UNIVERSITÀ DEGLI STUDI DI NAPOLI
PARTHENOPE

Sistemi Operativi

Message Passing

LEZIONE 13

prof. Antonino Staiano

Corso di Laurea in Informatica – Università di Napoli Parthenope

antonino.staiano@uniparthenope.it

Scambio di Messaggi

- Quando i processi interagiscono tra loro devono essere soddisfatti due requisiti fondamentali
  - Sincronizzazione
  - Comunicazione
    - Scambiando informazioni
- Lo scambio di messaggi è un approccio che fornisce ambo le funzioni
  - Particolarmente adatto per i sistemi distribuiti, multiprocessore con memoria condivisa e monoprocessore

Motivazioni

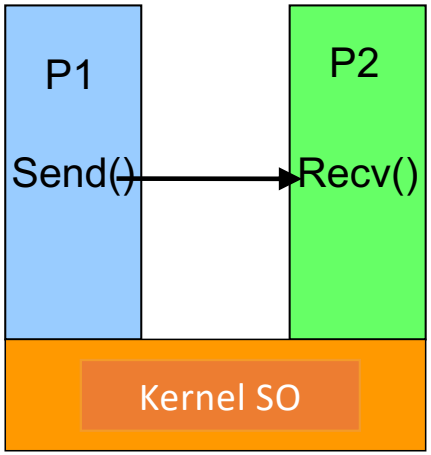
- Lock, semafori, monitor funzionano se lo spazio di indirizzamento è condiviso
  - Thread nello stesso processo
  - Processi che condividono uno spazio di indirizzamento
- Fin qui abbiamo assunto che i processi/thread comunicano via dati condivisi (contatore, buffer produttori/consumatori, …)
- Come facciamo a sincronizzare e comunicare tra processi con spazi di indirizzamento diversi?
  - Inter-process communication (IPC)
- E' possibile avere un insieme di primitive che funzionano in tutti i casi?
  - SO per macchina singola, stesso SO più machine, più machine e più SO, SO distribuiti

Spazio di indirizzamento non condiviso

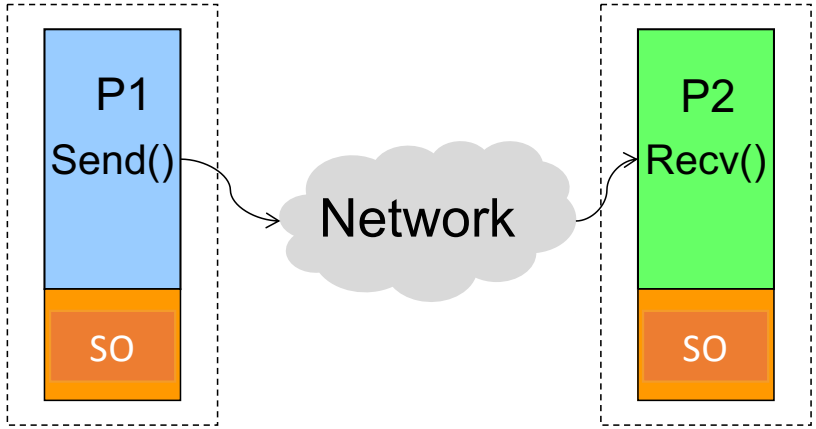
- Nessuna necessità di primitive per la mutua esclusione
  - I processi non possono manipolare direttamente gli stessi dati
- Comunicazione tramite invio e ricezione esplicite dei messaggi
  - Scambio messaggi
- La sincronizzazione è implicita nello scambio dei messaggi
  - Nessuna mutua esclusione esplicita
  - Ordinamento di eventi via invio e ricezione di messaggi
- Più portabile in diversi ambienti
  - Non priva di complessità da risolvere
- Tipicamente, in genere, la comunicazione è realizzata tramite un invio e una ricezione corrispondente

Invio di un Messaggio

In un computer



Attraverso una rete



- P1 può inviare a P2 e P2 può inviare a P1

Scambio di messaggi

- La funzionalità è fornita mediante una coppia di primitive

```
send (destination, message)
receive (source, message)
```

- Un processo invia le informazioni in forma di messaggio verso un altro processo designato come destinazione
- Un processo riceve informazioni eseguendo la primitiva `receive` che indica la sorgente e il messaggio

Sincronizzazione

La comunicazione di un messaggio tra due processi comporta la loro sincronizzazione

Il ricevitore non può ricevere un messaggio fino a che esso non è inviato da un altro processo

Quando è eseguita la primitiva receive da un processo ci sono due opzioni:

Se non c'è messaggio in attesa il processo è bloccato fino a che arriva un messaggio o il processo continua l'esecuzione, abbandonando il tentativo di ricezione

Se un messaggio è stato precedentemente inviato, lo stesso è ricevuto e l'esecuzione continua

Send e Receive Bloccanti

- Entrambi il mittente ed il destinatario sono bloccati fintantoché il messaggio non è consegnato
- Noto anche come rendezvous
- Permette una sincronizzazione stretta tra processi

Send non bloccante

send non bloccante, receive bloccante

- Il mittente continua ma il ricevitore è bloccato fino a che il messaggio richiesto arriva
- E' la combinazione più utile
- Invia uno o più messaggi a più destinazioni il più velocemente possibile
- Esempio: un processo server il cui compito è fornire un servizio o una risorsa ad altri processi

send non bloccante, receive non bloccante

- A nessuna delle controparti è richiesto di attendere

Indirizzamento

- Gli schemi per specificare i processi nelle primitive send e receive ricadono in due categorie
  - Indirizzamento diretto
  - Indirizzamento Indiretto

Indirizzamento Diretto

- La primitiva send include un identificatore specifico del processo destinatario
- La primitiva receive può essere gestita in due modi:
  - Richiedere che il processo indichi esplicitamente un processo mittente (naming simmetrico)
    - Efficace per processi concorrenti cooperanti
  - Indirizzamento implicito (o naming asimmetrico)
    - Il parametro *sorgente* della primitiva receive conterrà l'id del mittente quando l'operazione di ricezione è eseguita

Indirizzamento Indiretto

I messaggi sono inviati ad una struttura dati condivisa costituita da code che li memorizzano temporamenamente

Le code sono riferite come *mailbox*

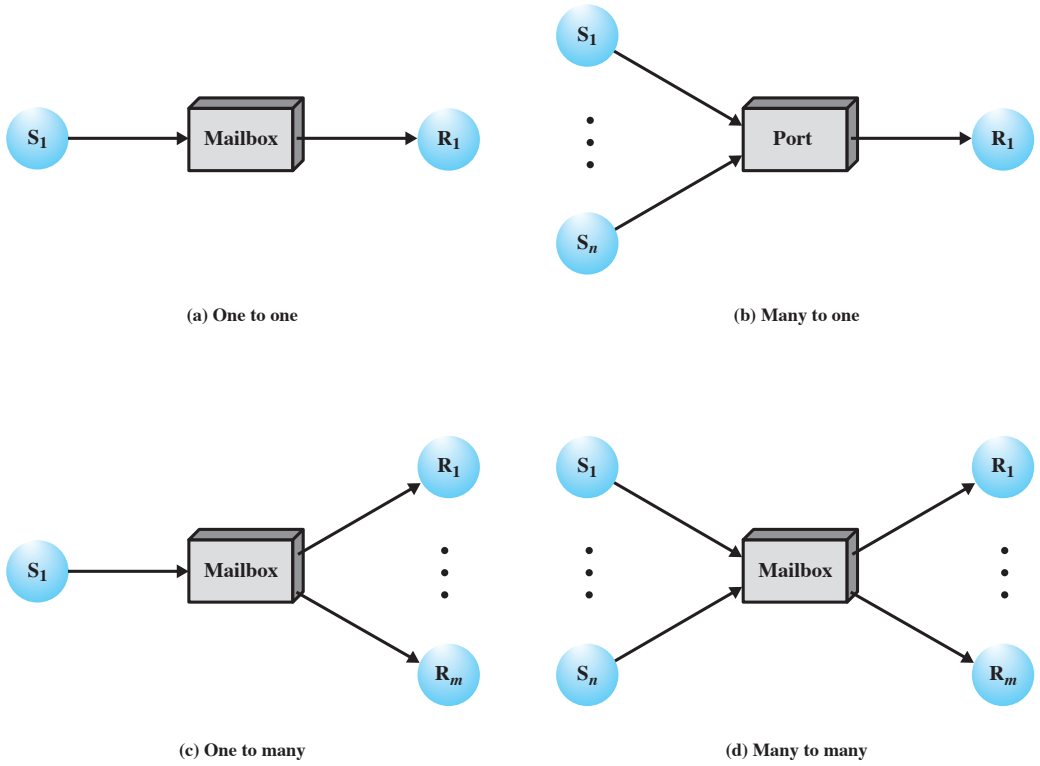
Un processo invia un messaggio alla mailbox e l'altro processo preleva il messaggio dalla mailbox

Permette una maggiore flessibilità nella gestione dei messaggi

Mailbox

- Repository per messaggi interprocesso
  - Ha un nome unico
  - Il proprietario di solito è il processo che l'ha creata
- Indirizzamento indiretto
  - Un qualsiasi processo che conosce il nome di una mailbox può inviargli messaggi
- Il kernel può fornire un insieme fissato di nomi di mailbox, o può permettere ai processi utente di assegnare i nomi
  - Livelli variabili di confidenzialità

Indirizzamento Indiretto con Mailbox

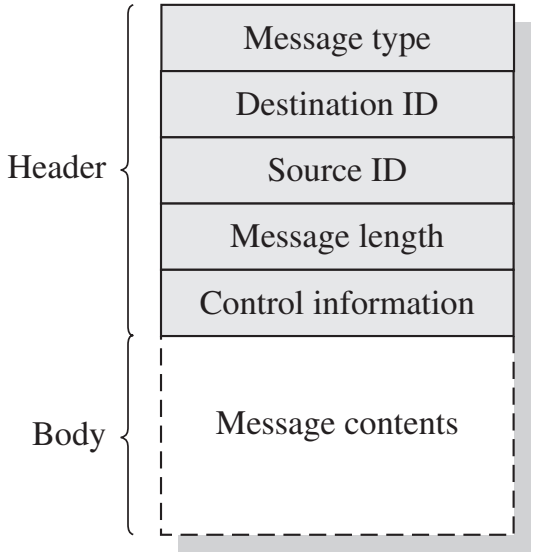


Associazione e Proprietà delle Mailbox

- Associazione Statica
  - Porta: creata ed assegnata in modo permanente al processo che la crea
  - Analogamente per le associazioni 1:1
- Associazione Dinamica
  - Molti mittenti
  - Implica uso di primitive tipo *connect* e *disconnect*
- Proprietà
  - Porta: ricevente
    - Quando il processo termina, la porta è eliminata
  - Mailbox
    - SO offre servizio di creazione
      - Proprietà del processo che le crea -> cancellate quando il processo termina
      - Proprietà del SO -> comando speciale per cancellarle

Formato generale dei Messaggi

- Il formato dei messaggi può essere
  - A lunghezza fissa
    - Per minimizzare overhead di eleaborazione e memorizzazione
    - Poco flessibile
  - A lunghezza variabile



Mutua Esclusione usando i Messaggi

```
int n /* numbero di processi */

void P(int i)
{
     message msg;
     while (true) {
        receive (box, msg);
        /* sezione critica */;
        send (box, msg);
        /* resto del codice */;
     }
}


create mailbox (box);
send (box, null);
parbegin (P(1), P(2),..., P(n))
```

I processi concorrenti condividono una mailbox box

Nota: assumiamo send non bloccante e receive bloccante

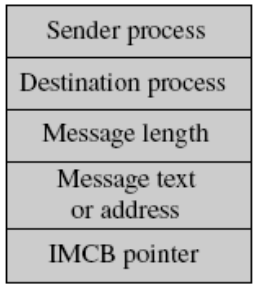
I messaggi costituiscono un token da inviare da processo a processo per andare in sezione critica

Produttore-Consumatore con Scambio messaggi

```
int dimensione = /* capacità del buffer */
null = /* messaggio vuoto */ ;

void produttore(){
  message pmsg;
  while (true) {
     receive (possoProdurre,pmsg);
     pmsg = produci();
     send (possoConsumare,pmsg);
  }
}

void consumatore(){
  message cmsg;
  while (true) {
     receive (possoConsumare,cmsg);
     consuma (cmsg);
     send (possoProdurre,null);
  }
}

create_mailbox (possoProdurre);
create_mailbox (possoConsumare);
for (int i = 1;i<= dimensione;i++)
       send (possoProdurre,null);
parbegin (producer,consumer)
```

Implementazione dello scambio dei messaggi

- Quando 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 nell'IMCB è usato per formare una lista di IMCB che semplifica la consegna dei messaggi



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_j esegue una `receive`, è usata la lista di IMCB per la coppia P_i-P_j 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

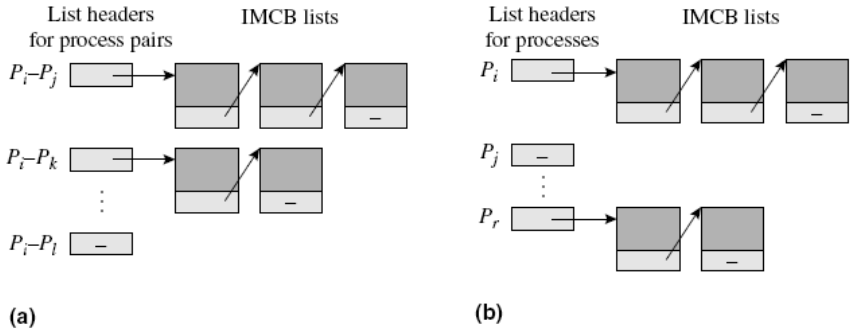


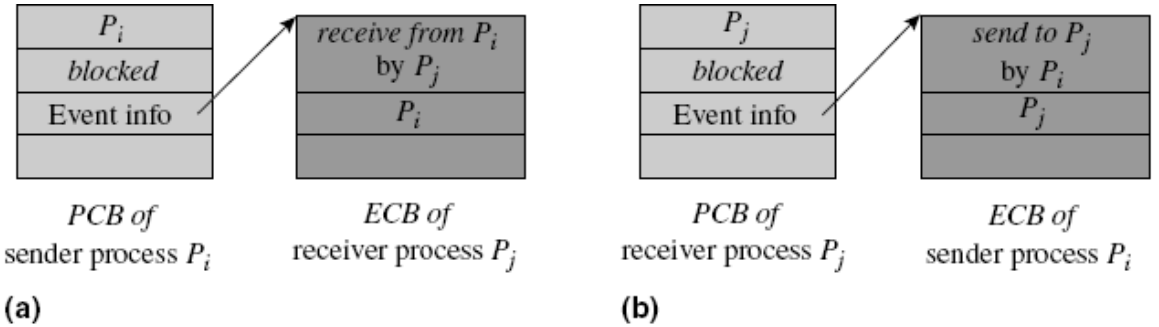
Figure 9.4 Lists of IMCBs for blocking *send*s in (a) symmetric naming; (b) asymmetric naming.

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
    a) 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
    b) 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`

Strategie di Accodamento

- La più semplice è del tipo FIFO
  - Inappropriata se qualche messaggio è più urgente di altri

- Alternative
  - Permettere di specificare una priorità per i messaggi sulla base del tipo di messaggio o come indicato dal mittente
  - Permettere al destinatario di ispezionare la coda dei messaggi e selezionare il messaggio successivo da ricevere

Message Passing 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

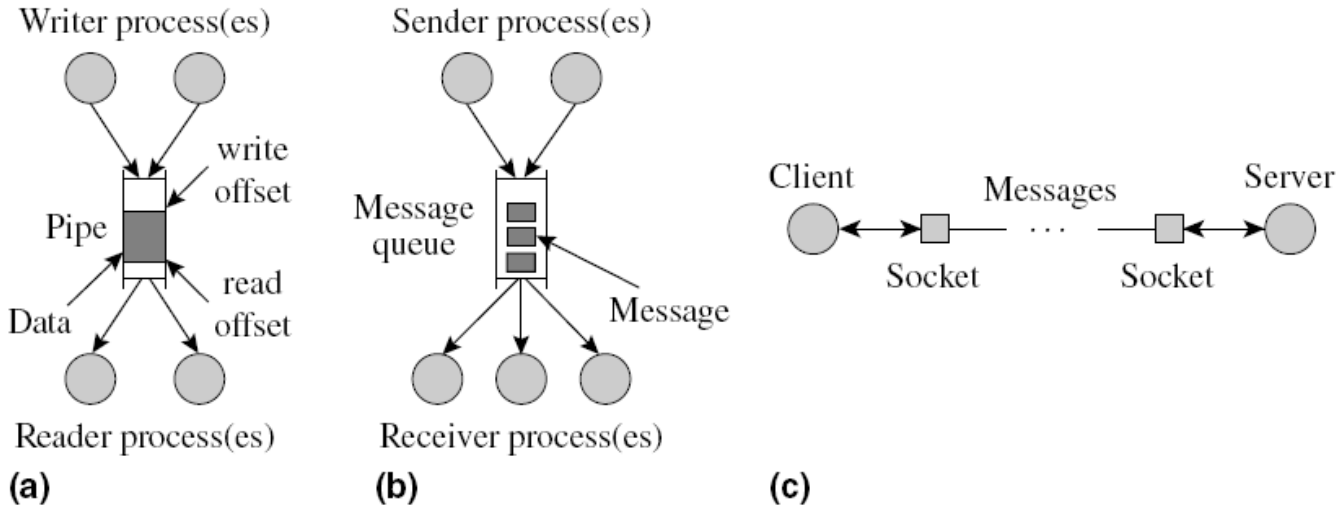


Figure 9.10 Interprocess communication in Unix: (a) pipe; (b) message queue; (c) socket.

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