Sistemi Distribuiti - Lezioni

Sara Angeretti
@Sara1798

2022/2023

Indice

Capitolo 1

Introduzione ai S. D.

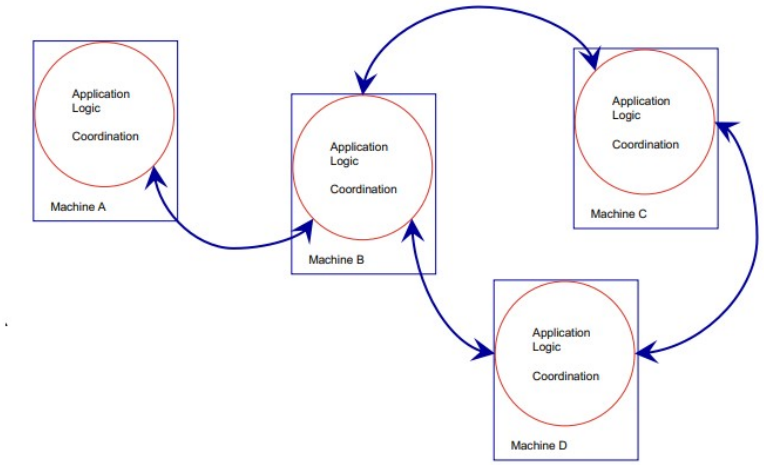
Cominciamo ad introdurre i seguenti argomenti:

- Definizione di sistema distribuito
- Architetture software
- Il modello client-server
- Proprietà e caratteristiche fondamentali

1.1 Alcune definizioni

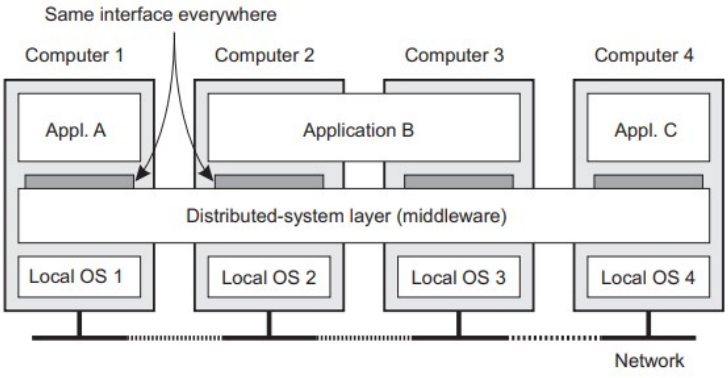
1.1.1 Definizione 1

Ci sono diverse definizioni con alcuni aspetti in particolare in comune.
Il libro di testo definire un *sistema distribuito* come un sistema in cui componenti hardware o software che sono localizzati in un sistema collegato alla rete, **comunicano** e **coordinano** le loro azioni solamente (**"only by"**) passandosi messaggi.
In inglese: "We define a distributed system as one in which **hardware or software components** located at **networked computers communicate** and **coordinate** their actions ***only by*** **passing messages**."
Ricorda che stiamo parlando di **processi**, anche se ci sono processi che operano senza una rete ma sono eccezioni.



1.1.2 Definizione 2

Un'altra definizione è la seguente.
Un *sistema distribuito* è definito come un sistema di **elementi computativi autonomi** (che coesistono) che appaiono ad un utente come un'unica applicazione, un **unico sistema coerente**.
In inglese: "A distributed system is a collection of **autonomous computing elements** that appears to its users as a **single coherent system**."



1.1.3 In sintesi

Definizione:

- Un sistema distribuito è definito come un sistema di elementi computativi autonomi che coesistono e che appaiono ad un utente come un'unica applicazione, un unico sistema coerente.

Caratteristiche tipiche:

- Elementi computativi autonomi, anche noti come *nodi*; composti da device hardware o processi software
- Unico sistema coerente: gli utenti o le applicazioni percepiscono un singolo sistema

⇒ i nodi **devono collaborare**.

1.2 Sistemi come collezioni di nodi autonomi

1.2.1 Comportamenti

Sono **autonomi** e **indipendenti**, ovvero possono progredire come vogliono, ognuno ha la propria concezione del tempo ⇒ *non* c'è un clock globale, *non* è tutto sincronizzato.
Tutto ciò porta a *fondamentali problemi* di **sincronizzazione** e **coordinazione**.

1.2.2 Collezioni di nodi

Come gestire **appartenenze di gruppo** (o *group membership*)?
I **gruppi** possono essere **aperti/dinamici** (qualunque nodo può partecipare) o **chiusi/fissi** (solo nodi ben selezionati possono entrare nel sistema, questa nozione verrà commentata ulteriormente e comunque non troppo a fondo).
Ma quindi, come faccio a sapere se il nodo con cui sto comunicando è effettivamente **autorizzato**? Non ha proprio risposto.

1.3 Sistemi coerenti

1.3.1 Essenzialmente

La collezione di nodi opera nello stesso modo indipendentemente da dove, quando e come avvengono le interazioni fra l'utente e il sistema.
Esempi

- Un utente finale (end user) non può dire dove stia avvenendo una computazione
- Dove i dati sono collezionati e stipati non dovrebbe essere rilevante per un'applicazione
- Se i dati siano stati replicati o meno è completamente nascosto

1.3.2 Trasparenza di come caratteristica fondamentale dei sistemi distribuiti

"**Trasparenza di distribuzione (?)**" come parola chiave. Da tradurre. Il problema principale: **fallimenti parziali**.

- È inevitabile che in qualsiasi momento solo una parte limitata del sistema distribuito fallisca.
- Nascondere fallimenti parziali e il loro ripristino è spesso molto complicato e generalmente impossibile da nascondere.

1.4 Sintesi delle caratteristiche di un sistema distribuito

Caratteristiche fondamentali per tutti i sistemi distribuiti:

Gestione della memoria?

- **Non c'è memoria condivisa**
- Comunicazione via scambio messaggi
- Non c'è stato globale: ogni componente (nodo, processo) conosce solo il proprio stato e può sondare lo stato degli altri.

Gestione dell'esecuzione?

- **Ogni componente è autonomo** =¿ esecuzione concorrente
- Il coordinamento delle attività è importante per definire il comportamento di un sistema/applicazione costituita da più componenti

Gestione del tempo (temporizzazione)?

- **Non c'è un clock globale**
- Non c'è possibilità di controllo/scheduling globale
- Solo coordinamento via scambio messaggi

Tipi di fallimenti?

- **Fallimenti indipendenti** dei singoli nodi (independent failures)
- Non c'è fallimento globale

Capitolo 2

Architetture Software

2.1 Definizione di architetture software

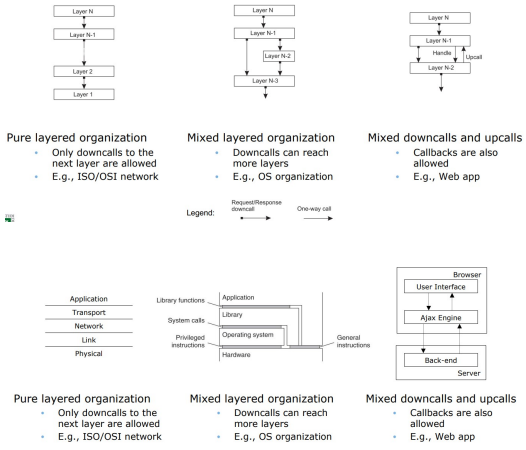
Un'architettura software definisce la struttura del sistema, le interfacce tra i componenti e i pattern di interazione (i protocolli).
I sistemi distribuiti possono essere organizzati secondo **diversi stili architetturali**.

2.2 Modello base: Architetture a strati (layered)

- Sistemi operativi
- Middleware

2.2.1 Definizione

Un'*architettura a strati* è un'architettura software che *organizza il software in **strati***.
Ogni strato è *costruito sopra uno strato diverso più generico*.
Uno strato può essere definito liberamente insieme di (sotto)sistemi con lo *stesso grado di generalità*.
Gli strati più alti sono più specifici per applicazioni e i più bassi sono più generali/generici.



Quella al centro è da sapere benissimo in quanto oggetto di questo insegnamento.

2.3 Architetture a livelli (tier)

- Le applicazioni client server (2-tier, 3-tier)

2.4 Architetture basate sugli oggetti

- Java-Remote Method Invocation (RMI)

2.5 Architetture centrate sui dati

- Il Web come file system condiviso

2.6 Architetture basate su eventi

- Applicazioni Web dinamiche basate su callback (AJAX)

2.6.1 Sistemi Operativi Ditribuiti

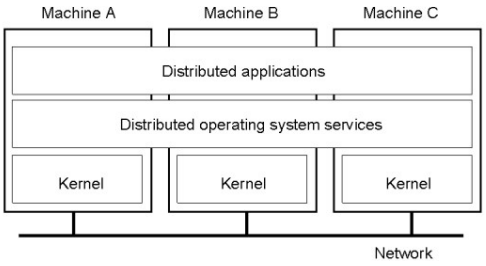
Diversi tipi:

- DOS (Distributed Operating Systems)
- NOS (Network Operating Systems)

- Middleware

System	Description	Main Goal
DOS	Tightly-coupled operating system for multi-processors and homogeneous multicomputers	Hide and manage hardware resources
NOS	Loosely-coupled operating system for heterogeneous multicomputers (LAN and WAN)	Offer local services to remote clients
Middleware	Additional layer atop of NOS implementing general-purpose services	Provide distribution transparency

DOS (Distributed Operating Systems)



Users not aware of multiplicity of machines

- Access to remote resources like access to local resources

Data Migration

- Transfer data by transferring entire file, or transferring only those portions of the file necessary for the immediate task

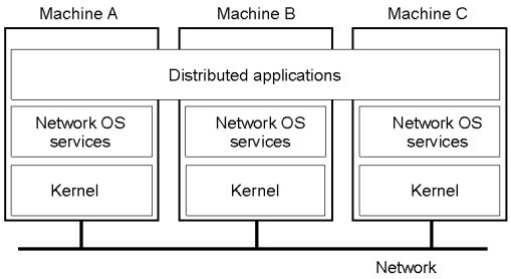
Computation Migration

- Transfer the computation, rather than the data, across the system

Process Migration - execute an entire process, or parts of it, at different sites

- Load balancing - distribute processes across network to even the workload
- Computation speedup - subprocesses can run concurrently on different sites
- Hardware preference - process execution may require specialized processor
- Software preference - required software may be available at only a particular site
- Data access - run process remotely, rather than transfer all data locally

NOS (Network Operating Systems)



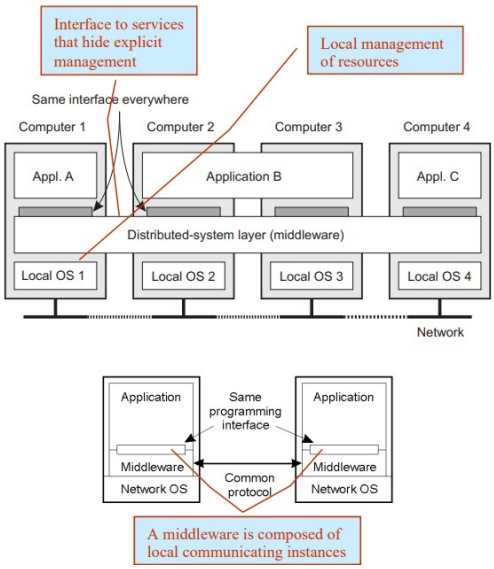
Users are aware of multiplicity of machines
NOS provides explicit communication features

- Direct communication between processes (socket)
- Concurrent (i.e., independent) execution of processes that from a distributed application
- Services, such as process migration, are handled by applications

Access to resources of various machines is done explicitly by:

- Remote logging into the appropriate remote machine (telnet, ssh)
- Remote Desktop (Microsoft Windows)
- Transferring data from remote machines to local machines, via the File Transfer Protocol (FTP) mechanism

Middleware



Distributed Operating Systems

- Make services (e.g., data storage and process execution) **transparent** to applications
- Rely on homogeneous machines (since they need to run the same software)

Network Operating Systems

- Services (e.g., data storage and process execution) **are explicitly managed** by applications
- Do not require homogeneous machines (since they may run different software)
- E.g., MacOSX, Windows10, Linux

Middleware

- Implements services (one or more) to **make them transparent** to applications
- E.g., Java/RMI

è importante capire che nel secondo schema dell'immagine, il middleware simula il comportamento dell'applicazione, ma i due middleware sono uguali ma possono comunicare tramite protocolli.

Servizi Middleware

Services can address several issues, from general to domain specific.
Naming (il più importante) ovvero come faccio ad identificare un sistema operativo: astrazione

- Symbolic names are used to identify entities that are part of a DS
- They can be used by registries to provide the real addresses (e.g., DNS, RMI registries), or implicitly by the middleware

Access transparency (il più importante)

- ... defines and offers a communication model that hides details on message passing

Persistence

- ... defines and offers an automatic service for data storage (on file system or DB)

Distributed transactions (non vedremo tanto a fondo)

- ... defines and offers a persistence models to automatically ensure consistency on read/write operations (usually on DBs)

Security (non vedremo tanto a fondo)

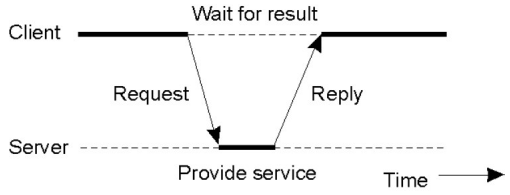
- ... defines and offers models to protect access to data and services (with different levels of permissions) and computation integrity

Item	Distributed OS		Network OS	Middleware-based OS
	Multiproc.	Multicomp.		
Degree of transparency	Very High	High	Low	High
Same OS on all nodes	Yes	Yes	No	No
Number of copies of OS	1	N	N	N
Basis for communication	Shared memory	Messages	Files	Model specific
Resource management	Global, central	Global, distributed	Per node	Per node
Scalability	No	Moderately	Yes	Varies
Openness	Closed	Closed	Open	Open

Capitolo 3

Il modello Client-Server

Il modello Client-Server è il modello di interazione tra un processo client e un processo server.



3.1 Visto ieri: Sintesi caratteristiche di un s.d.

Caratteristiche fondamentali per tutti i sistemi distribuiti:

Gestione della memoria?

- **Non c'è memoria condivisa**
- Comunicazione via scambio messaggi
- Non c'è stato globale: ogni componente (nodo, processo) conosce solo il proprio stato e può sondare lo stato degli altri.

Gestione dell'esecuzione?

- **Ogni componente è autonomo** =¿ esecuzione concorrente
- Il coordinamento delle attività è importante per definire il comportamento di un sistema/applicazione costituita da più componenti

Gestione del tempo (temporizzazione)?

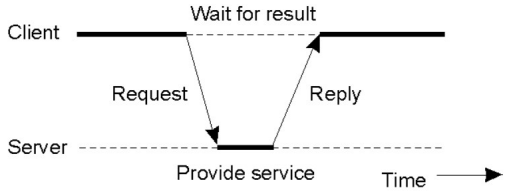
- **Non c'è un clock globale**
- Non c'è possibilità di controllo/scheduling globale
- Solo coordinamento via scambio messaggi

Tipi di fallimenti?

- **Fallimenti indipendenti** dei singoli nodi (independent failures)
- Non c'è fallimento globale

3.2 Il modello Client-Server

Il modello Client-Server è il modello di interazione tra un processo client e un processo server.



C'è uno strato verticale e uno orizzontale(?)

Configurazioni client/server

- Accesso a server multipli
- Accesso via proxy

3.3 Caratteristiche problematiche di ogni sistema distribuito

N.B.: (molto) probabile domanda d'esame.
Tutti i sistemi distribuiti vanno incontro a 4 problemi fondamentali che devono saper risolvere.
Vari step di risoluzione:

Identificare la controparte : fase di **naming**, dove assegnamo a un identificativo che deve necessariamente essere **univoco**;

Accedere alla controparte : fase di **access point**, una *reference* a cui possiamo fare riferimento;

Comunicazione 1 : fase di **protocol**, dove bisogna accordarsi su un formato condiviso di comunicazione (*ricevere l'informazione*);

Comunicazione 2 : questo è ancora un **open issue**, dove bisogna accordarsi su una convenzione di significato (*capire l'informazione*).

3.4 Trasparenza di distribuzione

Def.: consiste nel nascondere dettagli agli utenti, che ignorano cosa succede e (più importante) non possono influenzare il servizio fornito.

Naming
- Symbolic names are used to identify resources that are part of a distributed system

Access transparency
- Hide differences in data representation and how a local or remote resource is accessed

Location transparency
- Hide where a resource is located in the net

Relocation or mobility transparency
- Hide that a resource may be moved to another location while in use

Migration transparency
- Hide that a resource may move to another location

Replication transparency
- Hide that a resource is replicated

Concurrency transparency
- Hide that a resource may be shared by several independent users (ensuring state consistency)

Failure transparency
- Hide the failure and recovery of a resource.

Persistence transparency
- Hide that a resource is volatile or stored permanently

3.5 Le basi dei sistemi distribuiti

3.5.1 Il concetto di protocollo

Per poter capire le richieste e formulare le risposte i due processi devono concordare un **protocollo**.
I protocolli definiscono il **formato**, l'**ordine** di invio e di ricezione dei messaggi tra i dispositivi, il **tipo dei dati** e le **azioni** da eseguire quando si riceve un messaggio.
Le applicazioni su TCP/IP:

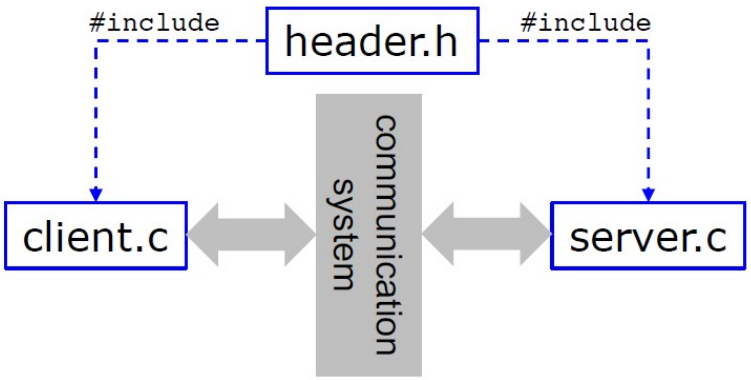
- si scambiano ***stream di byte*** di lunghezza infinita (il **meccanismo**)
- che possono essere segmentati in ***messaggi*** (la **politica**) definiti da un protocollo condiviso

Esempi di protocollo applicativi

- HTTP - HyperText Transfer Protocol
- FTP - File Transfer Protocol
- SMTP - Simple Mail Transfer Protocol

3.5.2 Elementi minimi per creare un'applicazione

Definizione del protocollo di comunicazione.
Condivisione del protocollo tra gli attori dell'applicazione.
Un esempio in C:



Esempi: file server remoto

Il file header.h definisce il protocollo che usano sia il client sia il server.

```
/* Definitions needed by clients and servers. */
#define TRUE 1
#define MAX_PATH 255          /* maximum length of file name */
#define BUF_SIZE 1024         /* how much data to transfer at once */
#define FILE_SERVER 243       /* file server's network address */

/* Definitions of the allowed operations*/
#define CREATE 1              /* create a new file */
#define READ 2                /* read data from a file and return it */
#define WRITE 3               /* write data to a file */
#define DELETE 4              /* delete an existing file */

/* Error codes. */
#define OK 0                  /* operation performed correctly */
#define E_BAD_OPCODE -1       /* unknown operation requested */
#define E_BAD_PARAM -2        /* error in a parameter */
#define E_IO -3               /* disk error or other I/O error */

/* Definition of the message format. */
struct message {
  long source;                /* sender's identity */
  long dest;                  /* receiver's identity */
  long opcode;                /* requested operation */
  long count;                 /* number of bytes to transfer */
  long offset;                /* position in file to start I/O */
  long result;                /* result of the operation */
  char name[MAX_PATH];        /* name of file being operated on */
  char data[BUF_SIZE];        /* data to be read or written */
};
```

Struttura di un semplice server che realizza un rudimentale file server remoto.

```
1.  #include <header.h>
2.  void main(void) {
3.    struct message m1, m2;                          /* incoming and outgoing messages */
4.    int r;                                          /* result code */

5.    while(TRUE) {                                   /* server runs forever */
6.      receive(FILE_SERVER, &m1);                    /* block waiting for a message */
7.      switch(m1.opcode) {                           /* dispatch on type of request */
8.        case CREATE: r = do_create(&m1, &m2); break;
9.        case READ: r = do_read(&m1, &m2); break;
10.       case WRITE: r = do_write(&m1, &m2); break;
11.       case DELETE: r = do_delete(&m1, &m2); break;
12.       default: r = E_BAD_OPCODE;
13.     }

14.   m2.result = r;                                  /* return result to client */

15.   send(m1.source, &m2);                           /* send reply */
16.   }
17. }
```

Un client che usa il servizio per creare una copia di un file

```
1.  #include <header.h>
2.  int copy( char *src, char *dst){                  /* procedure to copy file using the server */
3.    struct message m1;                              /* message buffer */
4.    long position;                                  /* current file position */
5.    long client = 110;                              /* client's address */
6.    initialize();                                   /* prepare for execution */
7.    position= 0;

8.    do {
9.      m1.opcode = READ;                             /* operation is a read */
10.     m1.offset = position;                         /* current position in the file */
11.     m1.count = BUF_SIZE;                          /* how many bytes to read */
12.     strcpy(&m1.name, src);                        /* copy name of file to be read to message */
13.     send(FILE_SERVER, &m1);                       /* send the message to the file server */
14.     receive(client, &m1);                         /* block waiting for the reply */

15.     /* Write the data just received to the destination file */
16.     m1.opcode = WRITE;                            /* operation is a write */
17.     m1.offset =position;                          /* current position in the file */
18.     m1.count = m1.result;                         /* how many bytes to write */
19.     strcpy(&m1.name, dst);                        /* copy name of file to be written to buf */
20.     send(FILE_SERVER, &m1);                       /* send the message to the file server */
21.     receive(client, &m1);                         /* block waiting for the reply */
22.     position += m1.result;                        /* m1.result is number of bytes written */
23.   } while(m1.result > 0);                         /* iterate until done */

24.   return(m1.result >= 0 ? OK : m1 result);        /* return OK or error code */
25. }
```

Capitolo 4

Stream-oriented communication - Le socket

4.1 Contenuti sintetici

Breve ripasso del modello ISO/OSI per TCP/IP
Identificazione dei processi Indirizzi IP e Porte L'interfaccia API per le socket
Le socket in Java
I modelli architatturali • Iterativo • Concorrente mono processo • Concorrente multi processo

4.2 Modello ISO/OSI

4.3 Comunicazione fisica - layering

4.4 Network edge

Chi è che si parla? I **processi**.

4.5 Processi e programmi

I programmi vengono eseguiti dai processi.

- Programma = sequenza di istruzioni eseguibili dalla "macchina"

I processi sono entità gestite dal Sistema Operativo

- Processo = area di memoria RAM per effettuare le operazioni e memorizzare i dati + registro che ricorda la prossima istruzione da eseguire + canali di comunicazione

Ogni processo comunica attraverso canali.

- Un canale gestisce flussi di dati in ingresso e in uscita (dati in formato binario o testuale)
- Per esempio lo schermo, la tastiera e la rete sono "canali"
- Dall'esterno ogni canale è identificato da un numero intero detto "porta"

Le socket sono particolari canali per la comunicazione tra processi che non condividono memoria (per esempio perché risiedono su macchine diverse). Per potersi connettere o inviare dati ad un processo A, un processo B deve conoscere la macchina (host) che esegue A e la porta cui A è connesso (wellknown port).

4.6 I servizi di trasporto Internet

Servizio TCP
- *Orientato alla connessione*: il client invia al server una richiesta di connessione
- *Trasporto affidabile (reliable transfer)* tra processi mittente e ricevente
- *Controllo di flusso (flow control)*: il mittente rallenta per non sommergere il ricevente
- *Controllo della congestione (congestion control)*: il mittente rallenta quando la rete è sovraccarica
- *Non offre* garanzie di banda e ritardo minimi

Servizio UDP
- Trasporto non affidabile tra processi mittente e ricevente
- Non offre connessione, affidabilità, controllo di flusso, controllo di congestione, garanzie di ritardo e banda

D: perché esiste UDP?

Può essere conveniente per le applicazioni che tollerano perdite parziali (es. video e audio) a vantaggio delle prestazioni

4.7 Politiche dei servizi TCP/UDP

Servizio UDP:

- Scompone il flusso di byte in segmenti
- Li invia, uno per volta, ai servizi network

Servizio TCP:

- Scompone e invia come UDP
- Ogni segmento viene numerato per garantire:
  • Riordinamento dei segmenti arrivati
  • Controllo delle duplicazioni (scarto i segmenti con ugual numero d'ordine)
  • Controllo delle perdite (rinvio i segmenti mancanti)
- Per progettare e realizzare sistemi distribuiti
  • NON è necessario conoscere il funzionamento (information hiding) dei processi
  • Ciò che importa è lo scambio dati (stream di byte) tra i processi

4.8 Socket: funzionamento di base

TCP

- Utilizza variabili e buffer per realizzare il trasferimento bidirezionale di flussi di bytes ("pipe") tra processi
- Prevede ruoli client/server durante la connessione
- NON prevede ruoli client/server per la comunicazione
- Utilizza i servizi dello strato IP per l'invio dei flussi di bytes

API: Application Programming Interface

- Definisce l'interfaccia tra applicazione e strato di trasporto

Socket: API per accedere a TCP e UDP

- Due processi (applicazione nel modello client server) comunicano inviando/leggendo dati in/da socket

4.9 Aspetti critici

Gestione del ciclo di vita di client e server • Attivazione/terminazione del cliente e del server (es. Manuale o gestita da un middleware) ¨ Identificazione e accesso al server • Informazioni che deve conoscere il cliente per accedere al server ¨ Comunicazione tra cliente e server • Le primitive disponibili e

le modalità per la comunicazione (es. TCP/IP: Stream di dati inviati con send/receive) ¨ Ripartizione dei compiti tra client e server • Dipende dal tipo di applicazione (es. controllo: una banca gestisce tutto lato server) • Influenza la prestazioni in relazione al carico (numero di clienti)

4.10 Identificare il server

Come fa il client a conoscere l'indirizzo del server? ¨ Alternative: • inserire nel codice del client l'indirizzo del server espresso come costante (es. il client di un servizio bancario) • chiedere all'utente l'indirizzo (es. web browser) • utilizzare un name server o un repository da cui il client può acquisire le informazioni necessarie (es. Domain Name Service - DNS - per tradurre nomi simbolici) • adottare un protocollo diverso per l'individuazione del server (es. broadcast per DHCP)

4.11 Problemi fondamentali e TCP/IP

Come sono trattate le 4 problematiche fondamentali dei sistemi distribuiti con TCP e IP?

Identifico la controparte (naming):

Low level identification: the name of hosts and protocols

Accedo alla controparte (access point):

Use of the IP address (host:port) to access a process

Comunicazione 1 (protocollo):

Stream of bytes

Comunicazione 2 (sintassi e semantica):

Application protocols with predefined semantics (http, smtp)

What level of transparency? Very low: the programmer/user need to

- know network addresses

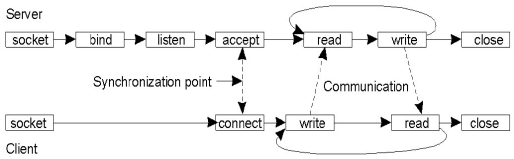
- parse bytes to get the content (message)

IMPORTANTE: TCP si può usare per qualsiasi tipo di comunicazione? Perché non c'è semantica. Quindi il punto di Comunicazione 2 non c'è. Questa potrebbe essere una domanda dell'esame.

4.12 Comunicazione via socket

La comunicazione TCP/IP avviene attraverso flussi di byte (byte stream), dopo una connessione esplicita, tramite normali system call read/write.
Read e write:

- Sono sospensive (bloccano il processo finché il sistema operativo non ha effettuato la lettura/scrittura)
- Utilizzano un buffer per garantire flessibilità (es: la read definisce un buffer per leggere N caratteri, ma potrebbe ritornare avendone letti solo k¡N)



4.13 API socket system calls (Berkeley)

Many calls are provided to access TCP and UDP services.
The most relevant ones are in the table below:

Primitive	Meaning
Socket	Create a new communication endpoint
Bind	Attach a local address to a socket, set the queue length
Listen	Announce willingness to accept connections
Accept	Block caller until a connection request arrives
Connect	Actively attempt to establish a connection
Write	Send some data over the connection
Read	Receive some data over the connection
Close	Release the connection