Reti di calcolatori e Internet: Un approccio top-down

7ª edizione

Jim Kurose, Keith Ross

Pearson Paravia Bruno Mondadori Spa

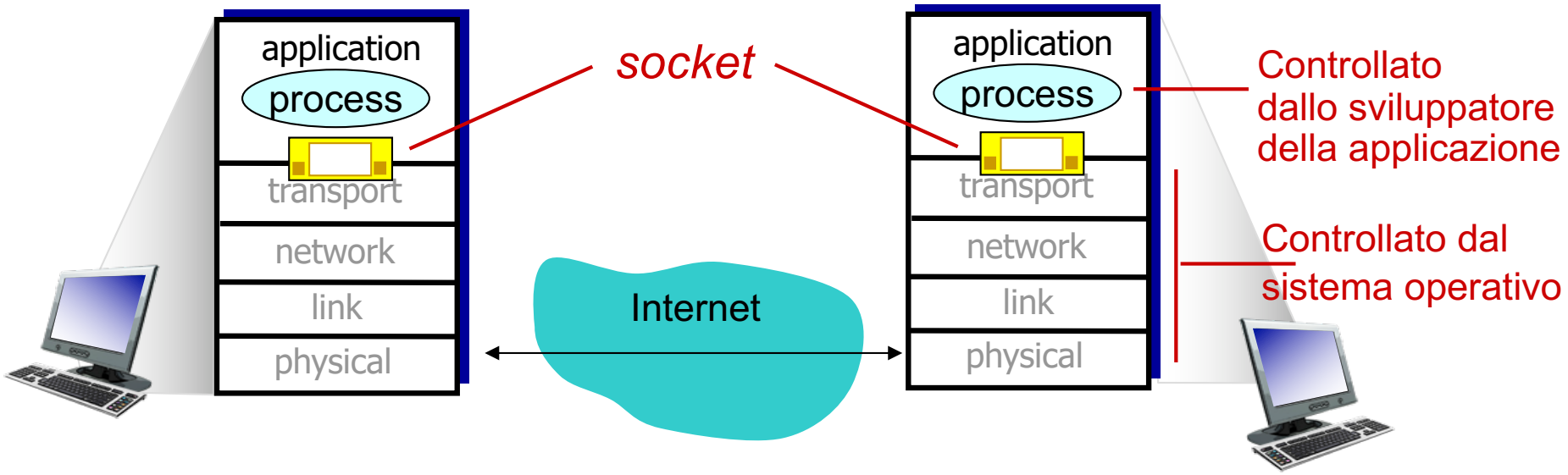
Beej's Guide to Network Programming

http://beej.us/guide/bgnet/

Programmazione delle socket

- *goal:* imparare a costruire un'applicazione client/server che comunica utilizzando le socket

socket: porta tra i protocolli del livello applicativo e i protocolli del livello trasporto



Programmazione delle socket

Due tipi di socket per due servizi di trasporto:

- *UDP:* datagrammi con trasporto non affidabile
- *TCP:* trasporto affidabile di un flusso di byte

Programmazione delle socket con TCP

Il client deve contattare il server

- Il processo server deve essere in corso di esecuzione
- Il server deve avere creato una socket (porta) che dà il benvenuto al contatto con il client

Il client contatta il server:

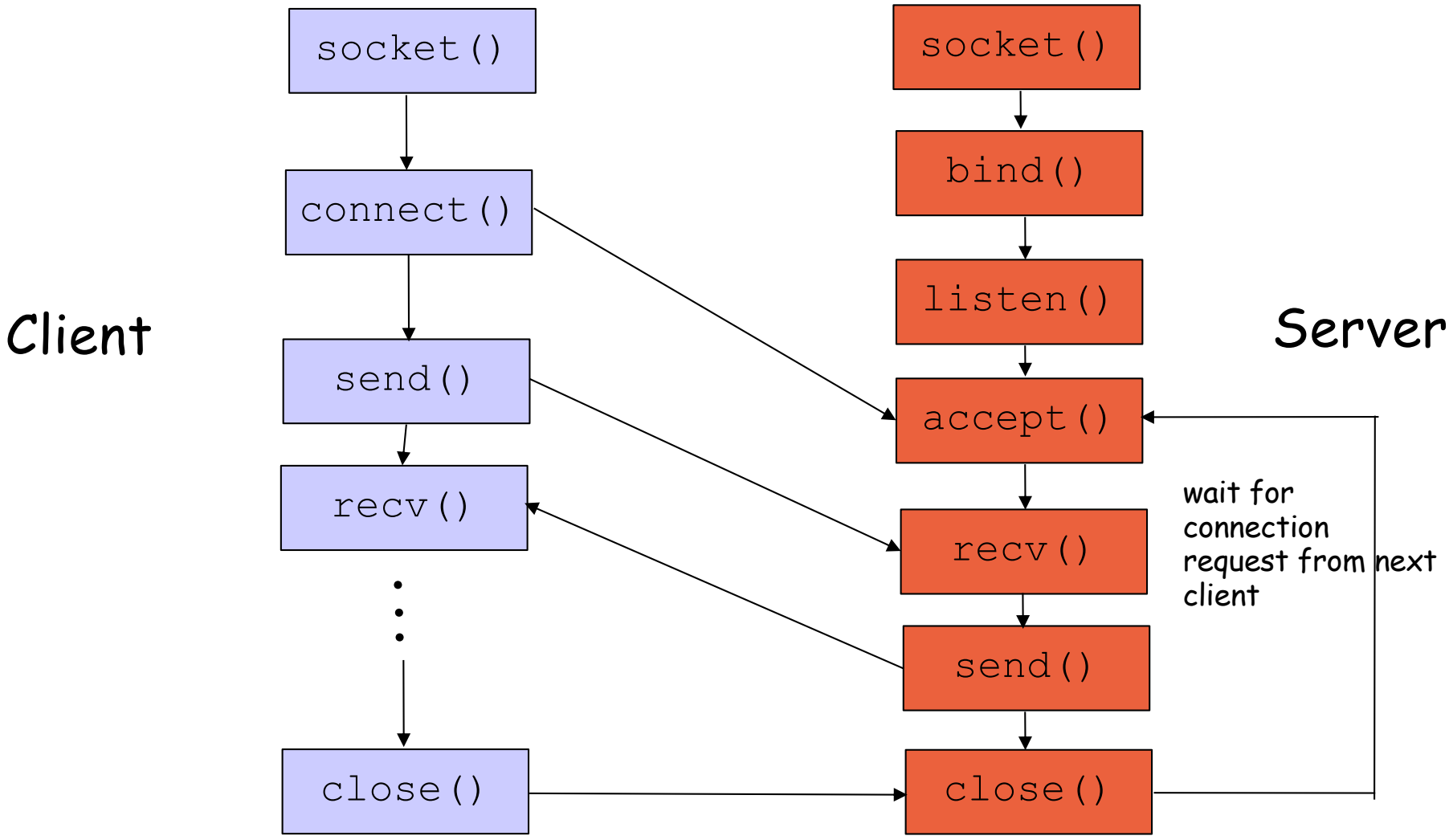
- Creando una socket TCP
- Specificando l'indirizzo IP, il numero di porta del processo server
- Quando il client crea la socket: il client TCP stabilisce una connessione con il server TCP

- Quando viene contattato dal client, il server TCP crea una nuova socket per il processo server per comunicare con il client
  - consente al server di comunicare con più client
  - numeri di porta origine usati per distinguere i client (maggiori informazioni nel Capitolo 3)

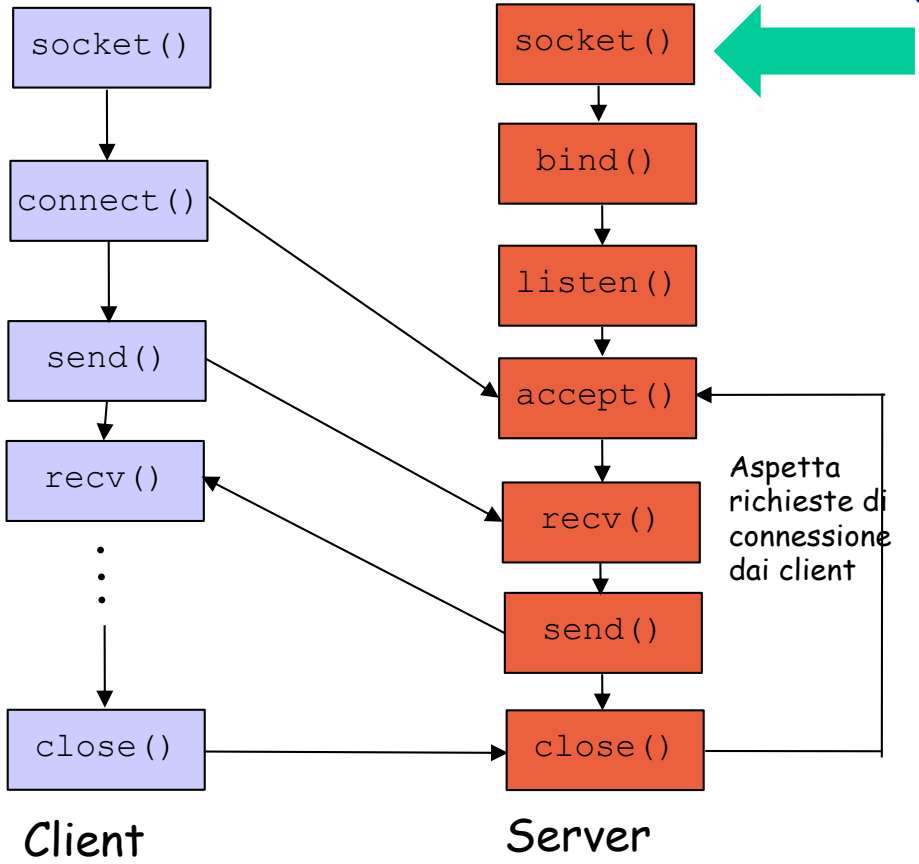
Punto di vista dell'applicazione

TCP fornisce un trasferimento di byte affidabile e ordinato ("pipe") tra client e server

Programmazione delle socket TCP

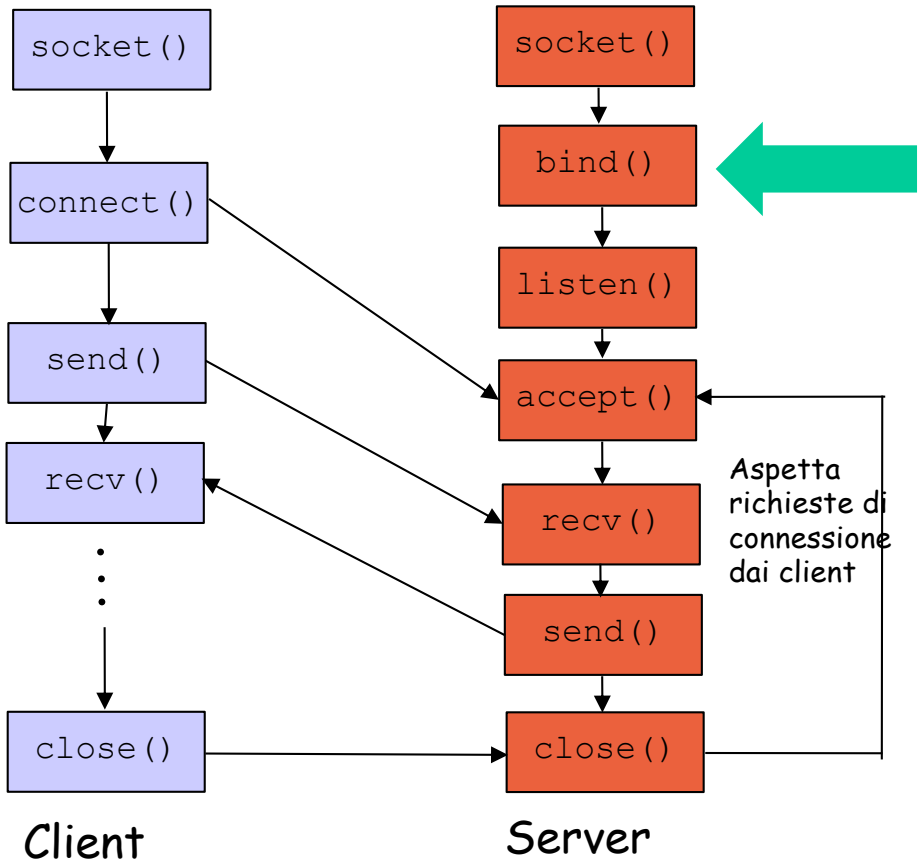


socket()



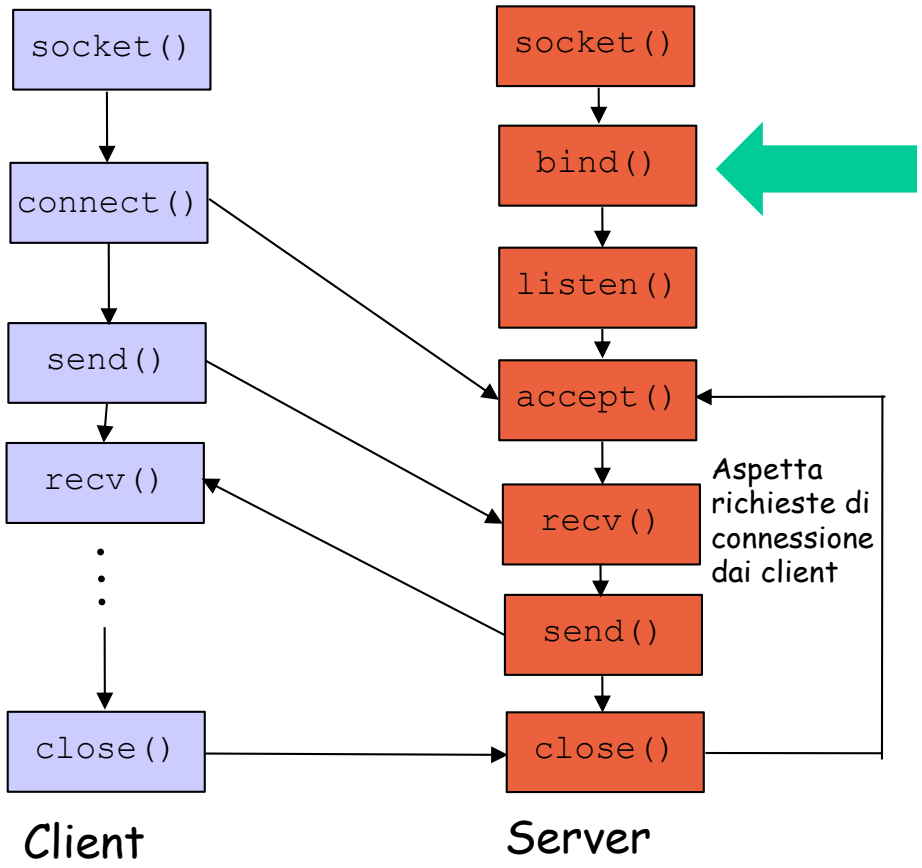
- `int s_listen = socket(family, type, protocol);`
  - family: AF_INET specitica Ipv4
  - type: SOCK_STREAM, SOCK_DGRAM
  - protocol: 0 (pseudo, IP). See /etc/protocols

bind()



- `bind(s_listen, localAdd, addLength)`
  - Il server specifica la porta su cui mettersi in ascolto
  - s_listen: descrittore della socket di ascolto
  - localAdd: socket address structure
  - addLength: lughezza della variabile localAdd

Address



- `bind(s_listen, localAdd, addLength)`
  - Il server specifica la porta su cui mettersi in ascolto
  - s_listen: descrittore della socket di ascolto
  - localAdd: socket address structure
  - addLength: lughezza della variabile localAdd

```
struct sockaddr_in {
u_char sin_len; // length of address
u_char sin_family; // family of
   address
u_short sin_port; // protocol port num
struct in_addr sin_addr; // IP Addr
char sin_zero[8]; // set to zero, used
   for padding
};
```

Strutture per gestire gli indirizzi

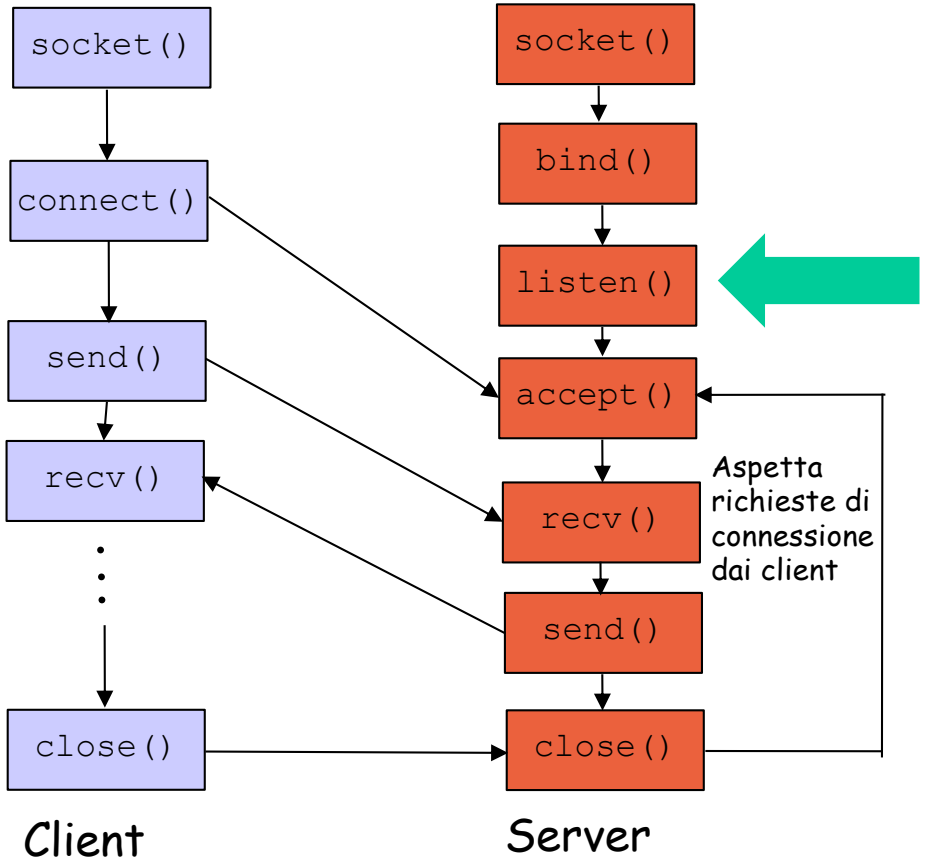
- Dichiarazione della struttura per gestire gli indirizzi
  - `struct sockaddr_in sockAdd;`
- Impostare la famiglia
  - `sockAdd.sin_family = AF_INET;`
- Impostare l'indirizzo IP (2 ways)

  //specify address to listen to
  - `inet_pton(AF_INET, "127.0.0.1", &sockAdd.sin_addr.s_addr)`

  //listen to any local address
  - `sockAdd.sin_addr.s_addr = htonl(INADDR_ANY)`

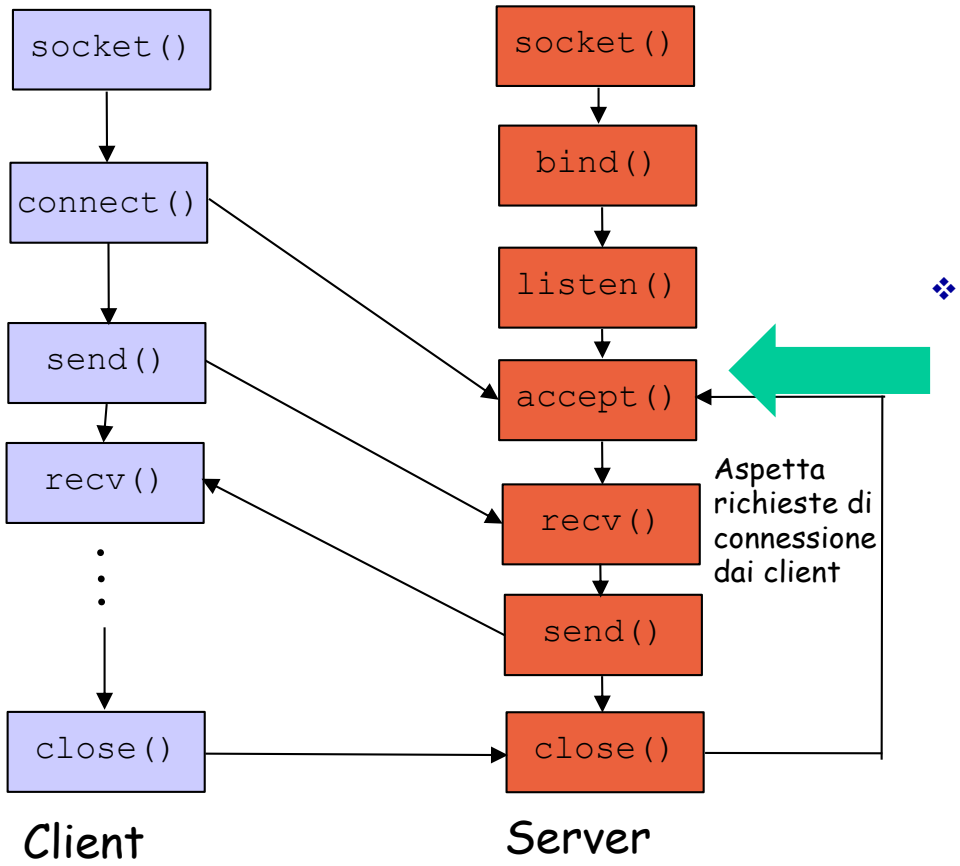
- Impostare la porta
  - `sockAdd.sin_port = htons(9999);`

listen()



- `int status = listen(s_listen, queuelength);`
  - status: -1 errore, 0 altrimenti
  - s_listen: descrittore della socket
  - queuelength: numero di client che possono stare in attesa
  - non-bloccante: ritorna immediatamente

accept()



- `int s_new = accept(s_listen, &clientAddress, &addLength);`
  - s_new: nuova socket per la comunicazione con i client
  - s_listen: descrittore della socket
  - clientAddress: struct sockaddr, indirizzo del client
  - addLength: dimensione della variabile clientAddress
  - bloccante: ritorna solo quando riceve una richiesta di connessione

Scambio dei dati

- `int send(int s_new, const void *buf, int len, int flags);`
  - s_new – descrittore della socket
  - buf – puntatore al buffer
  - len – dimensione del buffer
  - flags – da impostare a 0
- `int recv(int s_new, void *buf, int len, unsigned int flags);`
  - Simile alla send
  - buf conterrà i dati da ricevere

System calls - fork()

- fork() è una chiamata di sistema C usata per generare processi figli
  - L'esecuzione dei processi padre e figlio prosegue dall'istruzione successiva
  - fork() restituisce
    - 0 se e' il processo figlio
    - PID (>0) (identificativo del processo figlio) se è nel processo padre
    - <0 se c'è un errore
- Consente di gestire la conversazione su una socket dedicata, e di rimanere in attesa di richieste di connessioni

Programmazione delle socket

Esempio di applicazione:

1. Un client legge una riga di caratteri dalla tastiera e li invia al server
2. Il server riceve i dati e li analizza:

1. Se la stringa ricevuta è una T, manda al client l'orario corrente
2. Se la stringa è una N, manda il numero di richieste ricevute,
3. Altrimenti, manda legge una stringa da tastiera e la manda

3. Il client riceve la risposta dal server e la mostra sullo schermo

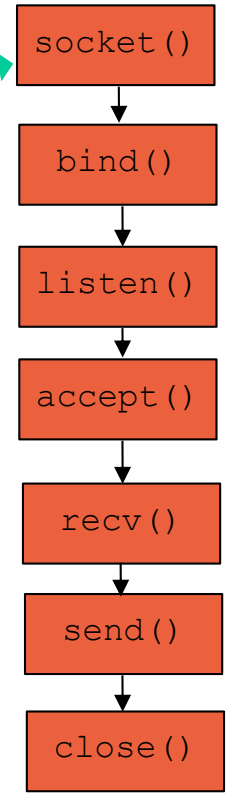
A simple server TCP (1)

```c
#include <stdio.h>
#include <stdlib.h>
#include <strings.h>
#include <time.h>       /* funzioni time() e ctime() */
#include <unistd.h>
#include <sys/types.h>  /* tipi di dati di sistema */
#include <sys/socket.h> /* definizioni utili per le socket() */
#include <netinet/in.h>

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

int main(int argc, char *argv[]) {
     int sockfd, newsockfd;
     int portno, clilen;
     char buffer[256];
     struct sockaddr_in serv_addr, cli_addr;
     int n, counter;
     pid_t pid;

     counter = 0;
```

A simple server (2)

```
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");
}

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");
```

Address



A simple server (3)

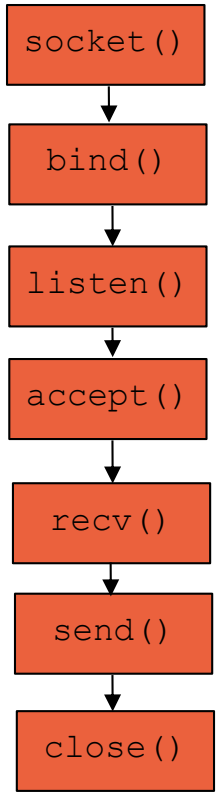
```
listen(sockfd,5);

do {
   clilen = sizeof(cli_addr);
   newsockfd = accept(sockfd,
               (struct sockaddr *) &cli_addr,
               (socklen_t *)&clilen);

   if (newsockfd < 0) {
         error("ERROR on accept");
   }

   pid=fork();
```

Crea una coda di attesa che può contenere al più 5 richieste di connesione

Chiamata bloccante

Crea un nuovo processo.
La chiamata restituisce 0 al figlio
e **pid** al padre (ID del nuovo processo)

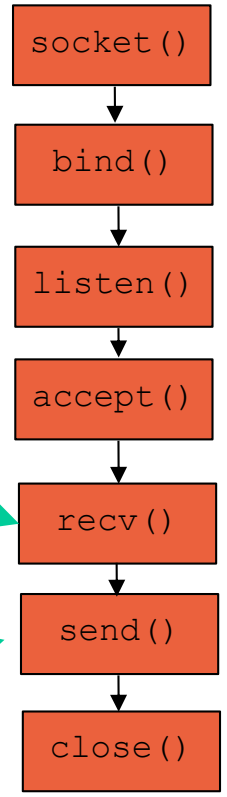
Qui il codice si deve differenziare a seconda che si tratti del processo padre (deve tornare in ascolto) o del processo figlio (deve elaborare le richieste)



A simple server (4)

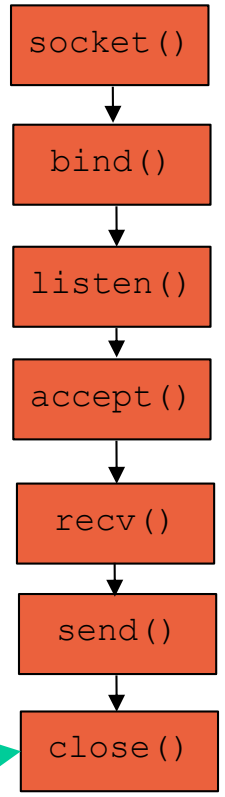
```
if (pid == 0) {

  close(sockfd);

  bzero(buffer,256);
  n = read(newsockfd,buffer,255);
  if (n < 0) {
    error("ERROR reading from socket");
  }

  if(!strcmp(buffer, "T\n")) {
    counter++;

    time_t t = time(NULL);

    char* timestr = ctime(&t);
    n = write(newsockfd, timestr, strlen(timestr)+1);
    if (n < 0) {
      error("ERROR writing to socket");
    }
  }
```

Il processo figlio chiude la socket di benvenuto



A simple server (5)

```
  else if(!strcmp(buffer, "N\n")) {
      char cnt[5];
    sprintf(cnt, "%d", counter);
    n = write(newsockfd, cnt, strlen(cnt));
    if (n < 0) {
      error("ERROR writing to socket");
    }
  }

  else {
    // delay
    printf("Please press a key...");
    char c;
    scanf("%c", &c);
    n = write(newsockfd, "Message received", 20);
    if (n < 0) {
      error("ERROR writing to socket");
    }
  }

  close(newsockfd);
  return 0;
}
```

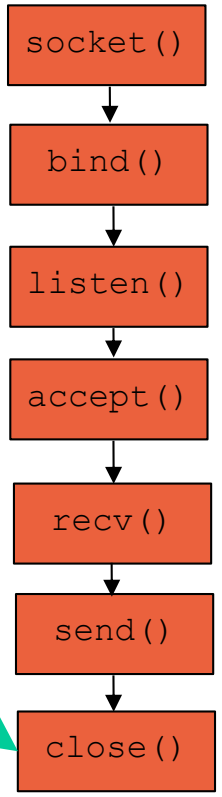


Il processo figlio chiude la socket di elaborazione e termina

A simple server (7)

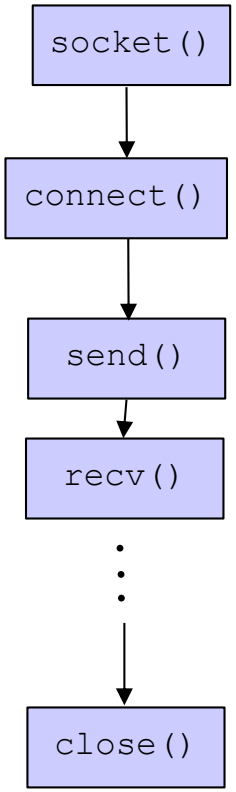
```
        close(newsockfd);


    } while (1);
    return 0;
}
```

Il processo padre chiude la socket di elaborazione (che non ha usato)
e si rimette in ascolto sulla socket di benvenuto (che non chiude)



A simple TCP client

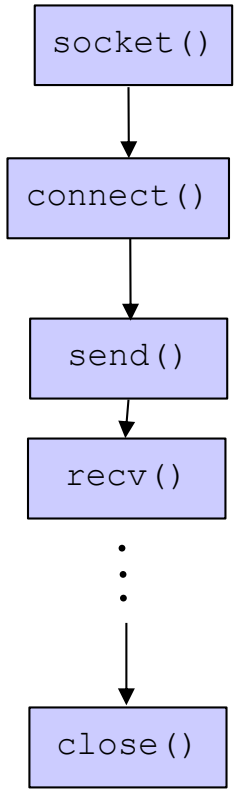
```c
#include <stdio.h>
#include <stdlib.h>     /* exit() */
#include <strings.h>    /* bzero(), bcopy() */
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

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

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");
```



A simple TCP client (2)

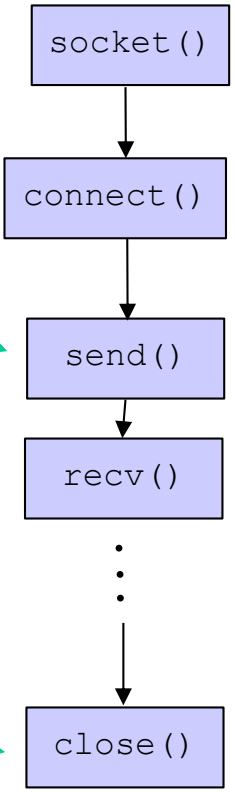
```
server = gethostbyname(argv[1]);
if (server == NULL) {
    fprintf(stderr,"ERROR, no such host\n");
    exit(0);
}

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);

if (connect(sockfd,(struct sockaddr*)&serv_addr,sizeof(serv_addr)) < 0)
    error("ERROR connecting");

printf("Please enter the command: ");
bzero(buffer,256);
fgets(buffer,255,stdin);
```

Address



A simple TCP client (3)

```
    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);

    close(sockfd);

    return 0;
}
```



Programmazione delle socket *con UDP*

UDP: non c'è "connessione" tra client e server

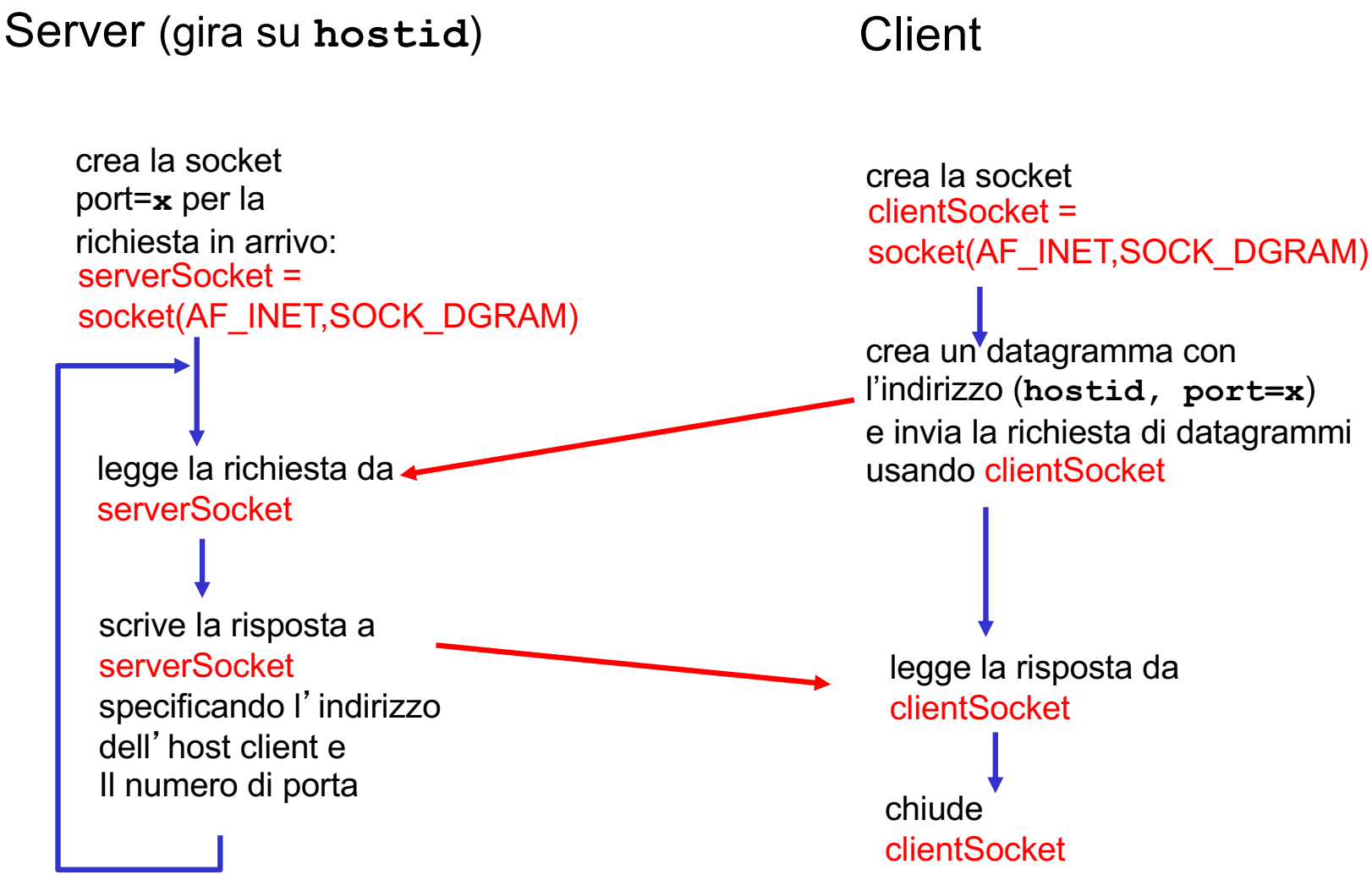
- Non c'è handshaking
- Il mittente allega esplicitamente a ogni pacchetto l'indirizzo IP e la porta di destinazione
- Il server deve estrarre l'indirizzo IP e la porta del mittente dal pacchetto ricevuto

UDP: i dati trasmessi possono perdersi o arrivare a destinazione in un ordine diverso da quello d'invio

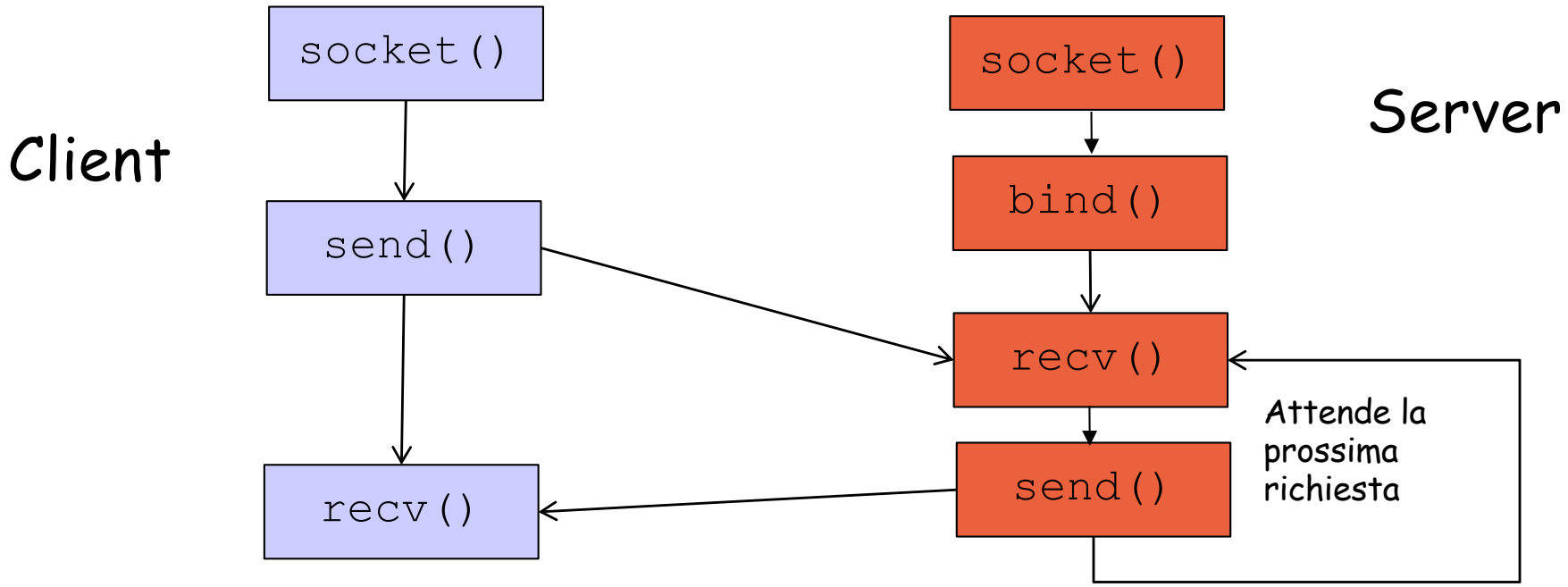
Punto di vista dell'applicazione

UDP fornisce un trasferimento inaffidabile di gruppi di byte ("datagrammi") tra client e server

Interazione delle socket client/server: UDP



Flusso della programmazione delle socket UDP

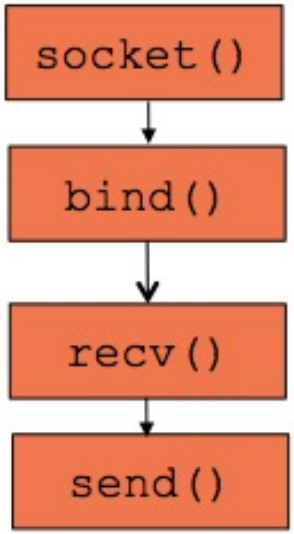


Un semplice server UDP (1)

```c
#include <stdio.h>
#include <stdlib.h>
#include <strings.h>
#include <time.h>       /* funzioni time() e ctime() */
#include <unistd.h>
#include <sys/types.h>  /* tipi di dati di sistema */
#include <sys/socket.h> /* definizioni utili per le socket() */
#include <netinet/in.h>

#define MAXLINE 4096

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

int main(int argc, char *argv[]) {
    int sockfd;
    int portno, clilen;
    int bytesread, counter;
    struct sockaddr_in serv_addr, cli_addr;
    char buffer[MAXLINE];
```

Un semplice server UDP (2)

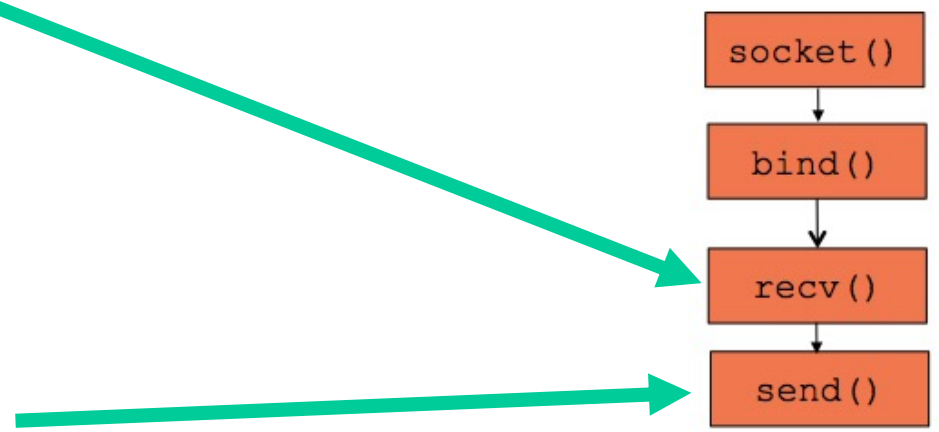
```
/* Creazione della socket:
 AF_INET indica la famiglia di protocolli usati da Internet;
 SOCK_DGRAM indica una socket non orientata alla connessione;
 0 e' l'identificativo del protocollo (con SOCK_DGRAM e' UDP).*/
sockfd = socket(AF_INET, SOCK_DGRAM, 0);
if (sockfd < 0) {
    error("ERROR opening socket");
}

/*Inizializzazione dei parametri della socket*/
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;
     /*indica che i dati sono accettati da qualsiasi indirizzo IP
serv_addr.sin_port = htons(portno);

/*Creazione della socket*/
if (bind(sockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) < 0) {
    error("ERROR on binding");
}
```



Un semplice Server UDP (3)

```
bytesread = recvfrom(sockfd, buffer, MAXLINE, 0, (struct sockaddr *)&cli_addr, (socklen_t *)&clilen);
```

Codice specifico dell'applicazione




```
sendto(sockfd, "ricevuto", 10, 0, (struct sockaddr *)&cli_addr, clilen);
```

Un semplice client UDP

```
    portno = atoi(argv[2]);
    sockfd = socket(AF_INET, SOCK_DGRAM, 0);
    if (sockfd < 0)
        error("ERROR opening socket");

    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);
        /* host-to-network short: converte un intero in un formato a 16 bit, indipendente dall'artchitettu
          big indian o little indian dell'host */
```

socket()

```
/*fgets legge una linea dallo standard input e la immagazzina nel buffer sendline*/
while (fgets(sendline, MAXLINE, stdin)) {
    /*Invia i dati letti dallo stream standard input al server*/
    sendto(sockfd, sendline, strlen(sendline), 0, (struct sockaddr*)&serv_addr,sizeof(serv_addr));

    /*Riceve una stringa in risposta che viene immagazzinata nel buffer recvline*/
    n = recvfrom(sockfd, recvline, MAXLINE, 0, NULL, NULL);

    recvline[n] = 0;

    /*Invia i dati ricevuti allo standard output*/
    fputs(recvline, stdout);
}
```

send()

receive()

Visualizzare processi e socket

- Visualizzare tutti processi in corso
  - >> ps –ax
- Visualizzare le socket aperte
  - Linux: >> netstat
  - Mac: >> netstat –nap TCP
- Terminare un processo
  - >> kill -9 <PID>

Esercizi – programmazione socket UDP

1. Nel codice del Server UDP fornito sul sito del corso c'è un errore logico. Quale? Correggere l'errore.

2. Modificare l'applicazione UDP (lato client e server) per fare visualizzare i parametri di rete di ciascun processo (indirizzo IP, numero di porta)

3. Realizzare un'applicazione UDP di tipo Echo, in cui il server restituisce al client la stringa che riceve, con i caratteri tutti in maiuscolo.

Esercizi – Programmazione socket TCP

1. Modificare l'applicazione TCP (lato client e server) per fare visualizzare i parametri di rete di ciascun processo (indirizzo IP, numero di porta)

2. Realizzare un'applicazione TCP di tipo Echo, in cui il server restituisce al client la stringa che riceve, con i caratteri tutti in maiuscolo.

3. Realizzare un'applicazione TCP per la realizzazione di una Chat. L'applicazione deve consentire sia al client che al server di mandare messaggi e di visualizzare i messaggi ricevuti, secondo uno schema alternato rigido (client-server-client-server ….).