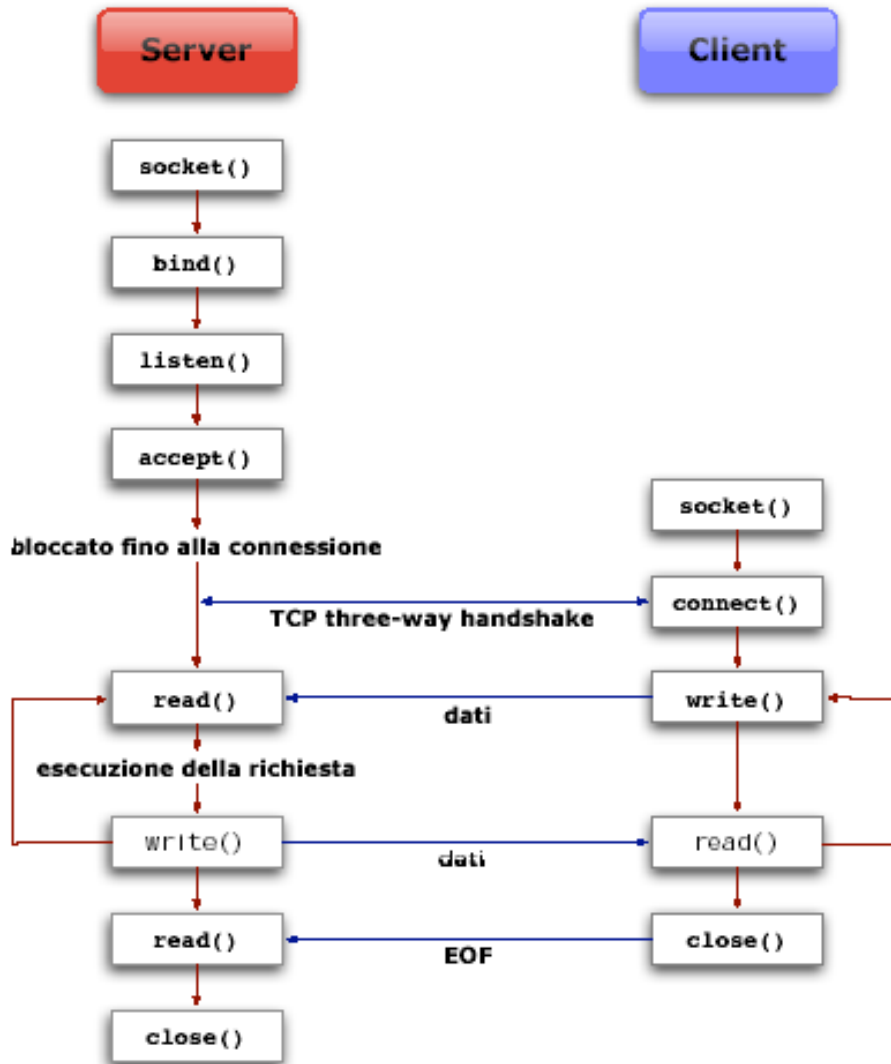


Interazione (TCP) Client-Server con le socket

Interazione TCP Client/Server



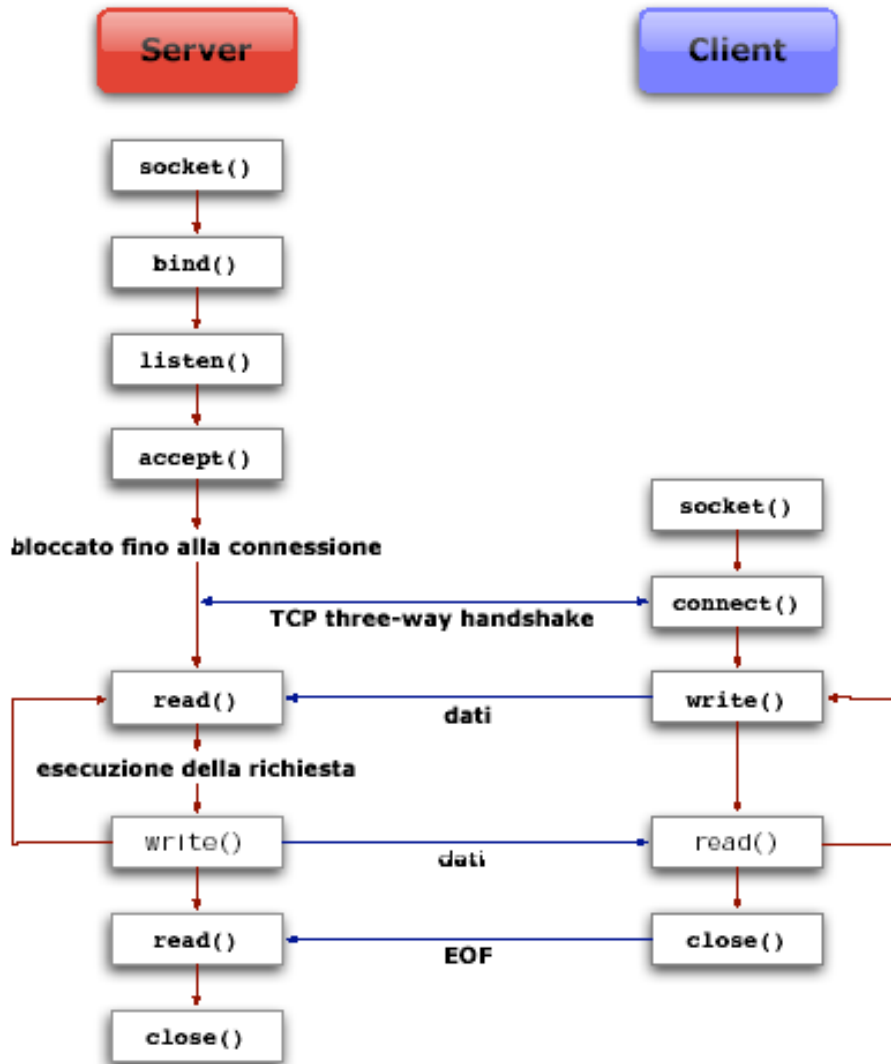
Server

1. Creare una socket
2. Assegnare un local address alla socket
3. Settare la socket all'ascolto
4. Iterativamente:
 - a. Accettare una nuova connessione
 - b. Inviare e ricevere dati
 - c. Chiudere la connessione

Client

1. Creare una socket
2. Connettersi al server
3. Inviare e ricevere dati
4. Chiudere la connessione

Interazione TCP Client/Server



Server

1. Creare una socket
2. Assegnare un local address alla socket
3. Settare la socket all'ascolto
4. Iterativamente:
 - a. Accettare una nuova connessione
 - b. Inviare e ricevere dati
 - c. Chiudere la connessione

Client

1. Creare una socket
2. Connettersi al server
3. Inviare e ricevere dati
4. Chiudere la connessione

Funzione socket()

- Crea una socket dedicata ad un fornitore di servizi specifico

```
int socket( int pf, int type, int protocol );
```

Famiglia di
protocolli

(**PF_INET**:
Internet
Protocol Family)

Tipo di socket

Particolare
protocollo da usare
con la socket per la
PF e il type indicati

(posto a **0** indica il
protocollo di default per
la coppia [pf, type])

Creazione di una socket

- Definire una variabile (int) che conterrà il descrittore della socket

```
int my_socket;
```

- Chiamare la funzione **socket()** assegnando il valore di ritorno alla variabile appena creata

```
my_socket = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
```

- Valutare la presenza di errori per assicurarsi che la socket sia valida

```
if (< 0) {  
    errorHandler("socket creation failed.\n");  
    return -1;  
}
```

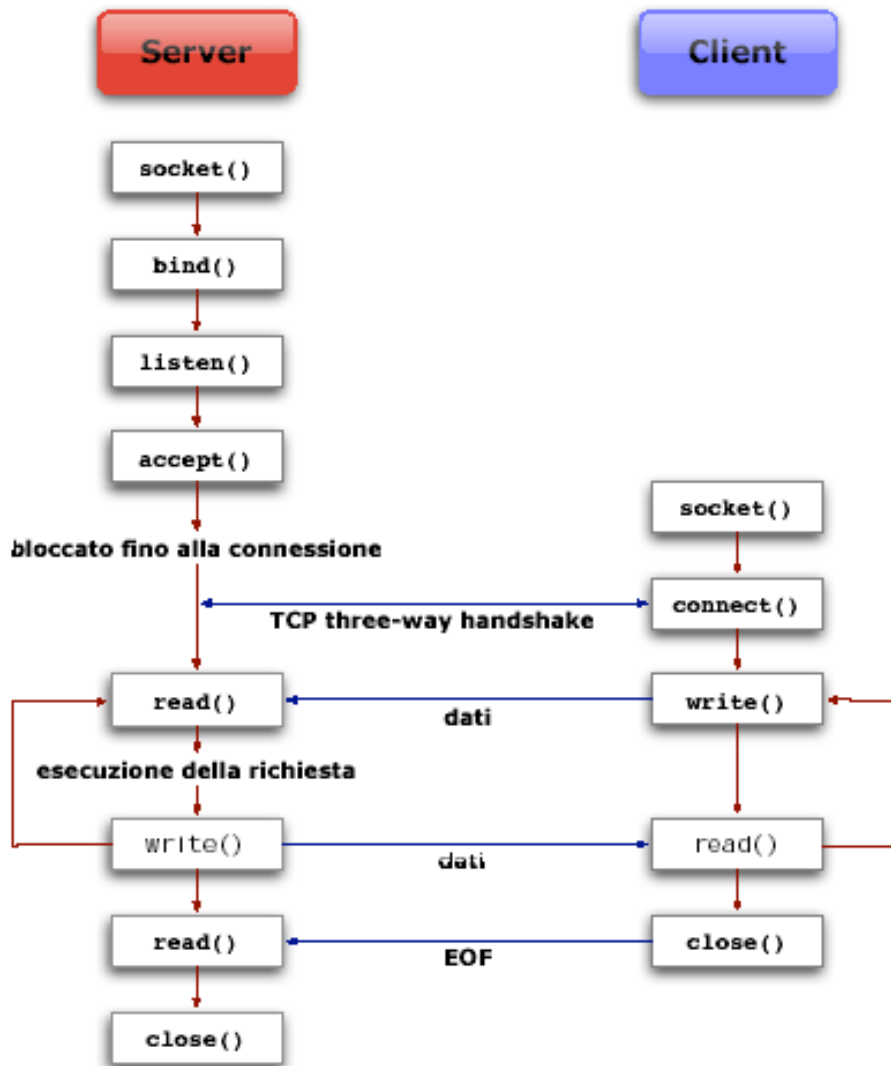
Tipi di Socket

<i>Type</i>	<i>Significato</i>
SOCK_STREAM	<p>Fornisce una connessione sequenziale, affidabile e full-duplex.</p> <p><i>Il protocollo TCP è basato su questo tipo di socket.</i></p>
SOCK_DGRAM	<p>Supporta i datagrammi (privo di connessione, messaggi inaffidabili di una lunghezza massima prefissata).</p> <p><i>Il protocollo UDP è basato su questo tipo di socket.</i></p>

Funzione socket(): valori di ritorno

- La funzione restituisce un **intero > 0** interpretato come un descrittore che referencia la nuova socket in caso di successo. Altrimenti restituisce **-1**
- In caso di esito positivo il valore di ritorno può essere considerato come un **socket descriptor** da passare ad altre funzioni dell'API per stabilire su quale socket eseguirle
- **NOTA:** il socket descriptor non è utilizzato dal Client per identificare la socket

Interazione TCP Client/Server



Server

1. Creare una socket
2. Assegnare un local address alla socket
3. Settare la socket all'ascolto
4. Iterativamente:
 - a. Accettare una nuova connessione
 - b. Inviare e ricevere dati
 - c. Chiudere la connessione

Client

1. Creare una socket
2. Connettersi al server
3. Inviare e ricevere dati
4. Chiudere la connessione

Assegnazione di un indirizzo alla socket (solo lato server)

- L'indirizzo di una socket è composto da
 - Numero di porta
 - Indirizzo IP
- Si utilizza la struttura dati ***sockaddr*** (16 byte)

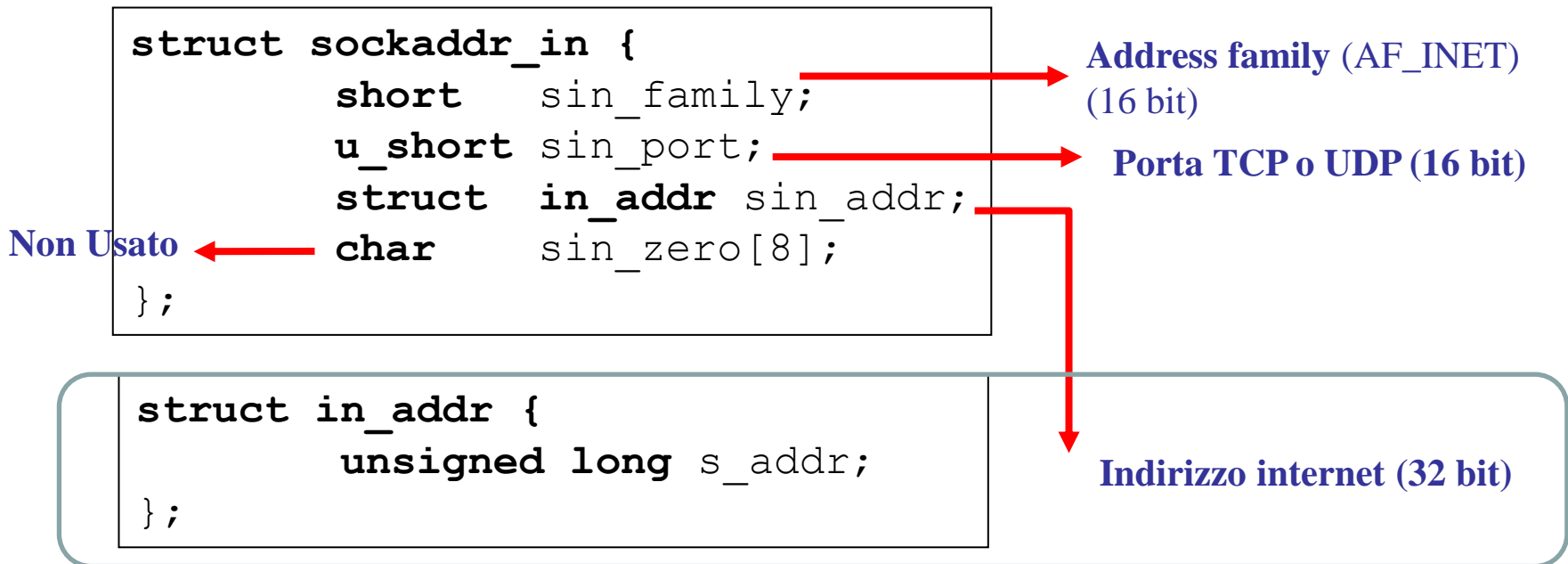
```
struct sockaddr {  
    unsigned short  sa_family;  
    char           sa_data[14];  
};
```

Address family (AF_INET)
(16 bit / 2 byte)

Struttura dati per
memorizzare l'indirizzo
specifico per la famiglia
scelta (14 byte)

sockaddr per le socket TCP/IP

- Si utilizza la struttura dati *sockaddr_in* (16 byte)



Assegnazione di un indirizzo alla socket (solo lato server)

- Creare un elemento di tipo `sockaddr_in`

```
struct sockaddr_in sad;
```

Converte un numero in notazione puntata in un numero a 32 bit

- Avvalorare l'elemento creato

```
sad.sin_family = AF_INET;  
sad.sin_addr.s_addr = inet_addr( "127.0.0.1" );  
sad.sin_port = htons( 27015 );
```

- Assegnare **porta** e **ip** alla socket e verificare la presenza di eventuali errori

```
if (bind(my_socket, (struct sockaddr*) &sad,  
    sizeof(sad)) < 0) {  
    errorhandler("bind() failed.\n");  
    closesocket(my_socket);  
    return -1;  
}
```

Converte un numero dal formato del computer locale a quello della rete (Big-Endian)

Conversione Big-Endian / Little-Endian

- Network Byte Order = Big Endian
- Host Byte Order = Big or Little Endian
- Funzioni per la conversione
 - htons() - "Host to Network Short"
 - htonl() - "Host to Network Long"
 - ntohs() - "Network to Host Short"
 - ntohl() - "Network to Host Long"


Funzione bind()

- Associa alla socket un indirizzo in modo da poter essere contattata da un client

```
int bind(int socket, struct sockaddr* localaddress, int addresslength);
```



Descrittore della
socket



Indirizzo da assegnare
(puntatore a una struttura
sockaddr)

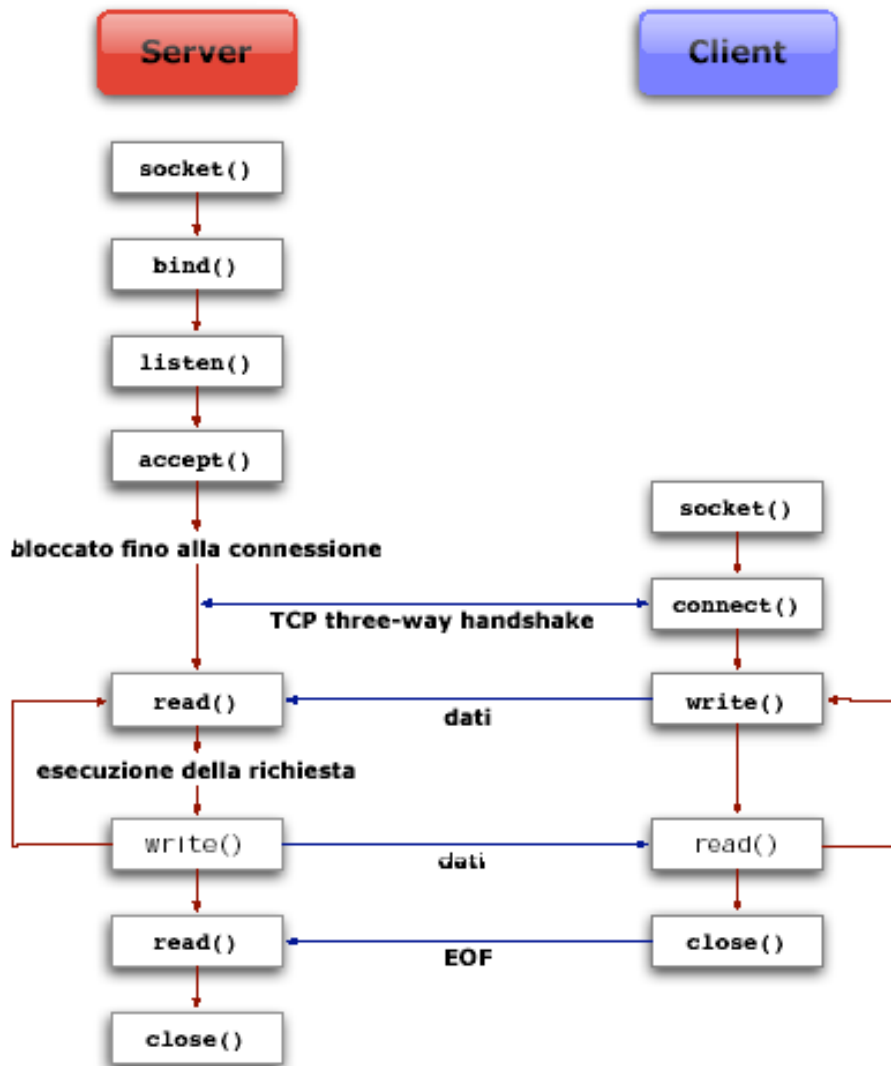


Lunghezza in byte
dell'indirizzo

Funzione bind(): valori di ritorno

- La funzione restituisce 0 in caso di successo, altrimenti -1
- **NOTA:**
 - Per TCP/IP se la porta è specificata come zero, il fornitore di servizi assegna una porta tra 1024 e 5000
 - L'applicazione server può usare la funzione **getsockname** (dopo la bind) per apprendere l'indirizzo IP e la porta assegnati

Interazione TCP Client/Server



Server

1. Creare una socket
2. Assegnare un local address alla socket
3. Settare la socket all'ascolto
4. Iterativamente:
 - a. Accettare una nuova connessione
 - b. Inviare e ricevere dati
 - c. Chiudere la connessione

Client

1. Creare una socket
2. Connettersi al server
3. Inviare e ricevere dati
4. Chiudere la connessione

Settare la socket all'ascolto (solo lato server)

- È necessario impostare la socket in modo che sia in grado di ascoltare le richieste di connessione
- Chiamare la funzione `listen()`, passando come parametri la socket creata e il massimo numero di connessioni pendenti

```
int qlen = 6;  
if (listen (my_socket, qlen) < 0) {  
    errorhandler("listen() failed.\n");  
    closesocket(my_socket);  
    return -1;  
}
```


Funzione listen()

- Setta la socket in uno stato in cui rimane in attesa di richiesta di connessioni

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



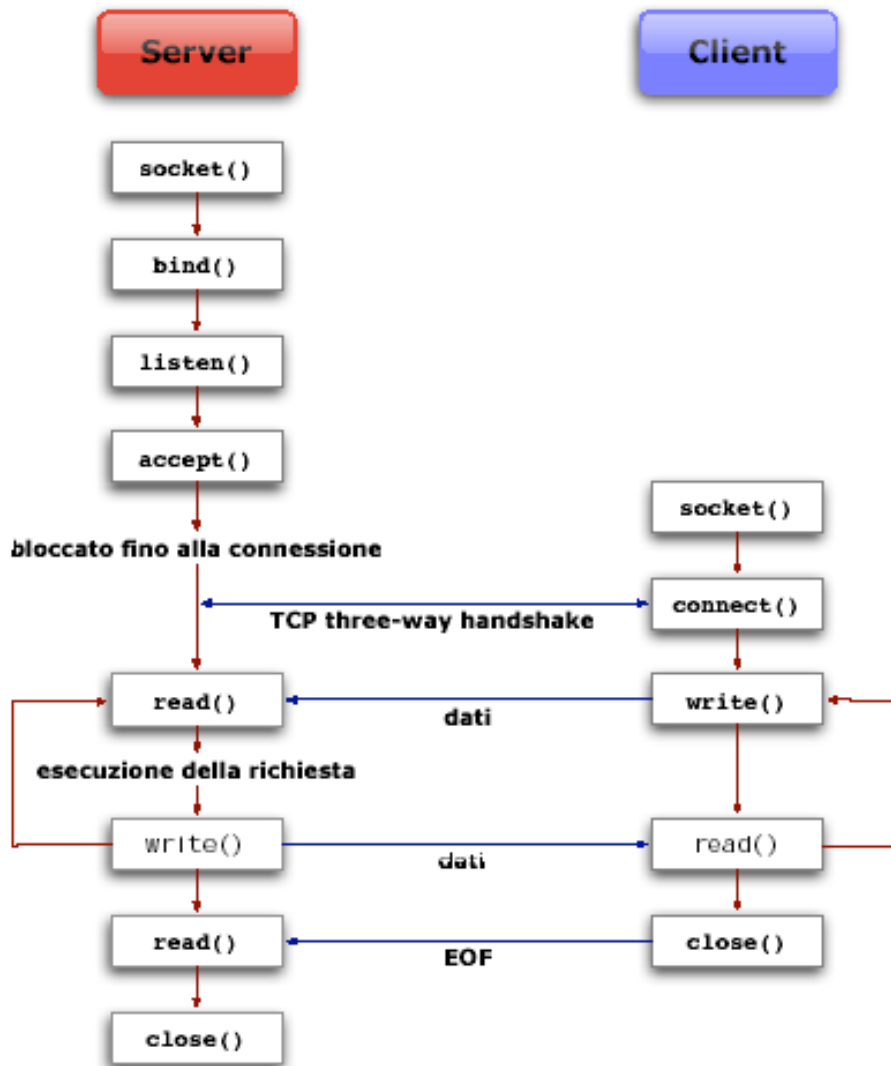
Descrittore
della socket



Massima lunghezza della
coda di connessioni
entranti

- La funzione restituisce 0 in caso di successo, altrimenti -1

Interazione TCP Client/Server



Server

1. Creare una socket
2. Assegnare un local address alla socket
3. Settare la socket all'ascolto
4. Iterativamente:
 - a. Accettare una nuova connessione
 - b. Inviare e ricevere dati
 - c. Chiudere la connessione

Client

1. Creare una socket
2. Connettersi al server
3. Inviare e ricevere dati
4. Chiudere la connessione

Accettare una nuova connessione (solo lato server)

- Creare un elemento dove poter memorizzare il descrittore di una socket temporanea a cui delegare la comunicazione con il client
- Effettuare un ciclo continuo per determinare se ci sono richieste di connessione e chiamare la funzione **accept()** per accettare eventuali connessioni
- Accettata una connessione, utilizzare la socket temporanea mentre l'originale resta in ascolto di altre connessioni


Funzione accept() (solo lato server)

- Consente un tentativo di connessione in entrata su una socket


```
int accept( int socket, struct sockaddr* addr, int* addrlen );
```



Descrittore
della socket



Puntatore opzionale ad
un buffer che riceve
l'indirizzo dell'entità
che fa richiesta di
connessione



Puntatore opzionale
che contiene la
lunghezza di addr

Funzione accept(): valori di ritorno (solo lato server)


- La funzione estrae la prima connessione dalla coda di pendenza delle connessioni sulla socket in input
- La funzione restituisce quindi il descrittore di una nuova socket connessa con il client
 - la socket in input non cambia e continua a restare in ascolto per nuove richieste

Accettare una nuova connessione (solo lato server)

```
/* Main server loop - accept and handle requests */
struct sockaddr_in cad; //structure for the client address
int client_socket;      //socket descriptor for the client
int client_len;         //the size of the client address
printf( "Waiting for a client to connect...");

while (1) {
    client_len = sizeof(cad); //set the size of the client address
    if ( (client_socket=accept(my_socket, (struct sockaddr *)&cad,
    &client_len)) < 0 ) {
        errorhandler("accept() failed.\n");
        closesocket(my_socket);
        clearwinsock();
        return -1;
    }
    // clientSocket is connected to a client
    printf( "Handling client %s\n", inet_ntoa(cad.sin_addr) );
    handleclientconnection(client_socket);
} // end of the while loop
```

Al contrario di `inet_addr()`,
converte un numero a 32 bit in
un numero in notazione puntata



Un esempio di codice (server)...

```
#if defined WIN32
#include <winsock.h>
#else
#define closesocket close
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>
#endif
#include <stdio.h>
#include <stdlib.h> // for atoi()
#define PROTOPORT 27015 // default protocol port number
#define QLEN 6 // size of request queue

void errorhandler(char *errorMessage) {
    printf ("%s", errorMessage);
}

void clearwinsock() {
    #if defined WIN32
    WSACleanup();
    #endif
}
```

...un esempio di codice (server)...

```
int main(int argc, char *argv[]) {
    int port;
    if (argc > 1) {
        port = atoi(argv[1]); // if argument specified convert argument to
        binary
    }
    else
        port = PROTOPORT; // use default port number
    if (port < 0) {
        printf("bad port number %s \n", argv[1]);
        return 0;
    }

    #if defined WIN32 // initialize Winsock
    WSADATA wsa_data;
    int result = WSASStartup(MAKEWORD(2,2), &wsa_data);

    if (result != 0) {
        errorhandler("Error at WSASStartup()\n");
        return 0;
    }
    #endif
}
```


...un esempio di codice (server)...

```
// CREAZIONE DELLA SOCKET
int my_socket;
my_socket = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
if (my_socket < 0) {
    errorhandler("socket creation failed.\n");
    clearwinsock();
    return -1;
}

// ASSEGNAZIONE DI UN INDIRIZZO ALLA SOCKET
struct sockaddr_in sad;
memset(&sad, 0, sizeof(sad)); // ensures that extra bytes contain 0
sad.sin_family = AF_INET;
sad.sin_addr.s_addr = inet_addr("127.0.0.1");
sad.sin_port = htons(port); /* converts values between the host and
network byte order. Specifically, htons() converts 16-bit quantities
from host byte order to network byte order. */
if (bind(my_socket, (struct sockaddr*) &sad, sizeof(sad)) < 0) {
    errorhandler("bind() failed.\n");
    closesocket(my_socket);
    clearwinsock();
    return -1;
}
```

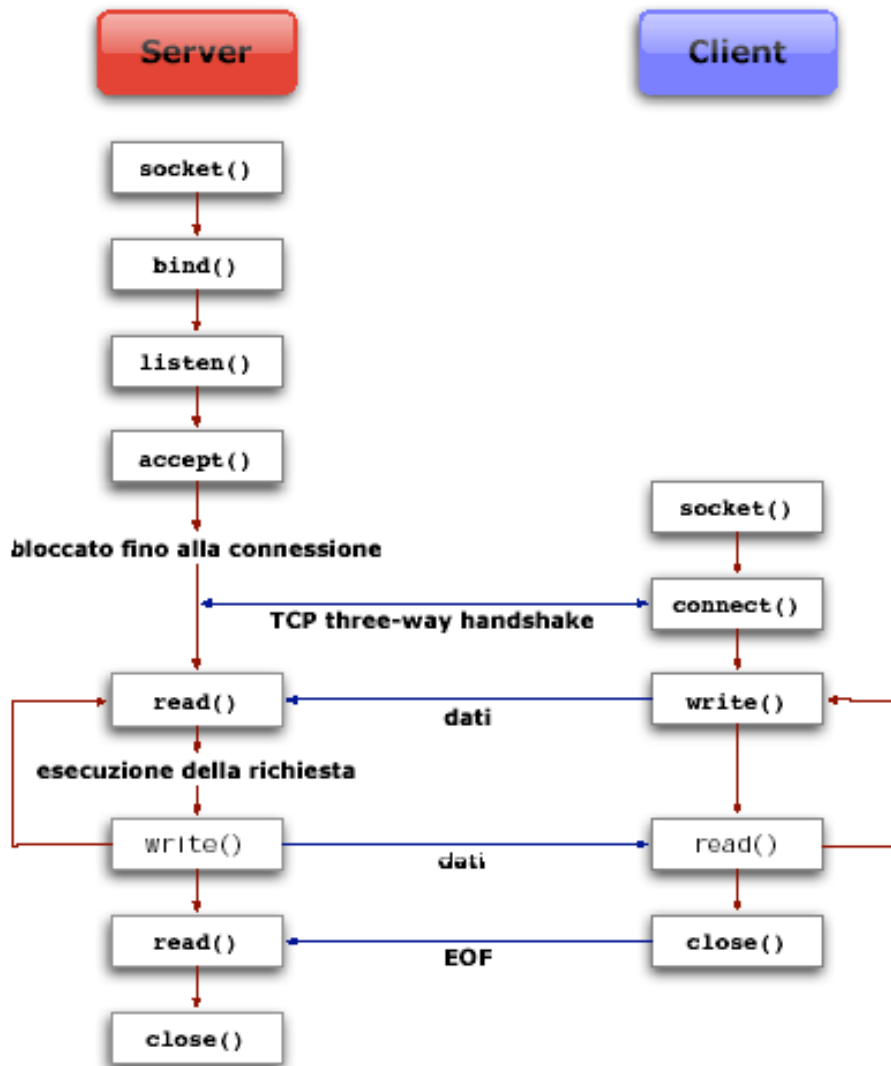
```

// SETTAGGIO DELLA SOCKET ALL'ASCOLTO
if (listen (my_socket, QLEN) < 0) {
    ErrorHandler("listen() failed.\n");
    closesocket(my_socket);
    ClearWinSock();
    return -1;
}

// ACCETTARE UNA NUOVA CONNESSIONE
struct sockaddr_in cad; // structure for the client address
int client_socket;      // socket descriptor for the client
int client_len;         // the size of the client address
printf("Waiting for a client to connect...");
while (1) { /* oppure for (;;) */
    client_len = sizeof(cad); // set the size of the client
address
    if ((client_socket = accept(my_socket, (struct sockaddr
*)&cad, &client_len)) < 0) {
        errorhandler("accept() failed.\n");
        // CHIUSURA DELLA CONNESSIONE
        closesocket(client_socket);
        clearwinsock();
        return 0;
    }
    printf("Handling client %s\n", inet_ntoa(cad.sin_addr));
} // end-while
} // end-main

```

Interazione TCP Client/Server



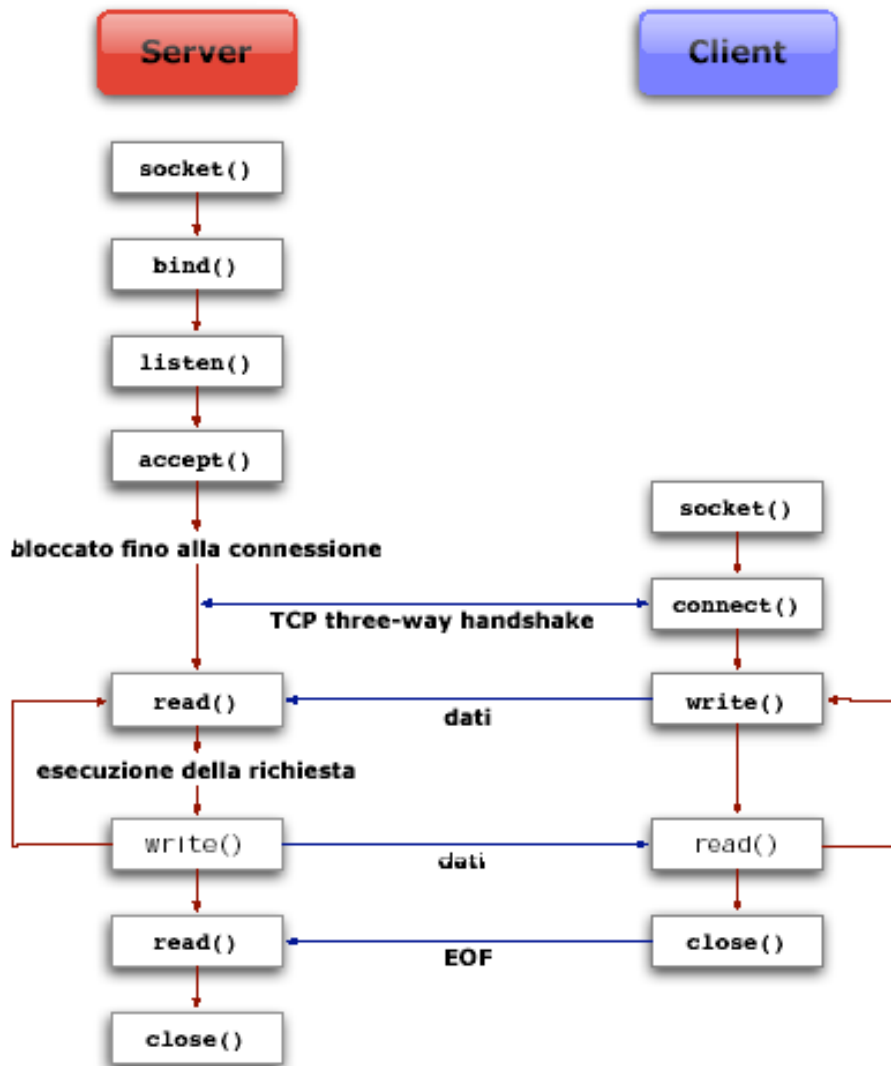
Server

1. Creare una socket
2. Assegnare un local address alla socket
3. Settare la socket all'ascolto
4. Iterativamente:
 - a. Accettare una nuova connessione
 - b. Inviare e ricevere dati
 - c. Chiudere la connessione

Client

1. Creare una socket
2. Connettersi al server
3. Inviare e ricevere dati
4. Chiudere la connessione

Interazione TCP Client/Server



Server

1. Creare una socket
2. Assegnare un local address alla socket
3. Settare la socket all'ascolto
4. Iterativamente:
 - a. Accettare una nuova connessione
 - b. Inviare e ricevere dati
 - c. Chiudere la connessione

Client

1. Creare una socket
2. Connettersi al server
3. Inviare e ricevere dati
4. Chiudere la connessione


Funzione connect() (solo lato client)

- Stabilisce una connessione ad una socket specificata (attraverso un indirizzo)

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



Descrittore di
una socket (non
ancora connessa)



Iddirizzo della socket
con cui dovrebbe
essere stabilita la
connessione



lunghezza di `addr`

- Restituisce 0 in caso di successo,
altrimenti -1

Connettersi al server (solo lato client)

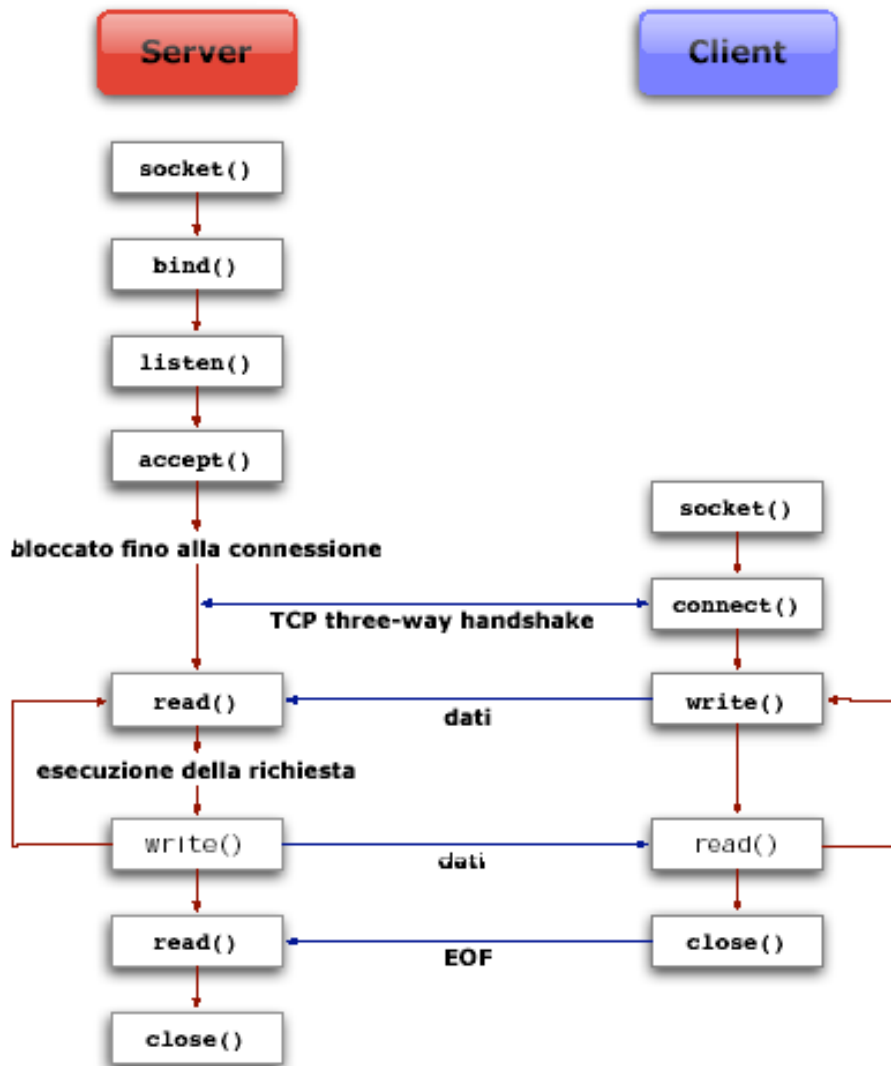
- Creare un elemento di tipo `sockaddr_in` per fare riferimento alla socket a cui connettersi

```
int c_socket;  
sockaddr_in sad;  
sad.sin_family = AF_INET;  
sad.sin_addr.s_addr = inet_addr( "127.0.0.1" );  
sad.sin_port = htons( 27015 );
```

- Chiamare la funzione **connect ()** passando come parametri una socket già creata e la struttura `sockaddr_in` che identifica una socket settata all'ascolto

```
if (connect(c_socket, (struct sockaddr *) &sad, sizeof(sad)) < 0)  
{  
    errorHandler( "Failed to connect.\n" );  
    return 0;  
}
```

Interazione TCP Client/Server



Server

1. Creare una socket
2. Assegnare un local address alla socket
3. Settare la socket all'ascolto
4. Iterativamente:
 - a. Accettare una nuova connessione
 - b. Inviare e ricevere dati
 - c. Chiudere la connessione

Client

1. Creare una socket
2. Connettersi al server
3. Inviare e ricevere dati
4. Chiudere la connessione

Inviare dati: funzione send()

- Invia dati ad una socket connessa
- La funzione restituisce il numero di byte trasmessi in caso di successo, altrimenti un valore ≤ 0

```
int send( int socket, const char* buf, int len, int flags );
```

Descrittore di
una socket
connessa

Puntatore al buffer
contenente i dati da
trasmettere

Indicatore che
specifica il modo in cui
la chiamata è fatta

Dimensione
del buffer in
buf, in byte

- Il flag può essere usato per influenzare il comportamento della funzione

Ricevere dati: funzione recv()

- Riceve dati da una socket connessa (o "legata")
- La funzione restituisce il numero di byte ricevuti in caso di successo, altrimenti un valore ≤ 0

```
int recv( int socket, char* buf, int len, int flags );
```

Descrittore di
una socket
connessa

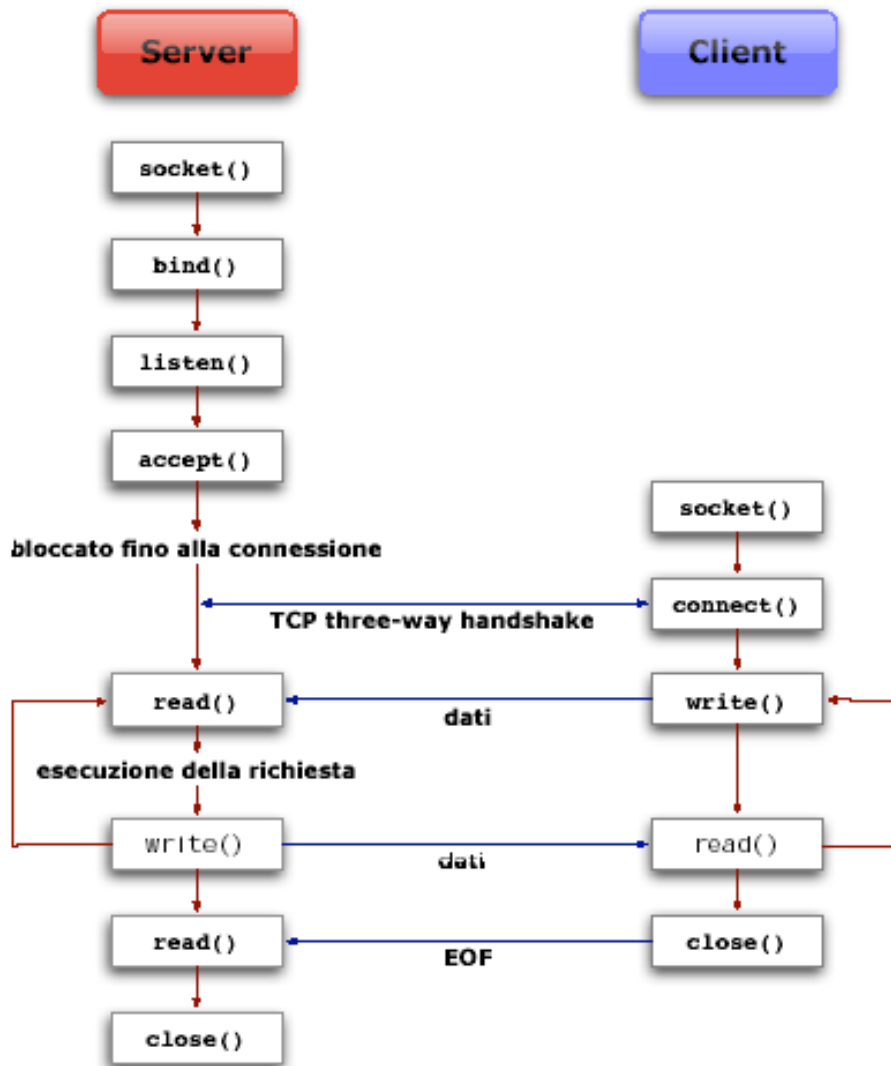
Puntatore al buffer
che conterrà i dati da
ricevere

Indicatore che
specifica il modo in cui
la chiamata è fatta

Dimensione
del buffer in
buf, in byte

- Il flag può essere usato per influenzare il comportamento della funzione

Interazione TCP Client/Server



Server

1. Creare una socket
2. Assegnare un local address alla socket
3. Settare la socket all'ascolto
4. Iterativamente:
 - a. Accettare una nuova connessione
 - b. Inviare e ricevere dati
 - c. Chiudere la connessione

Client

1. Creare una socket
2. Connettersi al server
3. Inviare e ricevere dati
4. Chiudere la connessione

Chiudere la connessione

- Quando si è conclusa l'interazione bisogna comunicare al livello sottostante di interrompere la comunicazione e deallocare le risorse

- In Windows

```
int closesocket (int socket);
```

- In Unix

```
int close (int socket);
```

- Restituisce 0 in caso di successo
- Restituisce -1 in caso di errore

Un esempio di codice (client)...

```
#if defined WIN32
#include <winsock.h>
#else
#define closesocket close
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>
#endif
#include <stdio.h>
#include <stdlib.h>
#define BUFFERSIZE 512
#define PROTOPORT 27015 // Numero di porta di default

void errorhandler(char *error_message) {
printf("%s",error_message);
}

void clearwinsock() {
#if defined WIN32
WSACleanup();
#endif
}
```

...un esempio di codice (client)...

```
int main(void) {
#ifdef WIN32
WSADATA wsa_data;
int result = WSStartup(MAKEWORD(2,2), &wsa_data);

if (result != 0) {
    printf ("error at WSASTurtup\n");
    return -1;
}
#endif

// CREAZIONE DELLA SOCKET
int c_socket;
c_socket = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
if (c_socket < 0) {
    errorhandler("socket creation failed.\n");
    closesocket(c_socket);
    clearwinsock();
    return -1;
}
```

...un esempio di codice (client)...

```
// COSTRUZIONE DELL'INDIRIZZO DEL SERVER
struct sockaddr_in sad;
memset(&sad, 0, sizeof(sad));
sad.sin_family = AF_INET;
sad.sin_addr.s_addr = inet_addr("127.0.0.1"); // IP del server
sad.sin_port = htons(27015); // Server port

// CONNESSIONE AL SERVER
if (connect(c_socket, (struct sockaddr *)&sad, sizeof(sad)) < 0)
{
    errorHandler( "Failed to connect.\n" );
    closesocket(c_socket);
    clearwinsock();
    return -1;
}

char* input_string = "prova"; // Stringa da inviare
int string_len = strlen(input_string); // Determina la lunghezza
```

...un esempio di codice (client)...

```
// INVIARE DATI AL SERVER
if (send(c_socket, input_string, string_len, 0) != string_len)
{
    errorhandler("send() sent a different number of bytes
than expected");
    closesocket(c_socket);
    clearwinsock();
    return -1;
}

// RICEVERE DATI DAL SERVER
int bytes_rcvd;
int total_bytes_rcvd = 0;
char buf[BUFFERSIZE];    // buffer for data from the server
printf("Received: ");    // Setup to print the echoed string
```

...un esempio di codice (client)

```
while (total_bytes_rcvd < string_len) {
    if ((bytes_rcvd = recv(c_socket, buf, BUFFERSIZE - 1, 0)) <= 0)
    {
        errorHandler("recv() failed or connection closed
prematurely");
        closesocket(c_socket);
        clearwinsock();
        return -1;
    }
    total_bytes_rcvd += bytes_rcvd; // Keep tally of total bytes
    buf[bytes_rcvd] = '\0'; // Add \0 so printf knows where to stop
    printf("%s", buf); // Print the echo buffer
}

// CHIUSURA DELLA CONNESSIONE
closesocket(c_socket);
clearwinsock();
printf("\n"); // Print a final linefeed
system("pause");
return (0);
}
```