E.T.S.I. TELECOMUNICACIÓN
FUNDAMENTOS DE SOFTWARE DE COMUNICACIONES
Laboratorio

APELLIDOS, NOMBRE:
DNI:
TITULACIÓN:
Número de PC:

Se requiere implementar una aplicación cliente/servidor en TCP para dar un servicio de cálculo de operaciones matemáticas. El cliente deberá solicitar por teclado al usuario el tipo de operación y los operandos, que enviará al servidor para que sea éste el que realice la operación solicitada. Cuando la tenga, le mandará una respuesta al cliente con el resultado. El servidor atiende peticiones de un cliente hasta que se desconecte. El formato de los mensajes con los que se intercambian datos es el siguiente:

- Del cliente al servidor, para solicitar la realización de una operación matemática:

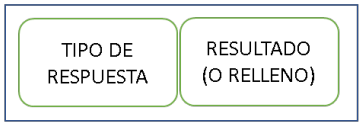
FORMATO DEL MENSAJE DE CLIENTE A SERVIDOR (OPERACIÓN)

Es un mensaje de longitud variable, donde:

  - **`uint8_t tipo de operación`**: indica el tipo de operación matemática a realizar. Sus valores pueden ser:
    - '**`s`**': para sumar.
    - '**`m`**': para multiplicar.
  - **`uint8_t número de operandos`**: número de campos de operandos que se incluyen, a continuación, en el mensaje.
  - **`int16_t operando (1 a N)`**: cada operando ocupa dos bytes. NOTA: Aunque aquí se sugiere usar **`int16_t`** (por respetar el signo), de cara a la red se manipulan igual que el **`uint16_t`**.

- Del servidor al cliente, para enviar el resultado de una operación matemática:

FORMATO DEL MENSAJE DE SERVIDOR A CLIENTE (RESPUESTA)



es un mensaje de longitud fija, 5 bytes, con estos dos campos:

  - **`uint8_t tipo`**: indica el tipo de respuesta:
    - **`1`**: indica que la operación se ha podido realizar, y el siguiente campo contiene el resultado
    - **`0`**: indica que el tipo de operación solicitada no está soportada (no era de tipo '**`s`**' ni '**`m`**'). En este caso, el campo siguiente se lee, pero no contiene un valor válido.
  - **`int32_t resultado`**: resultado de la operación, en caso de que el tipo de la respuesta sea **`1`**. En otro caso, se lee, pero se ignora su valor (da igual lo que haya dentro). NOTA: utilizar un **`int32_t`** o un **`uint32_t`** es indiferente de cara a su envío y recepción por la red. Aquí se sugiere ese tipo simplemente por respetar el signo de la operación.

La especificación detallada de cliente y servidor se incluye a continuación. En todos los apartados se debe garantizar que se lee/escribe lo esperado y que los recursos reservados se liberan correctamente.

Ejercicio 1. Servidor [1.5 puntos]

Implemente el servidor para dar soporte a la petición y cálculo de operaciones matemáticas como sigue.

1. El servidor escucha en el puerto **4950**.
2. Cuando se conecta un cliente, se inicia un proceso iterativo en el que:
    a. El servidor lee una operación que viene codificada en un mensaje con el formato **OPERACIÓN** descrito anteriormente. El mensaje llega entero, o no llega.
    b. Una vez recibida la operación, el servidor comprueba el tipo de operación:
        i. Si es correcto, es decir, que es o '**`s`**' o '**`m`**', entonces realiza la operación (una suma o una multiplicación) y construye un mensaje de respuesta, para mandar el resultado, según el formato **RESPUESTA**. Como la operación se ha realizado con éxito, el campo **`tipo`** tendrá el valor **`1`**, y el campo **`resultado`** almacenará el valor de la suma o multiplicación, según corresponda.
        ii. Si el tipo de operación no es correcta (no está soportada), entonces se construye un mensaje de **RESPUESTA** en el que el campo **`tipo`** tiene el valor **`0`** y el campo **`resultado`** llevará un relleno inservible, ceros por ejemplo.
    c. Se envía el mensaje de respuesta al cliente.
    d. Se vuelve al paso a). Este proceso se repite hasta que el cliente se desconecta. En ese momento, el servidor vuelve a esperar a que se conecte un nuevo cliente.

Se entrega el fichero **ejercicio1.c/cpp.**

Ejercicio 2. Cliente [1.5 puntos]

Implemente el cliente para este servicio, según la siguiente especificación:

1. El cliente recibe como argumento de entrada la **dirección IP** y el **puerto** del servidor.
2. Una vez conectado con el servidor, el cliente debe iterar solicitando al usuario el tipo de operación, número de operandos y valores de los operandos mientras que éste no introduzca por teclado el carácter **`f`**:

```
Introduzca tipo de operación ('f', sin comillas, para terminar):
s
Número de operandos: 3
Introduzca operando 1: 2
Introduzca operando 2: -1
Introduzca operando 3: 5
Recibida respuesta correcta del servidor
Tipo: 1
Resultado: 6
Introduzca tipo de operación ('f', sin comillas, para terminar):
f
alumno@debian:junio$
```

    a. Con la información recogida de teclado, el cliente construye un mensaje **OPERACIÓN**, y lo envía al servidor. NOTA: un mensaje válido debe incluir el campo tipo, el número de operandos y, al menos, dos operandos. Si no hay dos operandos, no se debe molestar al servidor con una operación imposible, y se solicita una nueva operación.
    b. El cliente entonces espera la respuesta del servidor según el formato descrito anteriormente:
        i. Si el campo **`tipo`** del mensaje de respuesta es **`1`**, entonces escribe por pantalla "**`Recibida respuesta correcta del servidor`**", el tipo del mensaje, y el resultado de la operación.

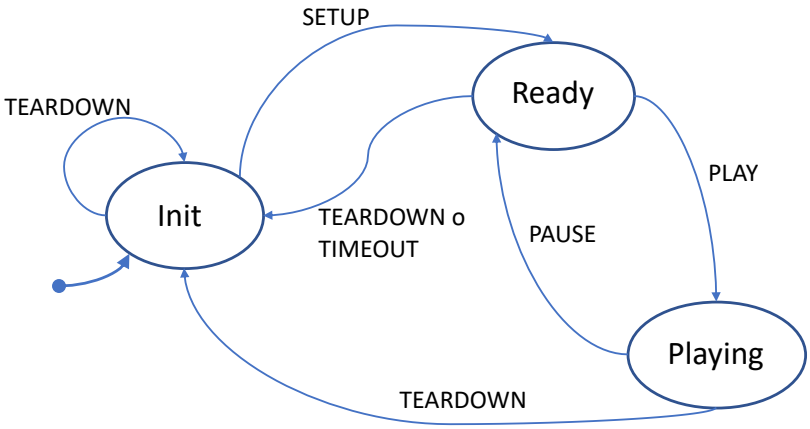
ii. Si el campo **`tipo`** es **`0`**, entonces escribe por pantalla "**`Operación no soportada en el servidor`**".

Se entrega el fichero **ejercicio2.c/cpp.**

Notas:

1. En todo caso se ha de garantizar que se lee y escribe lo esperado.
2. Se han de liberar correctamente todos los recursos reservados.
3. Sobre las variables que representan los operandos y el resultado de la operación: trabaja siempre con números enteros CON SIGNO. Piensa que, cuando se envíen y reciban por red, ocuparán el mismo tamaño que un entero sin signo, así que no te harán falta conversiones o castings.

Ejercicio 3. Máquina de estados [2.5 puntos]



La figura muestra una adaptación de la máquina de estados del protocolo para streaming en tiempo real (*Real-time Streaming Protocol*, por sus siglas en inglés), y se usa para la retransmisión de audio/video en remoto. Impleméntela siguiendo los siguientes requisitos:

- Se debe haber creado una **fifo** en el sistema de archivos, de nombre "**`fsc_fifo`**" (sin las comillas). Antes de entrar en la máquina de estados, abra la fifo en modo lectura.
- La máquina comienza en el estado **`Init`**, y permanece en él hasta que, por la **fifo**, se reciba la cadena "**`SETUP`**" (sin las comillas). Con este evento se está simulando que se ha solicitado reproducir un fichero multimedia. Se imprime por pantalla el mensaje **`Servidor de streaming listo para transmitir`**, y se pasa el estado **`Ready`**.
- En el estado **`Ready`**, la máquina espera la orden de reproducción, por lo que permanece aquí hasta que el proceso recibe el evento **`PLAY`**, que se simula con la llegada de la señal **`SIGUSR1`**. En ese momento, se simula la reproducción imprimiendo por pantalla **`Reproduciendo`**, y se pasa al estado **`Playing`**.
- La máquina también puede transitar desde el estado **`Ready`** hasta **`Init`** cuando pasen **5.5 segundos** sin que haya llegado otro evento.
- En el estado **`Playing`**, se puede pausar la reproducción cuando llega el evento **`PAUSE`**, que se simula con la llegada de la señal **`SIGUSR2`**. Entonces se activa el temporizador anterior y se transita a **`Ready`**.
- Desde cualquier estado, si llega el evento **`TEARDOWN`**, simulado por la señal **`SIGINT`**, se vuelve al estado **`Init`**.
- No se pueden perder señales, por lo que se aconseja utilizar una tubería de tipo **pipe** para encolarlas a medida que se vayan recibiendo.

- Recibir por la **fifo** una cadena distinta a "**`SETUP`**" (sin las comillas), se considera un error y se sale de la máquina.

Se entrega el fichero **ejercicio3.c/cpp.**

Código de referencia (arriba, partes de un servidor orientado a conexión; tras la línea de separación, partes de un cliente):

```
int sockfd;
struct sockaddr_in server_addr, client_addr;
socklen_t sin_size = sizeof(client_addr);

sockfd = socket(PF_INET, ???, 0);
/* ... */
memset((char *)&server_addr,0,sizeof(server_addr));
server_addr.sin_family = AF_INET;
server_addr.sin_port = ???;
server_addr.sin_addr.s_addr = INADDR_ANY;
bind(sockfd, (struct sockaddr *)&server_addr, sizeof(struct sockaddr));
/* ... */
???= listen(sockfd, 10);
/* ... */
???= accept(sockfd, (struct sockaddr *)&client_addr, &sin_size);
/* ... */

---------------------------------------

/* utilice la llamada al sistema socket() */
/*...*/
/* cambie esta IP de ejemplo por la correcta */
uint32_t dir=inet_addr("192.168.1.1");
struct sockaddr_in destino;
memcpy(&destino.sin_addr, &dir, 4);
/* siga rellenando la direccion destino */
/* y utilice la llamada al sistema connect()*/
```

```
#include <sys/socket.h>
#include <sys/types.h>
#include <sys/time.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <sys/wait.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>
#include <sys/select.h>
#include <signal.h>
```