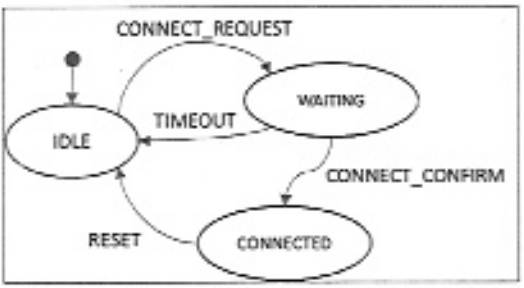
E.T.S.I. TELECOMUNICACIÓN
FUNDAMENTOS DE SOFTWARE DE COMUNICACIONES
Convocatoria ordinaria, septiembre 2020/2021

APELLIDOS, NOMBRE: Gutiérrez Criado, Manuel Eloy
DNI: 25.704.711-A
TITULACIÓN: Sistemas de Telecomunicación
Número de PC: 124

Ejercicio 1 [1.75 puntos]

En este ejercicio se va a trabajar con la máquina de estados de la derecha:



Descripción de los eventos que devuelve la función `espera_evento()`, y responsables de que evolucione la máquina:

- **CONNECT_REQUEST**: se produce al recibir la señal `SIGUSR1`.
- **TIMEOUT**: se produce al recibir la señal `SIGALRM` (tras expirar un temporizador de dos segundos activo al entrar en estado WAITING).
- **CONNECT_CONFIRM**: se produce al recibir la señal `SIGUSR2`.
- **RESET**: se produce al recibir un carácter desde la fifo `"fsc_maquina"` (sin las comillas).

Estos son los requisitos particulares a aplicar en el programa que debe implementar esta máquina de estados:

- Antes de arrancar la máquina, cree una **fifo** en el sistema de archivos de nombre `"fsc_maquina"` (sin las comillas), y ábrala en modo lectura.
- La máquina, tal y como está implementada, no termina nunca a no ser que se produzca un error en la función `espera_evento()`. No se considera error el que aparezca un evento no esperado. En este caso la máquina simplemente permanecerá en su estado actual.
- Si se reciben por la **fifo** cero bytes, esto sí se considera un error que permitirá salir de la máquina y del programa.
- No se pueden perder señales, por lo que se utilizará una tubería de tipo **pipe** para encolarlas a medida que se vayan recibiendo.
- Obviamente, la función destinada a esperar eventos debe tener en cuenta que puede llegarle información por la **pipe** o por la **fifo**. En esta ocasión, chequee siempre primero si hay datos en la pipe, lo que dará la menor prioridad a la aparición del evento **RESET**.

- Al arrancar la máquina, imprima por pantalla el mensaje "`La máquina arranca en estado IDLE`".
- Imprima por pantalla el nuevo estado de la máquina cada vez que se produzca un cambio de estado.

Implemente la máquina de estados en el fichero **ejercicio1.c**

Ejercicio 2 [0.75 puntos]

Cree un programa de pruebas que permita comprobar que el proceso que ejecuta la máquina de estados del ejercicio anterior evoluciona exactamente de acuerdo con este escenario:

La máquina comienza en **IDLE**, avanza a **WAITING**, retrocede a **IDLE**, avanza a **WAITING**, avanza a **CONNECTED** y, por último, avanza a **IDLE** (y permanece en dicho estado).

Por lo tanto, el programa de pruebas tendrá que enviar directamente, y de forma automática, las señales y/o información por la **fifo** al proceso que ejecuta la máquina, en el orden y tiempo necesarios.

Implemente este programa en el fichero **ejercicio2.c**

Ejercicio 3 [1.75 puntos]

Se pretende simular un **cliente** para un juego en red, que utiliza un protocolo de transporte fiable y se conecta a un servidor que escucha en el puerto **5050**. Su única misión consiste en enviar al servidor un mensaje para que actualicen datos del juego (asegurándose que se envían los datos del mensaje), y luego terminar.

La información a enviar al servidor se le pasa al cliente como argumentos al programa, de esta forma:

- Si se le pasa **un argumento** (de tipo entero): se interpreta como que es un ángulo de giro, en grados, cuyo valor está comprendido entre -360 y +360. El cliente tendrá que mandar esta información en un mensaje de tipo **GIRO**.
- Si se le pasan **dos argumentos** (de tipo entero): se interpretan como coordenadas **x** e **y**, respectivamente, cuyos valores están comprendidos entre **-2000** y **2000**. El cliente tendrá que mandar esta información en un mensaje de tipo **MOVIMIENTO**.
- Si se le pasan **tres argumentos** (de tipo entero): los dos primeros se interpretan como las coordenadas **x** e **y**, y el último como **el ángulo de giro**. El cliente tendrá que mandar esta información en un mensaje especial, que contenga dentro un mensaje **GIRO** y un mensaje **MOVIMIENTO**. Este *super mensaje* se llama **CONTENEDOR**.

¿Cómo enviar esta información al servidor? El cliente utiliza un tipo de mensaje como el que se muestra en la figura:

donde:

- **tipo** (`uint16_t`): indica el tipo de mensaje (**GIRO**, **MOVIMIENTO** o **CONTENEDOR**)
- **longitud** (`uint16_t`): en caso de mensajes **GIRO** o **MOVIMIENTO**, indica la longitud en bytes del bloque de datos que viene a continuación. En caso de ser un mensaje de tipo **CONTENEDOR**, indica el número de mensajes que contiene.
- **datos**: la información adecuada según el tipo de mensaje, que se detalla a continuación:

Creación de un mensaje de tipo GIRO:

- El tipo de mensaje es el **50**.
- En el campo de datos, se coloca un valor de tipo `int16_t` que contiene el valor de giro.

Creación de un mensaje de tipo MOVIMIENTO:

- El tipo de mensaje es el **70**.
- En el campo de datos se colocan dos valores de tipo `int16_t` consecutivos, con el contenido de las coordenadas **x** e **y**, respectivamente.

Creación de un mensaje de tipo CONTENEDOR:

- El tipo de mensaje es **1024**.
- En el campo de longitud se coloca el valor **2**, que indica que en su campo de datos van a encontrarse dos mensajes.
- En el campo de datos se meten (en el orden que se quiera) un mensaje de tipo **GIRO** y a continuación un mensaje de tipo **MOVIMIENTO**, cumpliendo el formato ya descrito para cada uno de ellos.

Importante: todos los campos de todos los mensajes se han de codificar en formato de red antes de ser enviados al servidor.

Implemente este programa en el fichero **ejercicio3.c**

Ejercicio 4 [1.75 puntos]

Se pretende implementar de forma simulada un **servidor** de juego para el cliente del ejercicio anterior, que utiliza un protocolo de transporte fiable y espera conexiones en el puerto **5050**. Por cuestiones de escalabilidad, el servidor solo permitirá que se conecten **TRES JUGADORES**.

- Cada vez que se conecte un jugador, el servidor lo atenderá con un proceso hijo.
- El hijo espera recibir un mensaje de juego (**GIRO**, **MOVIMIENTO** o **CONTENEDOR**), lo decodifica, muestra por pantalla la información recibida (por ejemplo, el ángulo de giro, y/o las coordenadas **x** e **y**), y termina.

- Si hay un error en la decodificación del mensaje (el mensaje recibido no es de tipo **GIRO**, **MOVIMIENTO** o **CONTENEDOR**), hay que enviar una señal `SIGUSR1` al proceso padre. Cada vez que el padre reciba esa señal, incrementará un contador de fallos. Cuando el padre termine su ejecución, mostrará por pantalla el número de fallos de decodificación obtenidos.
- No se puede ignorar `SIGCHLD` en el padre. Primero hay que esperar correctamente a que terminen los tres hijos y luego terminará el programa servidor.

Implemente este programa en el fichero **ejercicio4.c**

Código de referencia (arriba, partes de un servidor orientado a conexión; tras la línea de separación, partes de un cliente): **también disponible en /home/alumno/Documentos/codigo.de.apoyo.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()*/
```

```
#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>
```