Practica 1.1

Silviu Constantin Sofrone

Yedra Segovia

- **ping_oc.c**

```
#include <stdio.h> /* printf() and fprintf() */
#include <sys/socket.h> /* socket(), connect(), send(), and recv() */
#include <arpa/inet.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <time.h>
#include <signal.h> /* signal() */
#include <math.h>
#include <netdb.h> /* gethostbyname() */

#define RCVBUFSIZE 32
volatile sig_atomic_t stop; /* Entero que se puede acceder incluso en la ejecución de interrupciones asíncronas */

void CtrlCHand(int signum)
{
  stop = 1;
}

void DieWithError(char *errorMessage) /* Función que lanza el mensaje de error indicado */
{
        perror(errorMessage);
        exit(1);
}
```

```
long double desvEstandar(long double datos[], long double media, int cantidad) /* Ante un conjunto de muestras de tiempo, la media de dichos datos y la cantidad de muestras, desvEstandar() nos devuelve el valor de la desviación estándar */
{
 long double resultado = 0, var = 0;
 for(int y = 0; y < cantidad; y++)
  var += pow((datos[y] - media) ,2);
 var = var/cantidad;
 resultado = sqrt(var);
 return resultado;
}

unsigned long ResolveName(char name[])
{
 struct hostent *host; /* Estructura que contiene la información del host */
 if ((host = gethostbyname(name)) == NULL) /* Pasa la dirección a string */
 {
  fprintf(stderr, "gethostbyname() failed");
  exit(1);
 }

 return *((unsigned long *) host->h_addr_list[0]); /* Devuelve la dirección binaria ordenada por bytes de red */
}

int main(int argc, char *argv[])
{
        int sock; /* Instancia de la abstracción de socket */
        struct sockaddr_in pingServAddr; /* Estructura para la dirección del servidor ping */
        unsigned short pingServPort; /* Puerto del servidor ping */
        char *servIP;  /* Dirección IP del servidor */
```

```
        char *pingString; /* Mensaje que vamos a enviar al servidor para probar el ping, a diferencia de un echo, el contenido de este mensaje no es relevante */

        char echoBuffer[RCVBUFSIZE]; /* Buffer para el mensaje a enviar */

        unsigned int pingStringLen; /* Aquí almacenaremos el tamaño del mensaje a enviar */

/* Variables definidas para la comprobación de los mensajes enviados (a diferencia de echo, con el ping no son estrictamente necesarias estas comprobaciones) y para el cálculo de las estadísticas */

        int bytesRcvd, totalBytesRcvd, ttl = 64, Recv = 0;

 long double time = 0, time_total = 0, time_min = 0, avg = 0, time_max = 0, mdev = 0;

 long double array[] = { 0 };

 struct timespec time_send, time_recv;


        if ((argc < 2) || (argc > 3)) /* Se comprueba que el número de argumentos sean 3; el ejecutable, la dirección IP del servidor y el puerto destino */

        {

                fprintf(stderr, "Usage: %s <Server IP> [<PING Port>]\n", argv[0]); /* En caso de que los argumentos introducidos no sean correctos, se le muestra por pantalla al usuario como ejecutar correctamente el programa */

                exit(1);

        }


        servIP = argv[1]; /* El segundo argumento introducido se asocia con la variable servIP */

        pingString = "s"; /* Dado que el contenido del mensaje no es relevante a la hora de ejecutar un ping, le damos cualquier valor */

        pingServPort = atoi(argv[2]); /* Asociamos de igual manera el tercer argumento referente al puerto del servidor. Para ello utilizamos la función atoi(), que convierte una cadena a su valor entero */


 /* Creamos una instancia de la abstracción de socket mediante la función socket() */

        if ((sock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0)

                /* 1er parámetro: Indica el protocolo familia del socket, en nuestro caso, PF_INET especifica un socket que usa protocolos de la familia de protocolos de internet*/

                /* 2º parámetro: SOCK_STREAM especifíca un socket con semántica de flujo de bytes confiable (emplea el protocolo TCP)
```

```
                /* 3er parámetro: Indica el protocolo end-to-end a utilizar, utilizamos
IPPROTO_TCP para stream socket */

                /* Por último, socket() devuelve -1 en caso de fallar el proceso de creación, por
tanto, mediante un if, comprobamos e informamos si ha habido un error en caso de que socj
sea menor que 0 */

                DieWithError("socket() failed");


 /* Establecer TTL en la cabecera IP */

 if (setsockopt(sock, IPPROTO_IP, IP_TTL, &ttl, sizeof(ttl)) < 0)

/* Debemos indicarle el socket, el protocolo end-to-end a utilizar, el nivel del protocolo end-
to-end (IP_TTL), la variable asociada y su tamaño */

   DieWithError("setsockopt() failed");


/* Construir la estructura de la dirección del servidor antes de pasársela a connect() */

        memset(&pingServAddr, 0, sizeof(pingServAddr)); /* Nos aseguramos de que los bytes
extra contenidos en la estructura estén a 0 */

        pingServAddr.sin_family = AF_INET; /* En el campo sin_family indicamos con qué tipo
de direcciones puede comunicarse el socket mediante la familia de direcciones. En este caso,
con direcciones IPv4  */

 pingServAddr.sin_addr.s_addr = ResolveName(servIP); /* Llamada a la función ResolveName
con la dirección IP del servidor para el servicio DNS */

        pingServAddr.sin_port = htons(pingServPort); /* htons() convierte el entero sin signo
pingServPort del orden de bytes del host al orden de bytes de la red */


/* Establecer conexión entre este socket (cliente) y el socket del servidor ping */

        if (connect(sock, (struct sockaddr *) &pingServAddr, sizeof(pingServAddr)) < 0)

        /* Debemos pasarle como parámetros el socket del cliente, la estructura del socket del
servidor que contiene su dirección y puerto, y por último su tamaño */

                DieWithError("connect() failed");


        pingStringLen = strlen(pingString); /* Calculamos tamaño del mensaje “basura” */

        printf("PING %s \n", servIP);


 signal(SIGINT, CtrlCHand); /* Antes de enviar el ping llamamos a la función signal(). En caso de
pulsarse Ctrl + C salta a la función CtrlCHand que pone la variable stop a 1 */
```

```
 int i = 0; /* Contador de pings */

 while(!stop) /* Bucle infinito hasta que stop = 1, es decir, hasta que se pulse Ctrl + C */

 {

  i ++; /* Con cada iteración aumenta el contador */

  clock_gettime(CLOCK_REALTIME, &time_send); /* Obtenemos el tiempo de envío del mensaje en cuestión */

  if (send(sock, pingString, pingStringLen, 0) != pingStringLen) /* Mandamos el mensaje “basura”, indicando el tamaño del mensaje en cuestión y con los flags a 0. Comprobamos que send() esté mandando el número de bytes correcto */

        DieWithError("send() sent a different number of bytes than expected");


        totalBytesRcvd = 0;

        while (totalBytesRcvd < pingStringLen) /* De la función del echo rescatamos la comprobación que se hacía para comprobar que el servidor ha recibido el mensaje completo, ya que él nos lo va a volver a enviar para que nosotros hagamos la comprobación. Sin embargo, cómo ya hemos dicho, esta parte se puede obviar en el ping ya que solo nos importa saber si el servidor ha recibido “algo” */

        {

                if ((bytesRcvd = recv(sock, echoBuffer, RCVBUFSIZE - 1, 0)) <= 0) /* Como indicábamos en el comentario anterior, con comprobar solamente que se ha recibido algo sería suficiente */

                        break;

   else

   {

    Recv++; /* Contamos los paquetes recibidos de forma satisfactoria */

    totalBytesRcvd += bytesRcvd; /* Podríamos obviarlo */


    clock_gettime(CLOCK_REALTIME, &time_recv); /* Obtenemos los tiempos al recibir las respuestas para obtener el tiempo que tarda cada ping */

    double timeTranSeg = ((double)(time_recv.tv_nsec - time_send.tv_nsec)) *0.000000001; /* Pasamos el tiempo transcurrido en nanosegundos a segundos, desde el tiempo en segundos */

    time = (time_recv.tv_sec - time_send.tv_sec) + timeTranSeg;

    time /= 0.001; /* Sumamos todo en segundos para luego pasarlo a ms */

    time_total += time; /* Acumulamos todo en la variable time_total que indica el tiempo que ha tardado el comando ping en su conjunto */
```

```
    array[i-1] = time; /* Debemos guardar todos los instantes de tiempo en vistas a calcular la desviación estándar al finalizar el ping */


                    printf("%d bytes from %s: icmp_seq=%d ttl=%d time=%.3Lf ms \n", pingStringLen, servIP, i, ttl, time); /* Mostramos la información de cada ping */

    /* Calculamos las estadísticas referentes a los tiempos mínimos, máximos y media */

    if (time_min == 0 || time < time_min)

     time_min = time;


    if (time_max == 0 || time > time_max)

     time_max = time;


    avg = time_total/i;

    sleep(1); /* Invocamos a la función sleep() para que repita el proceso cada segundo y sea más fácil de visualizar para el cliente */

    }

          }

 }

 mdev = desvEstandar(array, avg, i); /* Llamada a la función que hemos implementado para realizar los cálculos referentes a la desviación estándar */


 printf("\n");

          printf("--- %s ping statistics ---\n", servIP);

 printf("%d packets transmitted, %d received, %.2f%% packet loss, time %.3Lf ms \n", i, Recv, ((i - Recv)/i) * 100.0, time_total);

 printf("rtt min/avg/max/mdev = %.3Lf/%.3Lf/%.3Lf/%.3Lf ms \n", time_min, avg, time_max, mdev); /* Mostramos los datos estadísticos calculados */

          close(sock); /* Cerramos la conexión del cliente con close() */

          exit(0);

}
```

- **ping_oc_serv.c**

A partir de aquí, no volveremos a explicar partes del código previamente explicadas.

```
#include <stdio.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>

#define RCVBUFSIZE 32
#define MAXPENDING 5

void DieWithError(char *errorMessage)
{
 perror(errorMessage);
 exit(1);
}

void HandleTCPClient(int clntSocket)
{
 char pingBuffer[RCVBUFSIZE];
 int recvMsgSize; /* Tamaño del mensaje “basura” recibido */

 /* Rescatamos las líneas de código del echo, teniendo en cuenta que en caso del ping no nos importa el contenido del mensaje y podríamos simplemente comprobar que han llegado datos*/
 if ((recvMsgSize = recv(clntSocket, pingBuffer, RCVBUFSIZE, 0)) < 0)
   DieWithError("recv() failed") ;

/* Manda los datos recibidos y recibe otra vez hasta el final de la transmisión, con volver a mandar cualquier mensaje, no necesariamente los datos recibidos, sería suficiente */
 while (recvMsgSize > 0) /* zero indicates end of transmission */
 {
```

```
    if (send(clntSocket, pingBuffer, recvMsgSize, 0) != recvMsgSize)
      DieWithError("send() failed");

    /* Comprueba si quedan datos por recibir */
    if ((recvMsgSize = recv(clntSocket, pingBuffer, RCVBUFSIZE, 0)) < 0)
      DieWithError("recv() failed");
  }

  close(clntSocket);
}

int main(int argc, char *argv[])
{
  int servSock; /* Instancia del socket del servidor*/
  int clntSock; /* Instancia del socket del cliente */
  struct sockaddr_in pingServAddr; /* Dirección del servidor */
  struct sockaddr_in pingClntAddr; /* Dirección del cliente*/
  unsigned short pingServPort; /* Puerto del servidor */
  unsigned int clntLen; /* Tamaño de la estructura de la dirección del cliente */

  if (argc != 2) /* El número de argumentos deben ser dos*/
  {
    fprintf(stderr, "Usage: %s <Server Port>\n", argv[0]) ;
    exit(1);
  }

  pingServPort = atoi(argv[1]);

/* Crear un socket para las conexiones entrantes */
  if ((servSock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0)
    DieWithError( "socket () failed");
```

```
/* Estructura de la dirección del servidor*/
  memset(&pingServAddr, 0, sizeof(pingServAddr));
  pingServAddr.sin_family =  AF_INET;
  pingServAddr.sin_addr.s_addr = htonl(INADDR_ANY);
  pingServAddr.sin_port = htons(pingServPort);

  /* Asignamos un número de puerto al socket servidor mediante bind() */
  if (bind(servSock, (struct sockaddr *)&pingServAddr, sizeof(pingServAddr)) < 0)
    DieWithError ( "bind () failed");

  /* Avisamos al sistema de que permita las conexiones al puerto indicado en la estructura del
socket servidor mediante listen()*/
  if (listen(servSock, MAXPENDING) < 0)
    DieWithError("listen() failed");

  for (;;) /* El programa corre continuamente */
  {
    /* Set the size of the in-out parameter */
    clntLen = sizeof(pingClntAddr);

    /*  Acepta la conexión de los clientes que se conectan al servidor */
    if ((clntSock = accept(servSock, (struct sockaddr *) &pingClntAddr, &clntLen)) < 0)
      DieWithError("accept() failed");

    printf("Handling client %s\n", inet_ntoa(pingClntAddr.sin_addr));
    HandleTCPClient (clntSock); /* Recibe el mensaje del socket cliente y se lo envía de vuelta */
}
/* NOT REACHED */
}
```

- **ping_noc.c**

```
#include <stdio.h> /* for printf() and fprintf() */
#include <sys/socket.h> /* for socket(), connect(), sendto(), and recvfrom() */
#include <arpa/inet.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <signal.h>
#include <time.h>
#include <math.h>
#include <netdb.h> /* gethostbyname() */

#define ECHOMAX 255
volatile sig_atomic_t stop;

void CtrlCHand(int signum)
{
  stop = 1;
}

void DieWithError(char *errorMessage);
long double desvEstandar(long double datos[], long double media, int cantidad);
unsigned long ResolveName(char name[]);

int main(int argc, char *argv[])
{
  int sock;
  struct sockaddr_in pingServAddr;
  struct sockaddr_in fromAddr; /* Dirección fuente del ping*/
  unsigned short echoServPort;
```

```
unsigned int fromSize;
char *servIP;
char *pingString; /* Contiene un mensaje “basura” */
char echoBuffer[ECHOMAX+1];
int pingStringLen;
int respStringLen;
int ttl = 64, Recv = 0;
long double time = 0, time_total = 0, time_min = 0, avg = 0, time_max = 0, mdev = 0;
long double array[] = { 0 };
struct timespec time_send, time_recv;

if ((argc < 2) || (argc > 3)) /* Test for correct number of arguments */
{
  fprintf(stderr,"Usage: %s <Server IP> [<Echo Port>]\n", argv[0]);
  exit(1);
}

servIP = argv[1] ;
pingString = "s" ;

/* Para crear un socket UDP utilizamos SOCK_STREAM para especificar un socket que usa
UDP como protocolo de transporte, y IPPROTO_UDP como protocolo end-to-end */
if ((sock = socket(PF_INET, SOCK_DGRAM, IPPROTO_UDP)) < 0)
  DieWithError( "socket () failed") ;

if (setsockopt(sock, IPPROTO_IP, IP_TTL, &ttl, sizeof(ttl)) < 0)
  DieWithError("setsockopt() failed");

memset(&pingServAddr, 0, sizeof(pingServAddr)); /* Zero out structure */
pingServAddr.sin_family = AF_INET; /* Internet addr family */
pingServAddr.sin_addr.s_addr = ResolveName(servIP); /* Server IP address */
```

```
 pingServAddr.sin_port = htons(echoServPort); /* Server port */


 printf("PING %s \n", servIP);

 signal(SIGINT, CtrlCHand);

 int i = 0;


 while(!stop)

 {

  i ++;

  clock_gettime(CLOCK_REALTIME, &time_send);

  /* En UDP solo le indicamos a sento() el servidor destino*/

  if (sendto(sock, pingString, pingStringLen, 0, (struct sockaddr *) &pingServAddr,
sizeof(pingServAddr)) != pingStringLen)

    DieWithError("sendto() sent a different number of bytes than expected");


  /* Al igual que en el caso de TCP, solo sería necesario comprobar que recibimos “algo” de
recvfrom(), no es necesario comprobar con pingStringLen, sin embargo, hemos mantenido esta
parte del código del echo*/

  fromSize = sizeof(fromAddr) ;

  if ((respStringLen = recvfrom(sock, echoBuffer, ECHOMAX, 0, (struct sockaddr *) &fromAddr,
&fromSize)) != pingStringLen)

    DieWithError("recvfrom() failed") ;

  if (pingServAddr.sin_addr.s_addr != fromAddr.sin_addr.s_addr)

  {

    fprintf(stderr,"Error: received a packet from unknown source.\n");

    exit(1);

  }

  clock_gettime(CLOCK_REALTIME, &time_recv);

  double timeTranSeg = ((double)(time_recv.tv_nsec - time_send.tv_nsec))*0.000000001;

  time = (time_recv.tv_sec - time_send.tv_sec) + timeTranSeg;

  time /= 0.001;

  time_total += time;
```

```
   array[i-1] = time;


   printf("%d bytes from %s: icmp_seq=%d ttl=%d time=%.0Lf ms \n", pingStringLen, servIP, i,
ttl, time);


   /* Stats */
   if (time_min == 0 || time < time_min)
      time_min = time;


    if (time_max == 0 || time > time_max)
      time_max = time;


    avg = time_total/i;
   sleep(1);
  }
  mdev = desvEstandar(array, avg, i);


  printf("\n");
            printf("--- %s ping statistics ---\n", servIP);
  printf("%d packets transmitted, %d received, %.2f%% packet loss, time %.0Lf ms \n", i, Recv,
((i - Recv)/i) * 100.0, time_total);
  printf("rtt min/avg/max/mdev = %.3Lf/%.3Lf/%.3Lf/%.3Lf ms \n", time_min, avg, time_max,
mdev);


  close(sock);
  exit(0);
}
```

- **ping_noc_serv.c**

```
#include <stdio.h>
#include <sys/socket.h>
```

```
#include <arpa/inet.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

#define ECHOMAX 255

void DieWithError(char *errorMessage);

int main(int argc, char *argv[])
{
  int sock;
  struct sockaddr_in pingServAddr;
  struct sockaddr_in pingClntAddr;
  unsigned int cliAddrLen; /* Tamaño de los datos recibidos, no sería necesario */
  char echoBuffer[ECHOMAX];
  unsigned short pingServPort;
  int recvMsgSize;

  if (argc != 2)
  {
    fprintf(stderr, "Usage: %s <UDP SERVER PORT>\n", argv[0]) ;
    exit(1);
  }

  pingServPort = atoi(argv[1]) ;

  /* Creamos un socket que reciba y mande datos, ya que en UDP no se establece conexión, no
necesitamos crear un socket para cada cliente y no necesitamos hacer la llamada a listen() */
  if ((sock = socket(PF_INET, SOCK_DGRAM, IPPROTO_UDP)) < 0)
    DieWithError("socket() failed");
```

```
 memset(&pingServAddr, 0, sizeof(pingServAddr)); /* Zero out structure */
 pingServAddr.sin_family = AF_INET; /* Internet address family */
 pingServAddr.sin_addr.s_addr = htons(INADDR_ANY); /* Any incoming interface */
 pingServAddr.sin_port = htons(pingServPort); /* Local port */

 /* “Unimos” con la dirección del servidor */
 if (bind(sock, (struct sockaddr *) &pingServAddr, sizeof(pingServAddr)) < 0)
   DieWithError("bind() failed");

 for (;;)
 {
   cliAddrLen = sizeof(pingClntAddr);

   /* Llamamos recvfrom() con el mismo socket, como siempre, no haría falta comprobar la integridad del mensaje, solo nos importa saber si hay conexión */
   if ((recvMsgSize = recvfrom(sock, echoBuffer, ECHOMAX, 0, (struct sockaddr *) &pingClntAddr, &cliAddrLen)) < 0)
     DieWithError("recvfrom() failed");

   printf("Handling client %s\n", inet_ntoa(pingClntAddr.sin_addr));

   /* Enviámos datos de vuelta al cliente para comprobar que funciona el ping*/
   if (sendto(sock, echoBuffer, recvMsgSize, 0, (struct sockaddr *) &pingClntAddr, sizeof(pingClntAddr)) != recvMsgSize)
     DieWithError("sendto() sent a different number of bytes than expected"); }
   /* NOT REACHED */
}
```

- **ping_oc.py**

```
#!/usr/bin/env python3
```

```
import sys
import socket
import time
import math

def desvEstandar(array, avg, seq):
  var = 0
  resultado = 0
  for i in range(len(array)):
    var += pow((array[i] - avg) ,2)
  var = var/seq
  resultado = math.sqrt(var)
  #print('resultado', resultado)
  return resultado

clk_id1 = time.CLOCK_REALTIME # System-wide real-time clock
ttl = 64
recv = 0 #variable que contendrá el nº de pings recibidos
array = []
time_total = 0
time_min = 0
time_max = 0

if len(sys.argv) < 2 or len(sys.argv) > 3:
  print('Usage:', sys.argv[0], '<Server IP> [<PING Port>]')

#Definición de variables
servIP = sys.argv[1]
pingServPort = sys.argv[2]
```

#Crear socket, en Python solo tenemos que pasar como parámetros la familia de direcciones (AF_INET) y la semántica de la transmisión de datos con el socket (SOCK_STREAM)

```
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

#Set ttl

sock.setsockopt(socket.SOL_IP, socket.IP_TTL, ttl)


#Para la conexión debemos proporcionarle a connect() la dirección IP y el puerto del servidor (recogidos de los argumentos) en un formato address compuesto por un par (host, port)

sock.connect((servIP, int(pingServPort)))


print('PING',servIP)

seq = 0


try:

  while True:

    seq += 1

    pingString = b"mensaje" #los datos que enviamos con send() deben pasarse a bytes

    time_send = time.time() #tiempo envío ping en seg

    sock.send(pingString); #envíamos el mensaje “basura” al socket con el que hemos establecido la conexión


    totalBytesRcvd = 0;

    pingStringLen = len(pingString)

    bytesRcvd = 0


    if sock.recv(16): #comprobamos sólo si recibimos respuesta de vuelta, no hace falta comprobar la integridad del mensaje. A recv() debemos indicarle un tamaño de buffer, en nuestro caso con 16 bytes será suficiente

      recv += 1

      time_recv = time.time()

      tiempo = (time_recv - time_send) * 1000

      print("{} bytes from {}: icmp_ser={} ttl={} time={:.3f} ms".format(pingStringLen, servIP, seq, ttl, tiempo))
```

```
    time_total += tiempo
    array.append(tiempo) #llenamos array con las muestras de tiempo

    if time_min == 0 or tiempo < time_min:
      time_min = tiempo

    if time_max == 0 or tiempo > time_max:
      time_max = tiempo

    avg = time_total/seq

  else:
    recv -= 1

  time.sleep(1)
except KeyboardInterrupt: #en caso de pulsarse Ctrl + C terminamos el envío de pings y pasamos a las estadísticas
  mdev = desvEstandar(array, avg, seq)
  print()
  print("--- {} ping statistics ---".format(servIP))
finally:
  print('{} packets transmitted, {} received, {}% packet loss, time {:.3f} ms'.format(seq, recv, ((seq - recv)/seq) * 100.0, time_total))
  print('rtt min/avg/max/mdev = {:.3f}/{:.3f}/{:.3f}/{:.3f} ms'.format(time_min, avg, time_max, mdev))
  sock.close()
```

- **ping_oc_serv.py**

```
#!/usr/bin/env python3

import sys
```

```
import socket


if len(sys.argv) != 2:
  print('Usage:', sys.argv[0], '<Server Port>\n')


pingServPort = sys.argv[1]


sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
#Recordamos que en TCP debemos unir (bind) el socket al puerto, pasando como argumentos a la función bind() el par (host, port), en nuestro caso la dirección del servidor y el puerto indicado por línea de comandos
sock.bind(('localhost', int(pingServPort)))
#habilita al servidor para aceptar conexiones
sock.listen(1)


while True:
  connection, clntPingAddr = sock.accept() #se identifica la conexión y se acepta
  try:
    print('conexion con', clntPingAddr)
    #rescatamos el código del echo en python, por el cual comprobamos los datos recibidos y los redirigimos al cliente con el que hemos establecido conexión
    while True:
      bytesRcvd = connection.recv(16)
      print('received {!r}'.format(bytesRcvd))
      if bytesRcvd:
        print('sending data back to the client')
        connection.send(bytesRcvd)
      else:
        print('no hay datos recibidos de', clntPingAddr)
        break


  finally:
```

```
connection.close() #finalmente cerramos la conexión
```

- **ping_noc.py**

```
#!/usr/bin/env python3
# -*- coding: cp1252 -*-

import sys
import socket
import time
import math

def desvEstandar(array, avg, seq):
  var = 0
  resultado = 0
  for i in range(len(array)):
    var += pow((array[i] - avg) ,2)
  var = var/seq
  resultado = math.sqrt(var)
  return resultado

clk_id1 = time.CLOCK_REALTIME
ttl = 64
recv = 0
array = []
time_total = 0
time_min = 0
time_max = 0
avg = 0
mdev = 0
```

```
if len(sys.argv) < 2 or len(sys.argv) > 3:
  print('Usage:', sys.argv[0], '<Server IP> [<PING Port>]')

servIP = sys.argv[1]
pingServPort = sys.argv[2]

#Crearmos el socket, esta vez del tipo SOCK_DGRAM ya que estamos en UDP
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

sock.setsockopt(socket.SOL_IP, socket.IP_TTL, ttl)

print('PING',servIP)
seq = 0

try:
  while True:
    seq += 1
    pingString = b"mensaje"
    time_send = time.time()
    sock.sendto(pingString, (servIP, int(pingServPort)));
    pingStringLen = len(pingString)

    if sock.recvfrom(16): #comprobamos sólo si recibimos respuesta de vuelta, e indicamos a recvfrom(), al igual que a recv(), un tamaño de buffer suficientemente grande
      recv += 1
      time_recv = time.time()
      tiempo = (time_recv - time_send) * 1000
      print("{} bytes from {}: icmp_ser={} ttl={} time={:.3f} ms".format(pingStringLen, servIP, seq, ttl, tiempo))
      time_total += tiempo
      array.append(tiempo)
```

```
    if time_min == 0 or tiempo < time_min:
      time_min = tiempo

    if time_max == 0 or tiempo > time_max:
      time_max = tiempo

    avg = time_total/seq

  else:
    recv -= 1

  time.sleep(1)
except KeyboardInterrupt:
  mdev = desvEstandar(array, avg, seq)
  print()
  print("--- {} ping statistics ---".format(servIP))
finally:
  print('{} packets transmitted, {} received, {}% packet loss, time {:.3f} ms'.format(seq, recv,
((seq - recv)/seq) * 100.0, time_total))
  print('rtt min/avg/max/mdev = {:.3f}/{:.3f}/{:.3f}/{:.3f} ms'.format(time_min, avg, time_max,
mdev))
  sock.close()
```

- **ping_noc_serv.py**

```
#!/usr/bin/env python3

import sys
import socket

if len(sys.argv) != 2:
  print('Usage:', sys.argv[0], '<Server Port>\n')
```

```
pingServPort = sys.argv[1]


sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

#Unimos (bind) el socket al puerto

sock.bind(('localhost', int(pingServPort)))


while True:

  message, clntPingAddr = sock.recvfrom(16) #Identificamos las conexiones en message y olo ten emos que comprobar que se ha recibido “algo” con recvfrom(), que nos indique que hay conexión

  print('Handling client', clntPingAddr)

  sock.sendto(message, clntPingAddr) #Respondemos al cliente para que sepa que el ping se ha ejecutado correctamente
```