MEMORIA P1

MONTERROSO BARCO, ALBERTO

Código fuente: https://github.com/Albermonte/LRSS/tree/master/P1.1

NOC

```c
// ping_noc_serv.c

#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <signal.h>
#include <string.h>
#include <unistd.h>

// File descriptor del socket a crear
int sock;

void sig_handler(int signum)
{
    if (signum == SIGINT || signum == SIGTSTP)
    {
        printf("Closing %d\n", signum);
        // Cerrar el socket cuando el usuario presiona Ctrl+C o Ctrl+Z
        close(sock);
        exit(EXIT_SUCCESS);
    }
}

void main(int argc, char *argv[])
{
    printf("## NOC SERVER ## \n\n");
    // Comprobamos que haya el número de parámetros requerido
    if (argc < 2)
    {
        printf("Missing param PORT.\n");
        exit(EXIT_FAILURE);
    }

    // Estructuras para guardar las direcciones de cliente y servidor
    struct sockaddr_in server_address, client_address;
    socklen_t addrlen = sizeof(client_address);
    // Convertir la entrada de texto a entero para que pueda ser leido por
htons
    int port = atoi(argv[1]);
    char data_received[1024];
```

```
    // Limpiamos buffer para que no aparezcan caracteres extraños
    memset(data_received, 0, sizeof(data_received));

    printf("Listening on %d\n", port);

    // Creamos el socket
    /**
     * socket(domain, type, protocol)
     * domain: PF_LOCAL, AF_INET (IPv4), AF_INET6 (IPv6)
     * type: SOCK_STREAM (TCP), SOCK_DGRAM (UDP)
     * protocol: 0 (IP), more at /etc/protocols
     */
    printf("Creating socket\n");
    sock = socket(AF_INET, SOCK_DGRAM, 0);
    if (sock < 0)
    {
        printf("Error creating socket\n");
        exit(EXIT_FAILURE);
    }

    // Definimos las señales para Ctrl+C y Ctrl+Z después de crear el socket
para así poder cerrarlo luego
    signal(SIGINT, sig_handler);  // handle ctrl+c
    signal(SIGTSTP, sig_handler); // handle ctrl+z

    server_address.sin_family = AF_INET;                // IPv4 address family
    server_address.sin_addr.s_addr = htonl(INADDR_ANY); // Give the local
machine address
    server_address.sin_port = htons(port);              // Port at which
server listens to the requests

    // Bindeando socket al puerto especificado por el usuario
    printf("Binding socket\n");
    if (bind(sock, (struct sockaddr *)&server_address, sizeof(server_address))
< 0)
    {
        printf("Binding socket to port %d failed\n", port);
        exit(EXIT_FAILURE);
    }

    // Esperar al mensaje desde el cliente
    printf("Waiting for msg...\n");
    while (1)
    {
        // Dentro del while para poder quedarnos escuchando siempre
        if (recvfrom(sock, data_received, 1024, 0, (struct sockaddr
*)&client_address, &addrlen) < 0)
        {
```

```
            printf("Error receiving data from client\n");
            exit(EXIT_FAILURE);
        }

        printf("Msg received..\tSending msg back\n");

        // Des comentar la siguiente línea para ver si la verificación del
cliente funciona
        // strcpy(data_received, "b");

        // Una vez recibido el mensaje lo reenviaremos al cliente para hacer
la parte "pong" del "ping"
        sendto(sock, data_received, strlen(data_received), 0, (struct sockaddr
*)&client_address, addrlen);

        memset(data_received, 0, sizeof(data_received)); // Clear buffer
    }
    // El servidor se quedará escuchando permanentemente hasta que se produzca
la interrupción por teclado
}

// Fuentes:
// https://tutorialspoint.dev/language/cpp/socket-programming-cc
// https://www.ibm.com/docs/en/zos/2.4.0?topic=sockets-using-sendto-recvfrom-
calls
// https://www.tutorialspoint.com/c_standard_library/c_function_signal.htm
// https://www.gnu.org/software/libc/manual/html_node/Local-Socket-
Example.html
// https://www.gnu.org/software/libc/manual/html_node/Socket-Addresses.html
// https://www.gnu.org/software/libc/manual/html_node/Inet-Example.html
// https://github.com/dheeraj-2000/task2_computernetworks/tree/master/udp
```

Antes de recibir nada:

```
albermonte@DESKTOP-HEU9CTS:/mnt/c/Users/Alberto/OneDrive - Universidad de Alcala/LRSS/LRSS/P1.1/NOC$ ./ping_noc_serv.out 3000
## NOC SERVER ##

Listening on 3000
Creating socket
Binding socket
Waiting for msg...
```

Después del ping:

```
albermonte@DESKTOP-HEU9CTS:/mnt/c/Users/Alberto/OneDrive - Universidad de Alcala/LRSS/LRSS/P1.1/NOC$ ./ping_noc_serv.out 3000
## NOC SERVER ##

Listening on 3000
Creating socket
Binding socket
Waiting for msg...
Msg received..  Sending msg back
Msg received..  Sending msg back
Msg received..  Sending msg back
```

```c
// ping_noc.c

#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <signal.h>
#include <string.h>
#include <unistd.h>
#include <sys/time.h>
#include <netdb.h>
#include <ctype.h>

// Definimos el número de mensajes a mandar para poder cambiarlo fácilmente
#define MSG_AMOUNT 3

int sock;

void sig_handler(int signum)
{
    if (signum == SIGINT || signum == SIGTSTP)
    {
        printf("Closing %d\n", signum);
        close(sock);
        exit(EXIT_SUCCESS);
    }
}

int isNumber(char s[])
{
    for (int i = 0; s[i] != '\0'; i++)
    {
        if (isdigit(s[i]) == 0)
        {
            return 0;
        }
    }
    return 1;
}

void main(int argc, char *argv[])
{
    printf("## NOC CLIENT ## \n\n");
    // Comprobamos que haya el número de parámetros requerido
    if (argc < 3)
    {
        printf("Missing params.\n");
```

```
        exit(EXIT_FAILURE);
    }

    // Comprobamos que el puerto sea un número, si no lo es podría producir un
"Core dump" al asignarlo al socket
    if (!isNumber(argv[2]))
    {
        printf("Port \"%s\" not numeric, usage: ./ping_noc.out host port\n",
argv[2]);
        exit(EXIT_FAILURE);
    }

    printf("Pinging to: %s, %s \n", argv[1], argv[2]);
    // Esperamos a la acción del usuario para crear el socket y lanzar el ping
    printf("Press ENTER key to Continue\n");
    getchar();

    // Estructuras para guardar las direcciones de cliente y servidor
    struct sockaddr_in server_address, client_address;
    socklen_t addrlen = sizeof(server_address);
    // Convertir la entrada de texto a entero para que pueda ser leido por
htons
    int port = atoi(argv[2]);
    char data_received[1024];
    // Limpiamos buffer para que no aparezcan carácteres extraños
    memset(data_received, 0, sizeof(data_received)); // Clear buffer

    printf("Creating socket\n");
    // Creamos el socket
    /**
     * socket(domain, type, protocol)
     * domain: PF_LOCAL, AF_INET (IPv4), AF_INET6 (IPv6)
     * type: SOCK_STREAM (TCP), SOCK_DGRAM (UDP)
     * protocol: 0 (IP), more at /etc/protocols
     */
    // La diferencia con el socket TCP está en el parámetro SOCK_DGRAM que se
usa en UDP, mientras en TCP se usa SOCK_STREAM
    sock = socket(AF_INET, SOCK_DGRAM, 0);
    if (sock < 0)
    {
        printf("Error creating socket\n");
        exit(EXIT_FAILURE);
    }

    // Definismos las señales para Ctrl+C y Ctrl+Z después de crear el socket
para así poder cerrarlo luego
    signal(SIGINT, sig_handler);  // handle ctrl+c
    signal(SIGTSTP, sig_handler); // handle ctrl+z
```

```
    client_address.sin_family = AF_INET;                // IPv4 address family
    client_address.sin_addr.s_addr = htonl(INADDR_ANY); // Give the local
machine address

    printf("Binding socket\n\n");
    if (bind(sock, (struct sockaddr *)&client_address, sizeof(client_address))
< 0)
    {
        printf("Binding socket failed\n");
        exit(EXIT_FAILURE);
    }

    server_address.sin_family = AF_INET;   // IPv4 address family
    server_address.sin_port = htons(port); // Port number at which the server
is listning

    // Convertimos el host en una IP
    struct hostent *hostname = gethostbyname(argv[1]);
    if (!hostname)
    {
        printf("Can't resolve hostname %s\n", argv[1]);
    }
    // Copiamos esta IP a la dirección del servidor
    bcopy(hostname->h_addr, &server_address.sin_addr, hostname->h_length);

    int msg_count = 0;
    double time[MSG_AMOUNT];
    struct timeval start, stop;
    double msecs = 0;
    double total_time = 0;
    char *msg = "a";

    printf("Pinging to %s with a total of %ld bytes:\n\n", argv[1],
strlen(msg) * MSG_AMOUNT);
    while (msg_count < MSG_AMOUNT)
    {
        // 1 char = 1 byte
        printf("Sending ping %d of %ld bytes...\t", msg_count, strlen(msg));

        // Guardamos el tiempo actual
        gettimeofday(&start, NULL);
        // Enviamos el paquete udp al servidor
        sendto(sock, msg, strlen(msg), 0, (struct sockaddr *)&server_address,
addrlen);

        // Quedamos a la espera de la respuesta desde el servidor
```

```
        if (recvfrom(sock, data_received, 1024, 0, (struct sockaddr
*)&server_address, &addrlen) < 0)
        {
            printf("Error receiving data from server\n");
            exit(EXIT_FAILURE);
        }
        // Guardamos el tiempo actual, otra vez
        gettimeofday(&stop, NULL);

        // Como mejora, comprobamos que el mensaje enviado y recibido sea el
mismo. Si no lo es, algo raro estaría pasando.
        // MITM, servidor equivocado, algun error en el servidor...
        if (strcmp(msg, data_received))
        {
            printf("Data received from server is not consistent\n");
            exit(EXIT_FAILURE);
        }

        // Cálculos para obtener los ms que ha pasado desde que enviamos hasta
que recibimos el paquete
        msecs = (double)(stop.tv_usec - start.tv_usec) / 1000;
        printf("...ping %d finished, took: %f ms\n", msg_count, msecs);
        total_time += msecs;
        msg_count++;
        memset(data_received, 0, sizeof(data_received)); // Clear buffer
    }

    // Cálculos finales para mostrar al usuario algunas estadísticas extra
    printf("\n## Total Time for %d pings: %f ms, Mean: %f ms ##\n\n",
msg_count, total_time, (total_time / msg_count));
    // Finalmente cerramos el socket
    close(sock);
}

// Fuentes:
// https://www.tutorialspoint.com/c_standard_library/c_function_signal.htm
// https://github.com/dheeraj-2000/task2_computernetworks/tree/master/udp
// https://www.codegrepper.com/code-examples/c/check+if+string+is+number+c
```

Antes del ping:

```
albermonte@DESKTOP-HEU9CTS:/mnt/c/Users/Alberto/OneDrive - Universidad de Alcala/LRSS/LRSS/P1.1/NOC$ ./ping_noc.out localhost 3000
## NOC CLIENT ##

Pinging to: localhost, 3000
Press ENTER key to Continue
```

Después del ping:

```
albermonte@DESKTOP-HEU9CTS:/mnt/c/Users/Alberto/OneDrive - Universidad de Alcala/LRSS/LRSS/P1.1/NOC$ ./ping_noc.out localhost 3000
## NOC CLIENT ##

Pinging to: localhost, 3000
Press ENTER key to Continue

Creating socket
Binding socket

Pinging to localhost with a total of 3 bytes:

Sending ping 0 of 1 bytes...    ...ping 0 finished, took: 0.155000 ms
Sending ping 1 of 1 bytes...    ...ping 1 finished, took: 0.088000 ms
Sending ping 2 of 1 bytes...    ...ping 2 finished, took: 0.080000 ms

## Total Time for 3 pings: 0.323000 ms, Mean: 0.107667 ms ##
```

```python
# ping_noc_serv.py

import sys
import signal
import socket

# Comprobamos que haya el número de parámetros requerido
if len(sys.argv) < 2:
    print("Missing param PORT.\n")
    quit()

# Convertir la entrada de texto a entero
PORT = int(sys.argv[1])
print(f"Running server on Port: {PORT}")

print("Creating Socket")
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

def sig_handler(signum, frame):
    print("\nClosing socket...")
    sock.close()
    quit()


# Definismos la señal para Ctrl+C después de crear el socket para así poder cerrarlo luego
signal.signal(signal.SIGINT, sig_handler)

print("Binding address and port")
# Bindeamos el host y puerto a la dirección del servidor
server_address = ('localhost', PORT)
sock.bind(server_address)

# while True para quedarnos a la escucha de nuevos clientes cuando el actual se desconecte
```

```python
while True:
    print("Waiting for connection")
    # Esperamos a un mensaje
    data, client_address = sock.recvfrom(1024)

    if data:
        print(f"Receiving data from {client_address}... Sending back")
        # Una vez recivido lo enviamos de vuelta
        sock.sendto(data, client_address)
```

Antes del ping:

```
albermonte@DESKTOP-HEU9CTS:/mnt/c/Users/Alberto/OneDrive - Universidad de Alcala/LRSS/LRSS/P1.1/NOC$ python3 ping_noc_serv.py 3000
Running server on Port: 3000
Creating Socket
Binding address and port
Waiting for connection
```

Después del ping:

```
albermonte@DESKTOP-HEU9CTS:/mnt/c/Users/Alberto/OneDrive - Universidad de Alcala/LRSS/LRSS/P1.1/NOC$ python3 ping_noc_serv.py 3000
Running server on Port: 3000
Creating Socket
Binding address and port
Waiting for connection
Receiving data from ('127.0.0.1', 42799)... Sending back
Waiting for connection
Receiving data from ('127.0.0.1', 42799)... Sending back
Waiting for connection
Receiving data from ('127.0.0.1', 42799)... Sending back
Waiting for connection
```

```python
# ping_noc.py

import sys
import socket
import signal
import time

# Comprobamos que haya el número de parámetros requerido
if len(sys.argv) < 3:
    print("Missing params.\n")
    quit()

# Comprobamos que el puerto sea un número, si no lo hacemos produciría un
error
if not sys.argv[2].isnumeric():
    print(
        f"Port \"{sys.argv[2]}\" not numeric, usage: python3 ping_noc.py host
port\n")
    quit()

HOST = sys.argv[1]
PORT = int(sys.argv[2])
```

```python
# Definimos el número de mensajes a mandar para poder cambiarlo fácilmente
MSG_AMOUNT = 3


print(f"Running client on {HOST}:{PORT}\n")
# Creamos el socket UDP usando SOCK_DGRAM
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)


def sig_handler(signum, frame):
    print("\nClosing socket...")
    sock.close()
    quit()


signal.signal(signal.SIGINT, sig_handler)

i = 0
total_time = 0
msg = b"a"

print(
    f"Pinging to {HOST} with a total of {sys.getsizeof(msg) * MSG_AMOUNT}
bytes:\n")

while i < MSG_AMOUNT:
    print(f"Sending ping {i} of {sys.getsizeof(msg)} bytes...\t", end='')
    # Guardamos el tiempo actual
    start = time.perf_counter_ns()
    # Enviamos el paquete udp al servidor
    sock.sendto(msg, (HOST, PORT))
    # Quedamos a la espera de la respuesta desde el servidor
    data = sock.recvfrom(1024)
    # Si no recibimos nada, salimos del bucle
    if not data:
        break
    # Guardamos el tiempo actual, otra vez
    end = time.perf_counter_ns()
    # Cálculos para obtener los ms que ha pasado desde que enviamos hasta que
recibimos el paquete
    elapsed_ms = (end - start)/1e6
    total_time += elapsed_ms
    print(f"...ping {i} finished, took {elapsed_ms} ms")
    i += 1

# Cálculos finales para mostrar al usuario algunas estadísticas extra
print(
    f"\n## Total Time for {i} pings: {total_time} ms, Mean: {total_time / i}
ms ##\n\n")
```

```
# Finalmente cerramos el socket
sock.close()
```

Después del ping:

```
albermonte@DESKTOP-HEU9CTS:/mnt/c/Users/Alberto/OneDrive - Universidad de Alcala/LRSS/LRSS/P1.1/NOC$ python3 ping_noc.py localhost 3000
Running client on localhost:3000

Pinging to localhost with a total of 102 bytes:

Sending ping 0 of 34 bytes...   ...ping 0 finished, took 0.312224 ms
Sending ping 1 of 34 bytes...   ...ping 1 finished, took 0.243656 ms
Sending ping 2 of 34 bytes...   ...ping 2 finished, took 0.126867 ms

## Total Time for 3 pings: 0.682747 ms, Mean: 0.22758233333333333 ms ##
```

OC

Para no repetir comentarios solo he vuelto a comentar las partes que cambian.

```c
// ping_oc_serv.c

#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <signal.h>
#include <string.h>
#include <unistd.h>

int sock;

void sig_handler(int signum)
{
    if (signum == SIGINT || signum == SIGTSTP)
    {
        printf("Closing %d\n", signum);
        close(sock);
        exit(EXIT_SUCCESS);
    }
}

void main(int argc, char *argv[])
{
    printf("## OC SERVER ## \n\n");
    if (argc < 2)
    {
        printf("Missing param PORT.\n");
        exit(EXIT_FAILURE);
    }

    struct sockaddr_in server_address;
```

```
    socklen_t addrlen = sizeof(server_address);
    int port = atoi(argv[1]);
    char data_received[1024];
    memset(data_received, 0, sizeof(data_received)); // Clear buffer

    printf("Listening on %d\n", port);

    // Create the socket.
    /**
     * socket(domain, type, protocol)
     * domain: PF_LOCAL, AF_INET (IPv4), AF_INET6 (IPv6)
     * type: SOCK_STREAM (TCP), SOCK_DGRAM (UDP)
     * protocol: 0 (IP), more at /etc/protocols
     */
    printf("Creating socket\n");
    sock = socket(AF_INET, SOCK_STREAM, 0);
    if (sock < 0)
    {
        printf("Error creating socket\n");
        exit(EXIT_FAILURE);
    }

    signal(SIGINT, sig_handler);  // handle ctrl+c
    signal(SIGTSTP, sig_handler); // handle ctrl+z

    server_address.sin_family = AF_INET;                // IPv4 address family
    server_address.sin_addr.s_addr = htonl(INADDR_ANY); // Give the local
machine address
    server_address.sin_port = htons(port);              // Port at which
server listens to the requests

    // Binding socket to specified port
    printf("Binding socket\n");
    if (bind(sock, (struct sockaddr *)&server_address, sizeof(server_address))
< 0)
    {
        printf("Binding socket to port %d failed\n", port);
        exit(EXIT_FAILURE);
    }

    // En esta ocasión quedamos a la escucha para más tarde aceptar la
conexión desde el servidor
    // El segundo argumento define el número máximo de conexiones que habrá en
cola
    // En este caso este argumento nos da igual ya que solo habrá 1
    if ((listen(sock, 3)) < 0)
    {
        printf("Listen failed\n");
```

```
        exit(EXIT_FAILURE);
    }

    printf("Waiting for msg...\n");
    int new_socket, valread;

    while (1)
    {
        // Esperamos a la conexión desde el cliente, dentro de un while para
poder escuchar a más
        // clientes una vez el actual se desconecte
        if ((new_socket = accept(sock, (struct sockaddr *)&server_address,
(socklen_t *)&addrlen)) < 0)
        {
            printf("Eror %d", new_socket);
            exit(EXIT_FAILURE);
        }
        while (1)
        {
            // Esperamos a los parques enviados por el cliente
            valread = read(new_socket, data_received, 1024);
            // Si no obtenemos nada salimos del bucle
            if (!valread)
                break;

            printf("Msg received..\tSending msg back\n");

            // Des comentar la siguiente línea para ver si la verificación del
cliente funciona
            // strcpy(data_received, "b");

            // Reenviamos lo recibido de nuevo hacia el cliente ("pong")
            send(new_socket, data_received, strlen(data_received), 0);
            memset(data_received, 0, sizeof(data_received)); // Clear buffer
        }
    }

    close(sock);
}

// Fuentes:
// https://www.gnu.org/software/libc/manual/html_node/Local-Socket-
Example.html
// https://ubidots.com/blog/how-to-simulate-a-tcpudp-client-using-netcat/
// https://www.tutorialspoint.com/c_standard_library/c_function_signal.htm
// https://www.gnu.org/software/libc/manual/html_node/Socket-Addresses.html
// https://www.gnu.org/software/libc/manual/html_node/Inet-Example.html
```

```
// https://github.com/dheeraj-2000/task2_computernetworks/blob/master/Multithreaded_TCP_Server_Client/server.cpp
```

Antes del ping:

```
albermonte@DESKTOP-HEU9CTS:/mnt/c/Users/Alberto/OneDrive - Universidad de Alcala/LRSS/LRSS/P1.1/OC$ ./ping_oc_serv.out 3000
## OC SERVER ##

Listening on 3000
Creating socket
Binding socket
Waiting for msg...
```

Después del ping:

```
albermonte@DESKTOP-HEU9CTS:/mnt/c/Users/Alberto/OneDrive - Universidad de Alcala/LRSS/LRSS/P1.1/OC$ ./ping_oc_serv.out 3000
## OC SERVER ##

Listening on 3000
Creating socket
Binding socket
Waiting for msg...
Msg received..  Sending msg back
Msg received..  Sending msg back
Msg received..  Sending msg back
```

```c
// ping_oc.c

#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <signal.h>
#include <string.h>
#include <unistd.h>
#include <sys/time.h>
#include <netdb.h>
#include <ctype.h>

#define MSG_AMOUNT 3

int sock;

void sig_handler(int signum)
{
    if (signum == SIGINT || signum == SIGTSTP)
    {
        printf("Closing %d\n", signum);
        close(sock);
        exit(EXIT_SUCCESS);
    }
}
```

```c
int isNumber(char s[])
{
    for (int i = 0; s[i] != '\0'; i++)
    {
        if (isdigit(s[i]) == 0)
        {
            return 0;
        }
    }
    return 1;
}

void main(int argc, char *argv[])
{
    printf("## OC CLIENT ## \n\n");
    if (argc < 3)
    {
        printf("Missing params.\n");
        exit(EXIT_FAILURE);
    }

    if (!isNumber(argv[2]))
    {
        printf("Port \"%s\" not numeric, usage: ./ping_noc.out host port\n",
argv[2]);
        exit(EXIT_FAILURE);
    }

    printf("Pinging to: %s, %s \n", argv[1], argv[2]);
    printf("Press ENTER key to Continue\n");
    getchar();

    struct sockaddr_in server_address;
    socklen_t addrlen = sizeof(server_address);
    int port = atoi(argv[2]);
    char data_received[1024];
    memset(data_received, 0, sizeof(data_received)); // Clear buffer

    printf("Creating socket\n");
    sock = socket(AF_INET, SOCK_STREAM, 0);
    if (sock < 0)
    {
        printf("Error creating socket\n");
        exit(EXIT_FAILURE);
    }

    signal(SIGINT, sig_handler);  // handle ctrl+c
```

```
    signal(SIGTSTP, sig_handler); // handle ctrl+z

    server_address.sin_family = AF_INET;   // IPv4 address family
    server_address.sin_port = htons(port); // Port number at which the server
is listning

    // Get IP from host
    struct hostent *hostname = gethostbyname(argv[1]);
    if (!hostname)
    {
        printf("Can't resolve hostname %s\n", argv[1]);
    }
    bcopy(hostname->h_addr, &server_address.sin_addr, hostname->h_length);

    // Al ser TCP tenemos que conectarnos primero al servidor antes de enviar
paquetes
    if (connect(sock, (struct sockaddr *)&server_address, addrlen) < 0)
    {
        printf("Connection failed\n");
        exit(EXIT_FAILURE);
    }
    printf("Connection successful\n\n");

    int msg_count = 0,
        valread;
    double time[MSG_AMOUNT];
    struct timeval start, stop;
    double msecs = 0;
    double total_time = 0;
    char *msg = "a";

    printf("Pinging to %s with a total of %ld bytes:\n\n", argv[1],
strlen(msg) * MSG_AMOUNT);
    while (msg_count < MSG_AMOUNT)
    {
        // 1 char = 1 byte
        printf("Sending ping %d of %ld bytes...\t", msg_count, strlen(msg));
        gettimeofday(&start, NULL);
        // Una vez conectados, enviamos el mensaje
        send(sock, msg, strlen(msg), 0);
        // Esperamos a la respuesta del servidor
        valread = read(sock, data_received, 1024);
        gettimeofday(&stop, NULL);

        if (strcmp(msg, data_received))
        {
            printf("Data received from server is not consistent\n");
            exit(EXIT_FAILURE);
```

```
        }

        msecs = (double)(stop.tv_usec - start.tv_usec) / 1000;
        printf("...ping %d finished, took: %f ms\n", msg_count, msecs);
        total_time += msecs;
        msg_count++;
        memset(data_received, 0, sizeof(data_received)); // Clear buffer
    }

    printf("\n## Total Time for %d pings: %f ms, Mean: %f ms ##\n\n",
msg_count, total_time, (total_time / msg_count));

    close(sock);
}

// Fuentes:
// https://www.tutorialspoint.com/c_standard_library/c_function_signal.htm
// https://github.com/dheeraj-
2000/task2_computernetworks/blob/master/Multithreaded_TCP_Server_Client/server
.cpp
```

Antes del ping:

```
albermonte@DESKTOP-HEU9CTS:/mnt/c/Users/Alberto/OneDrive - Universidad de Alcala/LRSS/LRSS/P1.1/OC$ ./ping_oc.out localhost 3000
## OC CLIENT ##

Pinging to: localhost, 3000
Press ENTER key to Continue
```

Después del ping:

```
albermonte@DESKTOP-HEU9CTS:/mnt/c/Users/Alberto/OneDrive - Universidad de Alcala/LRSS/LRSS/P1.1/OC$ ./ping_oc.out localhost 3000
## OC CLIENT ##

Pinging to: localhost, 3000
Press ENTER key to Continue

Creating socket
Connection successful

Pinging to localhost with a total of 3 bytes:

Sending ping 0 of 1 bytes...    ...ping 0 finished, took: 0.056000 ms
Sending ping 1 of 1 bytes...    ...ping 1 finished, took: 0.182000 ms
Sending ping 2 of 1 bytes...    ...ping 2 finished, took: 0.067000 ms

## Total Time for 3 pings: 0.305000 ms, Mean: 0.101667 ms ##
```

```
# ping_oc_serv.py

import sys
import signal
import socket

if len(sys.argv) < 2:
    print("Missing param PORT.\n")
```

```
    quit()

PORT = int(sys.argv[1])
print(f"Running server on Port: {PORT}")

print("Creating Socket")
# SOCK_STREAM ya que es TCP
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)


def sig_handler(signum, frame):
    print("\nClosing socket...")
    sock.close()
    quit()


signal.signal(signal.SIGINT, sig_handler)

print("Binding address and port")
server_address = ('localhost', PORT)
sock.bind(server_address)

print("Listening...")
# En esta ocasión quedamos a la escucha para más tarde aceptar la conexión
desde el servidor
sock.listen()

# while True para quedarnos a la escucha de nuevos clientes cuando el actual
se desconecte
while True:
    print("Waiting for connection")
    # Esperamos a la conexión desde el cliente, dentro de un while para poder
escuchar a más
    # clientes una vez el actual se desconecte
    connection, client_address = sock.accept()

    print(f"Connectiong from {client_address}")

    while True:
        # Esperamos a los parques enviados por el cliente
        data = connection.recv(1024)
        # Si no obtenemos nada salimos del bucle
        if not data:
            break
        print("Msg received... Sending back")
        # Reenviamos lo recibido de nuevo hacia el cliente ("pong")
        connection.sendall(data)
```

Antes del ping:

```
albermonte@DESKTOP-HEU9CTS:/mnt/c/Users/Alberto/OneDrive - Universidad de Alcala/LRSS/LRSS/P1.1/OC$ python3 ping_oc_serv.py 3000
Running server on Port: 3000
Creating Socket
Binding address and port
Listening...
Waiting for connection
```

Después del ping:

```
albermonte@DESKTOP-HEU9CTS:/mnt/c/Users/Alberto/OneDrive - Universidad de Alcala/LRSS/LRSS/P1.1/OC$ python3 ping_oc_serv.py 3000
Running server on Port: 3000
Creating Socket
Binding address and port
Listening...
Waiting for connection
Connectiong from ('127.0.0.1', 35378)
Msg received... Sending back
Msg received... Sending back
Msg received... Sending back
Waiting for connection
```

```python
# ping_oc.py

import signal
import sys
import socket
import time

if len(sys.argv) < 3:
    print("Missing params.\n")
    quit()

if not sys.argv[2].isnumeric():
    print(
        f"Port \"{sys.argv[2]}\" not numeric, usage: python3 ping_oc.py host
port\n")
    quit()

HOST = sys.argv[1]
PORT = int(sys.argv[2])
MSG_AMOUNT = 3

print(f"Running client on {HOST}:{PORT}\n")

# SOCK_STREAM ya que es TCP
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# Al ser TCP tenemos que conectarnos primero al servidor antes de enviar
paquetes
sock.connect((HOST, PORT))


def sig_handler(signum, frame):
```

```
    print("\nClosing socket...")
    sock.close()
    quit()


signal.signal(signal.SIGINT, sig_handler)

i = 0
total_time = 0
msg = b"a"

print(
    f"Pinging to {HOST} with a total of {sys.getsizeof(msg) * MSG_AMOUNT}
bytes:\n")

while i < MSG_AMOUNT:
    print(f"Sending ping {i} of {sys.getsizeof(msg)} bytes...\t", end='')
    start = time.perf_counter_ns()
    # Una vez conectados, enviamos el mensaje
    sock.sendall(b"a")
    # Esperamos a la respuesta del servidor
    data = sock.recv(1024)
    # Si no obtenemos nada salimos del bucle
    if not data:
        break
    end = time.perf_counter_ns()
    elapsed_ms = (end - start)/1e6
    total_time += elapsed_ms
    print(f"...ping {i} finished, took {elapsed_ms} ms")
    i += 1

sock.close()

print(
    f"\n## Total Time for {i} pings: {total_time} ms, Mean: {total_time / i}
ms ##\n\n")
```

Después del ping:

```
albermonte@DESKTOP-HEU9CTS:/mnt/c/Users/Alberto/OneDrive - Universidad de Alcala/LRSS/LRSS/P1.1/OC$ python3 ping_oc.py localhost 3000
Running client on localhost:3000

Pinging to localhost with a total of 102 bytes:

Sending ping 0 of 34 bytes...   ...ping 0 finished, took 0.221875 ms
Sending ping 1 of 34 bytes...   ...ping 1 finished, took 0.093775 ms
Sending ping 2 of 34 bytes...   ...ping 2 finished, took 0.091932 ms

## Total Time for 3 pings: 0.407582 ms, Mean: 0.13586066666666666 ms ##
```