

Laboratorio reti AA 2008/2009

Dott. Matteo Roffilli

roffilli@csr.unibo.it

**Ricevimento in ufficio
dopo la lezione**

Laboratorio reti AA 2008/2009

Per esercitarvi fate SSH su:

alfa.csr.unibo.it

si-tux00.csr.unibo.it

....

si-tux15.csr.unibo.it

Eventuali variazioni di orario/giorno verranno comunicate in anticipo via mail.

Laboratorio reti AA 2008/2009

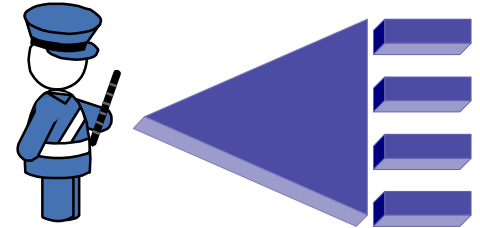
- **Marzo**
- 5 Intro,SSH,VI/VIM,GCC base
- 12 Richiami di C e Compilazione
- 19 Socket e Co.
- **26 Socket e Co. parte seconda**
- **Aprile**
- 2 Client
- Fine prima parte. Si riprende il 23 Aprile.
Controllate sempre il portale di CSR.

Esempio #1

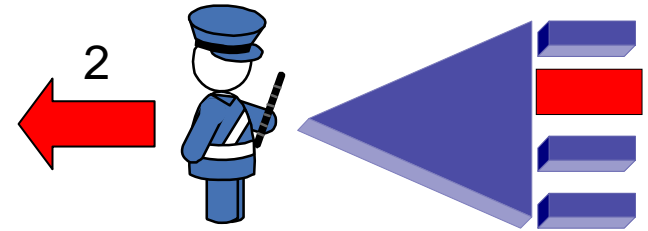
```
void main(void)
{

int sockfd;
struct sockaddr_in Serv;

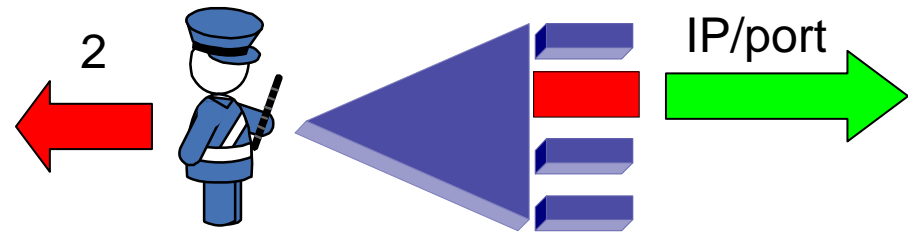
...
```



```
/* prende un socket per stream TCP */
socketfd=socket(PF_INET,SOCK_STREAM,0);
```



```
/* specifica l'indirizzo TCP/IP del server remoto */
Serv.sin_family=AF_INET;
Serv.sin_addr.s_addr=inet_addr(char*_IP);
Serv.sin_port = htons(int_port);
```



```
...
}
```

connect

La funzione **connect** è usata da un client IP per stabilire la connessione con un server IP.

```
#include <sys/socket.h>
```

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

connect

La funzione restituisce:

- **0** in caso di successo
- **-1** per un errore, nel qual caso **errno** assumerà i valori:

ECONNREFUSED non c'è nessuno in ascolto sull'indirizzo remoto.

ETIMEDOUT si è avuto timeout durante il tentativo di connessione.

ENETUNREACH la rete non è raggiungibile.

EINPROGRESS il socket è non bloccante e la connessione non può essere conclusa immediatamente.

EALREADY il socket è non bloccante e un tentativo precedente di connessione non si è ancora concluso.

EAGAIN non ci sono più porte locali libere.

EAFNOSUPPORT l'indirizzo non ha una famiglia di indirizzi corretta nel relativo campo.

EACCES, EPERM si è tentato di eseguire una connessione ad un indirizzo broadcast senza che il socket fosse stato abilitato per il broadcast.

altri errori possibili sono: EFAULT, EBADF, ENOTSOCK, EISCONN e EADDRINUSE.

sockaddr vs sockaddr_in

Le funzioni che utilizzeremo richiedono esplicitamente di tipizzare alcuni dati come strutture:

sockaddr

Nel Laboratorio precedente abbiamo visto invece la definizione della struttura

sockaddr_in

Un puntatore a quest'ultima può essere castato senza problemi ad un puntatore alla prima con **(struct sockaddr*)** in quanto è stata creata appositamente.

Infatti la **sockaddr_in** è derivata dalla **sockaddr** per essere utilizzata nel dominio IPv4.

Ad esempio esistono anche le strutture derivate: **struct sockaddr_in6**, **sockaddr_ipx**, ...

```
struct sockaddr_in
{
    sa_family_t    sin_family; /* address family: AF_INET */
    in_port_t      sin_port;   /* port in network byte order */
    struct in_addr sin_addr;    /* internet address (IP) */
};
```

socklen_t

Specifica la dimensione della struttura passata come argomento.

E' buona norma calcolarlo a run-time in quanto alcuni tipi potrebbero avere dimensioni differenti su architetture diverse.

Per calcolare la dimensione in byte di una struttura si usa l'operatore:

```
size_t sizeof();
```

Esempio:

```
struct sockaddr_in serv;  
sizeof(serv);
```

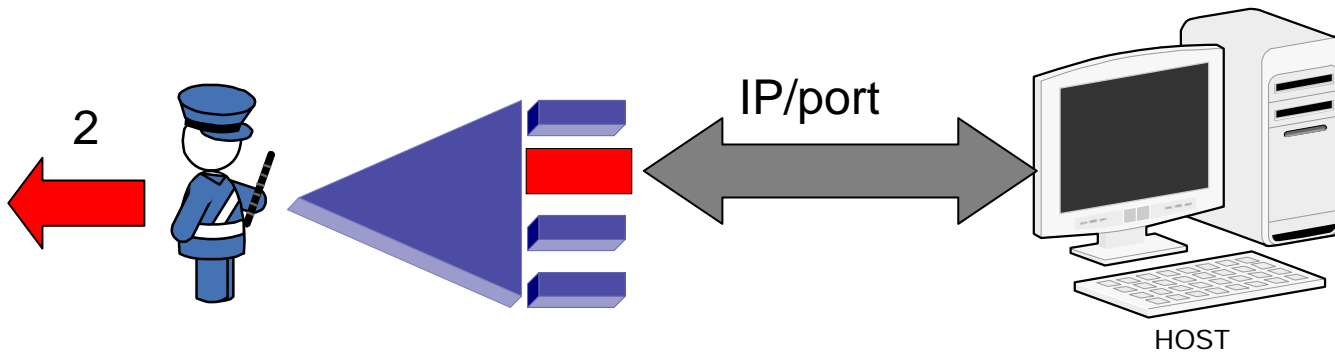

Esempio #2

```
void main(void)
{
    int sockfd;
    struct sockaddr_in Serv;
    int ris;

    /* prende un socket per stream TCP */
    socketfd=socket(PF_INET,SOCK_STREAM,0);

    /* specifica l'indirizzo
    TCP/IP del server remoto */
    Serv.sin_family=AF_INET;
    Serv.sin_addr.s_addr=inet_addr(char*_IP);
    Serv.sin_port = htons(int_port);

    ris = connect(sockfd, (struct sockaddr*) &Serv, sizeof(Serv));
}
```



close

La funzione standard Unix **close** che si usa sui file può essere usata con lo stesso effetto anche sui file descriptor associati ad un socket.

```
#include <unistd.h>
```

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

La funzione restituisce:

- **0** in caso di successo
- **-1** per un errore, nel qual caso bisogna controllare **errno**

L'azione di questa funzione quando applicata a socket è di marcarlo come chiuso e ritornare immediatamente al processo.

Una volta chiamata il socket descriptor non è più utilizzabile dal processo e non può essere usato come argomento per una write o una read (anche se l'altro capo della connessione non avesse chiuso la sua parte).

Il kernel invierà comunque tutti i dati che ha in coda prima di iniziare la sequenza di chiusura.

Esempio #3

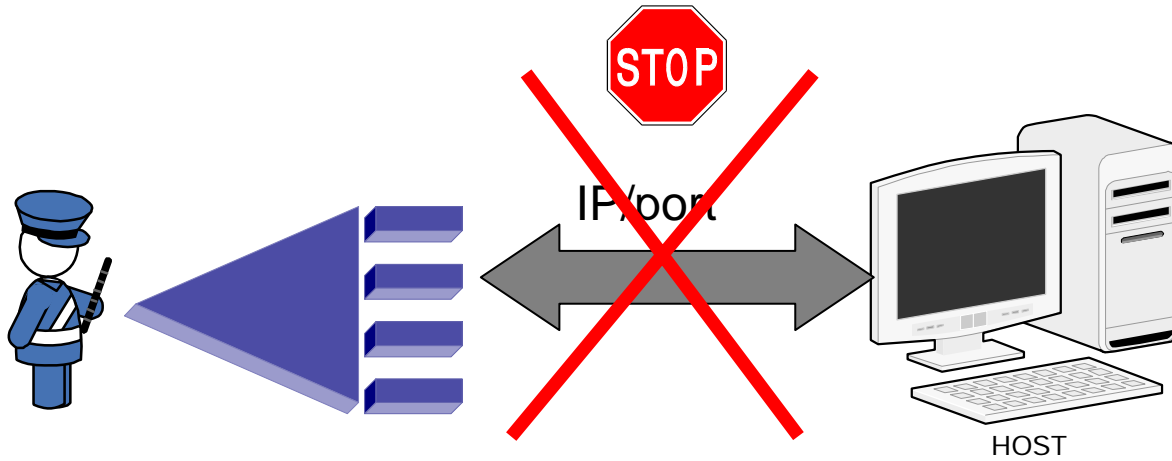
```
void main(void)
{

/* specifica l'indirizzo
TCP/IP del server remoto */
Serv.sin_family=AF_INET;
Serv.sin_addr.s_addr=inet_addr(char*_IP);
Serv.sin_port = htons(int_port);

ris = connect(socketfd, (struct sockaddr*) &Serv, sizeof(Serv));

/* chiusura */
close(socketfd);

}
```



bind

La funzione **bind** assegna un indirizzo locale ad un socket.

È usata cioè per specificare la prima parte della socket pair. Viene usata di solito sul lato server per specificare la IP e porta su cui poi ci si porrà in ascolto.

Il prototipo della funzione è il seguente:

```
#include <sys/socket.h>
```

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

“un indirizzo locale”

I processi comunicano verso l'esterno tramite le socket.

Il protocollo **PF_INET** (IPv4) comunica tramite indirizzi di tipo **IP:porta** (es.137.204.72.5:80).

Per permettere ad un processo di comunicare con un altro tramite il protocollo **PF_INET** bisogna collegare un indirizzo locale (cioè della macchina su cui il processo gira) di tipo **AF_INET** ad una socket aperta dal processo.

La **bind** fa esattamente questo!

Lato server questo linking è necessario mentre lato client è opzionale (se non specificato ci pensa in automatico la **connect**).

bind

La funzione restituisce:

- **0** in caso di successo
- **-1** per un errore, nel qual caso **errno** assumerà i valori:
 - EBADF il file descriptor non è valido.
 - EINVAL il socket ha già un indirizzo assegnato.
 - ENOTSOCK il file descriptor non è associato ad un socket.
 - EACCES si è cercato di usare una porta riservata senza sufficienti privilegi.
 - EADDRNOTAVAIL Il tipo di indirizzo specificato non è disponibile.
 - EADDRINUSE qualche altro socket sta già usando l'indirizzo.
 - ed anche EFAULT e per i socket di tipo AF_UNIX, ENOTDIR, ENOENT, ENOMEM, ELOOP, ENOSR e EROFS.

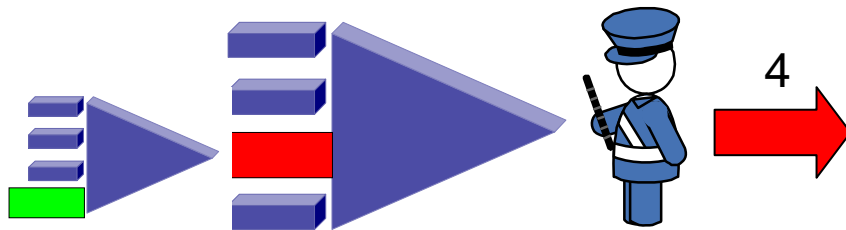
Esempio #4

```
void main(void)
{
    int sockfd,ris;
    struct sockaddr_in Serv,Local;

    /* prende un socket per stream TCP */
    sockfd=socket(PF_INET,SOCK_STREAM,0);

    /* name the socket */
    memset ( &Local, 0, sizeof(Local));
    Local.sin_family=AF_INET;
    Local.sin_addr.s_addr=htonl(INADDR_ANY);
    Local.sin_port=htons(local_port_number);

    ris = bind(sockfd, (struct sockaddr*) &Local, sizeof(Local));
}
```



listen

La funzione **listen** serve ad usare un socket in modalità passiva, cioè, come dice il nome, per metterlo in ascolto di eventuali connessioni; in sostanza l'effetto della funzione è di portare il socket dallo stato CLOSED a quello LISTEN.

In genere si chiama la funzione in un server dopo le chiamate a socket e bind e prima della chiamata ad accept.

```
#include <sys/socket.h>
int listen(int sockfd,
           int backlog)
```

backlog specifica la dimensione del buffer della listen e per i nostri scopi si può mettere fisso a 10.

La funzione restituisce:

- **0** in caso di successo
- **-1** per un errore, nel qual caso **errno** assumerà i valori:
 - EBADF l'argomento sockfd non è un file descriptor valido.
 - ENOTSOCK l'argomento sockfd non è un socket.
 - EOPNOTSUPP il socket è di un tipo che non supporta questa operazione.

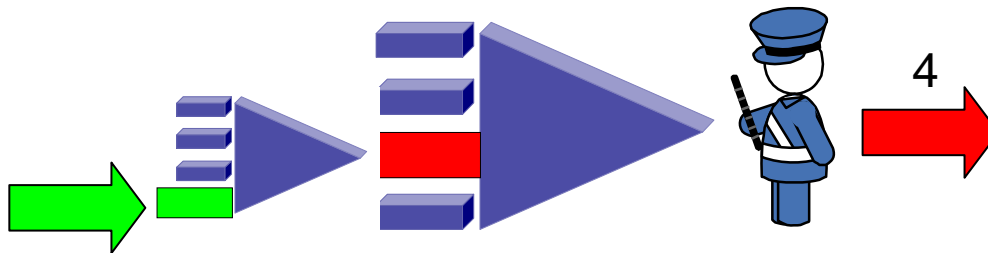
Esempio #5

```
void main(void)
{
    int sockfd,ris;
    struct sockaddr_in Serv,Local;

    /* prende un socket per stream TCP */
    sockfd=socket(PF_INET,SOCK_STREAM,0);

    ris = bind(sockfd, (struct sockaddr*) &Local, sizeof(Local));

    ris = listen(sockfd, 10 );
}
```



accept

La funzione **accept** è chiamata da un server per gestire la connessione una volta che sia stato completato il three way handshake, la funzione restituisce un nuovo socket descriptor su cui si potrà operare per effettuare la comunicazione. Se non ci sono connessioni completate il processo viene messo in attesa.

I due argomenti `addr` e `addrlen` (si noti che quest'ultimo è passato per indirizzo per avere indietro il valore) sono usati per ottenere l'indirizzo del client da cui proviene la connessione. Prima della chiamata `addrlen` deve essere inizializzato alle dimensioni della struttura il cui indirizzo è passato come argomento in `addr`; al ritorno della funzione `addrlen` conterrà il numero di byte scritti dentro `addr`.

```
#include <sys/socket.h>
```

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

accept

La funzione restituisce:

- **>0** in caso di successo (l'identificativo del socket descriptor)
- **-1** per un errore, nel qual caso **errno** assumerà i valori:

- EBADF l'argomento sockfd non è un file descriptor valido.
- ENOTSOCK l'argomento sockfd non è un socket.
- EOPNOTSUPP il socket è di un tipo che non supporta questa operazione.
- EAGAIN o EWOULDBLOCK il socket è stato impostato come non bloccante e non ci sono connessioni in attesa di essere accettate.
- EPERM, ENOBUFS, ENOMEM, EINTR, EMFILE, EINVAL, ENOSR, ENOBUFS, EFAULT, EPERM, ECONNABORTED, ESOCKTNOSUPPORT, EPROTONOSUPPORT, ETIMEDOUT, ERESTARTSYS.

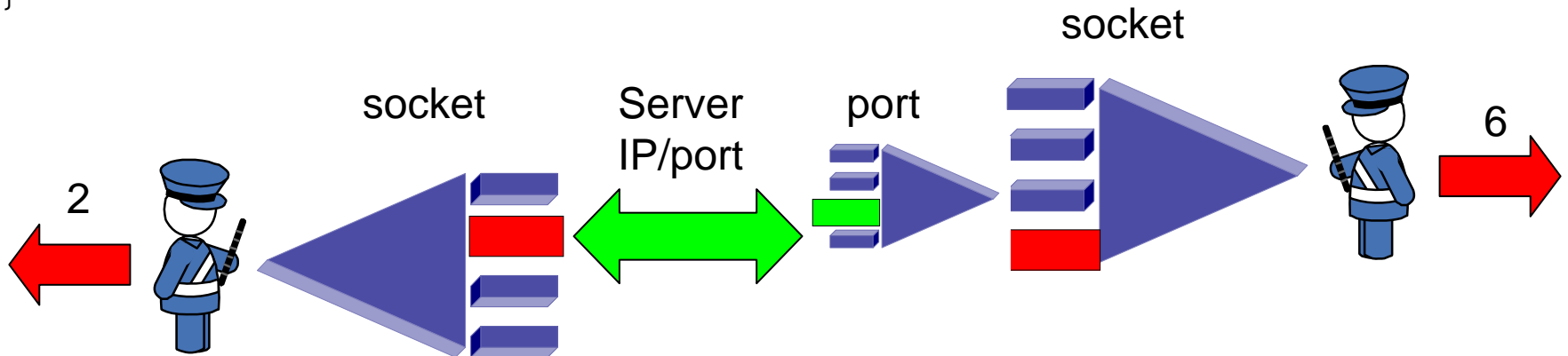
Esempio #6

```
void main(void)
{
    int sockfd,ris,newsocketfd;
    struct sockaddr_in Serv,Local,Client;
    int len;

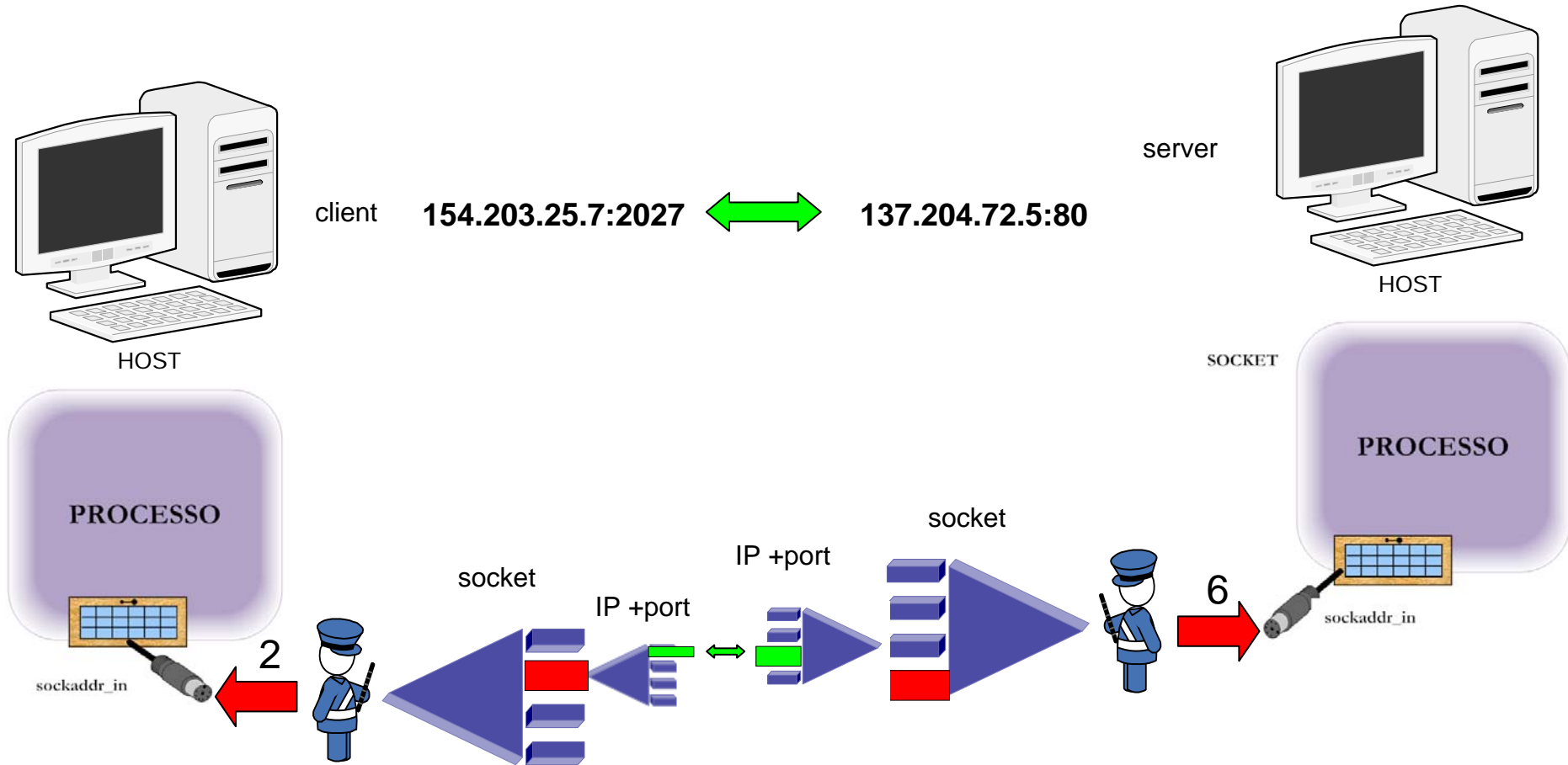
    /* prende un socket per stream TCP */
    sockfd=socket(PF_INET,SOCK_STREAM,0);

    ris = bind(sockfd, (struct sockaddr*) &Local, sizeof(Local));
    ris = listen(sockfd, 10 );

    len=sizeof(Client);
    newsocketfd = accept(sockfd,(struct sockaddr*)&Client, &len);
}
```



Visione generale



Controllo errori

```
int main(int argc, char* argv[])
{
    int sock_fd, i;
    struct sockaddr_in serv_addr;

    ...

    /* create socket */
    if ( (sock_fd = socket(PF_INET, SOCK_STREAM, 0)) < 0)
    {
        perror("Socket creation error");
        return -1;
    }

    ...

}
```

E ora puntatori !

Goal:

realizzare un programma che legge a riga di comando il numero di elementi da inserire, richiede gli elementi (numeri interi di tipo long) e stampa a video la loro somma

Requisiti:

1. Lo spazio per memorizzare i numeri richiesti deve essere allocato dinamicamente.
2. L'allocazione di memoria deve essere controllata.
3. L'accesso ai numeri deve utilizzare l'aritmetica dei puntatori
4. Il programma deve essere compilato, linkato e deve funzionare!!!

Tempo a disposizione:

25 minuti

Soluzione

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

int main(int argc, char* argv[])
{
    long* buffer;
    int num, i, somma=0;

    if(argc!=2) exit(EXIT_FAILURE);

    num=atoi(argv[1]);

    buffer=(long*)malloc(num*sizeof(long));
    if(buffer==NULL) exit(EXIT_FAILURE);

    for(i=0; i<num; i++) *(buffer+i)=0;

    for(i=0; i<num; i++)
    {
        scanf("\n%ld", buffer+i);
        somma+=*(buffer+i);
    }

    free(buffer);
    printf("\nsomma=%d\n", somma);
    return 0;
}
```