```c
/*************************************************************************
Prácticas SITR: programación de sockets

Ejemplo de programación sobre sockets PF_INET de tipo STREAM (TCP)
Extremo Cliente  ------>

Compilar con:
    gcc -o tcpclt1 tcpclt1.c -lnsl  -lsocket -lposix4

Uso del programa (2):
    tcpclt1     ip_servidor     puerto_servidor
    tcpclt1     ip_servidor     puerto_servidor     "mensaje"
*************************************************************************/

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <errno.h>
#include <unistd.h>

#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>


#define DIM 1024        // tamaño buffer de datos de mensaje

// Mensaje a enviar si no se indica como parámetro
char *mensaje_def= "Este es el mensaje enviado desde el cliente SOCK_STREAM";


int main(int argc, char *argv[])
{
    int socketID;                   // identificador de dispositivo para el socket
    struct sockaddr_in datosSocketSrv;  // dirección IP, tipo y puerto del servidor
    struct sockaddr_in datosSocketClt;  // dirección IP, tipo y puerto del cliente
    struct hostent *hp;             // dirección IP a partir de nombre simbólico
    ssize_t res;                    // resultado de un sento()
    size_t longdir;                 // tamaño estructura socaddr_in

    char mensaje[DIM];              // buffer memoria para los mensajes


    // Comprueba los parámetros
    //*****************************************************************************
    if(argc<3)
    {
        printf("Uso del programa: tcpclt1   servidor  puerto  <mensaje>\n");
        exit(0);
    }
    if(argc==4)     strcpy(mensaje,argv[3]);        // mensaje pasado como parámetro
    else            strcpy(mensaje,mensaje_def);    // mensaje por defecto


/*
1ª etapa:  Creamos un socket PF_INET de tipo SOCK_STREAM (obtenemos un manejador)
*************************************************************************
*/
    socketID = socket( PF_INET, SOCK_STREAM, 0);
    if(socketID<0)
    {
        perror("Creando socket");
        exit(-1);
    }

/*
2ª etapa: Obtenemos la dirección IP y puerto del servidor a partir de los
parámetros
*************************************************************************
*/
    // convierte el nombre simbólico a una dirección IP
    hp=gethostbyname(argv[1]);
    if(hp == 0)
    {
```



```c
        fprintf(stderr, "\n %s: host desconocido\n", argv[1]);
        close(socketID);
        exit(-2);
    }
    // copiamos la dirección IP en la estructura datosSocket
    memcpy( &(datosSocketSrv.sin_addr),  hp->h_addr, hp->h_length);
    datosSocketSrv.sin_family=AF_INET; // tipo de dirección
    // convierte el puerto a un número de red (htons)
    datosSocketSrv.sin_port = htons(atoi(argv[2]));

/*
3ª etapa:  Establecemos la conexión
*************************************************************************/
    res = connect(socketID, (struct sockaddr *) &datosSocketSrv, sizeof datosSocketSrv);
    if(res == -1)
    {
        perror("Conexión no aceptada");
        close(socketID);
        exit(-3);
    }
    // Datos del puerto del cliente (asignados automáticamente tras la conexión)
    longdir=sizeof datosSocketClt;
    res = getsockname(socketID,  (struct sockaddr *) &datosSocketClt, &longdir);
    if(res==0)
        printf("---Cliente -> IP: %s,  Puerto: %u---\n\n",
            inet_ntoa(datosSocketClt.sin_addr),     ntohs(datosSocketClt.sin_port));

/*
4ª etapa:  Enviamos información al servidor. Opciones:
ssize_t send(int socket, const void *buffer,  size_t length, int flags);
ssize_t write(int fildes, const void *buf, size_t nbytes)
*************************************************************************
*/
    res = send(socketID, mensaje, strlen(mensaje)+1, 0);
    if(res>0)
    {   // muestra lo que ha enviado
        printf("%d bytes Enviados a [%s:%u]: %s\n", res,
                inet_ntoa(datosSocketSrv.sin_addr), ntohs(datosSocketSrv.sin_port), mensaje)
    }

/*
5ª etapa:   Cuando terminemos de trasmitir datos Cerrar el socket
*************************************************************************
*/
    close(socketID);
    printf("[Conexión Terminada]\n");

    exit(0);
}
```

```c
/*************************************************************************
Prácticas SITR: programación de sockets

Ejemplo de programación sobre sockets PF_INET de tipo STREAM (TCP)
------> Extremo Servidor

Compilar con:
    gcc -o tcpsrv1 tcpsrv1.c -lnsl -lsocket  -lposix4

Uso del programa (2):
        tcpsrv1             -> usa puerto por defecto (se muestra en la consola)
        tcpsrv1 puerto
*************************************************************************/

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <errno.h>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <pthread.h>

#define DIM         1024    // tamaño buffer mensaje
#define CONEXIONES  5       // número de trasnmisiones esperadas
#define PUERTO      0       // asigna automáticamente un puerto libre


int main(int argc, char *argv[])
{
    int socketID;                   // identificador de dispositivo para el socket
    int socketDialogo;              // identificador socket dialogo
    in_port_t puerto;               // puerto publico
    struct sockaddr_in datosSocketSrv;  // dirección IP, tipo y puerto del servidor
    struct sockaddr_in datosSocketClt;  // dirección IP, tipo y puerto del cliente
    struct hostent *hp;             // dirección IP a partir de nombre simbólico
    ssize_t res;                    // resultado de un sento()
    size_t longdir;                 // tamaño estructura socaddr_in

    char mensaje[DIM];              // buffer memoria para los mensajes
    int cont;                       // contador de conexiones

    // Comprueba los parámetros
    //*************************************************************************
    if(argc>2)
    {
        printf("Uso del programa: tcpsrv1  <puerto>\n");
        exit(0);
    }
    if(argc==2)     puerto=atoi(argv[1]);   // puerto pasado como parámetro
    else            puerto=PUERTO;          // puerto por defecto


/*
1ª etapa:
Creamos un socket PF_INET de tipo SOCK_STREAM (obtenemos un manejador)
************************************************************************
*/
    socketID = socket( PF_INET, SOCK_STREAM, 0);
    if(socketID<0)
    {
        perror("Creando socket"); exit(-1);
    }

/*
2ª etapa:
Configuramos la estructura dirección:
puerto -> IPPORT_RESERVED+XX selección fija
la dirección IP se toma la suya propia INADDR_ANY ya que es el servidor
************************************************************************
*/
    datosSocketSrv.sin_family=AF_INET;          // tipo de dirección
    datosSocketSrv.sin_port = puerto;           // convierte el puerto a un número de red (h
```



```c
    datosSocketSrv.sin_addr.s_addr = htonl(INADDR_ANY); // convierte la dirección IP a un nú

/*
 3ª etapa:
Nombramos el socket asignándole el puerto y dirección
************************************************************************
*/
    res = bind(socketID, (struct sockaddr *) &datosSocketSrv, sizeof datosSocketSrv);
    if(res<0)
    {
        perror("Nombrando el socket");
        close(socketID);
        exit(-2);
    }

    // Pedimos los datos del puerto asignado
    longdir=sizeof datosSocketSrv;
    res = getsockname(socketID, (struct sockaddr *) &datosSocketSrv, &longdir);
    if(res<0)
    {
        perror("Leyendo información  del socket");
        close(socketID);
        exit(-2);
    }
    // muestra los datos del servidor
    printf("---Servidor IP: %s,  Puerto: %u---\n",
        inet_ntoa(datosSocketSrv.sin_addr),ntohs(datosSocketSrv.sin_port));

/*
4ª etapa:
Poner en escucha el servidor con una cola de 5 mensajes
cuando aperece una conexión aparece un nuevo socket
************************************************************************
*/
    listen(socketID, CONEXIONES);
    // aceptar hasta 5 conexiones
    for(cont=0; cont<CONEXIONES; cont++)
    {
        printf("\nEsperando Conexión (%d) ....\n", cont+1);

        longdir = sizeof datosSocketClt;
        socketDialogo = accept(socketID, (struct sockaddr *) &datosSocketClt, &longdir);

        // se ha creado un nuevo socket para la conexión en curso
        if(socketDialogo <0)
        {
            perror("Conexión no aceptada");
            continue;
        }
        /*
        5ª etapa:   Recibimos información del cliente. Opciones:
        ssize_t read(int fildes, void *buf, size_t nbytes)
        ssize_t recv(int socket, void *buffer, size_t length, int flags);
        ************************************************************************
        */
        res = recv(socketDialogo, mensaje, DIM, 0);
        if(res>0)
        {
            // muestra lo que ha leido
            printf("[Conexion %d, Recibido (%d bytes) desde %s:%u <-] %s\n", cont+1, res,
                        inet_ntoa(datosSocketClt.sin_addr), ntohs(datosSocketClt.sin_port),
        }
        close(socketDialogo); // cierra la conexión: socket de dialogo
        printf("[Conexión %d Terminada]\n", cont+1);
    }
/*
6ª etapa:
Cerrar el socket de Escucha
************************************************************************
*/
    close(socketID);
    printf("[Servidor Terminado]\n");
    exit(0);
}
```