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

APELLIDOS, NOMBRE:
TITULACIÓN:

Ejercicio 1 [1.5 puntos]

Se quiere implementar un programa de chat de conversación que esté atento a los mensajes que envían dos usuarios, simulados por dos fifos de nombre **`/tmp/f1`** y **`/tmp/f2`**, y que está moderado por un administrador que le introduce comandos por teclado. El programa, que sólo ha de atender las fifos mientras estén abiertas, simplemente lee y muestra por pantalla los mensajes recibidos.

El programa de chat está atento también a las acciones que recibe del moderador, a través del teclado. Dichas acciones son:

1. Activar la recepción de mensajes de las fifos: este modo se activa cuando escribe por teclado la cadena "**`/on`**" (sin comillas).
2. Desactivar la recepción de mensajes de las fifos, que entra en funcionamiento cuando escribe por teclado la cadena "**`/off`**" (sin comillas). En este modo, los mensajes de los usuarios quedan en la fifo (y al no leerlas el programa de chat, no se muestran por pantalla hasta que el moderador no vuelva a introducir la cadena "**`/on`**" -sin comillas-).
3. Terminar el servicio de chat cuando se introduce el mensaje "**`/exit`**" (sin comillas).

Cualquier otra cadena introducida por el moderador se muestra por pantalla como si fuese un mensaje normal. Inicialmente, el chat tiene activada la recepción de mensajes.

Para enviar mensajes por las fifos al proceso de chat se utilizarán dos consolas adicionales y el comando **`cat`** de la siguiente forma **`cat > /tmp/f1`**. (de la misma forma para **`/tmp/f2`**).

Se entrega el fichero **`ej1.c`**.

Ejercicio 2 [1.5 puntos]

Implemente la función:

```
int fsc_cron(char * c, int s)
```

que ejecuta el comando **c** cada **s** segundos hasta que el usuario interrumpe su ejecución enviándole la señal **SIGUSR1**. La función devuelve **0** en caso de ejecución correcta, y **-1** en caso de fallo. Se recomienda mirar en el manual cómo funciona la función de biblioteca

```
int system(char * commando)
```

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>**".

Se debe entregar un programa **ej2.c** donde esté definida la función solicitada y aparezca un **main** donde se invoque adecuadamente.

Notas:

1. En ningún caso, se tiene que utilizar directamente la función **system()**. Aquí solo se utiliza su página del manual, porque da las recomendaciones necesarias para abordar parte del ejercicio.
2. La implementación de la función **fsc_cron()** debe terminar su ejecución solo cuando todos los comandos activos en el momento de llegar **SIGUSR1** finalizan.
3. No se puede ignorar la señal **SIGCHLD**.
4. No deben quedar procesos zombies ni huérfanos.
5. Es importante usar **/bin/bash** en lugar de **/bin/sh.**

Ejercicio 3

Se desea implementar un servicio de envío de señales a procesos que se ejecutan en una máquina remota usando para ello sockets **TCP**. El sistema cuenta con un cliente que recibe como argumento de entrada la dirección **IP** del equipo donde corre el proceso, el **PID** del proceso en esa máquina y la señal que se le quiere enviar. El servidor se encarga de recibir el **PID** y la señal, y la envía a un proceso residente en su máquina usando la llamada al sistema **kill()**. Los detalles de la implementación son los siguientes.

Apartado 3.1. Cliente [1.5 puntos]

El cliente debe cumplir con las siguientes especificaciones:

1. El puerto de conexión es el **2119**.
2. Recibe tres argumentos de entrada: dirección IP del servidor, PID del proceso en la máquina remota, y nombre de la señal a enviar.

```
./cliente 127.0.0.1 7635 SIGUSR1
```

3. Una vez establecida la conexión con el servidor, el cliente envía un mensaje con dos campos: el PID y el número de señal.
   a. El PID, es de tipo **pid_t** y ocupa **4 bytes**.
   b. El nombre de señal se recibe como argumento al programa en formato cadena de caracteres, pero, internamente, en el cliente se convierte a su equivalencia numérica en **`uint8_t`** (ver **man 7 signal**), que es lo que realmente se manda al servidor. (No hace falta tener en cuenta todas las señales en el programa, vale con **`SIGUSR1`** y **`SIGUSR2`**).
4. El cliente espera recibir el resultado de la llamada al sistema **`kill()`** ejecutada por el servidor (que es de tipo **`int32_t`**), y muestra por pantalla si ha sido correcta o no (el valor cero indica que la ejecución remota fue correcta).
5. Termina su ejecución.

Apartado 3.2. Servidor [2 puntos]

El servidor debe cumplir con las siguientes especificaciones:

1. El puerto de escucha es el **2119**.
2. Atiende clientes indefinidamente, uno cada vez.
3. Por cada cliente que se conecta:
   a. Se recibe un mensaje con dos campos:
      i. El PID del proceso al que enviar la señal (**4 bytes**).
      ii. El código de la señal (**1 byte**).
   b. Procede al envío de la señal al proceso indicado, usando la función **`kill()`**.
   c. Devuelve al cliente el valor de retorno de dicha función, como un valor de tipo **`int32_t`**.
   d. Cierra la conexión con el cliente y espera al siguiente.

Para pruebas, se recomienda implementar un programa simple que registre manejadores para las señales **`SIGUSR1`** y **`SIGUSR2`** y comience una espera infinita, de tal forma que, cuando reciba alguna de estas señales, lo muestre por pantalla. El PID de este proceso de pruebas será el que se le pase al cliente, junto con una de las dos señales a enviarle.

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>
```