INTERFACCIA (API) DI SOCKET

File di libreria da includere

```
#include        <sys/types.h>
#include        <sys/socket.h>
#include        <netinet/in.h>
```

Creazione di un socket

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

/* Crea un socket e ne restituisce il descrittore, -1 = errore. Parametri: **family**, definisce la famiglia di protocolli (**AF_INET** per TCP/IP, AF_UP per Xerox, ...); **type**, specifica il tipo di comunicazione (**SOCK_STREAM** per servizio di consegna affidabile TCP, **SOCK_DGRAM** per datagramma senza connessione UDP, ...); **protocol**, specifica quale protocollo utilizzare se nella famiglia utilizzata ne esiste più di uno, normalmente vale **0**. Nota bene: il socket da solo non è ancora il canale di rete, è soltanto una struttura dati che serve per gestire il canale di rete; per aprire un canale di rete associandolo al socket occorre chiamare la primitiva connect oppure la primitiva accept */

Chiusura di un socket (eliminazione del canale di rete associato al socket)

int close (int sd)

/* Cancella un socket e ne rilascia il descrittore, -1 = errore. Il canale di rete associato al socket viene eliminato. Parametri: **sd**, è il socket da cancellare. Il socket cancellato non è più utilizzabile per aprire canali. */

Richiesta di apertura di un canale di rete in qualità di cliente (apertura attiva)

int connect (int sd, struct sockaddr_in * server_ep, int ep_len)

/* Invia una richiesta di collegamento in qualità di cliente, restituisce 0 se successo, -1 se errore. Parametri: **sd**, specifica il socket (che deve essere già stato creato) da associare al canale di rete; **server_ep**, specifica il punto terminale (endpoint) del destinatario della richiesta di collegamento, che è il server; **ep_len**, specifica la lunghezza in byte del punto terminale */

Chiamata tipica: **error = connect(sd, &server_ep, sizeof(server_ep));**

Associazione di un socket a una porta TCP

int bind (int sd, struct sockaddr_in * server_ep, int ep_len)

/* Associa un numero di porta TCP a un socket, restituisce 0 se successo, -1 se errore. Parametri: **sd**, specifica il socket da associare al numero di porta TCP; **server_ep**, specifica il punto terminale (endpoint) contenente il numero di porta da associare (l'indirizzo ha funzione di filtro); **ep_len**, specifica la lunghezza in byte del punto terminale */

Creazione della coda di richieste di connessione pendenti

int listen (int sd, int len)

/* Crea e dimensiona la coda di richieste di connessione pendenti associata al socket, restituisce 0 se successo, -1 se errore. Parametri: **sd**, specifica il socket per cui creare la coda; **len**, specifica la lunghezza max della coda */

Richiesta di apertura di un canale di rete in qualità di servente (apertura passiva)

int accept (int sd, struct sockaddr_in * client_ep, int * ep_len)

/* Accetta una richiesta di collegamento in qualità di servente, restituisce un nuovo socket (sempre >= 0) se successo, -1 se errore; il nuovo socket restituito è quello su cui portare avanti il dialogo con il cliente richiedente; il vecchio socket è disponibile per ulteriori accettazioni. Parametri: **sd**, specifica il socket (che deve essere già stato creato) su cui ricevere la richiesta di

collegamento proveniente dal cliente; **client_ep**, specifica la locazione in cui memorizzare il punto terminale (endpoint) del cliente; **ep_len**, specifica la locazione in cui memorizzare la lunghezza in byte del punto terminale */

Chiamata tipica: **new_sd = accept (sd, &client_ep, &ep_len);**

- **Acquisizione del punto terminale (endpoint) locale**

int getsockname (int sd, struct sockaddr_in * local_ep, int * ep_len)
/* Cerca il punto terminale locale del canale e lo restituisce. Parametri: **sd**, socket associato al canale il cui punto terminale locale si vuole conoscere; **local_ep**, punto terminale locale; **ep_len**, lunghezza del punto terminale */

- **Acquisizione del punto terminale (endpoint) remoto**

int getpeername (int sd, struct sockaddr_in * remote_ep, int * ep_len)
/* Come getsockname, ma acquisisce il punto terminale remoto del canale */

- **Invio dati attraverso un canale**

int send (int sd, char * message, int len, int flags)
/* Spedisce, attraverso il canale identificato da **sd**, **len** byte memorizzati nella stringa **message**. Restituisce il numero di byte effettivamente inviati, -1 se errore. Altri param.: **flags**, specifica funzioni speciali, di solito **0** */

- **Ricezione dati attraverso un canale**

int recv (int sd, char * message, int len, int flags)
/* Riceve, attraverso il canale identificato da **sd**, **len** byte e li memorizza nella stringa **message**. Restituisce il numero di byte effettivamente ricevuti, -1 se errore. Altri param.: **flags**, specifica funzioni speciali, di solito **0** */

- **Funzioni di conversione di formato di indirizzi IP e porte TCP**

unsigned int inet_addr (char * stringa)
/* Converte l'indirizzo IP da stringa di caratteri a formato di rete */
char * inet_ntoa (unsigned int addr)
/* Converte l'indirizzo IP da formato di rete a stringa di caratteri */
unsigned short int htons (unsigned short int port)
/* Converte il numero di port TCP da numero intero a formato di rete */
unsigned short int ntohs (unsigned short int port)
/* Converte il numero di port TCP da formato di rete a numero intero */

- **Dichiarazione di un punto terminale (endpoint)**

```
struct sockaddr_in {
     short int          sin_family;   /* tipo: = AF_INET per TCP/IP      */
     unsigned short int sin_port;     /* porta TCP in formato di rete    */
     struct in_addr     sin_addr;     /* indirizzo IP in formato di rete */
     char               sin_zero[8];  /* riempitivo, non è usato         */
} /* sockaddr_in */
```

- **Come inizializzare la variabile sockaddr_in (punto terminale, o endpoint)**

```
char *  indirizzo_IP[256];
int     porta_TCP;
struct  sockaddr_in ep;

bzero (&ep, sizeof(ep));
ep.sin_family = AF_INET;
ep.sin_port = htons ((unsigned short int) porta_TCP);
ep.sin_addr.s_addr = inet_addr (indirizzo_IP);
```