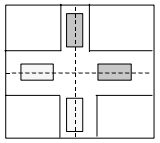
Indice

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

Deadlock

- In molte apps un proc necessita di usare piu' risorse in modo esclusivo
- Si ha deadlock quando due processi detengono ciascuno una risorsa che serve all'altro per proseguire
- Quando due o piu' 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 ($S1_0=S2_0=1$)

Processo 1	Processo 2
...	...
Wait(S1)	Wait(S2)
<SC – R1>	<SC- R2>
Wait(S2)	Wait(S1)
<SC – R1+R2>	<SC – R2+R1>
Signal(S2)	Signal(S1)
Signal(S1)	Signal(S2)

Definizione formale

- Un insieme S di processi e' in deadlock se ogni processo di S e' in attesa di un evento che solo un altro processo $\in$ S puo' causare
  - Tutti sono in attesa ➔ Nessuno puo' provocare l'evento che risveglia un altro processo
  - Generalmente l'evento e' 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

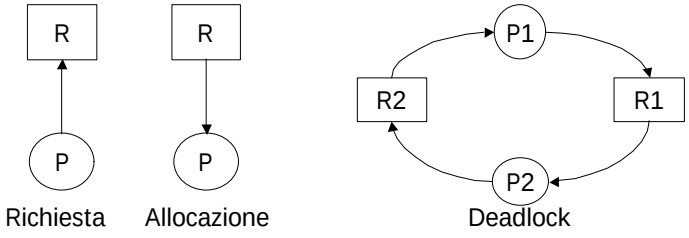
- Coffman *et al.* (1971) hanno dimostrato che e' necessario che si verifichino quattro condizioni perche' vi sia deadlock
  1. ***Mutua Esclusione***. Ogni risorsa risulta assegnata esattamente ad un processo oppure e' disponibile.
  2. ***Prendi e Aspeta*** . I processi che detengono risorse assegnategli in precedenza possono richiederne di nuove.
  3. ***Assenza di Prerilascio***. 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 che dover utilizzare una sola risorsa alla volta
  - Compromesso utilita' 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'e' un ciclo (deadlock) si terminano processi a caso nel ciclo fino ad interromperlo



Richiesta Allocazione Deadlock

Strategie per evitare il deadlock

- Prevenzione dinamica:
  allocazione attenta delle risorse
  - Vincolare i processi in modo che I deadlock risultino strutturalmente impossibili.
  - Es. Algoritmo del banchiere
    Il banchiere (SO) non soddisfa 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 - Threads
  (flussi di controllo separati 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)

Primitive del modello a scambio di messaggi

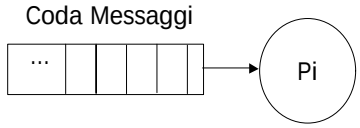
- Un msg si può pensare costituito da:

```
Type messaggio
  Origine:      ...;
  Destinazione: ...;
  Contenuto:    ...;
end
```

Primitive del modello a scambio di messaggi

Nel caso più semplice si può supporre che

- ∀ processo ∃ una coda per i messaggi in arrivo
- Primitive di comunicazione utilizzate:
  - Send(m): inserisce il msg nella coda del destinatario
  - Receive(m): preleva il msg dalla coda del proc corrente



Scambio di messaggi

- Con lo scambio di msg si realizza
  - Comunicazione:
    Un processo attraverso la ricezione di un msg ottiene valori da un processo mittente
  - Sincronizzazione:
    Un msg può essere rx solo dopo essere stato tx
    Ttale 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>

- L'esecuzione della Send determina il contenuto del msg tramite la valutazione di <expression_list>
- <dest_designator> indica la destinazione del msg

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

- L'esecuzione della Receive determina l'assegnamento dei valori contenuti nel messaggio alle variabili in <var_list> e la successiva distruzione del msg
- <source_designator> indica l'origine dei msg

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 msg che può essere ricevuto solo da P2
  - P2: Receive msg from P1
    - P2 Riceve un msg che può essere inviato solo da P1

Direct Naming - Esempio

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

```
Process reader
 var card: CardImage;
 Loop
  <read card from cardreader>
  send card to executer
 end
end;
```

```
Process executer
 var card: CardImage;
 var line: LineImage;
 Loop
  receive card from cardreader
  <process card and gen line>
  send line to printer
 end
end;
```

```
Process printer
 var line: LineImage;
 Loop
  receive Line from executer
  <print line on line printer>
 end
end;
```

Flusso dell'Informazione

Esempio piu' attuale: shell

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 e' l'Input di un altro.
  Sistema concepito in termini di flusso di informazione

Produttore-Consumatore con Scambio Messaggi Esplicito

```
Produttore
Begin
  repeat
     Receive  (Consumatore, Trigger)
     <Produzione Msg>
     Send (Consumatore, Msg)
  forever
end
```

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

Modello Client-Server

Uso di un Processo come Gestore di Risorse
(Panettiere)

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

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 eventuale risposta
- Questo schema facilita implementazione paradigma client-server (Es: browser e server web)

Modello Client-Server

- Schema da-molti-a-uno (Es: DBMS)
  - 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 Web (2° aspetto della stessa topologia)
  - I client inviano richieste ad un qualunque server tra un set di server equivalenti
  - Difficile da realizzazione con designazione esplicita. Richiede Designazione indiretta o globale (MailBox)

Modello Client-Server e Naming

- Direct naming e' in generale poco adatto per C/S
- Many-to-one: receive di un server dovrebbe consentire la ricezione di un messaggio da un qualsiasi cliente
  - Nel caso di una designazione esplicita simmetrica sarebbe necessaria almeno una receive per ogni cliente
- Many-to-many: la send di un cliente dovrebbe produrre un msg che possa essere ricevuto da uno qualsiasi dei server

⇓

Occorre uno schema piu' sofisticato pr la definizione dei canali di comunicazione: designazione globale o indiretta. Fa uso di nomi globali detti Mailbox

Designazione Globale

- Una MailBox puo' essere <dest_designator> o <source_designator> nelle istruzioni di send e receive di qualunque processo
- I messaggi inviati ad una specifica MailBox possono essere ricevuti da un qualsiasi processo che effettui una receive designando tale MailBox
- Notazione

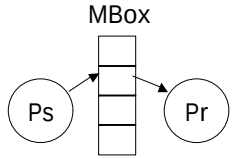
  send msg **to** A_mbox

  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

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)

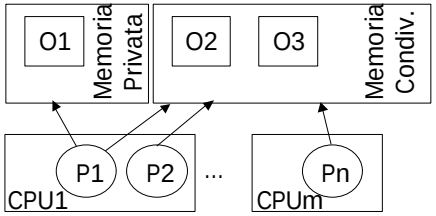
  B. Tipo di sincronizzazione
  - Sincrona
  - Asincrona
- Caratteristiche ortogonali
  - Le soluzioni proposte per A. e B. sono indipendenti

Uso delle MailBox

- Consente in modo immediato la programmazione delle 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 ...

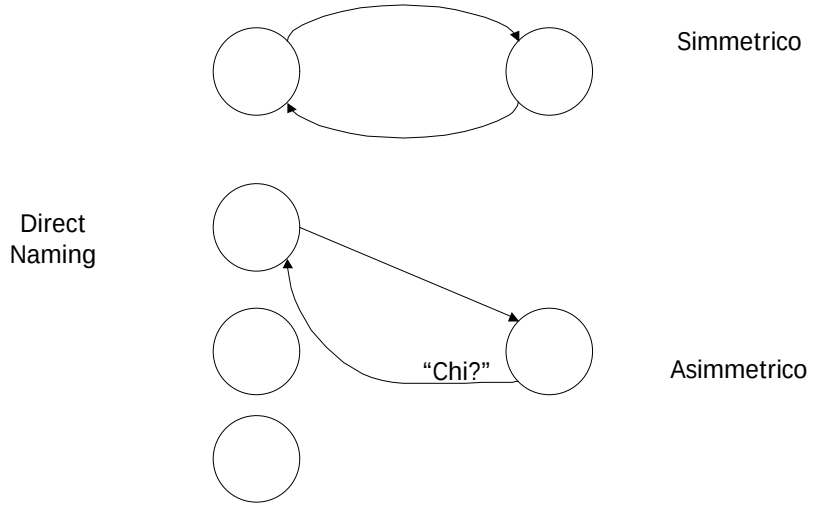
Modello Ad Ambiente Globale 2 (Lezione Precedente)

- Naturale astrazione di un Sistema in Multiprogrammazione
- Costituito da uno o piu' processori che hanno accesso ad una memoria comune



Tutte le Interazioni avvengono in mem condivisa

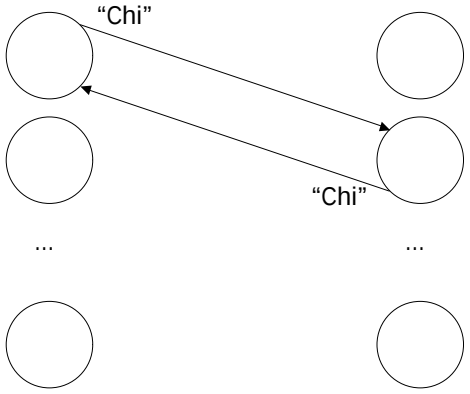
Direct Naming: Topologia



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 msg di richiesta indirizzato ad una mailbox sia inviato a tutti I processi che possono eseguire la receive su di essa
  - Non appena il msg e' ricevuto da un processo, esso non e' piu' disponibile per tutti gli altri servitori

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

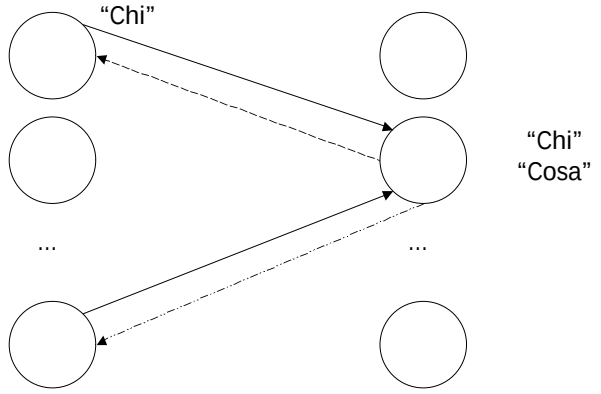
Porte

- Un processo puo' selezionare I msg che desidera ricevere attraverso l'uso di porte distinte
- Se un processo effettua una receive su una sola porta, lo schema di designazione e' logicamente equivalente ad un direct naming asimmetrico a meno di aspetti di modularita' e flessibilita'
  - Nel direct naming i client possono inviare richieste ≠ al server che deve selezionare cosa fare
  - Con le porte richieste ≠ giungono a porte ≠ se voglio aggiungere servizi aggiungo una porta

Porte

- Sono MailBox il cui nome puo' comparire solamente in un processo come <source_designator> in uno statement di receive
- Realizzazione piu' semplice delle mailbox: Tutte le receive che indicano una porta compaiono in un solo processo (mapping 1-1 porta, processo rx)
- Soluzione al problema "da-molti-a-uno" (ma non a quello "da-molti-a-molti")

Global Naming: Topologia



Naming

Naming		Comunicazione
Direct	(simmetrico)	One to One
	(asimmetrico)	Many to One
Global		Many to Many
Port		Many to One

- Global naming e' il caso piu' generale.
  Gli altri schemi:
  - limitano i tipi di interazione direttamente programmabili
  - sono piu' semplici da implementare

Naming

- Naming Statico
  - Impedisce ad un programma di comunicare tramite canali non noti a tempo di compilazione
  - Ne limita le capacita' di sopravvivenza in un ambiente dinamico
  - Il potenziale accesso di un programma ad un canale deve essere assicurato fin dall'inizio:
    Permanantemente
- Naming Dinamico
  - Lo schema Statico di base di designazione canali viene arricchito mediante variabili
    per la designazione di Sorgente o Destinazione

Naming

- La designazione dei canali puo' avvenire:
  - staticamente, a tempo di compilazione
  - dinamicamente, a tempo di esecuzione

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

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

- L’invio di un msg costituisce un pto di sincronizzazione sia per il mittente che per il destinatario
- Il trasferimento di informazioni avviene quando entrambi i processi sono pronti a comunicare
- L’Interazione viene definita come scambio di msg sincrono
- Ai processi sono associati canali privi di memoria, uno per ogni tipo di msg che il processo puo’ ricevere

Send Sincrona

- “Rendez-vous” Semplice
- Mittente si blocca in attesa che il msg sia stato ricevuto
- Un Msg ricevuto contiene informazioni corrispondenti allo stato attuale del processo mittente (Scrittura e verifica dei programmi + semplice)

Send Asincrona

- Il mittente continua l’ esecuzione dopo l’invio del messaggio
- Il messaggio ricevuto contiene informazioni che NON possono essere associate allo stato attuale del mittente (difficolta’ di verifica dei programmi)
- Interazione definita come scambio di msg asincrono
- 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 msg sia piena
  - La primitiva send solleva un'eccezione che viene notificata al processo mittente

Receive Sincrona

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

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 asincrona con interrogazione stato canale

- Soluzione:
  Specificare piu' canali di input $\forall$ proc,
  ciascuno dedicato a msg di tipo .
  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 msg presente o canale vuoto (**receive** non bloccante)

  Cio' consente ad un processo di selezionare l'insieme dei canali da cui prelevare un msg
- Inconveniente:
  per l'attesa di msg 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 Ł Service indica il processo server
    - Caso designazione indiretta (porte o mailbox) Ł 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 statement

- Specifica lato server come statement
- La procedura remota e' uno statement e come tale puo' essere collocato in un pto qualunque del processo server

  accept service

  (in <in_params>, out <out_params>) body
- L'esecuzione della accept sospende il server fino all'arrivo di un msg corrispondente alla call del servizio.
- L'esecuzione del corpo puo' fare uso dei valori dei parametri e di tutte le variabili accessibili dallo scope dello statement
- Al termine viene tx il msg di risposta al processo chiamante, dopo che di che il proc server continua la propria esecuzione
- ∀ servizio e' associata una coda distinta, generalmente FIFO

RPC lato server come Procedura

- Specifica del lato server come procedura

  Remote procedure service (in <in_params>, out <out_params>)

  <body>

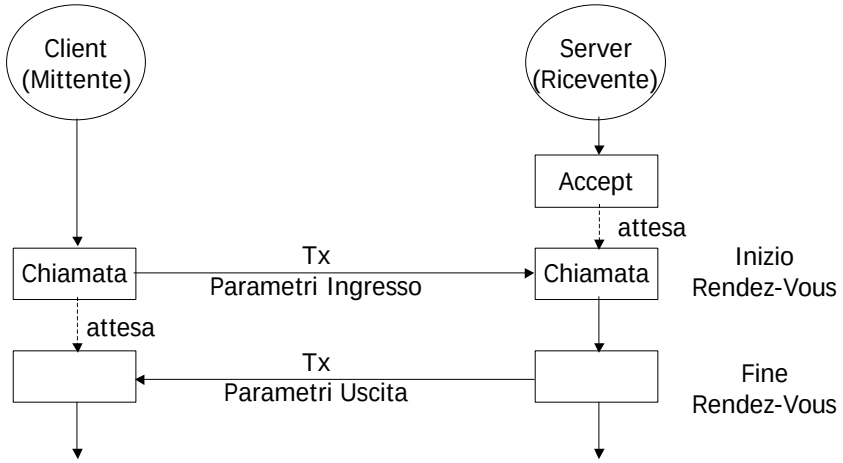
  end
- La procedura remota viene dichiarata come una procedura in un linguaggio sequenziale
- Implementata come un processo servitore che attende la rx di un msg, esegue <body> e tx un msg di risposta
- Puo' essere implementata:
  - come singolo processo: esegue richieste una alla volta sequenzialmente
  - con la creazione di un nuovo processo per ogni chiamata: le varie istanze sono eseguite concorrentemente e potranno eventualmente doversi sincronizzare tra loro.

Uso della accept

- RPC viene chiamata extended rendez-vous: Client e server si incontrano per la adurata della esecuzione del corpo della accept per poi proseguire separatamente
- Vantaggi
  - Server puo' fornire piu' tipi di servizi (accept diverse)
  - Server puo decidere quando servire le call dei client
  - Server puo' selezionare quali tipi di call servire
  - Le accept possono essere alternate o innestate
  - Vi possono essere piu'accept di chiamate allo stesso servizio con diverso <body> (ad esempio per l'inizializzazione)
- 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 si' che ad una richiesta possano corrispondere azioni diverse in funzione dello stato del processo server

  Ł 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'accoppiamento tra una chiamata priva di params e una accept priva di <body> rappresenta la trasmissione ed il relativo riconoscimento di un segnale di sincronizzazione
- Una chiamata con soli params di ingresso ed una accept priva di corpo definiscono invece un rendez vous stretto
- L'istruzione accept consente di considerare un processo come un modulo che incapsula un insieme di funizoni chiamabili dall'esterno ed eseguibili una alla volta, con analogie con il monitor in ambiente globale