# 75:42 - Taller de Programación I

Ejercicio Nº ___ Padrón _____

Alumno _____ Firma _____

| **Nota**: | | **Corrige**: | | **Entrega #1** |
|---|---|---|---|---|
| | | | | Fecha de entrega |
| | | | | |
| | | | | Fecha de devolución |
| | | | | |
| | | | | |

| **Nota**: | | **Corrige**: | | **Entrega #2** |
|---|---|---|---|---|
| | | | | Fecha de entrega |
| | | | | |
| | | | | Fecha de devolución |
| | | | | |
| | | | | |

El presente trabajo, así como la entrega electrónica correspondiente al mismo, constituyen una obra de creación completamente personal, no habiendo sido inspirada ni siendo copia completa o parcial de ninguna fuente pública, privada, de otra persona o naturaleza.

# Índice general

# 1 Introducción

El trabajo practico tiene como objetivo utilizar sockets TCP para simular un shield Ethernet y mediante un archivo que contiene datos en binario simular como evoluciona la lectura de un sensor de temperaturas del arduino.

El mismo se puede dividir en dos partes, un cliente y un servidor. Estos utilizan sockets TCP para simular el shield Ethernet. El cliente envía un petitorio http y espera la respuesta del servidor, que en caso de que el petitorio sea correcto recibe un template con los datos correspondientes.

# 2 Problemas afrontados

A medida que se realizó el presente trabajo práctico surgieron distintos problemas. Se ira mencionando y describiendo cada uno de ellos, dando finalmente la solución a la que se llego.

## 2.1. Problemas de diseño

En esta sección se mencionaran los problemas relacionados con el diseño del programa.

El problema más difícil de afrontar fue definir que TDA se utilizarían para encapsular correctamente el uso de sockets. Se probaron distintas implementaciones, la primera fue tener dos sockets donde uno era un socket para el cliente y otro socket para el servidor. Esta implementación se concluyó que no era correcta ya que además de que se copiaban en ambos sockets funciones como send y recive, el servidor tenia en el mismo TDA un socket pasivo para aceptar conexiones y un socket de comunicación. Esta implementación fue modificada de manera que haya un TDA para encapsular un socket de comunicación que lo utilizaba el cliente y el servidor, y por otro lado se creo un TDA para el socket pasivo que acepte las conexiones del lado del servidor. Como el socket que acepta las conexiones del lado del servidor crea un nuevo socket de conexión se decidió utilizar dos constructores en el TDA que encapsulaba el socket de conexión. En el siguiente capitulo se mostrara en detalle un esquema que muestra como se relacionan el cliente y servidor con los TDA utilizados para encapsular los sockets. Este problem de diseño fue uno de los mas difíciles de afrontar ya que hubo que hacer varios refactors e invertir mucho tiempo antes de llegar a la solución final.

Para guardar la el user-agent de cada cliente y su cantidad de visitas se tuvieron en cuenta distintas posibilidades. La primera fue utilizar dos TDA, uno que sea una lista enlazada para guardar en orden las visitas hechas al servidor y otra un hash table donde se tenga el user-agent y la cantidad de visitas. A pesar de que esta forma fue la propuesta mas rápida se opto por tener una lista enlazada que guarde el user-agent y la cantidad de visitas hechas. Por cada cliente que se conecta es necesario recorrer toda la lista para ver si ya se conecto una vez o si es la primera. En caso de que ya se halla conectado se le suma una visita y si no se agrega al final. Dado a que no son muchos la cantidad de clientes que se conectan y no se especifico que debe ser lo mas eficiente posible en velocidad se opto por el uso de un solo TDA que contenga toda la información (teniendo en cuenta que para casos con pocos clientes la diferencia de velocidad no sera grande).

Definir como el servidor iba a armar el template que responde al cliente fue otro de los problemas de diseño. Para ir actualizando la temperatura que el template contenia y que quede encapsulado correctamente se decidió utilizar dos TDA, uno que fuera el

template mismo y otro que maneje el archivo binario del cual se leían y actualizaban las temperaturas.

## 2.2. Problemas en la práctica

En esta sección se explicaran los problemas que están mas relacionados con el lenguaje, esto incluye perdidas de memoria y manejo de variables. Para solucionar este tipo de problemas se utilizaron distintas herramientas, en algunos casos basto con imprimir por stdout el valor de algunas variables y en otros bugs mas difíciles de encontrar hubo que recurrir al uso de gdb, valgrind y comandos como strace.

El bug mas complicado de encontrar fue al hacer un refactor, no se actualizo correctamente el valor de s del struct socket_connect en la funcion addrinfo_socket(socket_connect_t *skt_c,char *hostn, char* srvn); del TDA common_TDA_socket_connect.c. Lo que ocurrió fue que en el constructor del TDA se inicializa el dato s = 0 , entonces dentro de la función mencionada estaba creando una copia de s y se actualizaba únicamente la copia. El compilador no encuentra este tipo de errores y valgrind lo único que mostraba un "Segmentation fault"del lado del servidor, cuando lo único que se habia modifcado era el cliente. Parecía que el cliente estaba enviando bien los mensajes y el servidor lo único que recibía era NULL. Luego de utilizar el comando strace se veía que el connect estaba dando 0 pero el send decia que realizaba una operación en un socket no valido. Esto ocurría porque se actualizaba la copia de s pero cuando salia de la funcion seguía valiendo 0 (la copia de s estaba recibiendo lo que devolvía la funcion socket(...) dentro de la función mencionada).

Para los casos en que se estaba perdiendo memoria fue muy útil valgrind ya que muestra donde ocurre el error utilizando los flags adecuados. Hubo varios problemas de este tipo cuando se enviaban y recibían mensajes entre el cliente y el servidor.

# 3 Diagramas

En este capitulo se mostraran distintos diagramas que explicaran como esta implementado el tabrajo.

## 3.1. Realacion del cliente y servidor con los TDA

El siguiente diagrama muestra cuales tda usa el cliente. Fueron dos TDA, uno que se encarga de procesar el request que se debe enviar al servidor y el otro es el tda que encapsula el socket de conexion con el servidor.


Figura 3.1: Relacion entre el cliente y los TDA

El siguiente diagrama muestra cuales tda usa el servidor, se puede ver que este fue mas complejo que el cliente. El servidor se relaciona con seis TDA. Tiene dos TDA que encapsulan los sockets de conexcion y el otro el socket pasivo que recibe la conexion y la acepta. Utiliza un TDA lista que se encarga de guardar los user-agent que se conectan al servidor de forma ordenada y la cantidad de visitas de cada uno. Aparte hay un TDA request que es utilizado para el procesamiento del request recibido, verifica si es mismo valido. Por ultimo estan los TDA que encapsulan el manejo del archivo que simula las temperaturas del sensor arduino y el template que se respondera al cliente.

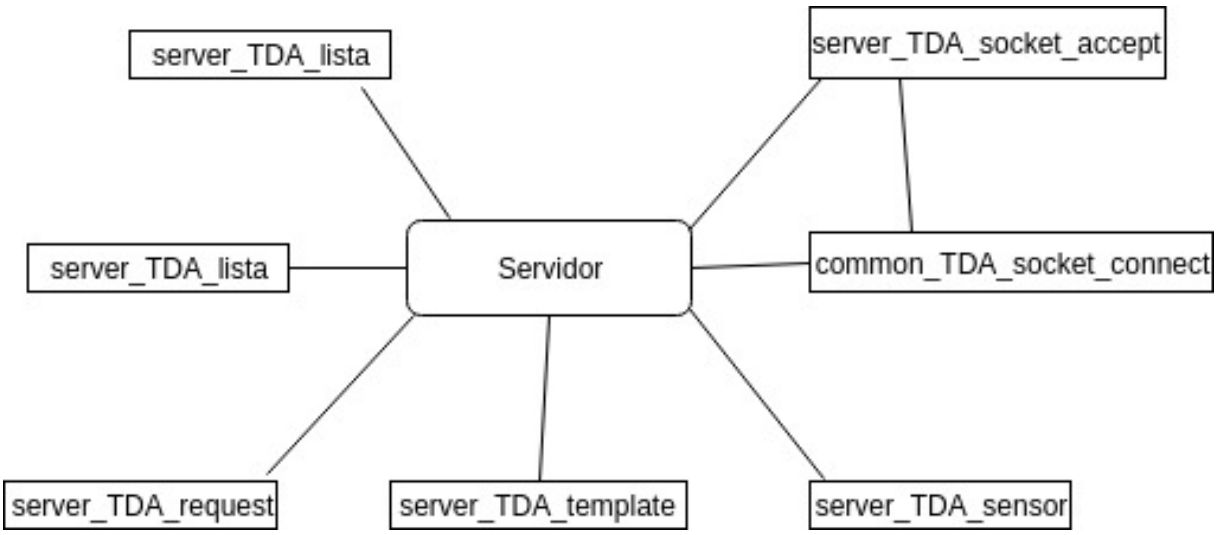


Figura 3.2: Relacion entre el servidor y los TDA

## 3.2. Comunicacion del cliente y servidor

Este diagrama trata de explicar como se establece la relacion que hay entre el cliente y el servidor. Hay dos TDA, uno que se encarga de aceptar una conexion al servidor y otro que es el socket por el cual se comunica el cliente y el servidor.

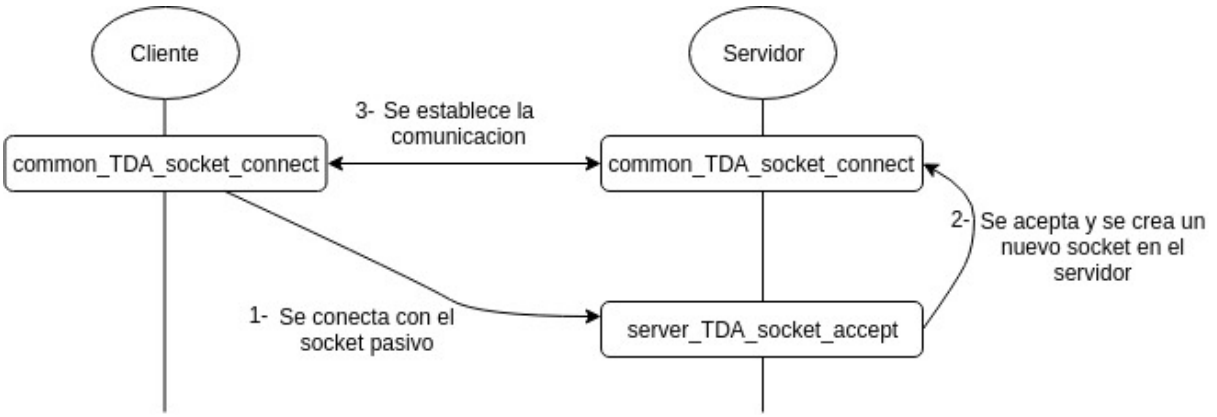


Figura 3.3: Comunicacion del cliente y servidor

## 3.3. Secuencia del programa

La siguente figura muestra en forma general los pasos que sigue el programa en caso de una ejecucion exitosa, donde se supone que el request del cliente es valido.

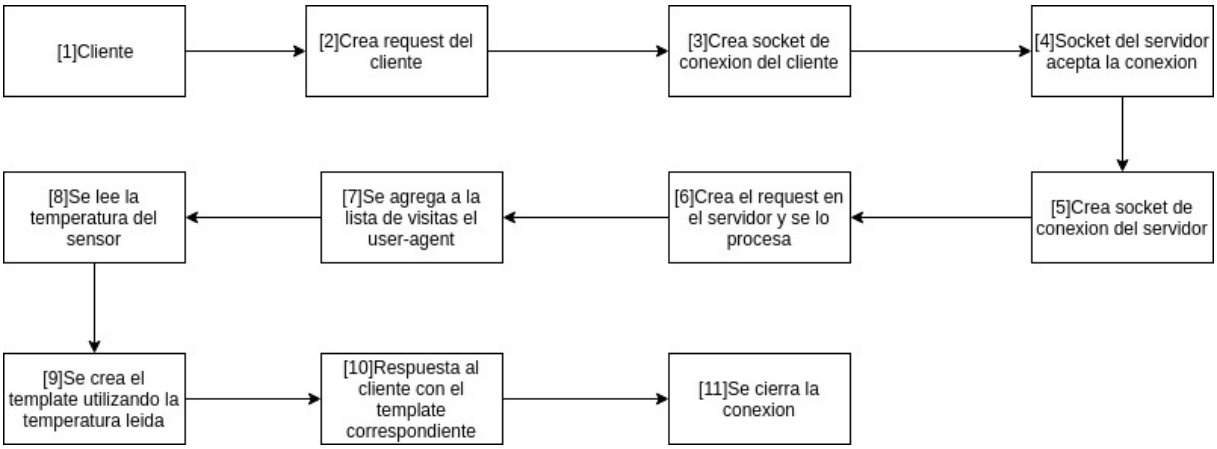


Figura 3.4: Secuencia del programa

# 4 Bibliografia


- https://github.com/Taller-de-Programacion/clases/tree/master/sockets/src
- https://aticleworld.com/socket-programming-in-c-using-tcpip/
- https://www.tutorialspoint.com/c_standard_library/index.htm