CÁTEDRA DE SISTEMAS OPERATIVOS II

Departamento de Computación

FCEFyN - UNC

Trabajo Práctico 1 - Sockets

Maero Facundo
Mat: 38479441

Córdoba, Abril de 2017

Índice

1. Introducción

En el presente Informe se mostrará el proceso de diseño, desarrollo, testing y documentación de un software de conexión, control y transferencia de datos telemétricos entre el un cliente y el servidor que contiene los datos de las AWS (arquitectura cliente servidor), desarrollado en C.

1.1 Propósito.

El propósito de este Trabajo Práctico es aprender a utilizar la estructura de sockets de Linux, sus características y limitaciones, el uso de sus diferentes opciones (orientados o no a conexión), y su posterior implementación y testing.
Los conceptos deben aplicarse a un caso particular referido a estaciones sensoras y los archivos que las mismas generan, permitiendo acceder a los mismos de forma ordenada y prolija desde la aplicación cliente.

1.2 Ámbito del Sistema.

El sistema se conforma de dos partes: la aplicación servidor, que se ejecuta en una placa Intel Galileo, y la aplicación cliente, que se ejecuta en la computadora del usuario. El cliente se comunica con el servidor, le envía comandos, y muestra por terminal los resultados que obtiene.

1.3 Definiciones, Acrónimos y Abreviaturas

- TCP: Transmission Control Protocol
- UDP: User Datagram Protocol

1.4 Descripción General del Documento

El presente documento muestra una visión del proceso desarrollo del trabajo práctico, su diseño, implementación, esquemas a seguir y conclusiones sacadas del mismo.

2. Descripción General

2.1 Perspectiva del Producto

El producto solicitado es un sistema que gestione la conexión con una plataforma servidor, para acceder a los datos telemétricos de diferentes estaciones sensoras. Las estaciones proporcionan sus datos al servidor, que los recopila en un archivo separado por comas (.CSV), el cual debe ser accedido e interpretado.

2.2. Funciones del Producto

Las funciones que proporciona el producto son:

- Conexión entre cliente y servidor mediante dirección IP y puerto.
- Autenticación del usuario mediante user y password.
- Acceso a los datos de las estaciones mediante diferentes funciones que brindan información más o menos específica sobre los sensores y las estaciones.

2.3. Características de los Usuarios

El usuario del programa no necesita tener conocimientos avanzados de Linux o sockets. Simplemente siguiendo las instrucciones en pantalla puede interactuar con el sistema, y obtener los resultados que solicita.

Cierto conocimiento de uso de una terminal puede ser necesario, ya que la ejecución del programa se realiza por esa vía.

2.4. Suposiciones y Dependencias

- Se supone que el servidor posee un archivo con la base de datos, llamado “datos_meteorologicos.CSV”, de donde se leerán los resultados de las mediciones.
- El servidor se inicializa en un puerto fijo (6020).
- El proyecto se ejecutará en una computadora con entorno Linux.
- Existe conectividad entre el servidor y el cliente, para poder realizar la comunicación.
- No hay ninguna dependencia específica, el proyecto debe funcionar en una distribución Linux estándarm utilizando make para compilar el código fuente.

3. Requisitos Específicos

3.1. Interfaces Externas

Para interactuar con el programa, se utiliza la interfaz estándar (terminal) en el cliente. Por este medio se muestran resultados al cliente, y se introducen comandos dirigidos al servidor.

El servidor muestra por pantalla los comandos recibidos e información de estado, pero no acepta comandos por parte de un usuario, es un elemento pasivo que responde solo a instrucciones del cliente.

3.2. Funciones

El servidor acepta los siguientes comandos:

- **listar**: muestra una lista de las estaciones de las que se tienen datos, y una lista de los sensores disponibles de cada una.
- **descargar <nro_estacion>**: descarga un archivo “descarga.txt” en el cliente, con el cotenido total de la estación solicitada.
- **diario_precipitacion <nro_estacion>**: muestra las precipitaciones totales por día de la estación dada.
- **mensual_precipitación <nro_estación>**: muestra las precipitaciones acumuladas en el mes de la estación dada.
- **pomedio <variable>**: muestra el promedio total de la variable solicitada, para todas las estaciones en la base de datos del servidor.
- **desconectar**: finaliza la sesión del usuario.
- **ayuda**: muestra un mensaje de ayuda con los comandos disponibles.

3.3. Requisitos de Rendimiento

El proyecto debe funcionar en cualquier computadora con especificaciones medianamente modernas. Ya que el grueso del procesamiento se realiza en el servidor, y como el mismo se ejecuta el la plataforma Intel Galileo, puede decirse que el programa no es muy demandante a nivel de cálculo y procesamiento.

3.4. Restricciones de Diseño

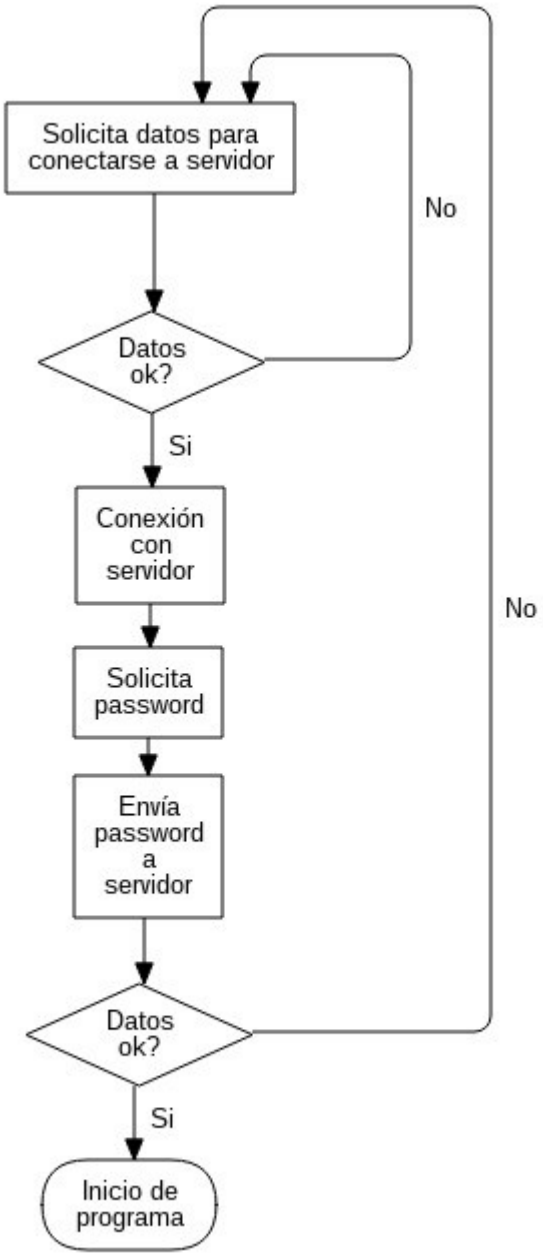
- La descarga de archivos debe realizarse utilizando conexión no segura (sin conexión).
- Durante la operación de transferencia del archivo, se debe bloquear la interfaz de usuario del programa, de tal forma que no se puedan ingresar nuevos comandos en la aplicación hasta que no haya finalizado la transferencia.
- Debe incluirse un mecanismo de control y manejo de errores por parte del servidor con comunicación al cliente.
- Todos los procesos deben ser mono-thread.
- No deben utilizarse IDEs para el desarrollo, prefiriendo el uso de editores de texto y archivos Makefile para la compilación y linkeo de archivos.
- Para la compilación es recomendable utilizar flags como -Werror, -Wall y -pedantic.

4. Diseño de solución

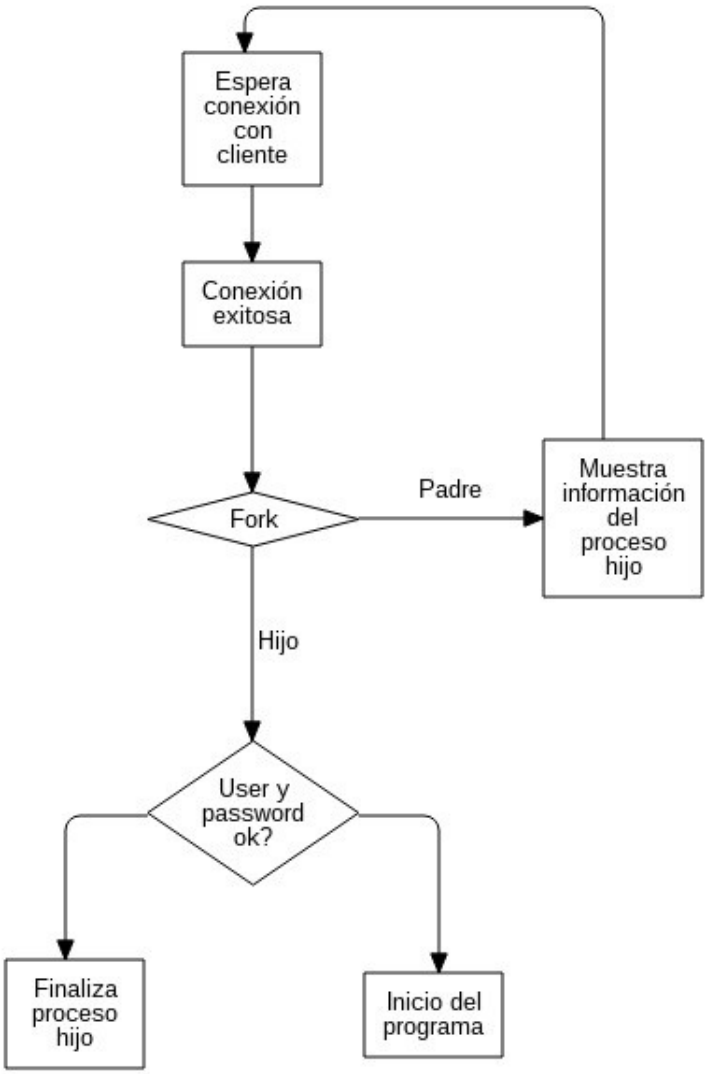
Para diseñar el software que cumpla estos requisitos, se utilizaron como base los códigos proporcionados por la cátedra, que brindan una comunicación por sockets orientados a conexión (utilizando TCP). A partir de allí, se procedió a diseñar la estructura de datos para alojar la información contenida en el archivo separado por comas.

Luego se extendió la funcionalidad de los programas base, diseñando un esquema de acceso y autenticación. A continuación se muestran los diagramas de flujo que explican el proceso, tanto para el cliente como para el servidor.

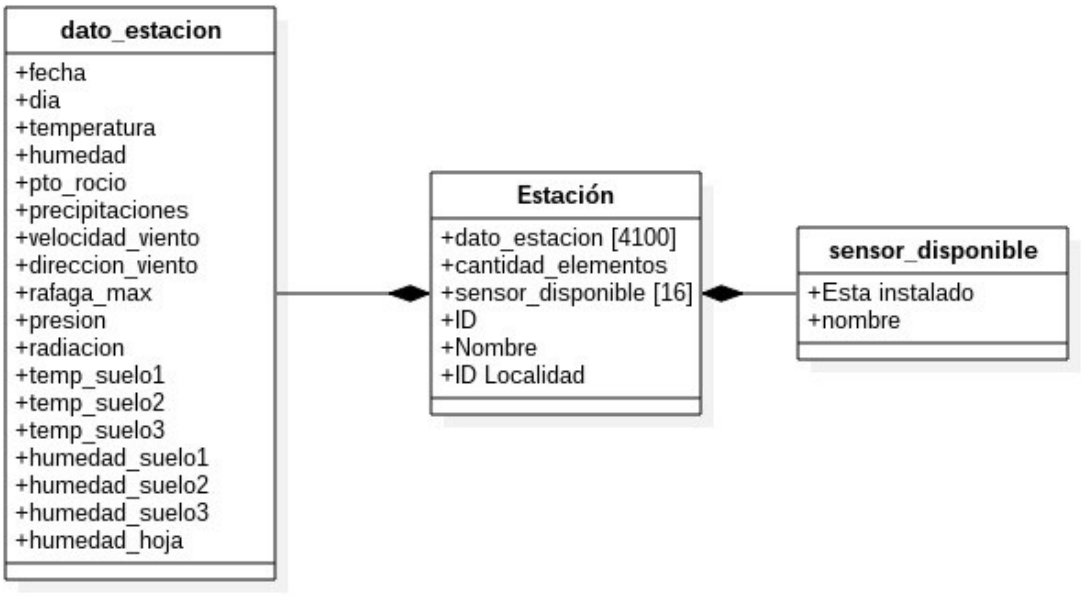
1) Autenticación cliente.



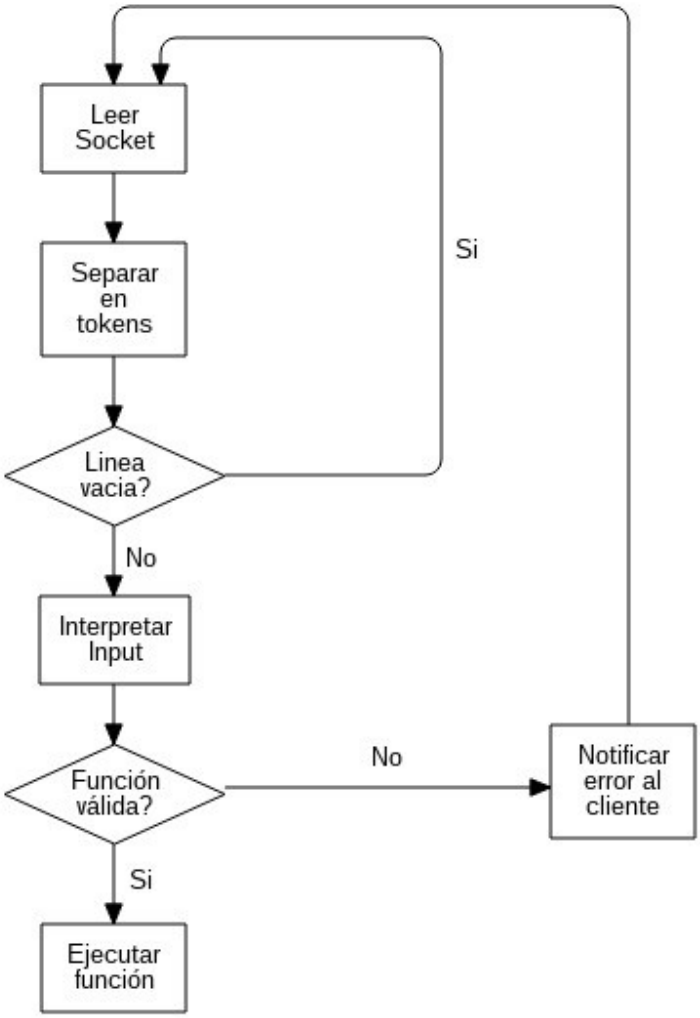
2) Autenticación servidor.



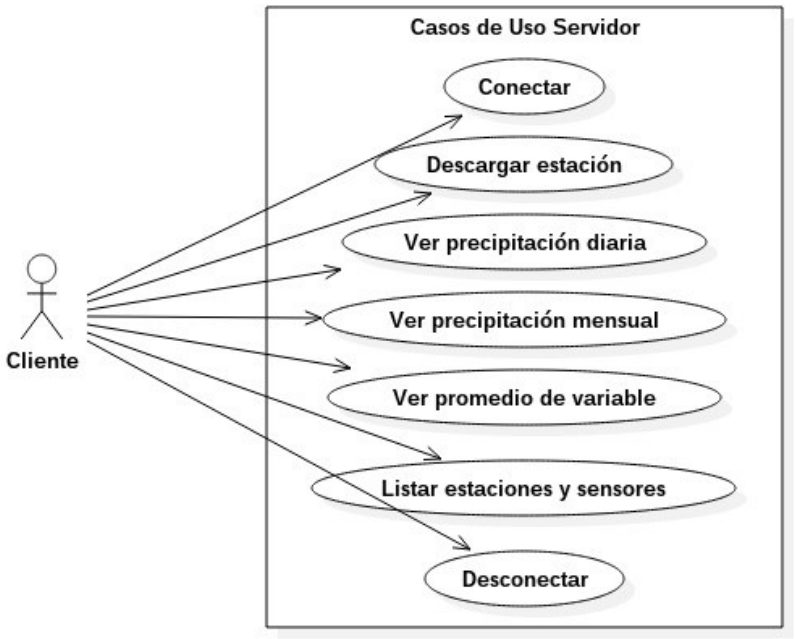
El archivo .CSV es leído por el servidor, y la información es alojada en una estructura de tipo Estación, que se organiza de la siguiente manera:



Luego, cuando el servidor recibe una cadena de instrucciones, la procesa y decide qué opción es la solicitada por el cliente. La secuencia de pasos es la siguiente:

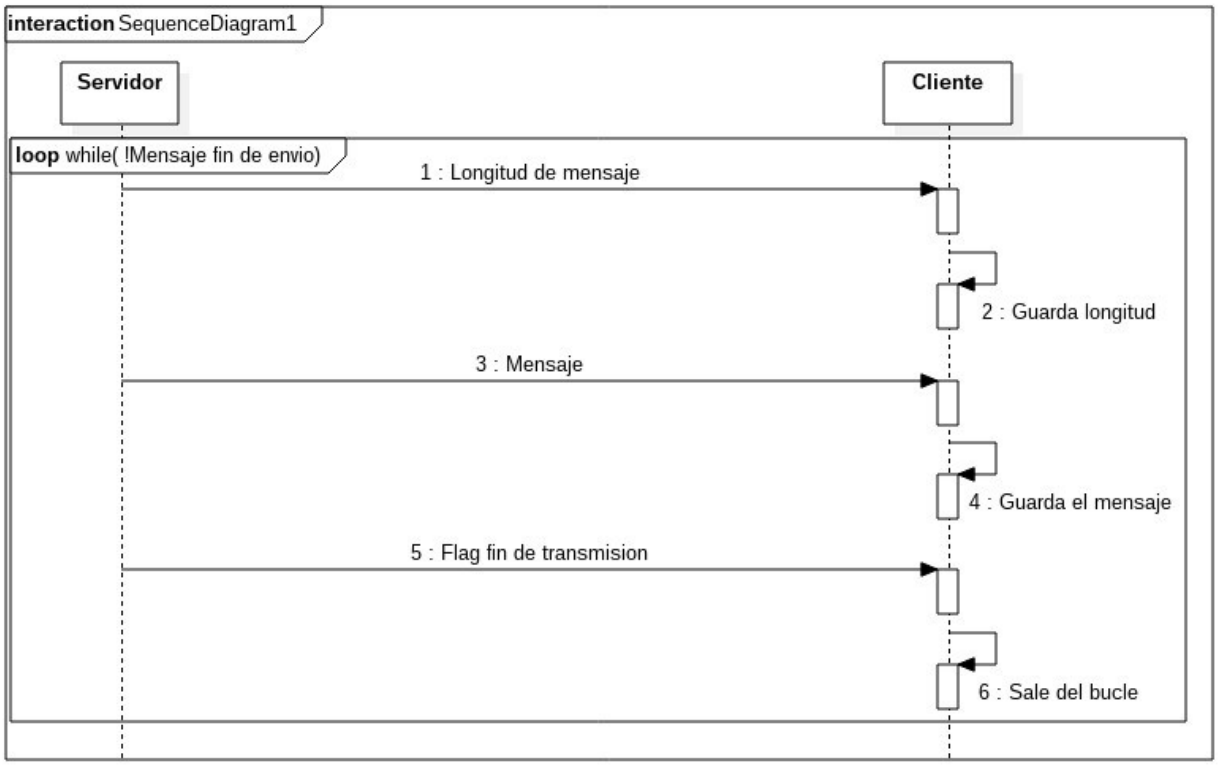


Desde el punto de vista del cliente, el diagrama de casos de uso que refleja los requerimientos solicitados es:



Para el diseño de la transferencia de datos, se siguió el esquema siguiente.

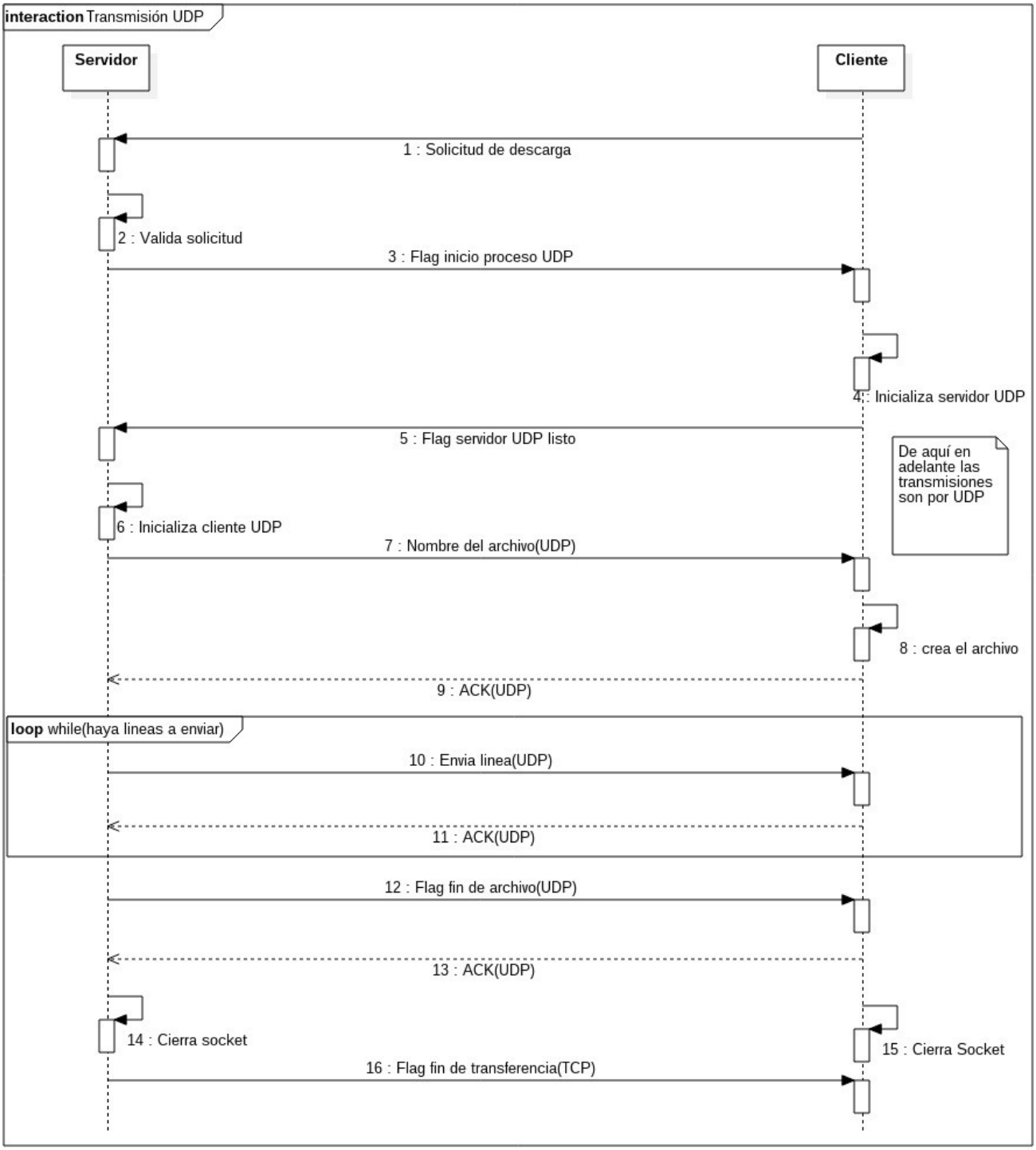
En el caso de la transferencia regular de información, se utilizó TCP, aprovechando las facilidades que brinda y la orientación a conexión del protocolo. Se agregó la transferencia del tamaño del buffer a ser enviado, previo envío del mensaje, para notificar al otro extremo de antemano y asegurar que se lea el mensaje en su totalidad. El cliente entra en un bucle infinito, donde lee indefinidamente mensajes provinientes del servidor, hasta que reciba un mensaje especial de fin de transmisión. Cuando esto suceda, se considera finalizada la recepción de información y se sale del bucle.



Para el caso de la transferencia de archivos mediante conexión no segura, la secuencia de pasos se vuelve más compleja:

- Aquí es necesario esperar un mensaje de acknowledgement, para así asegurarse que los envíos se producen en orden.
- El cliente solicita la descarga de un archivo. Si la solicitud es válida, el servidor envía un flag que inicia en ambos miembros el proceso UDP.
- El cliente inicializa un servidor UDP, y avisa al servidor cuando esté listo. Luego, éste envía el nombre del archivo, y comienza a transmitir información, hasta que finalmente envía un flag de fin de transferencia.
- Todas las transmisiones esperan un ACK antes de enviar el fragmento siguiente.

El diagrama de secuencia que explica este proceso es el siguiente:


interaction Transmisión UDP
Servidor
Cliente
1 : Solicitud de descarga
2 : Valida solicitud
3 : Flag inicio proceso UDP
4 : Inicializa servidor UDP
5 : Flag servidor UDP listo
De aquí en adelante las transmisiones son por UDP
6 : Inicializa cliente UDP
7 : Nombre del archivo(UDP)
8 : crea el archivo
9 : ACK(UDP)
loop while(haya lineas a enviar)
10 : Envia linea(UDP)
11 : ACK(UDP)
12 : Flag fin de archivo(UDP)
13 : ACK(UDP)
14 : Cierra socket
15 : Cierra Socket
16 : Flag fin de transferencia(TCP)

5. Implementación y Resultados

Para la implementación y desarrollo se utilizó el software Sublime Text. En un primer momento se trabajó sobre el archivo .CSV únicamente para testear funciones de lectura y parseo, verificar que las estaciones se llenaban con los datos correctamente.

En particular fue un desafío establecer la comunicación entre cliente y servidor de manera exitosa, ya que se tuvieron que revisar conceptos teóricos de los protocolos, entender como funcionan y realizar diferentes implementaciones de prueba que salven las falencias encontradas.
Al principio se llamaba a las funciones de lectura y escritura indiscriminadamente, pero al trabajar TCP con una secuencia de bytes, y no con paquetes estrictos, dos write pueden ser leídos por un solo read, por lo que no hay una relación 1 a 1, y el programa se quedaba esperando input que no iba a llegar nunca.

Para salvar este problema se transmitió el tamaño del mensaje antes del mensaje mismo, de modo que el extremo receptor llama a la función read con un tamaño de buffer correcto cada vez.

El cálculo de los resultados solicitados y el desarrollo de las aplicaciones fue resuelto sin mayores inconvenientes, salvo la función descargar, explicada más adelante, y la función calcular promedios.

Según Wikipedia, **reflexión** es la capacidad de un programa para observar y opcionalmente modificar su estructura de alto nivel. En otras palabras, es la capacidad de observar introspectivamente sus variables y utilizarlas en tiempo de ejecución.

Esta característica quiso utilizarse en un primer momento, ingresando el cliente el comando “descargar temp” por ejemplo. El servidor, tokenizando la palabra “temp” y guardádola en una variable cualquiera, podría acceder al campo de la estructura de datos dado por el **contenido de la variable,** no por su nombre.
Este tipo de comportamiento no es posible en C, por lo que hubo que buscarse otra alternativa.

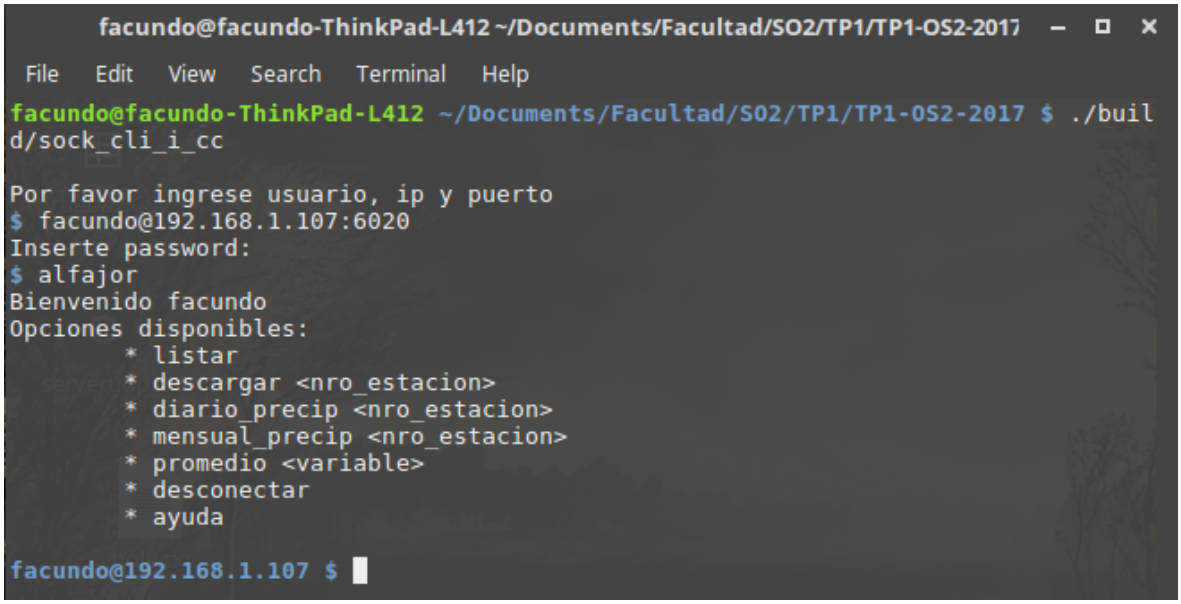
El enfoque utilizado es el siguiente: mediante la macro offsetof() se puede obtener la distancia en bytes desde el inicio de una estructura hasta un elemento de la misma. Al saber esto, se puede, con aritmética de punteros, recorrer el arreglo de datos en cada estación y acceder a los valores de la variable solicitada.
Para identificar la variable solicitada no hubo otra opción que una cadena de if else que revise la entrada del usuario, y en función de la coincidencia con el nombre de la variable, calcular el offset necesario.
Se gana a la hora de recorrer las estaciones en un bucle, ya que no se duplica código innecesariamente.

Con respecto a la transmisión de archivos por UDP, se realizó un procedimiento similar al three way handshake de TCP, implementando ACKs al recibir un mensaje.
Cabe mencionar que debe enviarse la dirección IP del cliente al servidor, para poder conectarse mediante UDP. Esto no está implementado, y para que la función descargar se ejecute correctamente, debe escribirse la dirección IP del cliente en el código que inicializa el cliente UDP (archivo sock_srv_i_cc.c línea 227).
La implementación ideal sería: el cliente genera el servidor UDP y le envía al servidor su dirección IP en tiempo de ejecución, sin hardcodear el valor.

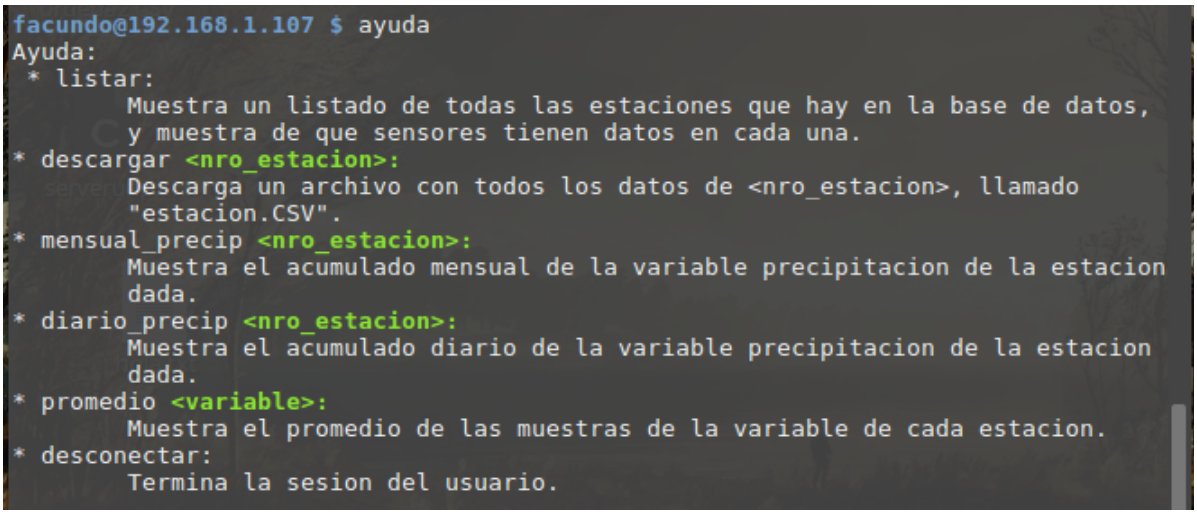
Con respecto a la estructura general del proyecto, se hizo un Makefile que compile y realice el linkeo de los archivos, guardando los .o en la carpeta obj, y los binarios finales en la carpeta build. Se incluyó un target para ejecutar CppCheck sobre el proyecto, guardando los resultados en el archivo err.txt.
Luego de un make clean se eliminan todos los archivos objeto, binarios compilados y logs de CppCheck, dejando solamente archivos .c y .h, además del log .CSV.

A continuación se adjuntan capturas de pantalla del programa en ejecución:

1) Ejecución del programa cliente, proporcionando usuario, IP y puerto. Luego se solicita la contraseña. Finalmente se muestra un mensaje de bienvenida al usuario, y una lista de opciones disponibles a ejecutar.



2)Output de la funcion ayuda.



3)Output de la funcion promedio, con un sensor instalado en todas las estaciones, otro instalado solo en algunas, y un sensor inexistente.

```
facundo@192.168.1.107 $ promedio humedad
Promedios variable humedad:

30135 - Yacanto Norte - E01: 69.3
30057 - MagyaCba 60 cuadras: 81.4
30061 - MagyaCba La Cumbrecita: 68.4
30099 - Cerro Obero - Capilla del Monte: 79.3
30069 - Magyacba Oliva: 83.5
facundo@192.168.1.107 $ promedio radiacion
Promedios variable radiacion:

30135 - Yacanto Norte - E01: No se encuentran datos.
30057 - MagyaCba 60 cuadras: 234.9
30061 - MagyaCba La Cumbrecita: 236.3
30099 - Cerro Obero - Capilla del Monte: No se encuentran datos.
30069 - Magyacba Oliva: 303.6
facundo@192.168.1.107 $ promedio calabaza
Sensor inexistente
facundo@192.168.1.107 $
```

4) Salida parcial de la función listar, donde se muestran las primeras 3 estaciones y sus sensores disponibles.

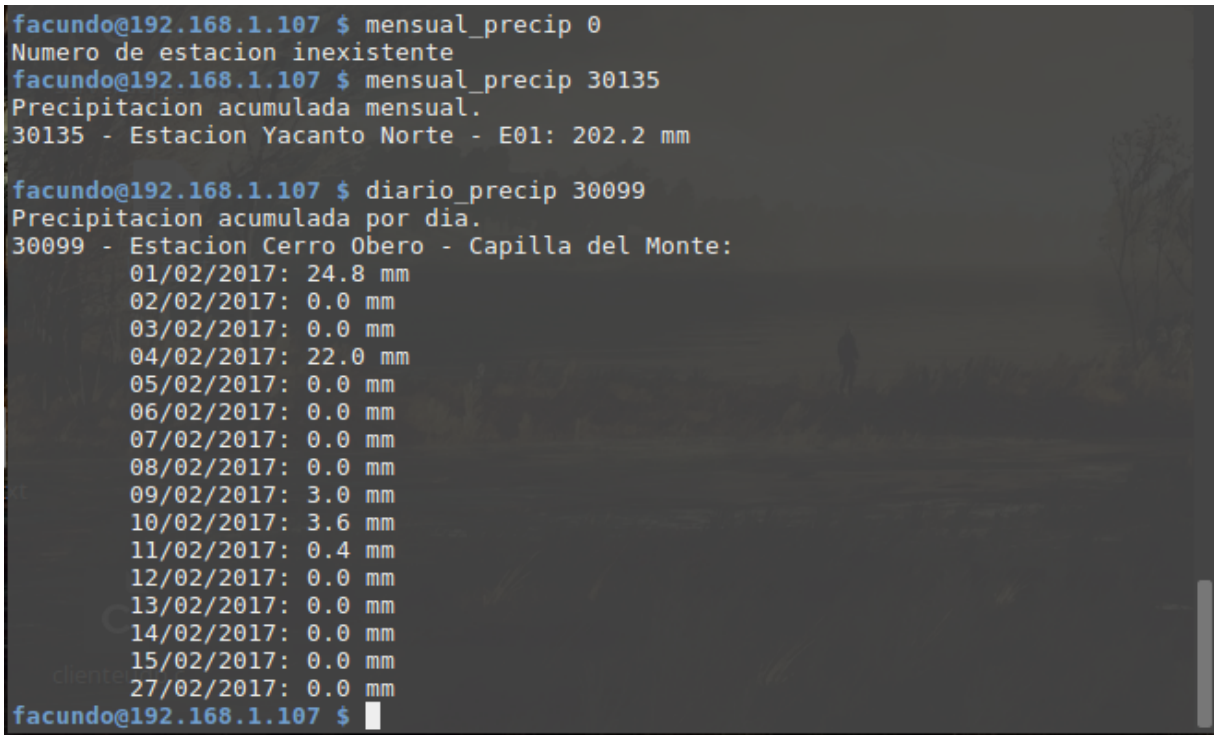
```
facundo@192.168.1.107 $ listar
Estaciones disponibles:

30135 -  Yacanto Norte - E01. Sensores con datos:
    Temperatura [ºC]
    Humedad [%HR]
    Punto de Rocío [ºC]
    Precipitación [mm]
    Velocidad Viento [Km/h]
    Dirección Viento
    Rafaga Maxima [Km/h]
    Presión [hPa]


30057 -  MagyaCba 60 cuadras. Sensores con datos:
    Temperatura [ºC]
    Humedad [%HR]
    Punto de Rocío [ºC]
    Precipitación [mm]
    Velocidad Viento [Km/h]
    Dirección Viento
    Rafaga Maxima [Km/h]
    Presión [hPa]
    Radiación Solar [W/m2]
    Temperatura Suelo 1 [ºC]


30061 -  MagyaCba La Cumbrecita. Sensores con datos:
    Temperatura [ºC]
    Humedad [%HR]
    Punto de Rocío [ºC]
    Precipitación [mm]
    Velocidad Viento [Km/h]
    Dirección Viento
    Rafaga Maxima [Km/h]
    Presión [hPa]
    Radiación Solar [W/m2]
```

5) Función mensual_precip llamada con un ID inexistente, y un ID correcto, y función diario_precip con un ID correcto.



6. Conclusiones

Como conclusión del proyecto, se aprendió a trabajar con sockets en Linux, a transferir texto y datos entre computadoras, y a realizar una conexión exitosa, teniendo que enfrentarse con los problemas y limitaciones de los protocolos de comunicaciones. El trabajo práctico fue una oportunidad muy buena de medirme como programador, como diseñador y como estudiante, permitiéndome ver mis limitaciones y aprender a superarlas. Tuvieron mucha influencia Google, los manuales de Linux, consultas en StackOverflow y los códigos proporcionados por la cátedra para poder realizar el trabajo. Mucha paciencia y horas de pruebas en distintos escenarios también hicieron falta para comprobar que todo funciona como se debe.

7. Apéndices

A la hora de realizar este trabajo se utilizó contenido de las siguientes URL:

- http://stackoverflow.com/questions/2799612/how-to-skip-the-first-line-when-fscanning-a-txt-file
- http://stackoverflow.com/questions/22805243/convert-integer-to-be-used-in-strcat
- https://support.microsoft.com/en-us/help/38335/how-to-sscanf-example-using-a-comma-,-as-delimiter
- http://stackoverflow.com/questions/623693/accessing-struct-members-directly
- https://www.tutorialspoint.com/c_standard_library/c_macro_offsetof.htm
- http://stackoverflow.com/questions/6074279/how-to-use-fprintf-for-writing-data-to-a-file