OPERATING SYSTEMS: SESSION4

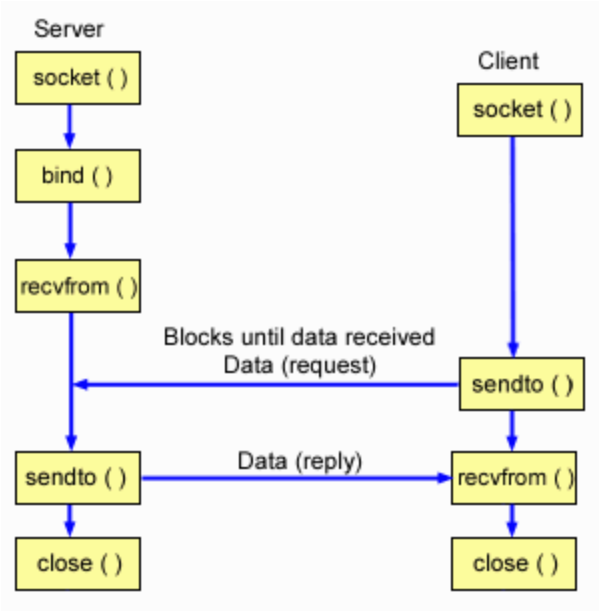
JANUARY 2023

TecnoCampus Escuela Superior Politécnica

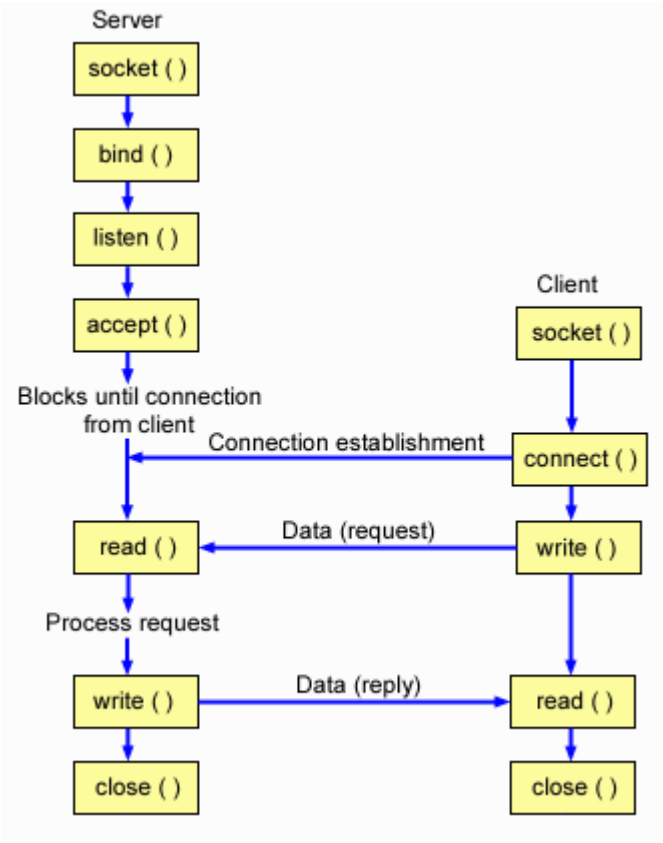
Centro adscrito a la Universitat Pompeu Fabra Barcelona

TecnoCampus 10 años

NON-CONNECTION ORIENTED COMMUNICATION



CONNECTION ORIENTED COMMUNICATION



ENDIANNESS ON SOCKETS

- ENDIANNESS:
  - The way information is written could be:
  - LITTLE ENDIAN/BIG ENDIAN
  - LITTLE ENDIAN:
    - DEC
    - INTEL
  - BIG ENDIAN:
    - MOTOROLA
    - TCP/IP PROTOCOL

1 BYTE DATA SEQUENCE:
0x47-0xEF-0X15-0X2A-0X89-0X37-0XB4-0X33

BIG ENDIAN

0x47	0xEF	0x15	0x2A	0x89	0x37	0xB4	0x33

0x47	0xEF	0x15	0x2A	0x89	0x37	0xB4	0x33

LITTLE ENDIAN

2 BYTES DATA SEQUENCE :
0x47EF-0X152A-0X8937-0XB433

BIG ENDIAN

0x47	0xEF	0x15	0x2A	0x89	0x37	0xB4	0x33

0xEF	0x47	0x2A	0x15	0x37	0x89	0x33	0xB4

LITTLE ENDIAN

4 BYTES DATA SEQUENCE :
0x47EF152A-0X8937B433

BIG ENDIAN

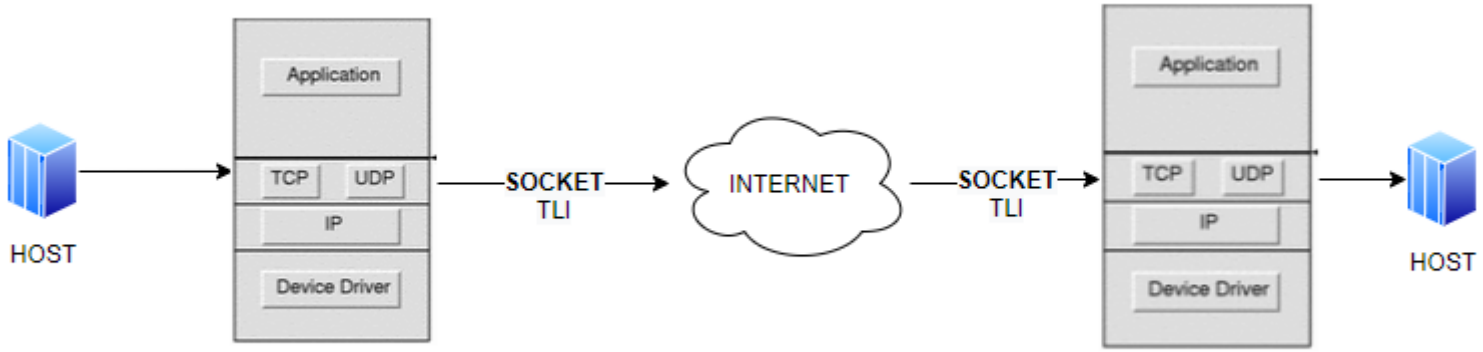
0x47	0xEF	0x15	0x2A	0x89	0x37	0xB4	0x33

0x2A	0x15	0xEF	0x47	0x33	0xB4	0x37	0x89

LITTLE ENDIAN

ENDIANNESS ON SOCKETS

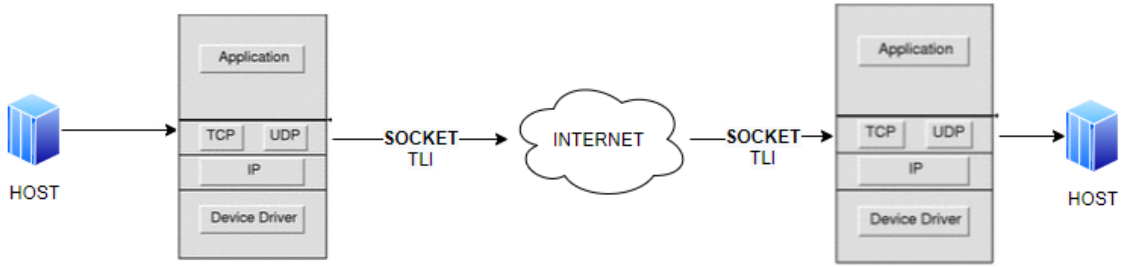
- When working with SOCKETS you must be CAREFUL with ENDIANNESS



HOST1	TLI 1	TLI 2	HOST2
LE	BE	BE	LE
BE	BE	BE	BE
LE	BE	BE	BE
BE	BE	BE	LE

ENDIANNESS ON SOCKETS

- When working with SOCKETS you must be CAREFUL with ENDIANNESS: SHORT—PORT, LONG—IP ADDRESS
- HTON→**htons**(), **htonl**()
- NTOH→**ntohs**(), **ntohl**()



HOST1	CONVERSION	TLI 1	TLI 2	CONVERSION	HOST2
LE	**HTON()**	BE	BE	**NTOH()**	LE
BE	**HTON()**	BE	BE	**NTOH()**	BE
LE	**HTON()**	BE	BE	**NTOH()**	BE
BE	**HTON()**	BE	BE	**NTOH()**	LE

C FUNCTIONS FOR SOCKETS: SOCKET: to have a SOCKET-FILE DESCRIPTOR

```c
/* Try to create TCP socket */
int sock
sock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
if (sock < 0) {
    err_sys("Error socket");
}
```

```
SOCKET(2)

NAME
       socket - create an endpoint for communication

SYNOPSIS
       #include <sys/types.h>          /* See NOTES */
       #include <sys/socket.h>

       int socket(int domain, int type, int protocol);

DESCRIPTION
       socket()  creates  an  endpoint  for communication and returns a file descriptor that
       refers to that endpoint.  The file descriptor returned by a successful call  will  be
       the lowest-numbered file descriptor not currently open for the process.
```

C FUNCTIONS FOR SOCKETS:

```
/* Try to create TCP socket */
sock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
if (sock < 0) {
    err_sys("Error socket");
}
```

INTERNET

TCP

```
void err_sys(char *mess) { perror(mess); exit(1); }
```

FAMILY: always **PF_INET** for INTERNET SOCKETS 'Protocol_Family INET (InterNET)'

TYPE: always **SOCK_STREAM** for CONNECTION-ORIENTED SOCKETS

PROTOCOL: always **IPPROTO_TCP** for CONNECTION-ORIENTED SOCKETS

```
SOCK_STREAM     Provides sequenced, reliable, two-way, connection-based byte streams.
                An out-of-band data transmission mechanism may be supported.
```

C FUNCTIONS FOR SOCKETS:

Name	Purpose	Man page
AF_UNIX	Local communication	unix(7)
AF_LOCAL	Synonym for AF_UNIX	
AF_INET	IPv4 Internet protocols	ip(7)
AF_AX25	Amateur radio AX.25 protocol	ax25(4)
AF_IPX	IPX - Novell protocols	
AF_APPLETALK	AppleTalk	ddp(7)
AF_X25	ITU-T X.25 / ISO-8208 protocol	x25(7)
AF_INET6	IPv6 Internet protocols	ipv6(7)
AF_DECnet	DECet protocol sockets	
AF_KEY	Key management protocol, originally developed for usage with IPsec	
AF_NETLINK	Kernel user interface device	netlink(7)
AF_PACKET	Low-level packet interface	packet(7)
AF_RDS	Reliable Datagram Sockets (RDS) protocol	rds(7) rds-rdma(7)
AF_PPPOX	Generic PPP transport layer, for setting up L2 tunnels (L2TP and PPPoE)	
AF_LLC	Logical link control (IEEE 802.2 LLC) protocol	
AF_IB	InfiniBand native addressing	
AF_MPLS	Multiprotocol Label Switching	
AF_CAN	Controller Area Network automotive bus protocol	
AF_TIPC	TIPC, "cluster domain sockets" protocol	
AF_BLUETOOTH	Bluetooth low-level socket protocol	
AF_ALG	Interface to kernel crypto API	
AF_VSOCK	VSOCK (originally "VMWare VSockets") protocol for hypervisor-guest communication	vsock(7)
AF_KCM	KCM (kernel connection multiplexor) interface	
AF_XDP	XDP (express data path) interface	

TCP

UDP

SOCK_STREAM Provides sequenced, reliable, two-way, connection-based byte streams. An out-of-band data transmission mechanism may be supported.

SOCK_DGRAM Supports datagrams (connectionless, unreliable messages of a fixed maximum length).

SOCK_SEQPACKET Provides a sequenced, reliable, two-way connection-based data transmission path for datagrams of fixed maximum length; a consumer is required to read an entire packet with each input system call.

SOCK_RAW Provides raw network protocol access.

SOCK_RDM Provides a reliable datagram layer that does not guarantee ordering.

SOCK_PACKET Obsolete and should not be used in new programs; see packet(7).

Some socket types may not be implemented by all protocol families.

Since Linux 2.6.27, the type argument serves a second purpose: in addition to specifying a socket type, it may include the bitwise OR of any of the following values, to modify the behavior of socket():

SOCK_NONBLOCK Set the O_NONBLOCK file status flag on the open file description (see open(2)) referred to by the new file descriptor. Using this flag saves extra calls to fcntl(2) to achieve the same result.

SOCK_CLOEXEC Set the close-on-exec (FD_CLOEXEC) flag on the new file descriptor. See the description of the O_CLOEXEC flag in open(2) for reasons why this may be useful.

ADDRESS CONFIGURATION is FAMILY DEPENDENT

For INTERNET FAMILY:

- IP ADDRESS
- PORT

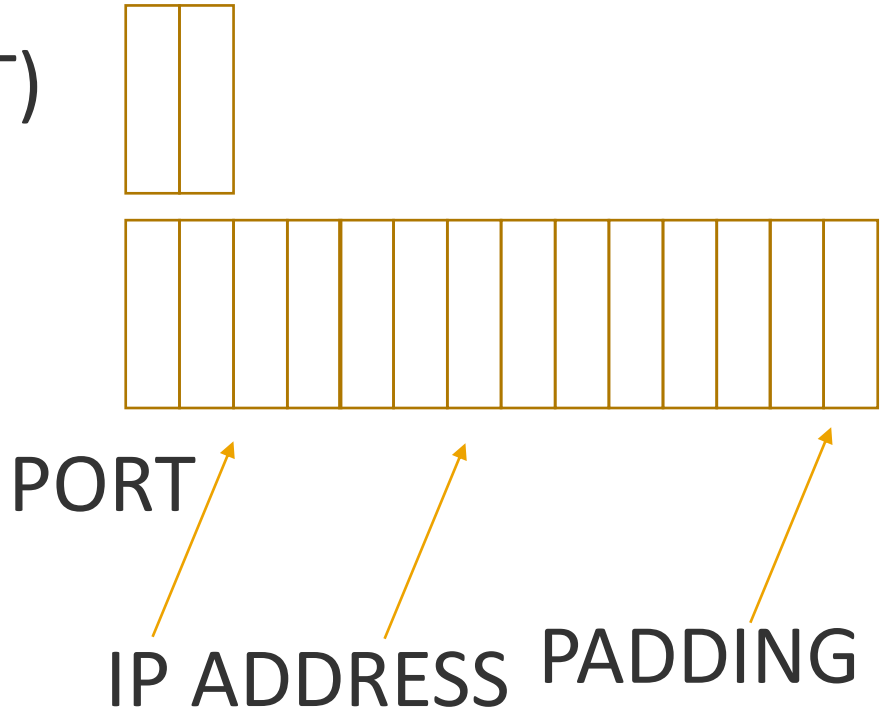
C FUNCTIONS FOR SOCKETS: ADDRESS CONFIGURATION

SOCKADDR

```
struct sockaddr {
        u_short sa_family;              /* address family */
        char    sa_data[14];            /* up to 14 bytes of direct address */
};
```

FAMILY (U_SHORT)

DATA (CHAR[14])



ADDRESS CONFIGURATION is FAMILY DEPENDENT

For INTERNET FAMILY:

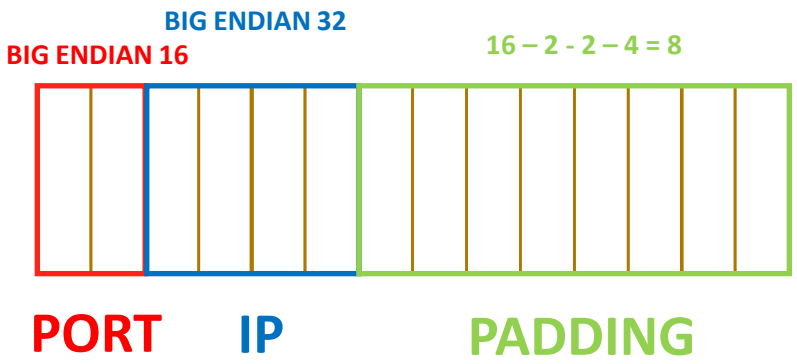
- IP ADDRESS
- PORT

C FUNCTIONS FOR SOCKETS: ADDRESS CONFIGURATION

```
/* Structure describing an Internet (IP) socket address. */
#if  __UAPI_DEF_SOCKADDR_IN
#define __SOCK_SIZE__    16              /* sizeof(struct sockaddr)      */
struct sockaddr_in {
  __kernel_sa_family_t  sin_family;      /* Address family               */
  __be16                sin_port;        /* Port number                  */
  struct in_addr        sin_addr;        /* Internet address             */

  /* Pad to size of `struct sockaddr'. */
  unsigned char         __pad[__SOCK_SIZE__ - sizeof(short int) -
                        sizeof(unsigned short int) - sizeof(struct in_addr)];
};
```

```
/* Internet address. */
struct in_addr {
    __be32  s_addr;
};
```



C FUNCTIONS FOR SOCKETS: ADDRESS CONFIGURATION

```c
struct sockaddr_in echoserver;
```

```c
/* Set information for sockaddr_in */
memset(&echoserver, 0, sizeof(echoserver));        /* reset memory */
echoserver.sin_family = AF_INET;                   /* Internet/IP */
echoserver.sin_addr.s_addr = inet_addr(argv[1]);   /* IP address */
echoserver.sin_port = htons(atoi(argv[3]));        /* server port */
```

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	7F	0	0	1	0	0	0	0	0	0	0	0
2	0	3	E8	7F	0	0	1	0	0	0	0	0	0	0	0

```
SO: ./tcp_client1b 127.0.0.1 hola 1000
[0]=0,[1]=0,[2]=0,[3]=0,[4]=0,[5]=0,[6]=0,[7]=0,[8]=0,[9]=0,[10]=0,[11]=0,[12]=0,[13]=0,[14]=0,[15]=0,
[0]=2,[1]=0,[2]=0,[3]=0,[4]=0,[5]=0,[6]=0,[7]=0,[8]=0,[9]=0,[10]=0,[11]=0,[12]=0,[13]=0,[14]=0,[15]=0,
[0]=2,[1]=0,[2]=0,[3]=0,[4]=7F,[5]=0,[6]=0,[7]=1,[8]=0,[9]=0,[10]=0,[11]=0,[12]=0,[13]=0,[14]=0,[15]=0,
[0]=2,[1]=0,[2]=3,[3]=E8,[4]=7F,[5]=0,[6]=0,[7]=1,[8]=0,[9]=0,[10]=0,[11]=0,[12]=0,[13]=0,[14]=0,[15]=0,
SO:
```

```c
{
int cont;
char* pointer;

pointer = (char*)&echoserver;
for (cont = 0; cont < sizeof(echoserver); cont++)
    printf("[%d]=%X,",cont,(unsigned char)*pointer++);
printf("\n");
}
```

C FUNCTIONS FOR SOCKETS: ADDRESS CONFIGURATION

```c
/* Set information for sockaddr_in */
memset(&echoserver, 0, sizeof(echoserver));        /* Reset memory */
{
  int cont;
  char* pointer;

  pointer = (char*)&echoserver;
  for (cont = 0; cont < sizeof(echoserver); cont++)
      printf("[%d]=%X,",cont,(unsigned char)*pointer++);
  printf("\n");
}
```

```c
echoserver.sin_family = AF_INET;                    /* Internet/IP */

  {
  int cont;
  char* pointer;

  pointer = (char*)&echoserver;
  for (cont = 0; cont < sizeof(echoserver); cont++)
      printf("[%d]=%X,",cont,(unsigned char)*pointer++);
  printf("\n");
}
```

```c
echoserver.sin_addr.s_addr = inet_addr(argv[1]);  /* Server address */

  {
  int cont;
  char* pointer;

  pointer = (char*)&echoserver;
  for (cont = 0; cont < sizeof(echoserver); cont++)
      printf("[%d]=%X,",cont,(unsigned char)*pointer++);
  printf("\n");
}
```

```c
echoserver.sin_port = htons(atoi(argv[3]));        /* Server port */

  {
  int cont;
  char* pointer;

  pointer = (char*)&echoserver;
  for (cont = 0; cont < sizeof(echoserver); cont++)
      printf("[%d]=%X,",cont,(unsigned char)*pointer++);
  printf("\n");
}

exit(1);
```

C FUNCTIONS FOR SOCKETS: ADDRESS CONFIGURATION

```
struct sockaddr_in echoserver;
```

```
/* Set information for sockaddr_in */
memset(&echoserver, 0, sizeof(echoserver));          /* reset memory */
echoserver.sin_family = AF_INET;                     /* Internet/IP */
echoserver.sin_addr.s_addr = inet_addr(argv[1]);     /* IP address */
echoserver.sin_port = htons(atoi(argv[3]));          /* server port */
```

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	7F	0	0	1	0	0	0	0	0	0	0	0
2	0	3	E8	7F	0	0	1	0	0	0	0	0	0	0	0

The inet_addr() function converts the Internet host address cp from IPv4 numbers-and-dots notation into binary data in network byte order. If the input is invalid, INADDR_NONE (usually -1) is returned. Use of this function is problematic because -1 is a valid address (255.255.255.255). Avoid its use in favor of inet_aton(), inet_pton(3), or getaddrinfo(3), which provide a cleaner way to indicate error return.

STRING ("192.168.0.1")

inet_addr()

INT (192 – 168 – 0 - 1)

```
uint32_t htonl(uint32_t hostlong);

uint16_t htons(uint16_t hostshort);

uint32_t ntohl(uint32_t netlong);

uint16_t ntohs(uint16_t netshort);
```

C FUNCTIONS FOR SOCKETS: ADDRESS CONFIGURATION

A **port number** is a **16-bit unsigned integer**, thus ranging from **0** to **65535**.

For **TCP**, port number 0 is reserved and cannot be used, while for **UDP**, the source port is optional and a value of zero means no port.

A **process** associates its input or output channels via an **Internet socket**, which is a type of **file descriptor**, associated with a **transport protocol**, an **IP address**, and a **port number**.

This is known as **binding**. A socket is used by a process to send and receive data via the network. The operating system's networking software has the task of transmitting outgoing data from all application ports onto the network and forwarding arriving network packets to processes by matching the packet's IP address and port number to a socket.

Applications implementing common services often use specifically reserved well-known port numbers for receiving service requests from clients. The **well-known ports** are defined by convention overseen by the Internet Assigned Numbers Authority (**IANA**).

Conversely, the client end of a connection typically uses a high port number allocated for short term use, therefore called an **ephemeral port**.

RANGE PORT:	WELL-KNOWN :	0→1023 (SYSTEM PORTS)
	NON-STANDARD SERVICES :	1024→4095 (49151)
	EPHEMERAL:	4096 (49152)→65535

C FUNCTIONS FOR SOCKETS: CONNECTION REQUEST

- From CLIENT point of view (for the 5-tuple parameters) we have:

  - THE PROTOCOL/FAMILY
  - THE REMOTE ADDRESS (IP+PORT)

  - DO WE NEED TO CONFIGURE THE LOCAL ADDRESS (IP+PORT)?

C FUNCTIONS FOR SOCKETS: CONNECTION REQUEST

- Do you need to KNOW YOUR PHONE NUMBER TO MAKE A CALL?
  - SO you DO NOT NEED TO CONFIGURE YOUR LOCAL ADDRESS TO REQUEST FOR A CONNECTION:
    - The OS will know the right LOCAL IP ADDRESS TO USE and will configure it by ITSELF
    - The OS will select a LOCAL PORT for the CLIENT (A RANDOM PORT NUMBER > 4096)

C FUNCTIONS FOR SOCKETS: 5-TUPLE PARAMETERS FOR CLIENT

- PROTOCOL/FAMILY: PF_INET/IPPROTO_TCP/AF_INET
- REMOTE IP: .sin_addr.s_addr
- REMOTE PORT: .sin_port
- LOCAL IP: OS SELECTED
- LOCAL PORT: OS SELECTED RANDOMLY

- So we can request for a connection to the SERVER as we have all 5 PARAMETERS

C FUNCTIONS FOR SOCKETS: CONNECTION REQUEST

- Asking for a CONNECTION:

```
/* Try to have a connection with the server */
result = connect(sock, (struct sockaddr *) &echoserver, sizeof(echoserver));
if (result < 0) {
    err_sys("Error connect");
}
```

C FUNCTIONS FOR SOCKETS: CLOSE CONNECTION

- Once communication has been done:
  - We must deallocate resources

```
CLOSE(2)                          Linux Programmer's Manual

NAME
       close - close a file descriptor

SYNOPSIS
       #include <unistd.h>

       int close(int fd);
```

C FUNCTIONS FOR SOCKETS: SERVER

- On the SERVER SIDE we have to:
  - Have a SOCKET-FILE DESCRIPTOR (**SOCKET**)
  - Prepare the LINK between the INTERNET CONNECTION and THE SERVER PROCESS (**BIND**)
  - Wait for a CONNECTION REQUEST (**ACCEPT**)
  - Close the CONNECTION (**CLOSE**)

C FUNCTIONS FOR SOCKETS: SERVER-SOCKET-FILE DESCRIPTOR

```c
int serversock, clientsock;
```

```c
/* Create TCP socket */
serversock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
if (serversock < 0) {
    err_sys("Error socket");
}
```

C FUNCTIONS FOR SOCKETS: SERVER-SOCKET-BINDING

- We need to BIND the socket to SERVER PROCESS:
  - Inform the OS that when a PACKET has DESTINATION IP (SERVER IP) AND DESTINATION PORT (SERVER PORT) it should be linked with the OPENED SOCKET
  - So the SERVER process will be able to manage the information

  - THAT'S THE BINDING PROCESS

C FUNCTIONS FOR SOCKETS: SERVER-SOCKET-BINDING

- We need a SOCKET DESCRIPTOR and LOCAL ADDRESS (SERVER) CONFIGURATION

```
/* Set information for sockaddr_in structure */
memset(&echoserver, 0, sizeof(echoserver));        /* we reset memory */
echoserver.sin_family = AF_INET;                   /* Internet/IP */
echoserver.sin_addr.s_addr = htonl(INADDR_ANY);    /* ANY address */
echoserver.sin_port = htons(atoi(argv[1]));        /* server port */
```

```
/* Bind socket */
result = bind(serversock, (struct sockaddr *) &echoserver, sizeof(echoserver));
if (result < 0) {
    err_sys("Error bind");
}
```

C FUNCTIONS FOR SOCKETS: SERVER-SOCKET-BINDING

- PORT must be UNIQUE

```
echoserver.sin_port = htons(atoi(argv[1]));        /* server port */
```

- ADDRESS could be:
  - ANY ADDRESS/INTERFACE from SERVER DEVICE

```
echoserver.sin_addr.s_addr = htonl(INADDR_ANY);    /* ANY address */
```

  - THE UNIQUELY CONFIGURED ADDRESS/INTERFACE from SERVER

```
echoserver.sin_addr.s_addr = inet_addr(argv[1]);   /* IP address */
```

C FUNCTIONS FOR SOCKETS: SERVER-SOCKET-BINDING

- If you have more than ONE IP
  - You can “accept” connection requests from more than one IP (interface)
  - Or just only one

C FUNCTIONS FOR SOCKETS: SERVER-ACCEPT CONNECTION REQUEST

- The SERVER is ready to wait for a CONNECTION REQUEST

- If the SERVER ACCEPT the connection the SOCKET will be ALREADY CONFIGURED
  - To have the ability to manage more than one connection at the same IP+PORT …
  - The OS will create a **NEW SOCKET DESCRIPTOR** with THE 5-TUPLE PARAMETERS to be used on the current CONNECTION
  - The former SOCKET could be used to ACCEPT a second CONNECTION REQUEST

C FUNCTIONS FOR SOCKETS: SERVER-ACCEPT CONNECTION REQUEST

```
/* Wait for a connection from a client */
clientsock = accept(serversock, (struct sockaddr *) &echoclient, &clientlen);
if (clientsock < 0) {
    err_sys("Error accept");
}
```

WHEN	SOCKET DESCRIPTOR	PROTOCOL	LOCAL-IP	LOCAL-PORT	REMOTE-IP	REMOTE-PORT
BEFORE ACCEPT	serversock	TCP	SERVER-IP	SERVER-PORT	NOT SET	NOT SET
	clientsock	TCP	SERVER-IP	SERVER-PORT	NOT SET	NOT SET
AFTER ACCEPT	serversock	TCP	SERVER-IP	SERVER-PORT	NOT SET	NOT SET
	clientsock	TCP	SERVER-IP	SERVER-PORT	CLIENT-IP	CLIENT-PORT

- You can ONLY use SOCKET DESCRIPTOR clientsock (THE NEW ONE) BECAUSE IT IS THE ONLY ONE WITH 5-TUPLE PARAMETERS

C FUNCTIONS FOR SOCKETS: SERVER-LISTEN

- What could happen if you get a second REQUEST before the first one is ACCEPTED?
  - Do you have a kind of QUEUE: CONNECTION REQUEST in queue…. Waiting to be processed and managed
  - OS will manage the QUEUE

```c
/* Listen socket */
result = listen(serversock, MAXPENDING);
if (result < 0) {
    err_sys("Error listen");
}
```

```
LISTEN(2)                                Linux Programmer's Manual

NAME
       listen - listen for connections on a socket
```

C FUNCTIONS FOR SOCKETS: EXAMPLES

```
/01-sockets/01-tcp
.
├── tcp_client.Makefile
├── tcp_client1.c
├── tcp_client2.c
├── tcp_client3.c
├── tcp_client4.c
├── tcp_server.Makefile
├── tcp_server1.c
├── tcp_server2.c
├── tcp_server3.c
└── tcp_server4.c
```

C FUNCTIONS FOR SOCKETS: EXAMPLE 1 (tcp_server1 – tcp_client1)

CLIENT

3 INPUT PARAMETERS:
1.- SERVER IP
2.- WORD
3.- SERVER PORT

```
Usage: ./tcp_client1 <ip_server> <word> <port>
```

```
int sock;
```

```
sock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
```

```
struct sockaddr_in echoserver;
```

```
memset(&echoserver, 0, sizeof(echoserver));
echoserver.sin_family = AF_INET;
echoserver.sin_addr.s_addr = inet_addr(argv[1]);
echoserver.sin_port = htons(atoi(argv[3]));
```

```
int result;
```

```
result = connect(sock, (struct sockaddr *) &echoserver, sizeof(echoserver));
```

```
while (1);
```

```
close(sock);
```

SERVER

1 INPUT PARAMETER:
1.- SERVER PORT

```
Usage: ./tcp_server1 <port>
```

```
int serversock;
```

```
serversock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
```

```
struct sockaddr_in echoserver;
```

```
memset(&echoserver, 0, sizeof(echoserver));
echoserver.sin_family = AF_INET;
echoserver.sin_addr.s_addr = htonl(INADDR_ANY);
echoserver.sin_port = htons(atoi(argv[3]));
```

```
int result;
```

```
result = bind(serversock, (struct sockaddr *) &echoserver, sizeof(echoserver));
```

```
#define MAXPENDING 5
```

```
result = listen(serversock, MAXPENDING);
```

```
while (1);
    int clientlen;
    clientlen = sizeof(echoclient);
    int clientsock;
    clientsock = accept(serversock, (struct sockaddr *) &echoclient, &clientlen);
    fprintf(stdout, "Client: %s\n", inet_ntoa(echoclient.sin_addr));
    handle_client(clientsock);
```

```
void handle_client(int sock) {...}
while (1);
close(sock);
```

C FUNCTIONS FOR SOCKETS: EXAMPLE 1 (tcp_server1 – tcp_client1)

- CLIENT:
  - 1.- REQUEST a connection to server
  - 2.- LOOP (doing NOTHING)

- SERVER:
  - 1.- BIND socket
  - 2.- LOOP (ready to accept a client request)
    - 2A.- ACCEPT connection request from client
    - 2B.- PRINF client information
    - 2C.- HANDLE CLIENT CONNECTION
      - 2C-1.- LOOP (doing NOTHING)

C FUNCTIONS FOR SOCKETS: EXAMPLE 1 (tcp_server1 – tcp_client1)

- OBJECTIVE:
  - CLIENT-SERVER can be connected
  - CLIENT-SERVER do not share information

C FUNCTIONS FOR SOCKETS: EXAMPLE 1 (tcp_server1 – tcp_client1)

- **nmap**:
  - CONNECTIONS MONITORING
- Execute SERVER (but not CLIENT):

```
LJG:./tcp_server1 6000 &
[1] 114
LJG:
```

- CHECK for PROCESS (background):

```
LJG:ps 114
  PID TTY      STAT   TIME COMMAND
  114 tty1     S      0:00 ./tcp_server1 6000
LJG:
```

- INSTALL **nmap**:

```
sudo apt install nmap
```

C FUNCTIONS FOR SOCKETS: EXAMPLE 1 (tcp_server1 – tcp_client1)

- nmap:
  - CONNECTIONS MONITORING
- nmap –sT 127.0.0.1 –p 6000: SCANNING PORT 6000 LOCALHOST TCP

ON WSL1

```
LJG:nmap -sT 127.0.0.1 -p 6000
Warning: Nmap may not work correctly on Windows Subsystem for Linux.
```

- DEVASC:

```
SERVER(01-tcp):./tcp_server1 6000 &
[1] 2759
SERVER(01-tcp):
```

```
SERVER(01-tcp):ps -aux | grep tcp_server
devasc      2759  0.0  0.0   2356   516 pts/0    S    20:19   0:00 ./tcp_server1 6000
```

```
SERVER(01-tcp):nmap -sT 127.0.0.1

Command 'nmap' not found, but can be installed with:

sudo snap install nmap  # version 7.91, or
sudo apt  install nmap  # version 7.80+dfsg1-2build1

See 'snap info nmap' for additional versions.

SERVER(01-tcp):
```

```
SO: nmap -sT 127.0.0.1 -p 1000
Starting Nmap 7.80 ( https://nmap.org ) at 2022-01-15 14:12 CET
Nmap scan report for localhost (127.0.0.1)
Host is up (0.00031s latency).

PORT     STATE SERVICE
1000/tcp open  cadlock

Nmap done: 1 IP address (1 host up) scanned in 0.20 seconds
SO:
```

NO PROBLEMS ON WSL2

C FUNCTIONS FOR SOCKETS: EXAMPLE 1 (tcp_server1 – tcp_client1)

- nmap:
  - CONNECTIONS MONITORING
- nmap –sT 127.0.0.1 –p 6000: SCANNING PORT 6000 LOCALHOST TCP

```
SERVER(01-tcp):nmap -sT 127.0.0.1
Starting Nmap 7.80 ( https://nmap.org ) at 2021-01-22 20:27 UTC
Client: 127.0.0.1
Nmap scan report for localhost (127.0.0.1)
Host is up (0.0014s latency).
Not shown: 997 closed ports
PORT     STATE SERVICE
631/tcp  open  ipp
6000/tcp open  X11
8081/tcp open  blackice-icecap

Nmap done: 1 IP address (1 host up) scanned in 0.74 seconds
SERVER(01-tcp):
```

- PORT 6000: X11?

C FUNCTIONS FOR SOCKETS: EXAMPLE 1 (tcp_server1 – tcp_client1)

- nmap: (again now on port 16000)

```
SERVER(01-tcp):ps -aux | grep tcp_server | grep 16000
devasc      3160  0.0  0.0   2356   576 pts/0    S    20:29   0:00 ./tcp_server1 16000
SERVER(01-tcp):nmap -sT 127.0.0.1
Starting Nmap 7.80 ( https://nmap.org ) at 2021-01-22 20:29 UTC
Client: 127.0.0.1
Nmap scan report for localhost (127.0.0.1)
Host is up (0.0012s latency).
Not shown: 996 closed ports
PORT      STATE SERVICE
631/tcp   open  ipp
6000/tcp  open  X11
8081/tcp  open  blackice-icecap
16000/tcp open  fmsas

Nmap done: 1 IP address (1 host up) scanned in 1.04 seconds
SERVER(01-tcp):
```

- PORT 16000: fmsas?

C FUNCTIONS FOR SOCKETS: EXAMPLE 1 (tcp_server1 – tcp_client1)

- nmap: (again now on port 45321)

```
SERVER(01-tcp):./tcp_server1 45321 &
[3] 3170
```

```
SERVER(01-tcp):ps -aux | grep tcp_server | grep 45321
devasc      3170  0.0  0.0   2356   584 pts/0    S    20:30   0:00 ./tcp_server1 45321
```

```
SERVER(01-tcp):nmap -sT 127.0.0.1
Starting Nmap 7.80 ( https://nmap.org ) at 2021-01-22 20:31 UTC
Nmap scan report for localhost (127.0.0.1)
Host is up (0.00057s latency).
Not shown: 996 closed ports
PORT      STATE SERVICE
631/tcp   open  ipp
6000/tcp  open  X11
8081/tcp  open  blackice-icecap
16000/tcp open  fmsas

Nmap done: 1 IP address (1 host up) scanned in 0.61 seconds
```

```
SERVER(01-tcp):nmap -sT 127.0.0.1 -p 45321
Starting Nmap 7.80 ( https://nmap.org ) at 2021-01-22 20:31 UTC
Nmap scan report for localhost (127.0.0.1)
Host is up (0.00014s latency).

PORT      STATE SERVICE
45321/tcp open  unknown

Nmap done: 1 IP address (1 host up) scanned in 0.29 seconds
Client: 127.0.0.1
SERVER(01-tcp):
```

- PORT 45321: unknown

C FUNCTIONS FOR SOCKETS: EXAMPLE 1 (tcp_server1 – tcp_client1)

- **Netstat** (PORT CONNECTION MONITORING)

```
SERVER(01-tcp):netstat -tulpn | grep tcp_server1
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
tcp        2      0 0.0.0.0:6000            0.0.0.0:*               LISTEN      2759/./tcp_server1
tcp        1      0 0.0.0.0:16000           0.0.0.0:*               LISTEN      3160/./tcp_server1
tcp        0      0 0.0.0.0:45321           0.0.0.0:*               LISTEN      3170/./tcp_server1
SERVER(01-tcp):
```

- BIND 1: LOCAL (0.0.0.0:6000) REMOTE (0.0.0.0:*) LISTEN PID=2759 ./tcp_server1
- BIND 2: LOCAL (0.0.0.0:16000) REMOTE (0.0.0.0:*) LISTEN PID=3160 ./tcp_server1
- BIND 3: LOCAL (0.0.0.0:45321) REMOTE (0.0.0.0:*) LISTEN PID=3170 ./tcp_server1

- WE HAVE 3 SERVERS READY TO ACCEPT CONNECTION REQUESTS…

```
SO: netstat -putona
(No info could be read for "-p": geteuid()=1000 but you should be root.)
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name     Timer
tcp        0      0 0.0.0.0:1000            0.0.0.0:*               LISTEN      -                    off (0.00/0/0)
SO:
```

C FUNCTIONS FOR SOCKETS: EXAMPLE 1 (tcp_server1 – tcp_client1)

- CONNECTION REQUEST (CLIENT)

```
CLIENT(01-tcp):./tcp_client1 127.0.0.1 hello 6000 &
[1] 3226
CLIENT(01-tcp):
```

```
SERVER(01-tcp):ps -aux | grep tcp
root       1559  8.4  2.9 699804 118048 tty7    Ssl+ 19:05   8:37 /usr/lib/xorg/Xorg -core :0 -seat seat0 -a
uth /var/run/lightdm/root/:0 -nolisten tcp vt7 -novtswitch
devasc     2759 47.7  0.0   2488   516 pts/0    R    20:19  13:00 ./tcp_server1 6000
devasc     3160 61.0  0.0   2488   576 pts/0    R    20:29  10:27 ./tcp_server1 16000
devasc     3170 57.2  0.0   2488   584 pts/0    R    20:30   9:03 ./tcp_server1 45321
devasc     3226 46.7  0.0   2356   516 pts/1    R    20:43   1:19 ./tcp_client1 127.0.0.1 hello 6000
devasc     3235  0.0  0.0   9032   660 pts/0    S+   20:46   0:00 grep --color=auto tcp
SERVER(01-tcp):
```

```
SERVER(01-tcp):netstat -tulpn | grep tcp_server1
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
tcp        3      0 0.0.0.0:6000            0.0.0.0:*               LISTEN      2759/./tcp_server1
tcp        1      0 0.0.0.0:16000           0.0.0.0:*               LISTEN      3160/./tcp_server1
tcp        0      0 0.0.0.0:45321           0.0.0.0:*               LISTEN      3170/./tcp_server1
SERVER(01-tcp):
```

- WHERE IS THE CONNECTION (ESTABLISHED CONNECTION?)

C FUNCTIONS FOR SOCKETS: EXAMPLE 1 (tcp_server1 – tcp_client1)

- CONNECTION REQUEST (CLIENT)

```
SERVER(01-tcp):ss -4 state established
Netid       Recv-Q      Send-Q          Local Address:Port              Peer Address:Port       Process
tcp         0           0                   127.0.0.1:x11                  127.0.0.1:39846
tcp         0           0                   127.0.0.1:39846                127.0.0.1:x11
SERVER(01-tcp):
```

```
SERVER(01-tcp):ss -4 -n state established
Netid       Recv-Q      Send-Q          Local Address:Port              Peer Address:Port       Process
tcp         0           0                   127.0.0.1:6000                 127.0.0.1:39846
tcp         0           0                   127.0.0.1:39846                127.0.0.1:6000
SERVER(01-tcp):
```

- THE CONNECTION IS:
  - 127.0.0.1:39846 (CLIENT) – 127.0.0.1:6000 (SERVER)

- KILL PROCESSES: you can kill ALL “YOUR PROCESSES”: **KILL -9 -1**

```
SERVER(01-tcp):ps -aux | grep tcp
root        1559  8.2  2.9 699804 118048 tty7    Ssl+ 19:05   8:56 /usr/lib/xorg/Xorg -core :0 -seat seat0 -a
uth /var/run/lightdm/root/:0 -nolisten tcp vt7 -novtswitch
devasc      2759 47.7  0.0   2488   516 pts/0    R    20:19  16:31 ./tcp_server1 6000
devasc      3160 57.0  0.0   2488   576 pts/0    R    20:29  13:59 ./tcp_server1 16000
devasc      3170 54.1  0.0   2488   584 pts/0    R    20:30  12:33 ./tcp_server1 45321
devasc      3226 47.5  0.0   2356   516 pts/1    R    20:43   4:51 ./tcp_client1 127.0.0.1 hello 6000
devasc      3270  0.0  0.0   9032   660 pts/0    S+   20:53   0:00 grep --color=auto tcp
SERVER(01-tcp):
```

```
SERVER(01-tcp):kill -9 2759
SERVER(01-tcp):
[1]   Killed                  ./tcp_server1 6000
SERVER(01-tcp):kill -9 3160
SERVER(01-tcp):kill -9 3170
[2]-  Killed                  ./tcp_server1 16000
SERVER(01-tcp):kill -9 3226
[3]+  Killed                  ./tcp_server1 45321
SERVER(01-tcp):
```

```
SERVER(01-tcp):ps -aux | grep tcp
root        1559  8.2  2.9 699804 118048 tty7    Rsl+ 19:05   9:07 /usr/lib/xorg/Xorg -core :0 -seat seat0 -a
uth /var/run/lightdm/root/:0 -nolisten tcp vt7 -novtswitch
devasc      3274  0.0  0.0   9032   732 pts/0    R+   20:56   0:00 grep --color=auto tcp
SERVER(01-tcp):ss -4 -n state established
Netid      Recv-Q      Send-Q             Local Address:Port             Peer Address:Port      Process
SERVER(01-tcp):netstat -tulpn | grep tcp_server1
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
```

```
SERVER(01-tcp):nmap -sT 127.0.0.1 -p 6000,16000,45321
Starting Nmap 7.80 ( https://nmap.org ) at 2021-01-22 20:57 UTC
Nmap scan report for localhost (127.0.0.1)
Host is up (0.00034s latency).

PORT      STATE  SERVICE
6000/tcp  closed X11
16000/tcp closed fmsas
45321/tcp closed unknown

Nmap done: 1 IP address (1 host up) scanned in 0.56 seconds
SERVER(01-tcp):
```

C FUNCTIONS FOR SOCKETS: EXAMPLE 1 (tcp_server1 – tcp_client1)

- IP ADDRESS?

```
SERVER(01-tcp):ifconfig
dummy0: flags=195<UP,BROADCAST,RUNNING,NOARP>  mtu 1500
        inet 192.0.2.1  netmask 255.255.255.255  broadcast 0.0.0.0
        inet6 fe80::74a7:38ff:fe2d:33a1  prefixlen 64  scopeid 0x20<link>
        ether 76:a7:38:2d:33:a1  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 11  bytes 770 (770.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.2.15  netmask 255.255.255.0  broadcast 10.0.2.255
        inet6 fe80::a00:27ff:fe90:deaf  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:90:de:af  txqueuelen 1000  (Ethernet)
        RX packets 74721  bytes 108821846 (108.8 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 14496  bytes 896267 (896.2 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

enp0s8: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.56.5  netmask 255.255.255.0  broadcast 192.168.56.255
        inet6 fe80::a00:27ff:fe2f:a32b  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:2f:a3:2b  txqueuelen 1000  (Ethernet)
        RX packets 709  bytes 76502 (76.5 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 104  bytes 16101 (16.1 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 6150  bytes 311834 (311.8 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 6150  bytes 311834 (311.8 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

SERVER(01-tcp):
```

```
SERVER(01-tcp):ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:90:de:af brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.15/24 brd 10.0.2.255 scope global dynamic enp0s3
       valid_lft 79346sec preferred_lft 79346sec
    inet6 fe80::a00:27ff:fe90:deaf/64 scope link
       valid_lft forever preferred_lft forever
3: enp0s8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:2f:a3:2b brd ff:ff:ff:ff:ff:ff
    inet 192.168.56.5/24 brd 192.168.56.255 scope global dynamic enp0s8
       valid_lft 433sec preferred_lft 433sec
    inet6 fe80::a00:27ff:fe2f:a32b/64 scope link
       valid_lft forever preferred_lft forever
4: dummy0: <BROADCAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN group default qlen 1000
    link/ether 76:a7:38:2d:33:a1 brd ff:ff:ff:ff:ff:ff
    inet 192.0.2.1/32 scope global dummy0
       valid_lft forever preferred_lft forever
    inet 192.0.2.2/32 scope global dummy0
       valid_lft forever preferred_lft forever
    inet 192.0.2.3/32 scope global dummy0
       valid_lft forever preferred_lft forever
    inet 192.0.2.4/32 scope global dummy0
       valid_lft forever preferred_lft forever
    inet 192.0.2.5/32 scope global dummy0
       valid_lft forever preferred_lft forever
    inet6 fe80::74a7:38ff:fe2d:33a1/64 scope link
       valid_lft forever preferred_lft forever
SERVER(01-tcp):
```

C FUNCTIONS FOR SOCKETS: EXAMPLE 1 (tcp_server1 – tcp_client1)

- CHECKING IP ADDRESS?

```
SERVER(01-tcp):ping -c 1 127.0.0.1
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.103 ms

--- 127.0.0.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.103/0.103/0.103/0.000 ms
SERVER(01-tcp):ping -c 2 10.0.2.15
PING 10.0.2.15 (10.0.2.15) 56(84) bytes of data.
64 bytes from 10.0.2.15: icmp_seq=1 ttl=64 time=0.107 ms
64 bytes from 10.0.2.15: icmp_seq=2 ttl=64 time=0.083 ms

--- 10.0.2.15 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1021ms
rtt min/avg/max/mdev = 0.083/0.095/0.107/0.012 ms
SERVER(01-tcp):ping -c 3 192.168.56.5
PING 192.168.56.5 (192.168.56.5) 56(84) bytes of data.
64 bytes from 192.168.56.5: icmp_seq=1 ttl=64 time=0.119 ms
64 bytes from 192.168.56.5: icmp_seq=2 ttl=64 time=0.077 ms
64 bytes from 192.168.56.5: icmp_seq=3 ttl=64 time=0.217 ms

--- 192.168.56.5 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2053ms
rtt min/avg/max/mdev = 0.077/0.137/0.217/0.058 ms
```

```
SERVER(01-tcp):ping -c 1 192.0.2.1
PING 192.0.2.1 (192.0.2.1) 56(84) bytes of data.
64 bytes from 192.0.2.1: icmp_seq=1 ttl=64 time=0.081 ms

--- 192.0.2.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.081/0.081/0.081/0.000 ms
SERVER(01-tcp):ping -c 1 192.0.2.2
PING 192.0.2.2 (192.0.2.2) 56(84) bytes of data.
64 bytes from 192.0.2.2: icmp_seq=1 ttl=64 time=0.223 ms

--- 192.0.2.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.223/0.223/0.223/0.000 ms
SERVER(01-tcp):ping -c 1 192.0.2.3
PING 192.0.2.3 (192.0.2.3) 56(84) bytes of data.
64 bytes from 192.0.2.3: icmp_seq=1 ttl=64 time=0.083 ms

--- 192.0.2.3 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.083/0.083/0.083/0.000 ms
SERVER(01-tcp):ping -c 1 192.0.2.4
PING 192.0.2.4 (192.0.2.4) 56(84) bytes of data.
64 bytes from 192.0.2.4: icmp_seq=1 ttl=64 time=0.084 ms

--- 192.0.2.4 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.084/0.084/0.084/0.000 ms
SERVER(01-tcp):ping -c 1 192.0.2.5
PING 192.0.2.5 (192.0.2.5) 56(84) bytes of data.
64 bytes from 192.0.2.5: icmp_seq=1 ttl=64 time=0.084 ms

--- 192.0.2.5 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.084/0.084/0.084/0.000 ms
SERVER(01-tcp):
```

C FUNCTIONS FOR SOCKETS: EXAMPLE 1 (tcp_server1 – tcp_client1)

- WORKING WITH IP DIFFERENT FROM LOCALHOST (127.0.0.1):

```
SERVER(01-tcp):./tcp_server1
Usage: ./tcp_server1 <port>
SERVER(01-tcp):./tcp_server1 6000 &
[1] 3305
SERVER(01-tcp):Client: 10.0.2.15

SERVER(01-tcp):
```

```
CLIENT(01-tcp):./tcp_client1 10.0.2.15 hello 6000 &
[1] 3306
CLIENT(01-tcp):
```

```
SERVER(01-tcp):ss -4 -n state established
Netid       Recv-Q      Send-Q          Local Address:Port            Peer Address:Port
tcp         0           0                   10.0.2.15:52738              10.0.2.15:6000
tcp         0           0                   10.0.2.15:6000               10.0.2.15:52738
SERVER(01-tcp):
```

C FUNCTIONS FOR SOCKETS: EXAMPLE 1 (tcp_server1 – tcp_client1)

- CAN WE HAVE MORE THAN ONE CONNECTION WITH THE SERVER?:
  - IMPOSSIBLE!
  - WHY?
    - ONCE THE SERVER ACCEPT THE CONNECTION FROM THE CLIENT THE SERVER WILL EXECUTE **handle_client()** FUNCTION
    - **handle_client()** FUNCTION HAS A WHILE(1) LOOP. SO THE PROGRAM IS ALWAYS THERE. IT WILL NEVER TRY TO ACCEPT A NEW CONNECTION REQUEST

- **NETCAT (NC)** CONNECTION TO THE SERVER

```
SERVER(01-tcp):./tcp_server1 7000 &
[1] 3344
SERVER(01-tcp):
SERVER(01-tcp):nc -vn 127.0.0.1 7000 &
[2] 3347
SERVER(01-tcp):Connection to 127.0.0.1 7000 port [tcp/*] succeeded!
Client: 127.0.0.1

SERVER(01-tcp):ss -4 -n state established
Netid      Recv-Q      Send-Q          Local Address:Port              Peer Address:Port
tcp        0           0                   127.0.0.1:56494                127.0.0.1:7000
tcp        0           0                   127.0.0.1:7000                 127.0.0.1:56494

[2]+  Stopped                 nc -vn 127.0.0.1 7000
SERVER(01-tcp):
```

```
SERVER(01-tcp):nc -vn 127.0.0.1 6000 &
[3] 3349
SERVER(01-tcp):nc: connect to 127.0.0.1 port 6000 (tcp) failed: Connection refused

[3]-  Exit 1                  nc -vn 127.0.0.1 6000
SERVER(01-tcp):
```

```
SERVER(01-tcp):ps
    PID TTY          TIME CMD
   2261 pts/0    00:00:00 bash
   3344 pts/0    00:01:47 tcp_server1
   3347 pts/0    00:00:00 nc
   3350 pts/0    00:00:00 ps
SERVER(01-tcp):
```