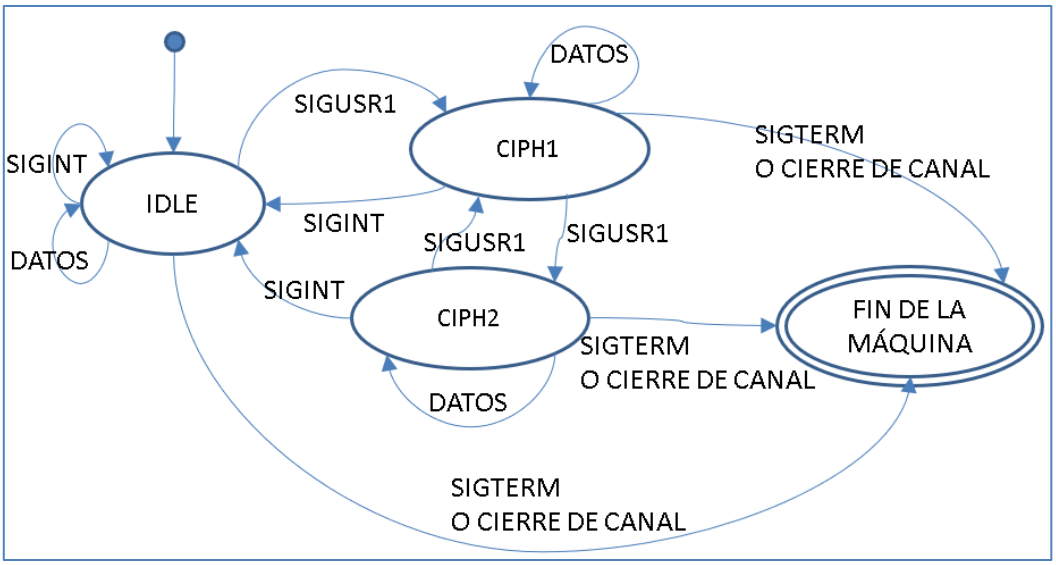
E.T.S.I. TELECOMUNICACIÓN

FUNDAMENTOS DE SOFTWARE DE COMUNICACIONES

Laboratorio

APELLIDOS, NOMBRE:
TITULACIÓN:

Ejercicio 1 [2.5 puntos]



La figura representa la máquina de estados que controla un proceso codificador de mensajes sencillo. Si la máquina está desocupada (estado **`IDLE`**), y se detectan datos disponibles en una fifo, el proceso los lee y saca por pantalla el mismo texto recibido sin alterar. Sin embargo, si la máquina está en el estado **`CIPH1`**, ante la llegada de datos el proceso muestra por pantalla un texto donde al valor de cada carácter **`ASCII`** recibido se le ha sumado uno. En el caso de encontrarse en el estado **`CIPH2`**, a cada carácter leído se le suman dos para mostrarlo. Como puede apreciarse, la máquina cambia de estado ante la llegada de señales. La finalización de la máquina, y con ella el proceso, sucede o bien cuando se recibe la señal **`SIGTERM`**, o cuando se detecta que la fifo por la que se leen los mensajes a codificar se ha cerrado desde el otro extremo escritor.

Se pide la implementación de esta máquina con los siguientes requisitos:

1) Manejadores de señal: los eventos recibidos como señales en el proceso se han de encolar en una pipe (cada evento ocupará **`1 byte`**).
2) Se asume que la fifo ya existe, y se llama "**`fifo_fsc`**".
3) Bucle de la máquina hasta que no se alcance la condición de fin:
    a. La espera de eventos se implementa mediante un bloqueo en **`select()`**, donde se estará escuchando tanto a la pipe como a la fifo, hasta que alguna tenga datos listos para lectura. Al salir del **`select()`**, se comprueba primero si hay datos a leer en la pipe de eventos. En caso afirmativo, directamente se lee un evento extrayéndolo de la pipe. Este evento es el que hará evolucionar la máquina en esta iteración. Si no ha habido eventos motivados por una señal, en condiciones normales debe haber datos listos en la fifo, así que se considera que el evento que hará evolucionar la máquina en esta iteración será **`DATOS`**.
    b. En la implementación de la máquina de estados (según el estado y evento adecuados) es donde se realiza, si procede, la lectura de la fifo, la codificación con el desplazamiento y la impresión por pantalla.
4) La salida del bucle implica acabar inmediatamente el **`main`** del programa. No se requiere hacer un cierre explícito de ningún descriptor de fichero abierto por el proceso.

NOTA: Se ha de compilar con la opción **`-std=c99`**, de tal forma que las funciones bloqueantes como **`select()`** pueden devolver **`-1`** ante la llegada de una señal. Se debe reanudar el comportamiento normal del programa ante tal eventualidad.

Para probar el programa, se recomienda abrir tres terminales: en el primero se ejecutará el proceso que implementa el ejercicio, el segundo se utilizará como emisor de señales al proceso y el tercero puede abrir la fifo para escribir mensajes en ella.

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

Ejercicio 2

Se requiere implementar un servicio de almacenamiento remoto de ficheros usando sockets **`TCP`** que codifica el contenido de los mismos para su transmisión por la red. La codificación se hará de forma sencilla, sumando un desplazamiento a cada byte a transmitir, y restándolo en recepción.

Apartado 2.1. Cliente [1.5 puntos]

Los detalles de implementación del cliente son los siguientes:

- Argumentos de entrada. El programa recibe 3 argumentos de la línea de comandos:
  `<IP del servidor> <puerto del servidor> <desplazamiento para codificar>`
- El cliente solicita nombres de fichero a los usuarios hasta que se introduzca la cadena "**`fin`**".
- Para cada fichero, se abre y se envía su contenido, por bloques previamente codificados, al servidor. Así, por cada bloque:
  - Se envía, primero, el tamaño de bloque (tipo **`uint16_t`**).
  - El bloque se codifica sumando a cada byte un desplazamiento (el que se recibe como argumento al programa):
    ```
    buffer[i] = buffer[i] + desplazamiento;
    ```
  - Se envía el contenido codificado.
- Al terminar la lectura de un fichero, se envía un tamaño de bloque igual a **`0`** (**cero**) para indicar al servidor que comienza el envío del siguiente.
- Cuando se lee "**`fin`**", se cierra la conexión y se termina ordenadamente.

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

Ejercicio 2.2. Servidor [1.5 puntos]

La especificación del servidor es la siguiente:

- Argumentos de entrada. El programa recibe un argumento de entrada:
  `<desplazamiento para decodificar contenidos>`
- Los nombres de fichero para su almacenamiento en el servidor se crean automáticamente.
  - Nomenclatura: "**`file1.dat`**", "**`file2.dat`**", …. Se recomienda utilizar la función **`sprintf()`**:
    ```
    char nombre[T];
    int indiceFichero = 1;
    sprintf(nombre,"file%d.dat", indiceFichero);
    ```
  - Los ficheros, si no existen, se crean y, si existen, se sobreescriben.
- El servidor acepta conexiones indefinidamente.

- Por cada conexión aceptada:
  - Se reciben bloques de datos en formato tamaño (tipo **`uint16_t`**) + contenido.
  - Por cada bloque recibido:
    - **Si el tamaño recibido es cero**, es decir, que se ha terminado de transferir un fichero, se cierra el descriptor y se prepara para abrir el siguiente usando la nomenclatura del paso previo.
    - **Si es mayor que cero**, se procede a leer el contenido del bloque.
    - Antes de escribirlo en disco, se decodifica restando el desplazamiento:

      `buffer[i] = buffer[i] - desplazamiento;`
  - La recepción de bloques se realiza **hasta que el otro extremo se desconecta**, en cuyo caso se liberan los recursos y cierra ordenadamente.
  - **No deben quedar ficheros vacíos en el servidor cuando un cliente se desconecta.**

Se entrega el fichero **ejercicio2.2.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_ad
/* ... */

---------------------------------------

/* 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>
```