

Socket I

Leonardo Querzoni

querzoni@dis.uniroma1.it
<http://www.dis.uniroma1.it/~querzoni>

MIDLAB

<http://www.dis.uniroma1.it/~midlab>

Sommario

- Strutture dati
- Funzioni
- Echo client/server
 - * Echo server TCP
 - * Echo client TCP
 - * Echo server UDP
 - * Echo client UDP
- Appendice

Strutture dati: sockaddr_in

Molte delle system call che vedremo utilizzano questa struttura dati per memorizzare informazioni relative ad un indirizzo IPv4.

```
struct sockaddr_in {
    u_char   sin_len;
    u_char   sin_family;
    u_short  sin_port;
    struct   in_addr sin_addr;
    char     sin_zero[8];
};

struct in_addr {
    in_addr_t s_addr;
};
```

sockaddr_in è definita in <netinet/in.h>

ATTENZIONE: ricordatevi sempre di inizializzare la struttura prima di usarla !

Ad esempio:

```
struct   sockaddr_in   servaddr;
memset(&servaddr, 0, sizeof(servaddr));
servaddr.sin_family    = AF_INET;
```

Strutture dati: sockaddr_in

Molte delle system call che vedremo utilizzano questa struttura dati per memorizzare informazioni relative ad un indirizzo IPv4.

```
struct sockaddr_in {
    u_char   sin_len;
    u_char   sin_family;
    u_short  sin_port;
    struct   in_addr sin_addr;
    char     sin_zero[8];
};

struct in_addr {
    in_addr_t s_addr;
};
```

Contiene la lunghezza della struttura stessa. È un parametro indispensabile per le strutture a lunghezza variabile. Viene calcolato automaticamente dal SO.

sockaddr_in è definita in <netinet/in.h>

ATTENZIONE: ricordatevi sempre di inizializzare la struttura prima di usarla !

Ad esempio:

```
struct   sockaddr_in   servaddr;
memset(&servaddr, 0, sizeof(servaddr));
servaddr.sin_family    = AF_INET;
```

Strutture dati: sockaddr_in

Molte delle system call che vedremo utilizzano questa struttura dati per memorizzare informazioni relative ad un indirizzo IPv4.

```
struct sockaddr_in {
    u_char  sin_len;
    u_char  sin_family;
    u_short sin_port;
    struct  in_addr sin_addr;
    char    sin_zero[8];
};
```

Contiene la "famiglia" di indirizzi a cui questo indirizzo appartiene (valori del tipo AF_XXX).

```
struct in_addr {
    in_addr_t s_addr;
};
```

sockaddr_in è definita in <netinet/in.h>

ATTENZIONE: ricordatevi sempre di inizializzare la struttura prima di usarla !

Ad esempio:

```
struct sockaddr_in servaddr;
memset(&servaddr, 0, sizeof(servaddr));
servaddr.sin_family = AF_INET;
```

Strutture dati: sockaddr_in

Molte delle system call che vedremo utilizzano questa struttura dati per memorizzare informazioni relative ad un indirizzo IPv4.

```
struct sockaddr_in {
    u_char  sin_len;
    u_char  sin_family;
    u_short sin_port;
    struct  in_addr sin_addr;
    char    sin_zero[8];
};
```

Indica una porta TCP o UDP con formato *network byte ordered*.

```
struct in_addr {
    in_addr_t s_addr;
};
```

sockaddr_in è definita in <netinet/in.h>

ATTENZIONE: ricordatevi sempre di inizializzare la struttura prima di usarla !

Ad esempio:

```
struct sockaddr_in servaddr;
memset(&servaddr, 0, sizeof(servaddr));
servaddr.sin_family = AF_INET;
```

Strutture dati: sockaddr_in

Molte delle system call che vedremo utilizzano questa struttura dati per memorizzare informazioni relative ad un indirizzo IPv4.

```
struct sockaddr_in {  
    u_char  sin_len;  
    u_char  sin_family;  
    u_short sin_port;  
    struct  in_addr sin_addr;  
    char    sin_zero[8];  
};
```

```
struct in_addr {  
    in_addr_t s_addr;  
};
```

Indica un indirizzo IPv4 in formato *network byte ordered*.

sockaddr_in è definita in <netinet/in.h>

ATTENZIONE: ricordatevi sempre di inizializzare la struttura prima di usarla !

Ad esempio:

```
struct  sockaddr_in  servaddr;  
memset(&servaddr, 0, sizeof(servaddr));  
servaddr.sin_family = AF_INET;
```

Strutture dati: sockaddr_in

Molte delle system call che vedremo utilizzano questa struttura dati per memorizzare informazioni relative ad un indirizzo IPv4.

```
struct sockaddr_in {  
    u_char  sin_len;  
    u_char  sin_family;  
    u_short sin_port;  
    struct  in_addr sin_addr;  
    char    sin_zero[8];  
};
```

```
struct in_addr {  
    in_addr_t s_addr;  
};
```

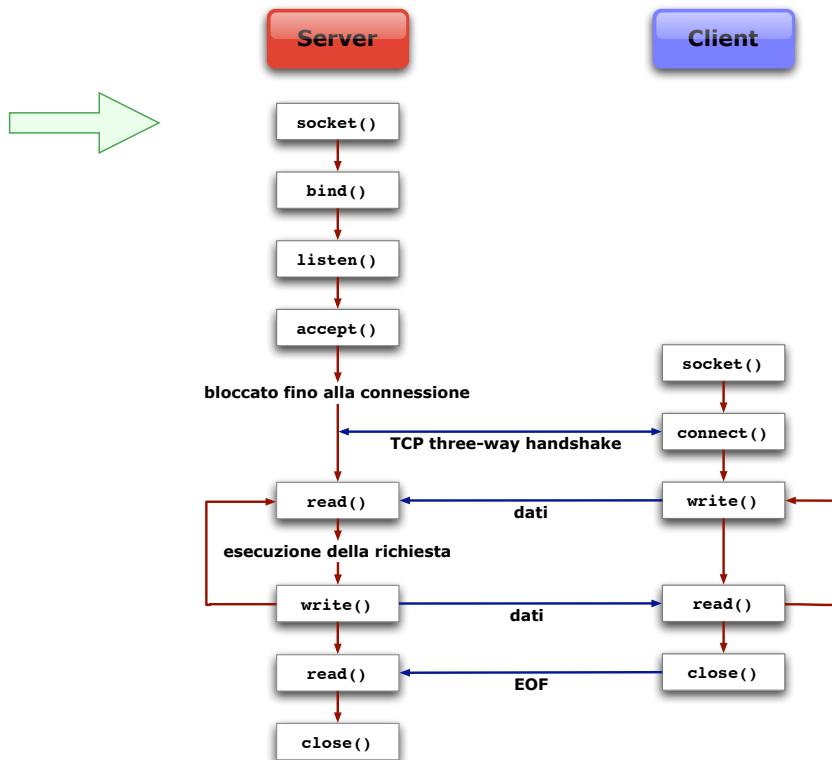
Non usato.

sockaddr_in è definita in <netinet/in.h>

ATTENZIONE: ricordatevi sempre di inizializzare la struttura prima di usarla !

Ad esempio:

```
struct  sockaddr_in  servaddr;  
memset(&servaddr, 0, sizeof(servaddr));  
servaddr.sin_family = AF_INET;
```



Funzioni: socket()

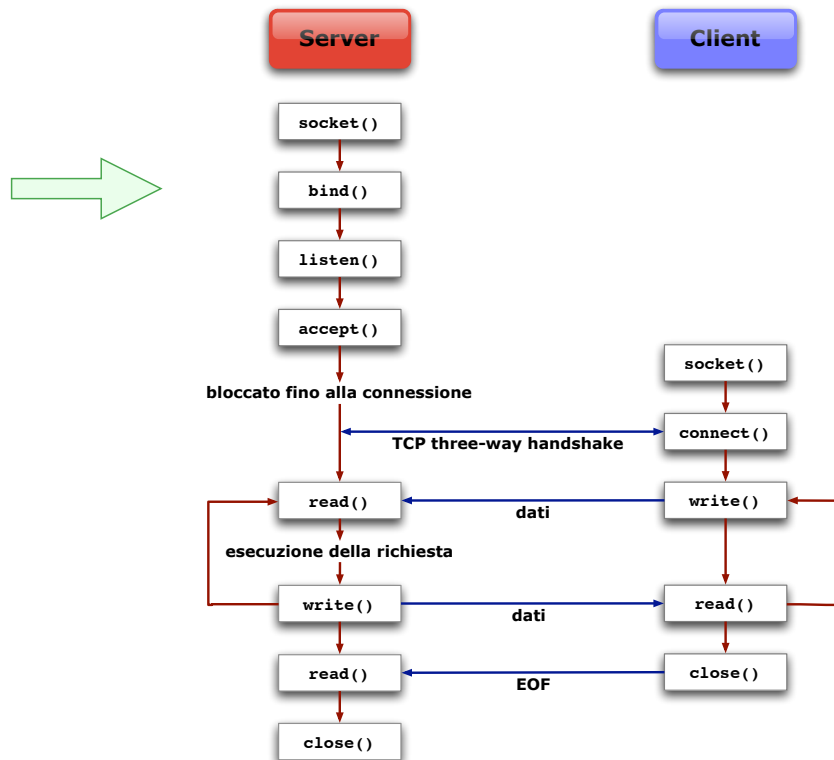
```
int socket(int family,
           int type,
           int protocol);
```

Crea un nuovo socket della *family* e *type* specificati; il campo *protocol* normalmente deve essere imposto pari a 0.

La funzione restituisce un *socket descriptor* (>0) in caso di successo o il valore -1 in caso di errore. Ogni *socket descriptor* identifica univocamente un canale di comunicazione.

<i>family</i>	Descrizione
AF_INET	IPv4
AF_INET6	IPv6
AF_LOCAL	Unix domain protocols
AF_ROUTE	Routing sockets
AF_KEY	Key socket

<i>type</i>	Descrizione
SOCKET_STREAM	Stream socket (TCP)
SOCK_DGRAM	Datagram socket(UDP)
SOCK_RAW	Unix domain protocols



Funzioni: bind()

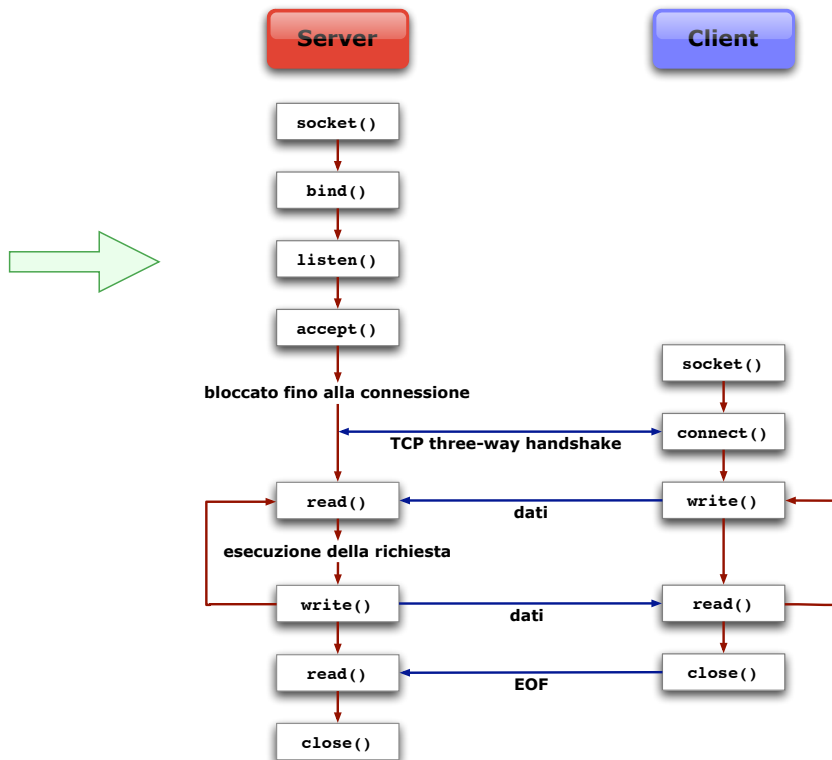
```
int bind(int sockfd,
        const struct sockaddr *myaddr,
        socklen_t addrlen);
```

Assegna la gestione di un indirizzo e di una porta locali ad un socket.

Nella struttura di tipo *sockaddr* possono essere specificati un indirizzo IP, o una porta (TCP o UDP), o entrambi, o nessuno dei due. Quando un valore non viene specificato esso viene scelto automaticamente dal SO.

Normalmente la funzione *bind()* viene usata solo quando si intende realizzare un server in quanto è necessario specificare su quale indirizzo IP e su quale porta il server rimarrà in attesa di connessioni da parte dei client.

La funzione restituisce 0 in caso di successo o -1 in caso di errore.



Funzioni: listen()

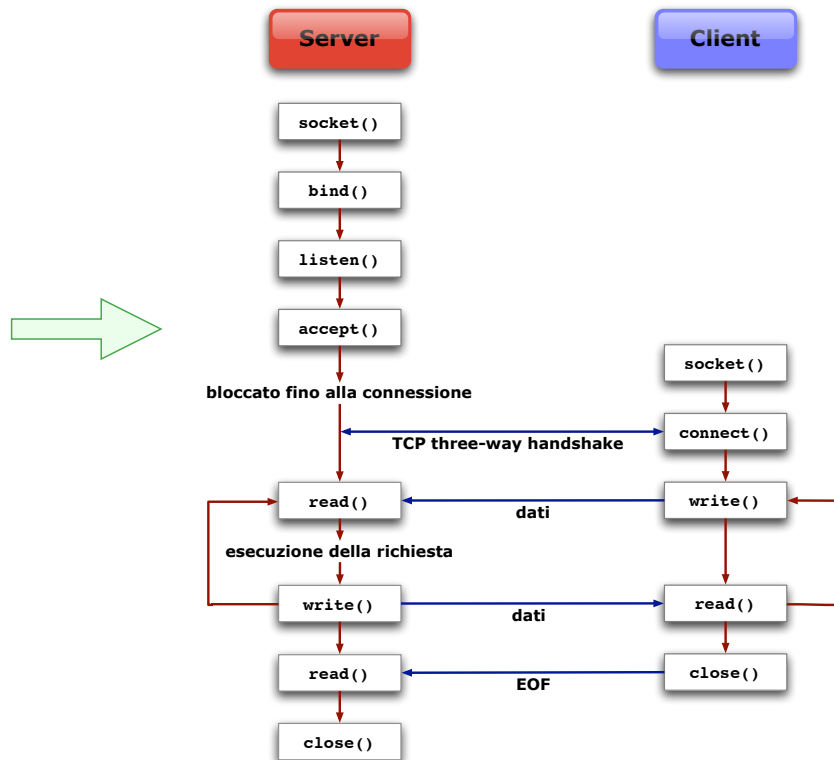
```
int listen(int sockfd,
          int backlog);
```

Rende il socket "passivo", ossia in grado di accettare connessioni in ingresso.

Questa funzione deve essere usata solo all'interno di un server.

Il parametro backlog stabilisce il numero massimo di connessioni in attesa di completamento, contemporaneamente gestibili dallo stesso socket.

La funzione restituisce 0 in caso di successo o -1 in caso di errore.



Funzioni: accept()

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

Restituisce un nuovo socket alla connessione di un client.

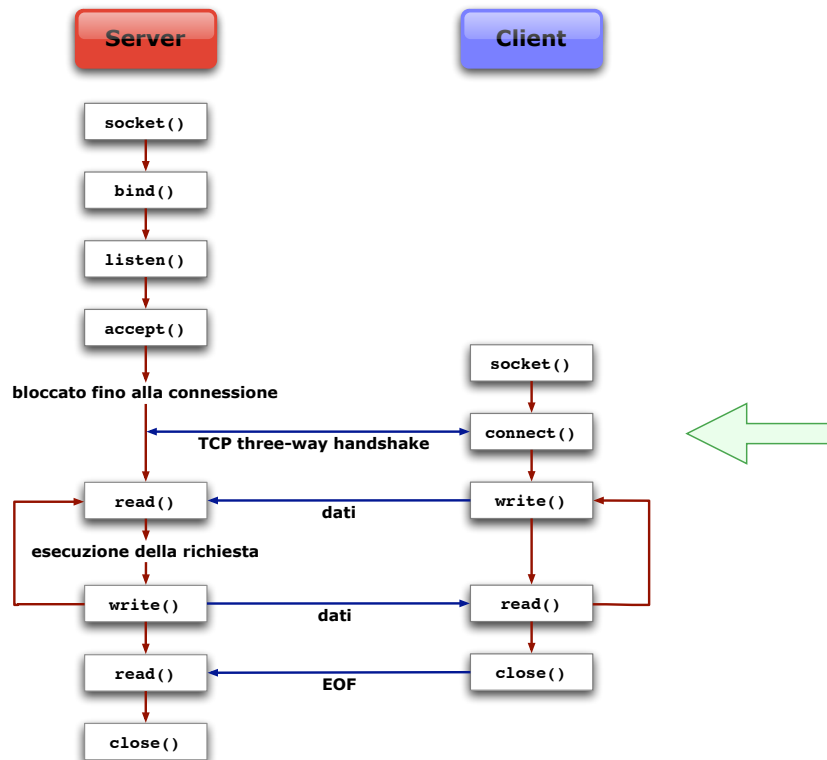
Se al momento dell'esecuzione della istruzione, nessun client si è collegato, la funzione mette in attesa il processo.

In caso di errore viene restituito il valore -1.

ATTENZIONE a non confondere i socket descriptor !

sockfd identifica il socket che era stato messo in modalità passiva tramite la chiamata a *listen()*. Questo è il socket che accetta le connessioni dai client.

Quando un client si connette viene creato un nuovo *socket descriptor*, restituito da *accept()*, che identifica il canale di comunicazione verso il client. Sarà questo il *sd* che dovremo usare per comunicare con il client.



Funzioni: connect()

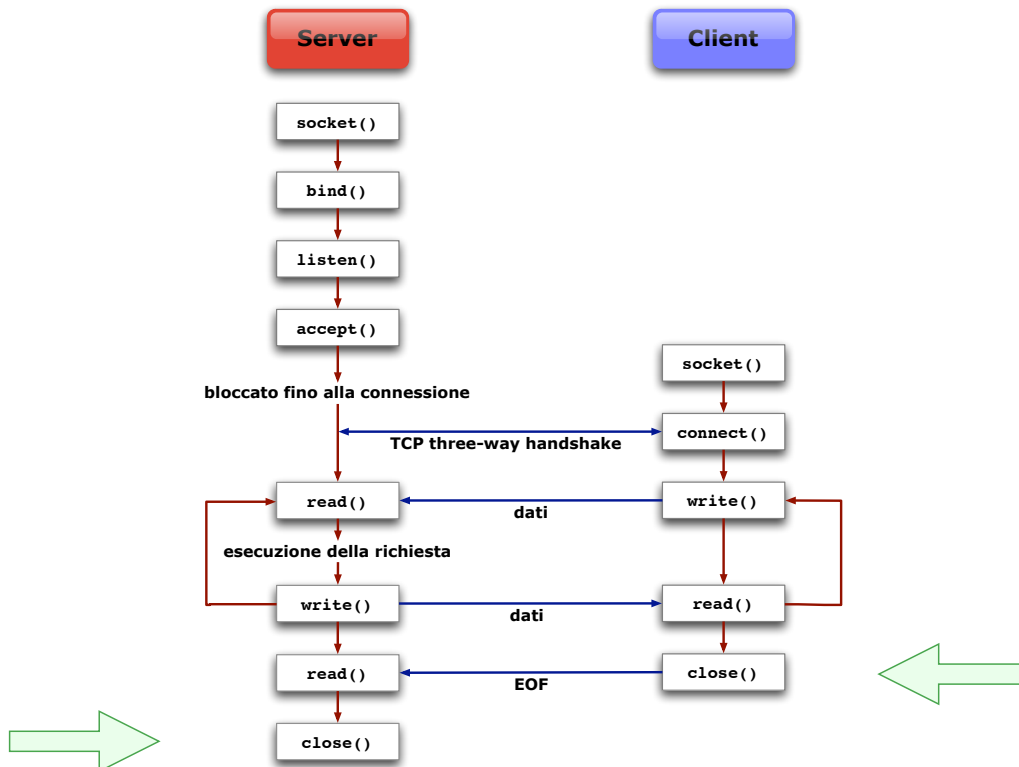
```
int connect(int sockfd,
            const struct sockaddr *addr,
            socklen_t addrlen);
```

Aprire una connessione TCP verso l'host indicato nella struttura *sockaddr*.

È usato solo nei client.

La struttura *sockaddr* deve necessariamente contenere l'indirizzo IP e la porta del server a cui ci si vuole connettere.

Non è necessario eseguire una chiamata a *bind()* prima di *connect()*; il sistema operativo sceglierà automaticamente quale indirizzo IP e quale porta locali utilizzare per il collegamento.



Funzioni: `close()`

```
int close(int sockfd);
```

Chiude il socket.

La funzione restituisce 0 in caso di successo o -1 in caso di errore.

Risoluzione nomi di dominio

Gli utenti sono abituati ad usare i nomi di dominio piuttosto che gli indirizzi IP numerici. Il sistema operativo mette a disposizione del programmatore una primitiva che, tramite l'uso del DNS, restituisce l'indirizzo IP associato ad un nome di dominio.

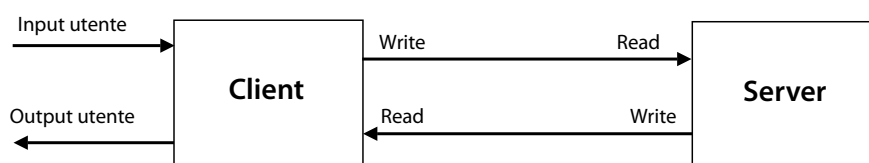
```
struct hostent *gethostbyname(const char *hostname);
```

La funzione interroga il DNS fornendo il nome passato in *hostname*. I dati sull'host restituiti dal DNS vengono inseriti in una struttura di tipo *hostent*:

```
struct hostent {  
    char    *h_name;           // official name of host  
    char    **h_aliases;      // alias list  
    int     h_addrtype;       // host address type  
    int     h_length;         // length of address  
    char    **h_addr_list;    // list of addresses from name server  
};
```

Echo server

1. Il client legge una stringa dallo standard input;
2. il client invia la stringa letta al server;
3. il server riceve la stringa inviata dal client;
4. il server invia la stringa ricevuta al client;
5. il client riceve la stringa inviata dal server;
6. il client scrive la stringa ricevuta sullo standard output.



TCP server single thread (1)

```
/*
    TCP Server single thread
*/

/* dichiarazioni ed include */

int main(int argc, char *argv[]) {

    /* definizione variabili */

    /* Interpretazione linea di comando */
    ParseCmdLine(argc, argv, &endptr);

    /* creazione socket */

    /* bind e listen sul socket */

    /* loop infinito */

    /* Chiusura del socket in ascolto */
    if ( close(list_s) < 0 ) {
        fprintf(stderr, "server: errore durante la close.\n");
        exit(EXIT_FAILURE);
    }
}
```

TCP server single thread (2)

```
/* dichiarazioni ed include */

#include <sys/types.h>          /* socket types          */
#include <sys/socket.h>        /* socket definitions    */
#include <arpa/inet.h>         /* inet (3) funtions    */
#include <unistd.h>           /* misc. UNIX functions */
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <signal.h>

/* Costanti global */
#define MAX_LINE                (1000)

/* Prototipi funzioni */
int ParseCmdLine(int argc, char *argv[], char **szPort);
ssize_t Readline(int fd, void *vptr, size_t maxlen);
ssize_t Writeline(int fc, const void *vptr, size_t maxlen);
```

TCP server single thread (3)

```
/* definizione variabili */

int     list_s;           /* listening socket descriptor */
int     conn_s;          /* connection socket descriptor */
short int port;          /* port number */
struct  sockaddr_in servaddr; /* socket address structure */
struct  sockaddr_in their_addr;
char    buffer[MAX_LINE]; /* character buffer */
char    *endptr;         /* for strtol() */
int     sin_size;
```

TCP server - ParseCmdLine()

```
/* Interpreta i parametri forniti da linea di comando */

int ParseCmdLine(int argc, char *argv[], char **szPort) {
    int n = 1;
    *szPort = 0;

    while ( n < argc ) {
        if ( !strcmp(argv[n], "-p", 2) || !strcmp(argv[n], "-P", 2) )
            *szPort = argv[++n];
        else if ( !strcmp(argv[n], "-h", 2) || !strcmp(argv[n], "-H", 2) ) {
            printf("Sintassi:\n\n");
            printf("    server -p (porta) [-h]\n\n");
            exit(EXIT_SUCCESS);
        }
        ++n;
    }

    if (*szPort==0) {
        printf("Sintassi:\n\n");
        printf("    server -p (porta) [-h]\n\n");
        exit(EXIT_SUCCESS);
    }
    return 0;
}
```

TCP server single thread (4)

```
/* creazione socket */

port = strtol(endptr, &endptr, 0);
if ( *endptr ) {
    fprintf(stderr, "server: porta non riconosciuta.\n");
    exit(EXIT_FAILURE);
}

printf("Server in ascolto sulla porta %d\n",port);

/* Create the listening socket */
if ( (list_s = socket(AF_INET, SOCK_STREAM, 0)) < 0 ) {
    fprintf(stderr, "server: errore nella creazione della socket.\n");
    exit(EXIT_FAILURE);
}
```

TCP server single thread (5)

```
/* bind e listen sul socket */

/* Vuotiamo la struttura servaddr e riempiamo i campi necessari */
memset(&servaddr, 0, sizeof(servaddr));
servaddr.sin_family = AF_INET;
servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
servaddr.sin_port = htons(port);

/* Effettua la bind sull'indirizzo e porta ora specificati */
if ( bind(list_s, (struct sockaddr *) &servaddr, sizeof(servaddr)) < 0 ) {
    fprintf(stderr, "server: errore durante la bind.\n");
    exit(EXIT_FAILURE);
}

/* Mette il socket in modalità di "ascolto" tramite la listen() */
if ( listen(list_s, LISTENQ) < 0 ) {
    fprintf(stderr, "server: errore durante la listen.\n");
    exit(EXIT_FAILURE);
}
```

TCP server single thread (6)

```
/* loop infinito */
while ( 1 ) {
    /* Attende una nuova connessione tramite la accept() */
    sin_size = sizeof(struct sockaddr_in);
    if ( (conn_s = accept(list_s, (struct sockaddr *)&their_addr, &sin_size) ) < 0 ){
        fprintf(stderr, "server: errore nella accept.\n");
        exit(EXIT_FAILURE);
    }

    printf("server: connessione da %s\n", inet_ntoa(their_addr.sin_addr));

    /* Legge una riga dal socket e semplicemente la riscrive nel socket stesso */
    Readline(conn_s, buffer, MAX_LINE-1);

    printf("server: il client ha scritto\n\t%s\n",buffer);

    Writeline(conn_s, buffer, strlen(buffer));

    /* Chiude la connessione */
    if ( close(conn_s) < 0 ) {
        fprintf(stderr, "server: errore durante la close.\n");
        exit(EXIT_FAILURE);
    }
}
}
```

TCP client (1)

```
/*
    TCP Client
*/

/* dichiarazioni ed include */
int main(int argc, char *argv[]) {

    /* definizione variabili */

    /* Interpretazione linea di comando */
    ParseCmdLine(argc, argv, &szAddress, &szPort);

    /* creazione socket */

    /* connessione al server */

    /* invio dati e chiusura socket*/

    return EXIT_SUCCESS;
}
}
```

TCP client (2)

```
/* dichiarazioni ed include */

#include <sys/types.h>          /* socket types          */
#include <sys/socket.h>        /* socket definitions    */
#include <arpa/inet.h>         /* inet (3) funtions    */
#include <unistd.h>            /* misc. UNIX functions */
#include <netinet/in.h>
#include <netdb.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>

/* Costatnti globali */
#define MAX_LINE (1000)

/* Prototipi funzioni */
int ParseCmdLine(int argc, char *argv[], char **szAddress, char **szPort);
ssize_t Readline(int fd, void *vptr, size_t maxlen);
ssize_t Writeline(int fc, const void *vptr, size_t maxlen);
```

TCP client (3)

```
/* definizione variabili */

int      conn_s;                /* connection socket     */
short int port;                 /* port number           */
struct   sockaddr_in servaddr;  /* socket address structure */
char     buffer[MAX_LINE];      /* character buffer       */
char     *szAddress;            /* Holds remote IP address */
char     *szPort;               /* Holds remote port     */
char     *endptr;               /* for strtol()          */
struct   hostent *he;

he=NULL;
```

TCP client - ParseCmdLine()

```
/* Interpreta i parametri forniti da linea di comando */

int ParseCmdLine(int argc, char *argv[], char **szAddress, char **szPort) {
    int n = 1;

    while ( n < argc ) {
        if ( !strcmp(argv[n], "-a", 2) || !strcmp(argv[n], "-A", 2) ) {
            *szAddress = argv[++n];
        } else if ( !strcmp(argv[n], "-p", 2) || !strcmp(argv[n], "-P", 2) ) {
            *szPort = argv[++n];
        } else if ( !strcmp(argv[n], "-h", 2) || !strcmp(argv[n], "-H", 2) ) {
            printf("Sintassi:\n\n");
            printf("    client -a (indirizzo server) -p (porta del server) [-h].\n\n");
            exit(EXIT_SUCCESS);
        }
        ++n;
    }
    if (argc==1) {
        printf("Sintassi:\n\n");
        printf("    client -a (indirizzo server) -p (porta del server) [-h].\n\n");
        exit(EXIT_SUCCESS);
    }
    return 0;
}
```

TCP client (4)

```
/* creazione socket */

/* Riconoscimento porta remota */
port = strtol(szPort, &endptr, 0);
if ( *endptr ) {
    printf("client: porta non riconosciuta.\n");
    exit(EXIT_FAILURE);
}

/* Creazione del socket */
if ( (conn_s = socket(AF_INET, SOCK_STREAM, 0)) < 0 ) {
    fprintf(stderr, "client: errore durante la creazione del socket.\n");
    exit(EXIT_FAILURE);
}

/* Vuotiamo la struttura servaddr e riempiamo i campi necessari */
memset(&servaddr, 0, sizeof(servaddr));
servaddr.sin_family = AF_INET;
servaddr.sin_port = htons(port);
```

TCP client (5)

```
/* connessione al server */

/* Impostazione dell'indirizzo IP del server remoto */
if ( inet_aton(szAddress, &servaddr.sin_addr) <= 0 ) {
    printf("client: indirizzo IP non valido.\nclient: risoluzione nome...");
    if ((he=gethostbyname(szAddress)) == NULL) {
        printf("fallita.\n");
        exit(EXIT_FAILURE);
    }
    printf("riuscita.\n\n");
    servaddr.sin_addr = *((struct in_addr *)he->h_addr);
}

/* connect() verso il server remoto */
if ( connect(conn_s, (struct sockaddr *) &servaddr, sizeof(servaddr)) < 0 ) {
    printf("client: errore durante la connect.\n");
    exit(EXIT_FAILURE);
}
```

TCP client (6)

```
/* invio dati e chiusura socket*/

/* Chiediamo all'utente di inserire una stringa */
printf("Inserire la stringa da spedire: ");
fgets(buffer, MAX_LINE, stdin);

/* Inviemo la stringa al server */
Writeline(conn_s, buffer, strlen(buffer));

/* Vuotiamo il buffer */
memset(buffer, 0, sizeof(char)*(strlen(buffer)+1));

/* Leggiamo la risposta inviata dal server */
Readline(conn_s, buffer, MAX_LINE-1);

/* Stampiamo la risposta ricevuta */
printf("Risposta del server: %s\n", buffer);
```

TCP client - Writeline()

```
/* Scrive una linea di testo su un socket */

ssize_t Writeline(int sockd, const void *vptr, size_t n) {
    size_t    nleft;
    ssize_t    nwritten;
    const char *buffer;
    buffer = vptr;
    nleft = n;

    while ( nleft > 0 ) {
        if ( (nwritten = write(sockd, buffer, nleft)) <= 0 ) {
            if ( errno == EINTR ) nwritten = 0;
            else return -1;
        }
        nleft -= nwritten;
        buffer += nwritten;
    }
    return n;
}
```

TCP client - Readline()

```
/* Legge una linea di testo da un socket */

ssize_t Readline(int sockd, void *vptr, size_t maxlen) {
    ssize_t n, rc;
    char    c, *buffer;
    buffer = vptr;

    for ( n = 1; n < maxlen; n++ ) {
        rc = read(sockd, &c, 1);
        if ( rc == 1 ) {
            *buffer++ = c;
            if ( c == '\n' ) break;
        } else if ( rc == 0 ) {
            if ( n == 1 ) return 0;
            else break;
        } else {
            /* Se la lettura è stata interrotta perche' il device è lento riproviamo */
            if ( errno == EINTR ) continue;
            return -1;
        }
    }
    *buffer = 0;
    return n;
}
```

UDP server (1)

```
/*  
    UDP Server  
*/  
  
/* dichiarazioni ed include */  
  
int main(int argc, char *argv[]) {  
  
    /* creazione socket e bind della porta 7000 */  
  
    /* loop infinito per la gestione delle connessioni */  
  
    return 0;  
}
```

UDP server (2)

```
/* dichiarazioni ed include */  
  
#include <sys/types.h>  
#include <sys/socket.h>  
#include <netinet/in.h>  
#include <arpa/inet.h>  
#include <netdb.h>  
#include <stdio.h>  
#include <unistd.h> /* close() */  
#include <string.h> /* memset() */  
  
#define LOCAL_SERVER_PORT 7000  
#define MAX_MSG 100
```

UDP server (3)

```
/* creazione socket e bind della porta 7000 */

int sd, rc, n, cliLen;
struct sockaddr_in cliAddr, servAddr;
char msg[MAX_MSG];

/* apertura socket */
sd=socket(AF_INET, SOCK_DGRAM, 0);
if(sd<0) {
    printf("%s: errore nell'apertura dell socket.\n",argv[0]);
    exit(1);
}

/* bind della porta 7000 */
servAddr.sin_family = AF_INET;
servAddr.sin_addr.s_addr = htonl(INADDR_ANY);
servAddr.sin_port = htons(LOCAL_SERVER_PORT);

rc = bind (sd, (struct sockaddr *) &servAddr,sizeof(servAddr));
if(rc<0) {
    printf("%s: errore nella bind %d \n", argv[0], LOCAL_SERVER_PORT);
    exit(1);
}
```

UDP server (4)

```
/* loop infinito per la gestione delle connessioni */

while(1) {
    /* inizializzazione buffer per la ricezione*/
    memset(msg, 0x0, MAX_MSG);

    /* ricezione messaggi */
    cliLen = sizeof(cliAddr);
    n = recvfrom(sd,
                msg,
                MAX_MSG,
                0,
                (struct sockaddr *) &cliAddr,
                &cliLen);

    if (n<0) {
        printf("%s: impossibile ricevere dati \n",argv[0]);
        continue;
    }

    /* stampa il messaggio ricevuto */
    printf("%s: dati da %s:UDP%u : %s \n", argv[0],
           inet_ntoa(cliAddr.sin_addr),
           ntohs(cliAddr.sin_port),
           msg);
}
```

UDP client (1)

```
/*  
    UDP Client  
*/  
  
/* dichiarazioni ed include */  
  
int main(int argc, char *argv[]) {  
  
    /* definizione variabili ed interpretazione linea di comando */  
  
    /* creazione socket e bind di una porta */  
  
    /* invio dati e chiusura socket*/  
  
    return 1;  
}
```

UDP client (2)

```
/* dichiarazioni ed include */  
  
#include <sys/types.h>  
#include <sys/socket.h>  
#include <netinet/in.h>  
#include <arpa/inet.h>  
#include <netdb.h>  
#include <stdio.h>  
#include <unistd.h>  
#include <string.h>  
#include <sys/time.h>  
  
#define REMOTE_SERVER_PORT 7000  
#define MAX_MSG 100
```

UDP client (3)

```
/* definizione variabili ed interpretazione linea di comando */

int sd, rc, i;
struct sockaddr_in cliAddr, remoteServAddr;
struct hostent *h;

/* controllo dei parametri inseriti via riga di comando */
if (argc<3) {
    printf("usage : %s <server> <data1> ... <dataN> \n", argv[0]);
    exit(1);
}

/* estrazione dell'indirizzo IP dalla riga di comando */
h = gethostbyname(argv[1]);
if (h==NULL) {
    printf("%s: host sconosciuto '%s' \n", argv[0], argv[1]);
    exit(1);
}

printf("%s: trasmissione dati a '%s' (IP : %s) \n", argv[0], h->h_name,
        inet_ntoa(*(struct in_addr *)h->h_addr_list[0]));

/* creazione struttura dati sockaddr_in relativa al server remoto */
remoteServAddr.sin_family = h->h_addrtype;
memcpy((char *) &remoteServAddr.sin_addr.s_addr, h->h_addr_list[0], h->h_length);
remoteServAddr.sin_port = htons(REMOTE_SERVER_PORT);
```

UDP client (4)

```
/* creazione socket e bind di una porta */

/* creazione socket */
sd = socket(AF_INET,SOCK_DGRAM,0);
if (sd<0) {
    printf("%s: impossibile aprire la socket \n",argv[0]);
    exit(1);
}

/* il bind può essere effettuato su una porta qualsiasi */
cliAddr.sin_family = AF_INET;
cliAddr.sin_addr.s_addr = htonl(INADDR_ANY);
cliAddr.sin_port = htons(0);

rc = bind(sd, (struct sockaddr *) &cliAddr, sizeof(cliAddr));
if(rc<0) {
    printf("%s: impossibile aprire la porta\n", argv[0]);
    exit(1);
}
```

UDP client (5)

```
/* invio dati e chiusura socket */  
  
for(i=2;i<argc;i++) {  
    rc = sendto(sd, argv[i], strlen(argv[i])+1, 0,  
               (struct sockaddr *) &remoteServAddr, sizeof(remoteServAddr));  
  
    if(rc<0) {  
        printf("%s: impossibile trasmettere dati %d \n",argv[0],i-1);  
        close(sd);  
        exit(1);  
    }  
}
```

Riferimenti

UNIX Network Programming (vol. 1)

W. Richard Stevens

Prentice Hall

Appendice

In appendice è riportato il codice del server TCP single thread per sistemi operativi Windows.

TCP server Win: differenze (1)

```
/*
    TCP Server Win single thread
*/

/* dichiarazioni ed include */

int main(int argc, char *argv[]) {

    /* definizione variabili */

    /* Interpretazione linea di comando */
    ParseCmdLine(argc, argv, &endptr);

    /* creazione socket */
    /* bind e listen sul socket */
    /* loop infinito */

    /* Chiusura del socket in ascolto */
    if ( close(list_s) == SOCKET_ERROR ) {
        fprintf(stderr, "server: errore durante la close.\n");
        exit(EXIT_FAILURE);
    }

    WSACleanup();
}
}
```

TCP server Win: differenze (2)

```
/* dichiarazioni ed include */

#include <stdio.h>
#include <winsock.h>
#include <stdlib.h>

/* Costanti global */
#define MAX_LINE          (1000)

/* Prototipi funzioni */
int ParseCmdLine(int argc, char *argv[], char **szPort);
SSIZE_T Readline(int fd, void *vptr, SIZE_T maxlen);
SSIZE_T Writeline(int fc, const void *vptr, SIZE_T maxlen);
```

TCP server Win: differenze (3)

```
/* definizione variabili */

SOCKET list_s;          /* listening socket */
SOCKET conn_s;         /* connection socket */
short int port;        /* port number */
struct sockaddr_in servaddr; /* socket address structure */
struct sockaddr_in their_addr;
char buffer[MAX_LINE]; /* character buffer */
char *endptr;          /* for strtol() */
int sin_size;
WSADATA wsaData;
```

TCP server Win: differenze (4)

```
/* creazione socket */

port = strtol(endptr, &endptr, 0);
if ( *endptr ) {
    fprintf(stderr, "server: porta non riconosciuta.\n");
    exit(EXIT_FAILURE);
}

printf("Server in ascolto sulla porta %d\n",port);

if (WSAStartup(MAKEWORD(1,1), &wsaData) != 0) {
    printf("errore in WSAStartup()\n");
    exit(EXIT_FAILURE);
}

/* Create the listening socket */
if ( (list_s = socket(AF_INET, SOCK_STREAM, 0)) == INVALID_SOCKET ) {
    fprintf(stderr, "server: errore nella creazione della socket.\n");
    exit(EXIT_FAILURE);
}
```

TCP server Win: differenze (5)

```
/* bind e listen sul socket */

/* Vuotiamo la struttura servaddr e riempiamo i campi necessari */
memset(&servaddr, 0, sizeof(servaddr));
servaddr.sin_family = AF_INET;
servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
servaddr.sin_port = htons(port);

/* Effettua la bind sull'indirizzo e porta ora specificati */
if ( bind(list_s, (struct sockaddr *) &servaddr, sizeof(servaddr)) == SOCKET_ERROR ) {
    fprintf(stderr, "server: errore durante la bind.\n");
    exit(EXIT_FAILURE);
}

/* Mette il socket in modalità di "ascolto" tramite la listen() */
if ( listen(list_s, LISTENQ) == SOCKET_ERROR ) {
    fprintf(stderr, "server: errore durante la listen.\n");
    exit(EXIT_FAILURE);
}
```

TCP server Win: differenze (6)

```

/* loop infinito */

while ( 1 ) {
    /* Attende una nuova connessione tramite la accept() */
    sin_size = sizeof(struct sockaddr_in);
    if ( (conn_s = accept(list_s, (struct sockaddr *)&their_addr, &sin_size) ) ==
                                                INVALID_SOCKET ){
        fprintf(stderr, "server: errore nella accept.\n");
        exit(EXIT_FAILURE);
    }

    printf("server: connessione da %s\n", inet_ntoa(their_addr.sin_addr));

    /* Legge una riga dal socket e semplicemente la riscrive nel socket stesso */
    Readline(conn_s, buffer, MAX_LINE-1);

    printf("server: il client ha scritto\n\t%s\n",buffer);

    Writeline(conn_s, buffer, strlen(buffer));

    /* Chiude la connessione */
    if ( close(conn_s) == SOCKET_ERROR ) {
        fprintf(stderr, "server: errore durante la close.\n");
        exit(EXIT_FAILURE);
    }
}

```

TCP client Win: differenze(1)

```

/* dichiarazioni ed include */
#include <stdio.h>
#include <winsock.h>
#include <stdlib.h>
/* Costatnti globali */
#define MAX_LINE (1000)
/* Prototipi funzioni */
int ParseCmdLine(int argc, char *argv[], char **szAddress, char **szPort);
SSIZE_T Readline(int fd, void *vptr, SIZE_T maxlen);
SSIZE_T Writeline(int fc, const void *vptr, SIZE_T maxlen);

/* definizione variabili */
int conn_s; /* connection socket */
short int port; /* port number */
struct sockaddr_in servaddr; /* socket address structure */
char buffer[MAX_LINE]; /* character buffer */
char *szAddress; /* Holds remote IP address */
char *szPort; /* Holds remote port */
char *endptr; /* for strtol() */
struct hostent *he;

u_long nRemoteAddr;
WSADATA wsaData;

he=NULL;

```

TCP client Win: differenze(2)

```
/* creazione socket */

/* Riconoscimento porta remota */
port = strtol(szPort, &endptr, 0);
if ( *endptr ) {
    printf("client: porta non riconosciuta.\n");
    exit(EXIT_FAILURE);
}

if (WSAStartup(MAKEWORD(1,1), &wsaData) != 0) {
    printf("errore in WSAStartup()\n");
    exit(EXIT_FAILURE);
}

/* Creazione del socket */
if ( (conn_s = socket(AF_INET, SOCK_STREAM, 0)) < 0 ) {
    fprintf(stderr, "client: errore durante la creazione del socket.\n");
    exit(EXIT_FAILURE);
}

/* Vuotiamo la struttura servaddr e riempiamo i campi necessari */
memset(&servaddr, 0, sizeof(servaddr));
servaddr.sin_family = AF_INET;
servaddr.sin_port = htons(port);
```

TCP client Win: differenze(3)

```
/* connessione al server */

/* Impostazione dell'indirizzo IP del server remoto */

nRemoteAddr = inet_addr(szAddress);
if (nRemoteAddr == INADDR_NONE) {
    printf("client: indirizzo IP non valido.\nclient: risoluzione nome...");
    if ((he=gethostbyname(szAddress)) == 0) {
        printf("fallita.\n");
        exit(EXIT_FAILURE);
    }
    printf("riuscita.\n\n");
    nRemoteAddr = *((u_long *)he->h_addr_list[0]);
}
servaddr.sin_addr.s_addr = nRemoteAddr;

/* connect() verso il server remoto */
if (connect(conn_s, (struct sockaddr *) &servaddr, sizeof(servaddr)) == SOCKET_ERROR) {
    printf("client: errore durante la connect.\n");
    exit(EXIT_FAILURE);
}
```

TCP client Win: differenze(4)

```
/* invio dati e chiusura socket*/

/* Chiediamo all'utente di inserire una stringa */
printf("Inserire la stringa da spedire: ");
fgets(buffer, MAX_LINE, stdin);

/* Inviemo la stringa al server */
Writeline(conn_s, buffer, strlen(buffer));

/* Vuotiamo il buffer */
memset(buffer, 0, sizeof(char)*(strlen(buffer)+1));

/* Leggiamo la risposta inviata dal server */
Readline(conn_s, buffer, MAX_LINE-1);

/* Stampiamo la risposta ricevuta */
printf("Risposta del server: %s\n", buffer);

WSACleanup();
```