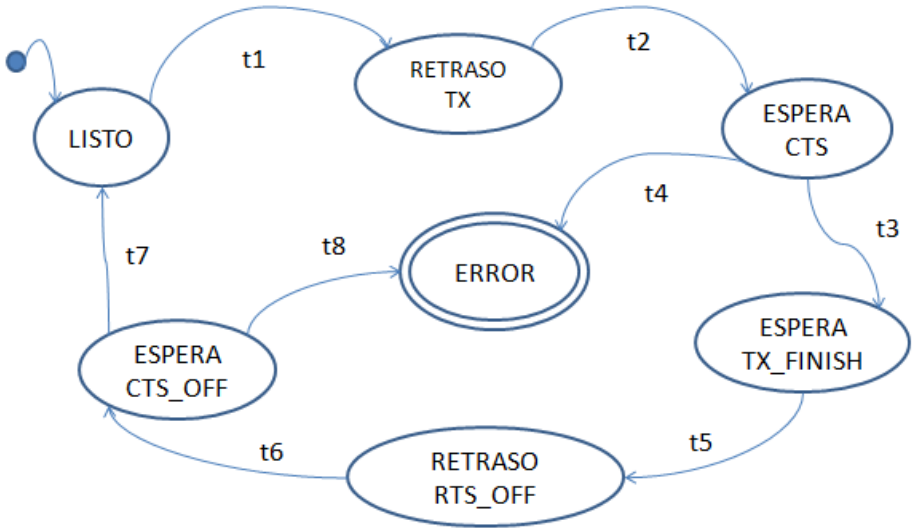
**E.T.S.I.TELECOMUNICACIÓN**
**FUNDAMENTOS DE SOFTWARE DE COMUNICACIONES**
**Convocatoria ordinaria. Junio 2015/2016**

**APELLIDOS, NOMBRE**.....
**TITULACIÓN**.....
**NÚMERO DEL PC**..... **AULA**.....

**IMPORTANTE: Se han de elegir dos (y sólo dos) ejercicios de los tres propuestos en este examen. SE ENTREGA UN ZIP CON LAS SOLUCIONES EN EL CAMPUS VIRTUAL**

## Ejercicio 1 (5 puntos)

Un equipo terminal de datos (DTE) envía información por una línea de baja velocidad a un equipo de control (DCE). El protocolo de envío se representa mediante la siguiente máquina de estados:




a) Implemente la máquina de estados representada en la figura, teniendo en cuenta estas transiciones:
   - t1: Evento de llegada: **MANDA_MENSAJE**. Acciones a realizar antes de cambiar de estado: activar un temporizador de 1'5 segundos
   - t2: Evento de llegada: **expiración del temporizador** activado en t1. Acciones a realizar antes de cambiar de estado: imprimir por pantalla ("RTS[1] activa"). Además, activar un temporizador a 3'3 segundos
   - t3: Evento de llegada: **CTS[2]**. Acciones a realizar antes de cambiar de estado: cancelar el temporizador activado en t2 (un temporizador se cancela si se vuelve a activar con valor cero), e imprimir por pantalla ("CTS activa")
   - t4: Evento de llegada: **expiración del temporizador** activado en t2. Acciones a realizar antes de cambiar de estado: imprimir por pantalla ("Error: CTS no activa"). Fin de la máquina.
   - t5: Evento de llegada: **TX_FINISHED**. Acciones a realizar antes de cambiar de estado: activar un temporizador de 1'5 segundos.

[1] RTS son las siglas de *ready to send*
[2] CTS son las siglas de *clear to send*

- t6: Evento de llegada: **expiración del temporizador** activado en t5. Acciones a realizar antes de cambiar de estado: imprimir por pantalla ("RTS desactivada"). Además, activar un temporizador a 4'2 segundos
- t7: Evento de llegada: **CTS_OFF**. Acciones a realizar antes de cambiar de estado: cancelar el temporizador activado en t6, e imprimir por pantalla ("CTS desactivada")
- t8: Evento de llegada: **expiración del temporizador** activado en t6. Acciones a realizar antes de cambiar de estado: imprimir por pantalla ("Error: CTS no desactivada"). Fin de la máquina.

En este apartado, deje vacía la función que espera un evento y que será la encargada de hacer avanzar la máquina (haga que devuelva cualquier valor). También puede dejar sin contenido cualquier manejador de señal que se utilice. Sin embargo, el código debe estar libre de errores de compilación/vinculación. **(2.5 puntos) SE ENTREGA UN FICHERO CON EL NOMBRE ejercicio1a.c (o .cpp)**

b) Haga funcionar la máquina rellenando las partes vacías del ejercicio anterior. Para ello, siga estas especificaciones:
   - Cree una pipe, al principio del programa, para poder encolar los eventos que lleguen a la máquina. La función *espera_evento()* se implementa entonces leyendo un byte de la pipe (y por tanto bloqueando el programa mientras no haya datos en ella). NOTA: los descriptores de la pipe tienen que declararse como variable global, para poder ser utilizados en varias funciones del programa.
   - La expiración de un temporizador dará como resultado la ejecución de un manejador de señal que insertará en la pipe un byte con el valor de evento que se haya elegido para representar el timeout en la máquina de estados.
   - De la misma forma, se elegirán otras señales para representar a los eventos MANDA_MENSAJE, CTS, TX_FINISHED y CTS_OFF. Por ejemplo SIGINT, SIGTERM, SIGUSR1 y SIGUSR2, respectivamente. Sus manejadores correspondientes insertarán en la pipe un byte con el valor de evento que se haya elegido para representarlos en la máquina de estados.

Para hacer pruebas y comprobar que la máquina evoluciona como debe, se recomienda utilizar el comando *kill* desde teclado para enviar señales al proceso. **(2.5 puntos) SE ENTREGA UN FICHERO CON EL NOMBRE ejercicio1b.c (o .cpp)**

**Ejercicio 2 (5 puntos)**

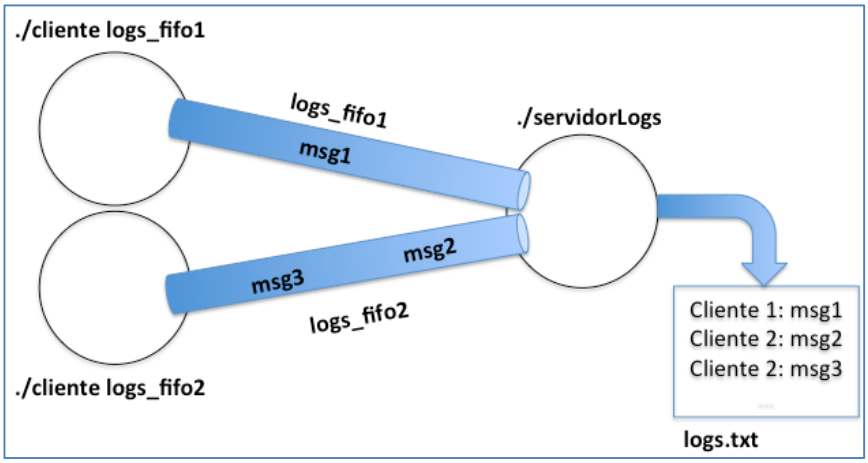
Hay que implementar un servidor de registro de sucesos (logs) remotos que va almacenando en un fichero la información que llega de dos clientes.

a) **(3.5 puntos)** Implementación del servidor atendiendo a las siguientes especificaciones:
   1. El servidor recibe mensajes de log de dos clientes que se comunican con él mediante dos FIFOs, llamadas: **`logs_fifo1`** y **`logs_fifo2`**. El cliente debe indicar de antemano al servidor la longitud de la cadena a leer, para que éste lea los caracteres necesarios del mensaje (ni de menos ni de más).
   2. Las conexiones con los clientes se atienden multiplexando la entrada/salida con la función **`select()`**.
   3. El servidor escribe los mensajes de log recibidos en un fichero llamado "**`logs.txt`**".

4. Para identificar al emisor, el servidor incluye un prefijo a cada mensaje de log recibido, dependiendo de la FIFO por la que llegue. El prefijo es "Cliente 1: " o "Cliente 2: " si el mensaje llega, respectivamente, por **`logs_fifo1`** o **`logs_fifo2`**.
5. Cada cadena se inserta en una línea nueva, sin líneas vacías entre ellas.
6. Si el fichero no existe, se crea y, si existe, se añaden entradas por el final.
7. **SE ENTREGA UN FICHERO CON EL NOMBRE ejercicio2a-servidor.c (o .cpp)**

b) **(1.5 puntos)** Implementar el código de los clientes según la siguiente especificación:
1. Los clientes reciben como argumento del main el nombre de la FIFO con la que se comunica con el servidor.
2. Se solicitan cadenas de texto por teclado al usuario hasta que éste escribe "**`fin`**".
3. Cada cadena se manda al servidor de manera independiente.
4. **SE ENTREGA UN FICHERO CON EL NOMBRE ejercicio2b-cliente.c (o .cpp)**

Una representación con el sistema sería:




En la imagen, y atendiendo al orden en el que aparecen los mensajes en el fichero **`logs.txt`**, el cliente que se conecta por **`logs_fifo1`** envía primero al servidor el mensaje "**`msg1`**", y después el cliente que se conecta por **`logs_fifo2`** envía, uno seguido de otro, los mensajes "**`msg2`**" y "**`msg3`**".

**Notas:**
1. Las FIFOs se crean desde la consola, puesto que existen antes de ejecutar los clientes y el servidor
2. En ambos casos se debe garantizar que se lee y escribe lo esperado.

## Ejercicio 3 (5 puntos)

Se requiere implementar un sistema cliente/servidor en TCP que permita la ejecución remota de comandos en el servidor. Para ello, el servidor recibe cadenas de texto desde un cliente que son comandos de shell, los ejecuta, y luego devuelve el valor de retorno al cliente.

a) **(4 puntos)** Especificación del servidor:
1. El servidor recibe de los clientes una cadena de texto con la orden que debe ejecutar la shell. Por ejemplo: **`"ls -al"`**. El cliente debe indicar de antemano al servidor la longitud de la cadena a leer, para que éste lea los caracteres necesarios del comando (ni de menos ni de más). La longitud del comando podría superar los 255 caracteres.

2. El servidor debe permitir atender a múltiples clientes, ejecutando sus comandos en paralelo, y empleando para ello la función **`system()`**.
3. El servidor devuelve al cliente el valor de retorno de **`system()`** en formato uint32_t, para informarle del resultado de la ejecución remota.
4. **SE ENTREGA UN FICHERO CON EL NOMBRE ejercicio3a-servidor.c (o .cpp)**

b) **(1 punto)** Especificación del cliente:
1. Recibe un único argumento en el **`main()`** con la cadena de texto que contiene la orden de la shell para ejecutar en remoto. Por ejemplo:
   **`./client "/bin/ls -al"`**.
   (Note el uso de comillas dobles para que solo se reconozca la cadena como un único argumento).
2. Tras enviar la información en el formato esperado por el servidor, se espera su respuesta y se muestra por pantalla si la ejecución remota ha sido correcta (valor de retorno 0) o no (valor de retorno distinto de 0).
3. **SE ENTREGA UN FICHERO CON EL NOMBRE ejercicio3b-cliente.c (o .cpp)**

**Notas**:
1. En ambos casos se debe garantizar que se lee y escribe lo esperado.
2. No deben quedar hijos zombies o huérfanos.
3. Compilación del servidor
   a. Si el servidor se compila con la opción "**`-std=c99`**", entonces el **`accept()`** devuelve **`-1`** si es interrumpido por alguna señal, y hay que hacer que el servidor siga esperando clientes ante esta eventualidad.
   b. Si no se compila sin la opción anterior, entonces el **`accept()`** funciona correctamente ante la llegada de señales.

Código de referencia (para el servidor):

```
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);
/* ... */
```

Código de referencia (para el cliente):

```
/* utilice la llamada al sistema socket() */
/*...*/
uint32_t dir=inet_addr("192.168.1.1"); /* cambie esta IP de ejemplo por la
correcta */
struct sockaddr_in destino;
memcpy(&destino.sin_addr, &dir, 4);
/* siga rellenando la direccion destino */
/* y utilice la llamada al sistema connect()*/
```