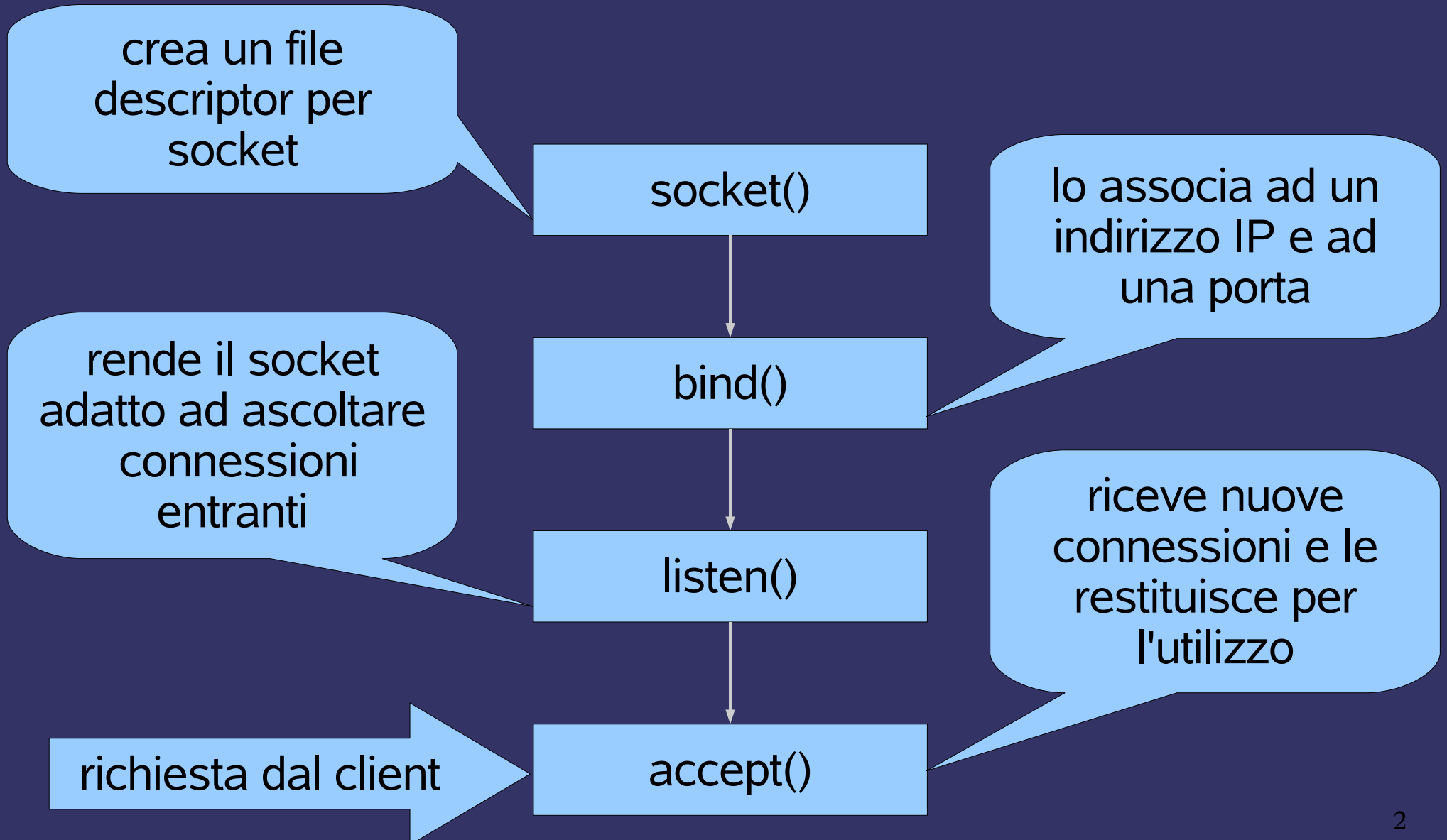


Sviluppo di software di rete con l'utilizzo di sistemi UNIX

Utilizzo dei socket: socket TCP

Claudio Vicari

Inizializzazione di un server TCP



Inizializzazione di un client TCP

crea un file
descriptor per
socket

socket()



connect()

lo associa ad un
indirizzo IP e ad
una porta

richiesta al server

La funzione socket

```
#include <sys/types.h>
#include <sys/socket.h>
int socket(int domain, int type, int protocol);
```

- ➔ questa funzione crea un *endpoint* per la comunicazione fra socket
- ➔ il tipo di ritorno è un *file descriptor*, che si può continuare ad usare come gli altri file descriptor
- ➔ con family si indica la famiglia di protocolli (es. AF_INET per indicare di utilizzare ipv4)
- ➔ con type si indica un sottotipo (es. SOCK_STREAM per protocolli come TCP)
- ➔ protocol specifica un ulteriore protocollo per il socket indicato. Normalmente è 0 perché c'è un solo protocollo per ogni coppia (family, type)

La funzione socket: possibilità per family e type

- ⇒ in family possiamo indicare:
 - AF_INET per IPv4
 - AF_INET6 per IPv6
 - AF_LOCAL per socket locali
 - ...
- ⇒ i corrispondenti PF_INET, ..., sono meno usati ma identici. Le specifiche Posix sono confuse al riguardo...
- ⇒ in type possiamo indicare:
 - SOCK_STREAM (TCP)
 - SOCK_DGRAM (UDP)
 - SOCK_RAW (socket di livello 3, network, non validi per i socket locali)

Breve esempio per socket

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>

int main() {
    printf( "%d\n", socket(PF_INET, SOCK_DGRAM, 0) );
    printf( "%d\n", socket(PF_INET6, SOCK_STREAM, 0) );
    printf( "%d\n", socket(PF_INET6, SOCK_STREAM, 1) );
    return 0;
}
```

La funzione connect

```
int connect(int sockfd,  
            const struct sockaddr *serv_addr,  
            socklen_t addrlen);
```

- ➔ connect richiede la connessione del socket verso l'*endpoint* indicato da serv_addr
- ➔ sockfd deve essere stato ottenuto con la funzione socket
- ➔ serv_addr è stato costruito rispettando la struttura di indirizzamento dei socket. Nel caso di TCP e UDP, indica server e porta a cui collegarsi
- ➔ addrlen specifica la dimensione in byte della struttura serv_addr

La funzione connect: effetti in TCP

- ➔ nel caso di TCP, la funzione connect provoca l'inizio del *three-way-handshake*. Può restituire diversi errori (in errno)
 - 1 in caso che non si riceva risposta alla richiesta di connessione (con tentativi di ritrasmissione ed un timeout), ritorna ETIMEDOUT
 - 2 in caso che riceva un pacchetto di RST (che indica che nessun processo è in ascolto su quella porta), restituisce ECONNREFUSED
 - 3 se riceve un messaggio ICMP di destinazione irraggiungibile (normalmente, vengono dai router), cerca di raggiungere l'host comunque. Alla scadenza di un altro timeout, restituisce EHOSTUNREACH o ENETUNREACH

La funzione connect: esempi

- ⇒ riprendiamo il client `daytime:intro/daytimetcpcli1.c`
 - `./daytimetcpcli 127.0.0.1`
- ⇒ se il server è avviato, riceviamo la risposta
- ⇒ fermiamo il server e riproviamo
- ⇒ proviamo anche:
 - `./daytimetcpcli <indirizzo a caso non raggiungibile>`
 - `./daytimetcpcli <un ip non usato nella sottorete>`
 - oppure facendo in modo che i pacchetti non possano giungere all'host perché scartati (es. con firewall)

La funzione bind

```
int bind(int sockfd,  
         const struct sockaddr *my_addr,  
         socklen_t addr_len );
```

- ➡ questa funzione associa un indirizzo locale ad un socket (bind: legare)
- ➡ l'indirizzo deve rispettare le regole del protocollo del socket specificato

La funzione bind /2

- ➔ per TCP e UDP, l'indirizzo è (IP,porta)
- ➔ possiamo specificare IP, una porta, o nessuno dei due
- ➔ se non si specifica una porta, il sistema ce ne assegna una in automatico
- ➔ il processo può scegliere uno specifico indirizzo IP, che sia associato ad una interfaccia di rete dell'host stesso
- ➔ un server normalmente fa bind:
 - specifica una porta stabilita, perché i client possano connettersi
 - non specifica un indirizzo IP, perché il sistema ascolta in automatico sulle interfacce uscenti
- ➔ un client normalmente non fa bind, ma solo connect. Con connect, il socket viene automaticamente legato ad una coppia (IP,porta) scelta dal kernel

La funzione bind /3

- ➔ se impostiamo `porta=0`, il kernel ci fornisce una porta fra quelle cosiddette *effimere*
 - i numeri di porta < 1024 (dette *well-known port*), sono normalmente usate solo da processi di root
- ➔ specificando `IP=0`, il kernel non ci lega veramente ad un IP specifico. Lo farà solo quando il socket diverrà connesso (quindi con un altro endpoint)
- ➔ con IPv4, specifichiamo con `INADDR_ANY` l'intero a 32 bit 'wildcard'
- ➔ con IPv6, si usa la variabile `in6addr_any`. In questa struttura, infatti, l'indirizzo è espresso come un vettore, non come un semplice intero

Esempi

client

daytimetcpcli.c

daytimetcpcliv6.c

server

daytimetcpsrv.c

daytimetcpsrvv6.c

La funzione listen

```
int listen(int sockfd, int backlog);
```

- ➔ quando si crea un socket, di default è un socket attivo, pronto per fare *connect*. Questa funzione indica al kernel che il socket invece deve accettare connessioni entranti
- ➔ il secondo argomento specifica quante connessioni entranti possono essere accodate – mai impostarlo a 0. Possiamo sempre specificarlo molto grande, senza avere errori
- ➔ normalmente chiamiamo questa funzione dopo aver fatto sia *socket* che *bind*
- ➔ se la coda è piena, il client non riceve alcun pacchetto di errore (RST) – l'errore è considerato temporaneo

La funzione accept

```
int accept(int sockfd, struct sockaddr *cliaddr,  
          socklen_t *addrlen);
```

- ➔ è chiamata da un server, per ottenere la prossima connessione entrante
- ➔ restituisce un nuovo fd per socket, che si riferisce alla connessione verso il client
- ➔ il primo argomento è detto socket in ascolto, che deve essere già stato preparato con listen
- ➔ secondo e terzo argomento possono essere NULL. Altrimenti, conterranno i dettagli sul client connesso

Esempio per accept

daytimetcpsrv1.c

- ➔ questo esempio mostra come usare la struttura `cliaddr`
- ➔ per il resto, è identico al server già visto in precedenza
- ➔ notare che, siccome usiamo la porta 13 (<1024), dobbiamo eseguire il server con privilegi di root

Funzioni close, shutdown

```
int close(int fd);  
int shutdown(int s, int how);
```

- ➡ close chiude il file descriptor come al solito. Ma TCP manda i pacchetti di chiusura connessione solo quando:
 - tutti i processi hanno chiuso il socket
 - tutti i dati in coda sono stati inviati
- ➡ inoltre, chiude in entrambe le direzioni

- ➡ shutdown invece garantisce l'invio della sequenza di chiusura. Con how possiamo indicare:
 - SHUT_RD per impedire ricezioni
 - SHUT_WR per impedire scritture
 - SHUT_RDWR per chiudere tutto

Esercizio

- ➔ scrivere un programma che prenda come argomento un indirizzo IP
- ➔ il programma userà bind per scoprire quali porte sono già utilizzate nel nostro host
- ➔ scrivere un programma che scopra quali porte sono usate, ma su un host esterno