Message Passing

Sistemi Operativi

Antonino Staiano
Email: antonino.staiano@uniparthenope.it

Message passing

- Il message passing è un modo con il quale i processi interagiscono tra loro

Process P_i

```
...
send (Pj,<message>);
...
```

Process P_j

```
...
receive (Pi, msg_area);
...
```

- I processi possono esistere nello stesso computer o in computer diversi connessi da una rete
- Message passing usato in diverse applicazioni:
  - Paradigma client-server
  - Backbone di protocolli di comunicazione di livelli più alti
  - Programmi paralleli e distribuiti

Produttore-Consumatore con scambio di messaggi

```
begin
  Parbegin
    var buffer : . . . ;                    var message_area : . . . ;
    repeat                                  repeat
      { Produce in buffer }                   receive (Pi, message_area);
      send (Pj, buffer);                      { Consume from message_area }
      { Remainder of the cycle }              { Remainder of the cycle }
    forever;                                forever;
  Parend;
  end.
```

Process P_i *Process* P_j

Message passing (cont.)

- Due importanti questioni nel message passing
  - Nomi dei processi
    - I nomi possono essere indicati in modo esplicito o dedotti dal kernel in qualche modo
  - Consegna dei messaggi
    - Se il mittente deve essere bloccato fino alla consegna
    - Quale è l'ordine in cui i messaggi sono consegnati ad un processo destinatario
    - Come gestire condizioni eccezionali

Naming diretto ed indiretto

- Nel **naming diretto**, i processi mittente e destinatario si citano a vicenda

```
send (<destination_process>, <message_length>, <message_address>);
receive (<source_process>, <message_area>);
```

  - Nel naming simmetrico, entrambi i processi specificano il nome dell'altro
  - Nel naming asimmetrico, il destinatario non specifica il nome del processo da cui intende ricevere un messaggio
    - il kernel gli invia un messaggio inviatogli da qualche processo
- Nel **naming indiretto**, i processi non menzionano i nomi l'uno dell'altro nelle istruzioni **`send`** e **`receive`**

Invii bloccanti e non bloccanti

- Una `send` bloccante blocca il processo mittente fino alla consegna del messaggio al processo destinatario
  - Scambio di messaggi sincrono
  - Semplifica la progettazione dei processi concorrenti
- Una chiamata `send` non bloccante permette al mittente di continuare le proprie operazioni dopo la chiamata
  - Scambio dei messaggi asincrona
  - Incrementa la concorrenza tra mittente e destinatario
- In ambo i casi, `receive` è bloccante
- Il kernel esegue la consegna dei messaggi pendenti bufferizzati

Implementazione dello scambio dei messaggi

- Un processo P_i invia un messaggio a P_j usando una `send` non bloccante
  - il kernel costruisce un IMCB (interprocess message control block) per memorizzare tutte le informazioni necessarie per consegnare il messaggio
  - Allo IMCB è allocato un buffer nel kernel
    - Quando P_j invoca receive, il kernel copia il messaggio dallo IMCB nell'area fornita da P_j
- Il campo puntatore di un IMCB è usato per formare una lista di IMCB che semplifica la consegna dei messaggi

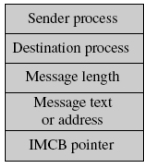


Figure 9.3 Interprocess message control block (IMCB).

Implementazione dello scambio dei messaggi

- Nel naming simmetrico, è usata una lista separata per ogni coppia di processi P_i-P_j che comunica
  - Quando un processo P_i esegue una `receive`, la lista IMCB per la coppia P_i-P_j è usata per consegnare il messaggio
- Nel naming asimmetrico, è usata una singola lista per ogni processo
  - Quando un processo esegue `receive`, il primo IMCB nella sua lista è elaborato per consegnare il messaggio

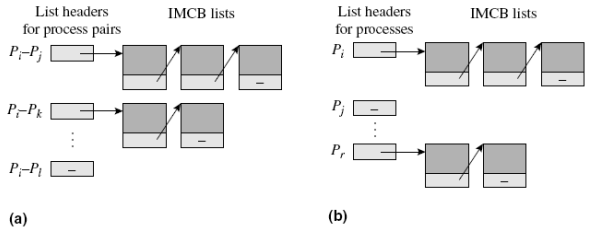


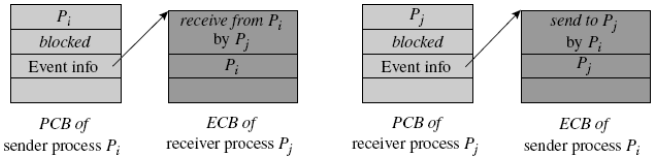
Figure 9.4 Lists of IMCBs for blocking *send*s in (a) symmetric naming; (b) asymmetric naming.

Consegna dei messaggi Interprocesso

- Quando un processo P_i invia un messaggio a P_j
  - Il kernel consegna il messaggio a P_j immediatamente, se P_j è bloccato su una **`receive`**
    - Dopo la consegna, il kernel cambia lo stato di P_j a ready
  - Il kernel si dispone a consegnare il messaggio successivamente, se P_j non ha ancora invocato **`receive`**
- Ricordiamo che il kernel usa un Event Control Block (ECB) per annotare le azioni da intraprendere quando si verifica un evento
  - ECB contiene
    - Descrizione evento
    - Id processo in attesa dell'evento
    - Puntatore a un ECB per la creazione delle liste di ECB

Consegna dei Messaggi Interprocesso

- Uso degli ECB per l'implementazione dello scambio dei messaggi
  - Naming simmetrico e **`send`** bloccante



  - Quando P_i invoca `send`
    - il kernel controlla se esiste un ECB per tale chiamata, cioè, se P_j ha fatto una chiamata a `receive` ed era in attesa che P_i inviasse un messaggio
      - Se questo non è il caso, il kernel sa che `receive` si verificherà in futuro, quindi crea un ECB per l'evento *«ricevi da P_i per P_j»* e specifica P_i come il processo coinvolto nell'evento
      - P_i è messo allo stato blocked e l'indirizzo dell'ECB è messo nel campo evento del suo PCB
    - P_j fa una chiamata a `receive` prima che P_i invochi `send`
      - Viene creato un ECB per *«invia a P_j da P_i»*. L'id di P_j è messo nell'ECB indicando che lo stato di P_j sarà modificato quando si verifica l'evento `send`

Azioni del kernel nello scambio dei messaggi (naming simmetrico e send bloccante)

At *send* to P_j by P_i:

Step	Description
S_1	*Create an IMCB and initialize its fields;*
S_2	**If** *an ECB for a 'send to P_j by P_i' event exists*
S_3	**then**
	(a) Deliver the message to P_j;
	(b) Activate P_j;
	(c) Destroy the ECB and the IMCB;
	(d) Return to P_i;
S_4	**else**
	(a) Create an ECB for a 'receive from P_i by P_j' event and put id of P_i as the process awaiting the event;
	(b) Change the state of P_i to blocked and put the ECB address in P_i's PCB;
	(c) Add the IMCB to P_j's IMCB list;

At *receive* from P_i by P_j:

Step	Description
R_1	**If** *a matching ECB for a 'receive from P_i by P_j' event exists*
R_2	**then**
	(a) Deliver the message from appropriate IMCB in P_j's list;
	(b) Activate P_i;
	(c) Destroy the ECB and the IMCB;
	(d) Return to P_j;
R_3	**else**
	(a) Create an ECB for a 'send to P_j by P_i' event and put id of P_j as the process awaiting the event;
	(b) Change the state of P_j to blocked and put the ECB address in P_j's PCB;

Passaggio dei Messaggi in Unix

- Tre supporti alla comunicazione interprocesso
  - Pipe:
    - Funzione per trasferimento dati
    - Pipe senza nome può essere usata solo da processi che appartengono allo stesso albero dei processi (hanno un antenato in comune)
  - Code di messaggi (*message queue*):
    - Usate dai processi nel dominio del sistema Unix
    - I permessi di accesso indicano quali processi possono inviare o ricevere messaggi
  - Socket: un'estremità di un percorso di comunicazione
    - Può essere usata per impostare dei percorsi di comunicazione tra processi nel dominio Unix e all'interno di alcuni domini Internet

Message Passing in Unix (cont.)

- Una caratteristica comune: i processi possono comunicare senza conoscere le rispettive identità

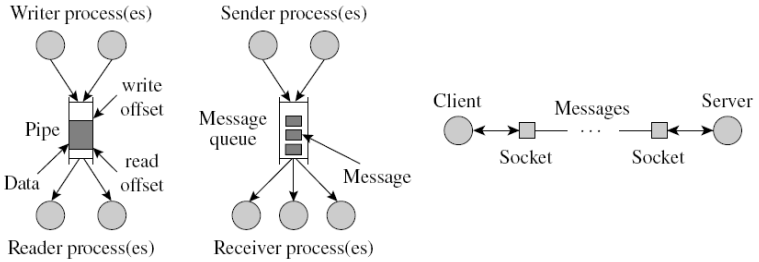


Figure 9.10 Interprocess communication in Unix: (a) pipe; (b) message queue; (c) socket.

Message Passing in Unix: Pipe

- Meccanismo FIFO per il trasferimento di dati tra processi chiamati lettori e scrittori
  - I dati messi in una pipe possono essere letti solo una volta
    - Rimossi dalla pipe quando sono letti da un processo
- Due tipi di pipe: con nome e senza nome
  - Create attraverso la chiamata di sistema pipe
  - Un pipe con nome ha un nome di file associato nel filesystem
- Come un file, ma la dimensione è limitata e il kernel la tratta come una coda

Message Passing in Unix: Socket

- Una socket è una lato di un percorso di comunicazione
  - Può essere usata per la comunicazione interprocesso nel dominio Unix e nel dominio Internet
- Il server può impostare i percorsi di comunicazione con molti client simultaneamente
  - Tipicamente, dopo una chiamata connect, il server crea un nuovo processo con fork per gestire la nuova connessione
    - Lascia le socket originali create dal processo server libera di accettare più connessioni
- Naming indiretto: usa gli indirizzi invece degli id di processo