**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:

1. P3 esegue *receive(P1, &buffer);* quindi il processo in esecuzione esegue *receive(P3, &buff);* quindi il processo in esecuzione esegue *send(P3, &mess).*
2. 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 (Filosofi a cena)**

Nel problema dei “filosofi a cena” N filosofi si riuniscono per la cena in un ristorante cinese, occupando un tavolo circolare, apparecchiato con un piatto per ogni filosofo e un bastoncino interposto fra ogni coppia di piatti adiacenti. Per mangiare, il filosofo di indice *i (0 ≤ i ≤ N-1*) deve avere a disposizione i due bastoncini posti rispettivamente alla sua sinistra e alla sua destra (cioè quelli di indice *i* e *(i+1) mod N)*, entrando pertanto in competizione con i due filosofi che siedono ai suoi lati.

Ogni filosofo alterna periodi in cui pensa a periodi in cui vuole mangiare. Quando decide di mangiare richiede con richiesta multipla i due bastoncini che gli sono necessari, applicando quindi una strategia di prevenzione statica dello stallo, e si sospende se non li ottiene entrambi. Dopo aver mangiato torna a pensare e rilascia i due bastoncini.

La riattivazione di un filosofo in attesa dei bastoncini può avvenire quando uno dei filosofi adiacenti rilascia quelli in suo possesso. La riattivazione è subordinata alla condizione che sia disponibile, oltre al bastoncino appena rilasciato dal filosofo che siede alla sua destra (o alla sua sinistra), anche quello interposto tra lui medesimo e il filosofo che siede alla sua sinistra (o alla sua destra).

In questo problema i filosofi sono processi ad ambiente locale e i bastoncini sono risorse. Si utilizza inoltre un processo *Server,* e le variabili che descrivono lo stato dei processi e delle risorse sono dati privati di questo processo. Per le interazioni tra il generico filosofo e il *Server* sono disponibili le seguenti primitive:

* *send(proc, &buffer)*: asincrona; invia al processo di indice *proc* il messaggio contenuto in *buffer*

*(*questa primitiva è utilizzabile anche per l’invio a un processo servente che riceva dalla propria porta con la primitiva *receive(,&mitt, &buffer),* che non specifica l’origine del messaggio)

* *send(mailbox, &buffer)*: asincrona; invia a *maibox* il messaggio contenuto in *buffer,*
* *receive(proc, &buffer)*: bloccante; riceve un messaggio dal processo di indice *proc* e lo trasferisce in *buffer*
* *receive(&mitt, &buffer)*: bloccante, senza specifica dell’origine: riceve un messaggio dalla propria porta e lo trasferisce in *buffer*; inoltre riceve l’indice del mittente e lo restituisce nella variabile *mitt*.

Lo stato dei filosofi e dei bastoncini è definito implicitamente dal vettore *stato*, la cui componente *stato[i]* è associata al filosofo di indice *i*. I possibili valori di *stato[i]* sono *pensa, HaFame* e *mangia*, con i seguenti significati:.

* *stato[i]= pensa:* il filosofo i non possiede i bastoncini di indice *i* e (*i+ 1) mod N*
* *stato[i]= mangia:* il filosofo i possiede i bastoncini di indice *i* e *(i+ 1) mod N*
* *stato[i]= HaFame:* il filosofo i è in attesa di ottenere i bastoncini di indice *i* e (*i+ 1) mod N*

Inotre la transizione *mangia🡪 pensa* di *stato[i]* corrisponde al rilascio dei bastoncini di indice i *e (i+ 1) mod N* .

Il generico filosofo esegue il seguente programma:

**while** true {

BufferUscita= “voglio\_mangiare” ; send(server, &BufferUscita)

*// richiede il bastoncino alla sua destra e quello alla sua sinistra//*

receive(server, &BufferIngresso);

*// la ricezione del messaggio conferma l’assegazione //*

<mangia per una durata di tempo casuale>

BufferUscita= “voglio\_pensare”; send(server, &BufferUscita)

*// rilascia i bastoncini senza attendere la risposta //*

<pensa per una durata di tempo casuale >

}

Si chiede di completare il programma del processo *Server*, inserendo in particolare le primitive di comunicazione e le espressioni che condizionano l’assegnazione dei bastoncini e la riattivazione dei filosofi sospesi.

**SOLUZIONE**

**Server**

{

**while** true {

receive(&mitt, &mess)

**if** mess= “voglio mangiare” {

stato[mitt]= HaFame;

**if** stato[(mitt-1) mod N]<> mangia and stato[(mitt+1) mod N]<> mangia {

stato[mitt]= mangia; OutBuff= “buon appetito”;

send(mitt, &OutBuff);

}

}

**else** {

*// ricevuto il messaggio “voglio pensare” //*

stato[mitt]= “pensa”;

**if** (stato[(mitt-1) mod N]== HaFame) **and** (stato[(mitt-2) mod N]<> mangia) {

stato[(mitt-1) mod N]= mangia; OutBuff= “buon appetito”;

send((mitt-1) mod N, &OutBuff));

*// il filosofo di indice (mitt-1) mod N ottiene i bastoncini di indice*

*(mitt-1) mod N e di indice (mitt-2) mod N, e pertanto viene riattivato //*

}

**if** (stato[(mitt+1) mod N]== HaFame) **and** (stato[(mitt+2) mod N]<> mangia) {

stato[(mitt+1) mod N]= mangia; OutBuff= “buon appetito”;

send((mitt+1)mod N, &OutBuff));

*// il filosofo di indice (mitt+1) mod N ottiene l bastoncini di indice*

*(mitt+1) mod N e di indice (mitt+2) mod N e pertanto viene riattivato //*

}

}

}

**ESERCIZIO SincrProcAmbLoc-5 (Barbiere dormiglione)**

Un negozio di un barbiere è dotato di un’unica una poltrona per il servizio dei clienti e di un numero illimitato di sedie per l’attesa. La poltrona è una risorsa, che può essere occupata da un cliente per il tempo necessario all’esecuzione del taglio dei capelli, oppure dal barbiere per attendere (dormendo) l’arrivo di clienti da servire. Le sedie sono risorse utilizzate dai clienti per attendere il proprio turno, dopo essere entrati nel negozio. Il barbiere e i clienti sono processi ad ambiente locale, che interagiscono con lo scambio di messaggi.

All’apertura del negozio il barbiere si addormenta sulla poltrona e sarà svegliato del primo cliente che entrerà nel negozio.

Il generico cliente ha il seguente comportamento:

* entra nel negozio e, con un messaggio, chiede al barbiere il taglio dei capelli; quindi occupa una sedia in attesa della risposta con la quale barbiere lo invita ad occupare la poltrona;
* si siede sulla poltrona e attende che venga eseguito il taglio dei capelli;
* al termine viene riattivato da un messaggio del barbiere che comunica la tariffa. Quindi esegue il pagamento e ne dà conferma al barbiere con un messaggio;
* infine esce dal negozio.

Il barbiere ha il seguente comportamento ciclico:

* occupa la poltrona e attende la richiesta da parte di un cliente. Se non la riceve immediatamente, si addormenta;
* quando riceve la richiesta si risveglia (se dormiva) e risponde al cliente invitandolo ad occupare la poltrona; quindi esegue il taglio dei capelli mentre il cliente passa in attesa;
* al termine risveglia il cliente comunicandogli la tariffa da pagare e attende dal cliente la conferma del pagamento.

Il problema viene risolto con le seguenti primitive di comunicazione:

* *send(destinatario, &messaggio).* Primitiva asincrona per la comunicazione diretta, che invia al processo di indice *destinatario* il messaggio contenuto nella variabile *messaggio;*
* *send(PortaDestinatario, &messaggio).* Primitiva asincrona per la comunicazione indiretta,che scrive il messaggio contenuto nella variabile *messaggio* nella porta del processo destinatario (mailbox), denominata *PortaDestinatario;*
* *receive(mittente, &messaggio),* Primitiva bloccante per la comunicazione diretta, che restituisce nella variabile *messaggio* un messaggio inviatodal processo di indice *mittente;*
* *receive(&mittente, &messaggio),* Primitiva bloccante per la comunicazione indiretta, che legge un messaggio dalla porta del processo che la esegue e restituisce l’indice del processo mittente nella variabile *mittente* e il messaggio nella variabile *messaggio.*

Si chiede di completare il programma del barbiere e il frammento di codice che controlla i clienti che entrano nel negozio, inserendo le opportune primitive di comunicazione.

**SOLUZIONE**

Barbiere

{

receive(&cliente, &RichiestaTaglio);

*// attende il primo cliente; se non vi sono clienti in attesa si addormenta; //*

*// il contenuto del messaggio è implicito e non viene verificato //*

**while**(true) {

messaggio=”si accomodi sulla poltrona”

send(cliente, &messaggio); */ il cliente prende posto sulla poltrona //*

<esegue il taglio dei capelli> *// nel frattempo il cliente si addormenta //*

tariffa= TariffaTaglio;  *// sveglia il cliente, comunicandogli la tariffa //*

send(cliente, &tariffa);

*// attende la conferma del pagamento, che riceverà nella variabile ConfermaPagamento. Si suppone che la // conferma arrivi in un tempo finito e che non sia necessario verificare il contenuto del messaggio //*

receive(cliente, &ConfermaPagamento);

receive(&cliente, &RichiestaTaglio); *// attende il cliente successivo; eventualmente si addormenta*

}

}

Cliente

*//Frammento di codice//*

< entra nel negozio >

mess= “RichiestaTaglio”; send(PortaBarbiere, &mess);

*// attende dal barbiere l’invito a occupare la poltrona; eventualmente si addormenta //*

receive(barbiere, &chiamata);  *// il contenuto del messaggio è implicito e non viene verificato //*

<raggiunge la poltrona e la occupa>

*// il barbiere inizia il taglio dei capelli; il cliente attende (dormendo) che il barbiere termini e comunichi la tariffa //*

receive(barbiere, &tariffa);

paga(tariffa);

ConfermaPagamento =”OK”;

send(barbiere, &ConfermaPagamento);

<esce dal negozio>

*// fine del frammento di codice //*

**ESERCIZIO SincrProcAmbLoc-6**

In un parcheggio per auto della capacità di *N* posti si accede da un unico ingresso e si esce da un’unica uscita, entrambi controllati da sbarre. Le auto (indicate come A1, A2, ... An) sono processi. Il parcheggio è gestito da un ulteriore processo, denominato *GestoreParcheggio.*

I processi sono ad ambiente locale e interagiscono con meccanismi di comunicazione. Sono disponibili le seguenti primitive:

* *send(destinatario, &messaggio).* Primitiva asincrona per la comunicazione diretta, che specifica l’indice del processo *destinatario* e il puntatore al buffer contenente il messaggio da inviare. E’ utilizzabile anche per l’invio a un processo servente, nel qual caso il messaggio viene scritto nella porta del processo destinatario, che a sua volta lo riceve con la primitiva *receive(&mittente, &messaggio);*
* *send(maibox, &messaggio).* Primitiva asincrona per la comunicazione indiretta, che trasferisce il messaggio nel *maibox* specificato;
* *receive(mittente, &messaggio),* Primitiva bloccante per la comunicazione diretta, che restituisce il *messaggio* un messaggio inviatodal processo di indice *mittente;*
* *receive(&mittente, &messaggio),* Primitiva bloccante, che legge un messaggio dalla porta del processo (servente) che la esegue e restituisce l’indice del mittente e il messaggio;
* *receive(mailbox, &mitt, &mess).* Primitiva bloccante, che riceve dal *mailbox* specificato e restituisce il messaggio e il nome del mittente.

Le auto richiedono di entrare nel parcheggio inviando una richiesta al processo *GestoreParcheggio* e ottengono l’accesso quando ricevono una risposta attraverso il mailbox *MailboxAutorizzazione,* eventualmente transitando per lo stato di attesa fino all’arrivo dell’autorizzazione. Dopo essere entrate occupando un posto, possono uscire facendone richiesta con un messaggio al *GestoreParcheggio.*

Per la gestione, il processo *GestoreParcheggio* dispone delle seguenti variabili private :

* *PostiDisponibili*, inizializzato al valore *N* , che esprime il numero di posti disponibili nel parcheggio;
* *AutoInAttesa*, inizializzato al valore *0*, che esprime il numero di auto che hanno fatto richiesta di entrare e sono in attesa dell’autorizzazione.

Si chiede di completare, con le opportune primitive di comunicazione, il frammento di codice eseguito dalla generica auto che accede al parcheggio, e lo schema di programma del processo *GestoreParcheggio,* dal quale si può desumere la politica di gestione.

**SOLUZIONE**

*// frammento di codice eseguito dalla generica auto //*

…………………………………

<l’auto arriva all’ingresso >

messaggio= igresso;

send(GestorePacheggio, &messaggio);

receive**(**(MailboxAutorizzazione, &mitt, &OK)

*// richiede di entrare ed eventualmente attende la disponibilità di un posto//*

<è disponibile un posto: la sbarra si alza e l’auto entra nel parcheggio>

<e l’auto sceglie e occupa il posto >

<alla fine, l’auto libera il posto e si dirige all’uscita >

messaggio= RichiestaUscita;

send(GestorePacheggio, &messaggio);

receive((GestoreParcheggio; OK)

<la sbarra si alza e l’auto esce dal parcheggio>

**………………………………**

Gestore Parcheggio

{

**while** (true) {

receive(&richiesta, &mittente)

*// legge una richiesta dalla propria porta //*

**if** richiesta= ingresso {

**if** PostiDisponibli>0 {

PostiDisponibili --;

send(MailboxAutorizzazione, &OK)

*// autorizza immediatamente l’ingresso //*

}

**else** AutoInAttesa ++

*// la risposta viene ritardata fino alla disp[onibilità di un posto, provocando l’attesa dell’auto richiedente //*

}

**else** {

*//richiesta= uscita //*

**if** AutoInAttesa>0 {

AutoInAttesa --;

send(MailboxAutorizzazione, OK)

*// l’aito richiedente viene riattivata //*

}

**else** PostiDisponibili ++;

send(mittente, OK}

*// alza la sbarra e fa uscire l’auto //*

}