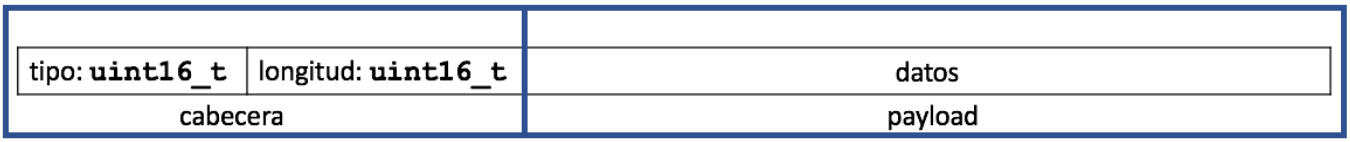
E.T.S.I. TELECOMUNICACIÓN
FUNDAMENTOS DE SOFTWARE DE COMUNICACIONES

APELLIDOS, NOMBRE:
DNI:
TITULACIÓN:
Número de PC:

Ejercicio 1

Para implementar un servicio fiable de envío de ficheros con un nivel de seguridad avanzado, se pueden utilizar funciones de resumen (**hash**) que permiten verificar al emisor si los datos se han interceptado y modificado en su tránsito por la red hasta el receptor (ataques de tipo "hombre-en-el-medio" o "man-in-the-middle"). En este ejercicio se solicita una implementación básica de este servicio en el que un cliente envía un fichero a un servidor que, cuando funciona en modo seguro, devuelve un valor resumen por cada bloque de fichero que recibe, para que el cliente pueda comprobar la integridad de los datos. El intercambio de información comienza con una fase de negociación en el que el servidor envía primero al cliente su nivel de seguridad actual.

Para el envío de datos del cliente al servidor, se usan mensajes con el siguiente formato:



donde:

- **tipo** (`uint16_t`): indica el tipo de mensaje, y que puede tomar dos valores:
  - `0`: mensaje con nombre de fichero
  - `1`: mensaje con bloque de fichero
- **longitud** (`uint16_t`): es la longitud del bloque de datos que viene a continuación.
- **datos**: que puede contener, bien el nombre de un fichero, o bien un bloque de datos del fichero, que el cliente transmite al servidor.

La especificación detallada de cliente y servidor se incluye a continuación. En todos los apartados se debe garantizar que se lee/escribe lo esperado y que los recursos reservados se liberan correctamente.

Apartado 1a. Servidor [1.5 puntos]

El servidor de almacenamiento de ficheros tiene los siguientes requisitos:

- El puerto de escucha es el `4950`.

- Recibe como argumento de entrada el nivel de seguridad con el que atiende a los clientes:
  - `0`: transferencia normal.
  - `1`: transferencia con envío del valor de resumen del bloque de datos.
- Si se quiere cambiar de nivel de seguridad, el servidor debe finalizar su ejecución de forma ordenada ante la llegada de la señal **`SIGINT`** y volver a ejecutarse con el valor del argumento entrada deseado.
- Funcionamiento cuando se acepta la conexión de un cliente:
  1. El servidor envía el nivel de seguridad al cliente, `0` ó `1`, en un **`uint8_t`**.
  2. Se recibe del cliente el nombre del fichero usando el formato de mensajes descrito anteriormente, y se crea en el sistema de archivos.
  3. A continuación, también usando el mismo formato de mensajes, se reciben y almacenan los bloques de datos del fichero, teniendo en cuenta que, si el nivel de **seguridad del servidor es 1**, después de recibir cada bloque, el servidor debe enviar al cliente un valor (**`uint32_t`**) de la función resumen que se muestra a continuación:

```
uint32_t resumen(char * b,int longitud) {
    uint32_t sum = 0;
    for (int i = 0; i < longitud; i++)
        sum += b[i];

    return sum*longitud;
}
```

  Dicha función tiene como argumentos el bloque de datos del mensaje recibido y su longitud. **Este envío del valor de la función resumen no hay que hacerlo cuando el nivel de seguridad es 0**.
  4. El cliente es atendido hasta que se desconecta, se cierra el fichero y se pasa a atender a un nuevo cliente.

Se entrega el fichero **servidor.c/cpp.**

Apartado 1b. Cliente [2 puntos]

El cliente del servicio de almacenamiento de ficheros tiene los siguientes requisitos:

- Recibe como argumentos de entrada la **IP** y **puerto de escucha** del servidor, y **el nombre del fichero** que se va a transferir.
- Una vez conectado al servidor, se recibe el nivel de seguridad con el que está funcionando (`0` ó `1`) en un **`uint8_t`**.

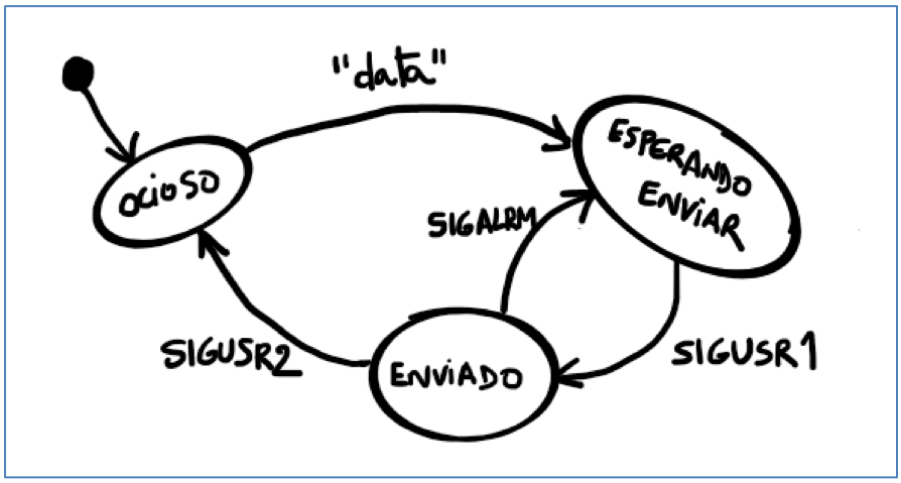
- Se abre el fichero recibido como argumento para lectura y, si no hay errores, se envía al servidor el nombre del fichero a transferir usando el formato de mensajes descrito arriba.
- Entonces se comienza con el envío del contenido del fichero, por bloques.
    1. Por cada bloque transmitido, y en caso de encontrarse en un **nivel de seguridad 1**, el cliente debe:
        i. Calcular el valor (**`uint32_t`**) de la función resumen del bloque enviado, usando para ello la misma función que la que se incluye en la descripción del servidor
        ii. Recibir el valor de la función resumen del servidor.
        iii. Comparar ambos valores. Si son iguales, se continua con el envío del siguiente bloque, pero si ambos valores difieren, entonces el cliente muestra por pantalla el siguiente mensaje y finaliza su ejecución:

        "**`Valor incorrecto de la función resumen. Posible interceptación de los datos`**",

    2. **Si el nivel de seguridad es 0, entonces el paso anterior no es necesario**.
- El cliente finaliza cuando ha transferido el fichero completo.

Se entrega el fichero **cliente.c/cpp.**

Ejercicio 2. Máquina de estados [2.5 puntos]



Dada la máquina de estados que simula parte del comportamiento de envío de un protocolo de transporte, impleméntela siguiendo estos requisitos.

- Antes de arrancar la máquina, cree una **fifo** en el sistema de archivos de nombre "**`fsc_fifo`**" (sin las comillas), y ábrala en modo lectura.
- La máquina comienza en el estado **`OCIOSO`**, y permanece en él hasta que, por la **fifo**, se reciba la cadena "**`data`**" (sin las comillas). Con este evento se

está simulando que hay datos encolados, y se pasa el estado **`ESPERANDO_ENVIAR`**.

- En el estado **`ESPERANDO_ENVIAR`**, la máquina espera la autorización de la capa inferior para enviar los datos, por lo que permanece aquí hasta que el proceso recibe la señal **`SIGUSR1`**. En ese momento, se simula el envío de datos -*NOTA*: vale con un **`printf`**`("datos enviados\n")` – se activa un temporizador de **1.5 segundos** y se pasa al estado **`ENVIADO`**.
- En el estado **`ENVIADO`**, si expira el temporizador, se vuelve al estado **`ESPERANDO_ENVIAR`** para simular que hay que retransmitir los datos. Por el contrario, si llega la señal **`SIGUSR2`**, se interpreta como que los datos han llegado bien al destino y la capa inferior lo ha notificado, por lo que se vuelve al estado **`OCIOSO`**, desactivando antes el temporizador que estaba en marcha.
- No se pueden perder señales, por lo que se aconseja utilizar una tubería de tipo **pipe** para encolarlas a medida que se vayan recibiendo.
- Recibir por la **fifo** una cadena distinta a "**`data`**" (sin las comillas), se considera un error y se sale de la máquina.
- Obviamente, la función destinada a esperar eventos debe tener en cuenta que puede llegarle información por la **pipe** o por la **fifo**.

Se entrega el fichero **ejercicio3.c/cpp.**

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 sockadd
/* ... */
???= 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>
```