Il modello a scambio di messaggio

Ciascun processo evolve in un proprio ambiente che non può essere modificato direttamente da altri processi. Quindi non esiste memoria condivisa e le risorse sono tutte private.

Pertanto:

- non esiste la **competizione** per l'uso delle risorse non esistendo risorse comuni
- la **cooperazione** si realizza mediante lo scambio diretto di messaggi per mezzo di primitive che il S.O. deve rendere disponibili

Il naturale supporto fisico al modello sono i sistemi di elaborazione con architettura distribuita.

I processi servitori

Il numero limitato di risorse disponibili impone comunque che alcune risorse debbano essere ***logicamente comuni*** nel senso che debbano poter essere utilizzate indirettamente da più processi pur non essendo private a nessuno di essi.

Ciò e ottenuto mediante ***processi servitori*** a cui le risorse logicamente comuni sono private.

Un processo servitore

- riceve messaggi di richiesta
- opera sulla risorsa
- fornisce eventuali risposte

I processi servitori

Un processo servitore può servire le richieste che riceve memorizzandole in una opportuna struttura dati e servendole secondo specifiche strategie di scelta. In tal caso costituisce anche il gestore della risorsa logicamente comune.

Peraltro è possibile scindere in processi servitori distinti la politica di scelta dei servizi da effettuare e l'esecuzione dei servizi stessi e quindi eseguire in distinti contesti la scelta dell'ordine di esecuzione dei servizi e servizi stessi. In tal caso avremo un processo servitore di tipo gestore e un processo servitore che opera sulla risorsa gestita

Le primitive per lo scambio di messaggio

Le primitive per lo scambio di messaggio appartengono a due distinte tipologie.

- ***primitive di tipo send***: per l'invio di un messaggio da parte di un processo mittente
- ***primitive di tipo receive***: per la ricezione di un messaggio da parte di un processo destinatario

Si distinguono per:

- modalità con cui si designano la provenienza e la destinazione
- tipo di sincronizzazione dei processi comunicanti

Le primitive per lo scambio di messaggio

Sono parametri di una send:

- il parametro che individua la destinazione del messaggio
- il parametro che individua il messaggio da trasmettere.

Sono parametri di una receive:

- il parametro che individua la provenienza del messaggio
- il parametro che individua il messaggio ricevuto.

Modalità di designazione della destinazione e della provenienza

La destinazione e la provenienza possono essere indicate:

- esplicitando nella send il pid del destinatario e nella receive il pid del mittente (***comunicazione diretta simmetrica***)
- esplicitando nella send il pid del destinatario mentre nella receive il parametro che specifica la provenienza è di uscita nel qual caso il destinatario conosce il pid del mittente con la ricezione del messaggio (***comunicazione diretta asimmetrica***)
- facendo riferimento ad una mailbox a cui la send invia il messaggio e da cui la receive lo preleva (***comunicazione indiretta***)

Tipi di sincronizzazione

send asincrona: il processo mittente non attende la consegna del messaggio al destinatario

send bloccante: il processo mittente attende la trasmissione del messaggio sulla rete

send sincrona: il processo mittente attende la consegna del messaggio al destinatario

receive bloccante: il processo ricevente si blocca se il messaggio non è disponibile

receive non bloccante: il processo ricevente comunque prosegue e gli viene segnalato se il messaggio è stato consegnato o meno

Send asincrona

Vantaggi

- semplicità implementativa
- parallelismo

Svantaggi

- il destinatario non può associare la ricezione del messaggio con uno stato del mittente
- difficoltà d'uso in quanto meccanismo di basso livello
- necessità che il S.O. disponga di una capacità di accumulo dei messaggi non consegnati immediatamente (buffers)
- eventuale blocco nascosto del mittente per carenza di buffer o, in alternativa, possibilità di perdita dei messaggi

Send bloccante

Vantaggi

- semplicità implementativa
- parallelismo

Svantaggi

- il destinatario non può associare la ricezione del messaggio con uno stato del mittente
- difficoltà d'uso in quanto meccanismo di basso livello
- necessità che il S.O. della macchina del destinatario disponga di una capacità di accumulo dei messaggi non consegnati immediatamente (buffers)
- possibilità di perdita dei messaggi

Send sincrona

Vantaggi

- alla ricezione del messaggio il destinatario conosce lo stato del mittente
- nessuna necessità di capacità di accumulo di messaggi da parte del S.O.
- meccanismo di più alto livello
- possibilità di implementazione con send asincrona
- nessuna possibilità di perdita di messaggi

Svantaggio

- riduzione del parallelismo

Receive bloccante

Vantaggio

- attesa passiva del processo che attende un messaggio

Svantaggio

- impossibilità di polling di più mailbox e conseguentemente ricezione di messaggi di diverso tipo attraverso una stessa mailbox o in caso di comunicazione diretta asimmetrica

Receive non bloccante

Vantaggio

- possibilità di polling di più mailbox e quindi di discriminare i messaggi a seconda del loro tipo

Svantaggio

- attesa attiva del processo che attende un messaggio da una specifica mailbox o nel caso di comunicazione diretta (simmetrica o asimmetrica)

Realizzazione di una mailbox per primitive asincrone

Un possibile ***tipo mailbox*** per primitive asincrone è:

record
contatore: *integer*;
primo: *coda;*
end;

ove:

- *primo* consente di implementare la coda dei messaggi (buffer) nella mailbox o dei processi in attesa di messaggi dalla mailbox
- *contatore* esprime il numero di elementi in coda e la loro natura (ad es. processi<0; messaggi>0)

Realizzazione di una send asincrona

var *buca_post, buffer_liberi*: *mailbox*; **var** *flag*:*boolean*
procedure *send_asincrona*(*buca_post, mess, flag*);
begin
  flag:= *true*;
  if *buca_post.contatore*<0 **then**
    begin
    <copia *mess* nel buffer di ricezione *mess1* del primo processo *proc* in attesa di un messaggio>
    (*mess1* è un parametro della receive bloccante che ha messo in attesa *proc*)
    <poni *proc* nella coda dei ready>
    end;
  else if *buffer_liberi.contatore*>0 **then**
    < copia *mess* in un buffer e collegalo a *buca_post.primo* >
    else *flag*:=*false*;
end;

Realizzazione di una receive bloccante

```
procedure receive_bloccante(buca_post, mess);
begin
  if buca_post.contatore>0 then
      <preleva un messaggio da buca_post.primo,  copialo in mess e
      collega il buffer liberato a buffer_liberi.primo >
  else
    begin
      <poni il processo running in attesa in buca_post.primo>;
      context_switch;
    end;
end;
```

Realizzazione di una receive non bloccante

```
var buca_post, buffer_liberi: mailbox, flag: boolean;
procedure receive_non_bloccante(buca_post, mess, flag);
begin
  if buca_post.contatore>0 then
    begin
      flag:=true;
      <preleva un messaggio da buca_post.primo,  copialo in mess e
      collega il buffer liberato a buffer_liberi.primo >
    end;
  else flag:=false;
end;
```

Realizzazione di una send sincrona mediante primitive asincrone

procedure *send_sincrona*(*buca_post*, *mess*)
begin
  send_asincrona (*buca1_post*, *mess1*)
  (mess1 è un messaggio di pronto ad inviare)
  receive_bloccante (*buca2_post*, *mess2*)
  (mess2 è un messaggio di pronto a ricevere)
  send_asincrona (*buca_post*, *mess*)
end;

Realizzazione di una receive bloccante per comunicazione con send sincrona

procedure *receive_bloccante_sincrona*(*buca_post*, *mess*)
begin
  receive_bloccante(*buca1_post*, *mess1*)
  (mess1 è un messaggio di pronto ad inviare)
  send_asincrona (*buca2_post*, *mess2*)
  (mess2 è un messaggio di pronto a ricevere)
  receive_bloccante(*buca_post*, *mess*)
end;

Realizzazione di una mailbox per primitive sincrone

Un possibile ***tipo mailbox*** per primitive sincrone è:

record
contatore: *integer*;
primo: *coda;*
end;

ove:

- *primo* consente di implementare la coda dei processi mittenti o destinatari in attesa di comunicare
- *contatore* esprime il numero di elementi in coda e la loro natura (ad es. processi destinatari<0; processi mittenti>0)

Realizzazione di una send sincrona

var *buca_post*: *mailbox*;
procedure *send_sincrona*(*buca_post*, *mess*);
begin
  if *buca_post.contatore*<0 **then**
    begin
    <copia *mess* nel buffer di ricezione *mess1* del primo processo *proc* in attesa di un messaggio>
    (*mess1* è un parametro della receive bloccante che ha messo in attesa *proc*)
    <poni *proc* nella coda dei ready>
    end;
  else begin
    < metti il processo running in attesa in *buca_post.primo* >
    context_switch; **end**;
end;

Realizzazione di una receive bloccante per send sincrona

procedure *receive_bloccante_sincrona*(*buca_post*, *mess*);
begin
if *buca_post.contatore*>0 **then**
begin
<copia *mess1* del primo processo *proc* in attesa di trasmettere un messaggio nel buffer di ricezione *mess* >
(*mess1* è un parametro della send sincrona che ha messo in attesa *proc*)
<poni *proc* nella coda dei ready>
end;
else begin
<poni il processo running in attesa in *buca_post.primo*>; *context_switch*; **end**;
end;
end;

Realizzazione di una receive non bloccante per send sincrona

var *buca_post*: *mailbox*, *flag*: *boolean*;
procedure *receive_non_bloccante_sincrona* (*buca_post*, *mess*, *flag*);
begin
if *buca_post.contatore*>0 **then**
begin
flag:=*true*;
<copia *mess1* del primo processo *proc* in attesa di trasmettere un messaggio nel buffer di ricezione *mess* >
(*mess1* è un parametro della send sincrona che ha messo in attesa *proc*)
<poni *proc* nella coda dei ready>
end;
else *flag*:=*false***;**
end;