# Memoria P1

Monterroso Barco, Alberto

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

## NOC

// 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

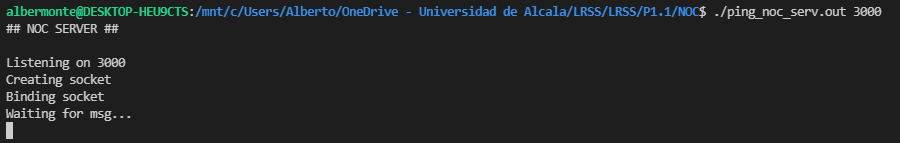
// https://www.tutorialspoint.com/c\_standard\_library/c\_function\_signal.htm

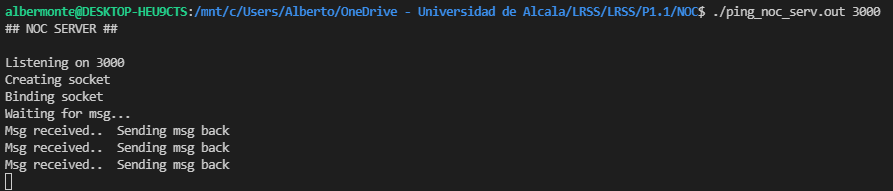
// https://www.gnu.org/software/libc/manual/html\_node/Local-Socket-Example.html

// 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:

Después del ping:

// 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);

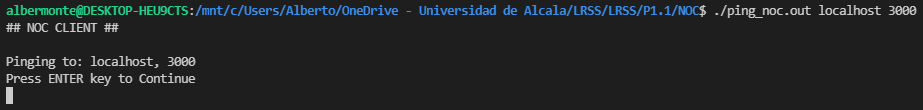
}

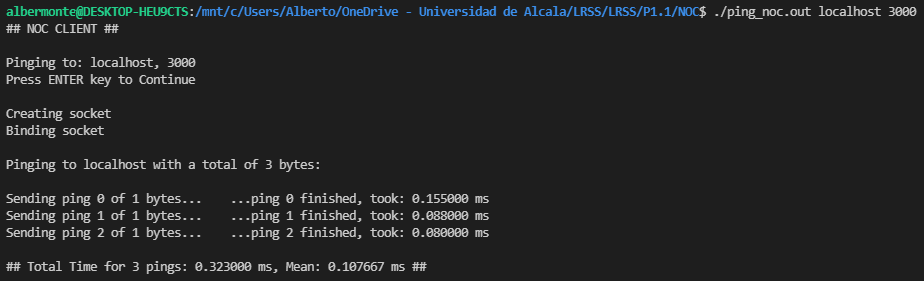
// Fuentes:

// 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:

Después del ping:

# 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

while True:

    print("Waiting for connection")

    # Esperamos a un mensaje

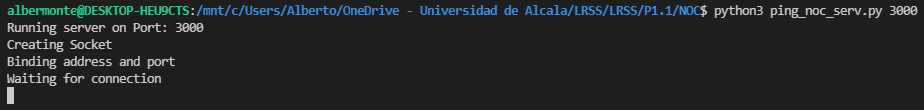
    data, client\_address = sock.recvfrom(1024)

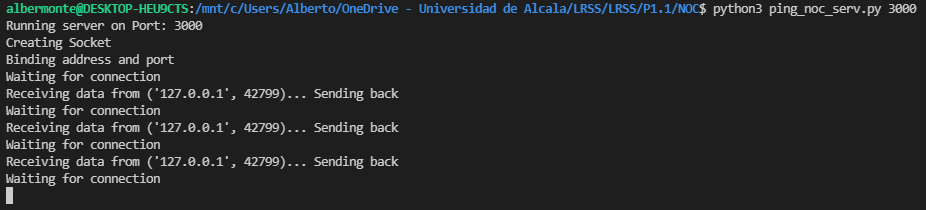
    if data:

        print(f"Receiving data from {client\_address}... Sending back")

        # Una vez recivido lo enviamos de vuelta

        sock.sendto(data, client\_address)

Antes del ping: 

Después del ping: 

# 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])

# 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

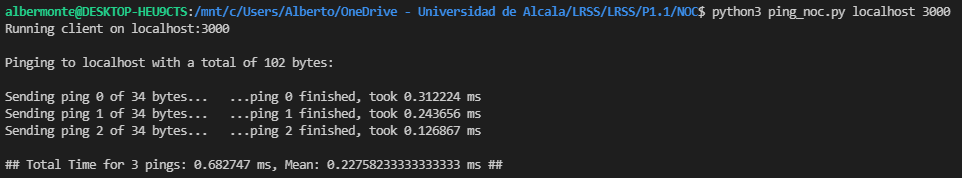
print(

    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:



## OC

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

// 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

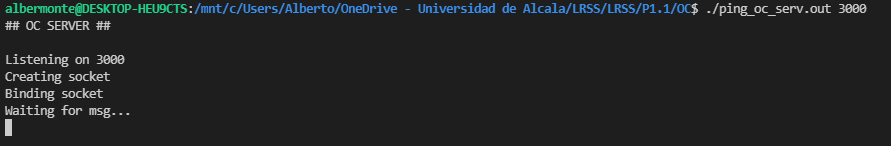
// https://ubidots.com/blog/how-to-simulate-a-tcpudp-client-using-netcat/

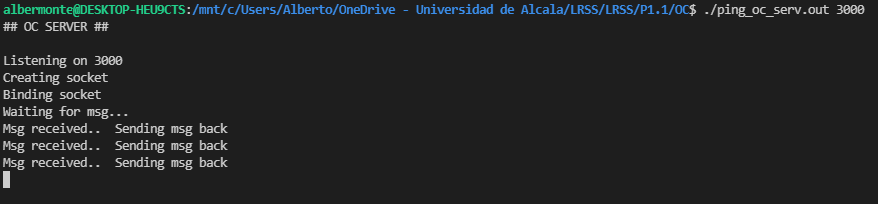
// https://www.tutorialspoint.com/c\_standard\_library/c\_function\_signal.htm

// 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: 

Después del ping: 

// 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);

    }

}

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));

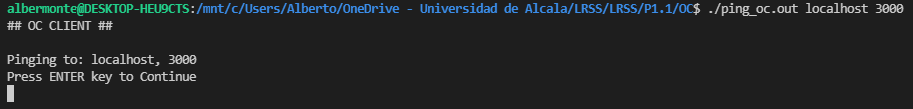
    close(sock);

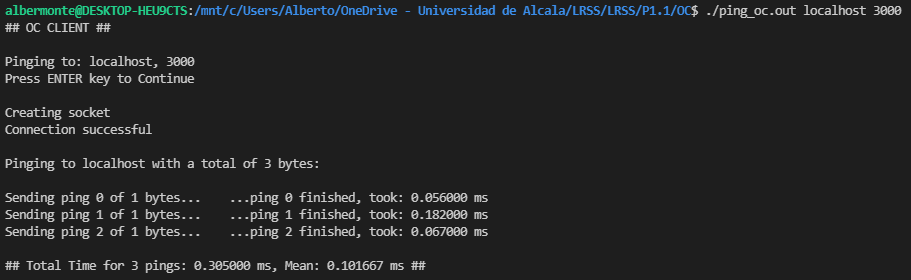
}

// 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: 

Después del ping: 

# 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

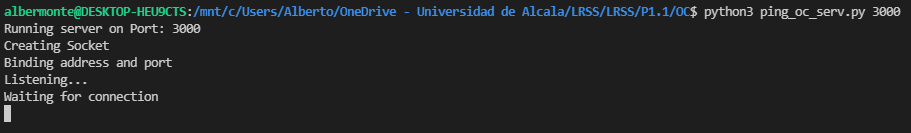
        if not data:

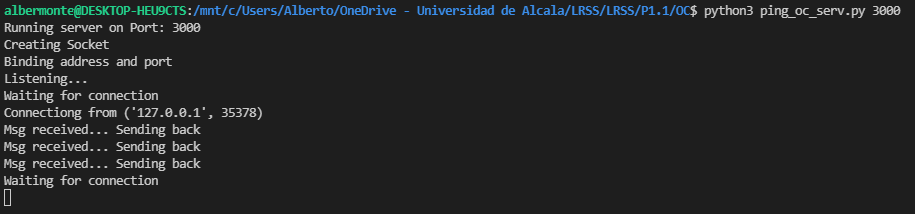
            break

        print("Msg received... Sending back")

        # Reenviamos lo recibido de nuevo hacia el cliente ("pong")

        connection.sendall(data)

Antes del ping: 

Después del ping: 

# 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: 