E.T.S.I. TELECOMUNICACIÓN

FUNDAMENTOS DE SOFTWARE DE COMUNICACIONES

Laboratorio

APELLIDOS, NOMBRE:

TITULACIÓN:

Ejercicio 1 (3 puntos)

Se requiere implementar una máquina de estados que chequee los mensajes provenientes de un canal de comunicaciones, donde cada mensaje es un carácter. La máquina comienza en un estado **`LEYENDO`**, y puede encontrarse con caracteres de tipo '**`W`**' (warning) o de tipo '**`E`**' (error). En el momento que contabilicen **2 errores**, que pueden venir de forma no consecutiva, la máquina pasa al estado **`CRÍTICO`**. En este estado, si aparece un nuevo error, la máquina termina, mostrando antes este mensaje por consola:

"**`Tercer error. Cierre de la maquina`**".

Si, estando en el estado **`CRÍTICO`**, pasan **`3'5` segundos** sin que aparezca ningún mensaje por el canal, se debe volver al estado **`LEYENDO`**, y se vuelve a empezar desde cero la cuenta de errores.

Requisitos adicionales:

- La máquina debe leer caracteres desde una fifo denominada "**`fsc_fifo`**", que se habrá creado antes de la ejecución del programa. En caso de que el otro extremo cierre la fifo para escritura, la máquina de estados reaccionará terminando, en cualquiera de sus estados, y mostrando el mensaje:

  "**`Cierre detectado. Fin de la máquina`**".

- Cualquier carácter leído que no sea '**`W`**' o '**`E`**' (podrían valer también esos caracteres en **minúscula**), genera un evento particular que se debe ignorar en cualquier estado de la máquina.

- Cuando se utilice un temporizador, se debe interrumpir cualquier bloqueo en lectura si llega la señal (comportamiento **System V**). Para ello, se debe compilar el código con **`-ansi`** o con la opción **`-std=c99`**.

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

Ejercicio 2

En los sistemas críticos, como la aviónica, la monitorización de sensores, el cómputo y todo lo relacionado con el hardware/software electrónico conlleva un proceso de replicación. Así, por ejemplo, para monitorizar la temperatura, se dispone de varios sensores que envían sus datos a un servidor que se encarga de detectar errores, inconsistencias, etc. La comunicación entre ambas entidades debe ser **segura** y **fiable**. Se requiere implementar un sistema que emule este comportamiento de un sistema crítico con las siguientes especificaciones:

Apartado 2.1 (1 punto): Sensores de temperatura

El sensor de temperatura será un proceso que **simule** la recogida de datos, que serán introducidos de forma interactiva por teclado. Las características específicas de funcionamiento son las siguientes:

1. Como argumento de entrada se recibe la IP del servidor, que escucha en el puerto **`4950`**.
2. Una vez establecida la conexión, el proceso entra en un bucle infinito en el que se solicita por teclado una temperatura en formato entero con valor entre **`[-260°C, 153°C]`**, y se envía al servidor.
3. El proceso se para cuando el usuario pulsa **`Ctrl+C`**, en cuyo caso se han de liberar los recursos reservados de forma ordenada.

Se entrega el fichero **ejercicio2.1.c/cpp**.

Apartado 2.2 (4 puntos): Monitorización

La monitorización de las temperaturas se lleva a cabo en un servidor recibe las temperaturas de los sensores e identifica varios eventos que pueden ocurrir en dicho proceso. Su funcionamiento es el siguiente:

1. El puerto de escucha es el **`4950`**.
2. El servidor recibe lecturas exactamente de **3**, y sólo **3**, sensores de temperatura como los definidos en el apartado anterior.
3. Se han de atender **simultáneamente** las lecturas de los 3 sensores en el mismo hilo de ejecución.
4. Una vez conectados los tres sensores, comienza el ciclo de monitorización de las temperaturas. Se han de considerar los siguientes casos:
    a. Si una vez que se comienza la lectura, el servidor no recibe ninguna lectura de ninguno de los 3 sensores después de **5 segundos**, debe mostrar por pantalla el mensaje:
    "**`Timeout`**"

    b. Cuando se recibe una temperatura de un sensor, se ha de comprobar que tiene un valor válido entre **`[-260°C, 153°C]`**. Si no es así, se muestra por pantalla:
    "**`Warning: valor incorrecto`**".

    c. Asumiendo que una lectura de un sensor sobreescribe cualquier lectura previa, una vez se tenga una lectura válida en los tres sensores, se ha de comprobar que los valores leídos son consistentes. Esto se implementa de la siguiente forma. Si las temperaturas recibidas son **`t1`**, **`t2`** y **`t3`**, se deben cumplir las tres condiciones siguientes:

    $$|t1 - t2| < 5$$
    $$|t1 - t3| < 5$$
    $$|t2 - t3| < 5$$

    donde **`|·|`** es la función valor absoluto **`int abs(int j)`**. Si alguna de esas condiciones no se cumple, entonces se muestra por pantalla el mensaje:
    "**`Warning: lectura inconsistente`**".

    d. Si se detecta que uno de los sensores (cliente) se ha desconectado, entonces el sistema termina completamente de forma ordenada.

Se entrega el fichero **ejercicio2.2.c/cpp**.

Apartado 2.3 (2 puntos): Información

Modifique el servidor de monitorización del Apartado 2.2 (**Monitor**) para que, antes de comenzar a atender sensores, cree un proceso hijo que simule la conexión con un panel de control. Al hijo, a partir de ahora, lo nombraremos **PanelControl**. Este proceso se encarga de mostrar los mensajes de los diferentes eventos. La comunicación entre el proceso **Monitor** y el proceso **PanelControl** se realiza mediante señales, de forma que, cuando se generen los eventos:

1. "**`Timeout`**" (Apartado 2.2, paso 4.a), **Monitor** envía **`SIGUSR1`** a **PanelControl**.
2. "**`Warning: ...`**" (Apartado 2.2, pasos 4.b y 4.c), **Monitor** envía **`SIGUSR2`** a **PanelControl**.
3. Si **Monitor** termina por la desconexión de un sensor de temperatura (Apartado 2.2, pasos 4.d), envía **`SIGTERM`** a **PanelControl**, para que termine también de forma ordenada.

El proceso **PanelControl** se mantiene indefinidamente a la espera de recibir una de esas dos señales y, en cada caso, imprime por pantalla los siguientes mensajes:

1. Si llega **`SIGUSR1`** se imprime "**`Error de lectura en los sensores`**".
2. Si llega **`SIGUSR2`** se imprime "**`Warning: lectura inconsistente`**".

Se entrega el fichero **ejercicio2.3.c/cpp**.

Esqueletos de guía del Ejercicio 2:

```c
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()*/
```

```c
#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>
```