

PARTE F

Socket programming e livello trasporto

Modulo 1: Tool livello trasporto

Comunicazioni di rete

- **Per comunicare in rete, un'applicazione utilizza i protocolli implementati dal sistema operativo**
 - TCP
 - UDP
 - ICMP / IP (raramente, richiede privilegi particolari)
- **Le *socket*** sono l'astrazione attraverso cui l'applicazione può accedere a tali protocolli
 - la ***API*** definita dalle ***socket BSD*** è quella da hanno avuto origine le attuali system call disponibili nei sistemi operativi Unix/Linux

Socket statistics - ss

- **Socket statistics (ss)** è una delle applicazioni che **permette di visualizzare e analizzare le socket utilizzate dal sistema**
- **Sostituisce il precedente comando *netstat***, comunque ancora spesso disponibile sulle distribuzioni GNU/Linux
- **connessioni TCP aperte**
- **porte in ascolto**
- **numerosi dettagli sulle informazioni di cui sopra, ad esempio:**
 - protocollo impiegato
 - processi che stanno gestendo le connessioni
 - indirizzi IP e porte in uso
- **possibilità di eseguire query complesse**

Uso di *ss*

- `ss [-tuelanp] [query]`
 - t → mostra connessioni TCP
 - u → mostra connessioni UDP
 - e → mostra informazioni aggiuntive
 - l → mostra le socket in stato “listen”
 - a → mostra le socket in qualsiasi stato esse siano
 - n → non effettua la risoluzione DNS degli indirizzi e delle porte “note”
 - p → mostra il programma associato alla connessione

Netcat

- **Netcat (nc) è un comando che permette di scambiare messaggi arbitrari impiegando i protocolli TCP e UDP.**
- ***Client:* aprire una connessione verso un server**

```
nc [-options] <hostname> <port>
```

***Server:* aprire una connessione in ascolto (server)**

```
nc [-options] -l -p <port>
```

- *NB: nell'implementazione BSD la sintassi e alcune funzionalità potrebbero essere diverse (ad esempio, non è richiesto l'uso di -p per indicare la porta lato server)*

Alcune delle opzioni più utilizzate:

- ***Sia client che server:***

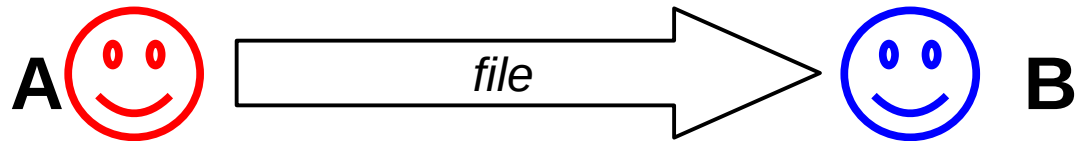
- `-u` : utilizza il protocollo UDP al posto di TCP
- `-v` : verbose output
- `-n` : non effettua risoluzioni o reverse lookup DNS
- `-U` : impiega una socket unix

- ***Solo client:***

- `-s <ipaddress>` : forza l'indirizzo IP sorgente indicato
- `-p <port>` : forza l'uso della porta sorgente indicata

Trasferire file con Netcat

Alcuni esempi d'uso per inviare file tramite netcat:



- Usare la ridirezione I/O per inviare file in rete

B: `nc -l -p <port> > <new_filename>`

A: `nc <ip-B> <port> < <filename>`

- Usare pipes per ridirezione gli output di altri comandi

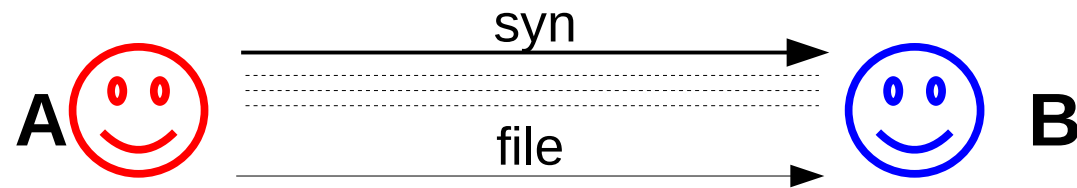
A: `cat <filename> | nc <ip-B> <port>`

A: `tar cz <filename> | nc <ip-B> <port>`

B: `nc -l -p <port> | tar xz`

- *NB: si può usare l'opzione **-q** <n> per chiudere la connessione dopo <n> secondi dal momento in cui nc incontra un EOF, ma potrebbe non servire con alcune implementazioni di nc*

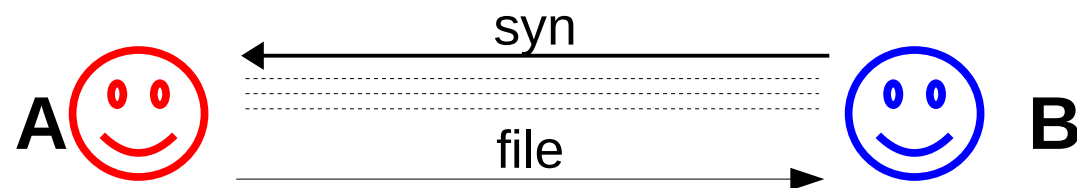
Trasferire file con Netcat



- I casi precedenti assumono la presenza di un server nc (o analogo) in ascolto. Ovvero, chi inizializza la connessione è chi sta per inviare il file. Per invertire lo situazione (comodo se chi deve ricevere è sotto *nat*):

A: nc -q <n> -l -p <port> < <filename>

B: nc -q <n> <ip-A> <port> > <new_filename>



Creazione di un file

- **Usiamo il comando dd**

```
dd if=/dev/zero of=file.bin bs=1024 count =1024
```

- **Abbiamo creato un file di 1MB pieno di byte con valore 0**

```
ls -lh
```

```
-rw-r--r-- 1 root root 1.0M [...] file.bin
```

Copia da H1 a H2

- **Chiamiamo fileH1.bin il file su H1**
- **Su H2 mettiamo il server in ascolto**
`nc -l -p 8080 | tar xzv &`
- **Controlliamo di avere il server in ascolto**
`ss -ltn`
- **Facciamo partire il trasferimento da H1**
`tar cz fileH1.bin | nc 2.2.2.2 8080 -q1`
- **Verifichiamo il risultato**
 - Su H2 compare il file fileH1.bin
 - Con il comando `md5sum` si vede che i file sono identici

Copia da H2 a H1

- **Chiamiamo fileH2.bin il file su H2**
- **I ruoli di client e server devono restare gli stessi, ma con flusso dati inverso**
- **Su H2 mettiamo il server in ascolto**

```
nc -l -p 8080 -c 'tar cz fileH2.bin'&
```

- **Facciamo partire il trasferimento da H1**

```
nc 2.2.2.2 8080 | tar xz
```

- **Verifichiamo il risultato**
 - Su H1 compare il file fileH2.bin

Modulo 2: Introduzione al Socket programming

Interazione protocollo di trasporto con applicazioni

- **Il client ed il server utilizzano un protocollo di trasporto (TCP o UDP) per comunicare**
- **Il software di gestione del protocollo di trasporto si trova all'interno del Sistema Operativo**
- **Il software dell'applicativo (e.g., il processo HTTP) si trova all'esterno del Sistema Operativo**

Come possono interagire?

Come si riesce a passare dallo spazio applicativo allo spazio kernel?

Applicazioni

Processo

Processo

?

?

Sistema operativo

TCP

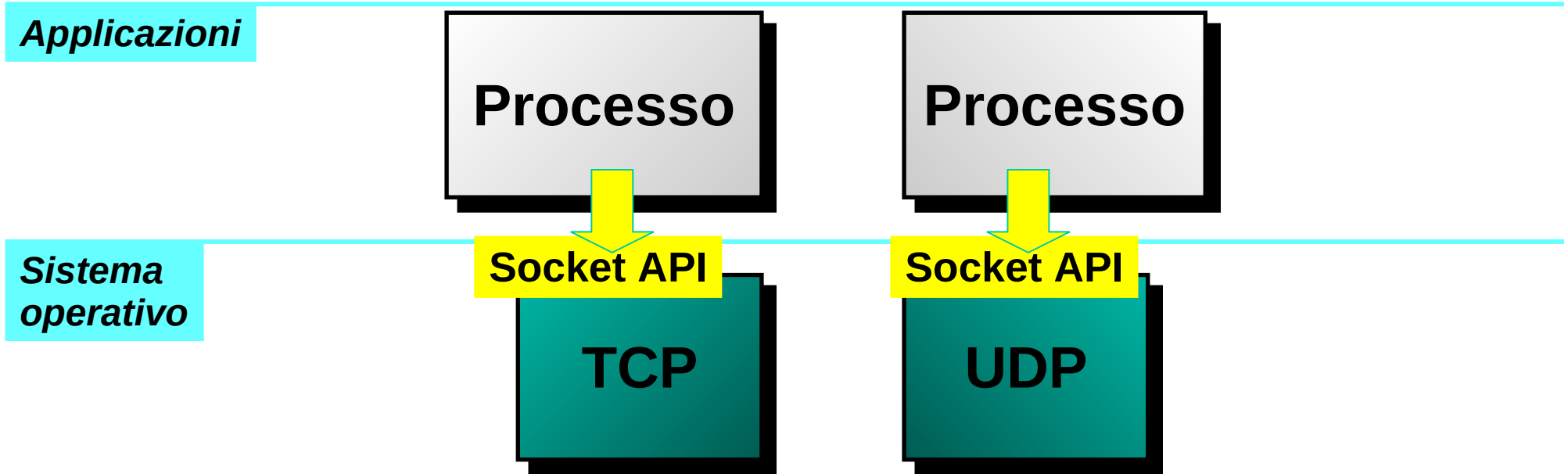
UDP

IP

Spesso hardware

Host-to-network

Il sistema operativo mette a disposizione delle socket API



Si utilizza un meccanismo che svolge il ruolo di ponte tra Sistema Operativo e applicativo di rete: *Application Program Interface (API)*

Application Program Interface

- **E' parte del Sistema Operativo**
- **Consente ai processi applicativi di utilizzare i protocolli di rete, definendo:**
 - Le operazioni consentite
 - Gli argomenti per ciascuna operazione
- **Socket API**
 - Definiti inizialmente per **BSD Unix** (B=Berkeley) per utilizzare i protocolli TCP/IP
 - Ora divenuti uno standard industriale, disponibile su vari Sistemi Operativi

Socket

- I socket sono delle Application Programming Interface (API) che consentono ai programmatori di gestire le comunicazioni tra processi
- A differenza degli altri costrutti di comunicazione (*pipe*, *code di messaggi* e *memoria condivisa*), i socket consentono la comunicazione tra processi che possono risiedere su host diversi, e dunque costituiscono lo strumento di base per realizzare servizi di rete
- In pratica, consentono ad un programmatore di effettuare trasmissioni TCP e UDP senza curarsi dei dettagli “di più basso livello” che sono uguali per ogni comunicazione (*three-way handshaking*, *finestre*, *buffer*,...)

- **Provvedono un interfacciamento verso il protocollo sottostante a livello di trasporto**
STREAM: TCP
DATAGRAM: UDP
- **Provvedono un interfacciamento direttamente verso lo strato RETE: *RAW SOCKET***
- **Si utilizzano in ambiente Linux/Unix in modo simile ai file per I/O**

Socket in BSD Unix

- **Socket**: crea un punto di comunicazione
- **Bind**: associa un indirizzo locale ad un socket
- **Listen**: si rende disponibile ad accettare connessioni
- **Accept**: si mette in attesa di una richiesta di connessione
- **Connect**: tenta di stabilire una connessione
- **Write**: invia dati sulla connessione
- **Read**: riceve dati dalla connessione
- **Close**: chiude la connessione

Fasi di una comunicazione

- **Dichiarazione al sistema operativo che si intende instaurare una connessione con specifica delle caratteristiche**
- **Apertura della connessione (differente dal lato server rispetto al lato client):**
 - il server assume di definire la connessione prima del client, e rimane in attesa che il client si connetta alla porta specificata
 - il client assume che il server sia già attivo e prova a connettersi specificando indirizzo e porta del server
- **Scambio di dati bidirezionale (trasmissione e ricezione)**
- **Chiusura della connessione**

Interfaccia Socket

- **Socket**
 - astrazione del sistema operativo (non hardware)
 - creato dinamicamente
 - persiste soltanto durante l'esecuzione dell'applicazione
 - identificato tramite un descrittore (concetti di UNIX I/O)
- **Descrittore**
 - un intero
 - uno per ogni socket attivo
 - significativo soltanto per l'applicazione che possiede il socket

Interfaccia Socket

- **Funzionalità di socket**
 - struttura socket completamente generale
 - Il socket può essere usato:
 - dal client
 - dal server
 - con un protocollo di trasporto orientato alla connessione (TCP)
 - con un protocollo di trasporto privo di connessione (UDP)
 - per inviare, ricevere dati

Operazioni con socket

- **creazione**

```
descriptor = socket(protofamily, type, protocol)
```

- descriptor: è un intero
- protofamily: PF_INET per Internet
- type: SOCK_STREAM o SOCK_DGRAM

- **chiusura**

```
close (socket) ← descriptor
```

- **binding (usata dal server per fornire un numero di porta)**

```
bind(socket, localaddr, addrlen)
```



descriptor



indirizzo su
cui ascoltare



address len

Operazioni con socket

- **ascolto**

`listen(socket, queuesize)`

- usata dal server per preparare il socket a ricevere una connessione
- il SO costruisce una coda di richieste per ciascun socket

Operazioni con socket

- **accettazione di una nuova richiesta di connessione**

```
newsock = accept(socket, caddress, caddreslen)
```

descriptor



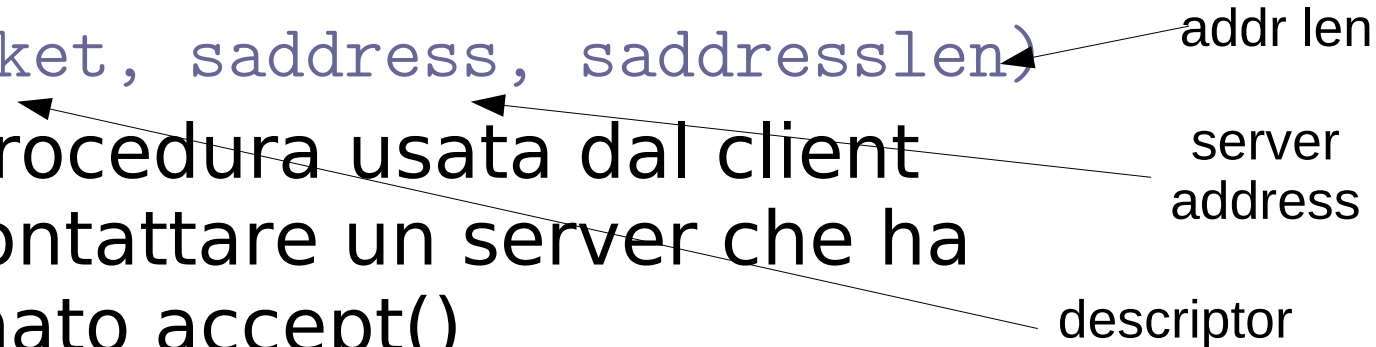
client
address

addr len

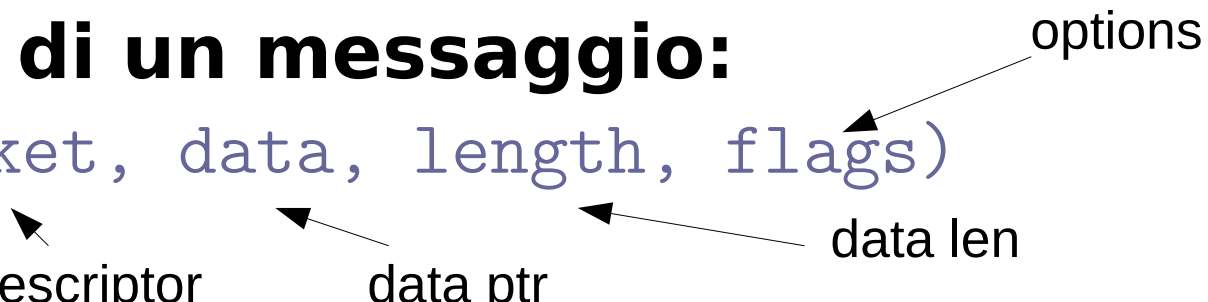
- usata dal server: attende nuova connessione (anche in coda) e crea un nuovo socket
- chiamata successivamente a `socket()` e `bind()` da un server che usa un protocollo di trasporto orientato al servizio
- la struttura `caddress` e `caddreslen` sono riempite da `accept()`
- `newsock`: descrittore del nuovo socket

Operazioni con socket

- **instaurazione di una connessione**

- `connect(socket, saddress, saddresslen)`
- è la procedura usata dal client per contattare un server che ha chiamato `accept()`
 - protocollo orientato alla connessione: `connect()` inizia la connessione
 - protocollo privo di connessione: `connect()` segna il socket come connessa e registra l'indirizzo del server
- 

- **invio di un messaggio:**

- `send(socket, data, length, flags)`
- options
 - data len
 - data ptr
 - descriptor
- 

Programmazione con socket TCP

Il client deve contattare il server

- **il processo server deve essere in esecuzione**
- **il server deve aver creato il socket su cui accettare il contatto del client**

Il client contatta il server

- **creando un socket TCP locale al client**
- **specificando l'indirizzo IP e il numero di porta del processo server**

- **Quando il client crea il socket: il client instaura la connessione al server TCP**
- **Quando è contattato dal client, il server TCP crea una nuova istanza di socket per far comunicare il processo server con il client**

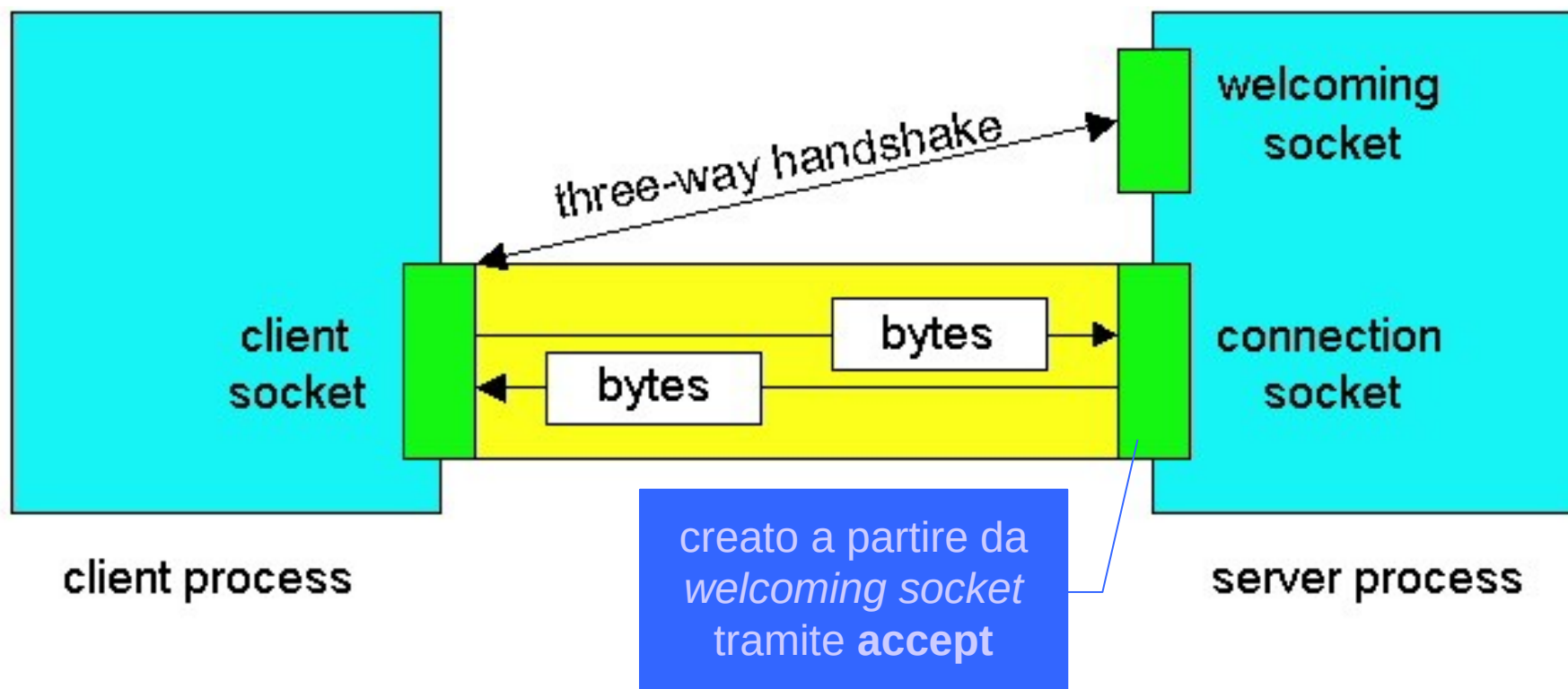
Punto di vista dell'applicazione

TCP fornisce un trasferimento di byte (pipe) affidabile, in sequenza, tra client e server

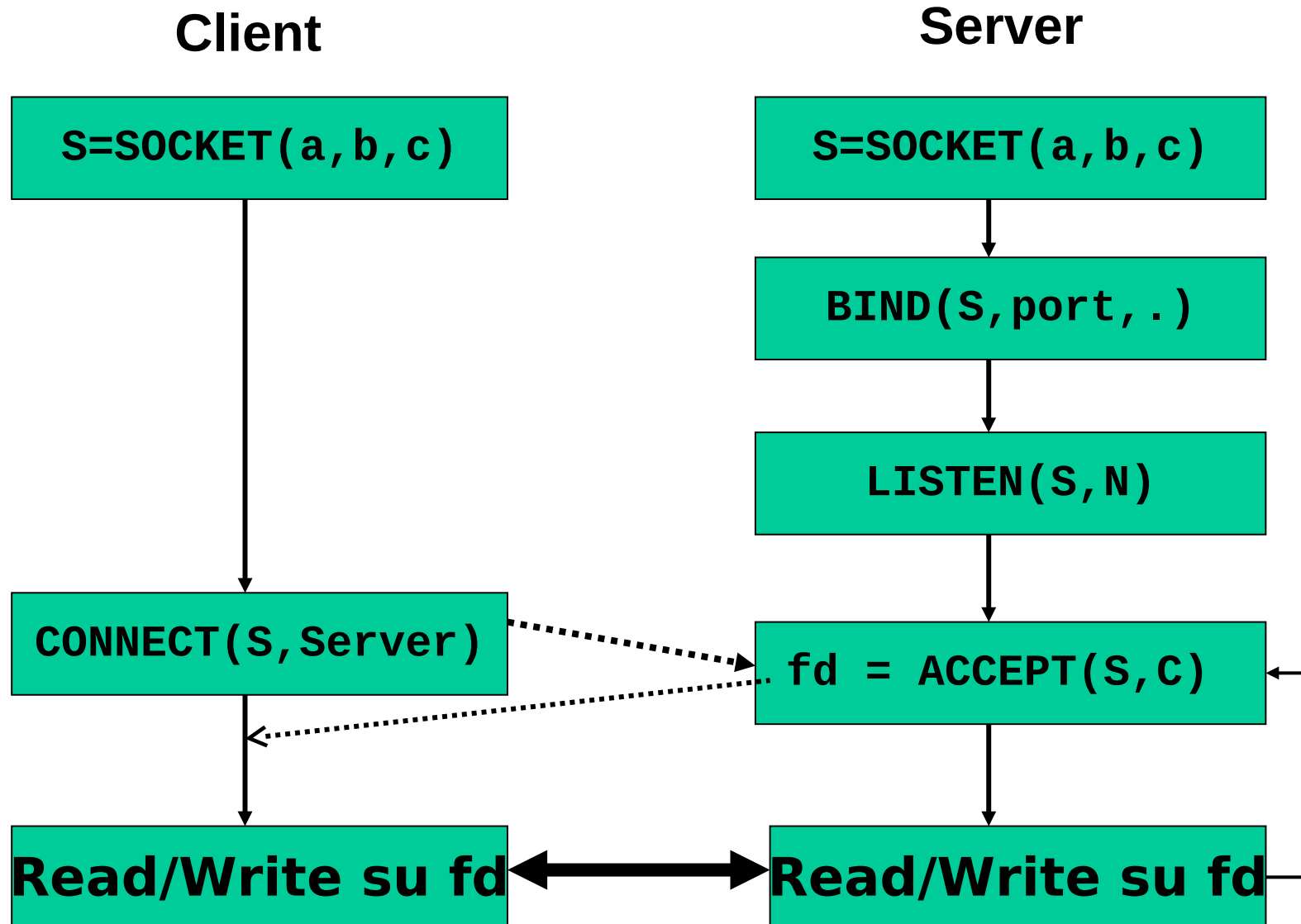
Programmazione con socket TCP

Per l'applicazione:

- la connessione TCP è un collegamento diretto tra il socket del client ed il socket di connessione del server
- il client può inviare i byte nel suo socket: TCP garantisce che il server riceverà (tramite il socket di connessione) ogni byte nello stesso ordine in cui è inviato



Sequenza dei socket *STREAM*



Socket e network programming

- **Allo stato attuale, in nessun corso della Laurea o Laurea Specialistica si approfondirà l'importante (e complessa) tematica del network programming**

- **Per saperne un po' di più ...**

D.E. Comer, “Computer Networks and Internet” (third edition), Prentice Hall, 2001

- **Per sapere tutto su ...**

W.R. Stevens, “Unix Network Programming”, (vol. 1, second edition), Prentice Hall, 1998

NUOVA EDIZIONE SU LINUX: Comer, Stevens (third ed.)

Modulo 3

Esempio con linguaggio C

Lato server

```
/* A simple server in the internet domain using TCP
   The port number is passed as an argument */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

void error(char *msg){
    perror(msg);
    exit(1);
}
```

Lato server

```
int main(int argc, char *argv[]){
    int sockfd, newsockfd, portno, clilen;
    char buffer[256];
    struct sockaddr_in serv_addr, cli_addr;
    int n;
    if (argc < 2) {
        fprintf(stderr,"ERROR, no port provided\n");
        exit(1);
    }
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd < 0)
        error("ERROR opening socket");
```

Lato server

```
bzero((char *) &serv_addr, sizeof(serv_addr));
portno = atoi(argv[1]);
serv_addr.sin_family = AF_INET;
serv_addr.sin_addr.s_addr = INADDR_ANY;
serv_addr.sin_port = htons(portno);
if (bind(sockfd, (struct sockaddr *) &serv_addr,
    sizeof(serv_addr)) < 0)
    error("ERROR on binding");
listen(sockfd,5);
clilen = sizeof(cli_addr);
```

Lato server

```
newsockfd = accept(sockfd,
                    (struct sockaddr *) &cli_addr,
                    &clilen);
if (newsockfd < 0)
    error("ERROR on accept");
bzero(buffer, 256);
n = read(newsockfd, buffer, 255);
if (n < 0) error("ERROR reading from socket");
printf("Here is the message: %s\n",buffer);
n = write(newsockfd, "I got your message",18);
if (n < 0) error("ERROR writing to socket");
return 0;
}
```

Lato client

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

void error(char *msg){
    perror(msg);
    exit(0);
}
```

Lato client

```
int main(int argc, char *argv[]){
    int sockfd, portno, n;
    struct sockaddr_in serv_addr;
    struct hostent *server;
    char buffer[256];
    if (argc < 3) {
        fprintf(stderr,"usage %s hostname port\n", argv[0]);
        exit(0);
    }
    portno = atoi(argv[2]);
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd < 0){error("ERROR opening socket");}
```

Lato client

```
server = gethostbyname(argv[1]);  
if (server == NULL) {  
    fprintf(stderr, "ERROR, no such host\n");  
    exit(0);  
}  
bzero((char *) &serv_addr, sizeof(serv_addr));  
serv_addr.sin_family = AF_INET;  
bcopy((char *)server->h_addr,  
      (char *)&serv_addr.sin_addr.s_addr,  
      server->h_length);  
serv_addr.sin_port = htons(portno);
```

Lato client

```
if (connect(sockfd, (const struct sockaddr *) &serv_addr,
    sizeof(serv_addr)) < 0){error("ERROR connecting");}
printf("Please enter the message: ");
bzero(buffer,256);
fgets(buffer,255,stdin);
n = write(sockfd, buffer, strlen(buffer));
if (n < 0)
    error("ERROR writing to socket");
bzero(buffer,256);
n = read(sockfd, buffer, 255);
if (n < 0){error("ERROR reading from socket");}
printf("%s\n",buffer);
return 0;
}
```


Verifica del funzionamento

Server

```
$ ./server 1025
```

```
Here is the message:  
    messaggio di prova
```

Client

```
$ ./client localhost 1025
```

```
Please enter the message:  
    messaggio di prova  
I got your message
```

Cosa succede durante l'esecuzione

- **Il server si pone in ascolto**
 - Uso di netstat o ss per vedere le porte usate
- **Il client si connette**
 - Uso di un analizzatore di rete (e.g. tcpdump o wireshark) per osservare il three-way handshake
- **Il client scambia dati**
 - Osservabile sempre con un analizzatore di rete
- **Cosa succede se il client tenta di connettersi senza che ci sia un server?**

Modulo 4

Esempio con linguaggio Python

Lato server

```
#!/usr/bin/env python3
import socket
import sys
import time

HOST = '127.0.0.1' # Standard loopback interface address (localhost)
PORT = int(sys.argv[1]) # Port to listen on
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.bind((HOST, PORT))
    s.listen()
    conn, addr = s.accept()
    #print('Connected by', addr)
    data = conn.recv(1024)
    print('Here is the message: %s'% data.decode('utf-8'))
    conn.sendall('I got your message'.encode('utf-8'))
    # socket must be closed by client! sleep for 1 second
    time.sleep(1) # otherwise socket goes to TIME_WAIT!
```

Lato client

```
#!/usr/bin/env python3
import socket
import sys

HOST=sys.argv[1]
PORT=int(sys.argv[2])

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.connect((HOST, PORT))
    msg = input('Please enter the message: ')
    s.sendall(msg.encode('utf-8'))
    data = s.recv(1024)
    s.close()
print('Received: %s'% data.decode('utf-8'))
```

Modulo 5

Qualche esercizio

Obiettivo

- **Applicazione Client/Server**
- **Server**
 - Attende connessioni
 - Invia messaggio di benvenuto
 - “Welcome from <hostname>”
- **Client**
 - Si connette al server
 - Riceve stringa
 - Stampa “Server name is: <hostname>”
- **Versione C + Versione Python**

Obiettivo

- **Osservazioni (linguaggio C)**
- **Per ottenere hostname usiamo funzione**
`#include <unistd.h>`
`int gethostname(char *name, size_t len);`
 - Parametro `name` → string pre-allocata
 - Parametro `len` → dimensione della stringa
 - Per info → `man gethostname`
- **Per manipolare stringhe**
 - Uso della funzione `strcpy`
- **Usare bene l'aritmetica dei puntatori!**

Obiettivo

- **Osservazioni (linguaggio Python)**
- **Per ottenere hostname usiamo funzione**
`socket.gethostname()`
 - Viene ritornata automaticamente una stringa
- **Per manipolare stringhe in Python**
 - Uso di interpolation
`str='blah string %s, int %d' % (mystr, myint)`
 - Concatenazione di stringhe
`str=str1+str2`
 - Uso di formattazione
`str=f'variable is: {variable}'`