

Socket

- Nei sistemi operativi moderni i servizi disponibili in rete si basano principalmente sul modello **client/server**.
- Tale architettura consente ai sistemi di condividere risorse e cooperare per il raggiungimento di un obiettivo.
- Per quanto riguarda la programmazione di sistema, esiste l'interfaccia delle **socket** che fornisce un'astrazione user-friendly dei meccanismi di base per implementare programmi client/server. Una socket ("presa, spinotto") è un'estremità di comunicazione tra processi.
- Una socket in uso è solitamente legata (**bound**) ad un indirizzo. La natura dell'indirizzo dipende dal **dominio di comunicazione** del socket. Processi comunicanti nello stesso dominio usano lo stesso formato di indirizzi.
- Una socket comunica in un solo dominio. I domini implementati sono descritti in `<sys/socket.h>`. I principali sono
 - il dominio UNIX (`AF_UNIX`)
 - il dominio Internet (`AF_INET`, `AF_INET6`)
 - il dominio Novell (`AF_IPX`)
 - il dominio AppleTalk (`AF_APPLETALK`)

Tipi di Socket

Esistono due modi principali per comunicare in rete:

- il **connection oriented model**;
- il **connectionless oriented model**.

In corrispondenza ai due paradigmi di comunicazione precedenti abbiamo i seguenti tipi di socket:

- **Stream socket** forniscono stream di dati affidabili, duplex, ordinati. Nel dominio Internet sono supportati dal protocollo TCP.
- **Socket a datagrammi** trasferiscono messaggi di dimensione variabile, preservando i confini ma senza garantire ordine o arrivo dei pacchetti. Supportate nel dominio Internet dal protocollo UDP.

Il dominio Internet: indirizzi IP

- Un indirizzo IP (Internet Protocol) è costituito da quattro decimali interi (con valori da 0 a 255) separati da punti e consente di individuare univocamente la posizione di una sottorete e di un host all'interno di quest'ultima.
- Per utilizzare gli indirizzi IP in programmi C, bisogna prima convertirli nel tipo apposito `in_addr_t` tramite la seguente system call:

```
#include <arpa/inet.h>
```

```
in_addr_t inet_addr(const char *ip_address);
```

Esempio:

```
in_addr_t server;
```

```
server = inet_addr("158.110.1.7");
```

- Nel file header `<netinet/in.h>` è definita la costante simbolica `INADDR_ANY` che consente di legare una socket alla macchina corrente, senza conoscerne l'indirizzo di rete.

Il dominio Internet: porte

- Oltre a conoscere l'indirizzo IP dell'host a cui connettersi, bisogna disporre dell'informazione sufficiente per collegarsi al processo server corretto.
- Per questo motivo esistono i **numeri di porta** (port number) che permettono di associare un servizio ad un ben determinato numero.
- Quindi le connessioni avvengono sempre specificando un indirizzo IP ed un numero di porta.
- I numeri da 0 a 1023 sono riservati per i servizi standard (e.g., FTP, HTTP ecc.), mentre i numeri da 1024 a 65535 sono disponibili per i processi utente.

Strutture dati per le socket

Per memorizzare l'indirizzo ed il numero di porta nel file header <sys/socket.h> sono definite le seguenti strutture standard:

- per indicare una socket generica:

```
struct sockaddr {
    short sa_family;          /* Address family */
    char  sa_data[];         /* Address data.  */
};
```

- per indicare una socket nel dominio AF_UNIX

```
struct sockaddr_un {
    short sa_family;          /* Flag AF_UNIX */
    char  sun_path[108];     /* Path name  */
};
```

- Per indicare una socket nel dominio AF_INET

```
struct sockaddr_in {
    short  sa_family;        /* Flag AF_INET */
    short  sin_port;        /* Numero di porta */
    struct in_addr sin_addr; /* indir. IP */
    char   sin_zero[8];     /* riempimento */
};
```

dove in_addr rappresenta un indirizzo IP

```
struct in_addr { u_long s_addr; /* 4 byte */ };
```

Chiamate di sistema per le socket (I)

Per comunicare attraverso la rete, due processi devono dapprima definire i rispettivi **transport end point** tramite la seguente chiamata di sistema:

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

```
s = socket(int domain, int type, int protocol);
```

dove:

- `domain` indica il dominio della socket (e.g., `AF_INET`);
- `type` indica il se verrà usato il paradigma connection oriented (`SOCK_STREAM`) oppure quello connectionless (`SOCK_DGRAM`);
- `protocol` specifica il protocollo da utilizzare: se impostato a 0, viene scelto automaticamente il protocollo più adatto in base ai primi due parametri.

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

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

```
int bind(int sockfd, const struct sockaddr *address, size_t add_len);
```

Lega l'indirizzo dell'host ad un socket identifier (solitamente `sockfd` è il valore restituito dalla system call `socket`).

Chiamate di sistema per le socket (II)

Il server si mette in ascolto per mezzo della seguente system call:

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

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

dove `queue_size` indica la dimensione della coda di client in attesa delle connessioni.

Il client si connette con una `connect` al server in corrispondenza della socket indicata dalla struttura `address`:

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

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

```
int connect(int sockfd, const struct sockaddr *address, size_t add_len);
```

Il server accetta una connessione tramite la chiamata di sistema `accept`:

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

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

```
int accept(int sockfd, struct sockaddr *address, size_t *add_len);
```

Accettando una richiesta di connessione, viene creata una **nuova** socket (il cui descrittore è il valore di ritorno di `accept`) che verrà usata per comunicare con il client.

Chiamate di sistema per le socket (III)

Una volta stabilita la connessione fra client e server, si possono usare le system call `send` e `recv` per trasmettere e ricevere dati attraverso le socket:

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

```
ssize_t send(int sockfd, const void *buffer, size_t length, int flags);
ssize_t recv(int sockfd, void *buffer, size_t length, int flags);
```

se `flags` vale 0, allora `send` e `recv` equivalgono, rispettivamente alle system call `write` e `read`. Altrimenti `flags` può assumere i seguenti valori, per quanto riguarda `send`:

- `MSG_OOB`: il processo invia dati “out of band”;
- `MSG_DONTROUTE`: vengono ignorate le condizioni di routing dei pacchetti sottostanti al protocollo utilizzato.

Per quanto riguarda `recv` invece `flags` può assumere i seguenti valori:

- `MSG_PEEK`: i dati vengono letti, ma non “consumati” in modo che una successiva `recv` riceverà ancora le stesse informazioni;
- `MSG_OOB`: legge soltanto i dati “out of band”;
- `MSG_WAITALL`: la `recv` ritorna soltanto quando la totalità dei dati è disponibile.

Una socket viene distrutta chiudendo il file descriptor associato alla connessione.

Esempio di applicazione connection oriented

Come esempio di applicazione connection oriented si vuole realizzare un “maiuscolatore”, i.e., un server che riceve delle stringhe di testo dai client, restituendole a questi ultimi dopo aver convertito in maiuscolo le lettere.

```
/* upperserver.c : un server per maiuscolare linee di testo */
#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 <ctype.h>
#include <sys/times.h>

#define SERVER_PORT 1313
#define LINESIZE 80
```

Esempio di server: maiuscolatore

```
void upperlines(int in, int out) {
    char inputline[LINESIZE];
    int len, i;

    while ((len = recv(in, inputline, LINESIZE, 0)) > 0) {
        for (i=0; i < len; i++)
            inputline[i] = toupper(inputline[i]);
        send(out, inputline, len, 0);
    }
}

int main (unsigned argc, char **argv) {
    int sock, client_len, fd;
    struct sockaddr_in server,client;

    /* impostazione del transport end point */
    if((sock = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
        perror("chiamata alla system call socket fallita");
        exit(1);
    }

    server.sin_family = AF_INET;
    server.sin_addr.s_addr = htonl(INADDR_ANY);
    server.sin_port = htons(SERVER_PORT);

    /* legghiamo l'indirizzo al transport end point */
    if (bind(sock, (struct sockaddr *)&server, sizeof server) == -1) {
```

```
    perror("chiamata alla system call bind fallita");
    exit(2);
}

listen(sock, 1);

/* gestione delle connessioni dei clienti */
while (1) {
    client_len = sizeof(client);
    if ((fd = accept(sock, (struct sockaddr *)&client, &client_len)) < 0) {
        perror("accepting connection");
        exit(3);
    }
    fprintf(stderr, "Aperta connessione.\n");
    send(fd, "Benvenuto all'UpperServer!\n", 27, 0);
    upperlines(fd, fd);
    close(fd);
    fprintf(stderr, "Chiusa connessione.\n");
}
}
```

Test del server

Prima di tutto compiliamo e lanciamo in esecuzione il server:

```
> gcc -o upperserver upperserver.c # compilazione del server
> ./upperserver                    # esecuzione del server
```

Per verificare il funzionamento del server, possiamo servirci del comando telnet:

```
> telnet 127.0.0.1 1313
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
Benvenuto all'UpperServer!
prova
PROVA
```

Per terminare la sessione premere `Ctrl +]` e impartire il comando `quit`:

```
^]
telnet> quit
Connection closed.
>
```

Connectionless oriented model

- Nel modello non orientato alla connessione, un processo che voglia inviare/ricevere dei messaggi in rete deve
 - creare la propria socket locale,
 - associare il proprio indirizzo di rete alla socket tramite una chiamata a `bind`.
- L'indirizzo di destinazione può anche essere di tipo "broadcast".
- Differenze dal punto di vista della programmazione rispetto al modello connection oriented:
 - il server, dopo aver legato il proprio indirizzo di rete alla socket locale, non serve che si metta in ascolto, in quanto il client farà la maggior parte del lavoro;
 - il client deve creare una propria socket locale ed associarvi il proprio indirizzo di rete con una chiamata a `bind`.

recvfrom e sendto

Per ricevere (risp., spedire) messaggi nel modello connectionless oriented, si utilizza la chiamata `recvfrom` (risp., `sendto`):

```
ssize_t recvfrom(int sockfd, void *message, size_t length,  
                int flags, struct sockaddr *send_addr,  
                size_t *add_len);
```

```
ssize_t sendto(int sockfd, void *message, size_t length,  
              int flags, const struct sockaddr *dest_addr,  
              size_t dest_len);
```

dove:

- `sockfd` è il socket file descriptor,
- `message` è il puntatore al buffer contenente il messaggio,
- `length` è la lunghezza del messaggio,
- `flags` rappresenta i flag come sono stati definiti per la `recv` e la `send`,
- `send_addr` (risp., `dest_addr`) è un puntatore alla struttura che identifica il mittente (risp., destinatario) del messaggio,
- `add_len` (risp., `dest_len`) è la lunghezza della struttura `send_addr` (risp., `dest_addr`).

Struttura del server

Sequenza tipica di chiamate di sistema di un server nel modello non orientato alla connessione:

- Creazione della socket:

```
sockfd = socket(AF_INET, SOCK_DGRAM, 0);
```

- Binding dell'indirizzo alla socket:

```
bind(sockfd, (struct sockaddr *)&server, sizeof(server));
```

- Loop per servire i client:

```
while(1) {  
    /* ricezione del messaggio (richiesta) inviato dal client */  
    recvfrom(sockfd,message1,len1,0,(struct sockaddr *)&client,&client_len);  
    ...  
    /* elaborazione della risposta */  
    ...  
    /* invio della risposta */  
    sendto(sockfd,message2,len2,0,(struct sockaddr *)&client,sizeof(client));  
}
```

Struttura del client

Sequenza tipica di chiamate di sistema di un client nel modello non orientato alla connessione:

- Creazione della socket:

```
sockfd = socket(AF_INET, SOCK_DGRAM, 0);
```

- Binding dell'indirizzo alla socket:

```
bind(sockfd, (struct sockaddr *)&client, sizeof(client));
```

- Invio della richiesta al server:

```
sendto(sockfd,message1,len1,0,(struct sockaddr *)&server,sizeof(server));
```

- Ricezione della risposta dal server:

```
recvfrom(sockfd,message2,len2,0,(struct sockaddr *)&server,&server_len);
```


Esercizi

- Si completi l'esempio del maiuscolatore, scrivendo il codice del client (caso del modello connection oriented). La struttura di quest'ultimo sarà la seguente:

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

main() {
    int sockfd;

    if( (sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
        perror("chiamata alla system call socket fallita");
        exit(1);
    }
    /* connessione al server */
    /* invio e ricezione della stringa */
    /* chiusura della connessione */
}
```

- Modificare il programma `upperserver.c` in modo che accetti più connessioni contemporaneamente (utilizzando la `fork`).