CLIENTE TCP – SOCKETS BSD

```
/*
 * File: cliente.c
 */

#include <stdio.h>
#include <malloc.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <netdb.h>
#include <arpa/inet.h>
#include <sys/time.h>

#include <stdlib.h>
#include <stdio.h>


/*
 * Cliente TCP
 */
int
main(int argc, char** argv) {

   int socket_datos;
   struct sockaddr_in sockname;
   char buffer[82];

   // Obteniendo parámetros de la línea de comandos...
   if (argc!=3)
      perror("Sintaxis: cliente IP_servidor puerto_servidor"),
      exit(1);
```

```
   char *servidor = argv[1]; // Dirección IP del servidor
   int port = atoi(argv[2]);   // Puerto de escucha del servidor

// Creando el socket de datos...
   if((socket_datos=socket(AF_INET,SOCK_STREAM,0))==-1)
      perror("Cliente: error en la llamada a la función socket"),exit(1);

   // Asignando puerto y dirección...
   sockname.sin_family=AF_INET;
   sockname.sin_addr.s_addr=inet_addr(servidor);
   sockname.sin_port=htons(port);

   // Conectándose con el servidor
   if(connect(socket_datos,(struct sockaddr *) &sockname,
                  sizeof(sockname))==-1)
      perror("Cliente: error en la llamada a la función connect"),exit(1);

   // Bucle para enviar mensajes hasta introducir "FIN"...
   do{
      printf("Teclee el mensaje a transmitir:\n");
      gets(buffer);

      printf ("Has tecleado: %s\n", buffer);

      // Mandando datos a través del socket...
      if(send(socket_datos,buffer,80,0)==-1)
         perror("Cliente: error en la llamada a la función send"),exit(1);
   }while(strcmp(buffer,"FIN")!=0);

   // Cerrando el socket de datos...
   close(socket_datos);

exit (0);
}
```

SERVIDOR ITERATIVO TCP – SOCKETS BSD

```
/*
 * File: servidor.c
 */

#include <stdio.h>
#include <malloc.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <netdb.h>
#include <arpa/inet.h>
#include <sys/time.h>

#include <stdlib.h>

/*
 * Servidor TCP iterativo
 */
int
main(int argc, char** argv) {

   int socket_control, from_len, socket_datos;
   struct sockaddr_in from,sockname;
   char buffer[82];
   int salir=0;

   // Obteniendo parámetros de la línea de comandos...
   if (argc!=2)
      perror("Sintaxis: servidor puerto_servidor"),exit(1);

   int port = atoi(argv[1]);

   // Creando el socket de control (para aceptar conexiones)...
   if((socket_control=socket(AF_INET,SOCK_STREAM,0))==-1)
      perror("Servidor: error en la llamada a la función socket"),exit(1);

   // Asignando dirección y puerto...
   sockname.sin_family=AF_INET;
   sockname.sin_addr.s_addr=INADDR_ANY;
   sockname.sin_port=htons(port);

   // Anunciándose como servidor...
   if(bind(socket_control,(struct sockaddr *) &sockname,sizeof(sockname))==-1)
      perror("Servidor: error en la llamada a la función bind"),exit(1);

   // Diciendo que será un socket de escucha...
   if(listen(socket_control,1)==-1)
      perror("Servidor1: error en la llamada a la función listen"),exit(1);

   // Bucle infinito para aceptar peticiones...
   do{

      // Aceptando conexiones de los diferentes clientes
      // (creándose un nuevo socket de datos)...
      from_len=sizeof(from);
      socket_datos=accept(socket_control,(struct sockaddr *) &from,&from_len)
      if(socket_datos==-1)
         perror("Servidor: error en la llamada a la función accept"),exit(1);

      // Recibiendo un mensaje y escribiéndolo en pantalla (hasta recibir FIN)...
      do{
         int nbytes = recv(socket_datos,buffer,80,0);
         if(nbytes==-1)
            perror("Servidor:Recv"),exit(1);

         if (nbytes==0)
            perror("El cliente se ha desconectado"),exit(1);

         printf("El mensaje recibido fue:\n%s\n",buffer);
      }while(strcmp(buffer,"FIN")!=0);

      // Cerrando el socket de datos...
      close (socket_datos);
   }while(!salir);

   // Cerrando el socket de control...
   close (socket_control);

   // Terminando el programa...
   exit(0);
}
```

SERVIDOR CONCURRENTE TCP – SOCKETS BSD

```
/*
 * File:   servidorconcurrente.c
 */

#include <stdio.h>
#include <malloc.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <netdb.h>
#include <arpa/inet.h>
#include <sys/time.h>

#include <stdlib.h>

/*
 * Servidor TCP concurrente
 */
int
main(int argc, char** argv) {

   int socket_control, from_len, socket_datos;
   struct sockaddr_in from,sockname;
   char buffer[82];
   int salir=0;

   // Obteniendo parámetros de la línea de
   // comandos...
   if (argc!=2)
      perror("Sintaxis: servidor
               puerto_servidor"),exit(1);

   int port = atoi(argv[1]);

   // Creando el socket de control (para aceptar
   // conexiones)...
   socket_control=socket(AF_INET,SOCK_STREAM,0);
   if((socket_control==-1)
      perror("Servidor: error en la llamada a la
               función socket"),exit(1);

// Asignando dirección y puerto...
sockname.sin_family=AF_INET;
sockname.sin_addr.s_addr=INADDR_ANY;
sockname.sin_port=htons(port);

// Anunciándose como servidor...
if(bind(socket_control,(struct sockaddr *)
            &sockname,sizeof(sockname))==-1)
   perror("Servidor: error en la llamada a la
            función bind"),exit(1);

// Diciendo que será un socket de escucha...
if(listen(socket_control,1)==-1)
   perror("Servidor: error en la llamada a la
            función listen"),exit(1);

// Bucle infinito para aceptar peticiones...
do{

   // Aceptando conexiones de los diferentes
   // clientes (creándose un nuevo socket de
   // datos)...
   from_len=sizeof(from);
   if((socket_datos=accept(socket_control,
     (struct sockaddr *) &from,&from_len))==-1)
      perror("Servidor: error en la llamada a la
               función accept"),exit(1);

   // Creación de un proceso hijo para atender al
   // cliente que se conectó...
   int pid;   // Identificador del proceso padre
   if ((pid=fork())==0){
      // Proceso hijo =
      // PROCESO DE ATENCIÓN AL CLIENTE

   // Recibiendo un mensaje y escribiéndolo en
   // pantalla (mientras no se reciba FIN)...
      do{
         int nbytes=recv(socket_datos,buffer,80,0);
         if(nbytes==-1)
            perror("Servidor: error en la llamada a la
                     función recv"),exit(1);

         if (nbytes==0)
            perror("El cliente se ha desconectado"),
            exit(1);

         printf("El mensaje recibido fue:\n%s\n",
                  buffer);

      }while(strcmp(buffer,"FIN")!=0);

      // Cerrando el socket de datos...
      close (socket_datos);

      // Finalización del proceso de atención al
      // cliente...
      exit(0);

   } else {
      // Proceso padre =
      // PROCESO QUE ESPERA PETICIONES

      // No hace nada, el bucle hará que vuelva a
      // esperar una petición (accept()).
   }

}while(!salir);

// Cerrando el socket de control...
close (socket_control);

// Terminando el programa...
exit(0);
}
```

CLIENTE TCP – JAVA

```
/**
 * <p>Title: Mínimo cliente TCP</p>
 * <p>Description: </p>
 * <p>Copyright: Copyright (c) 2007</p>
 * <p>Company: UGR</p>
 * @author not attributable
 * @version 1.0
 */

import java.net.*;
import java.io.*;

public class MinimoClienteTCP {

  // Atributos de la clase:
  static Socket socket_datos;
  static String direccionServidor; // Nombre o dirección IP
  static int puerto;
  static PrintWriter out;
  static BufferedReader in;

  public MinimoClienteTCP() {
  }

  public static void main (String args[]) {
    boolean error=false;
    String mensajeSolicitud;
    String mensajeRespuesta = "";
```

```
// Se piden 3 argumentos: dirección del servidor, puerto y mensaje a enviar.
  if (args.length<3) {
    System.err.println("Sintaxixs: MinimoClienteTCP <direccion-servidor>
                        <puerto> <mensaje a enviar>");
    System.exit(-1);
  }

  // Dirección (IP o nombre) del servidor
  direccionServidor = args[0];
  // Puerto
  puerto = Integer.parseInt(args[1]);
  // Mensaje a enviar
  mensajeSolicitud = args[2];

  // 1 - Se abre el socket y se conecta a la dirección y puerto del servidor.
  try {
    socket_datos = new Socket (direccionServidor, puerto);

    // Se obtienen los flujos de lectura y escritura para recibir y enviar
    // mensajes.
    out = new PrintWriter (socket_datos.getOutputStream(), true);
    in = new BufferedReader (new
                        InputStreamReader(socket_datos.getInputStream()));
  } catch (UnknownHostException e) {
    System.err.println ("Error: no se pudo encontrar al servidor " +
                        direccionServidor);
    System.exit(-2);
  } catch (IOException e) {
    System.err.println("Error: no se pudo establecer la conexión con el
                        servidor");
  }

  // (Continua en la siguiente transparencia …)
```

CLIENTE TCP – JAVA (continuación)

```
// (… continuación de MinimoClienteTCP)

 // 2 - Se escribe el mensaje de solicitud y leemos la respuesta:
 out.println(mensajeSolicitud);
 try {
  mensajeRespuesta = in.readLine();
 } catch (IOException e) {
  System.err.println ("Error: no se pudo leer la respuesta.");
 }

 // 3 - Se cierra la conexión.
 try {
  in.close();
  out.close();
  socket_datos.close();
 } catch (IOException e) {
  System.err.println ("Error: no se pudo cerrar la conexión.");
 }

 // Se muestra la respuesta:
 System.out.println("El mensaje enviado fue:  " + mensajeSolicitud);
 System.out.println("El mensaje recibido fue: " + mensajeRespuesta);

 }

}
```

SERVIDOR ITERATIVO TCP – JAVA

```
/**
 * <p>Title: Mínimo servidor iterativo TCP</p>
 */

import java.net.*;
import java.io.*;

public class MinimoServidorTCP {

  // Atributos de la clase:
  static ServerSocket socket_control;
  static Socket socket_datos;
  static int puerto;
  static PrintWriter out;
  static BufferedReader in;

  public MinimoServidorTCP() {
  }

  public static void main (String args[]) {
    boolean salir = false;
    boolean error = false;
    String mensajeSolicitud;
    String mensajeRespuesta;


    // El argumento es el puerto donde se
    // atenderá el servicio.
    if (args.length<1) {
      System.err.println("Sintaxis:
                  MinimoServidorTCP <puerto>");
      System.exit(-1);
    }

    // Se obtiene el puerto:
    puerto = Integer.parseInt(args[0]);
```

```
// 1 - Se abre el socket en modo "escucha"
    try {
      socket_control = new ServerSocket (puerto);
    } catch (IOException e) {
      System.err.println("Error: no se puede abrir el puerto
                          indicado.");
      System.exit(-2);
    }

    // Es un servidor iterativo: acepta una conexión, la
    // procesa y la cierra. Después se acepta otra conexión
    // y así sucesivamente.

    do {

      // 2 - Se bloquea la hebra actual en "accept", y se
      //     devuelve la conexión establecida con el cliente.

      try {
        socket_datos = socket_control.accept();

        // Se obtienen los flujos de entrada y salida para
        // recibir y enviar mensajes.
        try {
          out = new PrintWriter
                  (socket_datos.getOutputStream(), true);
          in = new BufferedReader (new InputStreamReader
                        (socket_datos.getInputStream()));
        } catch (IOException e) {
          System.err.println("Error: no se pudo obtener un
                              canal para los flujos");
          error = true;
        }
```

```
        // 3 - Código del servicio ofrecido.

        // 3a) Se lee una línea del cliente
        mensajeSolicitud = in.readLine();

        // 3b) Se aplica el servicio:
        mensajeRespuesta =
                    procesaServicio(mensajeSolicitud);

        // 3c) Se envía la respuesta:
        out.println(mensajeRespuesta);

      } catch (IOException e) {
        System.err.println("Error: no se pudo aceptar la
                            solicitud de una conexión");
        error = true;
      }

    } while (!salir);

  }


  static String procesaServicio (String mensaje) {

    // Aquí se podría poner el código que procesara el
    // mensaje recibido. Actualmente sólo lo devuelve tal
    // cual vino.

    return mensaje;
  }

}
```

SERVIDOR CONCURRENTE TCP – JAVA

```
/**
 * <p>Title: Mínimo servidor concurrente TCP</p>
 * <p>Description: </p>
 * <p>Copyright: Copyright (c) 2007</p>
 * <p>Company: UGR</p>
 * @author not attributable
 * @version 1.0
 */

import java.net.*;
import java.io.*;

public class MinimoServidorConcurrenteTCP {
 // Atributos de la clase
 static ServerSocket socket_control;
 static Socket socket_datos;
 static int puerto;
 static Servicio servicio;
 // Hebras para que el servicio sea concurrente

 public MinimoServidorConcurrenteTCP() {
 }

 static public void main (String args[]) {
  boolean salir = false;
  boolean error = false;

  // Al menos un argumento, el puerto.
  if (args.length<1) {
   System.err.println ("Sintaxis:
            MinimoServidorConcurrenteTCP <puerto>");
   System.exit(-1);
  }

  // Se obtiene el puerto.
  puerto = Integer.parseInt(args[0]);

  // 1 - Se abre el socket en modo "escucha".
  try {
   socket_control = new ServerSocket (puerto);
  } catch (IOException e) {
   System.err.println ("Error: no se puede abrir el
                        puerto indicado.");
   System.exit(-2);
  }

  // Bucle para aceptar conexiones. Por cada conexión
  // aceptada se creará una hebra a la que se le pasará el
  // socket para cursar el servicio.
  do {

   // 2 - Se bloquea la hebra actual en "accept"
   //   esperando una solicitud de conexión. Se devuelve
   //   un nuevo socket con la conexión establecida.
   try {
    socket_datos = socket_control.accept();

    // Se lanza una hebra para que sirva a este cliente
    // por "socket_datos".
    new Servicio(socket_datos).start();
   } catch (IOException e) {
    System.err.println("Error: no se pudo aceptar la
                        solicitud de una conexión.");
    error = true;
   }

  } while (!salir);
 }

}

class Servicio extends Thread {
 // Atributos de la clase
 Socket socket_datos;
 PrintWriter out;
 BufferedReader in;

 // El constructor recibirá como argumentos el socket
 // (abierto) que debe utilizar (se lo pasa la hebra
 // principal del servidor).
 public Servicio (Socket socket_datos_) {
  socket_datos= socket_datos_;

  // Se obtienen los flujos de lectura y de escritura para
  // enviar y recibir mensajes.
  try {
   out = new PrintWriter
         (socket_datos.getOutputStream(), true);
   in = new BufferedReader (new InputStreamReader
         (socket_datos.getInputStream()));
  } catch (IOException e) {
   System.err.println(this.getName() + " Error: no
         se pudo obtener un canal para los flujos.");
  }
 }

public void run() {
  String mensajeSolicitud = "";
  String mensajeRespuesta = "";

  try {
   // 3 - Código del servicio ofrecido.
   // 3a - Se lee el mensaje del cliente.
   mensajeSolicitud = in.readLine();
  } catch (IOException e) {
   System.err.println(this.getName() + " Error: no
                      se pudo leer el mensaje.");
  }

  // 3b - Se aplica el servicio.
  mensajeRespuesta = procesaServicio
                     (mensajeSolicitud);

  // 3c - Se envía la respuesta.
  out.println(mensajeRespuesta);

  // 4 - Se cierra la conexión establecida con
  // el cliente.
  try {
   in.close();
   out.close();
   socket_datos.close();
  } catch (IOException e) {
   System.err.println(this.getName() + " Error: no
                      se pudo cerrar la conexión.");
  }
 }

 static String procesaServicio (String mensaje) {
  // Aquí se podría poner el código que procesara
  // el mensaje recibido.
  // Actualmente sólo lo devuelve tal cual vino.
  return mensaje;
 }
}
```

CLIENTE UDP – JAVA

```
/**
 * Este programa transmite dos mensajes.
 */

import java.net.*;
import java.io.*;

public class EmisorUDP {
  public static void main(String args [] ) {
    // Los argumentos dan:
    // el nombre de la máquina receptora y 2 mensajes
    if (args.length != 3) {
      System.err.println("Uso: java EmisorUDP maquina msj1 msj2")
;
    }
    else try{
      // Crea su socket
      DatagramSocket elSocket = new DatagramSocket();

      // Construye la dirección del socket del receptor
      InetAddress maquina = InetAddress.getByName(args[0]);
      int puerto = 1234;

      // Crea el primer mensaje
      byte [] cadena = args[1].getBytes();
      DatagramPacket mensaje = new DatagramPacket(cadena,
                          args[1].length(), maquina, puerto);

      // Envía el primer mensaje
      elSocket.send(mensaje);

      // Crea el segundo mensaje
      cadena = args[2].getBytes();
      mensaje.setData(cadena);
      mensaje.setLength(args[2].length());

      // Envía el segundo mensaje
      elSocket.send(mensaje);

      // Cierra su socket
      elSocket.close();
    } catch(UnknownHostException e) {
      System.err.println("Desconocido: " + e.getMessage());
    } catch(SocketException e) {
      System.err.println("Socket: " + e.getMessage());
    } catch(IOException e) {
      System.err.println("E/S: " + e.getMessage());
    }
  }
}
```

SERVIDOR UDP – JAVA

```
/**
 * Este programa recibe dos mensajes.
 */

import java.net.*;
import java.io.*;

public class ReceptorUDP {
  public static void main(String args [] ) {
    // Sin argumentos
    if (args.length != 0) {
      System.err.println("Uso: java ReceptorUDP");
    }
    else try{
      // Crea su socket
      DatagramSocket elSocket = new DatagramSocket(1234);

      // Crea el espacio para los mensajes
      byte [] cadena = new byte[1000] ;
      DatagramPacket mensaje1 =
                    new DatagramPacket(cadena, cadena.length);

      // Recibe y muestra el primer mensaje
      elSocket.receive(mensaje1);
      System.out.println("Mensaje Recibido: (" +
         new String(mensaje1.getData(), 0, mensaje1.getLength()) +
                      ") longitud = " + mensaje1.getLength());
      DatagramPacket mensaje2 =
                    new DatagramPacket(cadena, cadena.length);

      // Recibe y muestra el segundo mensaje
      elSocket.receive(mensaje2);
      System.out.println("Mensaje Recibido: (" +
         new String(mensaje2.getData(), 0, mensaje2.getLength())
                      + ") longitud = " + mensaje2.getLength());
    } catch(SocketException e) {
      System.err.println("Socket: " + e.getMessage());
    } catch(IOException e) {
      System.err.println("E/S: " + e.getMessage()); }
  }
}
```