

Sistemi operativi distribuiti

Introduzione

- Definizione e collocazione
- Comunicazione in sistemi distribuiti
- Gestione di processi distribuiti
 - Concorrenza
 - Sincronizzazione
 - Deadlock
 - Migrazione
- Gestione di file system distribuiti
- Gestione memoria distribuita

DEFINIZIONE E COLLOCAZIONE

Definizione

- Sistema distribuito:
 - “Un sistema distribuito è un insieme di computer indipendenti che appaiono agli utenti come un singolo computer”
 - “Un sistema distribuito è un insieme di processori che non condividono memoria e clock”
- Termini chiave:
 - “indipendenti”
 - “senza condivisione”

Pro e Contro

- Vantaggi
 - Rapporto prezzo/prestazioni migliore rispetto a mainframe
 - Aumento della potenza di calcolo (load sharing)
 - Condivisione delle risorse (es.: periferiche costose disponibili per molti utenti, file condivisi, ...)
 - Affidabilità (tolleranza ai guasti)
 - Scalabilità (incrementare la potenza di calcolo richiede poco sforzo)

5

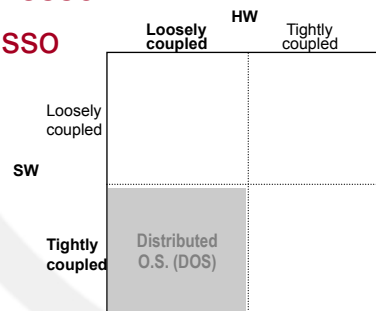
Pro e Contro

- Svantaggi
 - Complessità del SW
 - Problemi dovuti alla rete (se la rete satura?)
 - Sicurezza

6

Collocazione

- Collocazione nello spazio HW/SW
 - HW: scarsamente connesso
 - SW: fortemente connesso



COMUNICAZIONE

Comunicazione distribuita

- Sistemi senza memoria condivisa ➡ il meccanismo della comunicazione richiede schemi alternativi
 - Basati su scambio di messaggi
 - Problema: processi/host spesso remoti (rete)
 - Identificazione richiede conoscenza dell'architettura di rete

Comunicazione distribuita

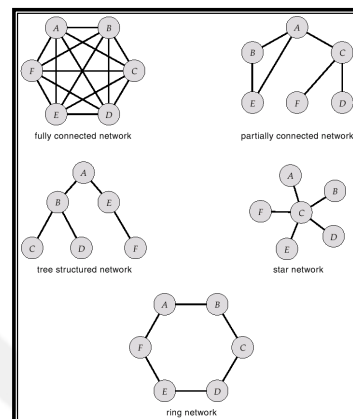
- Problemi fondamentali
 - Naming: come fanno due processi remoti a “trovarsi” per comunicare
 - Routing: come vengono inviati i messaggi sulla “rete”
 - Connessione: come due processi mandano una sequenza di messaggi
 - Contesa: come vengono risolti conflitti dovuti a richieste multiple della rete

Problematiche

- Struttura fisica: **architettura**
- Struttura logica: **protocolli**
- Schemi di **comunicazione**

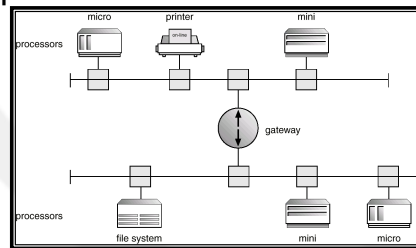
Architettura

- Vari modi di collegare i vari “nodi”
- Criteri di confronto
 - **Costo di base**
 - Quanto costa collegare i vari nodi?
 - **Costo della comunicazione**
 - Quanto tempo richiede l'invio del messaggio da A a B?
 - **Affidabilità**
 - Cosa avviene se un collegamento cessa di funzionare?



Tipi di rete: LAN

- Local-Area Network (LAN)
 - coprono piccole aree geografiche (<10Km)
- Topologie fisiche
 - bus, anello o stella
- Velocità $\approx 10 \text{ Mbps}/10\text{Gbps}$
- Broadcast veloce ed efficiente
- Nodi:
 - Workstation e/o PC
 - Uno o pochi “mainframe”



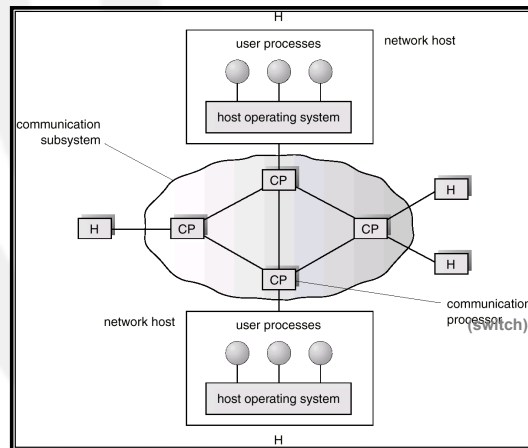
13

Tipi di rete: WAN

- Wide-Area Network (WAN)
 - collegano siti geograficamente lontani
- Connessioni punto-punto su linee tipicamente affittate da compagnie telefoniche
- Velocità variabile
 - Arpanet originale: 56 kbps
 - Link tipico (T1) = 1.5Mbps
 - Backbone tipico (T3) = 45Mbps
 - Link ATM = 155Mbps
- Broadcast richiede tipicamente messaggi multipli
- Nodi
 - Spesso una serie di “mainframe”

14

WAN: esempio



15

Protocolli

- L'operazione di scambio di informazioni tra due macchine va oltre la semplice presenza di un collegamento fisico
- Necessario definire
 - Regole relative al formato dei dati
 - Regole relative alla semantica (informazioni di controllo e gestione errori)
 - Regole relative alla tempistica (velocità)

16

Protocolli

- Protocollo = insieme di regole
 - Gestione delle regole tipicamente modulare
 - Protocolli a livelli
- Due tipi principali:
 - Protocolli orientati alla connessione
 - Connessione stabilita in modo esplicito
 - Es.: Telefonata
 - Protocolli privi di connessione
 - Dati scambiati senza accordo preventivo
 - Es.: Lettera

Protocolli

- Due “modelli” di protocollo
 - Modello OSI
 - 7 livelli ben definiti
 - Standard ISO
 - Poco utilizzato in pratica
 - Modello TCP/IP
 - Architettura più flessibile
 - 5 Livelli “concettuali”
 - Sviluppato a partire da protocolli esistenti ai vari livelli
 - Standard di fatto

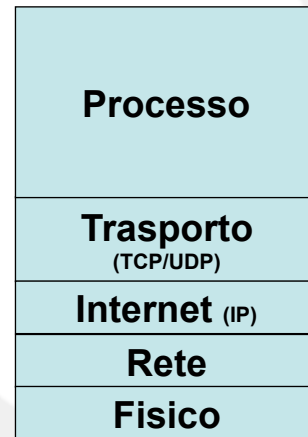
ISO-OSI vs. TCP/IP

OSI

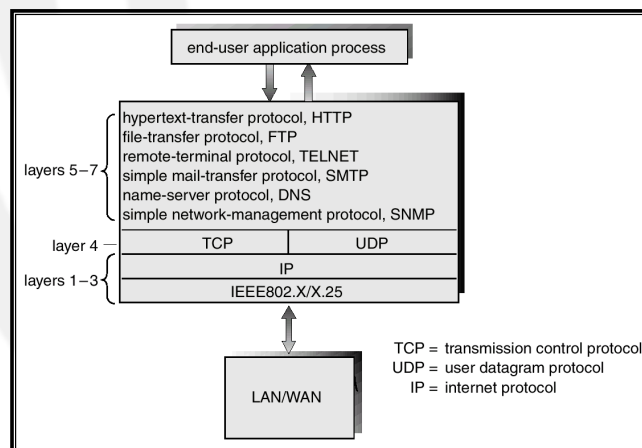


messaggio
pacchetto
frame
bit

TCP/IP



Protocollo TCP/IP: esempio stack



Schemi di comunicazione

- Architettura dei sistemi distribuiti “veri”
 - Tipo WAN
 - Schema basato su protocolli è adatto!
 - Gestione garantita dei vari dettagli
 - Overhead ammortizzato dalle (bassa) velocità della rete

Schemi di comunicazione

- Architettura di sistemi tipo cluster
 - Tipo LAN (esigenze simili)
 - Schema basato su stack (completo) di protocolli è pesante!
 - Necessario uno schema più essenziale (ed efficiente)
 - Overhead non deve penalizzare la velocità della rete
 - Due paradigmi
 - Modello client-server (protocollo richiesta/risposta - request/reply)
 - Chiamata di procedura remota (RPC e varianti)

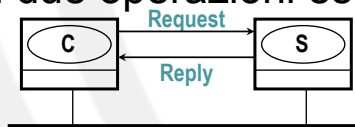
Modello client-server

- Server: programma che fornisce un servizio
 - Rende disponibile delle risorse ad altri programmi che sono eseguiti sulla rete
 - Risorse di vario tipo (DB, file system, stampante...)
 - Server esegue sulla macchina alla quale la risorsa è vincolata ed attende passivamente le richieste
 - Spesso lanciati al boot
- Client: programma che usa una risorsa
 - Può essere eseguito sulla macchina alla quale la risorsa è vincolata o su qualunque altra macchina
 - Effettua una connessione attraverso la rete al server

23

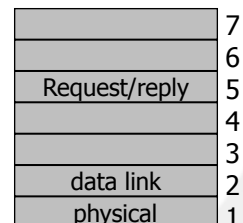
Protocollo richiesta/risposta

- Protocollo connectionless
- Basato su due operazioni essenziali



- In termini di protocolli
 - No routing
 - No connessioni
 - No sessione

Gestiti dall'HW {



24

Modello client-server

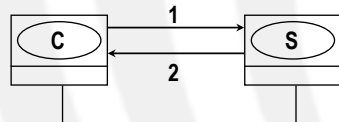
- Problematiche
 - Tipo di indirizzamento
 - Primitive bloccanti e non
 - Primitive bufferizzate e non
 - Primitive affidabili e non

Indirizzamento client-server

- Per spedire un messaggio, il client deve conoscere l'indirizzo del server
- Tre modi:
 - Cablare indirizzo unico nel codice del client
 - Sfruttare broadcast
 - Lookup dell'indirizzo attraverso un name-server

Indirizzamento client-server

1

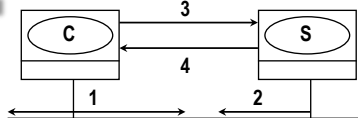


Cabla `macchina.processo` nel codice del client

1: Richiesta a 234.0

2: Risposta a 199.0

2



I server scelgono un numero a caso

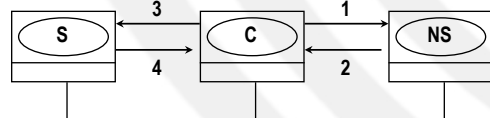
1: Richiesta `broadcast` per conoscere # macchina

2: Sono io!!

3: Richiesta

4: Risposta

3



Il client usa il nome del server

Cerca l'indirizzo a run-time

1: Richiesta indirizzo al `name-server`

2: Risposta del name-server

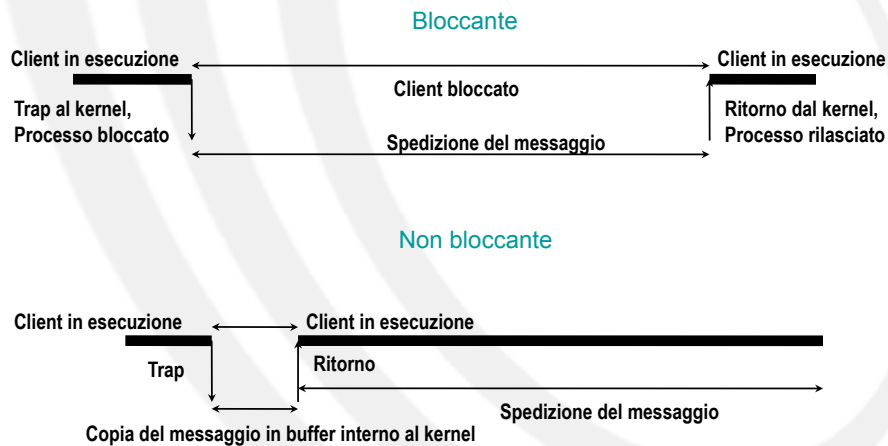
3: Richiesta al server

4: Risposta dal server

Indirizzamento client-server

- Macchina . Processo (cablato nel codice del client)
 - Non trasparente (Se il server cambia macchina?)
- Broadcast
 - Ogni server sceglie un ID univoco
 - Per sapere l'indirizzo della macchina su cui gira il server, il client invia una richiesta broadcast a tutte le macchine
 - La macchina su cui gira il server risponde fornendo il suo indirizzo
 - Oneroso a causa del broadcast
- Name-server
 - Nome del server cablato nel client
 - Il client chiede l'indirizzo reale ad un particolare name-server

Primitive bloccanti vs. non bloccanti



Primitive bloccanti vs. non bloccanti

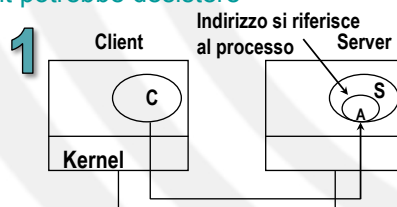
- Esiste sia SEND che RECEIVE bloccante e non bloccante
- SEND bloccante (sincrona)
 - Problema: CPU “sprecata” durante la trasmissione dei messaggi

Primitive bloccanti vs. non bloccanti

- SEND non bloccante (asincrona)
 - Problemi
 - Client non ha la certezza di quando il messaggio sia effettivamente arrivato
 - Quando posso cancellare/modificare il buffer usato per memorizzare il messaggio da spedire?
 - Soluzioni
 - Con copia
 - Buffer copiato su buffer interno al kernel
 - Tempo di CPU sprecato per copia
 - Con interrupt
 - Interrupt al mittente quando il messaggio è stato spedito
 - Difficile da programmare

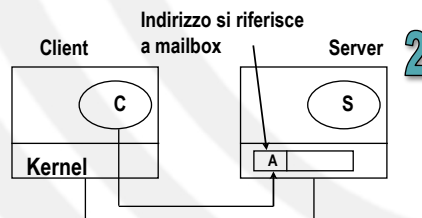
Primitive bufferizzate vs. non bufferizzate

- Non bufferizzate = no memoria locale nel server
 - Indirizzamento al singolo processo
 - Il server esegue receive(address, & message)
 - Problemi in caso di mancato “ascolto” del server
 - Il client deve ritrasmettere nella speranza che il server si metta in ascolto
 - Client potrebbe desistere



Primitive bufferizzate vs. non bufferizzate

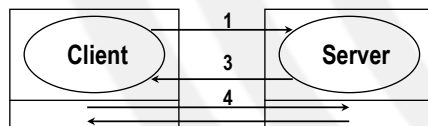
- Bufferizzate
 - Client invia messaggio a mailbox
 - Mailbox simile a una coda limitata
 - Server preleva il primo messaggio dalla mailbox quando desidera
 - Stessi problemi del caso non bufferizzato per mailbox piena



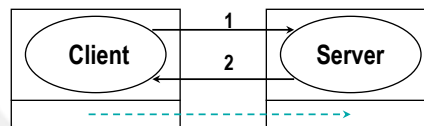
33

Primitive affidabili vs. non affidabili

- Affidabilità influenza semantica scambio messaggi
 - Send non affidabile (utente deve farsi carico del problema)
 - Send con acknowledgement (ACK) bidirezionale
 - Send con acknowledgement (ACK) unidirezionale
 - Risposta del server come ACK (nessuna garanzia)
 - Varianti intermedie



1. Richiesta del client al server
2. ACK del kernel S al kernel C
3. Risposta del server al client
4. ACK del kernel C al kernel S



1. Richiesta del client al server
2. Risposta del server al client
3. ACK del kernel C al kernel S (opzionale)

34

Modello client-server

	Opzione 1	Opzione 2	Opzione 3
Indirizzamento	Macchina.processo	Indirizzo casuale (broadcast)	Name server
Blocco	Primitive bloccanti	Non bloccanti con copia	Non bloccanti con interrupt
Bufferizzazione	NO: Messaggi inaspettati vengono scartati	NO: Messaggi inaspettati conservati per poco tempo	SI: Mailbox
Affidabilità	NO	Req-Ack Reply-Ack	Req-Rep-Ack

35

Modello client-server

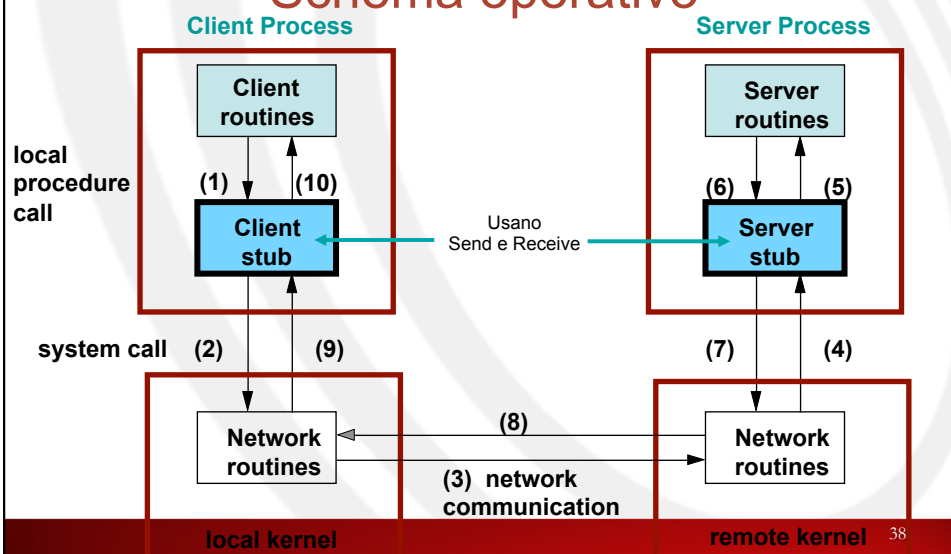
- Paradigma basato su I/O
 - Send e receive
- Diverso rispetto all'uso di servizi su un sistema centralizzato (chiamata a procedura)
- Paradigma alternativo:
 - Chiamate remote a procedura (Remote Procedure Calls)

36

RPC (Remote Procedure Call)

- Un processo su un sistema (locale) invoca una procedura (remota) in modo trasparente
- Utilizza lo schema classico della chiamata a procedura
 - Richiesta: la chiamata a procedura
 - Risposta: il risultato ritornato dalla procedura stessa

Schema operativo



Schema operativo

1. Il client invoca una procedura locale (client stub)
 - apparentemente invoca la procedura remota
 - Stub = procedura remota nello spazio di indirizzamento del client
2. Il client stub:
 - Impacchetta gli argomenti (**marshaling**) per la procedura remota, trasformandoli in un formato “standard”, costruendo uno o più messaggi
 - Chiede al kernel locale (system call) di inviare i messaggi al sistema remoto
 - Esegue *receive* bloccante

39

Schema operativo

3. Messaggi trasferiti sulla rete al sistema remoto
4. Server stub
 - In attesa, riceve la richiesta remota
 - Disimpacchetta (unmarshaling) gli argomenti dal messaggio e li converte nel formato locale
5. Server stub esegue una procedura locale che invoca l'effettiva funzione sul server
6. Procedura server termina e ritorna al server stub i risultati

40

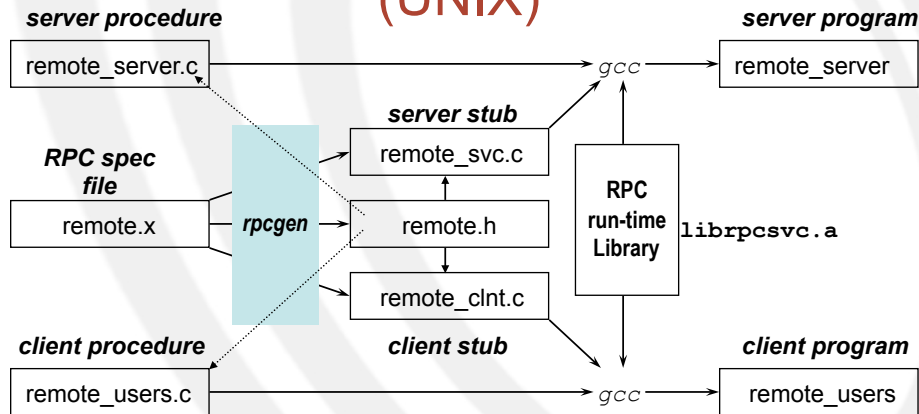
Schema operativo

7. Server stub
 - Converte risultati nel formato “standard” e marshal in messaggi di rete da mandare al client stub
 - Chiede al kernel di inviare i messaggi
 - Esegue *receive* bloccante
8. Messaggi trasferiti sulla rete al client stub
9. Il client stub:
 - In attesa, riceve la risposta
 - Unmarshal dei messaggi e conversione nel formato locale
10. Client stub ritorna alla funzione locale

Binding

- Meccanismo con cui il client individua il server
- Binding dinamico
 - Un server che offre servizio RPC si registra presso un binder
 - Binder = server in esecuzione sullo stesso host del servizio RPC
 - Associato ad una specifica porta (111 in Unix)
 - Registrazione = segnalazione di
 - Identificatore (numero del servizio RPC & versione)
 - Indirizzo di trasporto (porta su cui il servizio RPC è in ascolto)
 - Il client che deve fare una chiamata RPC contatta prima il binder del server per ottenere l'indirizzo corretto a cui la richiesta RPC deve essere mandata
- In UNIX ➡ rpcbind

Applicazioni Client-Server via RPC (UNIX)



43

remote.x, remote server.c

Linguaggio RPC

```

/* remote.x */
program REMOTE_PROG {
    version REMOTE_VERS {
        string REMOTE_USERS(void) = 1;
    } = 1;
} = 0x32345678;
  
```

- Use `rpcgen` to generate `remote.h`, `remote_svc.c`, `remote_clnt.c`
`rpcgen remote.x`
- To compile the server program:
`gcc -o remote_server remote_server.c remote_svc.c -lrpcsvc`
- To compile the client program:
`gcc -o remote_users remote_users.c remote_clnt.c -lrpcsvc`

```

/* remote_server.c - remote procedure;
called by server stub */
#include <stdio.h>
#include <string.h>
#include <utmp.h>
#include <rpc/rpc.h>
#include "remote.h"

char **remote_users_1(void *p, struct svc_req *r)
{
    static char *ptr;
    FILE *fp;
    struct utmp tmp;
    char buffer[1024];
    char tmp_buff[10];

    buffer[0] = '\0';
    fp = fopen("/etc/utmp", "rb");
    while (!feof(fp)){
        fread(&tmp, sizeof(struct utmp), 1, fp);
        if (strlen(tmp.ut_name)){
            sprintf(tmp_buff, "%.8s", tmp.ut_name);
            strcat(buffer, tmp_buff);
        }
    }
    fclose(fp);
    ptr = buffer;
    return(&ptr);
}
  
```

Info su
accessi
utenti

44

remote.h

```

/*
 * Please do not edit this file.
 * It was generated using rpcgen.
 */

#ifndef _REMOTE_H_RPCGEN
#define _REMOTE_H_RPCGEN

#include <rpc/rpc.h>

#define REMOTE_PROG ((unsigned long) (0x32345678))
#define REMOTE_VERS ((unsigned long) (1))
#define REMOTE_USERS ((unsigned long) (1))

extern char ** remote_users_1(void *p, struct svc_req *r);

extern int remote_prog_1_freeresult();

#endif /* !_REMOTE_H_RPCGEN */

```

remote_users.c

```

/* remote_users.c - Client program for the remote users service. */
#include <stdio.h>
#include <rpc/rpc.h>
#include "remote.h"

main(argc, argv)
int argc;
char *argv[];
{
    CLIENT *cl;
    char *server;
    char **sresult;

    if (argc != 2) {
        fprintf(stderr, "usage: %s hostname\n", argv[0]);
        exit(1);
    }
    server = argv[1];

    if ((cl = clnt_create(server, REMOTE_PROG, REMOTE_VERS, "udp")) == NULL) {
        clnt_pcreateerror(server);
        exit(2);
    }

    if ((sresult = remote_users_1(NULL, cl)) == NULL) {
        clnt_perror(cl, server);
        exit(3);
    }
    printf("Users logging on Server %s\n%s\n", server, *sresult);

    clnt_destroy(cl);
    exit(0);
}

```

Problemi implementativi

- Problemi
 - Passaggio di puntatori e array
 - Puntatore = indirizzo in memoria (quale: server o client?)
 - Soluzione
 - Call by copy/restore
 - » Copy solo per parametri che contengono dati inviati dal client necessari al server
 - » Restore solo per parametri che contengono dati modificati dal server necessari al client
 - Funzioni con numero di argomenti variabile
 - Si usa una struttura

Problemi implementativi

- Necessità della definizione della semantica in caso di errore
 - Impossibilità di trovare il server
 - Richieste perse
 - Risposte perse
 - Crash del server
 - Crash del client
- Soluzioni diverse per ogni tipo di errore

Semantica in caso di errore

- Impossibilità di trovare il server
 - Cause: server down, versioni incompatibili ...
 - Codice di errore speciale ➔ ambiguo
 - Es.: -1 può essere confuso con il risultato di un operazione aritmetica
 - Eccezioni/signal ➔ perdita di trasparenza rispetto alle procedure locali
- Risposte perse
 - Uso di timeout associato alla richiesta ➔ possibile solo per richieste idempotenti (ripetibili) (Es.: Non per trasferimenti di denaro!!!)
 - Associazione di numeri progressivi alle richieste se richieste non-idempotenti
- Richieste perse
 - Uso di timeout associato alla richiesta (e ritrasmissione)

Semantica in caso di errore

- Crash del server
 - Equivalente a risposta persa ma ...
 - ... crash prima o dopo dell' esecuzione richiesta?
 - Semantica diversa a seconda del comportamento del client (allo scadere del timeout)
 - At least once: messaggio viene inviato fino a che non si riceve una risposta (1 o più risposte)
 - At most once: eccezione allo scadere del timeout (al max una risposta, ma anche 0)
 - No promise: (0 o più risposte) facile da implementare
 - Exactly once: ideale, non realizzabile
- ➔ Sistemi isolati ≠ sistemi distribuiti
 - Nei primi macchina server = macchina client!

Semantica in caso di errore

- Crash del client
 - Risposta non trova il client ➡ processo sul server orfano
 - Orfani causano problemi
 - Spreco CPU
 - Blocco risorse (es.: file)
 - Client riparte, esegue la richiesta, arriva risposta orfano precedente!
 - ...

Semantica in caso di errore

- Eliminazione degli orfani
 - Basata su log (extermination)
 - Elenco richieste del client mantenuto in un file di log nel client
 - Al reboot del client orfani vengono uccisi
 - Basata sull'utilizzo di "epoche" (reincarnation)
 - Ogni reboot di un client inizia una nuova epoca, inviata a tutti gli host in broadcast
 - Orfani corrispondenti ad altre "epoche" vengono terminati
 - Basata su timeout (expiration)
 - Ad ogni RPC viene associato un tempo massimo di completamento
- Non esiste una soluzione ottima

Varianti di RPC

- RPC = schema base di comunicazione
- Altre varianti
 - RMI (Java): versione ad oggetti di RPC
- Modelli alternativi
 - Basati su tecnologie OO
 - Concetto di object broker (OB)
 - Sorta di elenco di servizi disponibili nella rete
 - Comunicazioni tra client e server passano attraverso l'OB
 - Esempi
 - Microsoft DCOM (OLE)
 - OMG CORBA

Comunicazione di gruppo

- Spesso utile/necessario comunicare non tra processi singoli ma tra gruppi di processi
 - Es.: un gruppo di file server cooperanti che offrono un unico servizio
- Paradigma RPC intrinsecamente 1 a 1!
- Concetto di gruppo è dinamico
 - Processi entrano ed escono dal gruppo
 - Processi possono appartenere a più gruppi
- Gruppo è visto come singola entità

Comunicazione di gruppo

- Problematiche
 - Tipo di comunicazione
 - Struttura dei gruppi
 - Regole di appartenenza al gruppo
 - Indirizzamento di gruppo
 - Atomicità
 - Ordine dei messaggi

Tipo di comunicazione

- Dipende dal supporto HW
- Multicasting
 - Indirizzo di rete su cui più host possono “ascoltare”
 - Comunicazione di gruppo tramite un indirizzo multicast per ogni gruppo
- Broadcasting
 - Indirizzo di rete su cui tutti gli host possono “ascoltare”
 - Comunicazione di gruppo tramite filtraggio dei pacchetti
- Unicasting
 - Assenza di multi-/broadcast
 - Comunicazione di gruppo tramite invio selettivo ai membri

Struttura dei gruppi

- Gruppi chiusi vs. aperti
 - Gruppi chiusi
 - solo i membri possono mandare messaggi al gruppo
 - I non membri possono mandare messaggi a singoli membri
 - Es.: calcolo parallelo – no interazione col mondo esterno
 - Gruppi aperti = non chiusi
 - Es.: gruppo di server replicati
 - client devono poter parlare con il gruppo
 - server devono poter comunicare tra loro

57

Struttura dei gruppi

- Gruppi paritari (peer) vs. gerarchici
 - Gruppi peer
 - Decisioni “collegiali”, non c'è un capo
 - Vantaggio: robustezza, se cade un processo non è un dramma
 - Svantaggio: complicato prendere decisioni → overhead x votazione
 - Gruppi gerarchici
 - C'è un coordinatore che distribuisce il “lavoro” tra i worker
 - Scarsa tolleranza ai guasti, se cade il coordinatore ... dramma
 - Decisioni semplificate (coordinatore può decidere da solo)

58

Regole di appartenenza al gruppo

- Meccanismo di entrata/uscita da un gruppo
 - Tramite group server
 - Tiene traccia di tutti i gruppi e di chi vi appartiene
 - Semplice ed efficiente ma ...
 - ... centralizzato → scarsa tolleranza ai guasti, se cade ...
 - Gestione distribuita
 - Entrata: messaggio di richiesta
 - Uscita: messaggi di addio a tutti

Regole di appartenenza al gruppo

- Problemi
 - Crash di un membro del gruppo → uscita senza messaggio
 - Gli altri se ne accorgono dopo un pò
 - Sincronizzazione tra entrata/uscita dal gruppo e ricezione dei messaggi
 - Quando P entra deve iniziare a ricevere tutti i messaggi inviati da altri dopo la sua entrata
 - Quando P esce non deve ricevere/spedire + nulla da/per gli altri

Indirizzamento di gruppo

- Tramite il supporto del S.O. e dell'HW
 - Multicast vs. broadcast vs. unicast
 - Trasparente rispetto al sender
- Tramite esplicita lista di destinazioni
 - Lista di indirizzi associata al messaggio
 - Non trasparente (serve conoscere chi appartiene al gruppo)
- Predicate addressing
 - Tipo broadcast, ma selettivo
 - Ogni messaggio contiene un predicato che deve essere valutato dall'host che lo riceve (Vero = accettato, Falso = respinto)
 - Es.: invio di un msg a tutte le macchine per chiedere a quelle con almeno 4MB di memoria libera di eseguire un processo

Atomicità

- Garanzia che un messaggio inviato ad un gruppo sia ricevuto o da tutti o da nessuno (es.: database replicato)
- Difficile da realizzare
 - Utilizzo di acknowledgement efficace solo se gli host del gruppo sono tutti attivi
 - Se il mittente cade e alcuni destinatari hanno perso il msg?
 - Soluzione possibile
 - Il processo manda un messaggio al gruppo e inizializza un timer
 - Ogni processo che riceve un messaggio "non già ricevuto", lo rimanda a tutti i membri del gruppo

Ordine dei messaggi

- Ordine deve essere uguale per tutti i membri!
- Problema
 - Mezzo di comunicazione spesso non deterministico (es. LAN)
- Soluzioni
 - Ordine di tempo globale: l'ordine vero (difficile)
 - Ordine di tempo consistente: non quello vero, ma uno consistente (uguale per tutti)
 - Ordine causale: ordine consistente solo per messaggi tra i quali esiste una relazione di causalità (irrelevante per gli altri)

63

Riepilogo

- Sistema operativo distribuito
 - Concetti applicabili a
 - Sistemi distribuiti “veri”
 - Sistemi tipo cluster
- Problematiche
 - Comunicazione basata su “scambio di messaggi”
 - Diverse esigenze tra sistemi con diversi supporti di comunicazione (WAN vs. LAN)
 - Protocolli classici vs. protocolli semplificati
 - Client/server
 - RPC
 - Comunicazione di gruppo

64