicazione. Per l'invio è disponibile la primitiva *send(destinatario,&messaggio)*, che è asincrona e specifica il nome del processo destinatario. Per la ricezione sono disponibili le primitive bloccanti *receive(mittente, &mess),* che specifica il nome del processo dal quale si vuole ricevere e restituisce il messaggio, e *receive (&mitt, &mess)*, che riceve da un qualsiasi processo e restituisce il messaggio e il nome del processo mittente.
Il generico processo cliente invia il messaggio e si sincronizza con il servente attendendo una risposta. Il processo servente riceve messaggi dai clienti e, per ogni messaggio ricevuto, si sincronizza con il mittente inviando una risposta (protocollo *rendez-vous esteso*).
Completare il codice del generico processo cliente Ai e del processo servente B.

SOLUZIONE

Processo Ai

```
while (true) {
        messaggio = produce_messaggio();
        send(B, &messaggio);
        receive(B,&risp);
        <consuma risp>
}
```

Processo B

```
while (true) {
        receive(&mitt, &mess);
        risposta= produci_risposta();
        send(mitt, &risposta);
}
```

ESERCIZIO SincrProcAmbLoc-3

In un sistema operativo ad ambiente locale, i processi A1, A2, ... , Am (clienti) e i processi B1,B2,…, Bn (serventi) comunicano mediante primitive di comunicazione.
Per l'invio dei messaggi il sistema fornisce le primitive:

- *send(destinatario, &messaggio)* che è asincrona e specifica il nome del processo destinatario.
- *send(mailbox , &messaggio)* che è asincrona e specifica il nome del mailbox al quale si vuole inviare.

Per la ricezione dei messaggi il sistema fornisce le primitive:

- *receive(mittente, &messaggio)* che è bloccante e specifica il nome del processo dal quale si vuole ricevere
- *receive(mailbox, &mitt, &mess)* che è bloccante, specifica il mailbox dal quale si vuole ricevere e restituisce il messaggio ricevuto e il nome del suo mittente.
- *receive(&mitt, &mess)* che è bloccante, riceve dalla porta associata al processo in esecuzione (alla quale tutti i processi possono inviare messaggi) e restituisce il messaggio ricevuto e il nome del suo mittente.

Il generico cliente può chiedere un servizio a un generico servente inviando un messaggio al mailbox e, dopo l'invio, si sincronizza attendendo la risposta del servente che ha erogato il servizio. La richiesta può essere servita da un servente arbitrario, che preleva il messaggio dal mailbox. eroga il servizio che poi invia la risposta al cliente.
Completare il codice del generico processo cliente Ai e del generico processo servente Bj.

SOLUZIONE

Ai:

```
while (true) {
      messaggio = produce_messaggio();
      send(mailbox, &messaggio);
      receive(&serv,&risp);
      <consuma risp>
}
```

Bj:

```
while (true) {
      receive(mailbox, &cliente, &mess);
      <eroga il servizio richiesto>;
      risposta= produci_risposta();
      send(cliente,&risposta);
}
```

ESERCIZIO SincrProcAmbLoc-4

In un sistema operativo ad ambiente locale, i processi P1, P2, ... , Pm (produttori) e i processi C1,C2,…, Cn (consumatori) si scambiano informazione, sotto forma di record di formato fisso, attraverso un vettore circolare di N celle, ciascuna capace di contenere un record.
Poiché l'ambiente non consente la condivisione di dati da parte di produttori e consumatori, il vettore circolare è un dato privato di un ulteriore processo, denominato *BuffServer*, e i produttori e i consumatori richiedono, rispettivamente, il deposito e l'inserzione di record comunicando con *BuffServer* mediante opportune primitive di comunicazione.
Le primitive di comunicazione utilizzate sono le seguenti:

- *send(destinatario, &messaggio)* che è asincrona e specifica il nome del processo destinatario;
- *send(mailbox , &messaggio)* che è asincrona e specifica il nome del mailbox al quale si vuole inviare;
- *receive(mailbox , &mitt, &mess)* che è bloccante, specifica il mailbox dal quale si vuole ricevere e restituisce il messaggio ricevuto e il nome del suo mittente.

Si utilizzano due mailbox, denominati *mailbox_per_deposito* e *mailbox_per_prelievo*. Il generico produttore può richiedere di depositare il contenuto della propria variabile *record* eseguendo la primitiva *send(mailbox_per_deposito, &record),* mentre il generico consumatore può richiedere il prelievo di un record eseguendo la primitiva *send(mailbox_per_prelievo, &vuoto),* dove il valore della variabile *vuoto* non è significativo, e sincronizzandosi successivamente con *BuffServer* mediante la primitiva *receive(BuffServ, &mess*), dove *mess* è la variabile destinata a ricevere il record restituito da *BuffServ.*
Il processo *BuffServer* riceve messaggi dai due mailbox, rispettivamente con le primitive *receive(mailbox_per_deposito, &mitt, &mess)* e *receive(mailbox_per_prelievo, &mitt, &mess)* ed esegue il deposito o il prelievo per conto dei clienti, gestendo opportunamente il vettore circolare.
Il processo *BuffServer* ha due flussi di esecuzione concorrenti, realizzati con i thread *ThreadGestioneProduttori* e *TreadGestioneConsumatori.* Questi thread condividono il *vettore_circolare*, della capacità di N celle e inizialmente vuoto, e i puntatori *testa* e *coda* associati al vettore circolare, entrambi inizializzati al valore 0.
I thread sono realizzati a livello kernel e si sincronizzano con semafori. I semafori utilizzati sono *cella_ disponibile* (valore iniziale N) e *cella_ occupata* (valore iniziale 0).
Per richiedere il deposito nel vettore circolare, il generico produttore esegue il seguente frammento di codice:

```
……..
record = produce_record();
send(mailbox_per_deposito, &record);
……..
```

Analogamente, per richiedere il prelievo dal vettore circolare, il generico consumatore esegue il seguente frammento di codice:

```
……..
send(mailbox_per_prelievo, &vuoto);
receive(BuffServ, &mess);
consuma_mess(mess)
……..
```

Si chiede di specificare il codice dei due thread del processo *BuffServer*.
Si chiede inoltre se è necessaria la mutua esclusione per le operazioni sui puntatori *testa* e *coda* eseguite dai due thread, motivando la risposta.

SOLUZIONE

```
ThreadGestioneProduttori
{
    while true {
        wait (cella_ disponibile);
        //esiste almeno una cella disponibile? Se non esiste si sospende//
        receive(mailbox_per_deposito, &mitt, &mess);
        //riceve la richiesta di un produttore, eventualmente sospendendosi in attesa che venga inviata//
        coda= (coda+1) mod N; vettore_circolare[coda]= mess;
        //deposista il record nel vettore circolare, gestendo il puntatore//
        signal (cella_ occupata);
        /si sincronizza con TreadGestioneConsumatori//
        }
}


TreadGestioneConsumatori
{
    while true {
        wait (cella_ occupata);
        //esiste almeno un record giacente? Se non esiste si sospende//
        receive(mailbox_per_prelievo, &mitt, &mess);
        //riceve la richiesta di un consumatore, eventualmente sospendendosi in attesa che venga inviata//
        send(mitt, &vettore_circolare[testa]; testa= (testa+1) mod N;
        //invia al richiedente il primo record del buffer, quindi aggiorna il puntatore//
        signal (cella_ disponibile);
        //si sincronizza con TreadGestioneProduttori//
        }
}
```

Per le operazioni sui puntatori *testa* e *coda* non è necessaria la mutua esclusione, perchè il puntatore *testa* è utilizzato solo da *TreadGestioneConsumatori*, mentre il puntatore *coda* è utilizzato solo da *TreadGestioneProduttori*.