TELEMATIC LAB NOTES

SOCKETS

Active sockets are the ones which initiate the TCP connection, typically clients who knows the address and port of the other end (passive socket typically at the server side).

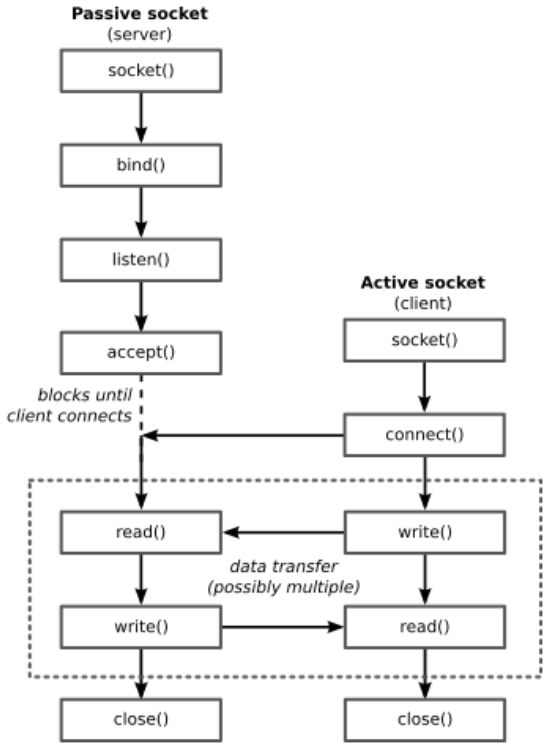
→ The syscall `socket()` creates a socket which will be specialized later into active or passive

→ When we invoke the syscall `connect()` on a socket, the socket becomes an active socket, and it will try to connect to the specified endpoint

→ TCP is in charge of initializing the sequence number

Once the connection has been created, the socket can be used to send/receive data to/from the other end, using `read() / recv()` and `write() / send()`

Passive sockets are typically server sockets and are used to wait for incoming connections and to create a new connected socket if the connection is accepted



SOCKET SYSTEM CALLS

`socket()` creates a socket which is yet not specialized into active or passive

`bind()` used by passive sockets to indicate the address (local network interface) where incoming connections are to be received

`connect()` used by active sockets to try to establish a connection with a remote socket (server)

`listen()` for passive sockets, allows to indicate we want to receive incoming connections at the address/port specified in bind().
- At the point we execute this syscall, incoming connections will be attended by the system

`accept()` for passive sockets, it accepts a connection on a socket
- If no pending connections are present on the queue, and the socket is not marked as nonblocking, accept() blocks the caller until a connection is present
- It returns an active socket (socket descriptor) that represent the client connection (this is used to send and receive data to/form the client)

`send() / write()` for active sockets (clients or sockets returned by the accept() function in the server)

→ to compile: `gcc -o file file.c`
`make clean`
`make`

→ to execute:
`./EchoClient it0xx serverhost 8xxx` (to get serverhost execute: `hostname`)
`./EchoServer 8xxx`
exit with Ctrl-C

→ check the server is running (another console): `ps aux`
→ to filter the result: `ps aux | grep helloServer`
→ to kill the process: `kill pid`
→ to make sure the server is listening to in the port: `netstat -putan | grep helloServer`
→ to check the available network interfaces in the machine: `ifconfig`
→ to log in as another host: `ssh it0xx`

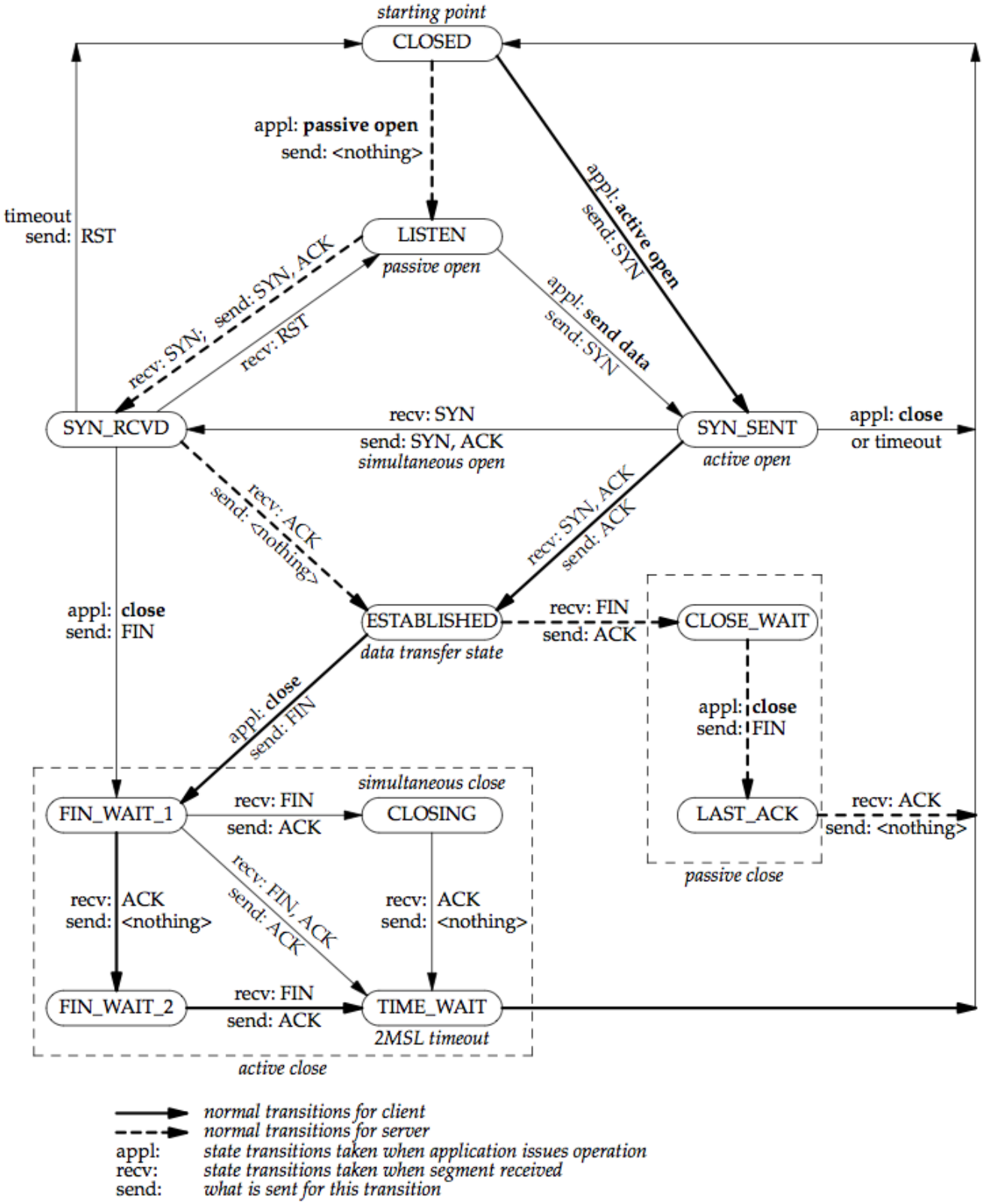
→ to observe traffic: `tcpdump`

`tcpdump -i lo port xxxx`

Flag	Abbreviation	Description
S	SYN	Synchronizes seq numbers
F	FIN	Sender has finished sending data
R	RST	Connection abort
P	PUSH	Segment requires immediate send and delivery
.		None of the above active

→ to listen to connections from different machines: `tcpdump -i eth0 port 8888`
→ to see process table: `ps x`

→ to show the different signals: `kill -l`
→ to send specified signal to specified process: `kill -signal pid`
→ to kill a process: `kill pid`
→ to see the pid of a process: `ps aux`
→ `dig help`



SEQUENTIAL SERVERS

Sequential servers are the simplest type of servers. They are generally used in services offered vía UDP (with sockets of type SOCK_DGRAM), but can also be found in TCP-based services (with sockets of type SOCK_STREAM).

1. Allocate the socket (**`socket`**)
2. Configure the socket (fill **`sockaddr_in`** structure)
3. Create the socket (**`open`**)
4. Associate the socket to a port (**`bind`**)
5. Put the socket in passive mode to prepare for incoming connections (**`listen`**)
6. Place the socket in the "waiting to handle incoming connections" state (**`accept`**), upon connection arrival, an active socket is generated
7. Read/write (**`read/write`**) or receive/send (**`recv/send`**) on the active socket
8. Close the active socket and free the connection (**`close`**)
9. Return to the waiting for connections state

1. Compile and execute

```
cd directory_where_the_files_are
gcc -o file file.c (or make clean; make)
./EchoServer 8xxx
./EchoClient serverhost 8xxx
```

→ To find the IP of a machine: `dig itxxx.lab.it.uc3m.es`

EchoServer_seq.c

- Variable that stores the passive socket: `sockfd`

- Variable that stores the active socket: `new_fd`

- The active socket is gotten when

```
new_fd = accept(sockfd, (struct sockaddr *)&their_addr, &sin_size);
```

`netstat -putan` (to verify if server is listening appropriately)

2. In another window

`tcpdump -i lo port 8xxx` (to report entire traffic passing via the local-loop interface whose origin or destination port is 8xxx)

3. Start the client a couple of times and terminate it with CTRL-D or CTRL-C. Now kill the server with CTRL-C. What do you observe with `tcpdump`? Why?

```
23:46:58.853861 IP vit125.52156 > vit125.8215: Flags [S], seq 2184986014, win 65495, options [mss 65495,sackOK,TS val 2115249841 ecr 0], length 0
23:46:58.853897 IP vit125.8215 > vit125.52156: Flags [S.], seq 2728387410, ack 2184986015, win 65483, options [mss 65495,sackOK,TS val 2115249841 ecr 2115249841], length 0
23:46:58.853927 IP vit125.52156 > vit125.8215: Flags [.], ack 1, win 65495, options [nop,nop,TS val 2115249841 ecr 2115249841], length 0
23:50:30.606891 IP vit125.52156 > vit125.8215: Flags [F.], seq 1, ack 1, win 65495, options [nop,nop,TS val 2115461594 ecr 2115249841], length 0
23:50:30.607209 IP vit125.8215 > vit125.52156: Flags [.], ack 2, win 65483, options [nop,nop,TS val 2115461594 ecr 2115461594], length 0
23:50:32.939517 IP vit125.52164 > vit125.8215: Flags [S], seq 549402076, win 65495, options [mss 65495,sackOK,TS val 2115463926 ecr 0], length 0
23:50:32.939551 IP vit125.8215 > vit125.52164: Flags [S.], seq 820146104, ack 549402077, win 65483, options [mss 65495,sackOK,TS val 2115463926 ecr 2115463926], length 0
23:50:32.939579 IP vit125.52164 > vit125.8215: Flags [.], ack 1, win 65495, options [nop,nop,TS val 2115463926 ecr 2115463926], length 0
23:50:33.526124 IP vit125.52164 > vit125.8215: Flags [F.], seq 1, ack 1, win 65495, options [nop,nop,TS val 2115464513 ecr 2115463926], length 0
23:50:33.526192 IP vit125.8215 > vit125.52164: Flags [.], ack 2, win 65482, options [nop,nop,TS val 2115464513 ecr 2115464513], length 0
23:50:36.638867 IP vit125.52166 > vit125.8215: Flags [S], seq 1380566526, win 65495, options [mss 65495,sackOK,TS val 2115467626 ecr 0], length 0
23:50:36.638901 IP vit125.8215 > vit125.52166: Flags [S.], seq 1141489074, ack 1380566527, win 65483, options [mss 65495,sackOK,TS val 2115467626 ecr 2115467626], length 0
23:50:36.638930 IP vit125.52166 > vit125.8215: Flags [.], ack 1, win 65495, options [nop,nop,TS val 2115467626 ecr 2115467626], length 0
23:50:37.242187 IP vit125.52166 > vit125.8215: Flags [F.], seq 1, ack 1, win 65495, options [nop,nop,TS val 2115468229 ecr 2115467626], length 0
23:50:37.243225 IP vit125.8215 > vit125.52166: Flags [.], ack 2, win 65483, options [nop,nop,TS val 2115468230 ecr 2115468229], length 0
23:50:41.523821 IP vit125.8215 > vit125.52166: Flags [F.], seq 1, ack 2, win 65483, options [nop,nop,TS val 2115472511 ecr 2115468229], length 0
23:50:41.523858 IP vit125.52166 > vit125.8215: Flags [.], ack 2, win 65495, options [nop,nop,TS val 2115472511 ecr 2115472511], length 0
23:50:41.523919 IP vit125.8215 > vit125.52164: Flags [F.], seq 1, ack 2, win 65482, options [nop,nop,TS val 2115472511 ecr 2115464513], length 0
23:50:41.523934 IP vit125.52164 > vit125.8215: Flags [.], ack 2, win 65495, options [nop,nop,TS val 2115472511 ecr 2115472511], length 0
23:50:41.523964 IP vit125.8215 > vit125.52156: Flags [F.], seq 1, ack 2, win 65483, options [nop,nop,TS val 2115472511 ecr 2115461594], length 0
23:50:41.523979 IP vit125.52156 > vit125.8215: Flags [.], ack 2, win 65495, options [nop,nop,TS val 2115472511 ecr 2115472511], length 0
23:50:41.524011 IP vit125.8215 > vit125.52154: Flags [F.], seq 5, ack 6, win 65479, options [nop,nop,TS val 2115472511 ecr 2115248200], length 0
23:50:41.524024 IP vit125.52154 > vit125.8215: Flags [R], seq 2398873254, win 0, length 0
```

The client sends its FIN and the server responds with ACK, but doesn't send its own FIN until que terminate the server. This is because of a bad connection termination.

To solve the problem:
The server does not send FIN because it is not closing the active socket, so we need to change the code in order to solve this problem. The socket `fd_new` is the same as the one in the TCPechod function (named `fd`). Therefore, we can either close the socket in the function when exiting the while loop or close the socket after exiting the function but inside the while(1).

```
void
TCPechod(int fd)
{
        char    buf[BUFSIZ];
        int     cc;
        while (cc = recv(fd, buf, sizeof(buf),0)) {
                if (cc < 0){
                        fprintf(stderr, "echo read: %s\n", strerror(errno));
                        exit(1);
            }

                write(0,buf,cc); /* stdin is the 0 file descriptor */

                if (send(fd, buf, cc, 0) < 0){
                        fprintf(stderr, "echo write: %s\n", strerror(errno));
                        exit(1);
            }
        }

        if (close(fd)<0){
        printf("Error al cerrar");
        }

        return;
}
```

```
while(1) {  // main accept() loop
        sin_size = sizeof their_addr;
        new_fd = accept(sockfd, (struct sockaddr *)&their_addr, &sin_size);
        if (new_fd == -1) {
                perror("accept");
                continue;
        }

        inet_ntop(their_addr.ss_family,
                  get_in_addr((struct sockaddr *)&their_addr),
                  s, sizeof s);
        printf("server: got connection from %s\n", s);

        TCPechod(new_fd);
        if(close(new_fd)<0){
          printf("Cerrado");
        }
}
```

Now, we can see that when the clients close the connection, the server responds them just after with FIN. This allows the clients to go to the TIME_WAIT state and, after some established time, close the connection.

```
vit122:~/sockets1_sequential_servers/psockets1> sudo tcpdump -i lo port 8215
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on lo, link-type EN10MB (Ethernet), capture size 262144 bytes
03:26:49.329382 IP vit122.58440 > vit122.8215: Flags [S], seq 1525093972, win 65495, options [mss 65495,sackOK,TS val 2179543687 ecr 0], length 0
03:26:49.329412 IP vit122.8215 > vit122.58440: Flags [S.], seq 1267839652, ack 1525093973, win 65483, options [mss 65495,sackOK,TS val 2179543687 ecr 2179543687], length 0
03:26:49.329466 IP vit122.58440 > vit122.8215: Flags [.], ack 1, win 65495, options [nop,nop,TS val 2179543687 ecr 2179543687], length 0
03:26:49.957676 IP vit122.58440 > vit122.8215: Flags [F.], seq 1, ack 1, win 65495, options [nop,nop,TS val 2179544315 ecr 2179543687], length 0
03:26:49.957792 IP vit122.8215 > vit122.58440: Flags [F.], seq 1, ack 2, win 65483, options [nop,nop,TS val 2179544315 ecr 2179544315], length 0
03:26:49.957816 IP vit122.58440 > vit122.8215: Flags [.], ack 2, win 65494, options [nop,nop,TS val 2179544315 ecr 2179544315], length 0
03:26:51.943756 IP vit122.58442 > vit122.8215: Flags [S], seq 1354888405, win 65495, options [mss 65495,sackOK,TS val 2179546301 ecr 0], length 0
03:26:51.943789 IP vit122.8215 > vit122.58442: Flags [S.], seq 3774018057, ack 1354888406, win 65483, options [mss 65495,sackOK,TS val 2179546301 ecr 2179546301], length 0
03:26:51.943817 IP vit122.58442 > vit122.8215: Flags [.], ack 1, win 65495, options [nop,nop,TS val 2179546301 ecr 2179546301], length 0
03:26:53.797917 IP vit122.58442 > vit122.8215: Flags [F.], seq 1, ack 1, win 65495, options [nop,nop,TS val 2179548156 ecr 2179546301], length 0
03:26:53.797977 IP vit122.8215 > vit122.58442: Flags [F.], seq 1, ack 2, win 65483, options [nop,nop,TS val 2179548156 ecr 2179548156], length 0
03:26:53.798015 IP vit122.58442 > vit122.8215: Flags [.], ack 2, win 65494, options [nop,nop,TS val 2179548156 ecr 2179548156], length 0
```

We can also confirm this behavior with `netstat`
`netstat -tn` (to observe status of client's network connections)

Before the change in the code:

```
vit122:~/sockets1_sequential_servers/psockets1> netstat -tn | grep 8215
tcp        0      0 163.117.172.173:8215    163.117.172.173:58466   CLOSE_WAIT
tcp        0      0 163.117.172.173:58466   163.117.172.173:8215    FIN_WAIT2
tcp        0      0 163.117.172.173:58468   163.117.172.173:8215    FIN_WAIT2
tcp        0      0 163.117.172.173:8215    163.117.172.173:58468   CLOSE_WAIT
```

After the change in the code:

```
vit122:~/sockets1_sequential_servers/psockets1> netstat -tn | grep 8215
tcp        0      0 163.117.172.173:58456   163.117.172.173:8215    ESTABLISHED
tcp        0      0 163.117.172.173:58454   163.117.172.173:8215    TIME_WAIT
tcp        0      0 163.117.172.173:8215    163.117.172.173:58456   ESTABLISHED
```

remote clients: to log in as another host (`ssh it0xx`)

4. Restart the (modified) server, start the client and kill the server without killing the client. What happens on re-starting the server? Why?

We initialize the server and the client, then kill the server. When killing the server, but not the client, we can see through `netstat` that the server remains in FIN_WAIT_2

```
vit126:~/Aptel/sockets1_sequential_servers/psockets1> netstat -ta | grep 8215
tcp        0      0 vit126:8215             vit127.lab.it.uc3:41350 FIN_WAIT2
```

If we know kill the client, the server stills in TIME_WAIT state, and will remain there 2MSL

```
vit126:~/Aptel/sockets1_sequential_servers/psockets1> netstat -ta | grep 8215
tcp        0      0 vit126:8215             vit126:48790            TIME_WAIT
```

If after we kill the server, we try to initialize, we will get an error because the server is still in TIME_WAIT and, during this time, the server is “being used” in that port. We cannot use that port to initialize the server until 2MSL has passed.

```
vit126:~/Aptel/sockets1_sequential_servers/psockets1> ./EchoServer_seq 8215
server: bind: Address already in use
server: bind: Address already in use
server: failed to bind
```

If we try to initialize the server after a few minutes, we will be able to do so (2MSL has

```
vit126:~/Aptel/sockets1_sequential_servers/psockets1> ./EchoServer_seq 8215
server: waiting for connections...
```

passed).

5. Kill the client, wait a few seconds then re-start the server. Now start two echo clients (on two different hosts) that connect to the same server. What happens if after starting the first client but before asking for an echo, we start the second? Both clients send information. What happens?

We start the server and the clients (different machines). We can see that in the server only the first client that we started has connected.

```
vit126:~/Aptel/sockets1_sequential_servers/psockets1> ./EchoServer_seq 8215
server: waiting for connections...
server: got connection from 163.117.172.178
```

Client 1:

```
vit127:~/Aptel/sockets1_sequential_servers/psockets1> ./EchoClient vit126 8215
client: connecting to 163.117.172.177
```

```
vit128:~/Aptel/sockets1_sequential_servers/psockets1> ./EchoClient vit126 8215
client: connecting to 163.117.172.177
```

Client 2:

If we send information from both, only client 1 receives an echo. If we look at the server, we can also see that it only sends answers to client 1.

```
vit127:~/Aptel/sockets1_sequential_servers/psockets1> ./EchoClient vit126 8215
client: connecting to 163.117.172.177
Cliente 1
Cliente 1
```

Client 2:

```
vit126:~/Aptel/sockets1_sequential_servers/psockets1> ./EchoServer_seq 8215
server: waiting for connections...
server: got connection from 163.117.172.178
Cliente 1
```

Client 1:

```
vit128:~/Aptel/sockets1_sequential_servers/psockets1> ./EchoClient vit126 8215
client: connecting to 163.117.172.177
Cliente 2+
Eoooo
```

If we look at `tcpdump`, we can see that the data being received from client 2 is being sent and received at the server, but the server is not responding this client. This is because the server is sequential, meaning that it can only answer one client at a time. The others remain in a queue waiting to be answered once the previous client closes the connection.

```
vit126:~/Aptel/sockets1_sequential_servers/psockets1> sudo tcpdump port 8215
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
12:06:31.628588 IP vit127.lab.it.uc3m.es.41384 > vit126.8215: Flags [S], seq 1552288332, win 64240, options [mss 1460,sackOK,TS val 3250407745 ecr 0], length 0
12:06:31.628666 IP vit126.8215 > vit127.lab.it.uc3m.es.41384: Flags [S.], seq 2256809140, ack 1552288333, win 65160, options [mss 1460,sackOK,TS val 3905452959 ecr 3250407745], length 0
12:06:31.628961 IP vit127.lab.it.uc3m.es.41384 > vit126.8215: Flags [.], ack 1, win 64240, options [nop,nop,TS val 3250407746 ecr 3905452959], length 0
12:06:33.101703 IP vit128.lab.it.uc3m.es.54346 > vit126.8215: Flags [S], seq 2986060086, win 64240, options [mss 1460,sackOK,TS val 3389305183 ecr 0], length 0
12:06:33.101763 IP vit126.8215 > vit128.lab.it.uc3m.es.54346: Flags [S.], seq 1979502712, ack 2986060087, win 65160, options [mss 1460,sackOK,TS val 2735973468 ecr 3389305183], length 0
12:06:33.101978 IP vit128.lab.it.uc3m.es.54346 > vit126.8215: Flags [.], ack 1, win 64240, options [nop,nop,TS val 3389305183 ecr 2735973468], length 0
12:06:36.339683 IP vit127.lab.it.uc3m.es.41384 > vit126.8215: Flags [P.], seq 1:5, ack 1, win 64240, options [nop,nop,TS val 3250412456 ecr 3905452959], length 4
12:06:36.339714 IP vit126.8215 > vit127.lab.it.uc3m.es.41384: Flags [.], ack 5, win 65156, options [nop,nop,TS val 3905457671 ecr 3250412456], length 0
12:06:36.339829 IP vit126.8215 > vit127.lab.it.uc3m.es.41384: Flags [P.], seq 1:5, ack 5, win 65156, options [nop,nop,TS val 3905457671 ecr 3250412456], length 4
12:06:36.340065 IP vit127.lab.it.uc3m.es.41384 > vit126.8215: Flags [.], ack 5, win 64236, options [nop,nop,TS val 3250412457 ecr 3905457671], length 0
12:06:38.199012 IP vit128.lab.it.uc3m.es.54346 > vit126.8215: Flags [P.], seq 1:5, ack 1, win 64240, options [nop,nop,TS val 3389310280 ecr 2735973468], length 4
12:06:38.199049 IP vit126.8215 > vit128.lab.it.uc3m.es.54346: Flags [.], ack 5, win 65156, options [nop,nop,TS val 2735978565 ecr 3389310280], length 0
```

If we now close client 1, we will see that the server answers client 2.

```
vit127:~/Aptel/sockets1_sequential_servers/psockets1> ./EchoClient vit126 8215
client: connecting to 163.117.172.177
Cliente 1
Cliente 1
^C
```

```
vit126:~/Aptel/sockets1_sequential_servers/psockets1> ./EchoServer_seq 8215
server: waiting for connections...
server: got connection from 163.117.172.178
Cliente 1
server: got connection from 163.117.172.179
Cliente 2+
Eoooo
```

```
vit128:~/Aptel/sockets1_sequential_servers/psockets1> ./EchoClient vit126 8215
client: connecting to 163.117.172.177
Cliente 2+
Eoooo
Cliente 2+
Eoooo
```

6. Now kill the second client and then kill the first. You should observe a RESET frame. Why?

If we kill client 2 and then client 1, the server will still answer client 2 after client 1 disconnects, but as client 2 has already been disconnected and closed the socket, the client doesn't know from whom is the information that it's getting so it sends a RESET frame to the server so that it can terminate the connection as there is no one listening in that port of that machine.

```
12:08:21.441452 IP vit128.lab.it.uc3m.es.54346 > vit126.8215: Flags [F.], seq 5, ack 1, win 64240, options [nop,nop,TS val 3389413522 ecr 2735978565], length 0
12:08:21.481520 IP vit126.8215 > vit128.lab.it.uc3m.es.54346: Flags [.], ack 6, win 65155, options [nop,nop,TS val 2736081847 ecr 3389413522], length 0
12:08:22.294749 IP vit127.lab.it.uc3m.es.41384 > vit126.8215: Flags [F.], seq 5, ack 5, win 64236, options [nop,nop,TS val 3250518412 ecr 3905457671], length 0
12:08:22.294938 IP vit126.8215 > vit127.lab.it.uc3m.es.41384: Flags [F.], seq 5, ack 6, win 65155, options [nop,nop,TS val 3905563626 ecr 3250518412], length 0
12:08:22.295139 IP vit126.8215 > vit128.lab.it.uc3m.es.54346: Flags [P.], seq 1:5, ack 6, win 65155, options [nop,nop,TS val 2736082661 ecr 3389413522], length 4
12:08:22.295218 IP vit127.lab.it.uc3m.es.41384 > vit126.8215: Flags [.], ack 6, win 64235, options [nop,nop,TS val 3250518412 ecr 3905563626], length 0
12:08:22.295498 IP vit128.lab.it.uc3m.es.54346 > vit126.8215: Flags [R], seq 2986060092, win 0, length 0
```

7. Change the size of the connection queue (backlog of the socket) of the server, defined by the constant BACKLOG in the file EchoServer_seq, to 1. Now compile and re-start four echo clients from another host. What do you notice in tcpdump? Kill the last client to be started. What is the effect of doing so? What explanation can you give for this?

```
#define BACKLOG 1
#define BUFSIZE
```

If we launch a fourth client, we can see that the **first** one that we launch is able to connect to the server (we can see this at the server). The next two are connected but waiting in the queue, as the server can only answer a client at a time. For the fourth client, nothing happens, and after some time, it says that it was not able to connect.

Client 1:

```
vit128:~/Aptel/sockets1_sequential_servers/psockets1> ./EchoClient vit126 8215
client: connecting to 163.117.172.177
```

Sever gets connection from client 1:

```
vit126:~/Aptel/sockets1_sequential_servers/psockets1> ./EchoServer_seq 8215
server: waiting for connections...
server: got connection from 163.117.172.179
```

Client 2:

```
vit127:~/Aptel/sockets1_sequential_servers/psockets1> ./EchoClient vit126 8215
client: connecting to 163.117.172.177
```

Client 3:

```
vit127:~/Aptel/sockets1_sequential_servers/psockets1> ./EchoClient vit126 8215
client: connecting to 163.117.172.177
```

Client 4:

```
vit128:~/Aptel/sockets1_sequential_servers/psockets1> ./EchoClient vit126 8215
```

Client 4 after some time:

```
vit128:~/Aptel/sockets1_sequential_servers/psockets1> ./EchoClient vit126 8215
client: connect: Connection timed out
client: connect: Connection refused
client: failed to connect
```

Analyzing the traffic at `tcpdump`, we can see that the connection is established for the first three clients but that the last one is not being answered (the client keeps sending S frames but after some time the server answers with a RESET frame so that the connection is terminated as the server cannot answer it).

```
vit126:~/Aptel/sockets1_sequential_servers/psockets1> sudo tcpdump port 8215
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
13:08:58.605322 IP vit128.lab.it.uc3m.es.54612 > vit126.8215: Flags [S], seq 3747847191, win 64240, options [mss 1460,sackOK,TS val 3393050686 ecr 0], length 0
13:08:58.605433 IP vit126.8215 > vit128.lab.it.uc3m.es.54612: Flags [S.], seq 1019178766, ack 3747847192, win 65160, options [mss 1460,sackOK,TS val 2739718971 ecr 3393050686], length 0
13:08:58.605688 IP vit128.lab.it.uc3m.es.54612 > vit126.8215: Flags [.], ack 1, win 64240, options [nop,nop,TS val 3393050686 ecr 2739718971], length 0
13:09:06.631481 IP vit127.lab.it.uc3m.es.42284 > vit126.8215: Flags [S], seq 1238616316, win 64240, options [mss 1460,sackOK,TS val 3254162748 ecr 0], length 0
13:09:06.631548 IP vit126.8215 > vit127.lab.it.uc3m.es.42284: Flags [S.], seq 3706187955, ack 1238616317, win 65160, options [mss 1460,sackOK,TS val 3909207962 ecr 3254162748], length 0
13:09:06.631863 IP vit127.lab.it.uc3m.es.42284 > vit126.8215: Flags [.], ack 1, win 64240, options [nop,nop,TS val 3254162749 ecr 3909207962], length 0
13:09:10.358344 IP vit127.lab.it.uc3m.es.42286 > vit126.8215: Flags [S], seq 1632401335, win 64240, options [mss 1460,sackOK,TS val 3254166475 ecr 0], length 0
13:09:10.358484 IP vit126.8215 > vit127.lab.it.uc3m.es.42286: Flags [S.], seq 117741359, ack 1632401336, win 65160, options [mss 1460,sackOK,TS val 3909211689 ecr 3254166475], length 0
13:09:10.358792 IP vit127.lab.it.uc3m.es.42286 > vit126.8215: Flags [.], ack 1, win 64240, options [nop,nop,TS val 3254166476 ecr 3909211689], length 0
13:09:13.329292 IP vit128.lab.it.uc3m.es.54614 > vit126.8215: Flags [S], seq 1070317574, win 64240, options [mss 1460,sackOK,TS val 3393065410 ecr 0], length 0
13:09:14.369975 IP vit128.lab.it.uc3m.es.54614 > vit126.8215: Flags [S], seq 1070317574, win 64240, options [mss 1460,sackOK,TS val 3393066451 ecr 0], length 0
13:09:16.418046 IP vit128.lab.it.uc3m.es.54614 > vit126.8215: Flags [S], seq 1070317574, win 64240, options [mss 1460,sackOK,TS val 3393068499 ecr 0], length 0
13:09:20.450050 IP vit128.lab.it.uc3m.es.54614 > vit126.8215: Flags [S], seq 1070317574, win 64240, options [mss 1460,sackOK,TS val 3393072531 ecr 0], length 0
13:09:28.514120 IP vit128.lab.it.uc3m.es.54614 > vit126.8215: Flags [S], seq 1070317574, win 64240, options [mss 1460,sackOK,TS val 3393080595 ecr 0], length 0
13:09:44.898077 IP vit128.lab.it.uc3m.es.54614 > vit126.8215: Flags [S], seq 1070317574, win 64240, options [mss 1460,sackOK,TS val 3393096979 ecr 0], length 0
13:10:17.153981 IP vit128.lab.it.uc3m.es.54614 > vit126.8215: Flags [S], seq 1070317574, win 64240, options [mss 1460,sackOK,TS val 3393129235 ecr 0], length 0
13:11:21.667223 IP6 2001:720:410:100a::179.50558 > vit126.8215: Flags [S], seq 559216663, win 64800, options [mss 1440,sackOK,TS val 458126061 ecr 0], length 0
13:11:21.667301 IP6 vit126.8215 > 2001:720:410:100a::179.50558: Flags [R.], seq 0, ack 559216664, win 0, length 0
```

8. Now allow to bind to a port that is already in use -by setting `SO_REUSEADDR` option (as long as there is no active listening socket already bound to it) with `setsockopt` function-. For that, uncomment the code lines in the `EchoServer_seq.c` y recompile the code. Observe the result of setting this option by revisiting question 4.

When killing the server and then the client (after changing the code), nothing from the state's changes. However, if we try to launch the server again right after killing it, we can (before we had to wait a certain time for the port to be free).

This is thanks to the modified code, that allows to use a port even though it may be in use as long as the socket that is using that port is a passive socket and not an active socket (listening)

```
int yes=1;
if    (setsockopt(sockfd,    SOL_SOCKET,    SO_REUSEADDR,    &yes,
sizeof[int])) == -1) {
     perror("setsockopt");
     exit(1);
}
```

9. Modify the size of the segments sent by the client - by setting the option TCP_MAXSEG (which is defined `in <netinet/tcp.h>`)

```
int maxseg = 88;
if (setsockopt(sockfd, SOL_TCP, TCP_MAXSEG, &maxseg,
sizeof(maxseg)) == -1) (…)
```

When checking the effect of this modified code we can see that if the client sends a bigger size than indicated, the echo answer will be cut to the specified maximum (88).

It is not possible to set values lower than 88 MSS because 48 are decided by Linux and 40 are for headers.

CONCURRENT SERVERS

Concurrent servers start a new process (using `fork()`) or thread (using `pthread_create()`) to provide a service to incoming connections.

A process has its own memory area. So if I use a process per client, each process will have its own variables with its values. However, threads share the same memory area and resources. The election of one or the other depends on the advantages and disadvantages of each of them, but both of them can be used (for example, in this assignment we use processes, but we could have used threads).

Thread Functions:
`pthread_create()` creates a new thread
`pthread_join()` makes a thread to wait for other thread completion
`pthread_mutex_lock()` allows 2+ thread to synchronize their access to a resource
`pthread_exit()` finish a threads execution

In general, TCP servers are concurrent in order to be able to handle several clients simultaneously (and UDP servers are sequential).

Child process: `fork() == 0` closes its reference to the passive socket descriptor inherited from the parent process (`msock`)

Parent process: `fork() != 0` closes the socket descriptor `ssock` returned by `accept()` (to which the client is connected) and executes accept again to wait for new connections

1. Compile

```
make clean
make
```

2. Examine the traffic whose origin/destination port is 8xxx with tcpdump, while executing two clients on two different machines

Server . `./TCPechod 8xxx`

```
it025:~/sockets2_concurrent_servers/psockets2> ./TCPechod 8134
Parent: Waiting for incomming connections at port 8134
```

Clients (different machines**) .** `./TCPecho itxxx 8xxx`

```
it025:~/sockets2_concurrent_servers/psockets2> ./TCPechod 8134
Parent: Waiting for incomming connections at port 8134
Parent: Incomming connection from 192.100.100.134 remote port 34054
Parent: Waiting for incomming connections at port 8134
Parent: Incomming connection from 192.100.100.134 remote port 34056
Parent: Waiting for incomming connections at port 8134
```

Traffic observed with `tcpdump`:

```
it025:~> sudo tcpdump -i lo port 8134
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on lo, link-type EN10MB (Ethernet), capture size 262144 bytes
15:29:02.722040 IP it025.34054 > it025.8134: Flags [S], seq 1138790142, win 65495, options [mss 65495,sackOK,TS val 1095527868 ecr 0], length 0
15:29:02.722079 IP it025.8134 > it025.34054: Flags [S.], seq 3427747893, ack 1138790143, win 65483, options [mss 65495,sackOK,TS val 1095527868 ecr 1095527868], length 0
15:29:02.722111 IP it025.34054 > it025.8134: Flags [.], ack 1, win 65495, options [nop,nop,TS val 1095527868 ecr 1095527868], length 0
15:29:16.001838 IP it025.34056 > it025.8134: Flags [S], seq 2215870364, win 65495, options [mss 65495,sackOK,TS val 1095541148 ecr 0], length 0
15:29:16.001876 IP it025.8134 > it025.34056: Flags [S.], seq 936433230, ack 2215870365, win 65483, options [mss 65495,sackOK,TS val 1095541148 ecr 1095541148], length 0
15:29:16.001910 IP it025.34056 > it025.8134: Flags [.], ack 1, win 65495, options [nop,nop,TS val 1095541148 ecr 1095541148], length 0
```

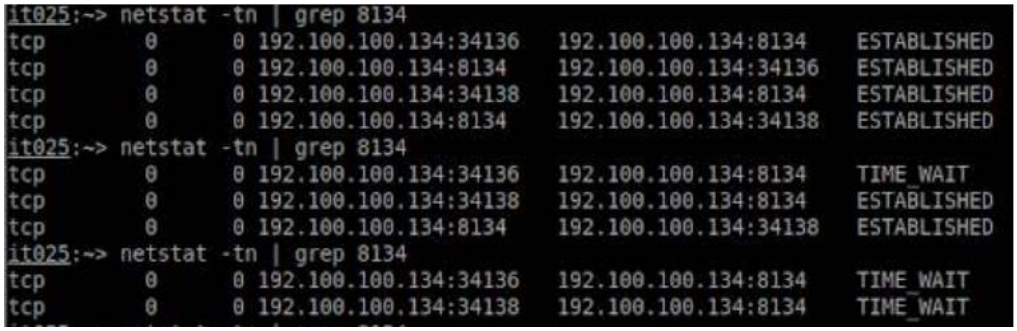
3. With this version of the server, do you observe the problem that the original sequential server had in section 3 of the first part of the sockets assignment (sequential sockets)? Why? Use the netstat command to observe the connections that are established and to see their ephemeral states and ports. For example:

The problem that we had with sequential servers was that when we closed the connection from a client, the server didn't send a FIN to the client (hence, the clients could not pass to the TIME_WAIT state). In this case, we can see in `tcpdump` that the server does send FIN to all clients and with `netstat` that they pass to TIME_WAIT and then disappear.

`tcpdump`

```
it025:~> sudo tcpdump -i lo port 8134
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on lo, link-type EN10MB (Ethernet), capture size 262144 bytes
15:41:03.800010 IP it025.34118 > it025.8134: Flags [S], seq 1694891702, win 65495, options [mss 65495,sackOK,TS val 1096248946 ecr 0], length 0
15:41:03.800048 IP it025.8134 > it025.34118: Flags [S.], seq 1517906326, ack 1694891703, win 65483, options [mss 65495,sackOK,TS val 1096248946 ecr 1096
248946], length 0
15:41:03.800082 IP it025.34118 > it025.8134: Flags [.], ack 1, win 65495, options [nop,nop,TS val 1096248946 ecr 1096248946], length 0
15:41:06.495518 IP it025.34120 > it025.8134: Flags [S], seq 3957818803, win 65495, options [mss 65495,sackOK,TS val 1096251641 ecr 0], length 0
15:41:06.495553 IP it025.8134 > it025.34120: Flags [S.], seq 2769368665, ack 3957818804, win 65483, options [mss 65495,sackOK,TS val 1096251641 ecr 1096
251641], length 0
15:41:06.495586 IP it025.34120 > it025.8134: Flags [.], ack 1, win 65495, options [nop,nop,TS val 1096251641 ecr 1096251641], length 0
15:41:13.443436 IP it025.34118 > it025.8134: Flags [F.], seq 1, ack 1, win 65495, options [nop,nop,TS val 1096258589 ecr 1096248946], length 0
15:41:13.443803 IP it025.8134 > it025.34118: Flags [F.], seq 1, ack 2, win 65483, options [nop,nop,TS val 1096258590 ecr 1096258589], length 0
15:41:13.443838 IP it025.34118 > it025.8134: Flags [.], ack 2, win 65494, options [nop,nop,TS val 1096258590 ecr 1096258590], length 0
15:41:15.234702 IP it025.34120 > it025.8134: Flags [F.], seq 1, ack 1, win 65495, options [nop,nop,TS val 1096260381 ecr 1096251641], length 0
15:41:15.234832 IP it025.8134 > it025.34120: Flags [F.], seq 1, ack 2, win 65483, options [nop,nop,TS val 1096260381 ecr 1096260381], length 0
15:41:15.234840 IP it025.34120 > it025.8134: Flags [.], ack 2, win 65494, options [nop,nop,TS val 1096260381 ecr 1096260381], length 0
```

netstat



4. With this version of the server, do you observe the problem that the original sequential server had in section 5 of the first part of the sockets assignment (sequential sockets)? Why?

When sending a message with each client at the same time, we can see that the server responds with an echo even though they are connected concurrently. Therefore, we don't have the problem from the sequential server's lab.

In the previous lab, there was only a process that could manage a client at a time. What the other clients sent remained in the buffer of the socket, waiting for the other client to close the connection.

In this case, we can see that the child processes are the ones attending the clients. The parent is waiting for more connections (if any) to create child processes and answer the clients.

SIGNALS

When a process is initiated, the course of its execution can be changed (pause, restart, cancel, etc.) or a parent process can be notified of the finalization of children processes using SIGNALS. Signals allow processes to communicate with each other, and the kernel can also communicate with them.

SIGCHLD: signal sent to the parent when one of the child processes finishes

defunct: process that has completed execution but still has an entry in the process table = zombie process
it doesn't consume system resources but occupy an entry in the process table

SIGPIPE: indicates that the connection has been broken (receives an RST)

5. Look at the code of the new server, above, and notice the instruction immediately following the creation of the socket:

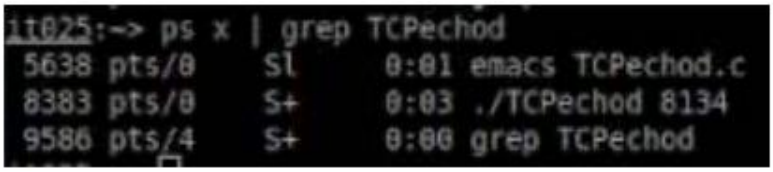
```
(void) signal(SIGCHLD, reaper);
void reaper(int sig){
	int status;
	while (wait3(&status, WNOHANG, (struct rusage *)0) > 0)
}
```

This is a handler for the SIGCHILD signal (sent by the system to indicate to the parent process that one of its child processes has terminated). The signal function registers the handler so that when the SIGCHILD signal is received the handler is executed. The handler enables the system to free all the resources that the child process was using. In the case where a parent process does not catch a terminated child process' signal, the child process remains in the zombie state. Inside the reaper function we see the system call wait3 (it is a variant of the wait call), that allows to block the server until the child process dies, but WNOHANG allows to specify that wait3 should not block waiting for a process to finish. In this way, it avoids that a wrong call blocks the server.

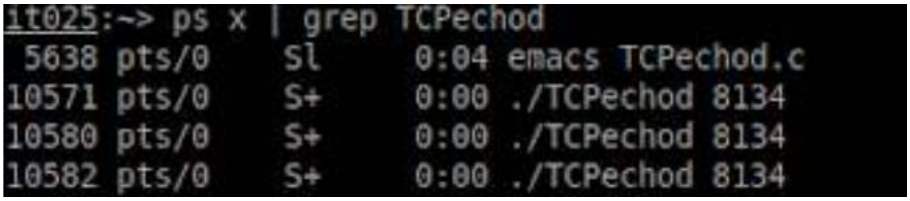
Make the following test: compile two concurrent servers, one with the call to signal and the other without it, and observe the differences between the two versions by examining the process table (with the ps x command) after a client has closed its connection. The ps command shows the processes in execution. The x option allows to show all the processes of the current user.

`ps x | grep TCPechod`
- ps allows us to see the processes that are being executed
- x is to see the processes of the actual user

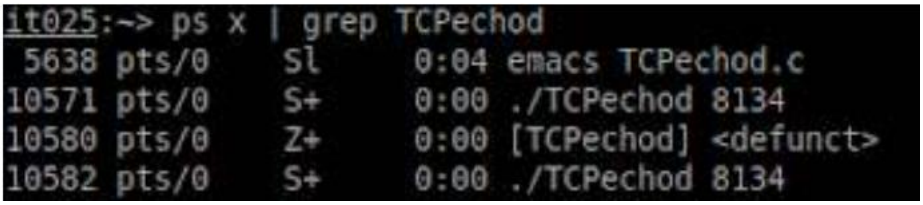
We can see the parent and the two children



We connect two clients to the server and use `ps x | grep TCPechod`



We kill the first client to see how the different process vary. We can see that one of the child processes appear as defunct, that is, the process is now a zombie process.



We kill the second client and see that the same thing happens as with the first client, both child processes are now zombie processes.

```
it025:~> ps x | grep TCPechod
 5638 pts/0    Sl     0:04 emacs TCPechod.c
10571 pts/0    S+     0:00 ./TCPechod 8134
10580 pts/0    Z+     0:00 [TCPechod] <defunct>
10582 pts/0    Z+     0:00 [TCPechod] <defunct>
```

If we now kill the server, nothing appears (gren and emacs appear but are not from the server but because we hace emacs open and grep appears when executing it on the terminal)

```
it025:~> ps x | grep TCPechod
 5638 pts/0    Sl     0:04 emacs TCPechod.c
10681 pts/4    S+     0:00 grep TCPechod
```

If we now uncomment the handler so that it works:

We see the processes at the beginning with the clients connected. We have 3 processes from the server, the parent and 2 children (emacs and grep are not relevant)

```
it025:~> ps x | grep TCPechod
 5638 pts/0    Sl     0:04 emacs TCPechod.c
10155 pts/0    S+     0:00 ./TCPechod 8134
10162 pts/0    S+     0:00 ./TCPechod 8134
10165 pts/0    S+     0:00 ./TCPechod 8134
10171 pts/4    S+     0:00 grep TCPechod
```

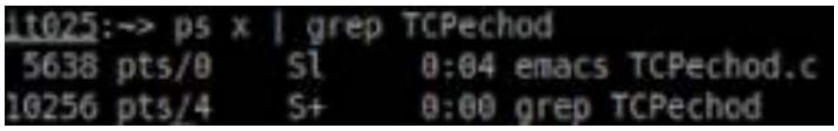
If we kill the first client, the process disappears and doesn't remain as zombie.

```
it025:~> ps x | grep TCPechod
 5638 pts/0    Sl     0:04 emacs TCPechod.c
10155 pts/0    R+     0:14 ./TCPechod 8134
10162 pts/0    S+     0:00 ./TCPechod 8134
10181 pts/4    S+     0:00 grep TCPechod
```

If we kill the second client, the same thing happens.

```
it025:~> ps x | grep TCPechod
 5638 pts/0    Sl     0:04 emacs TCPechod.c
10155 pts/0    S+     0:51 ./TCPechod 8134
10221 pts/4    S+     0:00 grep TCPechod
```

If we finally kill the server, as before, we will not see any process.



6. Another very important handler is the SIGPIPE signal, which, in the case of sockets, indicates that the connection has been broken (it receives an RST) . The SIGPIPE handler includes the code needed to treat this exception

`kill -l` (to see available signals)
`ps x | TCPechod`
`kill -signal pid`

CONCURRENT SERVERS – INPUT/OUTPUT (signal handles, poll, select)

BLOCKING AND NON-BLOCKING INPUT/OUTPUT

As is usual for input/output operations, input/output operations on sockets are generally blocking. That is, when a process performs an input/output operation on a socket, it goes into a sleeping state waiting for a condition to be satisfied before the operation can be completed. Thus:

- → If a process performs an input operation (read, recv, recvfrom,...) on a TCP socket and there is no data in the input buffer of that socket, the process goes to sleep until some data becomes available.
- → If a process performs an output operation (write, send, sendto,...) on a TCP socket - whereupon the kernel attempts to copy the output data from the application buffer to the output buffer - and there is no space in the output buffer of that socket, the process goes to sleep until sufficient space becomes available.

It is often useful to employ some mechanism that allows us to perform these operations in a non-blocking manner, enabling other tasks to be carried out instead of waiting for data, or space, to become available. In this lab session, we will look at some of the mechanisms that can be used to perform non-blocking input/output operations on sockets, in particular, polling and asynchronous mechanisms.

fcntl() FUNCTION AND THE POLLING MECHANISM

The `fcntl()` function is a control function that enables us to perform different operations on file descriptors, socket descriptors etc.

```
#include <fcntl.h>
  int fcntl(int fd, int cmd, /* int arg*/);
```

The `O_NONBLOCK` flag is used to indicate that the input/output operations on a given socket descriptor are non-blocking.

```
int flags;
// sd is the socket descriptor
if ( fcntl(sd, F_SETFL, O_NONBLOCK) < 0 )
  perror("fcntl: could not set non-blocking operations");
```

1. Using the code provided for the previous sockets lab exercise modify the client so that it can perform non-blocking input/output operations and so that it uses a polling mechanism to check the availability of data when performing read operations.

ASYNCRHONOUS MECHANISMS USING SIGNALS

The SIGIO signal is generated when the state of a socket changes, for example:
New data is available in the input buffer, so that new read operations can now be performed, or space has been freed in the output buffer, so that new write operations can now be performed.

A new client wishes to connect. In order for the SIGIO signal to be generated, the following calls to the fcntl function must be made on the corresponding socket (sd in the following):

```
if ( fcntl(sd, F_SETFL, O_ASYNC | O_NONBLOCK) < 0 ) {
    perror("fcntl error");
  }

  if ( fcntl(sd, F_SETOWN, getpid()) < 0 ) {
    perror("fcntl error");
  }
```

Asynchronous mechanisms use the occurrence of this signal to know when data is available, enabling other tasks to be carried out while no data is received.

As we saw in the previous lab session, a signal handler for a given signal is a function specifying the actions to be carried out when that signal occurs. The SIGIO signal handler will perform operations of the type: read any data available in the input buffer of the socket, send any data that the application is waiting to send via the socket, accept any new clients that wish to establish to establish a connection.

2. The code demand-accept.c is a simple example of a server that uses a `SIGIO` handler to detect when new clients connect. When the signal is generated, the handler accepts the connection of a new client, sends a message to this client and then closes the connection. Compile and test this code, executing the server (demand-accept) and a telnet client:

```
./demand-accept

telnet localhost 9999
```

CONTROLLING SEVERAL DESCRIPTORS USING THE SELECT CALL

Usually, various clients connect to a server program simultaneously; To enable our server program to handle simultaneous connections, we have two options:

1. Create a new process or thread for each new connection as we saw in the last lab session.
2. Use the `select()` function, which we now look at in detail.

The `select()` function allows several sockets to be checked at the same time. It is used to know which of the sockets being handled by our program are ready to read or write data, which have received a new connection, which have generated an exception, etc.

The prototype of the `select()` function is as follows:

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

  int select(int numfds, fd_set *readfds, fd_set *writefds,
             fd_set *exceptfds, struct timeval *timeout);
```

- → `numfds` is the value of the highest value socket descriptor plus one. Recall that a descriptor is an integer returned when a file, socket, or similar is opened. In the usual case, the values assigned to these descriptors are consecutive.
- → `readfds` is a pointer to the set of descriptors for which we wish to be notified when data is available to be read. We will also be notified when there is a new client or when a client closes the connection.
- → `writefds` is a pointer to the set of descriptors for which we wish to be notified when we can write to them without any problem. If the connection has been closed by the other side and we try to write to it, we will receive the SIGPIPE signal.
- → `exceptfds` is a pointer to the set of descriptors for which we wish to be notified when an exception has occurred.
- → `timeout` is the maximum time that we wish to wait. A NULL value indicates that the call to `select()` is to remain blocked until something occurs on one of the descriptors. A zero value indicates that we wish to know if something has occurred on one of the descriptors without blocking.

`select()` returns -1 if an error occurs (see errno), 0 if the timer expires and a number greater than zero (the number of descriptors in the descriptor set) if the call succeeds.

3. The server in smart-select.c uses `select()` to handle client connections. Five (`MAXPROCESSES`) echo server processes are started (this can be observed using ps x) before any connections are received; all of them accept connections from clients simultaneously on the same socket. Each server process has its own echo clients; new connection requests are distinguished from the arrival of new data by using the select() function and the FD_ISSET() macro. Compile the example and test it using multiple clients.

MAXPROCESSES: # children you can have
MAXCONNECTIONS: # processes that can be connected to each child

mirarlo con netstat -tnp | grep 8219

They can serve a max. of MAXPROCESSES*MAXCONNECTIONS

DNS

→ to find IP of a domain name: `nslookup website`
→ format of RR (Resource Record): <domain name> <TTL> <record type> <value>

`dig` [@server] [-p port#] [name] [type] [queryport]

name: name of the resource registry to be consulted
type: type of resource registry (ANY, A, MX…, if none=assumed type A)
queryport: +tcp / +ignore / +norecurse

`dig SOA`

```
carol@Air-de-Carol ~ % dig SOA it.uc3m.es

; <<>> DiG 9.10.6 <<>> SOA it.uc3m.es
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 4321
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
;it.uc3m.es.                    IN      SOA

;; ANSWER SECTION:
it.uc3m.es.             86400   IN      SOA     tamtam.it.uc3m.es. root.tamtam.it.uc3m.es. 2024101600 7200 3400 604800 600
  name                          class   type    primary server     zone admin.             serial     refresh retry expires min
;; Query time: 17 msec
;; SERVER: 10.37.164.214#53(10.37.164.214)
;; WHEN: Sun Oct 27 17:43:32 CET 2024
;; MSG SIZE  rcvd: 87
```

root.tamtam.it.uc3m.es = root@tamtam.it.uc3m.es
- it.uc3m.es is delegated by uc3m.es
- primary DNS server is tamtam

RDATA

MNAME: domain name of the zone's primary DNS server
RNAME: mailing address of the person responsible for the zone
- first point = @

SERIAL: serial number of the information version (32 bits)
REFRESH: time period secondary copies primary (32 bits)
RETRY: time after a refresh failure to retry (32 bits)
EXPIRE: maximum unupdated time to consider unauthorized
(if the secondary has not been able to update it will not be authorized) (32 bits)
MINIMUM: minimum TTL of any RR in the zone (32 bits)

`dig NS`

from the answer section we can see the name, TTL, class, type, Rdata of all the NS

```
;; ANSWER SECTION:
it.uc3m.es.             86400   IN      NS      varpa.it.uc3m.es.
it.uc3m.es.             86400   IN      NS      tamtam.it.uc3m.es.
it.uc3m.es.             86400   IN      NS      vorteX.uc3m.es.
```

Some nodes in the tree will have an address record:
A address: IPv4 address (32 bits)
AAAA address: IPv6 (128 bits)

```
;; ANSWER SECTION:
varpa.it.uc3m.es.    86256      IN    A     163.117.139.253
```

To identify a zone: `dig SoA` we should get an ANSWER SECTION with a SOA record

To find out if it has an A record: `dig A`

MAIL EXCHANGE (MX) domains indicate where to send mail to them

An MX type RR stores in RDATA:
PREFERENCE: 16 bits (the lower, the higher the priority)
EXCHANGE: mail server domain name

`dig MX`

```
carol@Air-de-Carol ~ % dig MX it.uc3m.es

; <<>> DiG 9.10.6 <<>> MX it.uc3m.es
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 41592
;; flags: qr rd ra; QUERY: 1, ANSWER: 5, AUTHORITY: 0, ADDITIONAL: 5

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
;it.uc3m.es.                    IN      MX

;; ANSWER SECTION:
it.uc3m.es.             86400   IN      MX      30 ASPMX3.GOOGLEMAIL.COM.
it.uc3m.es.             86400   IN      MX      10 ASPMX.L.GOOGLE.COM.
it.uc3m.es.             86400   IN      MX      20 ALT1.ASPMX.L.GOOGLE.COM.
it.uc3m.es.             86400   IN      MX      20 ALT2.ASPMX.L.GOOGLE.COM.
it.uc3m.es.             86400   IN      MX      30 ASPMX2.GOOGLEMAIL.COM.

;; ADDITIONAL SECTION:
ASPMX3.GOOGLEMAIL.COM.  264     IN      A       142.251.9.26
ASPMX.L.GOOGLE.COM.     141     IN      A       74.125.133.26
ALT2.ASPMX.L.GOOGLE.COM. 173    IN      A       142.251.9.27
ALT2.ASPMX.L.GOOGLE.COM. 173    IN      AAAA    2a00:1450:4025:c03::1b

;; Query time: 27 msec
;; SERVER: 10.37.164.214#53(10.37.164.214)
;; WHEN: Sun Oct 27 18:55:20 CET 2024
;; MSG SIZE  rcvd: 248
```

CANONICAL NAME (CNAME) indicates the canonical name of an alias consulted
- must always be another domain name (never an IP)

```
dig CNAME
;; ANSWER SECTION:
www.it.uc3m.es.          86400      IN    CNAME
     contrabajo.it.uc3m.es.
```

REVERSE RESOLUTION
Resolving an IP to a domain name

`dig -x IP`

QUERIES WITH A DNS CLIENT

1. Determine the IP address of the machine www.mec.es.

```
dig www.mec.es

;; ANSWER SECTION:
www.mec.es.          3463 IN    A     212.128.114.29
```

2. Check which machine has the IP address 193.110.128.199.

```
dig -x 193.110.128.199

;; ANSWER SECTION:
199.128.110.193.in-addr.arpa. 67288 IN  PTR  www.elmundo.es.
```

3. Find out the name and IP address of the DNS servers of the domain abc.es and say which of them is primary and which is secondary.

`dig abc.es SOA` (to identify the primary)
`dig abc.es NS` (to identify the remaining)

To find the IP address of each NS we just `dig` each of them

4. Obtain the SOA registry of the domain abc.es, first, by asking the local DNS and, second, by asking the primary server of the abc.es domain. Verify that in one case, the response is authoritative and in the other, it isn't.

`dig abc.es SOA`
`dig dnsadmin.abc.es` (local, we just get authority section)
`dig a1-229.akam.net` (we can check that it's primary because it has an answer section)

la parte 2 seria: dig NS abc.es y dig SOA abc.es, el que te salga en el SOA y en NS lo utilizas para hacer dig@primaryabc.es

5. If you had a problem with the DNS of abc.es and you had to send an e-mail to its administrator, to what address would you send it?
`dnsadmin.abc.es = dnsadmin@abc.es`
(replace first .)

6. Determine the name and IP address of the mail server of the administrator referred to in the previous question

`dig abc.es MX`

;; ANSWER SECTION: it.uc3m.es. 86399 IN SOA tamtam.it.uc3m.es. root.tamtam.it.uc3m.es. 2017041700
7200 3400 604800 600

Cuando haces un SOA:
– tamtam.it.uc3m.es is the primary server of the zone
– The zone administrator has the email root@tamtam.it.uc3m.es

– Serial number 2017041700, refresh time 7200, retry 3400, expires in 604800s and the minimum TTL is 600

7. How long will the IP address of www.vanguardia.es remain in the cache of your local DNS? Ask your local DNS for this address several times in succession. What do you observe in the TTL of the resource registry?

```
carol@MacBook-Air-de-Carol ~ % dig www.vanguardia.es

; <<>> DiG 9.10.6 <<>> www.vanguardia.es
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 34631
;; flags: qr rd ra; QUERY: 1, ANSWER: 3, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1410
;; QUESTION SECTION:
;www.vanguardia.es.             IN      A

;; ANSWER SECTION:
www.vanguardia.es.      86400   IN      CNAME   www.lavanguardia.es.
www.lavanguardia.es.    11350   IN      CNAME   ggcpvlhttpd.ggcdns.com.
ggcpvlhttpd.ggcdns.com. 180     IN      A       88.87.219.211

;; Query time: 126 msec
;; SERVER: 163.117.18.60#53(163.117.18.60)
;; WHEN: Tue Oct 29 18:19:08 CET 2024
;; MSG SIZE  rcvd: 129
```

The local DNS resolver will cache the record of the shortest TTL of the records associated with a domain. In this case, the final A record has a TTL of 180 seconds (shortest TTL).

When asking the local DNS for this address several times in succession (run `dig www.vanguardia.es` several times), the TTL starts decreasing after the TTL time has been reached (then it restarts).

8. Now ask the same to a root server (for example, J.ROOT-SERVERS.NET with IP address 192.58.128.30) and check in the reply packet that this server does not accept the recursive mode.

```
dig @192.58.128.30 www.vanguardia.es +norecurse
```

```
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 2962
;; flags: qr; QUERY: 1, ANSWER: 0, AUTHORITY: 4, ADDITIONAL: 9
```

The flag `qr` indicates that it's a query response. There's no `RA` (Recursion Available) flag, so it does not accept the recursive mode.

9. Find out how many computers are carrying out load balancing in the web server www.elpais.es. Do you always get the same ones in the same order?

`dig www.elpais.es` (if we execute it several times we see that there are 2 servers/computers but the order changes)

10. By making iterative queries, check the IP address of www.pcreview.co.uk. What are the steps you have taken?

Hint: if your DNS server has this record in the cache it is possible it does not answer with the next step but with the results. In that case, ask directly to a root server in this way: dig +norecurse www.pcreview.co.uk @A.ROOT-SERVERS.NET, and continue.

```
dig @A.ROOT-SERVERS.NET www.pcreview.co.uk +norecurse
```

```
; <<>> DiG 9.10.6 <<>> @A.ROOT-SERVERS.NET www.pcreview.co.uk +norecurse
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 47267
;; flags: qr; QUERY: 1, ANSWER: 0, AUTHORITY: 8, ADDITIONAL: 17

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;www.pcreview.co.uk.            IN      A

;; AUTHORITY SECTION:
uk.                     172800  IN      NS      dns1.nic.uk.
uk.                     172800  IN      NS      dns3.nic.uk.
uk.                     172800  IN      NS      nsc.nic.uk.
uk.                     172800  IN      NS      nsa.nic.uk.
uk.                     172800  IN      NS      nsd.nic.uk.
uk.                     172800  IN      NS      dns2.nic.uk.
uk.                     172800  IN      NS      dns4.nic.uk.
uk.                     172800  IN      NS      nsb.nic.uk.

;; ADDITIONAL SECTION:
dns1.nic.uk.            172800  IN      A       213.248.216.1
dns1.nic.uk.            172800  IN      AAAA    2a01:618:400::1
dns3.nic.uk.            172800  IN      A       213.248.220.1
dns3.nic.uk.            172800  IN      AAAA    2a01:618:404::1
nsc.nic.uk.             172800  IN      A       156.154.102.3
nsc.nic.uk.             172800  IN      AAAA    2610:a1:1009::3
nsa.nic.uk.             172800  IN      A       156.154.100.3
nsa.nic.uk.             172800  IN      AAAA    2001:502:ad09::3
nsd.nic.uk.             172800  IN      A       156.154.103.3
nsd.nic.uk.             172800  IN      AAAA    2610:a1:1010::3
dns2.nic.uk.            172800  IN      A       103.49.80.1
dns2.nic.uk.            172800  IN      AAAA    2401:fd80:400::1
dns4.nic.uk.            172800  IN      A       43.230.48.1
dns4.nic.uk.            172800  IN      AAAA    2401:fd80:404::1
nsb.nic.uk.             172800  IN      A       156.154.101.3
nsb.nic.uk.             172800  IN      AAAA    2001:502:2eda::3
```

We look in the authority section to see which nameservers are authoritative for uk. Then, we take one of the uk. nameservers and query it for www.pcreview.co.uk, again with recursion disabled.

```
dig @nsc.nic.uk www.pcreview.co.uk +norecurse
```

```
; <<>> DiG 9.10.6 <<>> @nsc.nic.uk www.pcreview.co.uk +norecurse
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 45098
;; flags: qr; QUERY: 1, ANSWER: 0, AUTHORITY: 2, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1232
;; QUESTION SECTION:
;www.pcreview.co.uk.            IN      A

;; AUTHORITY SECTION:
pcreview.co.uk.         172800  IN      NS      marjory.ns.cloudflare.com.
pcreview.co.uk.         172800  IN      NS      cleo.ns.cloudflare.com.

;; Query time: 32 msec
;; SERVER: 156.154.102.3#53(156.154.102.3)
;; WHEN: Tue Oct 29 23:16:08 CET 2024
;; MSG SIZE  rcvd: 105
```

This uk. nameserver returns the nameservers for co.uk (authority section). Now we choose one of the co.uk nameservers and query it for www.pcreview.co.uk.

```
dig @marjory.ns.cloudfare.com www.pcreview.co.uk +norecurse
```

```
carol@MacBook-Air-de-Carol ~ % dig @marjory.ns.cloudflare.com www.pcreview.co.uk +norecurse


; <<>> DiG 9.10.6 <<>> @marjory.ns.cloudflare.com www.pcreview.co.uk +norecurse
; (3 servers found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 29044
;; flags: qr aa; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1232
;; QUESTION SECTION:
;www.pcreview.co.uk.            IN      A

;; ANSWER SECTION:
www.pcreview.co.uk.     300     IN      A       104.21.6.187
www.pcreview.co.uk.     300     IN      A       172.67.135.45

;; Query time: 13 msec
;; SERVER: 108.162.192.193#53(108.162.192.193)
;; WHEN: Tue Oct 29 23:21:54 CET 2024
;; MSG SIZE  rcvd: 79
```

In the answer section we can see the IP address for www.pcreview.co.uk, which in this case has two IP addresses. The flag aa confirms that this response comes from an authoritative server for the domain.

Following the same steps (iterative queries), do you obtain the IP address of www.bbc.co.uk?

```
dig @A.ROOT-SERVERS.NET www.bbc.co.uk +norecurse
```

```
carol@MacBook-Air-de-Carol ~ % dig @A.ROOT-SERVERS.NET www.bbc.co.uk +norecurse

; <<>> DiG 9.10.6 <<>> @A.ROOT-SERVERS.NET www.bbc.co.uk +norecurse
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 25327
;; flags: qr; QUERY: 1, ANSWER: 0, AUTHORITY: 8, ADDITIONAL: 17

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;www.bbc.co.uk.                 IN      A

;; AUTHORITY SECTION:
uk.                     172800  IN      NS      dns1.nic.uk.
uk.                     172800  IN      NS      dns3.nic.uk.
uk.                     172800  IN      NS      nsc.nic.uk.
uk.                     172800  IN      NS      nsa.nic.uk.
uk.                     172800  IN      NS      nsd.nic.uk.
uk.                     172800  IN      NS      dns2.nic.uk.
uk.                     172800  IN      NS      dns4.nic.uk.
uk.                     172800  IN      NS      nsb.nic.uk.

;; ADDITIONAL SECTION:
dns1.nic.uk.            172800  IN      A       213.248.216.1
dns1.nic.uk.            172800  IN      AAAA    2a01:618:400::1
dns3.nic.uk.            172800  IN      A       213.248.220.1
dns3.nic.uk.            172800  IN      AAAA    2a01:618:404::1
nsc.nic.uk.             172800  IN      A       156.154.102.3
nsc.nic.uk.             172800  IN      AAAA    2610:a1:1009::3
nsa.nic.uk.             172800  IN      A       156.154.100.3
nsa.nic.uk.             172800  IN      AAAA    2001:502:ad09::3
nsd.nic.uk.             172800  IN      A       156.154.103.3
nsd.nic.uk.             172800  IN      AAAA    2610:a1:1010::3
dns2.nic.uk.            172800  IN      A       103.49.80.1
dns2.nic.uk.            172800  IN      AAAA    2401:fd80:400::1
dns4.nic.uk.            172800  IN      A       43.230.48.1
dns4.nic.uk.            172800  IN      AAAA    2401:fd80:404::1
nsb.nic.uk.             172800  IN      A       156.154.101.3
nsb.nic.uk.             172800  IN      AAAA    2001:502:2eda::3

;; Query time: 41 msec
;; SERVER: 198.41.0.4#53(198.41.0.4)
;; WHEN: Tue Oct 29 23:25:32 CET 2024
;; MSG SIZE  rcvd: 546
```

dig @dns1.nic.uk www.bbc.co.uk +norecurse

```
carol@MacBook-Air-de-Carol ~ %
dig @dns1.nic.uk www.bbc.co.uk +norecurse


; <<>> DiG 9.10.6 <<>> @dns1.nic.uk www.bbc.co.uk +norecurse
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 30125
;; flags: qr; QUERY: 1, ANSWER: 0, AUTHORITY: 8, ADDITIONAL: 9

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;www.bbc.co.uk.                 IN      A

;; AUTHORITY SECTION:
bbc.co.uk.              172800  IN      NS      dns0.bbc.co.uk.
bbc.co.uk.              172800  IN      NS      dns0.bbc.com.
bbc.co.uk.              172800  IN      NS      dns1.bbc.co.uk.
bbc.co.uk.              172800  IN      NS      dns1.bbc.com.
bbc.co.uk.              172800  IN      NS      ddns0.bbc.co.uk.
bbc.co.uk.              172800  IN      NS      ddns0.bbc.com.
bbc.co.uk.              172800  IN      NS      ddns1.bbc.co.uk.
bbc.co.uk.              172800  IN      NS      ddns1.bbc.com.

;; ADDITIONAL SECTION:
dns0.bbc.co.uk.         172800  IN      A       198.51.44.9
dns1.bbc.co.uk.         172800  IN      A       198.51.45.9
ddns0.bbc.co.uk.        172800  IN      A       148.163.199.1
ddns1.bbc.co.uk.        172800  IN      A       148.163.199.65
dns0.bbc.co.uk.         172800  IN      AAAA    2620:4d:4000:6259:7:9:0:1
dns1.bbc.co.uk.         172800  IN      AAAA    2a00:edc0:6259:7:9::2
ddns0.bbc.co.uk.        172800  IN      AAAA    2607:f740:e04e::1
ddns1.bbc.co.uk.        172800  IN      AAAA    2607:f740:e04e:4::1

;; Query time: 35 msec
;; SERVER: 213.248.216.1#53(213.248.216.1)
;; WHEN: Tue Oct 29 23:26:51 CET 2024
;; MSG SIZE  rcvd: 381
```

dig @dns0.bbc.co.uk www.bbc.co.uk +norecurse

```
carol@MacBook-Air-de-Carol ~ % dig @dns0.bbc.co.uk www.bbc.co.uk +norecurse

; <<>> DiG 9.10.6 <<>> @dns0.bbc.co.uk www.bbc.co.uk +norecurse
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 6128
;; flags: qr aa; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1232
;; QUESTION SECTION:
;www.bbc.co.uk.                 IN      A

;; ANSWER SECTION:
www.bbc.co.uk.          28800   IN      CNAME   www.bbc.co.uk.pri.bbc.co.uk.

;; Query time: 24 msec
;; SERVER: 198.51.44.9#53(198.51.44.9)
;; WHEN: Tue Oct 29 23:27:56 CET 2024
;; MSG SIZE  rcvd: 74
```

The response indicates that www.bbc.co.uk is an alias (or CNAME record) for another domain. The CNAME record points to www.bbc.co.uk.pri.bbc.co.uk., meaning that to find the actual IP address, you would have to resolve this new name (www.bbc.co.uk.pri.bbc.co.uk).

`dig @dns0.bbc.co.uk www.bbc.co.uk.pri.bbc.co.uk +norecurse`

Again: `dig @dns0.bbc.co.uk www.bbc.map.fastly.net. uk +norecurse`

This means that query was refused, so dns0.bbc.co.uk is not configured to resolve the domain.

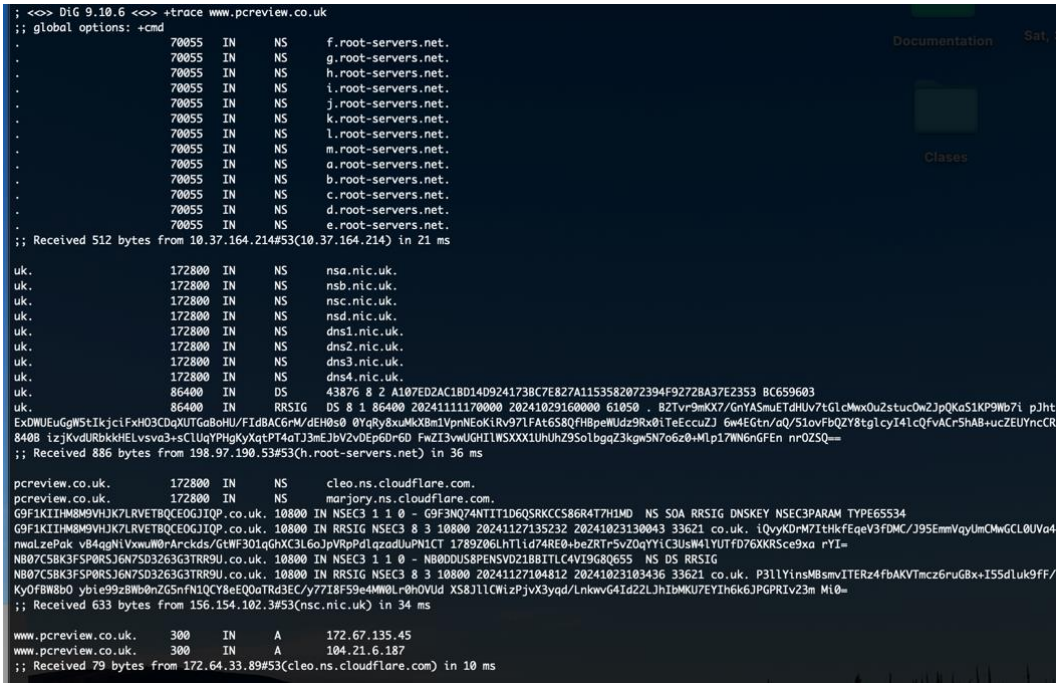
Try this command using a public (google) DNS server: `dig @8.8.8.8 bbc.map.fastly.net +norecurse`

```
carol@MacBook-Air-de-Carol ~ % dig @8.8.8.8 bbc.map.fastly.net +norecurse


; <<>> DiG 9.10.6 <<>> @8.8.8.8 bbc.map.fastly.net +norecurse
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 57890
;; flags: qr ra; QUERY: 1, ANSWER: 4, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
;bbc.map.fastly.net.            IN      A

;; ANSWER SECTION:
bbc.map.fastly.net.     26      IN      A       151.101.192.81
bbc.map.fastly.net.     26      IN      A       151.101.64.81
bbc.map.fastly.net.     26      IN      A       151.101.128.81
bbc.map.fastly.net.     26      IN      A       151.101.0.81

;; Query time: 15 msec
;; SERVER: 8.8.8.8#53(8.8.8.8)
;; WHEN: Tue Oct 29 23:35:31 CET 2024
;; MSG SIZE  rcvd: 111
```

The IP addresses listed above are the final resolved addresses for www.bbc.co.uk (can be verified with dig command)

11. The same can be done with the +trace option of dig. Check the result of doing so.

`dig +trace www.pcreview.co.uk`



12. Using the information available via DNS determine the computer or computers (name and IP address) that act as mail servers for the domain gmail.com.

`dig gmail.com MX`

```
carol@MacBook-Air-de-Carol ~ %
dig gmail.com MX


; <<>> DiG 9.10.6 <<>> gmail.com MX
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 12762
;; flags: qr rd ra; QUERY: 1, ANSWER: 5, AUTHORITY: 0, ADDITIONAL: 4

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
;gmail.com.                     IN      MX

;; ANSWER SECTION:
gmail.com.              1870    IN      MX      5 gmail-smtp-in.l.google.com.
gmail.com.              1870    IN      MX      40 alt4.gmail-smtp-in.l.google.com.
gmail.com.              1870    IN      MX      10 alt1.gmail-smtp-in.l.google.com.
gmail.com.              1870    IN      MX      30 alt3.gmail-smtp-in.l.google.com.
gmail.com.              1870    IN      MX      20 alt2.gmail-smtp-in.l.google.com.

;; ADDITIONAL SECTION:
alt3.gmail-smtp-in.l.google.com. 189 IN A       142.251.1.26
alt2.gmail-smtp-in.l.google.com. 245 IN A       142.251.9.27
gmail-smtp-in.l.google.com. 234 IN      A       74.125.71.27

;; Query time: 14 msec
;; SERVER: 10.37.164.214#53(10.37.164.214)
;; WHEN: Tue Oct 29 23:42:45 CET 2024
;; MSG SIZE  rcvd: 209
```

Now we should do `dig` for each nameserver to obtain their IP addresses.

13. What would you need to do to obtain all the resource registries of the the zone lab.it.uc3m.es?

First, we need to identify the authoritative name server.
`dig lab.it.uc3m.es NS`

```
carol@MacBook-Air-de-Carol ~ % dig lab.it.uc3m.es NS

; <<>> DiG 9.10.6 <<>> lab.it.uc3m.es NS
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 25354
;; flags: qr rd ra; QUERY: 1, ANSWER: 4, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
;lab.it.uc3m.es.                        IN      NS

;; ANSWER SECTION:
lab.it.uc3m.es.         60      IN      NS      tamtam.it.uc3m.es.
lab.it.uc3m.es.         60      IN      NS      lm000.lab.it.uc3m.es.
lab.it.uc3m.es.         60      IN      NS      it000.lab.it.uc3m.es.
lab.it.uc3m.es.         60      IN      NS      vortex.uc3m.es.

;; Query time: 78 msec
;; SERVER: 10.37.164.214#53(10.37.164.214)
;; WHEN: Tue Oct 29 23:46:34 CET 2024
;; MSG SIZE  rcvd: 125
```

Now, to obtain all the resource registries of each name server, we will have to do the following **for each** nameserver (it will return all available records for the domain).

```
dig @tamtam.it.uc3m.es lab.it.uc3m.es ANY
```

14. Find out which of the following domain names are a zone:
google.jobs, primevideo.com, inf.uc3m.es, it.uc3m.es.

To identify if a domain name is a zone, we have to `dig SOA` and look at the answer section. If we get a SOA record in the answer section then it is a zone.

```
dig SOA Google.jobs

;; ANSWER SECTION:
google.jobs.        60    IN    SOA  ns1.google.com.         dns-
admin.google.com. 690945392 900 900 1800 60
```

This indicates that google.jobs is a zone as it has a SOA record.

```
dig SOA primevideo.com

;; ANSWER SECTION:
primevideo.com.     900 IN  SOA ns-1933.awsdns-49.co.uk.
awsdns-hostmaster.amazon.com. 1 7200 900 1209600 900

dig SOA inf.uc3m.es
```

There is no answer section. Instead, the SOA record is in the authority section which suggests that inf.uc3m.es is not a separate zone, but likely a part of uc3m.es

To find the zone to which it belongs: `dig SOA uc3m.es` (which has a SOA record in its answer section).

```
dig SOA it.uc3m.es

;; ANSWER SECTION:
it.uc3m.es.         86400     IN    SOA  tamtam.it.uc3m.es.
root.tamtam.it.uc3m.es. 2024101600 7200 3400 604800 600
```

15. Identify the DNS servers, mail servers, the domain administrator's address, and identify the secondary-primary copy times, expiration time, as well as the minimum TTL for the domains it.uc3m.es and csic.es.

To identify the DNS Servers (NS Records):

```
dig it.uc3m.es
dig csic.es
```

To identify the Mail Servers (MX Records):

```
dig it.uc3m.es MX
dig csci.es MX
```

To identify the Domain Administrator's Address (SOA Record):

```
dig it.uc3m.es SOA
dig csic.es SOA
```

To identify the secondary-primary copy times, expiration time, minimum TTL also use SOA.

CREATING A DOMAIN IN A NAMED DNS SERVER

```
/usr/dist/sbin/named -c ./named.conf -f
```

File named.conf

1. Options section is used to specify configuration aspects such as the port on which named will listen (on startup in user mode it cannot be 53, we will need to specify a non-reserved port such as 10053) and the locaction of certain files that it will need to write during operation (which we will set to be in our account).

2. The zone section is used to define one or more DNS zones that this server is responsible for.

```
# file $HOME/rroo/named/named.conf
options {
 port 10053;
directory "$HOME/rroo/named/";
 pid-file "$HOME/rroo/named/named.pid";
};
zone "0.0.127.in-addr.arpa" IN {
 type master;
 file "$HOME/rroo/named/127.0.0";
};
zone "midominio.privado" IN {
 type master;
 file "$HOME/rroo/named/midominio.privado";
};
```

$HOME/rroo/named/127.0.0

```
$TTL 86400
@       IN      SOA       ns.midominio.privado.
mail.midominio.privado. (
                200403031 ; Numero de Serie: Fecha+Numero
                28800 ; Tiempo de Refresco
                7200 ; Tiempo de Reintento
                604080 ; Caducidad de la informacion
                86400) ; TTL para clientes
        NS      ns.midominio.privado.
1       PTR     localhost.
```

$HOME/rroo/named/midominio.privado

```
$TTL 86400
@       IN      SOA         ns.midominio.privado.
mail.midominio.privado. (
                            200403031
                            28800
                            7200
                            604800
                            86400 )
                NS          ns.midominio.privado.
                MX          10 mail.midominio.privado.
ns              A           192.168.123.1
mail            A           192.168.123.2
www             CNAME       ns
```

1. Copy the above files, start a named on your computer and use the dig tool to check that it is working properly.

2. Stop the named server and modify the configuration files so that it also serves the domain necessary for inverse resolution.

Edit `named.conf` (We have to replace $HOME with the full path of our home directory)

```
# file /usr/lab/alum/0475095/Escritorio/Telematic Applications/dns/named/named.conf
options {
  port 10053;
  directory "/usr/lab/alum/0475095/Escritorio/Telematic Applications/dns/named/";
  pid-file "/usr/lab/alum/0475095/Escritorio/Telematic Applications/dns/named/named.pid";
};
zone "0.0.127.in-addr.arpa" IN {
  type master;
  file "/usr/lab/alum/0475095/Escritorio/Telematic Applications/dns/named/127.0.0";
};
zone "midominio.privado" IN {
       type master;
       file "/usr/lab/alum/0475095/Escritorio/Telematic Applications/dns/named/midominio.privado";
};
```

Create zone files `127.0.0` and `midominio.privado` (they are probably already created once downloading the files from git)

Start the `named` Server:
- navigate to the directory containing `named.conf`
- Run named with the provided command:

```
/usr/dist/sbin/named -c ./named.conf -f
```

Test DNS with dig:

```
dig @localhost -p 10053 ns.midominio.privado
```

This should return the DNS records for your domain `midominio.privado`

MODIFY THE REVERSE RESOLUTION (Note the order in which the new IP address is written)

To add reverse resolution, we need to modify `named.conf` to include a new zone:

```
# file /usr/lab/alum/0475095/Escritorio/Telematic Applications/dns/named/named.conf
options {
  port 10053;
  directory "/usr/lab/alum/0475095/Escritorio/Telematic Applications/dns/named/";
  pid-file "/usr/lab/alum/0475095/Escritorio/Telematic Applications/dns/named/named.pid";
};
zone "0.0.127.in-addr.arpa" IN {
  type master;
  file "/usr/lab/alum/0475095/Escritorio/Telematic Applications/dns/named/127.0.0";
};
zone "midominio.privado" IN {
        type master;
        file "/usr/lab/alum/0475095/Escritorio/Telematic Applications/dns/named/midominio.privado";
};
zone "122.168.192.in-addr.arpa" IN {
    type master;
    file "/usr/lab/alum/0475095/Escritorio/Telematic Applications/dns/named/192.168.122";
};
```

192.168.122

```
$TTL 86400
@       IN      SOA     ns.midominio.privado.      mail.midominio.privado. (
                        200403031 ; Numero de Serie: Fecha+Numero
                        28800 ; Tiempo de Refresco
                        7200 ; Tiempo de Reintento
                        604080 ; Caducidad de la informacion
                        86400) ; TTL para clientes
        NS      ns.midominio.privado.
1       PTR     ns
2       PTR     mail
```

Restart named Server as before and test reverse DNS resolution with dig.

```
/usr/dist/sbin/named -c ./named.conf -f
dig @localhost -p 10053 -x 192.168.122
```

ESTADOS EN EL ps x

El proceso está en primer plano en un terminal, lo que significa que puede recibir señales como SIGINT al presionar Ctrl+C.
Ejemplos comunes de STAT en ps
R: El proceso está en ejecución.
S+: Un proceso en espera en primer plano.
Sl: Un proceso en espera que es multihilo.
Z: Proceso zombie.