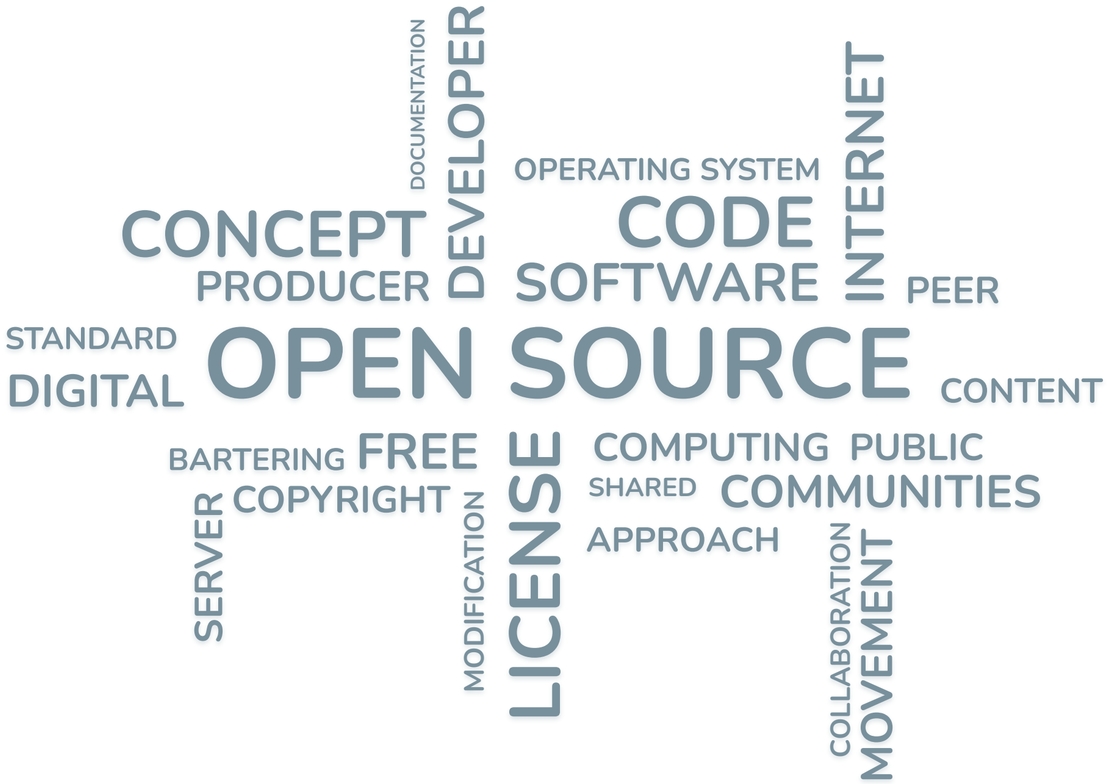
Università degli Studi di Cagliari


# Sistemi Operativi

## Modulo di Laboratorio 20


# 01

# Utilizzo dei socket in C



# Librerie da utilizzare

Per la creazione e l'utilizzo di socket è necessario importare diverse librerie:

- **`sys/types.h`**
  contiene le definizioni di un numero di tipi di dati usati nelle system calls da utilizzare.
- **`sys/socket.h`**
- **`netinet/in.h`**
  contiene le costanti e le strutture necessarie per gli indirizzi di dominio Internet.

In seguito sono riportate le principali funzioni da utilizzare quando si utilizzano dei socket.

# Creare un socket - (stream - datagram)

```
int socket (int *domain, int type, int protocol);
```

Questa funzione crea un endpoint per la comunicazione e restituisce un socket descriptor che rappresenta l'endpoint del socket.

- **domain** indica il dominio di comunicazione.
  Può avere diversi valori, tra i quali: **AF_INET** (IPv4 protocol), **AF_INET6** (IPv6 protocol), **AF_UNIX** (comunicazione locale).

- **type** indica il tipo di comunicazione.
  Può avere diversi valori, tra i quali: **SOCK_STREAM** (affidabile, bidirezionale, connection-based), **SOCK_DGRAM** (inaffidabile, connectionless).

- **protocol** rappresenta il protocollo da usare nel socket. Solitamente è utilizzato lo 0.

Attraverso questa funzione è possibile creare sia socket utilizzando i protocolli TCP (SOCK_STREAM) e  UDP (SOCK_DGRAM).

# SOCKADDR_IN

La struttura **sockaddr_in** è la struttura fondamentale per descrivere e gestire gli indirizzi internet, e quindi l'indirizzo della socket.

Viene definita come:

```
#include <netinet/in.h>

struct sockaddr_in {
   short           sin_family;   Indica la famiglia di indirizzi (e.g. IPv4 è AF_INET)
   unsigned short  sin_port;     Indica la porta che si vuole usare
   struct in_addr  sin_addr;     Serve indicare l'indirizzo con cui voglio comunicare
   char            sin_zero[8];  Padding per compatibilità con sockaddr
}

struct in_addr {
    unsigned long s_addr;
};
```

# Associare il socket ad un indirizzo - (stream - datagram)

```
int bind(int sockfd, const struct sockaddr* addr,
socklen_t addrlen);
```

Esegue il binding del socket specificato da **sockfd** all'indirizzo e al numero di porta specificati in addr.

- **sockfd** è il file descriptor che identifica la socket
- **addr** è un puntatore ad un’istanza della struttura **sockaddr** che conterrà l’indirizzo e la porta della macchina
- **addrlen** è la dimensione in byte della struttura **addr**

Restituisce 0 quando il binding ha avuto successo, mentre restituisce -1 quando c’è un errore.

# Ascolto su un socket passivo - (stream)

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

La funzione contrassegna il socket come un socket in ascolto, cioè un socket che verrà usato solamente per accettare le richieste di connessione.

- **sockfd** è il socket descriptor;
- **backlog** indica il numero massimo di client che possono effettuare una connessione al socket. In pratica, definisce la dimensione massima della coda delle connessioni in attesa sul socket.

Se una richiesta di connessione arriva quando la coda è piena il client potrebbe ricevere un errore oppure la richiesta potrebbe essere ignorata.

# Connessione ad una socket - (stream)

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

Questa funzione permette la connessione ad un socket. È utilizzata solo nel caso di socket stream.
Generalmente è effettuata dall'utente client, che tenta di connettersi ad un socket creato e inizializzato da un utente server.

- **sockfd** è il socket descriptor
- **addr** è l'indirizzo
- **addrlen** è la lunghezza di **addr**

Se la connessione ha successo viene restituito il valore 0, -1 altrimenti.
Questa funzione è bloccante.

# Connessione ad una socket - (stream)

```
int accept (int sockfd, struct sockadrr* addr,
socklen_t addrlen);
```

Questa chiamata accetta la connessione ad un socket (esclusivamente stream socket). È solitamente effettuata dall'utente che intende accettare la richiesta di **connect**() effettuata dall'utente che invia i messaggi.

- **sockfd**: il descrittore della socket in ascolto
- **addr**: indica l'indirizzo del client, solo a seguito dell'esecuzione della funzione punterà all'indirizzo corretto.
- **addrlen**: la dimensione dell'indirizzo del client.

Estrae la prima richiesta di connessione dalla coda di connessioni in attesa per il socket, crea un nuovo socket connesso e restituisce un nuovo file descriptor per riferirsi a questo socket. Il socket che resta in ascolto non è modificato.
Tale funzione è bloccante, in quanto attende una connessione.

# Scambio di dati in socket Stream (stream)

```
ssize_t send(int sockfd, const void* buf, size_t len,
int flags);

ssize_t recv(int sockfd, void* buf, size_t len, int
flags);
```

Se non ci sono messaggi nel socket, la funzione **recv** attende un nuovo messaggio e la funzione **send** la sblocca inviando il messaggio.

- **sockfd**: descriptor del socket su cui si vogliono scambiare messaggi
- **buf**: contiene il messaggio da leggere/trasmettere
- **len**: contiene la lunghezza in byte del messaggio
- **flags**: delle opzioni speciali, solitamente si imposta 0.

Queste chiamate sono bloccanti finché non ricevono o inviano i dati.

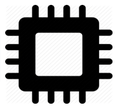
# Scambio di messaggi in socket Datagram (datagram)

```
ssize_t sendto(int sockfd, const void* buf, size_t len,
    int flags, const struct sockaddr* dest_addr,
    socklen_t addrlen);

ssize_t recvfrom(int sockfd, void* buf, size_t len,
    int flags, struct sockaddr* src_addr, socklen_t* addrlen);
```

- **sockfd**: descriptor del socket su cui si vogliono scambiare i dati
- **buf**: puntatore al messaggio
- **len**: lunghezza in bytes del messaggio
- **flags**: opzioni speciali, solitamente lasciate 0.
- **dest_addr**: indirizzo della destinazione
- **src_addr**: indirizzo del client
- **addrlen**: lunghezza dell'indirizzo (**dest_addr**, **src_addr**).

Queste chiamate sono bloccati.





# Chiusura di una socket - (stream - datagram)

```
int close (int fd);
```

La funzione **close** si occupa di chiudere il file descriptor in modo che esso non si riferisca più ad un file e possa essere riusato.

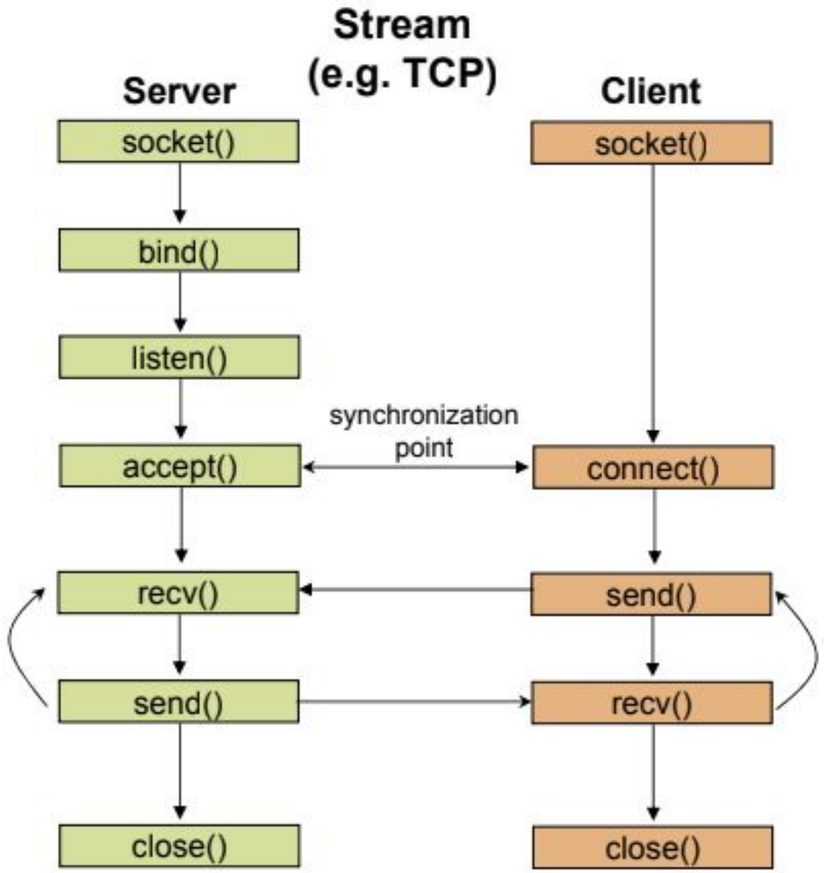
Restituisce 0 quando il file descriptor è chiuso con successo, -1 se c'è un errore.

Quando si chiude un socket si libera la porta usata dal socket e si chiude la connessione (quando si parla di stream socket).

# ESEMPIO DI UTILIZZO TCP SOCKET (1/8)

Per comprendere meglio il funzionamento dei socket, pensiamo alla casistica in cui un client vuole comunicare con un server attraverso un socket Stream.

Supponiamo per semplicità che il server rinvii semplicemente i messaggi ricevuti dal client.



# ESEMPIO DI UTILIZZO TCP SOCKET (2/8)

La sequenza di passi da effettuare sarà:

**SERVER**

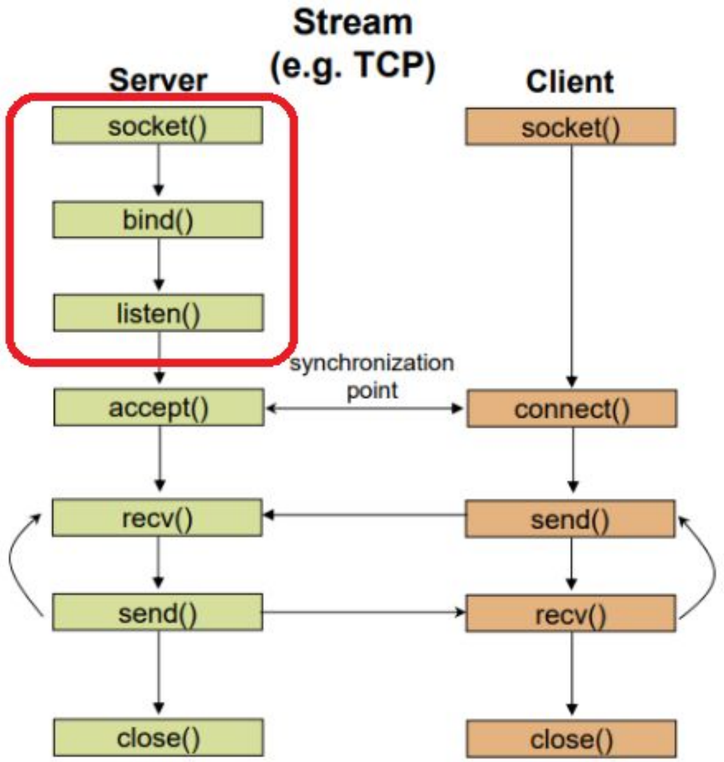
1. Creazione di un socket TCP
2. Assegnamento di un indirizzo al socket
3. Preparare il socket all'ascolto
4. Iterativamente:
   a. Accettare una nuova connessione
   b. Comunicare con il socket
   c. Chiudere la connessione

**CLIENT**

1. Creazione di un socket TCP
2. Stabilire una connessione con il server
3. Comunicare
4. Chiudere la connessione

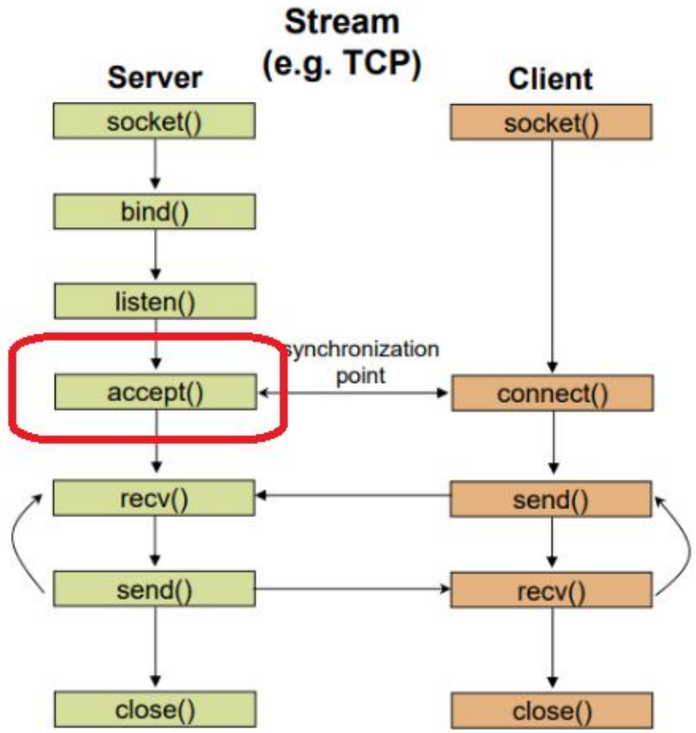
# ESEMPIO DI UTILIZZO TCP SOCKET (3/8)

```
struct sockaddr_in serverAddr;
//creazione di una nuova socket per le connessioni
serverSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)
if(serverSocket <0) {
   perror("socket() failed");
}

//dichiarazione dell'indirizzo della socket
serverAddr.sin_family=AF_INET; //Internet address family
serverAddr.sin_addr.s_addr=htonl(INADDR_ANY);//INADDR_ANY
restituisce l'indirizzo IP della macchina attuale, altrimenti
deve essere inserito l'indirizzo IP da utilizzare
serverAddr.sin_port=htons(serverPort); // porta locale
//htons converts a port number in host byte order to
a port number in network byte order
if(bind(serverSocket,(struct sockaddr*)&serverAddr,
   sizeof(serverAddr)) < 0){

   perror("bind() failed");
}

if(listen(serverSocket, MAXPENDING) < 0){

   perror("listen() failed");
}
```



# ESEMPIO DI UTILIZZO TCP SOCKET (4/8)

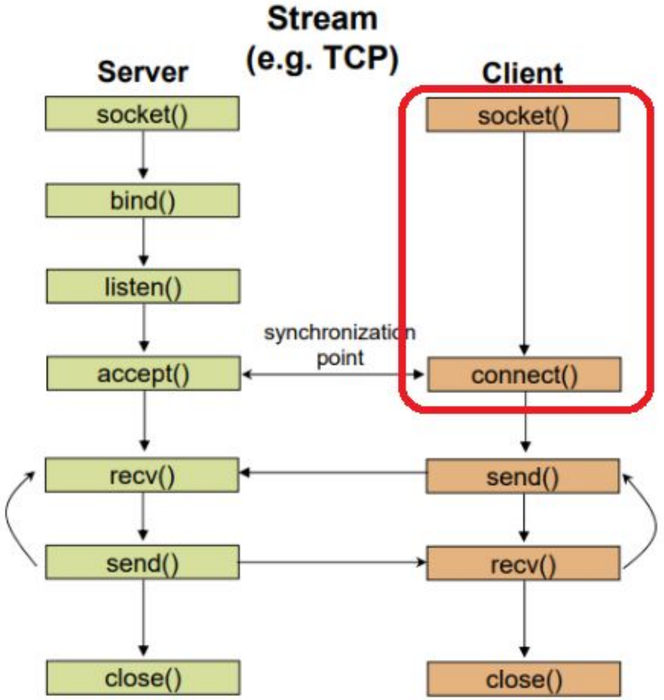
```
//ciclo infinito che si occuperà delle connessioni

for(;;){

   clientLen = sizeof(clientAddr);

   clientSocket = accept(serverSocket, (struct sockaddr*)
        &clientAddr, &clientLen);

   if(clientSocket < 0){

      perror("accept() failed");
   }

   // gestione concorrente tramite processi o thread

}
```




A questo punto il server è bloccato in attesa di una richiesta di connessione dal client.

# ESEMPIO DI UTILIZZO TCP SOCKET (5/8)

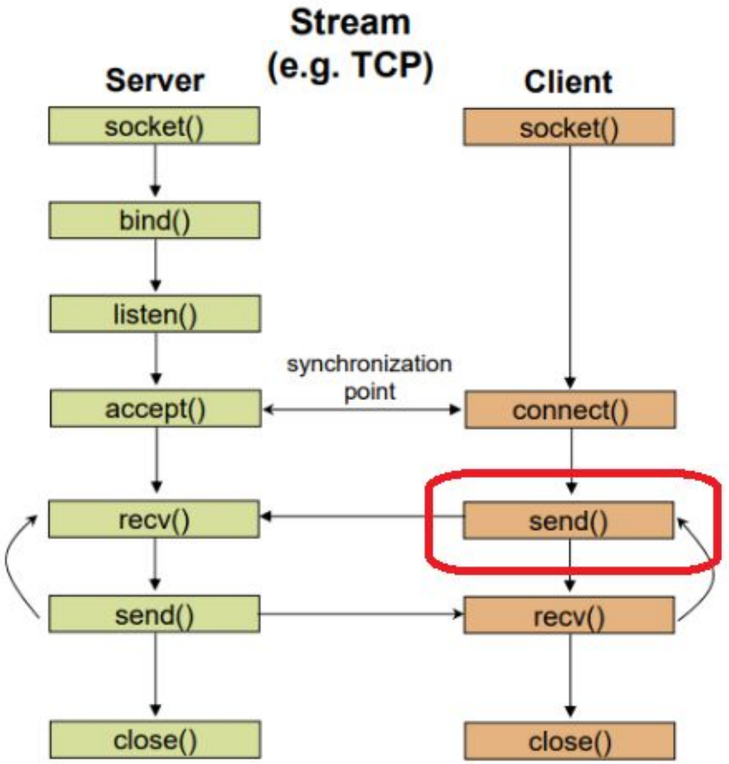
```
struct sockaddr_in serverAddr;
//creazione di una socket
clientSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
if(clientSocket < 0){

  perror("socket() failed");
}

serverAddr.sin_family = AF_INET;
// In generale per connettersi ad un server in un'altra macchina
bisogna specificare il suo IP, come nella seguente istruzione:
serverAddr.sin_addr.s_addr = inet_addr(serverIP);
// Però, se i due processi lavorano nella stessa macchina allora
 si può utilizzare l'istruzione seguente
serverAddr.sin_addr.s_addr = INADDR_ANY

serverAddr.sin_port = htons(serverPort);

if(connect(clientSocket, (struct sockaddr*) &serverAddr,
  sizeof(serverAddr)) < 0){

  perror("connect() failed");
}
```




Quando il client effettua la chiamata alla funzione connect, il server sarà sbloccato!

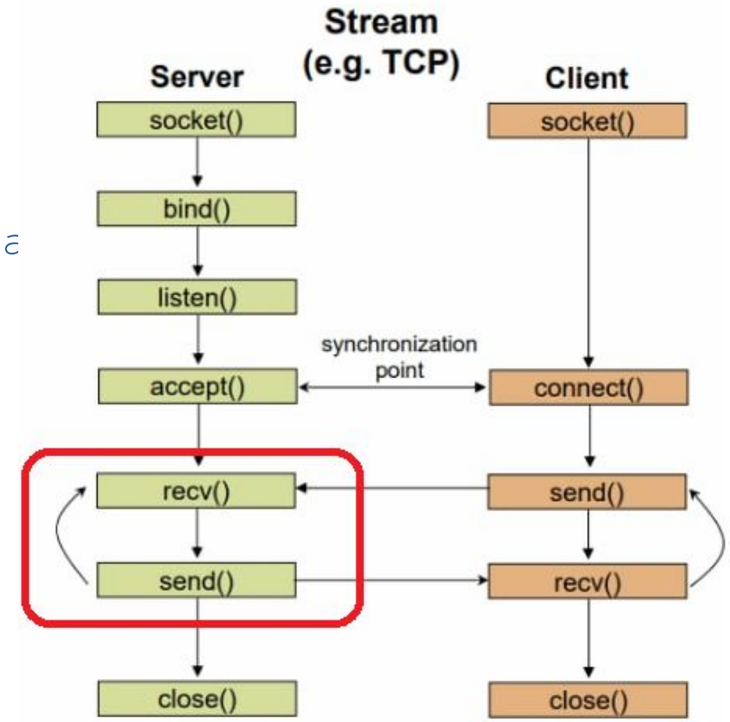
# ESEMPIO DI UTILIZZO TCP SOCKET (6/8)

```
//stringa è la variabile che conterrà la stringa che il
client invierà al server

stringLen = strlen(stringa);

len = send(clientSocket, stringa, stringLen, 0);

if ( len != stringLen ){

   perror("send() sent a different number of bytes
      than expected!");
}
```



# ESEMPIO DI UTILIZZO TCP SOCKET (7/8)

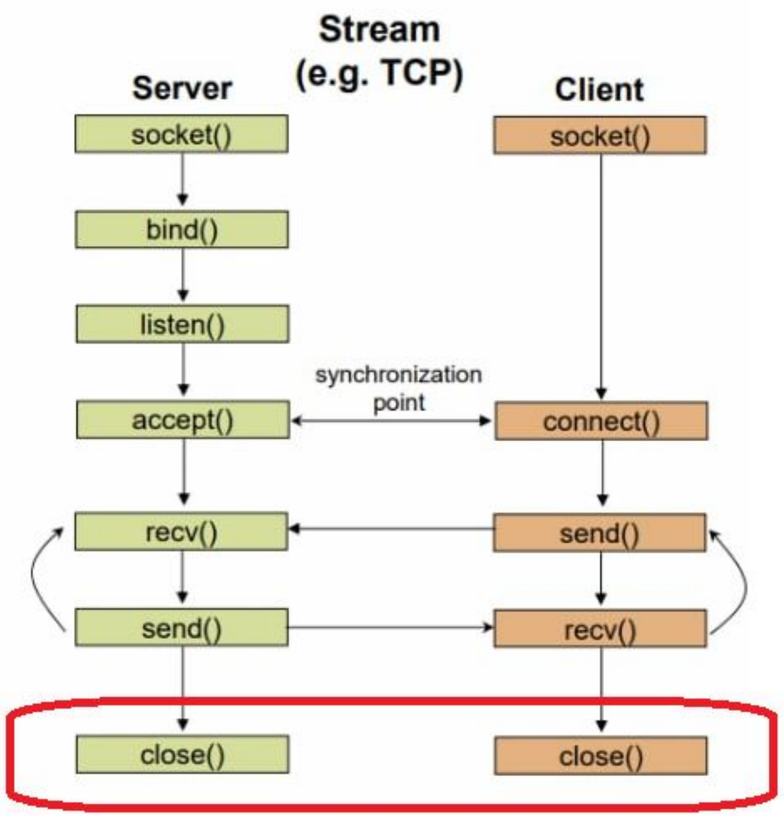
```
//riceve un messaggio dal client
recvMsgSize = recv(clientSocket, buffer, RCVBUFSIZE, 0);

if (recvMsgSize < 0)
  perror("recv() failed");


//rimanda il messaggio ricevuto e continua
della comunicazione
while(recvMsgSize > 0){

  len = send(clientSocket, buffer, recvMsgSize, 0);
  if( len != recvMsgSize )
    perror("send() failed");

  recvMsgSize = recv(clientSocket, buffer, RCVBUFSIZE, 0);
  if (recvMsgSize < 0)
    perror("recv() failed");

}
```




La comunicazione continuerà finché il messaggio ricevuto sarà vuoto.
Una situazione analoga si verifica quando è il client a ricevere i messaggi dal server.

# ESEMPIO DI UTILIZZO TCP SOCKET (8/8)

```
close(clientSocket);
```




Al termine della comunicazione, entrambi gli utenti dovranno chiudere il socket dedicata alla comunicazione.

# Esercitazione sui socket:

# TRIS

Utilizzando il linguaggio C, realizzare un'implementazione del gioco TRIS.
Due processi potranno sfidarsi durante una partita a questo gioco e dovranno comunicare tra loro tramite una socket. Per semplificare l'implementazione e il testing del codice, è sufficiente far comunicare due processi sulla **stessa** macchina.

```
addr.sin_addr.s_addr = INADDR_ANY;
```

Utilizzate una porta disponibile nella vostra macchina, come ad esempio la porta 8088.

Nella socket i due processi si passeranno la posizione della loro mossa e un parametro per indicare se l'utente ha vinto. Ad esempio, supponiamo che win sia un intero e position una coppia di coordinate:

```
send(socket, &position, 2*sizeof(int),0);
send(socket, &win, sizeof(int),0);
```

Fine