Práctica 1

Mi capa de TCP/IP

Jorge Alejandro Flores Triana

ID: 714510

ITESO, Universidad Jesuita de Guadalajara

Instituto Tecnológico y de Estudios Superiores de Occidente

Profesor – Sergio Nicolás Santana

Tópico: Sistemas de Comunicación

Mi capa de TCP/IP

Link Video: https://drive.google.com/drive/folders/18rpOBIPAgmBrwcaXkDslcyxDzO0WqaTx?usp=sharing

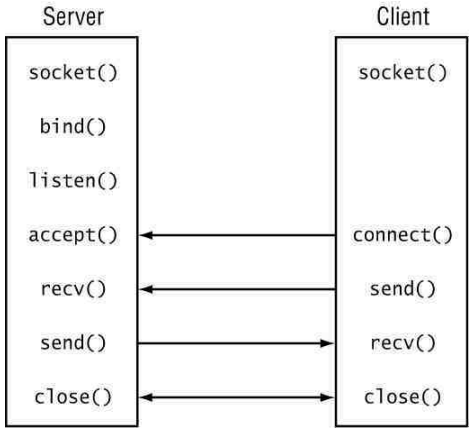
Link Código: https://github.com/a01633675/SistemasComunicacion

Objetivo:

El objetivo de esta práctica era crear una capa propia dentro del modelo de TCP/IP, el cual consistía en recibir/mandar una trama de TCP, encriptar/desencriptar el mensaje utilizando AES 128 y calcular el CRC del mismo.

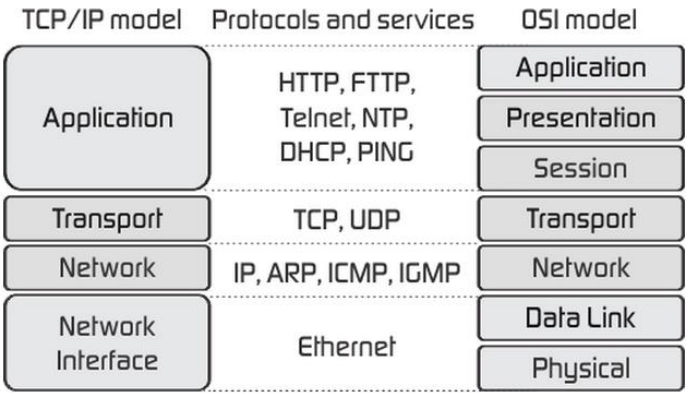
Implementación:

Los sockets son un mecanismo que nos permite establecer un enlace entre dos programas que se ejecutan independientes el uno del otro (generalmente un programa cliente y un programa servidor). En el modelo de TCP/IP la implementación de un cliente – servidor se vería de la siguiente manera.



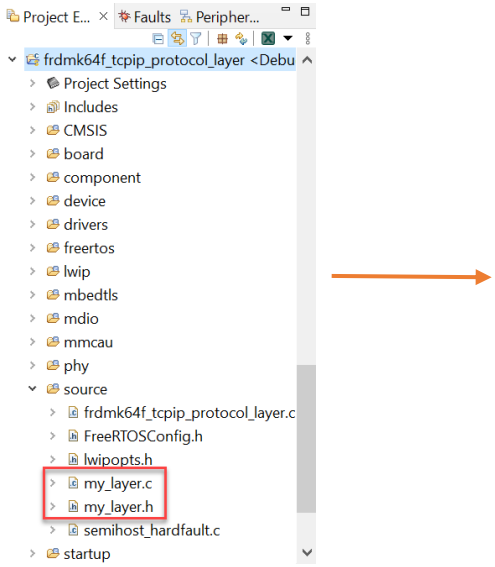
Como se puede observar, el servidor crea un socket y espera por la conexión de un cliente. Cuando el cliente hace la petición, el servidor acepta la conexión y la comunicación puede iniciar. Para el caso de nuestra práctica se decidió que la tarjeta Freedom K64 sería el servidor y la computadora el cliente.

Por otro lado, es importante tomar como referencia el modelo TCP/IP que se muestra a continuación para entender en que sección del modelo se iba a estar trabajando.



Para la práctica se tomó como base el ejemplo de TCP echo que consiste en mandar tramas de TCP desde un cliente hacia un servidor y viceversa, agregando las tareas de encriptado y cálculo de CRC.

En la práctica se implementó un driver que contiene las siguientes funciones que pueden ser utilizadas en por cualquier usuario.




```
struct netconn* my_socket_create (void)

struct netconn* my_socket_accept (struct netconn *conn)

char* my_socket_receive (struct netconn *conn)

err_t my_socket_send (struct netconn *conn, char *data)

void my_socket_close (struct netconn *conn)
```

Cada función puede ser utilizada independientemente y su implementación específica pude ser encontrada en el código fuente. Sin embargo, a continuación se da una pequeña descripción del código:

- my_socket_create (): Esta función crea un socket de TCP, donde por medio de unos #defines se puede configurar la dirección IP y el puerto que se quiere utilizar.

- my_socket_accept (): Esta función es implementada por el servidor para aceptar conexiones del cliente. Regresa el socket para entablar la comunicación.

- my_socket_receive (): Esta función recibe una trama de TCP y la separa en el cuerpo del mensaje y el CRC (últimos 4 bytes de la trama). Después, se calcula el CRC del cuerpo del mensaje y se compara con el valor recibido. Si son iguales, se procede a desencriptar el cuerpo del mensaje utilizando AES 128. Finalmente, se regresa un puntero con el mensaje original.

- my_socket_send (): Esta función recibe como parámetro un puntero con el dato que se quiere enviar. Posteriormente, se le hace un padding de ceros a 16 bytes y se encripta el mensaje con AES 128. Finalmente, se calcula el CRC del mensaje encriptado y se agrega el resultado a los últimos 4 bytes de la trama antes de ser enviada.

- my_socket_close (): El servidor Cierra la conexión y elimina el socket utilizado.

Una vez, con las funciones definidas, se implementó una tarea de FreeRTOS en donde se levanta el socket del servidor y se espera por una conexión. Después, cuando un cliente busca conectarse, se acepta la solicitud y se comienza a recibir/enviar mensajes.

El cliente (script de python en la computadora) envía un mensaje desde la consola y el servidor (Freedom K64) recibe la trama. Si el cálculo del CRC y el proceso de desencriptado son correctos, el servidor envía una respuesta al cliente y éste lo recibe para checar la integridad del mensaje. Este proceso de recibir y luego enviar se repite 8 veces. Finalmente, el servidor cierra la conexión.

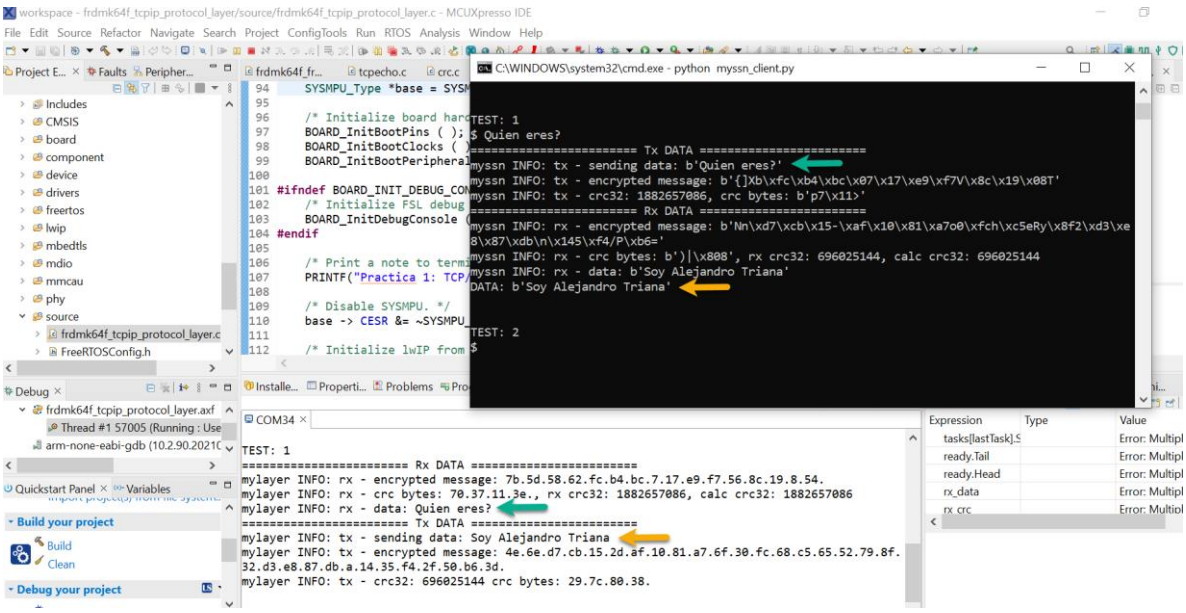
Resultados:

Para poder ver los resultados de la aplicación, se imprime el contenido de distintas variables en la terminal incluyendo el CRC, el mensaje encriptado y el mensaje original.

Como se observa en la primer imagen, el usuario escribe un mensaje en la consola para que el cliente lo envíe al servidor. El mensaje es: “Quién eres?”. El cliente encripta el mensaje y le agrega el CRC para posteriormente enviarlo. El servidor recibe el mensaje, lo valida y obtiene la información original como se muestra en la terminal del IDE.

Después, el servidor manda una un mensaje: “Soy Alejandro Triana”. Dicha respuesta se envía de manera encriptada y con su respectivo CRC. Finalmente, el cliente recibe la información, la valida y la decodifica para obtener el mensaje final, como se muestra en la consola.

Este proceso se repite 8 veces donde el servidor recibe un mensaje por parte del cliente y una vez validado, envía una respuesta previamente definida por la aplicación.



Conclusión:

Durante la práctica aprendí a identificar de mejor manera las distintas capas del modelo TCP/IP y cómo se relacionan dentro de una aplicación. Esto dejo en claro uno de los objetivos que es mantener independientes cada nivel de la aplicación, facilitando su mantenimiento debido a la modularidad y a la escalabilidad. Por otro lado, a nivel de la implementación fue interesante crear una capa propia para entender cómo se construyen las aplicaciones en casos de uso de la vida real.

Por otro lado, hablando de la práctica en sí, creo que ayudó en gran parte haber corrido los ejemplos de TCP en clase ya que nos ayudaron a entender de mejor manera cómo funciona el stack de lwip. En cuanto a las dificultades, creo que el ejercicio de encriptar y calcular el CRC fue algo nuevo para mi ya que nunca había visto temas relacionados con la seguridad. Y aunque no es el objetivo principal de la clase el saber encriptar, creo que sirvió en gran medida para entender cómo se construyen las capas.