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 TCP para dar un servicio de monitorización de la memoria RAM utilizada en un equipo remoto, usando para ello la llamada al sistema **`sysinfo()`** de la siguiente forma:

```
#include <sys/sysinfo.h>

struct sysinfo info;
sysinfo(&info); /* llamada al sistema, chequear errores */
uint32_t v1 = info.totalram;
uint32_t v2 = info.freeram;
uint32_t v3 = info.sharedram;
uint32_t v4 = info.bufferram;
```

Los clientes iniciarán la solicitud de monitorización y podrán también pedir a los usuarios un cambio en la información se visualiza, de entre los recibidos por el servidor. El formato de los mensajes con los que se intercambian datos es el siguiente:

- Del cliente al servidor, para iniciar la monitorización:

Tipo: **`uint8_t`**	Número de muestras: **`uint16_t`**

  - **Tipo** (**`uint8_t`**): indica el tipo de mensaje. Su único valor válido es **`1`**.
  - **Número de muestras** (**`uint16_t`**): indica el número de veces que se monitoriza la memoria del sistema en el servidor.
- Del servidor al cliente, para enviar los datos de memoria obtenidos por **`sysinfo()`**:

Tipo: **`uint8_t`**	v1: **`uint32_t`**	v2: **`uint32_t`**	v2: **`uint32_t`**	v4: **`uint32_t`**

  - **Tipo** (**`uint8_t`**): indica el tipo de respuesta:
    - **`1`**: indica que los datos de memoria se han podido obtener de forma correcta.
    - **`0`**: indica que los datos no son válidos (por ejemplo, por un fallo en la llamada a **`sysinfo()`**).
  - **v1** (**`uint32_t`**): memoria total del sistema.
  - **v2** (**`uint32_t`**): tamaño de memoria disponible.
  - **v3** (**`uint32_t`**): cantidad de memoria compartida.
  - **v4** (**`uint32_t`**): memoria empleada por buffers.
- El servidor envía un mensaje de respuesta al cliente cada vez que se llama a **`sysinfo()`**, hasta completar el número de veces indicado en el mensaje del cliente.
- Todos los mensajes intercambiados entre cliente/servidor se **envían/reciben completos**, es decir, que incluyen todos los campos.

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 monitorización del sistema remoto como sigue:

1. El servidor escucha en el puerto **4950**.
2. Después de aceptar la conexión de un cliente, el servidor espera a recibir el mensaje de inicio de monitorización del cliente, según el formato anterior.
    a. **[Valor no válido]** Si el tipo del mensaje es distinto de **1**, el servidor compone un mensaje de respuesta con código no válido (`tipo = 0`) y cualquier valor en los campos `v1` a `v4`.
    b. [Valor válido] El servidor extrae del mensaje el número de muestras que tiene que tomar. Sea `n` este valor. Entonces, exactamente `n` veces, se realizan las siguientes operaciones:
        i. Se llama a `sysinfo()` y se extraen los valores de los campos correspondientes, usando el código incluido más arriba.
        ii. Se compone un mensaje de respuesta con código válido (`tipo = 1`) y los valores anteriores y se envía al cliente
        iii. Se duerme el proceso durante **5 segundos**.
3. Una vez finalizado el envío de las **n muestras** del estado de la memoria del equipo, el servidor atiende al siguiente cliente.

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 argumentos de entrada:
    a. La **dirección IP** del servidor.
    b. El **puerto** del servidor.
    c. El número de veces que se monitorizará la memoria en la máquina que ejecuta el servidor. Por ejemplo:

    ```
    ./cliente 127.0.0.1 4950 10
    ```

    indica que el servidor llama `10` veces a `sysinfo()`.

2. La información se recibe siguiendo el formato explicado más arriba y, por defecto, se muestra por pantalla con de la siguiente forma:

    ```
    Memory: 2092290048 306675712 51793920 50814976
    ```

    donde aparecen, en este orden, los valores de memoria total (`totalram`), memoria libre (`freeram`), memoria compartida (`sharedram`) y memoria usada en buffers (`bufferram`).

3. Mientras se está recibiendo la información, el cliente debe además poder recibir órdenes del usuario a través del teclado para cambiar el formato en el que se visualiza la información recibida desde el servidor. El formato de las órdenes es `/show <modo>`, donde `modo` es un carácter que puede tomar los siguientes valores:
    a. `'t'`: muestra sólo la memoria total
    ```
    Total memory: 2092290048
    ```
    b. `'f'`: muestra sólo la memoria libre
    ```
    Free memory: 306675712
    ```
    c. `'s'`: muestra sólo la memoria compartida
    ```
    Shared memory: 51793920
    ```
    d. `'b'`: muestra sólo la memoria usada en buffers.

`Buffered memory: 50814976`

e. **`'a'`**: restablece el modo de visualización completo

`Memory: 2092290048 306675712 51793920 50814976`

f. Si el carácter no es ninguno de los anteriores, entonces se muestra:

`Modo de visualización incorrecto`

4. El cliente itera hasta que el servidor cierra la conexión.

Un ejemplo de la salida del cliente sería la siguiente:

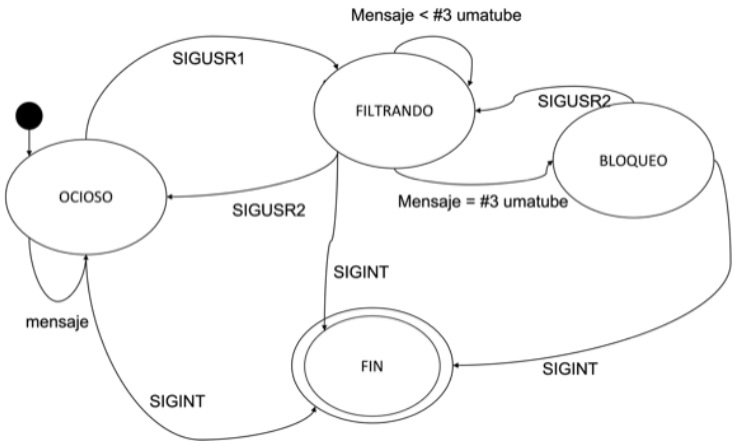
```
Memory: 2092290048 306675712 51793920 50810880
Introduzca el modo de visualización (/show [t|f|s|b|a]):
Memory: 2092290048 306675712 51793920 50810880
Introduzca el modo de visualización (/show [t|f|s|b|a]):
Memory: 2092290048 306675712 51793920 50814976
Introduzca el modo de visualización (/show [t|f|s|b|a]):
/show t
Total memory: 2092290048
Introduzca el modo de visualización (/show [t|f|s|b|a]):
/show f
Free memory: 307449856
Introduzca el modo de visualización (/show [t|f|s|b|a]):
/show s
Shared memory: 51793920
/show b
Buffered memory: 50814976
Introduzca el modo de visualización (/show [t|f|s|b|a]):
/show a
Memory: 2092290048 307449856 51793920 50814976
Introduzca el modo de visualización (/show [t|f|s|b|a]):
/show e
Modo de visualización incorrecto
```

Nótese, no obstante, que la salida del programa no tiene que coincidir con la anterior, y que es sólo un ejemplo para mostrar su funcionamiento.

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

Ejercicio 3. Máquina de estados [2 puntos]

Se propone el diseño e implementación de una máquina de estados que actúa como filtro de la información que circula por un proxy (máquina intermedia), y que se puede ver en la imagen siguiente.



El filtrado se va a realizar de la siguiente forma. Al principio, y puesto que se va a utilizar en una simulación y no en un escenario real, el programa leerá cadenas de texto que llegarán por una fifo llamada **`fsc_filtro`**. Sin embargo, al activar el modo filtrado (se explica más adelante como hacerlo), cada vez que se introduzca un mensaje que contenga la palabra **`umatube`**, la máquina lo cambiará por ******* (manteniendo el resto del mensaje intacto) antes de imprimirlo por pantalla. A la tercera vez que se detecte

la introducción de **umatube**, la máquina dejará de imprimir mensajes, mostrando por pantalla el aviso **Usted ya no está autorizado a usar el proxy**.

Las acciones de activar/desactivar el filtro, así como la terminación de la máquina, se realizan mediante la recepción de las señales **SIGUSR1**, **SIGUSR2** y **SIGINT**, respectivamente. La implementación debe garantizar que no se pierde ningún evento.

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>
```