TEMI

COMUNICAZIONE

SINCRONIZZAZIONE

NAMING

PROCESSI

FILE SYSTEM

RISORSE

mobilità
protezione

SICUREZZA

ESEMPI

COMUNICAZIONE

Assumendo che non sia presente **memoria condivisa**
==> COMUNICAZIONE **Inter Processo**
necessità di ottenere un punto di vista globale
per la cooperazione dei processi

Ambienti diversi
Rete di interconnessione, Sistema Operativo
Linguaggio

Buona performance => organizzazione a primitive

a livelli di astrazione diversi
scambio messaggi (message passing)
Remote Procedure Call (**RPC**)
transazioni (consistenza e
semantica di raggruppamento)
trend **Semantica di comunicazione di gruppo**

SCAMBIO DI MESSAGGI

THE Dijkstra
sistema operativo Hansen
Demos e **Thoth**

messaggio
insieme di dati formata da un header (lunghezza fissa) e da un corpo di dimensione variabile
inviato da un mittente ad un destinatario

Caratteristiche dello scambio di messaggi
categorizzazione tenendo conto degli standard

PROPRIETÀ delle PRIMITIVE

diretto/indiretto
simmetrico/asimmetrico

bloccante/non bloccante
sincrono/asincrono
bufferizzato/ non
reliable/unreliable

multiplo
multicast/broadcast
con semantica diversa

semantica delle azioni

SINRONICITÀ

la azione implica la attesa del completamento e della comunicazione dal pari

In caso **asincrono**, non si ottiene alcuna informazione sul completamento, e, tanto meno, un risultato

PRIMITIVE bloccanti/non bloccanti

ritardo nella invocazione della primitive
dovuto a diverse durate della primitiva stessa

non bloccante ==> nessun ritardo
DISTINZIONE a livello locale
nessuna implicazione a livelli di semantica

PRIMITIVE NON BLOCCANTI

*primitive **non bloccanti***

- send termina appena il messaggio è stato accodato o è stato copiato per la trasmissione (localmente)
- receive segnala la disponibilità a ricevere e mette a disposizione un area di buffer (come trovare valore poi?) *primitiva di accesso collegata all'arrivo del risultato*
- uso di interruzioni per segnalare il completamento della operazione

VANTAGGI
massima flessibilità
durata prevedibile (real time)

SVANTAGGI E PROBLEMI

- possibilità di accesso multiplo all'area del messaggio
- difficoltà di programmazione: il partner della comunicazione si trova in uno stato impredicibile
- difficoltà di prova dei programmi

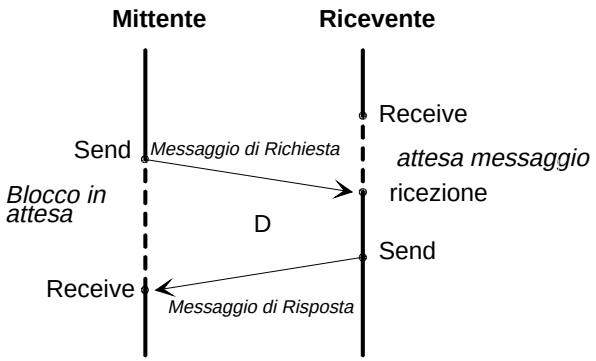
PRIMITIVE BLOCCANTI

si combina **comunicazione** e **sincronizzazione**
con aggancio alla semantica

- **send** non termina fino a quando il messaggio non è stato trasmesso
  (possibilità, in caso di affidabilità, di attendere anche la avvenuta ricezione da parte del destinatario)
- **receive** non termina fino a quando il messaggio è nel buffer del processo ricevente

RECEIVE
bloccante
non-bloccante (spesso detta asincrona)
primitiva di verifica di ricezione

Uso mescolato di primitive dei due tipi non produce conflitto



VANTAGGI
massima sicurezza
durata imprevedibile

SVANTAGGI
limitazione del parallelismo

Primitive sincrone (bloccanti)

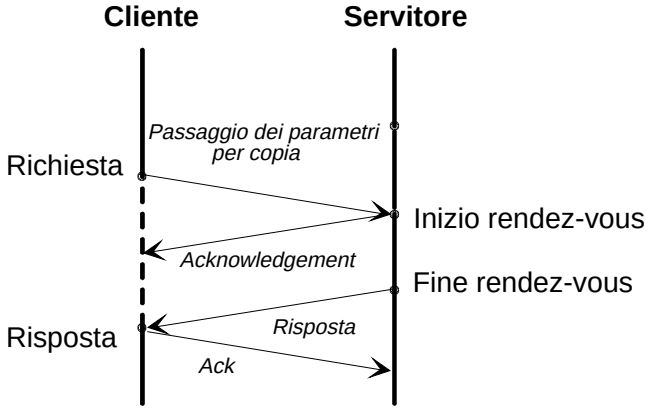
Rendez-vous vedi **CSP ed occam**

Rendez-vous remoto

cliente/servitore con spazi di indirizzamento diversi
visibilità di interfaccia reciproca
conoscenza dell'identità reciproca

Ancora **semantica di attesa**
Problemi se si vogliono send multiple o receive multiple

Uso di pacchetti di trasmissione a livello di supporto
Necessità di registrazione dei processi ad un gestore dei nomi (name server) per ottenere la conoscenza reciproca



PRIMITIVE bufferizzate e non

buffer ***risorsa di memorizzazione***
rappresenta un'area di accodamento del messaggio tra la trasmissione e la ricezione

DIMENSIONE DEL BUFFER

nessun buffer
vs.
buffer di dimensione infinita

buffer pieno ==>
- send ritardata
- send errore al mittente

buffer vuoto ==>
- receive ritardata
- receive eccezione al ricevente

BUFFER INFINITO ==>
send non ha mai ritardo

send primitiva asincrona

Il mittente può andare oltre il ricevente di un numero qualunque di passi
==>
*il ricevente **non** può fare ipotesi sullo stato del mittente*

NO BUFFER ==>
send e receive sono sincronizzate

primitive sincrone (bloccanti)

BUFFER => IMPEGNO di MEMORIA

buffer di dimensioni finite ==>
- il ricevente può specificare la dimensione del buffer (porta o mailbox) - tipo sliding window
- il mittente può essere avanti rispetto al partner delle dimensione del buffer al massimo

Sistemi bufferizzati:
- **maggiore complessità**
- **gestione dei nuovi oggetti**

Spesso la gesitone è a carico del protocollo

confronto
SINCRONICITÀ ==> **SEMANTICA**
BLOCCO ==> **DECISIONE LOCALE**

Primitive **sincrone bloccanti** =>
sincronizzazione tra mittente e destinatario

Primitive **sincrone non bloccanti** =>
sincronizzazione nel messaggio
possibile ricupero *(successivo)* del risultato

Primitive **asincrone non bloccanti** =>
Primitive **asincrone bloccanti** =>
non c'è l'idea di un risultato

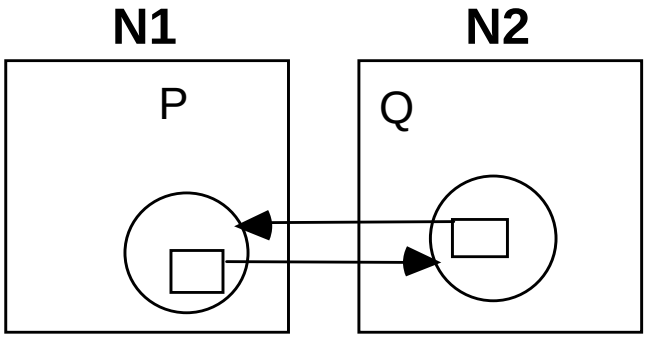
ATTESA ==> **PER QUANTO TEMPO**
introduzione di time-out nei sistemi reali
specie per il cliente (in caso che il servitore non ci sia)

PROBLEMI DI MEMORIA

Supponiamo che P mandi a Q e Q mandi a P

In caso di
- **operazioni sincrone**
- **limiti di memoria**

possibilità di DEADLOCK



Se i nodi non hanno più memoria
i due processi sono obbligati a sospendere ogni attività

naturalmente, *la condizione di* **deadlock** *può derivare da un ciclo non tra due vicini diretti, ma anche a causa di una catena di vicini*

Esempio per T800
In CSTools possibili due modi di comunicare:
- attraverso **canali** (hw o sw): necessità di routing;
- attraverso **transport** come accesso ad una comunicazione non dipendente dalla allocazione si fornisce un routing automatico

TRANSPORT
Ad ogni transport è associato
un ***end-point*** per la ricezione e/o la trasmissione

Primitive di send (csn_tx) e receive (csn_rx)
P1: csn_tx (TX1,sync/async, nomelocaleTXdi2, messaggio, lunghmessaggio)

& parametri di uscita
P2: csn_rx (TX2, &nomelocTXdi1 *(da chi si riceve)***, &messaggio, &lunghmessaggio)**
P2: csn_rxnb (TX2, &messaggio, &lunghmessaggio)

P2: csn_test (TX2, flag *(selezionedell'attesa)***, timeout, &nomeintTXdi1, &areamessaggi, &status)**

BLOCCO ==> il processo si incarica dell'azione
NONBLOCCO ==> si delega un processo CSN

SINCRONO/ASINCRONO => comportamento
o del processo o del processo di comunicazione

Registrazione di transport

Un processo per *ricevere/trasmettere*
deve avere definito e reso noto almeno
un ***transport*** (registrato ad un name server)

I TX di altri processi sono visti attraverso un nome o *identificatore locale* associato (***netid***), ottenuto da un name server, attraverso un accordo su *nomi globali*

Un processo P1 deve definire un ***transport***
csn_open (indice, &Transport)

Registrazione ad un name server
csn_registername (Transport, NomeEsternoUnico)

Il nome ***NomeEsternoUnico*** *è una semplice stringa*

Richiesta ad un name server per ottenere un nome locale per un transport registrato con un nome globale
csn_lookupname (&Netid, NomeEsternoUnico, AttesaoMeno)

NomeEsternoUnico	unico nel sistema
Transport	struttura di CSN per un transport
netid	nome locale per un transport

PRIMITIVE reliable/unreliable

Affidabilità

possibilità di superare problemi/errori come:
- messaggi ricevuti in ordine sbagliato
- perdita di un messaggio in trasmissione
- nodo partner non più raggiungibile

e per fare fronte ai problemi
- duplicazione di messaggi

Unreliable send

nessuna garanzia di consegna, ritrasmissione, terminazione corretta, etc.

Reliable send/receive

messaggi persi ==> ritrasmissione
uso di time-out
e di conferme (acknowledgement)

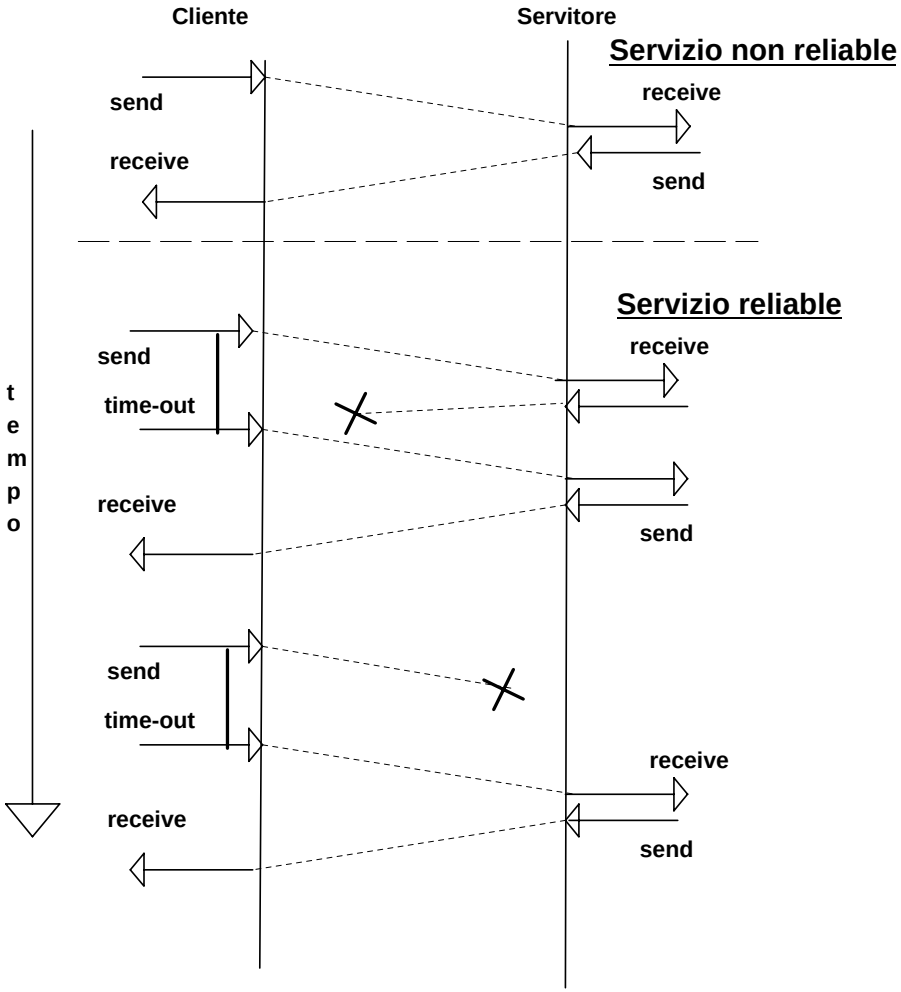
al completamento, il messaggio è stato ricevuto

Quale livello si deve occupare di questo?

basso livello (trasmissione)
confronto con il costo a basso livello
non economicità

recovery a livello di processo
aggancio con le opportune eccezioni a livello applicativo

Affidabilità



Possibilità di
(quante?) ritrasmissione e
(quali?) time-out

SEMANTICA delle azioni conseguenti

may-be

l'azione può essere stata fatta o meno

at-least-once

l'azione può essere stata anche **più** volte a causa della duplicazione dei messaggi dovuti a dissincronizzazioni
*in caso di insuccesso **nessuna** informazione*

==>

- azioni **idempotenti** oppure
- inconsistenza delle azioni del ricevente

at-most-once

l'azione può essere avvenuta **al più** una volta
l'azione può **non** essere stata fatta per niente
*in caso di insuccesso **nessuna** informazione*

==>

il mittente non sa se l'azione ha avuto luogo o meno

- Si ritrasmette e si devono numerare le azioni al server

exactly-once

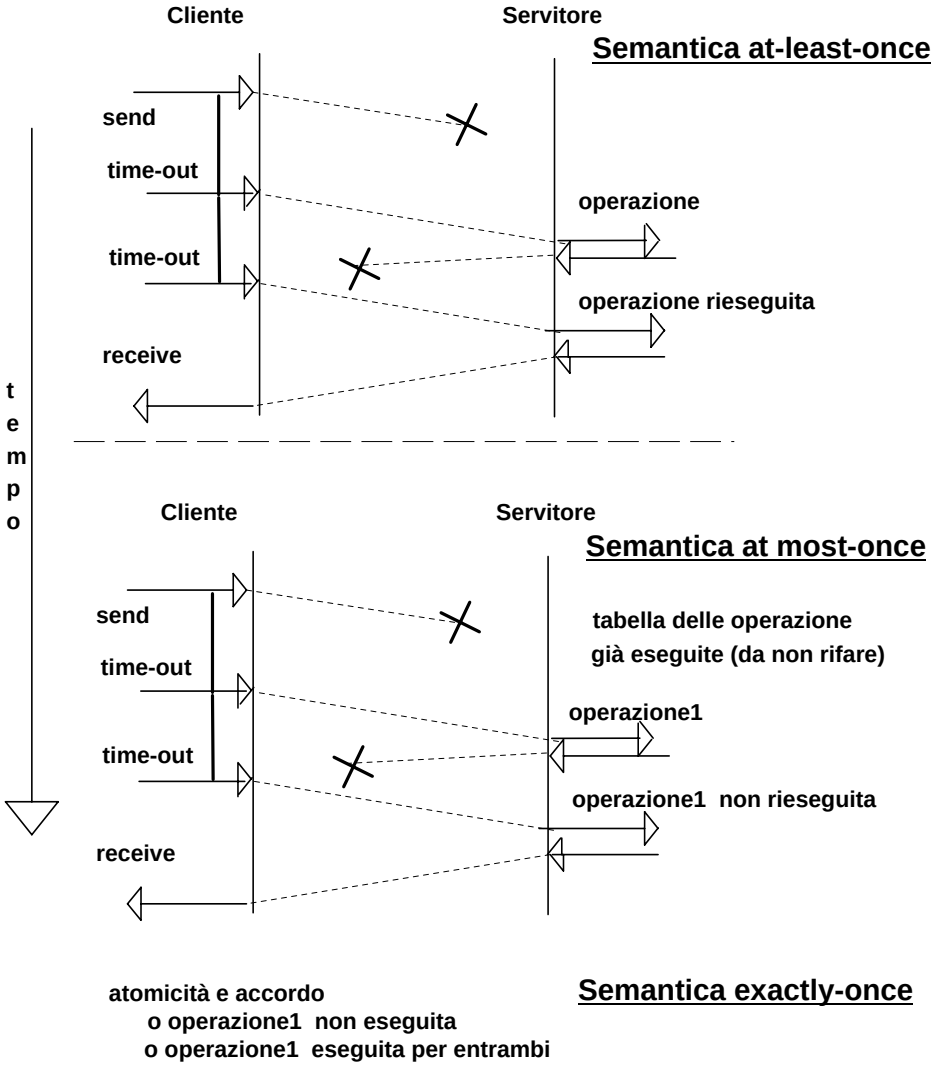
l'azione è stata eseguita **una volta sola** oppure eccezione (conoscenza dello stato del partner) e l'azione **non è** stata fatta per niente

==>

Il ricevente deve tenere traccia di tutte le azioni fatte e scartare quelle richieste più volte
Per quanto tempo si tiene conto delle azioni fatte?

(semantica con conoscenza dello stato dell'altro e durata del protocollo)

Semantica



exactly-once

Al termine sappiamo se l'operazione è stata fatta o meno anche dalla parte del cliente

PRIMITIVE dirette/indirette

Uso di entità intermedie cui inviare i messaggi

Dirette
PRIMITIVE simmetriche

P: send (ProcessQ, message)
Q: receive (ProcessP, message)

Si crea uno ed un solo link bidirezionale tra due processi attraverso cui si scambiano i messaggi

PRIMITIVE asimmetriche

P: send (ProcessQ, message)
Q: receive (AnyProcess, message)

Il modo diretto lascia libertà ai processi
Difficile leggibilità e scarsa modularità

Capacità espressiva limitata
- NON si possono avere più clienti
- NON modello cliente/servitore

PRIMITIVE Indirette

Uso di ***link***, ***porte***, o qualunque altra entità diversa dai processi partner

LINK (**DEMOS** e **Charlotte**)

Un **link** ==> un *canale di comunicazione unidirezionale per un processo* (utente, di sistema, o del kernel stesso)
Meccanismo ***asimmetrico*** di comunicazione: il cliente conosce il servitore e non viceversa
Un **link** come oggetto protetto gestito da un name server centralizzato (kernel)

send (link, buffer) **receive (link, buffer)**

Ogni processo può generare un link a se stesso e può passarne la conoscenza ad altri
Il kernel mantiene la tabella dei link per ogni processo

Il modello cliente servitore completo:
- il cliente deve avere un link per il servitore;
- il cliente invia un link a se nel messaggio al servitore, che lo usa per la risposta

VANTAGGI
Facilità alla **migrazione** (**dinamicità**)
Possibilità di **trasmissione** anche di aree di dimensioni elevate con ottimizzazioni dell'area usata

SVANTAGGI
Overhead di gestione
Link **dangling**

LINK BIDIREZIONALI

Un **bilink** ==> un canale di comunicazione *bidirezionale*, con bufferizzazione o meno e con blocco o meno tra due processi
Si possono fare azioni di invio/ricezione da parte di entrambi i processi

Buffer
Gestione accurata a livello utente

Non Blocking
un processo può avere più richieste pendenti (send o receive)
==> modello cliente/servitore multiplo

necessità di avvertire il completamento

Un bilink può anche essere trasferito da un processo ad un altro, cambiandone il processo agganciato, per esempio tramite una primitiva

send (link, buffer, link-da-agganciare)

Overhead

LINK ==>
astrazione di una ***connessione tra due processi***
(anche aggancio dinamico)

PORTE (ACCENT)

Una **mailbox** è un nome globale per la comunicazione indiretta tra qualunque processo

send (mailbox, buffer) receive (mailbox, buffer)

Overhead nella implementazione:
- una send deve tenere conto di tutte le possibili receive
- una receive deve estrarre il messaggio per tutti gli altri

Una **porta** (*oggetto protetto di kernel*) consente operazioni di **invio**, **ricezione**, **proprietà**: un solo processo può fare le operazioni di ricezione in un certo istante, in base al valore dei diritti

send (port, buffer) receive (port, buffer)
Anche porte bidirezionali

Il processo owner può distribuire i diritti e passare anche la porta

owner legato alla porta stessa: se termina, termina la porta, eccezione per gli altri (all'invio, ricezione)

Il processo owner può anche cedere la ownership

IMPLEMENTAZIONE

la porta è associata ad una coda FIFO, per il processo ricevente

In ACCENT

Ogni comunicazione avviene attraverso le porte

Ogni processo definisce le proprie porte
Anche il Kernel comunica attraverso porte

Tre tipi di processi:
di utente
di sistema
kernel vero e proprio

Il gradi di bufferizzazione di una porta è limitato e dipende da quanto specificato alla creazione

BUFFER PIENO

- il processo sospeso fino a che non si trova spazio
  tipico in *comunicazione per processi utente con un server*
- il processo riceve una eccezione (dopo un time-out)
  tipico di *comunicazione non cliente/servitore*
- il processo non riceve indicazioni, ma il messaggio è accodato dal kernel fino a quando l'invio non è possibile
  tipico di *comunicazione per processi servitori*

MESSAGGI

PASSAGGIO DEI PARAMETRI

per valore

per riferimento (necessità di ***spazio comune*** o di ***trasmissione***)

In caso di **trasmissione**, interazione con la gestione della memoria e copia della struttura logica

Uso di riferimento e valore **Accent** e **V-system**

I PARAMETRI DEVONO ESSERE IMPACCATI IN UN MESSAGGIO UNICO LINEARIZZAZIONE

(anche per grafi rappresentati da puntatori/indirizzi)

DIMENSIONE DEI MESSAGGI

Fissa programmazione più vincolata, facile supporto

Variabile programmazione facile, supporto più complesso

prima un messaggio di dimensione e poi i dati veri

ECCEZIONI

necessità di fornire una buona gestione delle eccezioni

Crash del receiver

il messaggio non può essere ricevuto

Crash del sender

il messaggio non può essere ricevuto

Il receiver/sender riceve un time-out o terminare

La terminazione attuata dal sistema operativo

le risorse usate trattate secondo strategie opportune

SCHEMI DI COMUNICAZIONE ULTERIORI

Per comunicazione uno a molti

BROADCAST

come invio generalizzato di messaggi a tutti i processi del sistema

NON si può simulare facilmente:

- incapacità espressiva
- tempo ed overhead
- eccesso di reliability

IMPLEMENTAZIONE diretta su LAN

vedi Ethernet o bus comune in generale

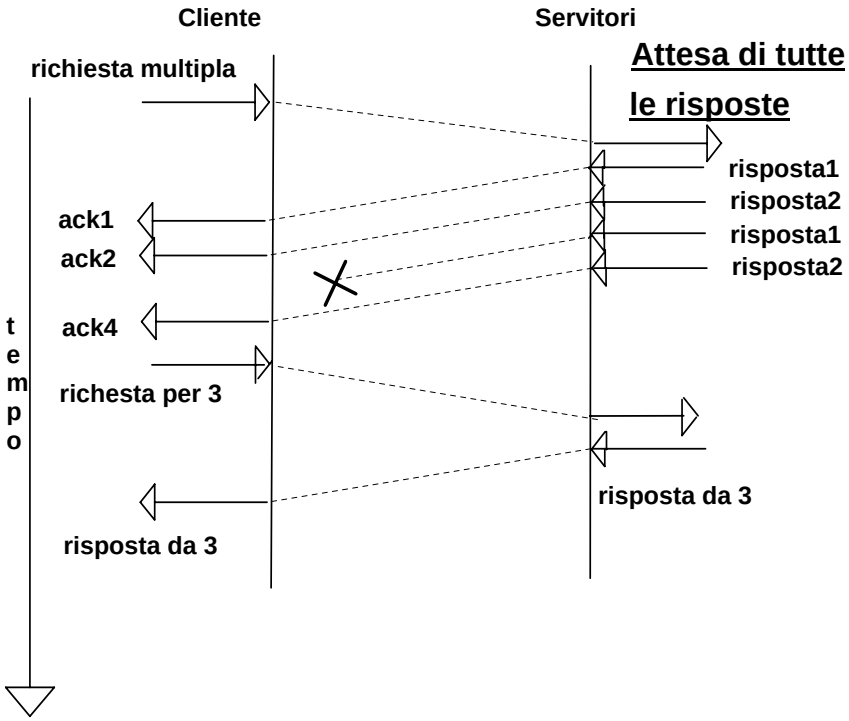
MULTICAST 1:K

come invio generalizzato di messaggi a un sottinsieme di processi nel sistema

Come trattare le risposte?

- **senza attesa**
- attesa di **una sola** risposta
- attesa di **alcune** (quante?) risposte
- attesa di **tutte** le risposte

Come specificare le azioni conseguenti alla rticezione di ogni risposta



Sollecito **selettivo** vs. Sollecito **globale**

Conferma **positiva ack** vs. Conferma **negativa nack**

Politiche diverse

Si può anche attendere solo una delle risposte

la semantica della primitiva dipende da queste decisioni

Comunicazione di gruppo implementazione

Per il supporto il multicast può ricorrere a soluzioni:

- **indirizzi di gruppo**

  possiamo approfittare di **indirizzi di gruppo** specifici che consentano di identificare gruppi noti
  (**classe D** in IP)

  possiamo usare il **supporto al broadcast**

  possiamo usare un **punto-a-punto in sequenza**

- **lista di distribuzione**

  possiamo sempre considerare la lista, che viene ad essere parte dei messaggi per il gruppo

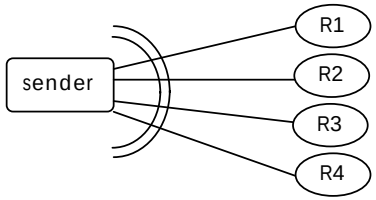
- **uso di attributi**

  possiamo inviare il messaggio con un predicato
  se il predicato è soddisfatto, allora il ricevente ha diritto al messaggio
  (ad esempio, classe B, e hostid pari)

COMUNICAZIONE A GRUPPI

MULTICAST

azione di **multicast** rende atomica l'operazione di invio multiplo



motivazioni

- *localizzazione di oggetti in un sistema*
- *fault tolerance*
- *uso di dati replicati*
- *cambiamenti multipli*

due aspetti semantici

ATOMICITÀ

garanzia di ricezione da parte di tutti i componenti del gruppo

RELIABILITY

reliable => garanzia di consegna
unreliable => 1 solo tentativo (Chorus)

e l'ordinamento dei messaggi in caso di più azioni?

RELIABLE MULTICAST

problemi se:

- omissione di un messaggio
- failure del sender

necessità di **monitoring**

- controllo di ogni comunicazione in atto
- eventuale ritrasmissione
- rimozione dei componenti falliti
- protocollo di rientro nel gruppo

IMPLEMENTAZIONE

- Invio di ogni messaggio a tutto il gruppo ed attesa time-out e ritrasmissione

Quis custodiet custodes?

- E se fallisce il controllore?
controllo da parte degli altri che il protocollo si compia
- Ma quanto si aspetta?
attesa fino ad un messaggio di completamento

efficienza

hold-back => Non si fornisce il messaggio fino a che non si è sicuri che sia quello giusto
In caso di numerazione, il messaggio è ritardato fino alla comparsa dei precedenti (**no ack**)

negative ack => numerazione dei messaggi;
in piggybacking si invia un contatore usato per indicare eventuali perdite (in modo selettivo)

Ordinamento FIFO (multicast FIFO)

dallo stesso processo allo stesso ricevente
per i messaggi in broadcast successivi

INVIO FIFO: PROBLEMI

A manda una news N_a
B riceve la news e invia una risposta N_b

C riceve prima N_b poi N_a
D riceve prima N_a poi N_b

se vogliamo evitarlo ==>

Ordinamento causale

ordinamento tra eventi di un sistema

INVIO TENENDO CONTO CAUSA/EFFETTO

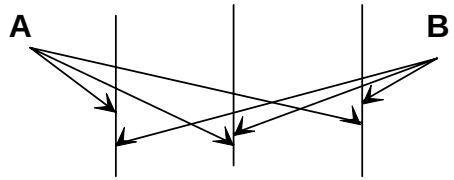
In caso di eventi scorrelati, ma con vincolo
A aggiunge tot lire a tutti i suoi conti
B aggiunge l'interesse ai conti (9%)

C1 riceve prima tot poi 9%
C2 riceve prima 9% poi tot

se vogliamo evitarlo ==>

Ordinamento atomico

ordinamento totale tra eventi di un sistema
ci possono essere molti **ordinamenti totali**



MULTICAST FIFO

multicast con ordinamento solo dallo stesso mittente allo stesso destinatario

i messaggi di uno stesso mittente arrivano e sono consegnati in ordine ai destinatari
(si possono aggiungere e mescolare in modo qualunque con altri)

MULTICAST CAUSALE

multicast con ordinamento causale (logico)

i messaggi arrivano e sono consegnati in un ordine che rispetta le comunicazione per il gruppo

se l'evento **A** viene prima di **B**, i messaggi sono in questo ordine per tutti i componenti del gruppo

ordinamento di Lamport

MULTICAST ATOMICO

nel caso di **atomicità** richiediamo lo stesso ordine (qualunque sia) per ogni componente del gruppo
quindi: **multicast con ordinamento totale**

i messaggi arrivano e sono consegnati nello stesso ordine ad ogni componente del gruppo

I due multicast causale ed atomico sono diversi
in generale, non si sussumono l'un l'altro

IMPLEMENTAZIONE MULTICAST ATOMICO

Una implementazione possibile **centralizzata**
Un **unico gestore centrale** che riceve le richieste e le ordina tutte secondo una sua logica (tipicamente unfair)

> le realizzazione è ***unfair*** per tutti gli utilizzatori e crea un pesante ***collo di bottiglia***

Una implementazione **distribuita**
richiede il coordinamento di un **insieme di gestori** (riceventi) che decidono sull'ordine e delle richieste e le servono **in modo indipendente coordinandosi**

> le realizzazione può avere un gestore per ogni richiesta che contratta con gli altri e ottiene tutte le risposte per numerare gli altri (realizzazione a **limitata scalabilità**)

Implementazione efficiente
solo in casi particolari
uso di broadcast a basso livello
per risolvere alcuni problemi

CATOCS
CAusal & **T**otally **O**rdered **C**ommunication
operations **S**upport

Ma è sempre necessario avereun coordinamento in ogni istante?

ISIS

sistema basato su ***replicazione attiva***
con decisione **dinamica** del master di ogni operazione

> necessità di una visione identica ai diversi componenti del gruppo

ABCast (Atomic BCast)
uso di una coda per ogni componente
messaggi marcati con un time-stamp iniziale e restituiti solo se in ordine giusto
Ogni messaggio arrivato richiede una fase di coordinamento per determinare il time-stamp finale:
consegna avviene con decisione locale in base al time-stamp finale

ordinamento ottenuto tramite un **coordinatore** deciso per ogni azione che centralizza la decisione (dopo un messaggio a tutti gli altri)

problemi: *ritardo* ed *overhead*

Altre implementazioni
uso di un anello logico con un unico token che designa il master; il master decide il time-stamp per ogni messaggio

guasti

per efficienza anche
CBCast (Causal BCast)

MULTICAST SOLO PER ALCUNI EVENTI

possibilità di ordinare solo alcuni degli eventi ordinamento parziale

CBCast (Causal BCast)
questo multicast tende a considerare solo alcuni eventi che sono etichettati come da ordinare
gli altri possono essere ordinati da ogni componente del gruppo in qualunque ordine siano arrivati (limitando così i costi di coordinamento)

In genere il Causal Broadcast richiede un coordinamento ai mittenti che devono adeguare un proprio "orologio logico" e fare arrivare la informazione ai riceventi
I componenti del gruppo devono solo rispettare l'ordinamento di questi

CBcast tende a imporre un comportamento aldifuori del gruppo dei riceventi
la relazione causa effetto va rilevata al momento della generazione degli eventi

ABCast tende a non imporre comportamento (costo) aldifuori del gruppo dei riceventi

MULTICAST A GRUPPI DINAMICI

possibilità di definire gruppi di processi
cui si possono inserire e togliere componenti

V-kernel anche stati inconsistenti
messaggio arriva solo ad una parte dei componenti del gruppo

ISIS GBCast (Group Bcast)
il messaggio arriva in uno dei due stati:
- tutti i componenti prima del cambiamento
- tutti i componenti dopo il cambiamento
ordinamento consistente di tutti gli eventi di bcast o prima o dopo queste azioni

monitoring degli eventi di inserimento e
di estrazione (failure)

uso di una **tabella** per ogni componente
con le appartenenze al gruppo
la tabella è aggiornata con un GBCast

Il GBCast richiede che un messaggio di questo tipo venga ricevuto solo dopo che lo sono stati tutti i BCast precedenti ancora in atto

*Per **efficienza***
organizzazione gerarchica con membri di un vicinato ed alcuni clienti in numero limitato

IMPLEMENTAZIONE

INTEGRAZIONE IPC E MEMORIA: ACCENT

integrazione tra IPC e gestione della memoria

anche V-kernel, Mach, etc.
integrano **IPC, memoria e file**

Integrazione significa
possibilità di gestire in modo remoto lo spazio di indirizzamento di un altro processo

anche attraverso i **page fault**

Accent
memoria virtuale per ogni processo utente
solo ***memoria locale***
(e per i processi di kernel)

spazio piatto di indirizzamento con indirizzi lineari

Segmenti ***temporanei e permanenti***
temporanei per esigenze locali dei processi

permanenti per trasferimento tra processi

La trasmissione (anche di grossi blocchi) di informazioni rispetta il principio che non si possono condividere dati

Trasmissione di informazioni tra spazi distinti

Distinguiamo in base alle dimensioni

BLOCCHI ordinari
spostati da un processo ad un altro direttamente
Uso di copia da uno spazio ad un altro
Il kernel, dalla parte del **sender** copia il messaggio al proprio interno, lo passa al kernel destinatario, che lo passa al **receiver**

BLOCCHI più grandi
spostati dal mittente al kernel all'invio e acquisiti dal kernel al ricevente solo alla effettiva ricezione
Non si copia immediatamente
spesso se il sender non cambia la informazione e il receiver non ne ha bisogno, si può differire lo spostamento

SOLO una COPIA da uno spazio di indirizzamento ad un altro

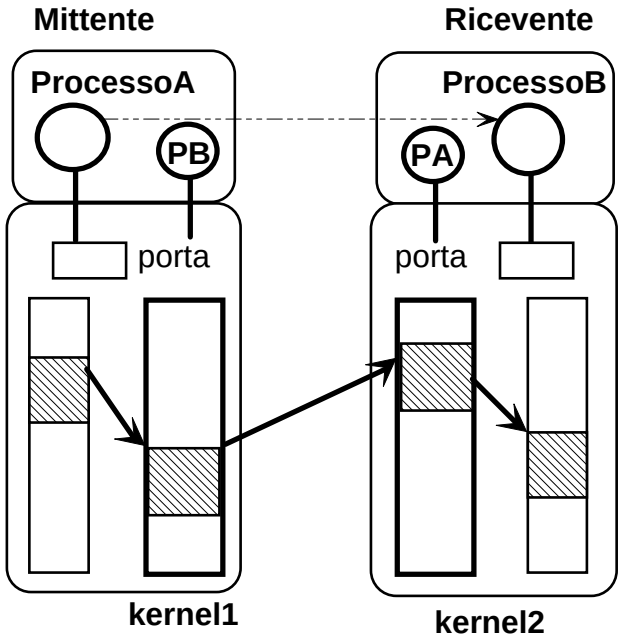
NON DUE COPIE IN SPAZI DIVERSI

tecnica **copy-on-write**
sia nel sender, sia nel receiver
La pagina può stare nel sender, se non la modifica e fino a che non la modifica

Solo le pagine effettivamente necessarie, e solo al momento del bisogno, sono effettivamente copiate

VANTAGGI della INTEGRAZIONE

- uso della memoria virtuale per migliorare la comunicazione
- memoria virtuale come risorsa di un processo
- uso di **copy-on-reference**, cioè la copia viene fatta solo al momento in cui il server ha bisogno dei dati stessi



Kernel

IMPLEMENTAZIONE

OTTIMIZZAZIONI DELLA COMUNICAZIONE

Molto dell'overhead associato ai messaggi è legato al ritardo conseguito, non solo nel trasporto delle informazioni

tempo di comunicazione

osservato tra mittente e destinatario

$T_C = T_T + T_{LM} + T_{LD}$

T_T tempo di trasmissione
T_L tempo di latenza (mittente o destinatario)

Il tempo di latenza è dovuto all'overhead della **gestione locale**,
ai **due estremi** e anche agli **intermedi**

Il messaggio è passato tra diversi livelli,
kernel, sistema, driver, fino alla applicazione
interferendo con la esecuzione normale
con la gestione della memoria, processi

ACTIVE MESSAGE

il messaggio porta le informazioni di trattamento e specifica in modo preciso come si possa ottimizzarne il recupero (permettendo
composizione,
trattamento a basso livello, etc.)