

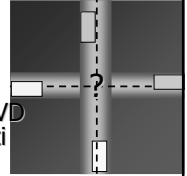
Indice

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

1

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



2

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

3

Esempio di deadlock: Programmazione

- Due processi A, B
- Due risorse R1, R2 protette da semafori S1, S2 ($S1_0=S2_0=1$)

Processo A

```
...  
Wait(S1)  
<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)
```

4

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

5

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 è disponibile.
2. **Prendi e Aspetta.** I processi che detengono risorse assegnategli in precedenza possono richiederne di nuove.
3. **Impossibilità di Preemption.** 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 è in attesa di una risorsa posseduta dal processo che segue nella lista.

6

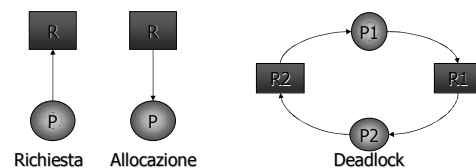
Strategie per evitare il deadlock

- 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 è costoso

7

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



8

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)

10

Struttura di un messaggio

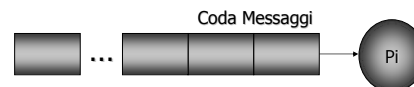
- Un msg si può pensare costituito così:
Type messaggio
Origine: ...;
Destinazione: ...;
Contenuto: ...;
end
- Contenuto può essere vuoto: sincronizzazione

11

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



12

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

14

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

15

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

16

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

17

Direct Naming (Designazione Esplicita)

- Semplice da implementare e utilizzare:
- 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

18

Direct Naming - Esempio

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

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

Esempio piu attuale: shell

Flusso dell'informazione

Produttore-Consumatore con Scambio Messaggi Esplicito

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

20

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

21

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

22

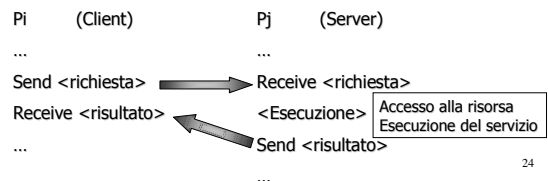
Direct Naming Schema asimmetrico

- 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)

23

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



24

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)

25

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

26

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

27

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

28

Designazione Globale

- Notazione

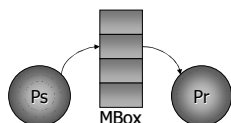
Send msg **to** A_mbox

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

29

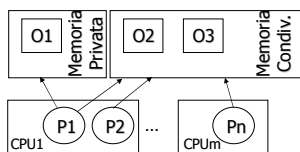
Uso delle MailBox

- 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'identità del particolare server ...

30

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



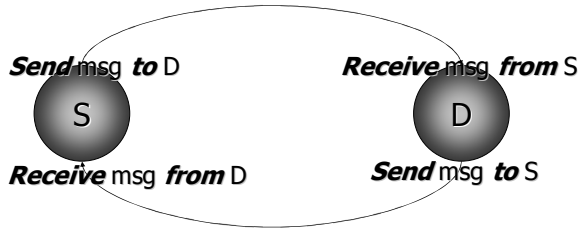
31

Uso delle Mailbox

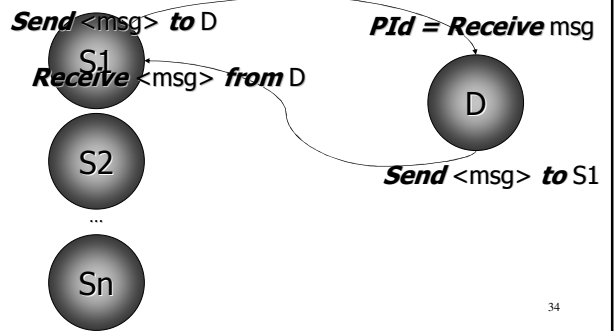
- L'implementazione di una mailbox in ambiente distribuito presenta problemi di natura realizzativa
- Il supporto run-time del linguaggio deve garantire che:
 - Un messaggio di richiesta indirizzato ad una mailbox sia inviato a tutti i processi che possono eseguire la **Receive** su di essa
 - Non appena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

32

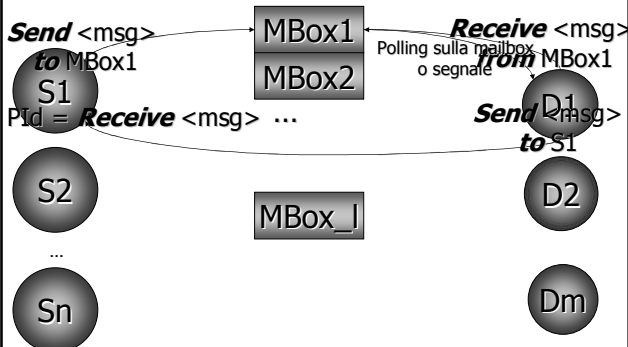
Direct Naming Simmetrico: Topologia



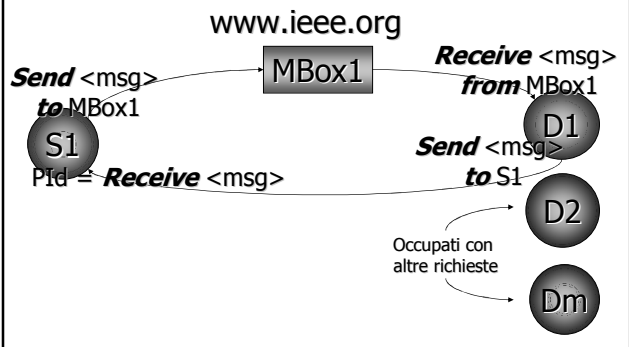
Direct Naming Asimmetrico: Topologia



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

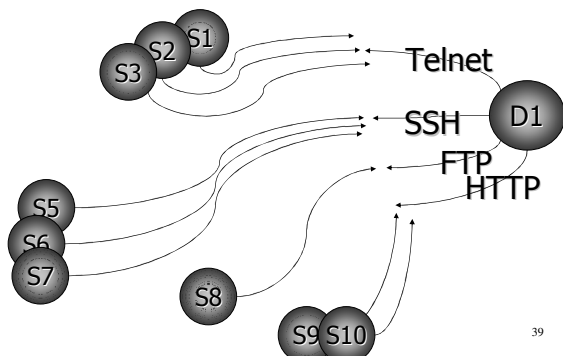
37

Porte

- Sono mailbox il cui nome può comparire solamente come <source_designator> in uno statement di **Receive**, ovvero solo un processo accede in **Receive** alla mailbox
- 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"

38

Global (Port) Naming: Topologia



39

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

40

Naming

<u>Naming</u>		<u>Comunicazione</u>
Direct	(simmetrico) (asimmetrico)	One to One Many to One
Global		Many to Many
Port		Many to One

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

41

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

42

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 essere assicurato fin dall'inizio: **permanentemente**
- Naming Dinamico
 - Lo schema statico di base di designazione canali viene arricchito mediante variabili per la designazione di Sorgente o Destinazione

43

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

44

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

45

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

46

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

47

Send Asincrona

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

48

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

49

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

50

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

51

Receive Sincrona

- Normalmente e' bloccante se non vi sono messaggi sul canale
- Punto di sincronizzazione per il processo ricevente
- Problema: a volte 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

52

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

53

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 ⁵⁴

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

55

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
- 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 potranno eventualmente doversi sincronizzare tra loro

56

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

57

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 puo' fare uso dei valori dei parametri e di tutte le variabili accessibili dallo scope (ambito) dello statement
- Al termine viene trasmesso il messaggio di risposta al processo chiamante, dopo di che il processo server continua la propria esecuzione
- Ad ogni servizio e' associata una coda distinta, generalmente FIFO

58

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 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)

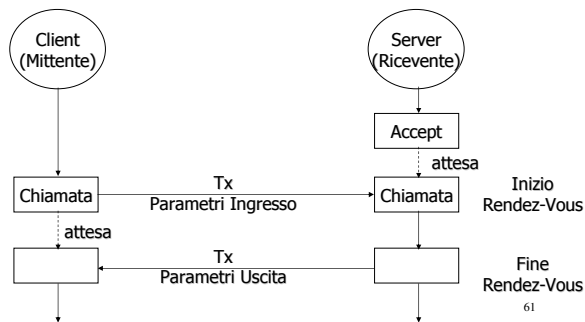
59

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 si che ad una richiesta possano corrispondere azioni diverse in funzione dello stato del processo server
 - Netta distinzione rispetto alla definizione di procedura

60

Schema di comunicazione RPC



Uso della RPC

- Lo schema di comunicazione realizzato dal meccanismo della chiamata a procedura remota è 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

62

Uso della RPC: osservazioni

- 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

63