Ejercicios de laboratorio

Muchas aplicaciones que utilizan sockets emplean protocolos de aplicación que se basan en enviar cadenas de caracteres entre procesos clientes y servidores. Cualquier dato se codifica en una cadena de caracteres y está se envía como petición y respuesta. Suele ser habitual que estas cadenas de caracteres se representen como un conjunto de caracteres que finalizan con el código ASCII 0 (8 bis a 0), el cual sirve como delimitador de la cadena, de igual que forma que ocurre en lenguajes de programación como C. En otras ocasiones la cadena de caracteres finaliza con el código ASCII de salto de línea (‘\n’), en este caso el protocolo está orientado al envío y recepción de líneas.

El objetivo de este laboratorio es realizar pequeñas aplicaciones con sockets TCP que utilizan cadenas de caracteres como mecanismo de comunicación.

En aula global se dispone del material de apoyo para el este laboratorio. El material incluye los siguientes archivos:

```
Makefile
lines.h
lines.c
prueba.c
cliente.c
servdor.c
cliente.java
```

El archivo `lines.c` incluye la definición de las siguientes funciones:

```
int enviar(int socket, char *mensaje, int longitud);
```

Esta función incluye el código para enviar un mensaje de una determinada longitud por un socket.

```
int recibir(int socket, char *mensaje, int longitud);
```

Esta función incluye el código para recibir un mensaje de una determinada longitud por un socket.

```
ssize_t readLine(int fd, void *buffer, size_t n);
```

Esta función lee de un descriptor (de fichero o de socket) una cadena de una longitud máxima de n bytes (incluido el código terminador de la cadena). Si se sobrepasa este número, el resto de los caracteres se descartan. La función devuelve una cadena de caracteres que finaliza con el código ASCII 0. Esta función lee una secuencia de bytes y los almacena en el buffer pasado como segundo argumento hasta que se han leído n-1 bytes. El resto de los bytes leídos o recibidos se descartan. La función finaliza retornando una cadena de caracteres (con el código terminador incluido) cuando ocurre alguna de las siguientes circunstancias:

- El descriptor de fichero hace referencia a un fichero y se alcanza el fin de fichero.
- El descriptor de fichero hace referencia a un fichero y se lee el código ASCII ‘\0’ o el salto de línea (‘\n’). La cadena devuelta no incluye el salto de línea.

- El descriptor de fichero hace referencia a un socket y se lee el código ASCII '\0' o el salto de línea ('\n'). La cadena devuelta no incluye el salto de línea.

La función devuelve además la longitud de la cadena leída. Devuelve -1 en caso de error.

```
ssize_t writeLine(int fd, void *buffer, size_t n);
```

Esta función escribe en un descriptor (de fichero o de socket) una cadena de una longitud n bytes. La función envía a continuación el carácter de salto de línea (`\n`). La función devuelve 0 en caso de éxito o -1 en caso de error.

Cuando se quiere enviar por un socket una cadena de caracteres (por ejemplo, `cadena`) incluido el código de fin de cadena (código ASCII 0) se puede utilizar la función enviar de la siguiente forma:

```
enviar(cadena, strlen(cadena)+1);
// envía todos los caracteres y el código ASCII 0.
```

El archivo `prueba.c` incluye también el código que hace uso de la función `readLine` y `writeLine`. Compile y pruebe este programa para entender el funcionamiento de las funciones anteriores. Este programa lee líneas que finalizan con el salto de línea ('\n') por la entrada estándar y las imprime por pantalla.

Ejercicio 1. El objetivo de la primera parte del laboratorio consiste en programar un cliente y un servidor utilizando sockets de tipo *stream* en C. El cliente y el servidor se envían cadenas de caracteres finalizadas con el código '`\0`'; Se proporcionan los archivos `cliente.c` y `servidor.c` que se utilizarán como base para desarrollar estos programas.

El cliente realizará las siguientes acciones:

1. El cliente recibe como argumento en la línea de mandatos la dirección y el puerto del servidor. Para probarlo de forma local con un servidor ejecutando en el puerto 2000, puede invocar al cliente de esta forma:
   ```
   ./cliente    localhost    2000
   ```
2. Se conecta con el servidor arrancado en la dirección y puerto pasados como argumentos.
3. Ejecuta un bucle infinito (del que se sale con Ctrl-c). En cada iteración lee de la entrada estándar una cadena utilizando la función `readLine` y la envía al servidor utilizando `enviar`. A continuación, recibe del servidor la misma cadena y la imprime por pantalla. El cliente leerá la cadena devuelta por el servidor utilizando `readLine` y la imprimirá por pantalla utilizando `enviar`.

El servidor (en esta primera tarea será secuencial) acepta una conexión de un cliente y entrará en un bucle en el que recibirá una cadena del cliente y le devolverá la misma cadena de vuelta al cliente. Del bucle se saldrá cuando se reciba del cliente la cadena "EXIT". El servidor recibe el puerto en el que tiene que aceptar las conexiones como argumento en la línea de mandatos.

Ejerccio 2. El objetivo de la segunda parte es convertir el código del servidor en concurrente utilizando procesos ligeros.

Ejercicio 3. En la tercera parte del laboratorio se realizará la implementación de un cliente en java que utilizará el servidor previamente implementado. Se proporciona el archivo `cliente.java` que se utilizará como base. Tenga en cuenta que un cliente programado en Java puede crear un socket *stream* ejecutando directamente la siguiente llamada:

```
Socket sc = new Socket(host, 4200);
```

La llamada anterior crea un socket y realiza directamente la conexión con proceso que ejecuta, en este caso, en la máquina `host` y puerto `4200`.

Asociado a un socket, existen dos flujos de datos que se pueden utilizar para enviar y recibir datos del socket. Por ejemplo:

```
DataOutputStream out = new DataOutputStream(sc.getOutputStream());
DataInputStream  in  = new DataInputStream(sc.getInputStream());
```

Se puede utilizar el flujo de datos `out` para enviar datos por el socket y el flujo de datos `in` para recibir datos del socket.

El siguiente fragmento de código envía una cadena de caracteres (finalizada clon el código ASCII '\0') a través del `DataOutputStream` asociado a un socket:

```
String mensaje = new String("Hola");
out.writeBytes(mensaje);
out.write('\0');      // inserta el código ASCII 0 al final
```

El fragmento que permite leer una cadena de caracteres que finaliza con el código '\0' a través del `DataInputStream` asociado a un socket es el siguiente:

```
byte[] ch = new byte[1];
String mensaje = new String();

do{
        ch[0] = in.readByte();
        if (ch[0] != '\0'){
                    String d = new String(ch);
                    mensaje = mensaje + d;
        }
}while(ch[0] != '\0');
```

Este fragmento de código lee byte a byte de un flujo de entrada `in` (`DataInputStream`) y con ellos construye el `String mensaje`.