Reti di Calcolatori - Laboratorio

Lezione 5

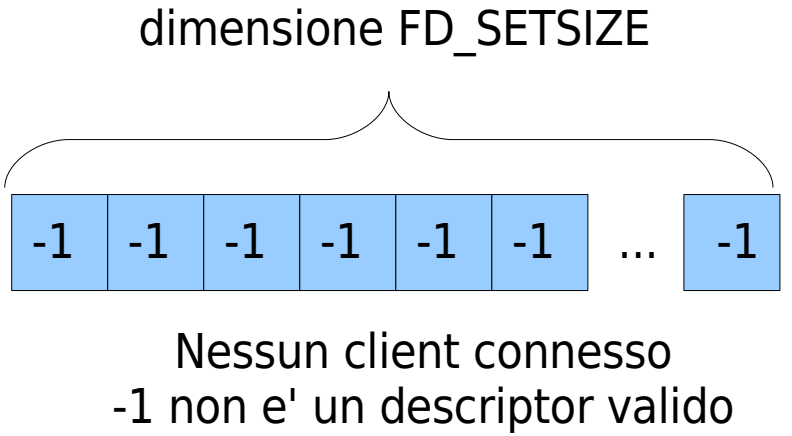
Gennaro Oliva

Server basato su I/O Multiplex

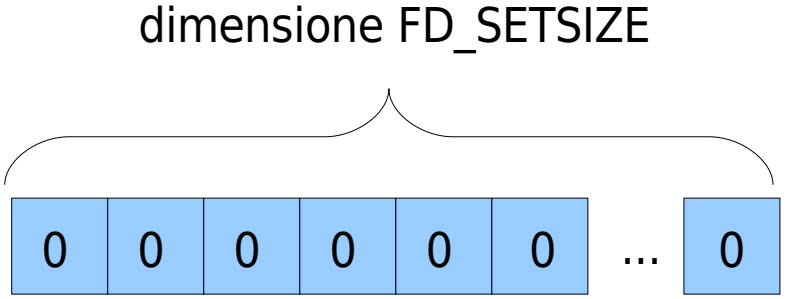
- Per realizzare un server è possibile utilizzare l'I/O Multiplex
- Un unico processo iterativo gestisce il socket che accetta nuove connessioni e tutti i socket connessi
- Il server memorizza:
  - L'insieme di descrittori di tipo fd_set da monitorare con select
  - Un array di interi che memorizza i socket descriptor relativi alle connessioni con i client

Strutture dati Server

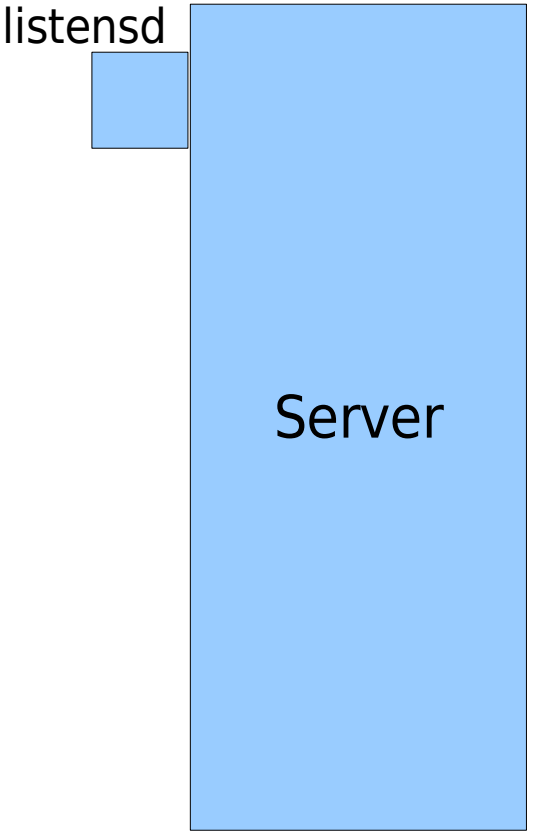
Array di descrittori dei socket connessi con i client connsd[ ]



fd_set dei descrittori da monitorare in lettura



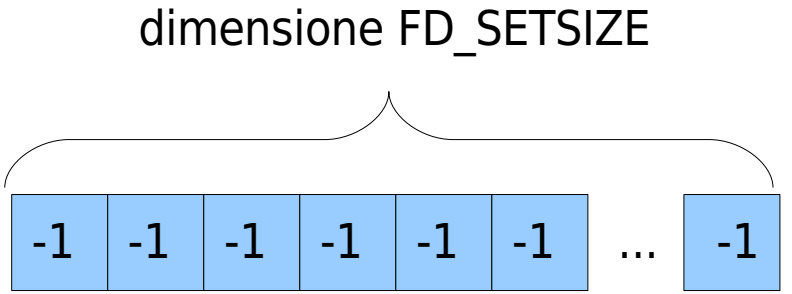
Server basato su I/O Multiplex



- Il server in origine ha solo il socket in ascolto
- Escludendo standard in out ed err il primo descrittore libero per il socket in ascolto è 3

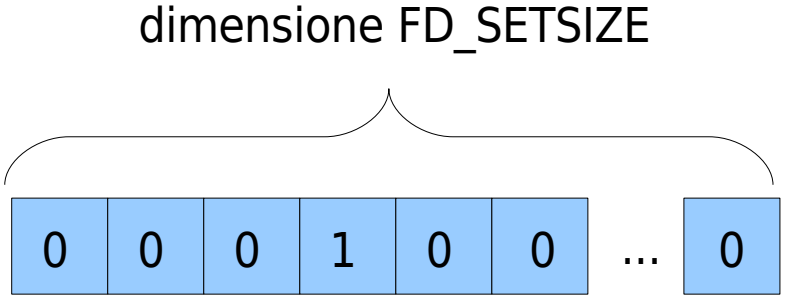
Strutture dati Server

Array di descrittori dei socket connessi con i client connsd[ ]



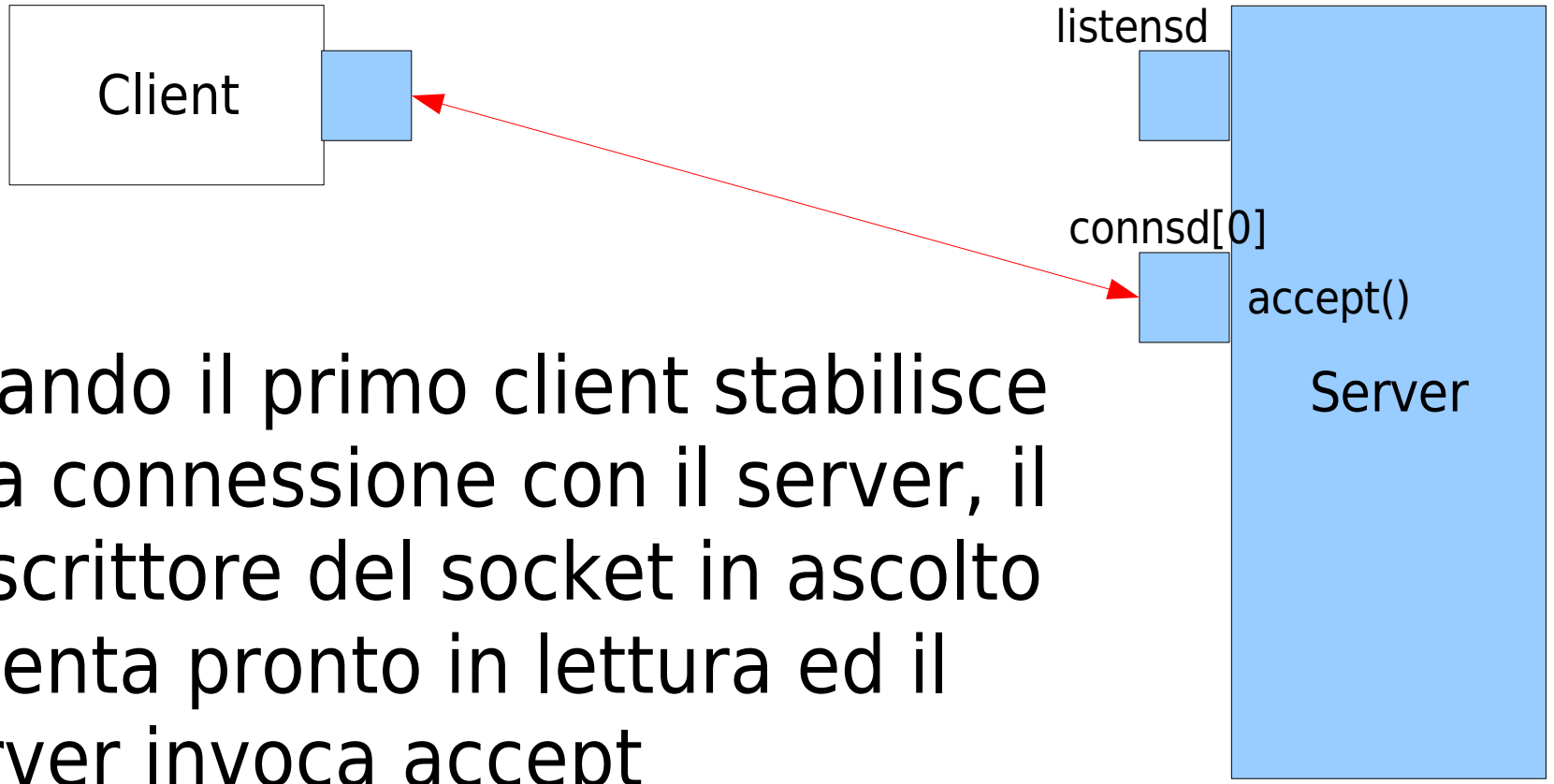
Nessun client connesso
-1 non e' un descriptor valido

fd_set dei descrittori da monitorare in lettura



l'unico descrittore da monitorare e' quello in ascolto ovvero il 3
maxfdp1=4

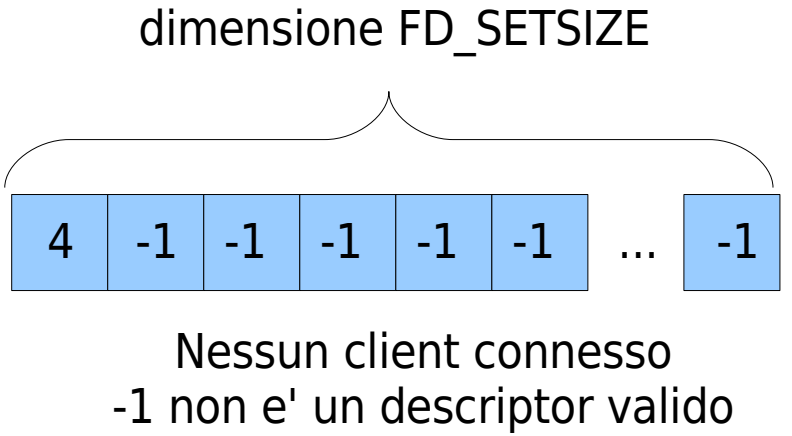
Server basato su I/O Multiplex



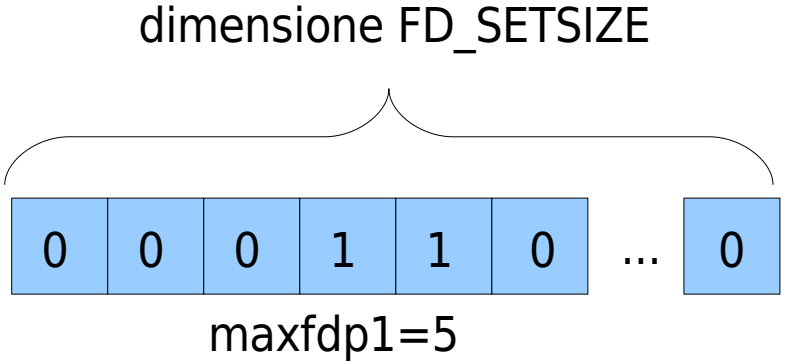
Quando il primo client stabilisce una connessione con il server, il descrittore del socket in ascolto diventa pronto in lettura ed il server invoca accept
Si supponga che Il socket descriptor restituito da accept è 4

Strutture dati Server

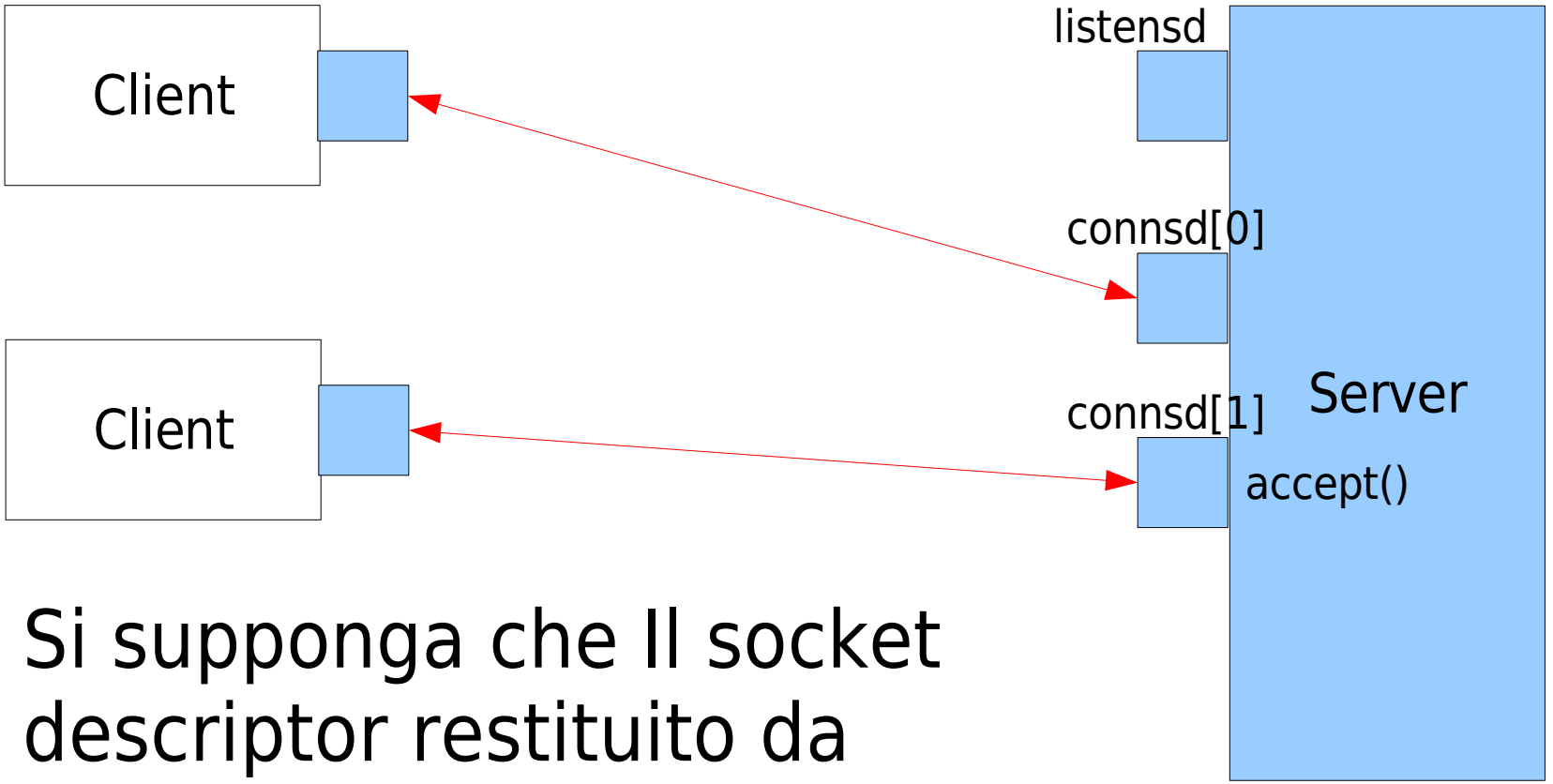
Array di descrittori dei socket connessi con i client connsd[ ]



fd_set dei descrittori da monitorare in lettura



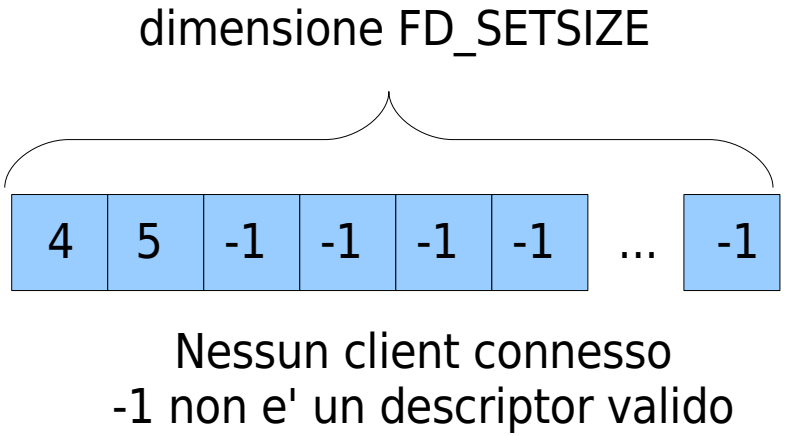
Server basato su I/O Multiplex



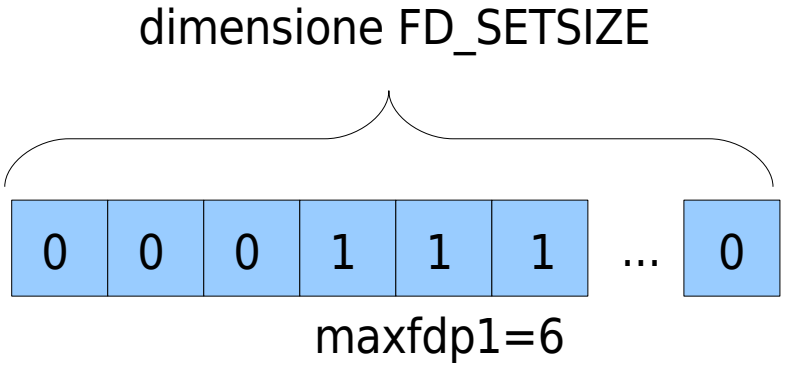
Si supponga che Il socket descriptor restituito da accept dopo la connessione del secondo client sia 5

Strutture dati Server

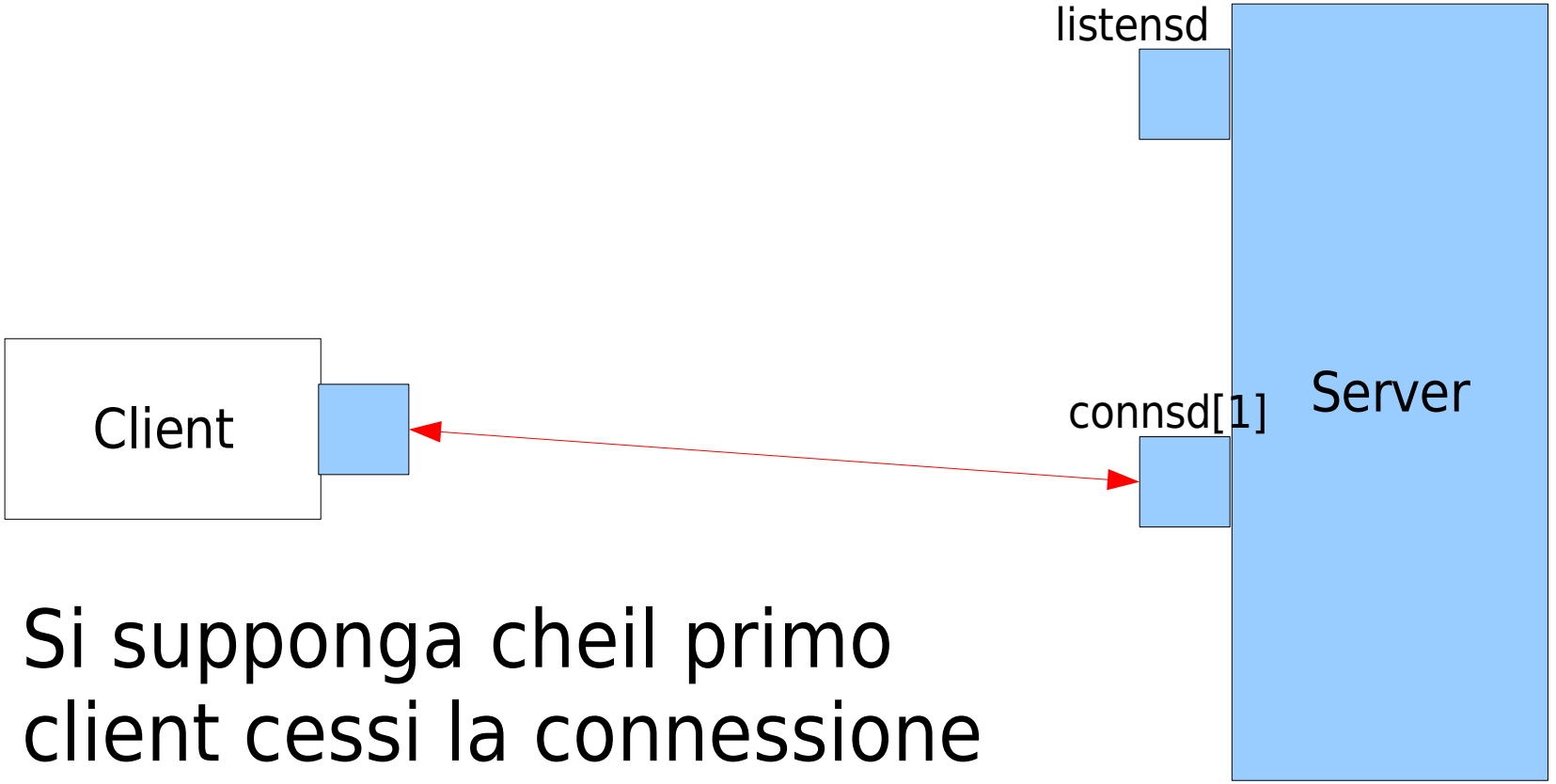
Array di descrittori dei socket connessi con i client connsd[ ]



fd_set dei descrittori da monitorare in lettura



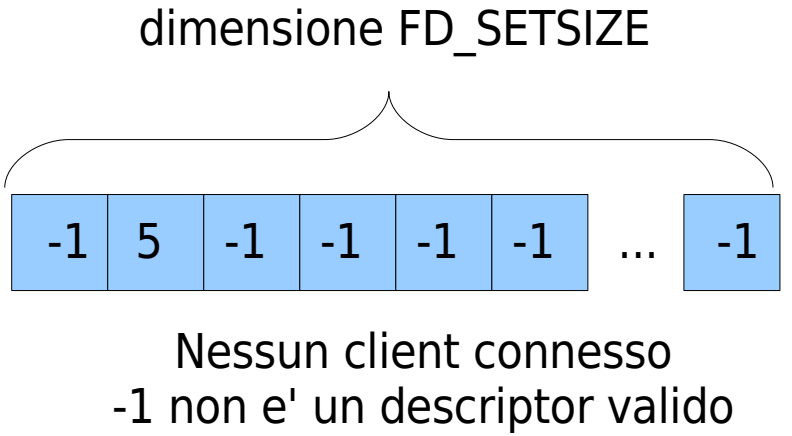
Server basato su I/O Multiplex



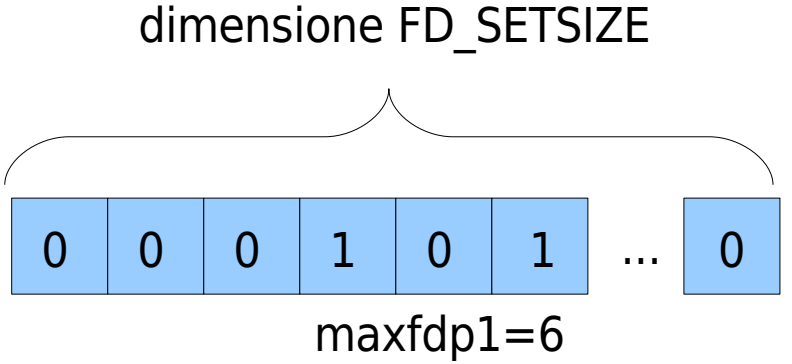
Si supponga cheil primo client cessi la connessione

Strutture dati Server

Array di descrittori dei socket connessi con i client connsd[ ]

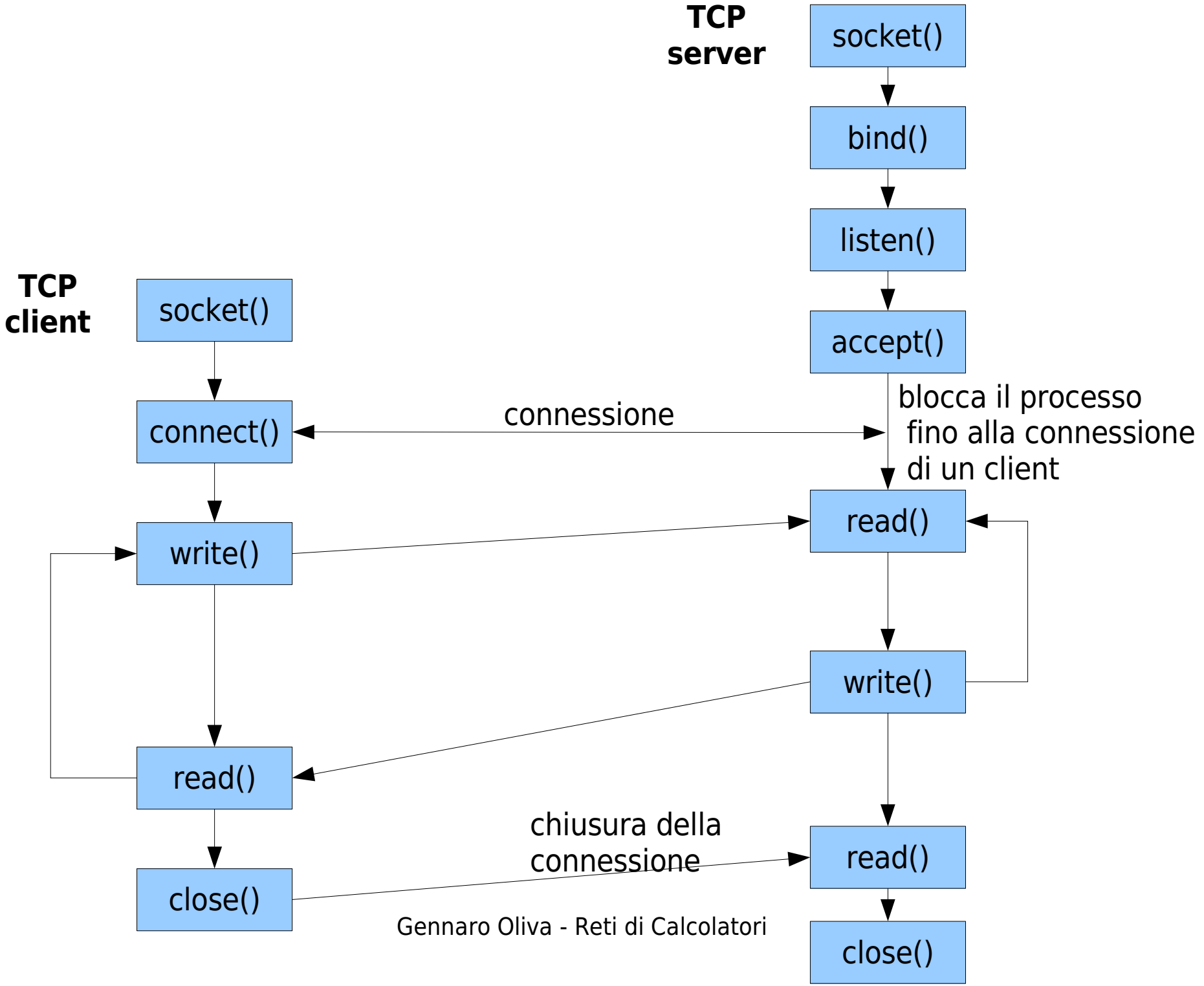


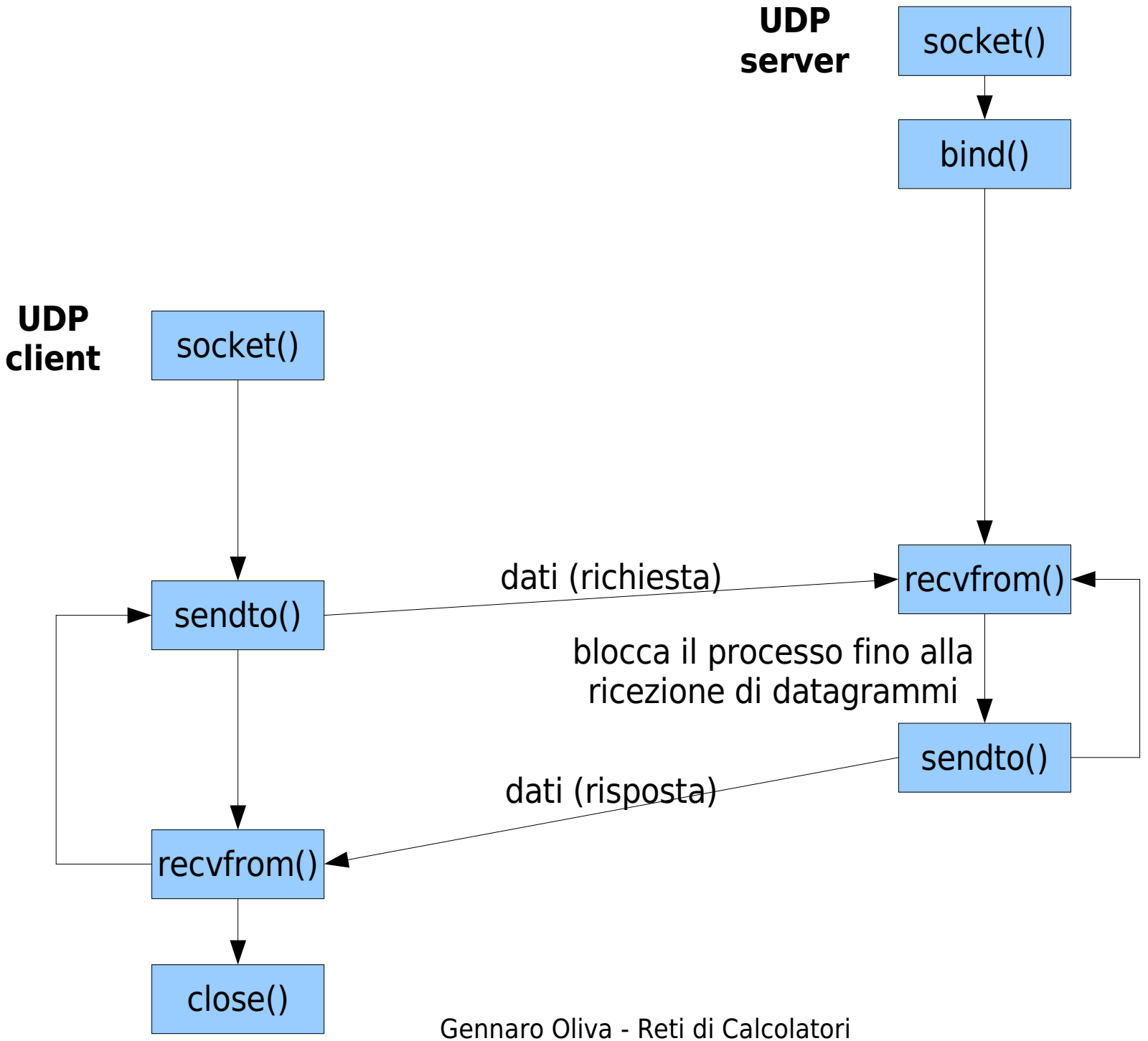
fd_set dei descrittori da monitorare in lettura

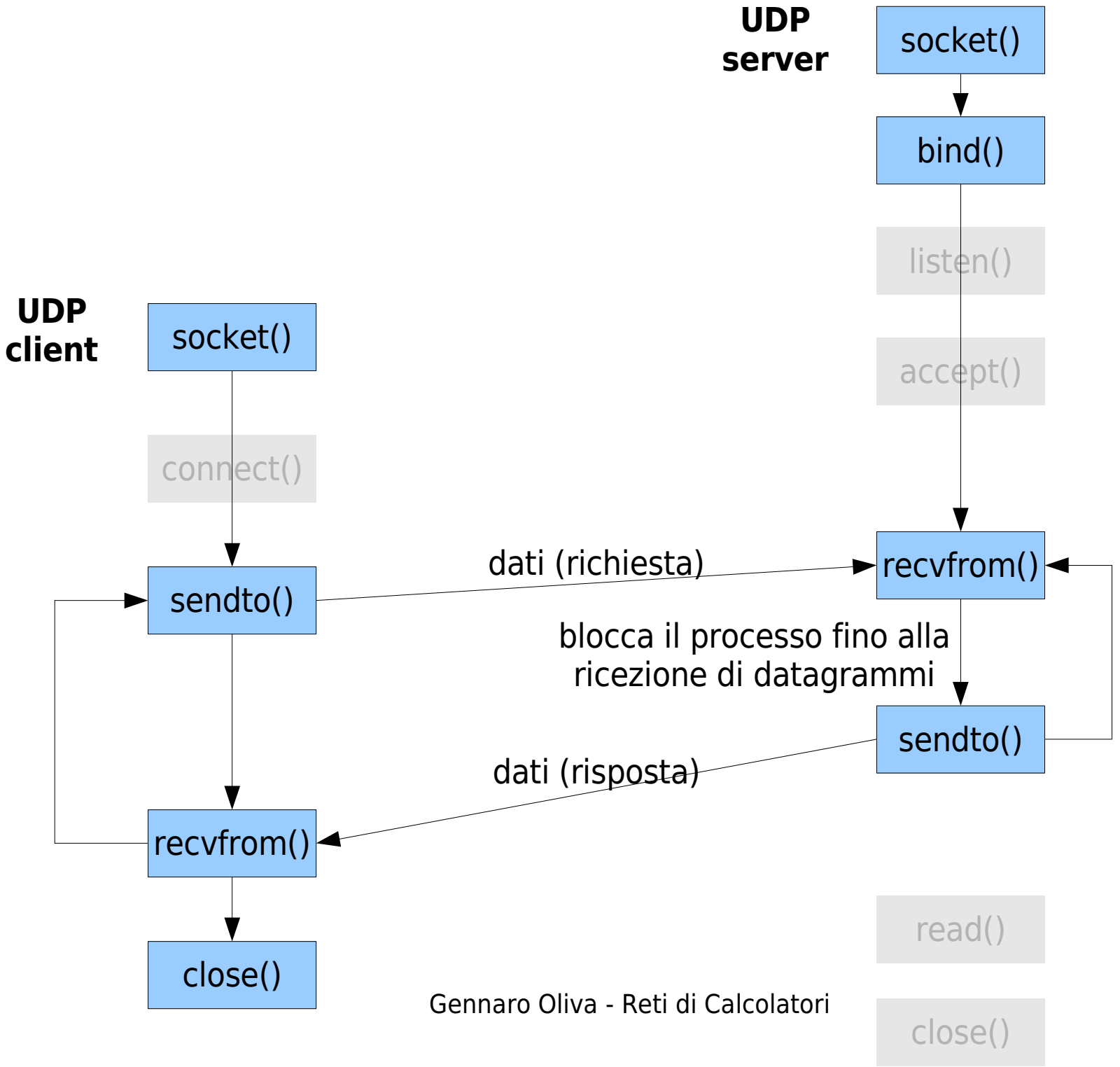


UDP

- In una comunicazione dati Datagram il canale:
  - non è affidabile
  - e' condiviso
  - non preserva l’ordine delle informazioni
- Applicazioni scritte usando UDP:
  - DNS
  - NFS
  - SNMP


TCP server
socket()
bind()
listen()
accept()
blocca il processo fino alla connessione di un client
TCP client
socket()
connect()
connessione
write()
read()
write()
read()
chiusura della connessione
read()
close()
close()


UDP server
socket()
bind()
UDP client
socket()
sendto()
dati (richiesta)
recvfrom()
blocca il processo fino alla ricezione di datagrammi
sendto()
dati (risposta)
recvfrom()
close()


UDP server
socket()
bind()
listen()
accept()
recvfrom()
UDP client
socket()
connect()
sendto()
dati (richiesta)
blocca il processo fino alla ricezione di datagrammi
sendto()
dati (risposta)
recvfrom()
close()
read()
close()

socket

- La funzione socket per comunicazioni UDP viene invocata con i seguenti parametri:

socket(AF_INET,SOCK_DGRAM, 0);

- Per leggere e scrivere su un socket UDP si utilizzano funzioni differenti dal TCP:

int recvfrom(int sd, void* buf, int n, int flags, struct sockaddr* from, socklen_t *len);

int sendto(int sd, const void* buf, int n, int flags, const struct sockaddr* to, socklen_t len);

recvfrom

ssize_t recvfrom(int sd, void *buf, size_t len, int flags, struct sockaddr *from, socklen_t *fromlen);

- Riceve un messaggio da un socket
  - sockfd - socket descriptor
  - buf - buffer di memorizzazione del messaggio
  - len - lunghezza del buffer
  - flags - imposta la modalità di funzionamento della comunicazione
  - from - memorizza l'indirizzo del mittente
  - fromlen – memorizza la dimensione di from

recvfrom: read e accept

```
ssize_t recvfrom(int sd, void *buf, size_t len, int
  flags, struct sockaddr *from, socklen_t
  *fromlen);
```

- In analogia con quanto accade nelle connessioni TCP dove:

```
int accept(int sd, struct sockaddr *from,
  socklen_t *fromlen);

ssize_t read(int sd, void *buf, size_t len);
```

sendto

int sendto(int sockfd, const void* buf, size_t len, int flags, const struct sockaddr* to, socklen_t len);

- Invia un messaggio su un socket
  - sockfd - socket descriptor
  - buf - buffer che memorizza il messaggio da inviare
  - len - lunghezza del messaggio
  - flags - imposta la modalità di funzionamento della comunicazione
  - to - memorizza l'indirizzo del destinatario
  - len – memorizza la dimensione di to

sendto

```
ssize_t sendto(int sd, const void *buf, size_t len,
  int flags, const struct sockaddr *to, socklen_t
  len);
```

- In analogia con quanto accade nelle connessioni TCP dove:

```
int connect(int sd, const struct sockaddr *to,
  socklen_t len);

ssize_t write (int sd, const void *buf, size_t len);
```

Struttura di client e server UDP

Server

```
Socket(...);
Bind(...);
for ( ; ; ) {
  Recvfrom(...);
  Sendto(...);
}
```

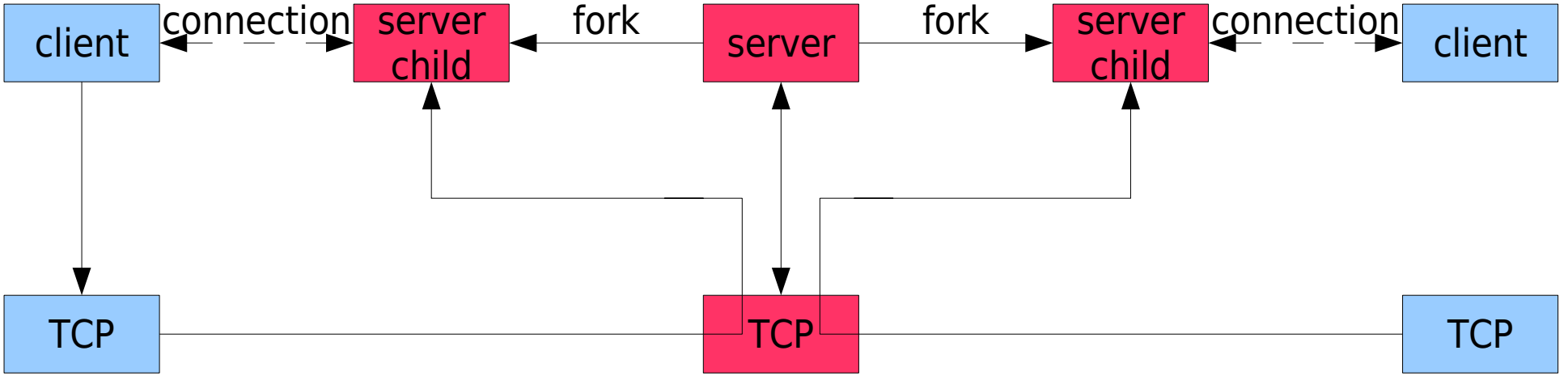
Client

```
Socket(...)
...
Sendto(...);
Recvfrom(...);
...
```

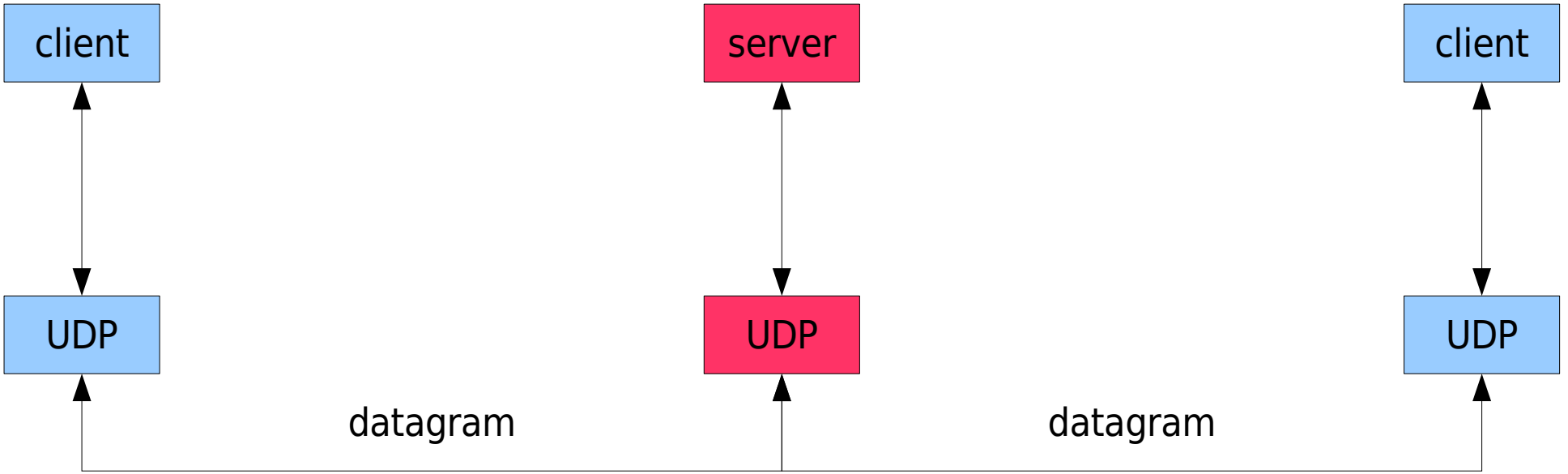
Assegnazione di porte effimere

- Il client non chiede al kernel l'assegnazione di una porta effimera come accade nel TCP mediante la funzione connect
- Con i socket UDP una sendto effettuata su un socket a cui non e' stata assegnata una porta ne causa l'assegnazione

Schema server TCP



Schema server UDP



Inffidabilita'

- La comunicazione tra client e server mediante UDP non e' affidabile:
- Se si perde il pacchetto della richiesta o della risposta l'applicazione client resta bloccata in ricezione
- E' possibile impostare un timeout per la recvfrom
- Non e' possibile sapere quale dei due datagrammi e' andato perso

Verifica del mittente

- Un’applicazione UDP in ascolto su una porta accetta tutti i datagrammi ricevuti
- Per assicurarsi che una risposta provenga da un determinato interlocutore è necessario verificare indirizzo e porta di provenienza del datagramma
- In pratica i client confrontano l’indirizzo del socket da cui ricevono il datagramma con quello del socket a cui hanno inviato la richiesta e scartano i datagrammi di diversa provenienza

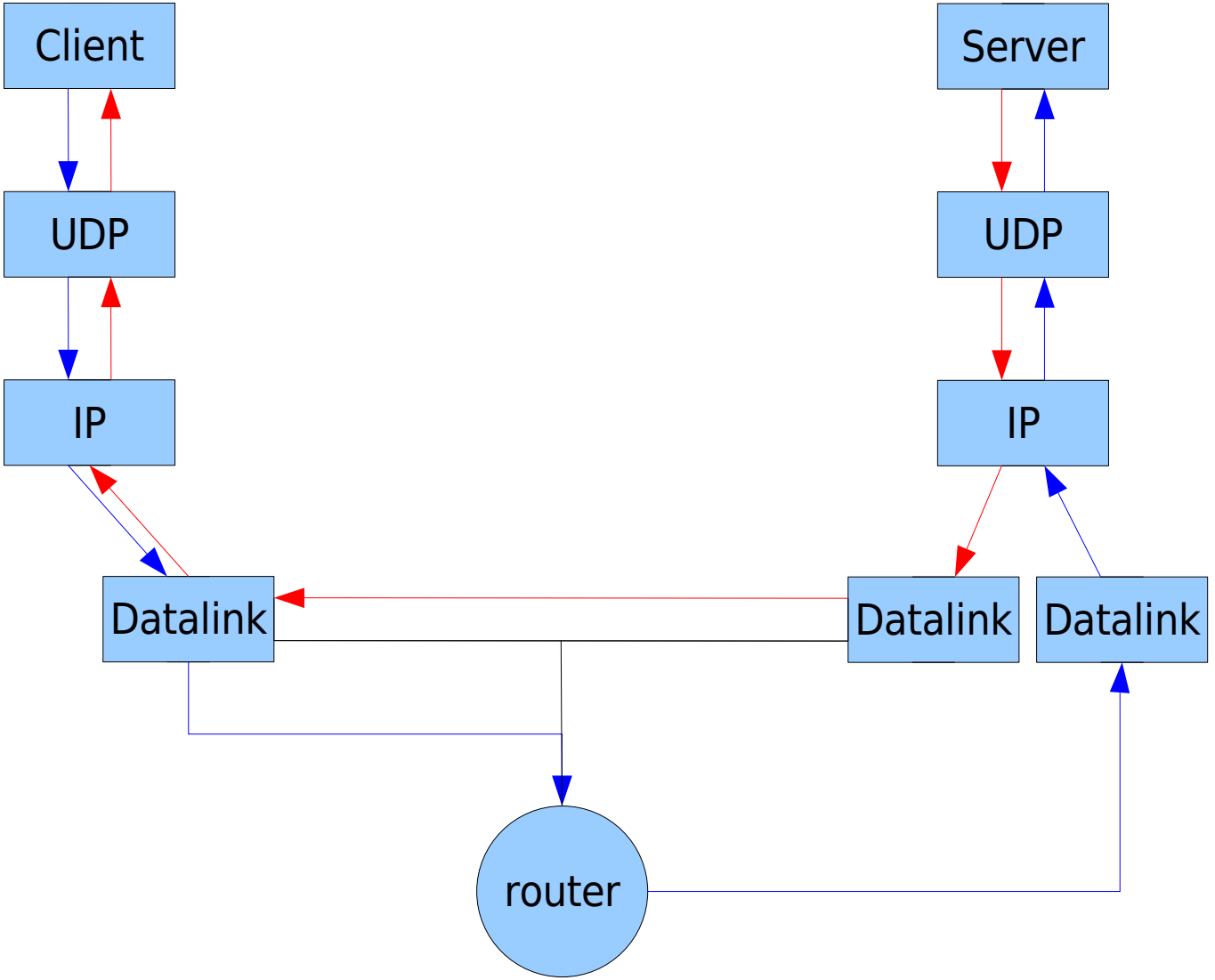
Verifica del mittente

- Se il server ha più indirizzi IP utilizzando confrontando indirizzo di destinazione della richiesta e indirizzo di provenienza della risposta si possono erroneamente scartare datagrammi legittimi
- L'indirizzo IP restituito da recvfrom potrebbe appartenere allo stesso host a cui si e' mandato il datagramma, ma non essere lo stesso a cui la richiesta e' stata inviata

Verifica del mittente

- Due soluzioni alternative sono:
  - verificare che gli indirizzi corrispondano ad uno stesso nome simbolico effettuando una query al dns
  - effettuare un bind per ogni indirizzo IP del server (non specifcando INADDR_ANY), monitorare i vari socket tramite select e rispondere su quello da cui si viene contattati

Verifica del mittente



Assenza di Server

- Se l'applicazione server non e' in esecuzione il client resta bloccato nella recvfrom

```
careca:~# tcpdump -n host 140.164.14.31
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
18:05:11.815687 arp who-has 140.164.14.31 tell 140.164.14.30
18:05:11.815780 arp reply 140.164.14.31 is-at 00:08:c7:b3:1c:aa
18:05:11.815806 IP 140.164.14.30.32792 > 140.164.14.31.9877: UDP, length 5
18:05:11.815850 IP 140.164.14.31 > 140.164.14.30: ICMP 140.164.14.31 udp port 9877 unreachable, length 41
```

- L'errore udp port unreachable e' un errore asincrono:
  l'errore e' causato da sendto ma non viene notificato all'applicazione in quanto la funzione restituisce un valore non negativo

Segnalazione dell'assenza di server

- Perché la situazione di errore non viene segnalata?
  - In sendto non è possibile dato che l'attesa di una risposta che potrebbe essere molto lunga o non arrivate affatto.
  - Nella recvfrom non è possibile in quanto in caso di invio a più server il kernel non può restituire al programma l'indirizzo del destinatario che ha generato l'errore

connect

- Un errore asincrono viene notificato soltanto quando il socket UDP è connesso
- la funzione connect e' utilizzabile anche su socket UDP ma non generea nessuna connessione sul modello del TCP (non c'e' niente di simile al three-way handshake)
- Il kernel verifica eventuali errori (una destinazione non raggiungibile) e memorizza l'indirizzo IP e la porta dell'applicazione con cui si intende comunicare

connect

- L'applicazione che invoca connect può inviare e ricevere datagrammi soltanto dall’indirizzo specificato
- Per inviare i datagrammi non si utilizza sendto ma le funzioni write o send
- i datagrammi vengono automaticamente spediti all’indirizzo specificato nella chiamata a connect

connect

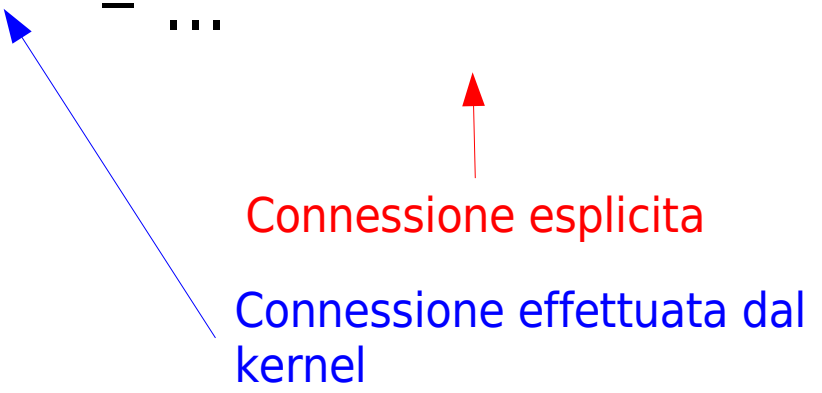
- Per ricevere datagrammi non si usa recvfrom, ma read o readv
- Si possono ricevere solo datagram inviati dall’indirizzo specificato nella chiamata alla connect

connect e applicazioni UDP

- Le applicazioni client e server UDP usano connect solo quando utilizzano un socket per comunicare con un solo interlocutore
- Solitamente il client utilizza connect, ma nel caso in cui il server comunica con un dato client a lungo (per esempio TFTP) e' utilie utilizzare la connect anche sul server
- La connessione di un socket consente nel caso in cui sia necessario inviare piu' datagrammi di avere migliori prestazioni

Performance

- Questo poiche' la funzione sendto per alcuni kernel effettua comunque connessione e disconnessione ad ogni invio
  - Connette il socket
  - Invia il 1° datagramma
  - Disconnette il socket
  - Connette il socket
  - Invia il 2° datagramma
  - Disconnette il socket
  - ...
  - Connette il socket
  - Invia il 1° datagramma
  - Invia il 2° datagramma
  - ...



Server che usa connect

- Quando il server utilizza la connect i datagrammi che arrivano da un altro indirizzo IP vengono instradati ad un’altra connessione socket UDP sullo stesso host I
- l pacchetto verrà scartato se non ci sono altre connessioni UDP su quella porta
- Viene generato un errore “ICMP port unreachable”

Disconnessione

- Un processo con un socket UDP connesso puo' chiamare nuovamente la connect per quel soket per:
  - Specificare un nuovo indirizzo IP e porta (non e' possibile per i socket TCP)
  - Disconnettere un socket
- Per disconnettere un socket udp si specifica AF_UNSPEC nel campo sin_family
- La connect potrebbe restituire l’errore EAFNOSUPPORT, ma si ottiene comunque la disconnessione

Server TCP e UDP

- Schema di un server che fornisce uno stesso servizio mediante i protocolli TCP e UDP

```
tcpsd = Socket(...)
Bind(...)
Listen(...)
udpsd = Socket(...)
Bind(...)
Select(...)
 if (FD_ISSET(tcpsd, &rset))
   Accept()
   ...
f (FD_ISSET(udpsd, &rset))
   ...
```

Esercizi

- Si scrivano i wrapper per le funzioni sendto e recvfrom
- Si scriva in pascal-like la struttura del server che conta i caratteri in modo che gestisca le connessioni mediante mutliplex
- Si scrivano un server ed un client connesso UDP per il conteggio di caratteri di stringhe di lunghezza arbitraria
- Si modifichi il server dell'esercizio precedente in modo che accetti anche connessioni TCP