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

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

Se requiere implementar un servicio de monitorización de la temperatura mediante una aplicación cliente/servidor TCP. En este caso el servidor se encarga de ir midiendo la temperatura de un sistema (simulado aleatoriamente) cada segundo y enviarla los clientes que se conectan. El servicio se provee hasta que el cliente se desconecta. El cliente, por su parte, muestra por pantalla los diferentes valores de temperatura recibidos hasta que el usuario introduce por teclado la palabra **`fin`**, en cuyo caso cierra la conexión y termina. Los valores de temperatura se intercambian usando mensajes de datos con el siguiente formato:

temperatura: **`uint16_t`**	control: **`uint32_t`**

donde:

- **temperatura** (**`uint16_t`**): contiene el valor de temperatura leída por el servidor.
- **control** (**`uint32_t`**): es un código de control para poder verificar que el valor del campo anterior es correcto. Así, si el valor de temperatura es **`t`** el campo de control es **t^3**.

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 el servicio de monitorización de temperatura como sigue.

1. El servidor escucha en el puerto **2119**.

2. Cuando se conecta un cliente, se inicia un proceso iterativo en el que:
    a. El servidor toma una muestra de la temperatura del sistema. Esto te simula en este caso con un valor aleatorio usando la siguiente expresión:

```
uint16_t temperatura = rand() % 10; //valor aleatorio entre 0 y 9
                                    //rand() está en stdlib.h
```

    b. Se calcula entonces el valor de control, y se envían ambos valores usando el formato detallado anteriormente.
    c. Este proceso se repite **cada segundo** hasta que el cliente se desconecta. En ese momento, el servidor vuelve a esperar a que se conecte un nuevo cliente.
3. Detalles de implementación:
    a. En todo caso se ha de garantizar que se lee y escribe lo esperado.
    b. Se han de liberar correctamente todos los recursos reservados.
    c. **El servidor no debe acabar ni ante la llegada de SIGPIPE, ni ante errores de escritura de tipo EPIPE.** Según el manual del sistema, cuando la función **`write()`** intenta escribir en un canal que han cerrado en el otro extremo, el kernel envía la señal **`SIGPIPE`** y, en caso de que dicha señal se ignore o se maneje, entonces devuelve **`-1`** y **`errno`** tiene el valor **`EPIPE`**.

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

Ejercicio 2. Cliente [1.5 puntos]

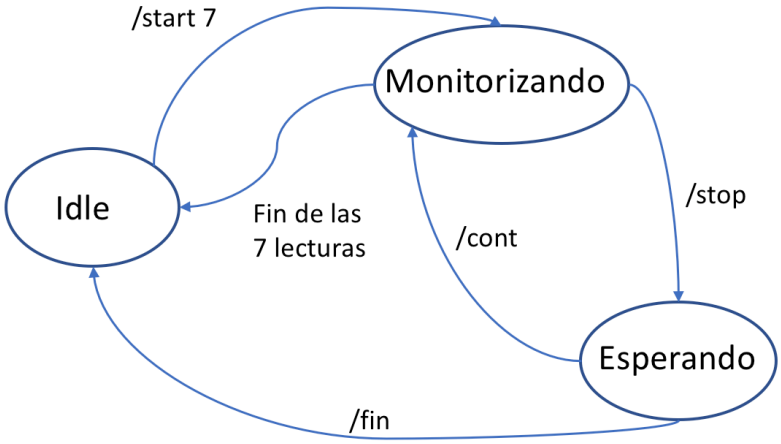
Implemente el cliente para este servicio, según la siguiente especificación:

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 mientras que el usuario no introduzca por teclado la palabra **`fin`**:
    a. Se recibe el mensaje con la temperatura según el formato explicado previamente.
    b. Se comprueba si la temperatura recibida es correcta, comprobando que el código de control es correcto. Es decir, si se recibe una temperatura **`t = 9`**, entonces el valor de control debe ser **`c = 9*9*9`**.
    c. Si la confirmación es correcta, se muestra por pantalla el valor de la temperatura.

   d. En caso contrario, se imprime **`Temperatura incorrecta`**.
3. Notas:
   a. En todo caso se ha de garantizar que se lee y escribe lo esperado.
   b. Se han de liberar correctamente todos los recursos reservados.

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

Ejercicio 3. Máquina de estados [2.5 puntos]



La figura muestra una máquina de estados que gestiona la monitorización de temperaturas de un determinado dispositivo que se simula como un proceso independiente. La máquina opera en base a instrucciones que llegan por una fifo. Cuando está en estado **`IDLE`**, la máquina espera la llegada del comando con formato **`/start <n>`**, por ejemplo, **`/start 7`**, y entonces transita al estado **`Monitorizando`**, para lo que se crea un proceso hijo que será el que simule la lectura de temperaturas como en el ejercicio anterior, usando la función:

```
    uint16_t temperatura = rand() % 10; //valor aleatorio
entre 0 y 9
```

El proceso hijo debe realizar mediciones **cada segundo**, hasta un máximo de **`n`** veces, en cuyo caso finaliza la ejecución. Es decir, **`/start 7`** lleva a que el hijo haga **7** lecturas de temperatura (una cada segundo) y después termina.

Mientras se está monitorizando la temperatura, la máquina debe atender la fifo, por la que el usuario puede escribir tres órdenes más:

- **`/stop`**: que hace que el proceso hijo se pare enviándole la señal **`SIGSTOP`**, en cuyo caso la máquina transita a **`Esperando`**.

- **`/cont`**: envía la señal **`SIGCONT`** al proceso hijo de monitorización, que reanuda la toma de temperaturas cada segundo, y pasa la máquina a estado **`Monitorizando`**.
- **`/fin`**: que envía la señal **`SIGKILL`** al proceso hijo de monitorización para parar su ejecución. En este caso, la máquina vuelve a estar ociosa (**`Idle`**) para poder iniciar otra medición por una nueva orden que llegue por la fifo.

La llegada de cualquier evento no esperado provoca automáticamente la finalización de la máquina, que se debe realizar de forma ordenada, devolviendo todos los recursos reservados al sistema operativo.

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_ad
/* ... */

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

/* 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>
```