# Socket UDP

## Librerie

Le librerie necessarie per lo sviluppo di un client/server UDP sono le seguenti:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <arpa/inet.h> // struct in_addr
```

## Variabili

Per una buona compresione del codice. è fondamentale capire quali variabili siano necessaria per il client.

- `int sockfd;`, per conservare il descrittore per la socket.
- `struct sockaddr_in local_addr, remote_addr;`, per conservare rispettivamente indirizzo per il mittente e per il destinatario.
- `socklen_t len = sizeof(struct sockaddr_in);`, per conservare l'effettiva dimensione della socket.
- `char buffer[1000];`, per descrivere il messaggio da inviare al destinatario.

> `if((sockfd = socket(PF_INET, SOCK_DGRAM, 0)) < 0) {}` è necessario al fine di chiudere il tutto se vi è stato un problema durante l'apertura della socket.

- `memset(&local_addr, 0, sizeof(local_addr));`, copies the character **c** (i.e. 0) to the first **n** (i.e. `sizeof(local_addr)`) characters of the string pointed to, by the argument **str** (i.e. `&local_addr`).
- `local_addr.sin_family = AF_INET;`, per settare la famiglia di appartenenza per il mittente.
- `local_addr.sin_port = htons(atoi("<PORT>"));` per settare la porta per il mittente.

### `socket()`

**Come può essere creata una socket**? Utilizzando le librerie `<sys/types.h>` e `<sys/socket.h>`, creare una socket è possibile mediante la funzione `socket(int domain, int type, int protocol)`.

> `socket()` restituisce un descrittore per la socket, `null` altrimenti.

All'interno della funzione, possiamo distinguere i seguenti parametri:

- `domain`, il quale può essere uguale ad `PF_INET` o `PF_INET6` (analogamente, possono essere specificati `AF_INET` o `AF_INET6`).
- `type`, il quale può essere, se in precedenza è stato specificato `PF_INET`, uguale a `SOCK_DGRAM` o `SOCK_STREAM`.
- `protocol`, il quale può essere, se in precedenza è stato specificato `PF_INET`, uguale a `IPPROTO_UDP` o `IPPROTO_TCP`. Se specificato `0`, allora verrà scelto il *protocollo più adatto*.

### `bind()`

```
#include <sys/types.h>
#include <sys/socket.h>
bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
```

`bind()` è una funzione necessaria ad inserire i dati (i.e. *indirizzo* e *porta*) all'interno della socket.
Una `bind()` *in input* è utile ad indicare al sistema i parametri che identificano la macchina a cui egli stesso deve inviare informazioni.
Una `bind()` *in output* è utile ad indicare al sistema i parametri relativi al mittente nell'intestazione dei pacchetti.

> L'utilizzo di `bind()` è comunemente legato al server, il quale resta in ascolto per future connessioni su una specifica porta.

All'interno della funzione, possiamo distinguere i seguenti parametri:

- `sockfd`, il descrittore per la socket restituito dalla chiamata `socket()`.
- `*addr`, il quale indica l'indirizzo IP. Egli può essere rappresentato talvolta mediante macro come `INADDR_ANY` e `INADDR_BROADCAST`, rispettivamente per accettare da qualsiasi indirizzo o inviare messaggi in broadcast.

### `sendto()`

`sendto(int socket, void *buffer, size_t size, int flags, struct sockaddr *addr, size_t length)` è la funzione utilizzata per trasmettere un messaggio verso il destinatario.

All'interno della funzione, possiamo distinguere i seguenti parametri:

- `socket`, indica il descrittore per la socket restituito dalla chiamata `socket()`.
- `*buffer`, un buffer contenente il messaggio da inviare al destinatario (e.g. `msg`, `char msg[DIM]`).
- `size`, indica la lunghezza del buffer. Per semplicità possiamo settare questo a `strlen(msg)`.

- `flags`, indica dei flags per il messaggio. Per semplicità possiamo settare questo a $0$.
- `*addr`, indica l'indirizzo a cui inviare, ovvero `(struct sockaddr *) &remote_addr`.
- `length`, indica l'effettiva lunghezza per `*addr`. Per semplicità possiamo settare questo a `sizeof(dest_addr)`.

### `recvfrom()`

`recvfrom(int socket, void *buffer, size_t size, int flags, struct sockaddr *addr, size_t *length)` è la funzione utilizzata per restare in ascolto di un messaggio in arrivo.

All'interno della funzione, possiamo distinguere i seguenti parametri:

- `socket`, indica il descrittore per la socket restituito dalla chiamata `socket()`.
- `*buffer`, un buffer contenente il messaggio da inviare al destinatario (e.g. `msg`, `char msg[DIM]`).
- `size`, indica la lunghezza del buffer. Per semplicità possiamo settare questo a `DIM - 1`.
- `flags`, indica dei flags per il messaggio. Per semplicità possiamo settare questo a $0$.
- `*addr`, indica l'indirizzo a cui inviare, ovvero `(struct sockaddr *) &remote_addr`.
- `length`, indica l'effettiva lunghezza per la socket. Per semplicità possiamo settare questo a `&len`.

### `close()`

Utilizzando la funzione `close()` è possibile chiudere una socket precedentemente aperta.

---

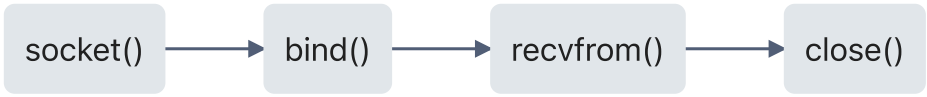
> Per effettuare una connessione SSH tra VSCode e VM, inserire `<username>@<IP>`. Si noti che l'interfaccia per la VM deve essere *attached to* `Host-only Adapter`. Benché quest'ultima parte dovrebbe essere già presente, si noti che talvolta potrebbe essere necessario il set del file `/etc/network/interfaces` secondo DHCP.

> Come può essere convertito un indirizzo da stringa ad effettivo indirizzo? Attraverso la funzione `pton()`, acronimo di *presentation to network* ( e.g. `inet_pton(AF_INET, argv[1], &(dest_addr.sin_addr))` );

> Come può essere convertito un indirizzo da effettivo indirizzo a stringa?
> Attraverso la funzione `ntop()`, acronimo di *network to presentation*.

## Grafi

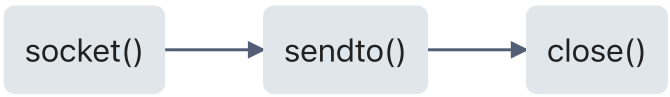
### Oneway Server




```
/* UDP Receiver - Server*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <arpa/inet.h>
#include <errno.h>
#define DIM 256

int main(int argc, char* argv[]) {
        int sockfd; // Socket descriptor
        struct sockaddr_in local_addr, remote_addr; // Init of local and remote
IP address
        socklen_t len = sizeof(struct sockaddr_in);
        char buffer[DIM];

        // Input params check
        if (argc < 2) { printf("Error! Use: receiver listening_PORT"); return 0;
}

        // Create socket
        if ((sockfd = socket(PF_INET,SOCK_DGRAM,0)) <0) { printf("Error");
return -1; }

        // Setup local address
        memset(&local_addr, 0, len);
        local_addr.sin_family = AF_INET;
        local_addr.sin_port = htons(atoi(argv[1]));



        // Binding part
        if (bind(sockfd, (struct sockaddr *) &local_addr, len) < 0) {
printf("Binding error!"); return -1; }

        // Recv
```

```
        for (;;) {

                recvfrom(sockfd, buffer, DIM,0, (struct sockaddr *)
&remote_addr, &len);
                printf("Packet from IP:%s Port:%d msg:%s \n",
inet_ntoa(remote_addr.sin_addr), ntohs(remote_addr.sin_port), buffer);
        }
}
```

## Oneway Client

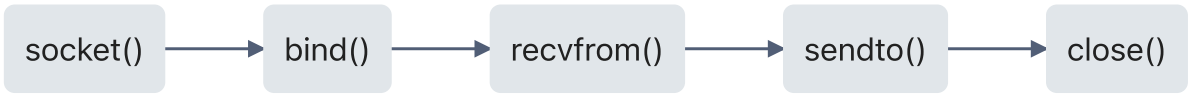



```
/* UDP Sender */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <arpa/inet.h>
#define DIM 1000
int main(int argc, char* argv[]) {
        int sockfd; // Socket descriptor
        struct sockaddr_in dest_addr; // Init dest IP address
        char buffer[DIM]; // Message
        // Input params check
        if (argc < 3) { printf("Use: sender IP_dest PORT_dest"); return 0; }

        // Create socket
        if ((sockfd = socket(PF_INET,SOCK_DGRAM,0)) < 0) { printf("\nError!");
return -1; }

        // Setup address
        memset( &dest_addr, 0, sizeof(dest_addr));
        dest_addr.sin_family = AF_INET;
        inet_pton(AF_INET, argv[1], &(dest_addr.sin_addr));
        dest_addr.sin_port = htons(atoi(argv[2]));

        for (;;) {
                // Input message
                memset(&buffer, 0, sizeof(buffer));
                printf("\nInsert an message: ");
                scanf("%s", buffer);

                // Send part
                printf("Sending %s\n",buffer);
                sendto(sockfd, buffer, sizeof(buffer), 0, (struct sockaddr *)
&dest_addr, sizeof(dest_addr));
```

```
        }
}
```

## Bidirectional ClientServer

- Server




- Client


```
/* Simple IPv4 UDP bidirectional chat */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#define DIM 1000
int main(int argc, char**argv) {

        int sockfd, n; // Socket descriptor and n
        struct sockaddr_in local_addr, remote_addr; // Init IP Address

        // Set sockaddr struct lenght, useful for bind(), sendto() and rcvfrom()
        socklen_t len = sizeof(struct sockaddr_in);

        // Init msg to send to receiver
        char msg[DIM];

        // Input params check
        if (argc < 4) {
                printf("Error! Use <Destination_IP> <Destination_Port>
<ListeningPORT>");
                return 0;
        }

        // Create process, one (i.e. child) will have the role of receiver,
        // the other (i.e. parent) will have the role of sender
        // Reminder: On success, the PID of the child process is returned in the
parent,
        // and 0 is returned in the child.
        if (!fork()) {
```

```
                // Child code, receiver role
                // Create socket
                if ((sockfd = socket(PF_INET,SOCK_DGRAM,0)) < 0) {
printf("\nError"); return -1; }

                // Setup local IP address

                // Make sure that the data structure is empty
                memset(&local_addr, 0, sizeof(local_addr));

                // Set IPv4 as local address family
                local_addr.sin_family = AF_INET;

                // Set listening port to receiver
                // N.B. atoi() converts str to int, htons converts port from
host to network byte order
                local_addr.sin_port=htons(atoi(argv[3]));

                // Set local addr to socket
                if (bind(sockfd, (struct sockaddr *) &local_addr, len) < 0) {
                        printf("Binding error!");
                        return -1;
                }

                while (1) {
                        // recvfrom() reads incoming data and captures the
address from which the data was sent.
                        // recvfrom() returns the number of bytes received.
                        n = recvfrom(sockfd, msg, DIM - 1, 0, (struct sockaddr
*) &remote_addr, &len);

                        // Mark the end of message with \0
                        msg[n] = 0;

                        // ntoa() returns address in dot notation,
                        // ntohs() returns port from network to host bye order
                        printf("From IP:%s Port:%d msg:%s \n",
inet_ntoa(remote_addr.sin_addr), ntohs(remote_addr.sin_port), msg);
                }

                return 0;
        } else {
                // Parent code, sender role
                // Create socket

                if ((sockfd = socket(PF_INET,SOCK_DGRAM,0)) < 0) {
printf("\nError!"); return -1; }

                // Setup remote IP address
```

```
                memset(&remote_addr, 0, len);

                // Set IPv4 as remote addr family
                remote_addr.sin_family = AF_INET;

                // Set dest address (i.e. argv[1]) to remote_addr.sin_addr
                inet_pton(AF_INET, argv[1], &(remote_addr.sin_addr));

                // Set receiver listening port
                // N.B. atoi() converts str to int, htons converts port from
host to network byte order
                remote_addr.sin_port = htons(atoi(argv[2]));

                // Send message
                // Reminder: fgets(msg, len, *fp) read input from file as stdin
                while (fgets(msg, DIM, stdin) != NULL) {
                        sendto(sockfd, msg, strlen(msg), 0, (struct sockaddr *)
&remote_addr, len);
                }
                return 0;
        }

}
```

> Attraverso il codice `chat.c`, lo stesso codice viene utilizzato sia dal client che dal server. In questo modo, ogni host si comporta sia da client che da server attraverso i due processi creati mediante `fork()`.