# PRACTICA 1.1. Comunicación de aplicaciones mediante sockets.

# Nombres: Fernando Fernández Ortiz e Ismael Barber Sanchez-Heredero

Todos los códigos se encuentran comentados explicando la funcionalidad que hace cada apartado. Se asume que con los comentarios añadidos en el código la memoria quedaría explicada.

* **Código para socket cliente en modo no conectivo lenguaje C.**

#include <netinet/in.h>

#include <netinet/ip.h>

#include <netinet/ip\_icmp.h>

#include <arpa/inet.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <stdio.h>

#include <stdlib.h>

#include <netdb.h>

#include <unistd.h>

#include <string.h>

#include <ctype.h>

#include <time.h>

#include <math.h>

#define LONG\_PAQUETE 32

#define PUERTO 2000

//Creación del socket

int main(int argc, char \*argv[])

{

    /\*

    Se crean una serie de variables que se van a utilizar para el cálculo del tiempo entre el envio y la recepción de

    los mensajes. También se crean otras variables para calcular del número de paquetes enviados, cuantos se reciben

    y cuantos se pierden.

    \*/

    double t\_ms[3];

    double t\_min = 0.0, t\_max = 0.0, t\_medio = 0.0;

    int paquetes\_enviados = 0, paquetes\_recibidos = 0, paquetes\_perdidos = 0; //Número de mensajes enviados

    struct sockaddr\_in datos\_servidor, datos\_cliente;

    /\*

    Se calculan el tamaño de los datos del cliente y del servidor que van a ser utilizados a posteriori

    bind() y sendto()

    \*/

    int long\_serv = sizeof(datos\_servidor);

    int long\_cliente = sizeof(datos\_cliente);

    /\*

    Se crean variables de tipo puntero a char que van a ser del tamaño 32 bytes, es decir, del tamaño del mensaje

    que se va a enviar mediante el socket

    \*/

    char\* msg\_enviado[LONG\_PAQUETE];

    memset(msg\_enviado,0,LONG\_PAQUETE);

    char\* msg\_recibido[LONG\_PAQUETE];

    // En solicitud se guarda el "segundo" parámetro que se pasa por argumento.

    char\* solicitud = argv[1];

    // Se rellena la estructura de los datos del servidor  y del cliente con 0's.

    bzero((char\*) &datos\_servidor, sizeof(datos\_servidor));

    bzero((char\*) &datos\_cliente, sizeof(datos\_cliente));

    /\*

    Con este if se pretende comprobar que el "segundo" parametro que se pasa por argumento

    al llamar al programa sea "ping" tanto minúsculas como mayúsculas.

    \*/

    if(strcmp(argv[1], "ping") != 0 && strcmp(argv[1], "PING") != 0)

    {

        printf("El primer argumento que se debe de pasar debe de ser 'ping' (MAYUS o MINUS)\n");

        return -1;

    }

    /\*

    En la siguientes variables se almacenan la 'IP' pasada por argumento a la que se encuentra el servidor escuchando

    y su puerto. Como los argumentos que se pasan por la linea de comandos van por defecto en formato CHAR, para guardar el puerto

    como entero utilizamos la funcion atoi().

    \*/

    char \*nombre\_direccion = argv[2];

    char \*puerto\_char = argv[3];

    int puerto = atoi(puerto\_char);

    /\*

    En la siguientes lineas, la estructura sockaddr\_in que hemos creado anteriormente y que hemos nombrado datos\_servidor

    va a almacenar los datos del servidor con el que nos queremos comunicar y cuyos datos hemos pasado como parámetros al

    iniciar el programa. Utilizamos la función htons() para traducir el orden de los bytes del host al orden de los bytes de la red.

    También, la función inet\_addr se utiliza para traducir una dirección IP en formato de cadena de caracteres en un formato de 32 bits

    a un formato adecuado para su uso en red.

    \*/

    datos\_servidor.sin\_family = AF\_INET;

    datos\_servidor.sin\_port = htons(puerto);

    datos\_servidor.sin\_addr.s\_addr = inet\_addr(nombre\_direccion);

    /\*

    Para el siguiente condicional if, se verifica que el numero de argumentos no sea mayor a 4.

    \*/

    if(argc > 4)

    {

        printf("El número de argumentos pasado debe ser 'nombre del programa', 'direccion IP o nombre de host' y 'puerto de escucha'");

        return 0;

    }

    /\*

    En caso de que se cumpla que haya 4 o menos argumentos. Se verifica que el puerto que se ha introducido

    por la linea de argumentos este comprendido en el rango de 1024-5000, ya que es el rango de puertos

    que es utilizado para servidores como viene comentado en el libro.

    \*/

    else

    {

        if(puerto < 1023 || puerto > 5000)

        {

            printf("El número de puerto del servidor escogido deber ser 1024-5000\n");

            return -1;

        }

    }

    // Creación de un socket AF\_INET(DOMINIO INTERNET TCP/IP), en modo no conectivo SOCK\_DGRAM (UDP)

    //En caso de fallo devuelve -1, si todo va bien devuelve un valor entero bajo.

    int sockfd = socket(AF\_INET, SOCK\_DGRAM, 0);

    if(sockfd < 0)

    {

        printf("Error creando el socket\n");

        return -1;

    }

    /\*

    Para la estructura del cliente se añade el tipo de familia que va a utilizar, en este caso

    AF\_INET(TCP/IP) y se le agrega el puerto que va a utilizar definido por defecto el 2000,

    \*/

    datos\_cliente.sin\_family = AF\_INET;

    datos\_cliente.sin\_port = htons(PUERTO);

    /\*

    Definimos una variable llamada ttl con valor 64. Con la función setsockopt() se establece

    el valor del campo TTL de los paquetes IP enviados. TTL permite establecer el numero de saltos de red

    que puede atravesar un paquete IP a través de los routers.

    IPPROTO\_IP sirve para especificar que se esta configurando a nivel IP. IP\_TTL especifica la opción de TTL.

    &ttl es un puntero a la variable que contiene el valor del TTL deseado, por último sizeof(ttl) calcula el

    tamaño de la variable ttl en bytes.

    \*/

    int ttl = 64;

    int ttl\_check = setsockopt(sockfd, IPPROTO\_IP, IP\_TTL, &ttl, sizeof(ttl));

    if (ttl\_check < 0)

    {

        printf("Error asociando TTL\n");

        return -1;

    }

    /\*

    Se realiza la asociación una dirección local con el socket que hemos creado anteriormente.

    El segundo parámetro contiene la dirección a la que se quiere asociar el socket.

    El tercer parámetro especifica el tamaño de la estructura en bytes (calculado anteriormente).

    Provoca un error si el valor devuelto en 'asociación' es '-1'.

    \*/

    int asociacion = bind(sockfd,(struct sockaddr \*) &datos\_cliente, long\_cliente);

    if(asociacion < 0)

    {

        printf("Error en asociación 'bind'. Cerrando conexión...\n");

        // Se cierra la conexión

        close(sockfd);

        return -1;

    }

    else

    {

        /\*

        Se muestran datos por pantalla especificando la dirección IP y puerto a la que se van a enviar los datos,

        para verificar que no ha habido ningun error.

        \*/

        printf("Asociación del socket creada.\n\tPreparado para enviar por el puerto: %d\n", ntohs(datos\_cliente.sin\_port));

        printf("\tEnviando a la dirección IP: %s\n", inet\_ntoa(datos\_servidor.sin\_addr));

    }

    printf("Enviando datos...\n\n");

    printf("\nHaciendo ping con %s: Enviando una cantidad de datos de %ld bytes\n\n", nombre\_direccion, sizeof(msg\_enviado)/8);

   // Se realiza un bucle for para que por defecto se realicen 4 envios.

   for(int i = 0; i < 4; i++)

    {

        /\*

        Mediante la función send() se envian datos mediante el socket creado.

        Se toman como argumentos el descriptor del socket creado, un puntero al buffer que contiene

        los datos que se van a enviar, además de su tamaño en bytes.

        Además, también se indican los datos del servidor con el que nos queremos comunicar.

        En caso de dar error, se devuelve un '-1'.

        \*/

        int send = sendto(sockfd, msg\_enviado, LONG\_PAQUETE, 0, (struct sockaddr \*) &datos\_servidor, long\_serv);

        /\*

        Se hace uso de la función clock() definida en la libreria 'time.h' para iniciar un contador de tiempo cuando

        se realiza el envio del mensaje.

        \*/

        clock\_t comienzo = clock();

        if(send < 0)

        {

            printf("Error enviando datos\n");

            // En caso de que ocurra un error, el contador de paquetes perdidos se incrementa en 1.

            paquetes\_perdidos += 1;

        }

        else

        {

            // En caso de que se envie correctamente, el contador de paquetes enviados se incrementa en 1.

            paquetes\_enviados += 1;

        }

        /\*

        A continuación, el cliente espera a recibir la respuesta por parte del servidor, a través del decriptor del socket.

        Esta respuesta va a ser almacenada en la variable 'msg\_recibido' y de la LONG\_PAQUETE especificada.

        En caso de error, recvfrom devuelve '-1'

        \*/

        int n = recvfrom(sockfd, msg\_recibido, LONG\_PAQUETE, 0, (struct sockaddr \*) 0, (int \*) 0);

        // Se hace uso de la función sleep para que los datos aparezcan por pantalla de forma 'más realista'.

        sleep(1.5);

        if(n < 0)

        {

            /\*

            Si ocurre un error el contador de paquetes perdidos se incrementa en 1.

            \*/

            printf("Error\n");

            paquetes\_perdidos += 1;

        }

        else

        {

            /\*

            En caso contrario, de haber recibido el paquete de forma correcta, el contador

            de paquetes recibidos se incrementa en 1.

            En el instante de recibir el paquete, se inicia otro contador de tiempo denominado como 'final'

            \*/

            paquetes\_recibidos += 1;

            clock\_t final = clock();

            double tiempo\_transcurrido = 0.0;

            /\*

            Se calcula el tiempo transcurrido desde el envio del paquete hasta la recepción del mismo

            calculando su diferencia. El resultado dado sera por tiempos de reloj. Se divide por 'CLOCKS\_PER\_SEC'

            para obtener el tiempo en segundos.

            Finalmente, se hace uso de la función pow() para pasar el tiempo a milisegundos.

            \*/

            tiempo\_transcurrido += (double)(final - comienzo) / CLOCKS\_PER\_SEC;

            t\_ms[i] = tiempo\_transcurrido / pow(10,-3);

            t\_min = t\_ms[0];

            /\*

            Se realizan una serie de condicionales para verificar cual es el tiempo

            mínimo y máximo del envio de cada uno de los paquetes.

            \*/

            if(t\_ms[i] > t\_max)

            {

                t\_max = t\_ms[i];

            }

            if(t\_ms[i] < t\_min)

            {

                t\_min = t\_ms[i];

            }

            // Finalmente se muestra por pantalla los datos relacionados con la recepción de los mensajes.

            printf("Respuesta desde: %s bytes: %ld tiempo: %.3fms TTL = %d \n", inet\_ntoa(datos\_servidor.sin\_addr),sizeof(msg\_recibido)/8, t\_ms[i], ttl);

        }

    }

    /\*

    Por último se realizan una serie de cálculos para mostrar por pantalla las estadísticas

    de ping para la dirección IP del servidor con la que hemos solicitado la conexión.

    \*/

    for(int i = 0; i < 4; i++)

    {

        t\_medio += t\_ms[i];

    }

    double perc\_perdida = (paquetes\_perdidos \* 100)/paquetes\_recibidos;

    double media = t\_medio/4;

    printf("\nEstadísticas de ping para: %s:\n", inet\_ntoa(datos\_servidor.sin\_addr));

    printf("\tPaquetes: Enviados = %d, Recibidos = %d, Perdidos = %d (%.2f%% perdida),\n", paquetes\_enviados, paquetes\_recibidos, paquetes\_perdidos, perc\_perdida);

    printf("Tiempo aproximado de vuelta en mili-segundos:\n");

    printf("\tMinimo = %.3fms, Máximo = %.3fms, Media = %.3fms\n\n", t\_min, t\_max, media);

}

* **Código del socket servidor en modo no conectivo en código C.**

#include <netinet/in.h>

#include <netinet/ip.h>

#include <netinet/ip\_icmp.h>

#include <arpa/inet.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <stdio.h>

#include <stdlib.h>

#include <netdb.h>

#include <unistd.h>

#include <string.h>

#include <time.h>

#define LONG\_BUFFER 32

#define TTL 115

//Creación del socket

int main(int argc, char \*argv[])

{

    struct sockaddr\_in datos\_servidor, datos\_cliente;

    /\*

    Se calculan el tamaño de los datos del cliente y del servidor que van a ser utilizados a posteriori

    bind() y sendto()

    \*/

    int long\_serv = sizeof(datos\_servidor);

    int long\_cliente = sizeof(datos\_cliente);

    /\*

    Se crean variables de tipo puntero a char que van a ser del tamaño 32 bytes, es decir, del tamaño del mensaje

    que se va a enviar mediante el socket

    \*/

    char msg\_recv[LONG\_BUFFER];

    char msg\_enviado[LONG\_BUFFER];

    int recv = 0;

    // Se rellena la estructura de los datos del servidor con 0's.

    bzero((char\*) &datos\_servidor, sizeof(datos\_servidor));

    /\*

    Se guarda el puerto introducido por la línea de comandos y se convierte a entero

    para agregarlo posteriormente a la estructura sockaddr\_in.

    \*/

    char \*puerto\_char = argv[1];

    int puerto = atoi(puerto\_char);

    /\*

    En la siguientes lineas, la estructura sockaddr\_in que hemos creado anteriormente y que hemos nombrado datos\_servidor

    va a almacenar los datos del servidor con el que nos queremos comunicar y cuyos datos hemos pasado como parámetros al

    iniciar el programa. Utilizamos la función htons() para traducir el orden de los bytes del host al orden de los bytes de la red.

    También, la función inet\_addr se utiliza para traducir una dirección IP en formato de cadena de caracteres en un formato de 32 bits

    a un formato adecuado para su uso en red.

    \*/

    datos\_servidor.sin\_family = AF\_INET;

    datos\_servidor.sin\_port = htons(puerto);

    datos\_servidor.sin\_addr.s\_addr = inet\_addr("172.22.58.104");

    if(argc > 2)

    {

        printf("El número de argumentos pasado debe ser 'nombre del programa' y 'puerto de escucha'");

        return 0;

    }

     /\*

    En caso de que se cumpla que haya menos de 2 argumentos. Se verifica que el puerto que se ha introducido

    por la línea de argumentos este comprendido en el rango de 1024-5000, ya que es el rango de puertos

    que es utilizado para servidores como viene comentado en el libro.

    \*/

    else

    {

        if(puerto < 1023 || puerto > 5000)

        {

            printf("El número de puerto escogido deber ser 1024-5000\n");

            return -1;

        }

    }

    // Creación de un socket AF\_INET(DOMINIO INTERNET TCP/IP), en modo no conectivo SOCK\_DGRAM (UDP)

    //En caso de fallo devuelve -1, si todo va bien devuelve un valor entero bajo.

    int sockfd = socket(AF\_INET, SOCK\_DGRAM, 0);

    if(sockfd < 0)

    {

        printf("Error creando el socket\n");

        return -1;

    }

    /\*

    Se realiza la asociación una dirección local con el socket que hemos creado anteriormente.

    El segundo parámetro contiene la dirección a la que se quiere asociar el socket.

    El tercer parámetro especifica el tamaño de la estructura en bytes (calculado anteriormente).

    Provoca un error si el valor devuelto en 'asociación' es '-1'.

    \*/

    int asociacion = bind(sockfd,(struct sockaddr \*) &datos\_servidor, long\_serv);

    if(asociacion < 0)

    {

        printf("Error en asociación 'bind'. Cerrando conexión...\n");

        // Se cierra la conexión

        close(sockfd);

        return -1;

    }

    else

    {

        /\*

        Se muestran datos por pantalla especificando la dirección IP y puerto mediante la cual el servidor va

        a escuchar las peticiones.

        \*/

        printf("Asociación del socket creada.\n\tEscuchando por el puerto: %d\n", ntohs(datos\_servidor.sin\_port));

        printf("\tDirección IP: %s\n", inet\_ntoa(datos\_servidor.sin\_addr));

    }

    do

    {

        /\*

        Se hace uso del bucle do... while para que el servidor se mantenga escuchando las peticiones

        excepto cuando haya algún error a la hora de recibir los datos.

        \*/

        printf("\n\tEsperando datos a recibir...\n");

        /\*

        A continuación, el servidor espera a recibir el envió por parte del cliente, a través del descriptor del socket.

        Esta respuesta va a ser almacenada en la variable 'msg\_recv' y de la LONG\_PAQUETE especificada.

        En caso de error, recvfrom devuelve '-1'

        \*/

        int recv = recvfrom(sockfd, msg\_recv, LONG\_BUFFER, 0, (struct sockaddr \*) &datos\_cliente, &long\_cliente);

        if(recv < 0)

        {

            printf("\tError recibiendo datos\n");

        }

        else

        {

            printf("\tRecibiendo datos desde: %s\n", inet\_ntoa(datos\_cliente.sin\_addr));

        }

        /\*

        Mediante la función send() se envían datos mediante el socket creado.

        Se toman como argumentos el descriptor del socket creado, un puntero al buffer que contiene

        los datos que se van a enviar, además de su tamaño en bytes.

        Además, también se indican los datos del cliente que se está comunicando con nosotros.

        En caso de dar error, se devuelve un '-1'.

        \*/

        if(sendto(sockfd, msg\_enviado, recv, 0, (struct sockaddr \*) &datos\_cliente, long\_cliente) != recv)

        {

            printf("Error\n");

        }

        else

        {

            printf("\tEnviando datos...\n");

        }

    } while(recv != -1);

}

* **Código del cliente socket conectivo (TCP) en lenguaje C.**

#include <netinet/in.h>

#include <arpa/inet.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <stdio.h>

#include <stdlib.h>

#include <netdb.h>

#include <unistd.h>

#include <string.h>

#include <ctype.h>

#include <time.h>

#include <math.h>

#define LONG\_PAQUETE 32

#define PUERTO 2000

//Creación del socket

int main(int argc, char \*argv[])

{

    //Creacion de la estructura sockaddr\_in inicializada a 0.

    struct sockaddr\_in datos\_servidor, datos\_cliente;

    // Se rellena la estructura de los datos del servidor  y del cliente con 0's.

    bzero((char\*) &datos\_servidor, sizeof(datos\_servidor));

    bzero((char\*) &datos\_cliente, sizeof(datos\_cliente));

    /\*

    En la siguientes lineas se crean una serie de variables que calculan la longitud

    de la estructura datos\_servidor y datos\_cliente.

    También se crean variables para el envio y la recepción de los mensajes de un tamaño

    definido previamente en #define LONG\_PAQUETE. Además de las correspondientes variables para tener

    almacenado el tamaño correspondiente de cada mensaje en bytes (dividido entre 8).

    También se crean variables para almacenar en un array los tiempo entre envio y recepción del paquete.

    Por último se crean variables locales para el cálculo de las estadísticas del envio de los

    paquetes.

    \*/

    int long\_serv = sizeof(datos\_servidor);

    int long\_cliente = sizeof(datos\_cliente);

    char\* msg\_enviado[LONG\_PAQUETE];

    int lg\_msg\_enviado = sizeof(msg\_enviado)/8;

    char\* msg\_recibido[LONG\_PAQUETE];

    int lg\_msg\_recibido = sizeof(msg\_recibido)/8;

    double t\_ms[4];

    double t\_min = 0.0, t\_max = 0.0, t\_medio = 0.0;

    int paquetes\_enviados = 0, paquetes\_recibidos = 0, paquetes\_perdidos = 0;

    /\*

    En las siguientes lineas de código se almacena el valor del 'segundo' parámetro pasado por la linea de ordenes.

    Se compara que el dato introducido sea 'ping' tanto minúsculas como mayúsculas. En caso contrario se solicita

    al usuario que vuelva a introducir los datos.

    \*/

    char\* solicitud = argv[1];

    if(strcmp(argv[1], "ping") != 0 && strcmp(argv[1], "PING") != 0)

    {

        printf("El primer argumento que se debe de pasar debe de ser 'ping' (MAYUS o MINUS)\n");

        return -1;

    }

    /\*

    Se crean más variables locales de tipo puntero a char para almacenar la dirección IP y el puerto

    en la que se encuentra escuchando el servidor. Se hace uso de la función atoi() para pasar el valor del puerto

    a entero.

    Respecto a los datos del servidor, se utiliza la familita AF\_INET especificando así que se va a utilizar

    TCP/IP. Se agrega a la estructura el valor del puerto y también la dirección IP.

    \*/

    char \*nombre\_direccion = argv[2];

    char \*puerto\_char = argv[3];

    int puerto = atoi(puerto\_char);

    datos\_servidor.sin\_family = AF\_INET;

    datos\_servidor.sin\_port = htons(puerto);

    datos\_servidor.sin\_addr.s\_addr = inet\_addr(nombre\_direccion);

    /\*

    Respecto a los datos del cliente, es decir, nosotros mismos, especificamos de nuevo que se va a utilizar

    TCP/IP. Y se agrega el puerto que va a utilizar el cliente definido por defecto mediante

    #define PUERTO 2000.

    \*/

    datos\_cliente.sin\_family = AF\_INET;

    datos\_cliente.sin\_port = htons(PUERTO);

    /\*

    Se verifica que el numero de argumentos pasado por la linea de ordenes sea menor que 4.

    En caso de que se introduzca de forma correcta, se verifica que el puerto este comprendido entre

    los rangos 1025-5000.

    \*/

    if(argc > 4)

    {

        printf("El número de argumentos pasado debe ser 'nombre del programa', 'direccion IP o nombre de host' y 'puerto de escucha'");

        return 0;

    }

    else

    {

        if(puerto < 1023 || puerto > 5000)

        {

            printf("El número de puerto del servidor escogido deber ser 1024-5000\n");

            return -1;

        }

    }

    // Creación de un socket AF\_INET(DOMINIO INTERNET TCP/IP), en modo no conectivo SOCK\_DGRAM (UDP)

    //En caso de fallo devuelve -1, si todo va bien devuelve un valor entero bajo.

    int sockfd = socket(AF\_INET, SOCK\_STREAM, 0);

    if(sockfd < 0)

    {

        printf("Error creando el socket\n");

        return -1;

    }

    /\*

    Definimos una variable llamada ttl con valor 64. Con la función setsockopt() se establece

    el valor del campo TTL de los paquetes IP enviados. TTL permite establecer el numero de saltos de red

    que puede atravesar un paquete IP a través de los routers.

    IPPROTO\_IP sirve para especificar que se esta configurando a nivel IP. IP\_TTL especifica la opción de TTL.

    &ttl es un puntero a la variable que contiene el valor del TTL deseado, por último sizeof(ttl) calcula el

    tamaño de la variable ttl en bytes.

    \*/

    int ttl = 64;

    int ttl\_check = setsockopt(sockfd, IPPROTO\_IP, IP\_TTL, &ttl, sizeof(ttl));

    if (ttl\_check < 0)

    {

        printf("Error asociando TTL\n");

        return -1;

    }

    /\*

    Se utiliza connect para establecer conexión con el servidor.

    Como argumentos que se le pasan, primero el descriptor del socket, creado anteriormente.

    El socket se configura para establecer la conexión en los datos del servidor que hayamos

    introducido al inicio del programa.

    En caso de error devuelve -1.

    \*/

    int conexion = connect(sockfd,(struct sockaddr \*) &datos\_servidor, long\_serv);

    if (conexion < 0)

    {

        printf("Cliente: no se puede conectar con el servidor\n");

    }

    else

    {

        printf("\nConexión creada con el servidor.\n\n");

    }

    printf("Enviando datos...\n\n");

    printf("\nHaciendo ping con %s enviando una cantidad de %d bytes:\n\n", nombre\_direccion, lg\_msg\_enviado);

    /\*

    Se hace uso de un bucle for para enviar un total de 4 mensajes al servidor por defecto.

    \*/

    for(int i = 0; i < 4; i++)

    {

        /\*

        Una vez establecida la conexión con el servidor, y aceptada por el mismo se comienzan

        a enviar mensajes con la función write().

        Esta toma como argumentos el descriptor del socket creado anteriormente.

        Un puntero al buffer de datos (&msg\_enviado), y la cantidad de bytes del buffer de datos.

        \*/

        int wr = write(sockfd, &msg\_enviado, lg\_msg\_enviado);

        /\*

        Se comienza un contador de tiempo en formato 'CLOCKS' (tiempos de reloj)

        mediante la funcion clock definida en la libreria'time.h'.

        \*/

        clock\_t comienzo = clock();

        /\*

        La función write devuelve el número de bytes que se han escrito con exito. Por tanto,

        si se devuelve un número diferente, es que ha ocurrido un error a la hora de enviar

        el mensaje, lo que quiere decir que el páquete que se ha tratado de enviar se ha perdido.

        Así, se añade un contador a la variable creada anteriormente como paquete\_perdido.

        \*/

        if(wr != lg\_msg\_enviado)

        {

            printf("Error al enviar mensaje con write\n");

            paquetes\_perdidos += 1;

        }

        else

        {

            /\*

            En caso de que se envie de forma correcta, se añadirá un contador a la cantidad de

            paquetes\_enviados.

            \*/

            paquetes\_enviados += 1;

        }

        /\*

         Añadimos una parada entre el envio y la recepción de mensajes mediante read() para que

         la recepción sea un poco más 'realista'.

        \*/

        sleep(1.25);

        /\*

        Una vez el paquete ha sido enviado, se pretende recibir la respuesta por parte del servidor,

        para ello se emplea la función read() que recibe como argumentos el descriptor del socket,

        un puntero al buffer de destino (&msg\_recibido), y la cantidad de bytes desde el socket

        del servidor.

        En caso de error devuelve '-1' y se añadiría un contador al número de paquetes\_perdidos,

        Devuelve 0 si el servidor ha cerrado la conexión. Por tanto, se cerraría el socket mediante

        la funcion close()

        \*/

        int rd = read(sockfd, &msg\_recibido, lg\_msg\_recibido);

        if (rd == -1)

        {

            printf("Error recibiendo datos\n");

            paquetes\_perdidos += 1;

        }

        if(rd == 0)

        {

            printf("El servidor ha cerrado la conexión\n");

            close(sockfd);

        }

        else

        {

            /\*

                En caso contrario de que se empiecen a recibir paquetes de forma correcta,

                se añade un contador al número de paquetes\_recibidos y se emplea de nuevo la función

                clock() para a posteriori calcular el tiempo en segundos transcurrido entre el envio

                del paquete al servidor y la recepción.

                Estos tiempos se almacenan en un array de datos para cada uno de los paquetes enviados. Se realiza

                el calculo en milisegundos mediante la función pow().

                Se realizan una serie de comparaciones con condicionales if para verificar cual

                es el paquete con menor y mayor tiempo de envio y recepción.

            \*/

            paquetes\_recibidos += 1;

            clock\_t final = clock();

            double tiempo\_transcurrido = 0.0;

            tiempo\_transcurrido += (double)(final - comienzo) / CLOCKS\_PER\_SEC;

            t\_ms[i] = tiempo\_transcurrido / pow(10,-3);

            t\_min = t\_ms[0];

            if(t\_ms[i] > t\_max)

            {

                t\_max = t\_ms[i];

            }

            if(t\_ms[i] < t\_min)

            {

                t\_min = t\_ms[i];

            }

            /\*

                Finalmente se muestran por pantalla para cada uno de los paquetes recibidos

                los siguientes datos por pantalla. IP del servidor, número de bytes recibidos,

                tiempo transcurrido y TTL(Time-To-Life)

            \*/

            printf("Respuesta desde: %s bytes: %d tiempo: %.3fms TTL = 115 \n", inet\_ntoa(datos\_servidor.sin\_addr),lg\_msg\_recibido, t\_ms[i]);

        }

    }

    /\*

        Se realiza un bucle for para sumar en cada iteración el valor de cada uno de

        los tiempos obtenidos entre el envio y la recepción del mensaje, para finalmente dividirlo

        entre el número total de mensajes enviados y obtener el tiempo medio.

        Se realiza también el cálculo de porcentaje de perdida.

        Finalmente, se muestran los datos de las estadísticas por pantalla y por último se cierra

        la conexión del socket.

    \*/

    for(int i = 0; i < 4; i++)

    {

        t\_medio += t\_ms[i];

    }

    double perc\_perdida = (paquetes\_perdidos \* 100)/paquetes\_recibidos;

    double media = t\_medio/4;

    printf("\nEstadísticas de ping para: %s:\n", inet\_ntoa(datos\_servidor.sin\_addr));

    printf("\tPaquetes: Enviados = %d, Recibidos = %d, Perdidos = %d (%.2f%% perdida),\n", paquetes\_enviados, paquetes\_recibidos, paquetes\_perdidos, perc\_perdida);

    printf("Tiempo aproximado de vuelta en mili-segundos:\n");

    printf("\tMinimo = %.3fms, Máximo = %.3fms, Media = %.3fms\n\n", t\_min, t\_max, media);

    close(sockfd);

}

* **Código del servidor en modo conectivo (TCP) en lenguaje C.**
* #include <netinet/in.h>
* #include <arpa/inet.h>
* #include <sys/types.h>
* #include <sys/socket.h>
* #include <stdio.h>
* #include <stdlib.h>
* #include <netdb.h>
* #include <unistd.h>
* #include <string.h>
* #include <time.h>
* #define LONG\_PAQUETE 32
* #define N\_PETICIONES 4
* //Creación del socket
* int main(int argc, char \*argv[])
* {
* //Creacion de la estructura sockaddr\_in inicializada a 0.
* struct sockaddr\_in datos\_servidor, datos\_cliente;
* // Se rellena la estructura de los datos del servidor con 0's.
* bzero((char\*) &datos\_servidor, sizeof(datos\_servidor));
* bzero((char\*) &datos\_cliente, sizeof(datos\_cliente));
* /\*
* En la siguientes lineas se crean una serie de variables que calculan la longitud
* de la estructura datos\_servidor y datos\_cliente.
* También se crean variables para el envio y la recepción de los mensajes de un tamaño
* definido previamente en #define LONG\_PAQUETE. Además de las correspondientes variables para tener
* almacenado el tamaño correspondiente de cada mensaje en bytes (dividido entre 8).
* También se crean variables para almacenar en un array los tiempo entre envio y recepción del paquete.
* Por último se crean variables locales para el cálculo de las estadísticas del envio de los
* paquetes, etnre otras funciones.
* \*/
* int long\_serv = sizeof(datos\_servidor);
* int long\_cliente = sizeof(datos\_cliente);
* char msg\_recv[LONG\_PAQUETE];
* char msg\_enviado[LONG\_PAQUETE];
* memset(msg\_enviado,0,LONG\_PAQUETE);
* int rd;
* int n\_sockfd = 0;
* /\*
* Se guarda el puerto introducido por la linea de comandos y
* se convierte a entero meidante atoi() para agregarlo posteriormente
* a la estructura sockaddr\_in declarada como datos\_servidor.
* \*/
* char \*puerto\_char = argv[1];
* int puerto = atoi(puerto\_char);
* /\*
* Respecto a los datos del servidor, se utiliza la familita AF\_INET especificando así que se va a utilizar
* TCP/IP. Se agrega a la estructura el valor del puerto y también la dirección IP.
* \*/
* datos\_servidor.sin\_family = AF\_INET;
* datos\_servidor.sin\_port = htons(puerto);
* datos\_servidor.sin\_addr.s\_addr = inet\_addr("192.168.2.3");
* /\*
* Se verifica que el numero de argumentos pasado por la linea de ordenes sea menor que 2,
* es decir, el nombre del programa y el puerto donde se desea que escuche el servidor.
* En caso de que se introduzca de forma correcta, se verifica que el puerto este comprendido entre
* los rangos 1025-5000.
* \*/
* if(argc > 2)
* {
* printf("El número de argumentos pasado debe ser 'nombre del programa' y 'puerto de escucha'");
* return 0;
* }
* else
* {
* if(puerto < 1023 || puerto > 5000)
* {
* printf("El número de puerto escogido deber ser 1024-5000\n");
* return -1;
* }
* }
* // Creación de un socket AF\_INET(DOMINIO INTERNET TCP/IP), en modo no conectivo SOCK\_DGRAM (UDP)
* //En caso de fallo devuelve -1, si todo va bien devuelve un valor entero bajo.
* int sockfd = socket(AF\_INET, SOCK\_STREAM, 0);
* if(sockfd < 0)
* {
* printf("Error creando el socket\n");
* return -1;
* }
* // Se realiza la asociación de los datos del servidor con el socket.
* // En caso de error devuelve '-1'.
* int asociacion = bind(sockfd,(struct sockaddr \*) &datos\_servidor, long\_serv);
* if(asociacion < 0)
* {
* printf("Error en asociación 'bind'. Cerrando conexión...\n");
* // Se cierra la conexión
* close(sockfd);
* return -1;
* }
* else
* {
* printf("Asociación del socket creada.\n");
* printf("Escuchando por la dirección IP  : %s\n", inet\_ntoa(datos\_servidor.sin\_addr));
* printf("Por el puerto: %d\n\n", puerto);
* }
* /\*
* La función listen se utlizar para aceptar conexiones entrante.
* Toma como argumentos el descriptor del socket, y el número máximo de conexiones
* en cola que se permiten establecer para ese socket. Por defecto se han establecido
* un máximo de 4 peticiones.
* Devuelve un valor negativo en caso de error.
* \*/
* int list = listen(sockfd, N\_PETICIONES);
* if(list < 0)
* {
* printf("Error en la espera de peticiones\n");
* close(sockfd);
* }
* /\*
* A continucación se emplea un bucle do...while para que el servidor
* se encuentre escuchando excepto si se produce algun error en la función accept.
* La función accept lo que permite es aceptar las conexiones que esta escuchando
* el servidor. Esta función recibe como argumentos el descriptor del socket que esta
* esperando las conexiones, un puntero a una estructura de datos que almacena los datos
* del cliente que se conecta al socket. Es una función bloqueante, es decir, se mantiene
* a la espera hasta que recibe peticiones.
* Cuando se recibe una conexión entrante devuelve un nuevo descriptor de socket
* y que es la que indica el establecimiento de la conexión con el cliente.
* Este nuevo descriptor de socket es el que se utilizara para la recepción y el envio de
* los datos.
* \*/
* do
* {
* printf("\n\nEsperando peticiones...\n\n");
* n\_sockfd = accept(sockfd, (struct sockaddr \*) &datos\_cliente, &long\_cliente);
* if (n\_sockfd < 0)
* {
* printf("Error en la petición procedente de %s\n", inet\_ntoa(datos\_cliente.sin\_addr));
* break;
* }
* else
* {
* printf("\tAceptando conexión procedente de %s\n\n", inet\_ntoa(datos\_cliente.sin\_addr));
* }
* /\*
* A continuación se emplea otro bucle do...while por si ocurriera algun error
* en la recepción de los datos mediante read(). En caso de que ocurriera algun error, se cerraría la
* conexión con el NUEVO descriptor del socket, manteniendose abierto el descriptor
* que almacena las peticiones de conexión.
* \*/
* do
* {
* rd = read(n\_sockfd, &msg\_recv, sizeof(msg\_recv));
* if (rd <= 0)
* {
* printf("\tCerrando conexión TCP con el cliente ubicado en %s\n", inet\_ntoa(datos\_cliente.sin\_addr));
* close(n\_sockfd);
* break;
* }
* else
* {
* printf("\t\nLeyendo datos de %s\n", inet\_ntoa(datos\_cliente.sin\_addr));
* }
* /\*
* Una vez se han leido los mensajes enviados por el cliente se procede
* al envio de los datos del servidor al cliente.
* La función write() recibe como argumentos el nuevo descriptor de socket,
* un puntero al buffer de datos que se van a enviar, y la cantidad de datos
* que se van a enviar a través del buffer de datos.
* \*/
* int wr = write(n\_sockfd, &msg\_enviado, LONG\_PAQUETE);
* if(wr < 0)
* {
* printf("\nError al enviar el mensaje al cliente ubicado en %s\n\n", inet\_ntoa(datos\_cliente.sin\_addr));
* }
* else
* {
* printf("\n\tEnviando el mensaje de tamaño %ld bytes al cliente ubicado en %s\n", sizeof(msg\_enviado), inet\_ntoa(datos\_cliente.sin\_addr));
* }
* } while (rd != -1);
* } while (n\_sockfd != -1);
* }
* **Código del cliente en modo no conectivo (UDP) en lenguaje Python:**

import sys

import socket

import time

import array

"""

Almacenamos en ip\_serv el segundo parametro que pasamos al ejecutar el programa,

que será la dir IP. (El primero es el nombre del programa)

Almacenamos en port\_serv el tercer parametro que pasamos al ejecutar el programa, que será el puerto.

Metemos en address\_serv Dir IP y puerto en una sola cadena que usaremos para la funcion connect

"""

ip\_serv= sys.argv[1]

port\_serv= int(sys.argv[2])

address\_serv=(ip\_serv,port\_serv)

tiempo\_total=0

paquetes\_enviados=0

paquetes\_recibidos=0

paquetes\_perdidos=0

tiempos = []

t\_max = 0

t\_min = 0

tiempo\_total = 0

i = 0

#Comprobamos que se le han pasado el numero adecuado de argumentos

if(len(sys.argv)!=3):

    print('El número de parametros que me has pasado no ha sido el correcto')

    sys.exit()

#Comprobacion del puerto adecuado

if(port\_serv < 1024 or port\_serv > 5000):

    print("El puerto que se debe elegir debe estar comprendido en el rango 1024-5000")

    sys.exit()

#Creacion del socket. AF\_INET indica uso de direcciones IPv4. SOCK\_STREAM para que sea orientado a conexion

s=socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

print("\nSocket creado\n")

print("Haciendo ping con", ip\_serv, "con 32 bytes de datos:\n")

while i<4:  #Ejecuta 4 veces el bucle

    mensaje= "32"

    try:

        """

        Con Sendto codificamos y enviamos el mensaje.

        Se inicia un contador de tiempo que se va a emplear posteriormente para el

        cálculo de estadísticas.

        Se utiliza la excepción para verificar que se envia el paquete de forma

        correcta. En caso afirmativo, se añade un contador al numero de

        paquetes\_enviados.

        En caso de error, se iniciaría la excepción añadiendo un contador

        al número de paquetes perdidos.

        """

        s.sendto(mensaje.encode(),address\_serv)

        inicio=time.time()

        paquetes\_enviados += 1

    except:

        paquetes\_perdidos += 1

    try:

        """

        Con recvfrom recibimos datos del servidor. Permitimos recibir hasta

        32 bytes de datos de una sola vez.

        Se crea una variable fin para establecer el contador en el final.

        Si todo es correcto, se añade un contador al numero de paquetes\_recibidos.

        En caso de error, se inicia la excepción, añadiendo un contador al

        número de paquetes\_perdidos.

        """

        data, address = s.recvfrom(32)

        fin=time.time()

        paquetes\_recibidos += 1

    except:

        paquetes\_perdidos += 1

    """

    En las siguientes líneas de código se realizan una serie de operaciones

    y cálculos para las estadísticas de ping.

    """

    diferencia = fin - inicio

    milisegundos = pow(10, 3)

    diferencia = diferencia \* milisegundos

    tiempo\_total += diferencia

    tiempos.append(diferencia)

    t\_min = tiempos[0]

    if (tiempos[i] > t\_max):

        t\_max = tiempos[i]

    if (tiempos[i] < t\_min):

        t\_min = tiempos[i]

    print("Respuesta por parte de", ip\_serv,": bytes=32 tiempo={:.3f}ms".format(diferencia), "TTL= 115")

    time.sleep(1.25)#Añade un retraso de 2s

    i = i + 1

tiempo\_medio = tiempo\_total / 4

Prc\_pp = (paquetes\_perdidos \* 100)/paquetes\_recibidos

print("\nEstadísticas de ping para",ip\_serv)

print("\tPaquetes: Enviados =", paquetes\_enviados, "Recibidos:", paquetes\_recibidos, "Perdidos:", paquetes\_perdidos, "(", Prc\_pp,"% perdidos),\n")

print("Tiempo aproximado de vuelta en mili-segundos")

print("\tMinimo={:.3f}ms".format(t\_min), "Máximo={:.3f}ms".format(t\_max), "Media:{:.3f}ms".format(tiempo\_medio))

s.close()

* **Código del servidor en modo no conectivo (UDP) en lenguaje Python:**
* import sys
* import socket
* port = int(sys.argv[1]) #Almacena en la variable port el parametro que se le pasa por teclado
* server\_address = ('192.168.2.3', port)
* #Comprobamos que se le han pasado el numero adecuado de argumentos
* if(len(sys.argv)!=2):
* print('El numero de parametros que me has pasado no ha sido el correcto')
* sys.exit()
* #Comprobacion del puerto adecuado
* if(port < 1024 or port > 5000):
* print("El puerto que se debe elegir debe estar comprendido en el rango 1024-5000")
* sys.exit()
* #Creacion del socket. AF\_INET indica uso de direcciones IPv4. SOCK\_STREAM para que sea orientado a conexion
* s=socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)
* print('Creacion del socket')
* #Asociacion del socket a la dirección del servidor.
* s.bind(server\_address)
* print('\nAsociacion del socket')
* print('\nEsperando escuchas...\n')
* while True:
* """
* Con recvfrom recibimos datos del servidor. Permitimos recibir hasta
* 32 bytes de datos de una sola vez.
* """
* data, address = s.recvfrom(32)
* print("Se ha conectado al servidor el ", address)
* print("\nEl cliente ha enviado",data.decode())
* #Con Sendto codificamos y enviamos el mensaje. Los parámetros que se le pasan
* #serán el mensaje que se va a enviar. Y la dirección del cliente que se ha conectado
* #que será la dirección IP donde se van a enviar los datos.
* mensaje= data.decode()
* s.sendto(mensaje.encode(),address)
* print("\nMensaje enviado")
* s.close()
* **Código del cliente ping en modo conectivo (TCP) en lenguaje Python:**

import sys

import socket

import time

import numpy as np

ip\_serv= sys.argv[1] #Almacenamos en ip\_serv el segundo parametro que pasamos al ejecutar el programa, que será la dir IP. (El primero es el nombre del programa)

port\_serv= int(sys.argv[2]) #Almacenamos en port\_serv el tercer parametro que pasamos al ejecutar el programa, que será el puerto.

address\_serv=(ip\_serv,port\_serv) #Metemos en address\_serv Dir IP y puerto en una sola cadena que usaremos para la funcion connect

tiempo\_total=0

paquetes\_enviados=0

paquetes\_recibidos=0

paquetes\_perdidos=0

tiempos = []

t\_max = 0

t\_min = 0

i = 0

#Comprobamos que se le han pasado el numero adecuado de argumentos

if(len(sys.argv)!=3):

    print('El numero de parametros que me has pasado no ha sido el correcto')

    sys.exit()

#Comprobacion del puerto adecuado

if(port\_serv < 1024 or port\_serv > 5000):

    print("El puerto que se debe elegir debe estar comprendido en el rango 1024-5000")

    sys.exit()

#Creacion del socket. AF\_INET indica uso de direcciones IPv4. SOCK\_STREAM para que sea orientado a conexion

s=socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

print('\nCreacion del socket')

s.connect(address\_serv) #A traves de address\_serv indicamos a que dir IP y puerto queremos que se conecte

print("Haciendo ping con", ip\_serv, "con 32 bytes de datos:\n")

while i<4:

    # Enviar datos al servidor

    mensaje = "32"

    try:

        s.send(mensaje.encode()) #Codificamos y enviamos el mensaje

        inicio=time.time()

        paquetes\_enviados += 1

    except:

        paquetes\_perdidos += 1

    try:

        # Recibir datos del servidor

        data = s.recv(32).decode() #Utilizamos recv para recibir mensajes del servidor. Permitimos recibir hasta 1024 Bytes de datos de una sola vez.

        fin=time.time()

        paquetes\_recibidos += 1

    except:

        paquetes\_recibidos += 1

    diferencia = fin - inicio

    milisegundos = pow(10, 3)

    diferencia = diferencia \* milisegundos

    tiempo\_total += diferencia

    tiempos.append(diferencia)

    t\_min = tiempos[0]

    if (tiempos[i] > t\_max):

        t\_max = tiempos[i]

    if (tiempos[i] < t\_min):

        t\_min = tiempos[i]

    print("Respuesta por parte de", ip\_serv,": bytes=32 tiempo={:.3f}ms".format(diferencia), "TTL= 115")

    time.sleep(1.25)#Añade un retraso de 2s

    i = i + 1

tiempo\_medio = tiempo\_total / 4

Prc\_pp = (paquetes\_perdidos \* 100)/paquetes\_recibidos

print("\nEstadísticas de ping para",ip\_serv)

print("\tPaquetes: Enviados =", paquetes\_enviados, "Recibidos:", paquetes\_recibidos, "Perdidos:", paquetes\_perdidos, "(", Prc\_pp,"% perdidos),\n")

print("Tiempo aproximado de vuelta en mili-segundos")

print("\tMinimo={:.3f}ms".format(t\_min), "Máximo={:.3f}ms".format(t\_max), "Media:{:.3f}ms".format(tiempo\_medio))

# Cerrar la conexión

s.close()

print("\nConexión cerrada.")

* **Código del servidor ping en modo conectivo (TCP) en lenguaje Python:**
* import sys
* import socket
* port = int(sys.argv[1]) #Almacena en la variable port el parametro que se le pasa por teclado
* server\_address = ('192.168.2.3', port)
* #Comprobamos que se le han pasado el numero adecuado de argumentos
* if(len(sys.argv)!=2):
* print('\nEl numero de parametros que me has pasado no ha sido el correcto')
* sys.exit() #Se sale del programa
* #Comprobacion del puerto adecuado
* if(port < 1024 or port > 5000):
* print("El puerto que se debe elegir debe estar comprendido en el rango 1024-5000")
* sys.exit()
* #Creacion del socket. AF\_INET indica uso de direcciones IPv4. SOCK\_STREAM para que sea orientado a conexion
* s=socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)
* print('Creacion del socket')
* #Asociamos el socket a una dir IP y puerto especificos. Pasandole la cadena server\_address que almacena dir IP y puerto del servidor
* s.bind(server\_address)
* print('Asociacion del socket')
* print('Esperando peticiones...\n')
* s.listen(4) #Indicamos que permita hasta 4 conexiones en espera/cola
* while True:
* #Accept acepta una conexion entrante. Devuelve un socket nuevo. Y la dir IP y puerto del cliente que se ha conectado, que almacenamos en client\_address
* n\_socket, direccion\_cliente = s.accept() #Genera socket nuevo client\_socket. client\_address almacena la dir IP y puerto del cliente
* print('\nSe ha conectado el cliente ', direccion\_cliente, '\n')
* while True:
* datos = n\_socket.recv(32) #Utilizamos recv para recibir mensajes del cliente. Permitimos recibir hasta 32 Bytes de datos de una sola vez.
* mensaje = "32"
* if datos:
* n\_socket.send(mensaje.encode())    #Codificamos y enviamos el mensaje
* print('Mensaje enviado\n')
* else:
* print("Ningun dato más recibido por parte del cliente:", direccion\_cliente)
* print("Se cierra conexión TCP. A la espera de más peticiones...")
* break
* s.close()