

Indice

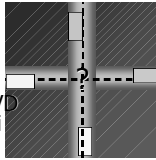
- Deadlock
- Primitive modello a scambio di messaggi
- Costrutti linguistici e topologie di comunicazione

Deadlock

- In molte applicazioni un processo necessita di usare alcune risorse in modo esclusivo
- Si ha deadlock quando due processi detengono ciascuno una risorsa che serve all'altro per proseguire
- Quando due o più processi entrano nello stato di deadlock rimangono bloccati per sempre

Esempi di deadlock: Alto livello

- Azienda specializzata nel produrre mappe demografiche
- Le informazioni risiedono su un DVD contenente censimenti ed altri dati
- Elaborazione su rete locale
- La stampa avviene su un plotter a colori A0
- Due processi (diversi) A e B :
 - A chiede l'accesso all' unita' DVD → lo ottiene
 - B chiede l'accesso al plotter → lo ottiene
 - A chiede l'accesso al plotter → si blocca
 - B chiede l'accesso all'unita' DVD → si blocca



Esempio di deadlock: Programmazione

- Due processi A, B
- Due risorse R1, R2 protette da semafori S1, S2 ($S1=0, S2=1$)

Processo A

```
...  
Wait(S2)  
<SC - R1>  
Wait(S2)  
<SC - R1+R2>  
Signal(S2)  
Signal(S1)
```

Processo B

```
...  
Wait(S2)  
<SC - R2>  
Wait(S1)  
<SC - R2+R1>  
Signal(S1)  
Signal(S2)
```

Dipende dalla velocità di esecuzione relativa!

Definizione formale

- Un insieme S di processi è in deadlock se ogni processo di S è in attesa di un evento che solo un altro processo $\in S$ può causare
 - Tutti sono in attesa → Nessuno può provocare l'evento che risveglia un altro processo
 - Generalmente l'evento è il rilascio di una risorsa posseduta da un'altro membro dell'insieme
 - Il numero dei processi e di risorse coinvolte non ha importanza

Condizioni per il deadlock

Coffman et al. (1971) hanno dimostrato che è necessario che si verifichino quattro condizioni perché vi sia deadlock:

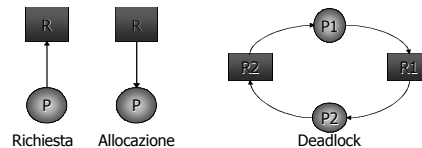
1. **Mutua Esclusione** . Ogni risorsa risulta assegnata esattamente ad un processo oppure e' disponibile.
2. **Prendi e Aspetta** . I processi che detengono risorse assegnategli in precedenza possono richiederne di nuove.
3. **Assenza di Preilascio** . Le risorse precedentemente assegnate non possono essere revocate forzatamente.
4. **Attesa circolare** . Ci deve essere una lista circolare di almeno due processi ognuno dei quali e' in attesa di una risorsa posseduta dal processo che segue nella lista.

Strategie per evitare il deadlock

- Ignorare il problema
 - Algoritmo dello struzzo ...
 - Anche UNIX soffre di deadlock
ES: tabella dei processi
Tabella piena → fork() fallisce e riprova dopo un T Casuale
100 entry, 10 proc ciascuno crea 12 sotto-processi
Quando ogni proc iniziale ha creato 9 proc → deadlock
 - Meglio accettare deadlock occasionali piuttosto che dover utilizzare una sola risorsa alla volta
 - Compromesso utilità correttezza
eliminare deadlock e' costoso

Strategie per evitare il deadlock

- Individuare il deadlock e risolverlo
 - Periodicamente si controlla il grafo di allocazione delle risorse
 - Se c'è un ciclo (deadlock) si terminano processi a caso nel ciclo fino ad interromperlo



Strategie per evitare il deadlock

- Prevenzione dinamica: allocazione attenta delle risorse
 - Vincolare i processi in modo che il deadlock risulti strutturalmente impossibile
 - Es: algoritmo del banchiere
Il banchiere (SO) non soddisfa le richieste di credito (risorse) dei clienti (processi) che possono portare a stati di deadlock
- Prevenzione Strutturale: negare una delle 4 condizioni precedenti

Strumenti di Programmazione (Lez. Prec.)

- Indipendenti dal linguaggio
Es: Threads (flussi di controllo indipendenti in un processo)
- Specifici del linguaggio
Es: Java (Monitor e Semafori)
- Ambiente Globale
 - Monitor (Brinch Hansen, Hoare 1973)
 - Semafori e Primitive di sincronizzazione (Dijkstra, 1965)
 - Altri strumenti
- Scambio di Messaggi ←
 - Send(m), Receive(m)

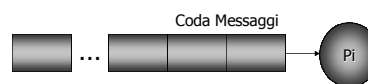
Struttura di un messaggio

- Un msg si può pensare costituito così:


```
Type messaggio
    Origine:  ...;
    Destinazione: ...;
    Contenuto: ...;
end
```
- Contenuto può essere vuoto: sincronizzazione

Primitive del modello a scambio di messaggi

- Nel caso più semplice si può supporre che
- \forall processo \exists una coda per i messaggi in arrivo
 - Primitive di comunicazione utilizzate:
 - Send (m): inserisce il messaggio nella coda del destinatario
 - Receive (m): preleva il messaggio dalla coda del processo corrente



Scambio di messaggi

- Con lo scambio di messaggi si realizza:
 - Comunicazione:
Un processo attraverso la ricezione di un messaggio ottiene valori da un processo mittente
 - Sincronizzazione:
Un messaggio può essere ricevuto solo dopo essere stato trasmesso (Questa relazione di causa-effetto vincola l'ordine in cui i due eventi possono avvenire)
- La mutua esclusione non è più un problema: nel modello ad ambiente locale ogni risorsa è privata viene acceduta serialmente (tipo monitor)

Costrutti Linguistici e Topologie di comunicazione

- Classificazione:
 - A. Designazione dei processi sorgente e destinatario
 - Diretta o esplicita: (direct naming)
 - Simmetrica
 - Asimmetrica
 - Indiretta o Globale:
 - Mailbox e Porte (global naming e port naming)
 - B. Tipo di sincronizzazione
 - Sincrona
 - Asincrona
- Caratteristiche ortogonali
 - Le soluzioni proposte per A. e B. sono indipendenti

Costrutti Linguistici e Topologie di comunicazione

- Classificazione:
 - A. Designazione dei processi sorgente e destinatario
 - Diretta o esplicita: (direct naming)
 - Simmetrica
 - Asimmetrica
 - Indiretta o Globale:
 - Mailbox e Porte (global naming e port naming)
 - B. Tipo di sincronizzazione
 - Sincrona
 - Asincrona
- Caratteristiche ortogonali
 - Le soluzioni proposte per A. e B. sono indipendenti

Primitive con Designazione Esplicita

Send <expression_list> **to** <dest_designator>

- Valutazione <expression_list> → contenuto del messaggio
- <dest_designator> indica la destinazione del messaggio

Receive <var_list> **from** <source_designator>

- <var_list> riceve i valori contenuti nel messaggio
- <source_designator> indica l'origine del messaggio
- L'istanza del messaggio viene distrutta

Direct Naming (Designazione Esplicita)

- La coppia (<dest_designator>, <source_designator>) definisce un canale di comunicazione
- Schema simmetrico: i processi si nominano esplicitamente (direct naming) e reciprocamente
- Esempio:
 - P1: Send msg to P2
P1 invia un messaggio che può essere ricevuto solo da P2
 - P2: Receive msg from P1
P2 riceve un messaggio che è stato inviato da P1

Direct Naming (Designazione Esplicita)

- Semplice da implementare e utilizzare:
Controllo selettivo degli intervalli di tempo di ricezione
- Utilizzato nei modelli del tipo pipeline:
P1|P2|...|Pn
 - Collezione di processi concorrenti in cui l'output di un processo è l'input di un altro
 - Sistema concepito in termini di flusso di informazione

Direct Naming - Esempio

Elaborazione batch mediante scambio di messaggi
Esempio di sistema a paradigma Pipeline

```

Process reader          Process executer          Process printer
var card: CardImage;   var card: CardImage;   var line: LineImage;
Loop                   var line: LineImage;   Loop
<read card from cardreader>
Send card to executer      Receive card from cardreader      <print line on line printer>
end
end;                   <process card and gen line>      end
                        Send line to printer      end;
                        end;
                        end;
    
```

Esempio piu attuale: shell

Flusso dell'Informazione

Produttore-Consumatore con Scambio Messaggi Esplicito

```

Produttore          Consumatore
Begin <source_des>   Begin
repeat               for i=0 to N
  Receive (Consumatore, Trigger)
  <Produzione Msg>   Send (Produttore, Trigger)
  Send (Consumatore, Msg)
forever              repeat
end                   Receive (Produttore, Msg)
                    <Consumazione Msg>
                    Send (Produttore, Trigger)
                    forever
end                  end
                    end
    
```

Costrutti Linguistici e Topologie di comunicazione

- Classificazione:
 - A. Designazione dei processi sorgente e destinatario
 - Diretta o esplicita: (direct naming)
 - Simmetrica
 - Asimmetrica
 - Indiretta o Globale:
 - Mailbox e Porte (global naming e port naming)
 - B. Tipo di sincronizzazione
 - Sincrona
 - Asincrona
- Caratteristiche ortogonali
 - Le soluzioni proposte per A. e B. sono indipendenti

Direct Naming Schema asimmetrico

- Il mittente nomina esplicitamente il destinatario
 - Il destinatario **non** esprime il nome del processo con cui desidera comunicare
- Notazione:
- ```

Send <msg> to Pi Oppure Send (Pi, Msg)
Pid := Receive <msg> Receive (Pid, Msg)

```
- Pid riceve, nel destinatario, l'identità del mittente per una eventuale risposta
  - Questo schema facilita l'implementazione del paradigma client-server (Es: browser e server web)

## Modello Client-Server

- Uso di un server processo come
  - Gestore di risorse non possedute dal client: spooler
  - Fornitore di servizi non erogabili dal client: google
- Client e server possono risiedere su macchine remote

```

Pi (Client) Pj (Server)
...
Send <richiesta> Receive <richiesta>
Receive <risultato> <Esecuzione>
... Send <risultato>

```

Accesso alla risorsa  
Esecuzione del servizio

## Modello Client-Server

- Schema da-molti-a-uno
  - Es: DBMS, browser e server web (primo aspetto)
    - I client specificano il destinatario delle loro richieste
    - Il server è pronto a ricevere richieste da un qualunque client
- Schema da-uno-a-molti o da-molti-a-molti
  - Es: browser e server web (2° aspetto della topologia)
    - I client inviano richieste ad un qualunque server tra un set di server equivalenti
    - Richiede designazione indiretta o globale (MailBox)

## Modello Client-Server e Naming

- Direct naming NON é adatto per realizzare sistemi client server
- Many-to-one: la Receive sul server dovrebbe poter ricevere messaggi trasmessi da un qualsiasi cliente
  - Con designazione esplicita simmetrica occorrerebbe almeno una Receive per cliente
- Many-to-many: la Send di un client dovrebbe poter produrre messaggi ricevibile da uno qualsiasi dei server
  - Con designazione esplicita simmetrica occorrerebbe almeno una Send per server

## Costrutti Linguistici e Topologie di comunicazione

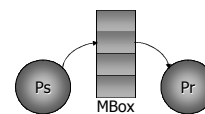
- Classificazione:
  - A. Designazione dei processi sorgente e destinatario
    - Diretta o esplicita: (direct naming)
      - Simmetrica
      - Asimmetrica
    - ➔ ● Indiretta o Globale:
      - Mailbox e Porte (global naming e port naming)
  - B. Tipo di sincronizzazione
    - Sincrona
    - Asincrona
- Caratteristiche ortogonali
  - Le soluzioni proposte per A. e B. sono indipendenti

## Designazione Globale

- Una MailBox puo' essere <dest\_designator> o <source\_designator> nelle istruzioni di Send e Receive di un qualunque processo
- I messaggi inviati ad una specifica MailBox possono essere ricevuti da un qualsiasi processo che effettui una Receive designando tale MailBox

## Designazione Globale

- Notazione
  - Send msg to A\_mbox
  - PID := Receive message from A\_mbox
  - oppure
  - Send (A\_mbox, msg)
  - Receive** (A\_mbox, msg)
- I processi possono selezionare i tipi di messaggio che desiderano ricevere effettuando Receive sulle mailbox opportune

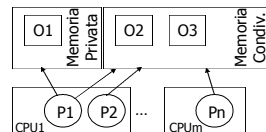


## Uso delle MailBox

- Consente in modo immediato di programmare interazioni client-server anche nel caso da-molti-a-molti
- I client eseguono una Send sulla mailbox associata al servizio, i server una Receive
- Analogia con il caso della mailbox in ambiente a memoria comune in cui non e' specificata l'identita' del particolare server ...

## Modello ad ambiente globale (Lezione Precedente)

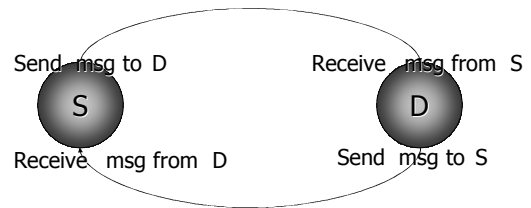
- Astrazione per Sistemi Multiprogrammati
- Costituiti da uno o più processori che hanno accesso ad una memoria comune
- Interazioni in memoria condivisa



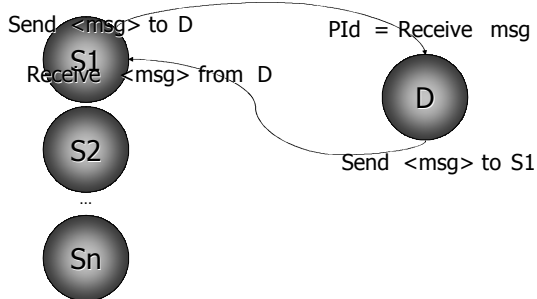
## Uso delle Mailbox

- L'implementazione di una mailbox in ambiente distribuito presenta problemi di natura realizzativa
- Il supporto run-time del linguaggio deve garantire che:
  - Un messaggio di richiesta indirizzato ad una mailbox sia inviato a tutti i processi che possono eseguire la Receive su di essa
  - Non appena il messaggio e' ricevuto da un processo, esso non e' disponibile per gli altri

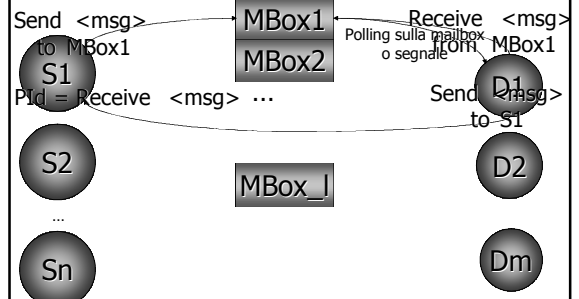
## Direct Naming Simmetrico: Topologia



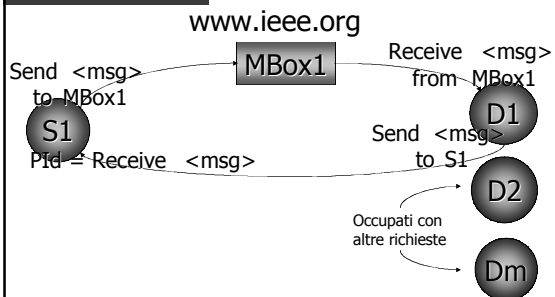
## Direct Naming Asimmetrico: Topologia



## Global Naming: Topologia



## Global Naming: Topologia



## Costrutti Linguistici e Topologie di comunicazione

- Classificazione:
  - A. Designazione dei processi sorgente e destinatario
    - Diretta o esplicita: (direct naming)
      - Simmetrica
      - Asimmetrica
    - Indiretta o Globale
      - Mailbox e Porte (global naming e port naming)
  - B. Tipo di sincronizzazione
    - Sincrona
    - Asincrona
- Caratteristiche ortogonali
  - Le soluzioni proposte per A. e B. sono indipendenti

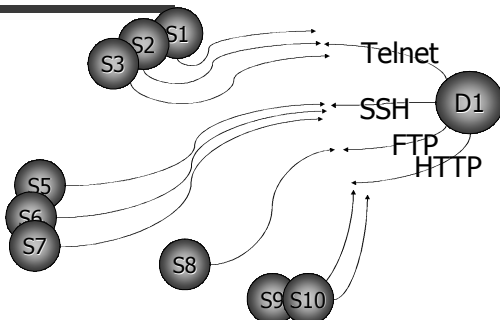
## Porte

- Sono mailbox il cui nome può comparire solamente come <source\_designator> in uno statement di Receive
- Realizzazione più semplice delle mailbox: tutte le Receive che indicano una porta compaiono in un solo processo (mapping 1:1 porta:processo ricevente)
- Soluzione al problema "da-molti-a-uno" ma non a quello "da-molti-a-molti"

## Porte

- Un processo può selezionare i messaggi che desidera ricevere attraverso l'uso di porte distinte
- Se un processo effettua una Receive su una sola porta, lo schema di designazione è logicamente equivalente ad un direct naming asimmetrico
  - Nel direct naming i client possono inviare richieste diverse al server che deve selezionare cosa fare
  - Con le porte, richieste diverse giungono a porte diverse: per aggiungere un servizio è sufficiente utilizzare una nuova porta

## Global Naming: Topologia



## Naming

| Naming                            | Comunicazione             |
|-----------------------------------|---------------------------|
| Direct (simmetrico) (asimmetrico) | One to One<br>Many to One |
| Global                            | Many to Many              |
| Port                              | Many to One               |

- Global naming è il caso più generale.
  - Difficile da realizzare
- Gli altri schemi:
  - limitano i tipi di interazione direttamente programmabili
  - sono più semplici da implementare

## Naming

- Una ulteriore classificazione ortogonale è la designazione del canale
- La designazione dei canali può avvenire:
  - staticamente, a tempo di compilazione
  - dinamicamente, a tempo di esecuzione

## Naming

- Naming Statico
  - Obbliga un programma a comunicare attraverso canali noti a tempo di compilazione
  - Ne limita le capacità di sopravvivenza in un ambiente dinamico
  - Il potenziale accesso di un programma ad un canale deve **permanentemente**
- Naming Dinamico
  - Lo schema statico di base di designazione canali viene arricchito mediante variabili per la designazione di Sorgente o Destinazione

## Costrutti Linguistici e Topologie di comunicazione

- Classificazione:
  - A. Designazione dei processi sorgente e destinatario
    - Diretta o esplicita: (direct naming)
      - Simmetrica
      - Asimmetrica
    - Indiretta o Globale:
      - Mailbox e Porte (global naming e port naming)
  - B. Tipo di sincronizzazione ←
- Caratteristiche ortogonali
  - Le soluzioni proposte per A. e B. sono indipendenti

## Classificazione in base al Tipo di Sincronizzazione

- Send Sincrona ("rendez-vous" semplice)
- Send Asincrona
- Send Tipo RPC ("rendez-vous" esteso)
  
- Receive Sincrona
- Receive Asincrona e con Interrogazione dello stato di un canale

## Send Sincrona

- "Rendez-vous" Semplice
- Mittente si blocca in attesa che il messaggio sia stato ricevuto
- Un messaggio ricevuto contiene informazioni corrispondenti allo stato attuale del mittente
- Semplifica scrittura e verifica dei programmi

## Send Sincrona

- L'invio di un messaggio costituisce un punto di sincronizzazione per mittente e destinatario
- Il trasferimento di informazioni avviene quando entrambi i processi sono pronti a comunicare
- L'interazione viene definita come scambio di messaggi sincrono
- Ai processi sono associati canali senza memoria Uno per ogni tipo di messaggio che il processo può ricevere

## Send Asincrona

- Il Mittente continua l'esecuzione dopo invio messaggio
- Il messaggio ricevuto contiene informazioni che NON ASSOCIABILI allo stato attuale del mittente (difficoltà di verifica dei programmi)
- L'interazione viene definita come scambio di messaggi asincrono

## Send Asincrona

- Per la memorizzazione dei messaggi il supporto del linguaggio deve mettere a disposizione:
  - Caso direct naming:
    - una coda in ingresso ad ogni processo
  - Caso global naming:
    - una coda in ingresso ad ogni porta/mailbox

## Send Asincrona

- Richiede un buffer di capacita' illimitata
- Si puo' ovviare modificandone la semantica (Send Ibrida)
  - Un processo mittente si blocca qualora la coda dei messaggi sia piena
  - La primitiva send solleva un'eccezione che viene notificata al processo mittente

## Send di tipo RPC

- "Rendez-vous" esteso
- Il mittente rimane in attesa fino a che il destinatario non ha terminato di svolgere la procedura richiesta
- Analogia semantica (spesso sintattica) con la chiamata di procedura:
  - Un processo cliente "chiama" la procedura eseguita da un processo server su una macchina potenzialmente remota
  - Il nome della procedura remota identifica un processo nel caso di direct naming o un servizio nel caso di port o mailbox naming
- Programmi facilmente verificabili grazie alla localizzazione dei vincoli di sincronizzazione
- Orientata al modello Client-Server

## Receive Sincrona

- Normalmente e' bloccante se non vi sono messaggi sul canale
- Punto di sincronizzazione per il processo ricevente
- Problema: si desidera ricevere solo alcuni messaggi ritardando l'elaborazione di altri
  - Esempio: Processi gestori di risorse: ricezione di messaggi compatibili con lo stato delle risorse

## Receive asincrona con interrogazione stato canale

- Soluzione:
  - Specificare piu' canali di input per ogni processo, ciascuno dedicato a messaggi di tipo diverso
  - Deve essere possibile specificare su quali canali attendere, in base allo stato interno della risorsa
- Si ricorre ad una primitiva che:
  - verifica lo stato del canale
  - restituisce indicazione di messaggio presente o canale vuoto (**receive** non bloccante)
- Cio' consente ad un processo di selezionare l'insieme dei canali da cui prelevare un messaggio
- Inconveniente: per l'attesa di messaggi da specifici canali occorre Busy Wait

## Chiamata di Procedura Remota (RPC)

- Consente di esprimere a piu' alto livello e in maniera piu' sintetica le interazioni di tipo Client-Server
- Lato Client
  - Call service (<in\_params>, <out\_params>)
    - Service e' il nome di un canale:
      - Caso designazione diretta → Service indica il processo server
      - Caso designazione indiretta (porte o mailbox) → Service indica il tipo di servizio
    - La Call puo' essere tradotta in una send seguita immediatamente da una receive; il cliente quindi non si puo' "dimenticare" di attendere la risposta
- Lato Server
  - Come procedura chiamata separatamente
  - Come statement collocato in un punto qualunque del processo

## RPC lato server come Procedura

- Specifica del lato server come procedura

```
Remote procedure service (in <in_params>, out <out_params>)
<body>
End
```
- La procedura remota viene dichiarata come una procedura in un linguaggio sequenziale

## RPC lato server come Procedura

- Implementata come un processo servitore che attende la ricezione di un messaggio, esegue <body> e trasmette un messaggio di risposta
- Può essere implementata:
  - come singolo processo: esegue richieste una alla volta sequenzialmente
  - con la creazione di un nuovo processo per ogni chiamata: le varie istanze sono eseguite concorrentemente potranno eventualmente doversi sincronizzare tra loro

## RPC lato server come statement

- Specifica lato server come statement
  - La procedura remota è uno statement e come tale può essere collocato in un pto qualunque del processo server
- ```
accept service  
    (in <in_params>, out <out_params>) → body
```

RPC lato server come statement

- L'esecuzione della accept sospende il server fino all'arrivo di un messaggio corrispondente alla call del servizio
- L'esecuzione del corpo può fare uso dei valori dei parametri e di tutte le variabili accessibili dallo scope dello statement
- Al termine viene trasmesso il messaggio di risposta al processo chiamante, dopo che il processo server continua la propria esecuzione
- Ad ogni servizio è associata una coda distinta, generalmente FIFO

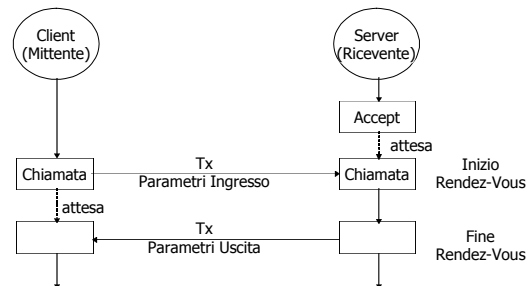
Uso della accept

- RPC viene chiamata extended rendez-vous: client e server si incontrano per la durata della esecuzione del corpo della accept per poi proseguire separatamente
- Vantaggi:
 - Server può fornire più tipi di servizi (accept diverse)
 - Server può decidere quando servire le call dei client
 - Server può selezionare quali tipi di call servire
 - Le accept possono essere alternate o innestate
 - Vi possono essere più accept di chiamate allo stesso servizio con diverso <body> (ad esempio per l'inizializzazione)

Uso della accept

- Accept viene spesso combinata con comunicazioni selettive per consentire ad un servitore di attendere e selezionare una tra diverse richieste di servizio
- Accept diverse per lo stesso servizio fanno sì che ad una richiesta possano corrispondere azioni diverse in funzione dello stato del processo server
 - Netta distinzione rispetto alla definizione di procedura

Schema di comunicazione RPC



Uso della RPC

- Lo schema di comunicazione realizzato dal meccanismo della chiamata a procedura remota e' asimmetrico e da-molti-a-uno
- L'istruzione accept consente di considerare un processo come un modulo che incapsula un insieme di funzioni chiamabili dall'esterno ed eseguibili una alla volta

Uso della RPC

- L'accoppiamento tra una chiamata priva di parametri e una accept priva di <body> rappresenta la trasmissione ed il relativo riconoscimento di un segnale di sincronizzazione
- Una chiamata con soli parametri di ingresso ed una accept priva di corpo definiscono invece un rendez vous stretto