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

APELLIDOS, NOMBRE……
TITULACIÓN……
NÚMERO DEL PC…… AULA……

IMPORTANTE: SE ENTREGA UN ZIP CON LAS SOLUCIONES EN EL CAMPUS VIRTUAL

Ejercicio 1 (3 puntos)

Se pide implementar un cliente y un servidor de ECO (servicio en el que se envía un mensaje y se recibe el mismo) usando TCP con unas características especiales que se especifican a continuación **[véase el código de referencia de la última página del enunciado]**:

a) **Cliente (1 punto)**. Especificación:
   1. Los mensajes de ECO son cadenas de texto introducidas por teclado.
   2. Las cadenas se pasan al servidor enviando primero su longitud (de tipo `uint16_t`) y después la cadena en sí.
   3. Una vez enviada una cadena, el cliente espera la respuesta del servidor (que viene también en formato longitud + cadena) y la muestra por pantalla.
   4. El cliente continúa los pasos 1 a 3 hasta que el usuario introduzca la cadena "**fin**" (esa cadena finaliza el cliente, y no debe llegar al servidor).
   5. Se entrega un fichero con el nombre **ej1-cliente.c (o cpp).**

b) **Servidor (2 puntos).** Especificación:
   1. El puerto de escucha es el 4950.
   2. El servidor atiende en paralelo a cada cliente conectado.
   3. Por cada cliente, el servidor espera primero la longitud y después la cadena, y responde con el mismo formato. Esta acción se repite indefinidamente hasta que el cliente se desconecta.
   4. Se entrega un fichero con el nombre **ej1-servidor.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. Para la compilación del servidor:
   a. Si el servidor se compila con la opción "`-std=c99`", entonces la llamada al sistema `accept()` devuelve `-1` si es interrumpido por alguna señal, y hay que hacer que el servidor siga esperando clientes ante esta eventualidad.
   b. Si se compila sin la opción anterior, entonces `accept()` no se ve interrumpida ante la llegada de señales.

Ejercicio 2 (2 puntos)

A partir del sistema cliente/servidor ECO del Ejercicio 1, se pide implementar un cliente que permita enviar peticiones de ECO de manera asíncrona. Así, una vez se lee la cadena del teclado, se envía al servidor y, sin esperar respuesta, se vuelve a solicitar otra cadena al usuario desde teclado. Esto supone que el cliente debe multiplexar su entrada/salida ya que puede estar leyendo datos tanto por el teclado (por donde lee las cadenas del usuario), como por el socket, por donde el servidor responderá al ECO. Por lo demás, los detalles específicos de este cliente asíncrono son los mismos del ejercicio anterior, con las particularidades siguientes:

1. Cuando el usuario introduce la cadena "**fin**" se sale del programa, aunque pueda haber respuestas de ECO pendientes.
2. Se entrega un fichero con el nombre **ej2-cliente.c (o cpp).**

Notas:

1. Se debe garantizar que el cliente lee y escribe lo esperado.
2. Para simular mejor el comportamiento asíncrono, modifique el servidor del Ejercicio 1 únicamente incluyendo esta línea justo antes de responder al cliente (después de recibir su petición de ECO):

   ```
   sleep(5); /*Por favor, incluya <unistd.h> para usar sleep()*/
   ```

   Entienda que esta modificación es opcional, para probar bien el cliente de este ejercicio, pero no es necesario incluirla dentro de la solución ej1-servidor.c (o cpp).

Ejercicio 3 (2.5 puntos)

A partir del sistema cliente/servidor ECO del Ejercicio 1 (no es necesario usar el cliente asíncrono del Ejercicio 2), se pide implementar un proxy TCP, también multiproceso. El proxy es una aplicación que está en medio del cliente y del servidor, y toda la información que viaja entre ellos, pasa a través del proxy.

Especificación detallada:

1. El proxy actúa como un servidor TCP en el puerto 2119, que espera una conexión de un cliente de ECO. Es decir, el cliente de ECO se conecta al puerto del proxy, **no al del servidor ECO real**.
2. En el momento que se acepta la conexión, el proxy, en un nuevo hijo, crea otro socket TCP que actúa como cliente para conectarlo inmediatamente al servidor ECO real (al puerto 4950). A partir de aquí:
   a. El proxy espera un mensaje de ECO del cliente y lo reenvía al servidor de ECO real.
   b. A continuación, el proxy espera la respuesta de ECO del servidor, y la reenvía al cliente.
   c. Se vuelve al punto a), hasta que el cliente se desconecte del proxy, momento en el que se cierra también la conexión con el servidor de ECO y el hijo finaliza su ejecución.
3. Se entrega un fichero con el nombre **ej3-proxy.c (o cpp).**

Notas:

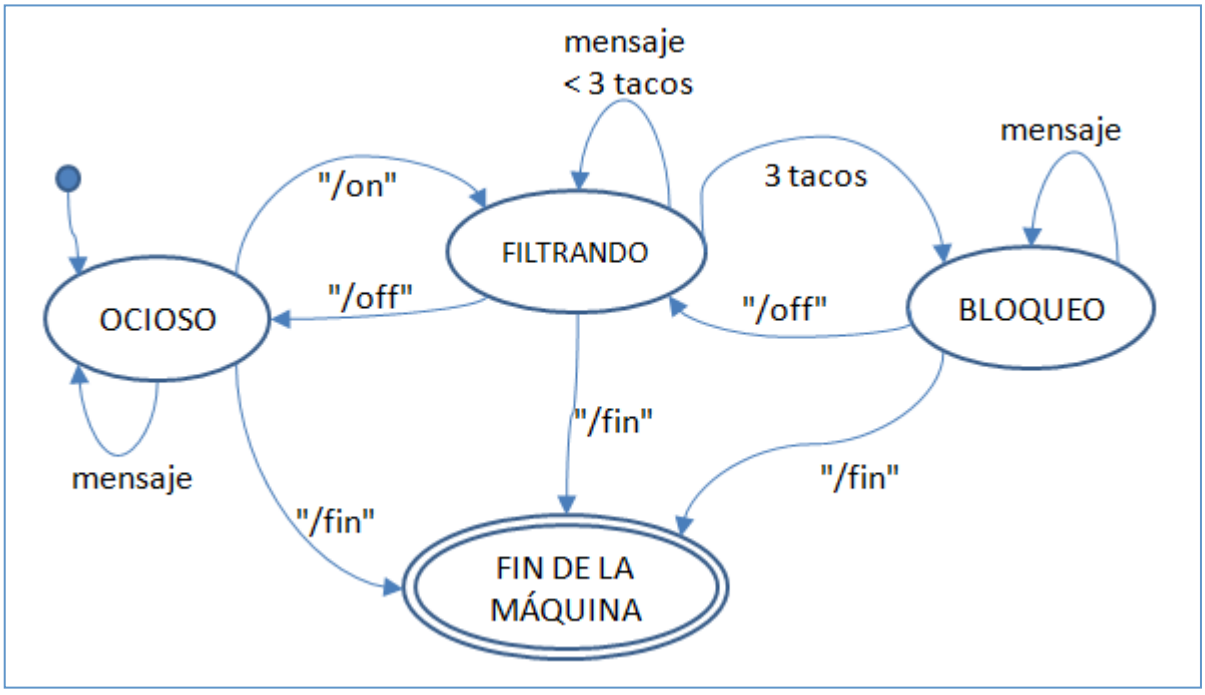
1. Se debe garantizar que el proxy lee y escribe lo esperado.
2. No deben quedar hijos zombies o huérfanos.
3. Compilación del proxy
   a. Si el servidor se compila con la opción "`-std=c99`", entonces el `accept()` devuelve `-1` si es interrumpido por alguna señal, y hay que hacer que el servidor siga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

Ejercicio 4 (2.5 puntos)

Diseñe una máquina de estados pensada para censurar información que circule por un proxy (**aunque no se tiene que integrar en los ejercicios anteriores, este ejercicio es independiente**).
¿Cómo se va a realizar esta censura? Al principio, y puesto que se va a utilizar en una simulación y no en un escenario real, el programa se limitará a imprimir por pantalla cadenas de texto que previamente se introduzcan por teclado. Sin embargo, al activar el modo censura (se explica más adelante como hacerlo), cada vez que se introduzca por teclado un mensaje que contenga la palabra "taco", la máquina lo cambiará por "xxxx" (manteniendo el resto del mensaje intacto) antes de imprimirlo por pantalla **[véase un ejemplo en el código de referencia de la última página del enunciado]**. A la tercera vez que se detecte la introducción de un "taco", la máquina dejará de imprimir mensajes, mostrando por pantalla el aviso "usted ya no esta autorizado a leer contenido".

La forma de activar la censura por filtro es mediante la introducción por teclado de la cadena "/on", para desactivar el modo de filtrado de tacos, o el de bloqueo total de información, se utiliza la cadena "/off". Nótese como, si se está en el modo de bloqueo total, un "/off" hace volver a la máquina al estado de filtrado de tacos, mientras que un segundo "/off" desactivaría la censura. Para salir de la máquina se utiliza la cadena "/fin".

SE ENTREGA UN FICHERO CON EL NOMBRE ejercicio4.c (o .cpp)



Código de referencia para los servidores:

```
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 los clientes:

```
/* 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()*/
```

Código de referencia para buscar una cadena dentro de otra:

La función para comparar cadenas es **`strstr()`**, cuya descripción se puede encontrar en el manual. Básicamente, se devuelve un puntero a la primera aparición de la subcadena dentro de la cadena original. En caso de no encontrarla, devuelve NULL. Nótese que podría haber más de una aparición, aunque sólo se muestra la primera. A continuación se muestra un ejemplo de su funcionamiento:

```
char * cadenaOriginal = "Vaya taco voy a decir";
char * puntero_a_taco = strstr(cadenaOriginal, "taco");
printf("%s\n", puntero_a_taco); // se imprime por pantalla
                                // "taco voy a decir"
```