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

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

Se requiere implementar un servicio remoto de envío de avisos síncronos. Se trata de una aplicación cliente/servidor TCP donde el cliente, solicita al usuario **un número de segundos** y **microsegundos** por teclado, y los envía al servidor, que arma un temporizador con esa información y comienza a enviar cadenas de texto "**`Evento 1`**", "**`Evento 2`**", … (sin comillas) hasta que el cliente se desconecte. El cliente puede cambiar la frecuencia de envío de eventos solicitando de nuevo **dos nuevos valores** de segundos y microsegundos, que envía al servidor. En caso de que ambos valores sean cero, el cliente cierra la conexión con el servidor, que atiende a otro cliente. Los detalles de la implementación son los siguientes.

Ejercicio 1. Servidor [2 puntos]

Implemente el servidor para dar el servicio de envío de avisos síncronos como sigue.

1. El servidor escucha en el puerto **2119**.
2. Cuando se conecta un cliente, recibe un número de segundos **`s`** (**`uint8_t`**) y un número de microsegundos **`ms`** (**`uint32_t`**).
3. Se inicia un proceso iterativo en el que:
    a. Se comienza a contar el tiempo indicado por **`s`** y **`ms`**.
    b. Si transcurre todo el tiempo anterior, se envía la cadena "**`Evento i`**" (sin comillas), donde **`i`** es el número de secuencia del mensaje, empezando por **`i = 1`**. Se recomienda el uso de la función **`sprintf()`**.
    c. Si no trascurre ese tiempo, puede ser porque:
        i. El cliente ha enviado valores nuevos para el temporizador, no se envía la cadena y se vuelve al Paso 3.a.
        ii. El cliente se ha desconectado, en cuyo caso se cierra la conexión y se atiende al siguiente cliente.
4. 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.

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

Ejercicio 2. Cliente [2 puntos]

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

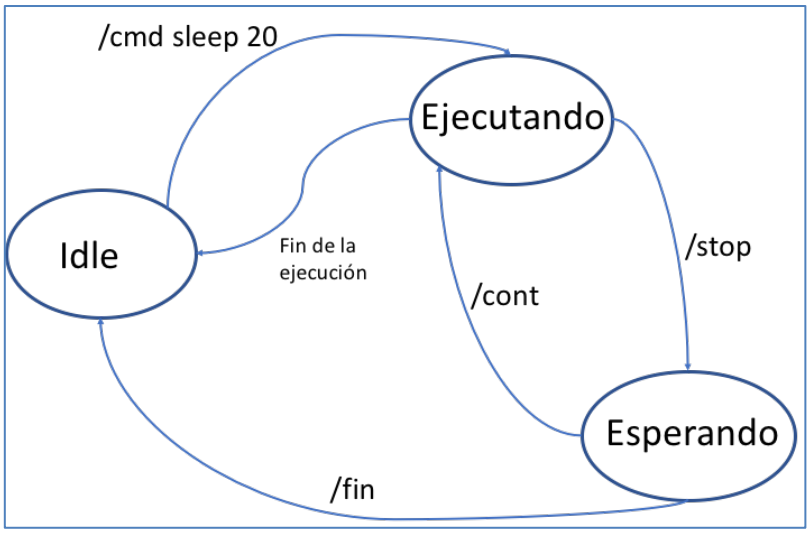
1. El cliente recibe como argumento de entrada la dirección IP del servidor.
2. Una vez conectado con el servidor, el cliente itera como sigue:
    a. Solicita por teclado un número de segundos s y un número de microsegundos ms.
    b. Si ambos valores son iguales a cero, entonces cierra la conexión con el servidor.
    c. En otro caso:
        i. Envía **`s`** (**`uint8_t`**) y **`ms`** (**`uint32_t`**) al servidor.

ii. El cliente entonces espera a que:
    1. El usuario introduzca dos nuevos valores de tiempo, en cuyo caso se vuelve al **Paso 2.b**, o,
    2. Llegue la cadena del servidor, en cuyo caso la muestra por pantalla y continúa la espera.

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 y monitoriza la ejecución de comandos leídos por una fifo. Cuando está en estado **`IDLE`**, la máquina espera la llegada de comandos por la fifo con formato "**`/cmd <comando>`**", e.g., "**`/cmd sleep 20`**" (sin comillas), y entonces transita al estado **`Ejecutando`**. La ejecución del comando se ha de realizar simulando el comportamiento de la función:

```
int system(char * comando)
```

Se recomienda mirar en el manual cómo funciona la función de biblioteca, donde se explica que, internamente, lo que hace es ejecutar el comando pasado como argumento con una función de la familia **`exec*`**, lanzando una shell: "**`/bin/bash -c <comando>`**".

Mientras se está ejecutando el comando, la máquina debe atender la fifo por la que pueden llegar tres órdenes más:

- **`/stop`**: que hace que el proceso se pare, en cuyo caso la máquina transita a **`Esperando`**. Para parar la ejecución del comando se debe enviar una señal **`SIGSTOP`** al proceso que lo ejecuta.
- **`/cont`**: envía la señal **`SIGCONT`** al proceso que ejecuta el comando, y pasa la máquina a estado **`Ejecutando`**.
- **`/fin`**: que envía la señal **`SIGKILL`** al proceso para parar su ejecución. En este caso, la máquina vuelve a estar ociosa (**`Idle`**) para poder ejecutar otro comando 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_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>
```