



Universidad  
Nacional  
de Córdoba

UNIVERSIDAD NACIONAL DE CÓRDOBA  
Facultad de Ciencias Exactas, Físicas y Naturales

Cátedra de Sistemas Operativos II

**Sockets**  
***Tarazi Pedro Esequiel***

17 de Abril de 2016

## **INDICE**

Introducción.....	3
Descripción del Problema.....	4
Requerimientos y Tareas.....	5
Diseño y Documentación.....	6
Resultados e Implementación.....	8
Conclusión.....	14

## **INTRODUCCIÓN**

Un socket es un método para la comunicación entre un programa cliente y un programa servidor. En una comunicación por socket, el cliente debe conocer la dirección IP del servidor para poder conectarse. Existen varios tipos de sockets, pero los más conocidos y usados son dos: Sockets de Flujo (Stream Sockets), y Sockets de Datagrama (Datagram Sockets). Un socket de flujo define la comunicación en dos direcciones, de manera segura y con conexión. La conexión segura la obtiene mediante un protocolo llamado TCP (Protocolo de Control de Transmisión), que asegura que la información llegue en la forma que se envía y sin errores. En cambio, en la conexión por Datagrama no se asegura que la información llegue a destino, y si llega, puede hacerlo de manera desordenada, pero sin errores. Son sin conexión porque no se necesita mantener una comunicación abierta como si se necesita en TCP.

En el proyecto creado se usan sockets para comunicarse entre un Centro de Operaciones, en este texto llamado COP, y una estación meteorológica, aquí llamada AWS. El COP se comunica con una AWS con el objetivo de recibir información obtenida por una AWS. Las AWS contienen datos tales como fecha, hora, humedad, precipitación y temperatura en el lugar donde la misma está ubicada.

En el presente informe se realizará una breve descripción del problema, se detallarán los requerimientos, además de mostrar el diseño y la documentación del proyecto, para finalmente mostrar los resultados obtenidos.

## **DESCRIPCION DE PROBLEMA**

El objetivo del proyecto es conectar el COP del Servicio Meteorológico Nacional (SMN) situado en la Ciudad Autónoma de Buenos Aires, con una serie de AWS instaladas a lo largo del país. Las AWS toman datos de los sensores cada un segundo, con un formato especificado.

Se pide que la conexión entre el COP y la AWS sea segura, y que se permitan algunos comandos para conectarse entre ellos. Los comandos se detallarán en la sección de requerimientos, donde además se podrá ver que, para el envío de archivos, se pide que la conexión sea de modo no seguro, es decir, mediante UDP.

Si ocurre algún problema en la conexión o en el ingreso de comandos, el programa debe avisar sobre el error. Esto se hace mediante control y manejo de errores.

La implementación se realiza sobre una plataforma de desarrollo creada por la empresa INTEL, llamada INTEL Galileo V1, la cual contiene un sistema operativo que permitirá hacer la simulación.

## REQUERIMIENTOS Y TAREAS

- **CONEXIÓN**

- El software debe poder realizar una conexión entre el COP y, como mínimo, una AWS.
- El programa que corre en el COP debe poder conectarse a una AWS de forma segura, utilizando un puerto fijo.
- El programa que corre en el COP debe tomar un puerto libre de su sistema operativo para conectarse con la AWS.
- La transferencia de archivos se debe realizar con conexión no segura.
- Si fallo la conexión, debe avisar en pantalla.

- **EJECUCION**

- El programa que corre en el COP debe proveer un “prompt”, identificando que está conectado.
- En el contexto de este “prompt” se podrán ejecutar comandos propios de la aplicación. (Ver Comandos)

- **COMANDOS**

- **connect ip port:** debe conectarse a la AWS que posea el ip y el puerto especificados.
- **disconnect:** finaliza la conexión entre el COP y el AWS.
- **get\_telemetry:** obtiene el último registro de telemetría.
- **get\_datta:** descarga el archivo (del registro) con todos los datos de telemetría obtenidos hasta el momento.
- **erase\_datta:** borra la memoria con los datos de telemetría de la memoria interna de la AWS.
- **list:** muestra la lista de comandos aceptados.
- **exit:** finalizada la conexión, permite finalizar la ejecución del COP.

- **TRANSFERENCIA**

- Durante la operación de transferencia de archivo, se debe bloquear la interfaz de usuario del programa, de forma que no se puedan ingresar nuevos comandos hasta que no haya finalizado la transferencia.

- **CONTROL DE ERRORES**

- Debe incluirse manejo de errores por parte del AWS. Si un comando es invalido, debe comunicar esta situación al COP ante la imposibilidad de ejecutarlo.

- **FORMA DE TRABAJO**

- El AWS debe tomar cada 1 segundo uno de los registros de telemetría y guardarlo en un registro interno.

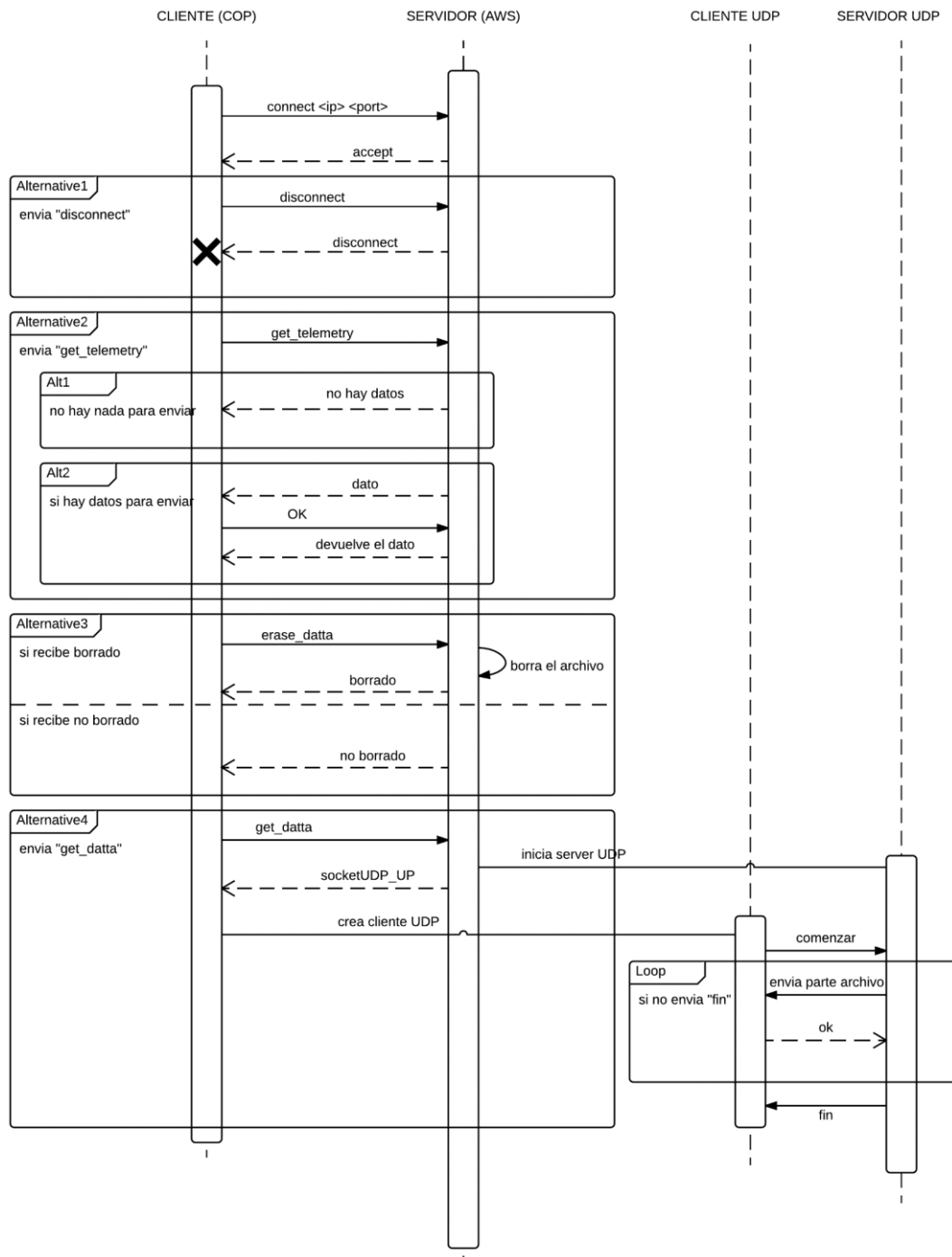
- **OTROS**

- El programa debe contener procesos mono-hilo.

## DISEÑO Y DOCUMENTACION

### Diseño

A continuación, se muestra un diagrama de secuencias UML para indicar el funcionamiento del programa.



## Documentación

Lo primero que debe ejecutarse es el simulador de datos, encargado de leer cada 1 segundo, los datos que fueron entregados mediante un archivo .csv. La ejecución se realiza desde la INTEL Galileo V1, por lo que debemos ingresar al sistema operativo incluido en la misma mediante SSH. Los comandos son:

1. `ssh root@ip`
2. Escribir YES para responder la pregunta de seguridad
3. `Cd /media/card/Pedro`
4. `./simulador`

Luego de ejecutar el simulador, se debe ejecutar el servidor desde la misma placa INTEL Galileo V1. Los comandos son:

1. `make`
2. `./server <puerto>`

El programa que pertenece al COP debe ejecutarse en una PC, utilizando la terminal de Linux. Desde la terminal, nos situamos en la carpeta que contiene el programa y lo ejecutamos mediante los comandos:

1. `make`
2. `./cliente`

Una vez ejecutado, nos aparecerá un “prompt” en el cual debemos ejecutar el comando de conexión entre el COP y la AWS. Esto es, usando “connect <ip> <port>”.

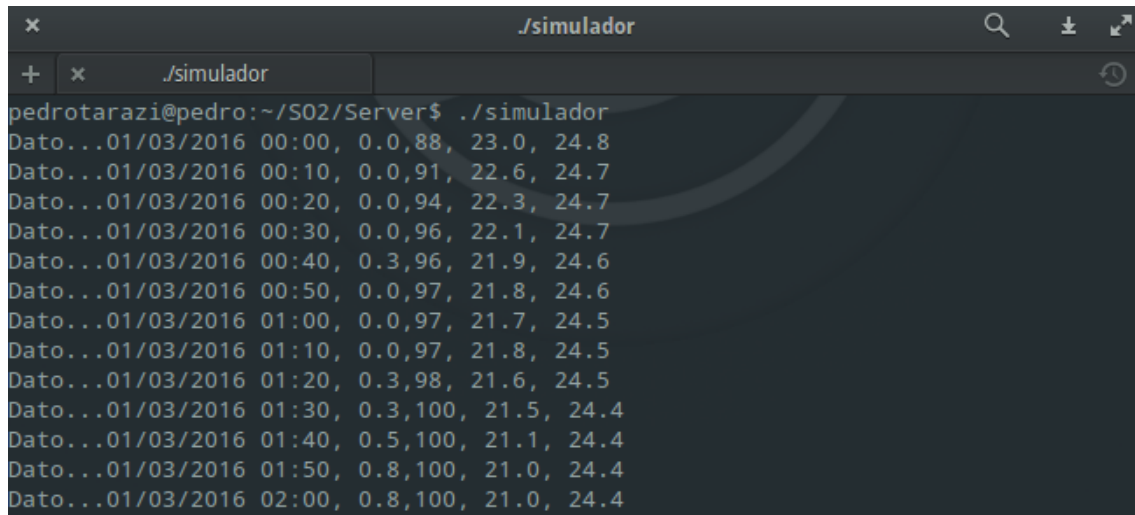
Si la conexión fue exitosa, aparecerá otro “prompt” indicando quien se conectó, y se podrán escribir órdenes y pedidos a la AWS mediante los comandos detallados en la sección Requerimientos.

Luego de finalizada la conexión, se podrá finalizar la ejecución del programa mediante el comando “exit”.

## IMPLEMENTACION Y RESULTADOS

A continuación, se mostrarán los resultados obtenidos en la ejecución del programa.

La Imagen 1.1 muestra el Simulador ejecutándose. El mismo va mostrando en pantalla el dato sensado cada 1 segundo.



```
pedrotarazi@pedro:~/S02/Server$ ./simulador
Dato...01/03/2016 00:00, 0.0,88, 23.0, 24.8
Dato...01/03/2016 00:10, 0.0,91, 22.6, 24.7
Dato...01/03/2016 00:20, 0.0,94, 22.3, 24.7
Dato...01/03/2016 00:30, 0.0,96, 22.1, 24.7
Dato...01/03/2016 00:40, 0.3,96, 21.9, 24.6
Dato...01/03/2016 00:50, 0.0,97, 21.8, 24.6
Dato...01/03/2016 01:00, 0.0,97, 21.7, 24.5
Dato...01/03/2016 01:10, 0.0,97, 21.8, 24.5
Dato...01/03/2016 01:20, 0.3,98, 21.6, 24.5
Dato...01/03/2016 01:30, 0.3,100, 21.5, 24.4
Dato...01/03/2016 01:40, 0.5,100, 21.1, 24.4
Dato...01/03/2016 01:50, 0.8,100, 21.0, 24.4
Dato...01/03/2016 02:00, 0.8,100, 21.0, 24.4
```

*Imagen 1- Simulador en ejecución*

Luego de ejecutado el simulador, como se dijo anteriormente, debemos ejecutar el Server o AWS. En la Imagen 2 se puede ver esto.




```
pedrotarazi@pedro:~/S02/Server$ ./server 6020
>>> Socket disponible: 6020
```

*Imagen 2- Server ejecutándose. Socket disponible a la espera de que un cliente se conecte.*

Como se dijo, el server queda a la espera de la conexión de un cliente. En la Imagen 3 se puede visualizar como se ejecuta el programa Cliente. En la Imagen 4 se muestra el momento de conexión del Cliente y en la Imagen 5 se ve como el Server detecta una conexión.





```
x ./cliente
+ x ./cliente
pedrotarazi@pedro:~/S02/Client$ ./cliente
>>/home/pedrotarazi/S02/Client$
```

*Imagen 3- Cliente ejecutándose. Se muestra el prompt y se espera que se ingrese el comando de conexión.*



```
x ./cliente
+ x ./cliente
pedrotarazi@pedro:~/S02/Client$ ./cliente
>>/home/pedrotarazi/S02/Client$ connect 192.168.10.102 6020
Conectado al Server con IP 192.168.10.102 mediante el puerto 6020
>>AWSQuines-6020$
```

*Imagen 4- El cliente se conectó al Server y da permiso al ingreso de nuevos comandos.*

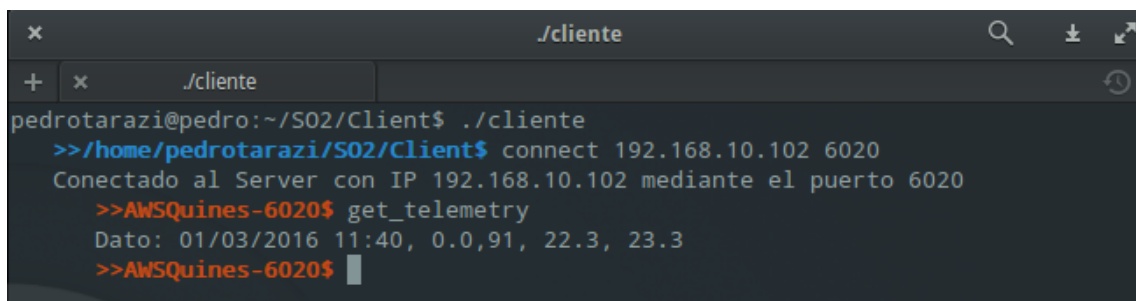


```
x ./server 6020
+ x ./server 6020
pedrotarazi@pedro:~/S02/Server$ ./server 6020
>>$ Socket disponible: 6020
>>$ Cliente 'AWS_Quines' conectado...

```

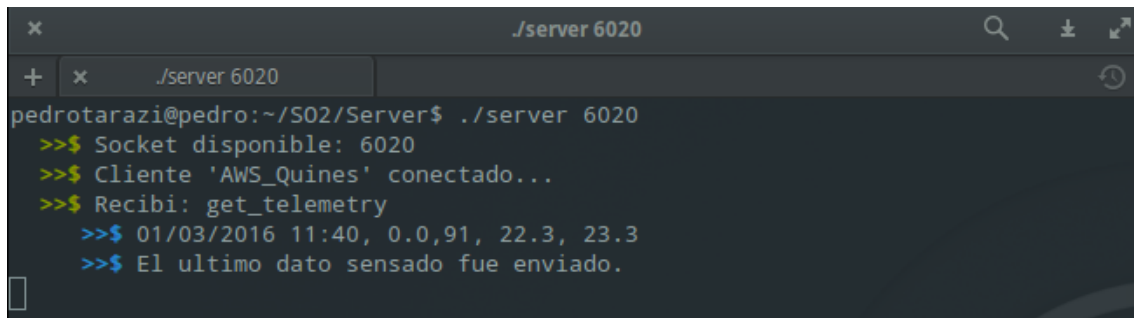
*Imagen 5- El server detecta que un nuevo cliente se conectó y muestra su nombre en pantalla.*

En las siguientes imágenes, se puede que cuando el cliente ingresa un comando aceptado por el server, este último le contesta y envía lo requerido. Cuando se ingresa un comando invalido, el server recibe el comando y avisa al cliente que el comando es no es válido.



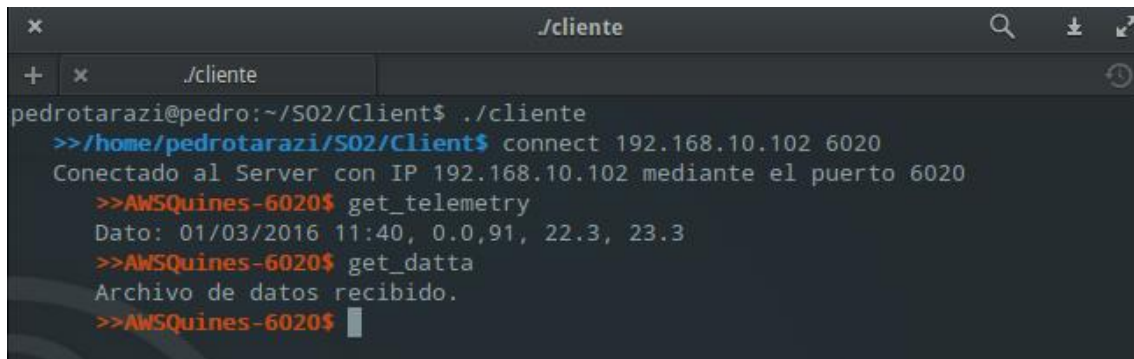
```
x ./cliente
+ x ./cliente
pedrotarazi@pedro:~/S02/Client$ ./cliente
>>/home/pedrotarazi/S02/Client$ connect 192.168.10.102 6020
Conectado al Server con IP 192.168.10.102 mediante el puerto 6020
>>AWSQuines-6020$ get_telemetry
Dato: 01/03/2016 11:40, 0.0,91, 22.3, 23.3
>>AWSQuines-6020$
```

*Imagen 6- El cliente envía "get\_telemetry". Recibe el dato y lo muestra en pantalla.*



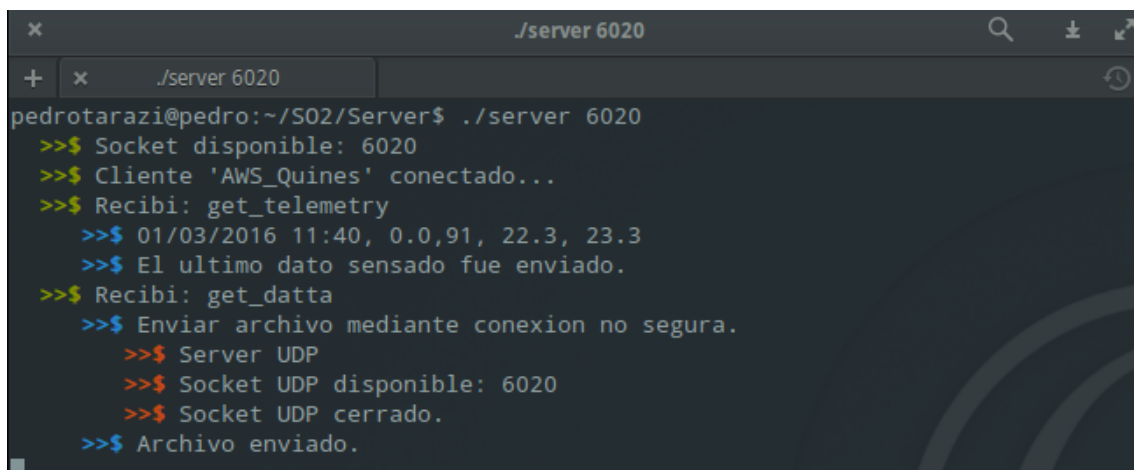
```
pedrotarazi@pedro:~/S02/Server$ ./server 6020
>>$ Socket disponible: 6020
>>$ Cliente 'AWS_Quines' conectado...
>>$ Recibi: get_telemetry
>>$ 01/03/2016 11:40, 0.0,91, 22.3, 23.3
>>$ El ultimo dato sensado fue enviado.
```

*Imagen 7- El server recibe "get\_telemetry", busca el último dato sensado y se lo envía al cliente.*



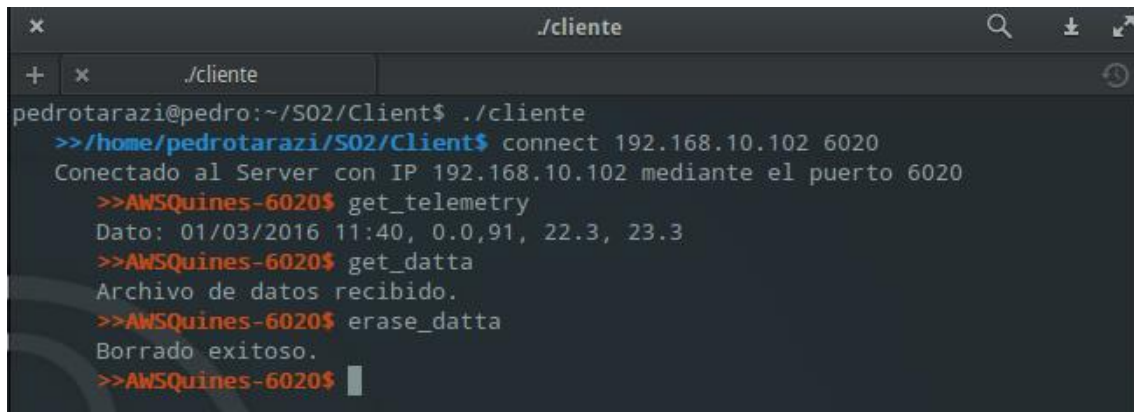
```
pedrotarazi@pedro:~/S02/Client$ ./cliente
>>/home/pedrotarazi/S02/Client$ connect 192.168.10.102 6020
Conectado al Server con IP 192.168.10.102 mediante el puerto 6020
>>AWSQuines-6020$ get_telemetry
Dato: 01/03/2016 11:40, 0.0,91, 22.3, 23.3
>>AWSQuines-6020$ get_datta
Archivo de datos recibido.
>>AWSQuines-6020$
```

*Imagen 8- El cliente ingresa "get\_datta", y recibe un archivo con todos los datos sensados hasta ese momento.*



```
pedrotarazi@pedro:~/S02/Server$ ./server 6020
>>$ Socket disponible: 6020
>>$ Cliente 'AWS_Quines' conectado...
>>$ Recibi: get_telemetry
>>$ 01/03/2016 11:40, 0.0,91, 22.3, 23.3
>>$ El ultimo dato sensado fue enviado.
>>$ Recibi: get_datta
>>$ Enviar archivo mediante conexion no segura.
>>$ Server UDP
>>$ Socket UDP disponible: 6020
>>$ Socket UDP cerrado.
>>$ Archivo enviado.
```

*Imagen 9- Cuando el server recibe "get\_datta", crea una conexión UDP, envía el archivo y cierra la conexión.*



```

x                                     ./cliente
+ x                                     ./cliente
pedrotarazi@pedro:~/S02/Client$ ./cliente
>>/home/pedrotarazi/S02/Client$ connect 192.168.10.102 6020
Conectado al Server con IP 192.168.10.102 mediante el puerto 6020
>>AWSQuines-6020$ get_telemetry
Dato: 01/03/2016 11:40, 0.0,91, 22.3, 23.3
>>AWSQuines-6020$ get_datta
Archivo de datos recibido.
>>AWSQuines-6020$ erase_datta
Borrado exitoso.
>>AWSQuines-6020$

```

*Imagen 10- El cliente ingresa el comando “erase\_datta” y recibe la confirmación de que el archivo alojado en el server ha sido borrado.*

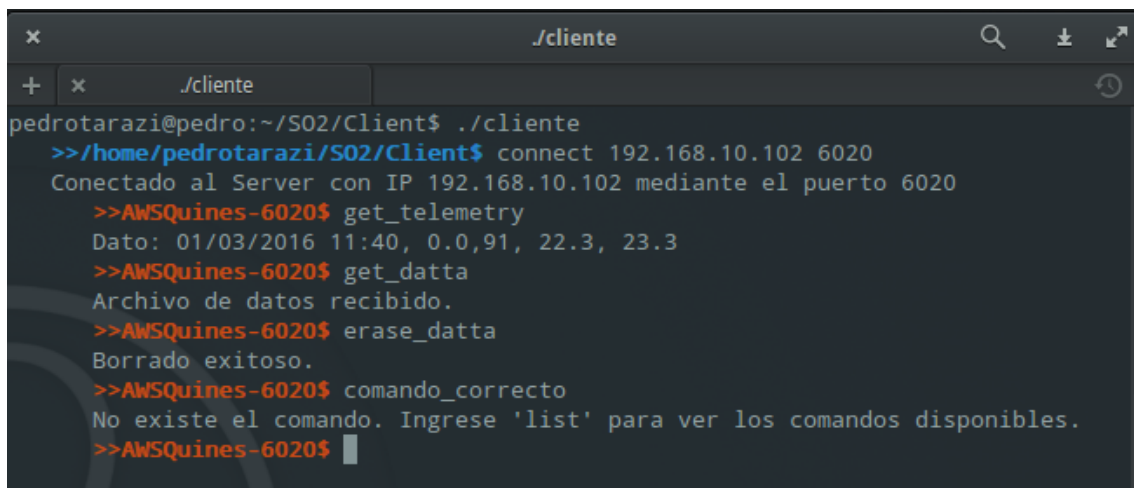


```

x                                     ./server 6020
+ x                                     ./server 6020
pedrotarazi@pedro:~/S02/Server$ ./server 6020
>>$ Socket disponible: 6020
>>$ Cliente 'AWS_Quines' conectado...
>>$ Recibi: get_telemetry
>>$ 01/03/2016 11:40, 0.0,91, 22.3, 23.3
>>$ El ultimo dato sensado fue enviado.
>>$ Recibi: get_datta
>>$ Enviar archivo mediante conexion no segura.
>>$ Server UDP
>>$ Socket UDP disponible: 6020
>>$ Socket UDP cerrado.
>>$ Archivo enviado.
>>$ Recibi: erase_datta
>>$ Debo borrar datos.
>>$ Borrado exitoso.

```

*Imagen 11- El server recibe “erase\_datta”, y borra los datos de la memoria interna.*

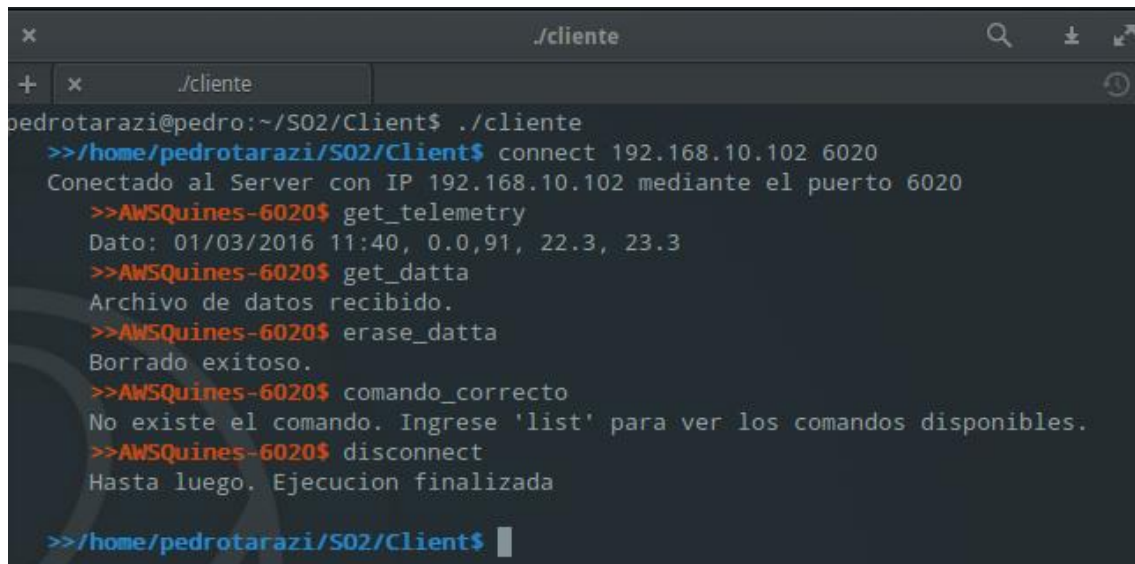


```

x                                     ./cliente
+ x                                     ./cliente
pedrotarazi@pedro:~/S02/Client$ ./cliente
>>/home/pedrotarazi/S02/Client$ connect 192.168.10.102 6020
Conectado al Server con IP 192.168.10.102 mediante el puerto 6020
>>AWSQuines-6020$ get_telemetry
Dato: 01/03/2016 11:40, 0.0,91, 22.3, 23.3
>>AWSQuines-6020$ get_datta
Archivo de datos recibido.
>>AWSQuines-6020$ erase_datta
Borrado exitoso.
>>AWSQuines-6020$ comando_correcto
No existe el comando. Ingrese 'list' para ver los comandos disponibles.
>>AWSQuines-6020$

```

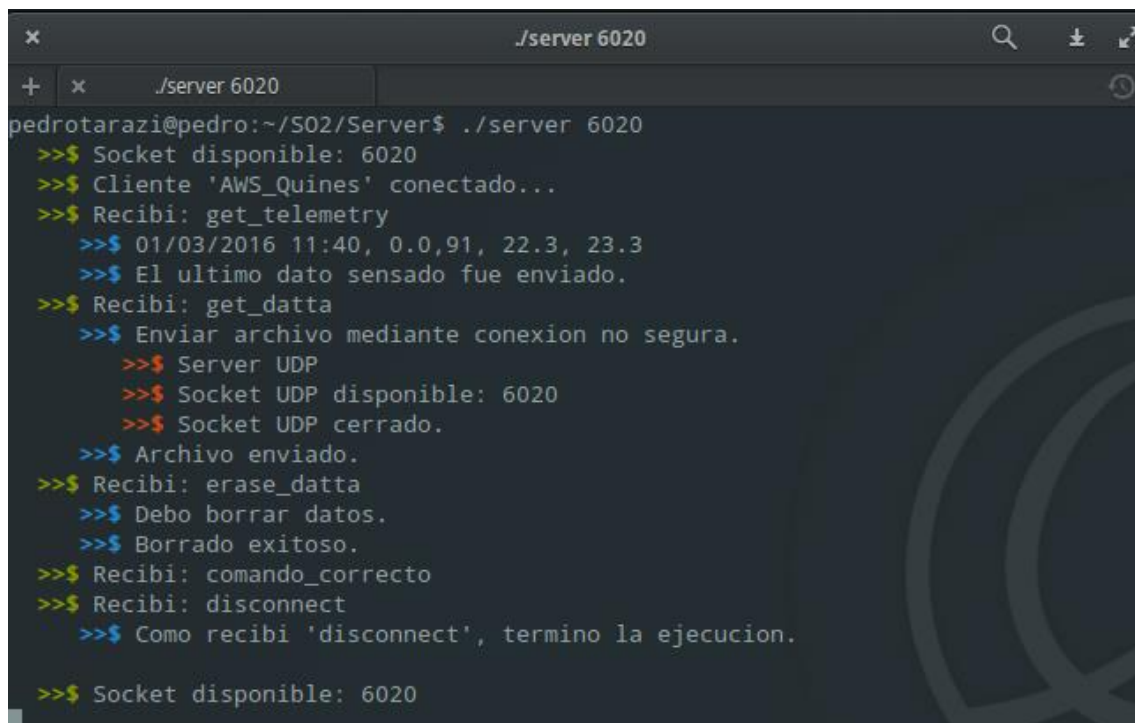
*Imagen 12- El cliente ingresa un comando erróneo, y el server le avisa que puede ingresar el comando “list” para ver los comandos aceptados.*



```
x .cliente
+ x .cliente
pedrotarazi@pedro:~/S02/Client$ ./cliente
>>/home/pedrotarazi/S02/Client$ connect 192.168.10.102 6020
Conectado al Server con IP 192.168.10.102 mediante el puerto 6020
>>AWSQuines-6020$ get_telemetry
Dato: 01/03/2016 11:40, 0.0,91, 22.3, 23.3
>>AWSQuines-6020$ get_datta
Archivo de datos recibido.
>>AWSQuines-6020$ erase_datta
Borrado exitoso.
>>AWSQuines-6020$ comando_correcto
No existe el comando. Ingrese 'list' para ver los comandos disponibles.
>>AWSQuines-6020$ disconnect
Hasta luego. Ejecucion finalizada

>>/home/pedrotarazi/S02/Client$
```

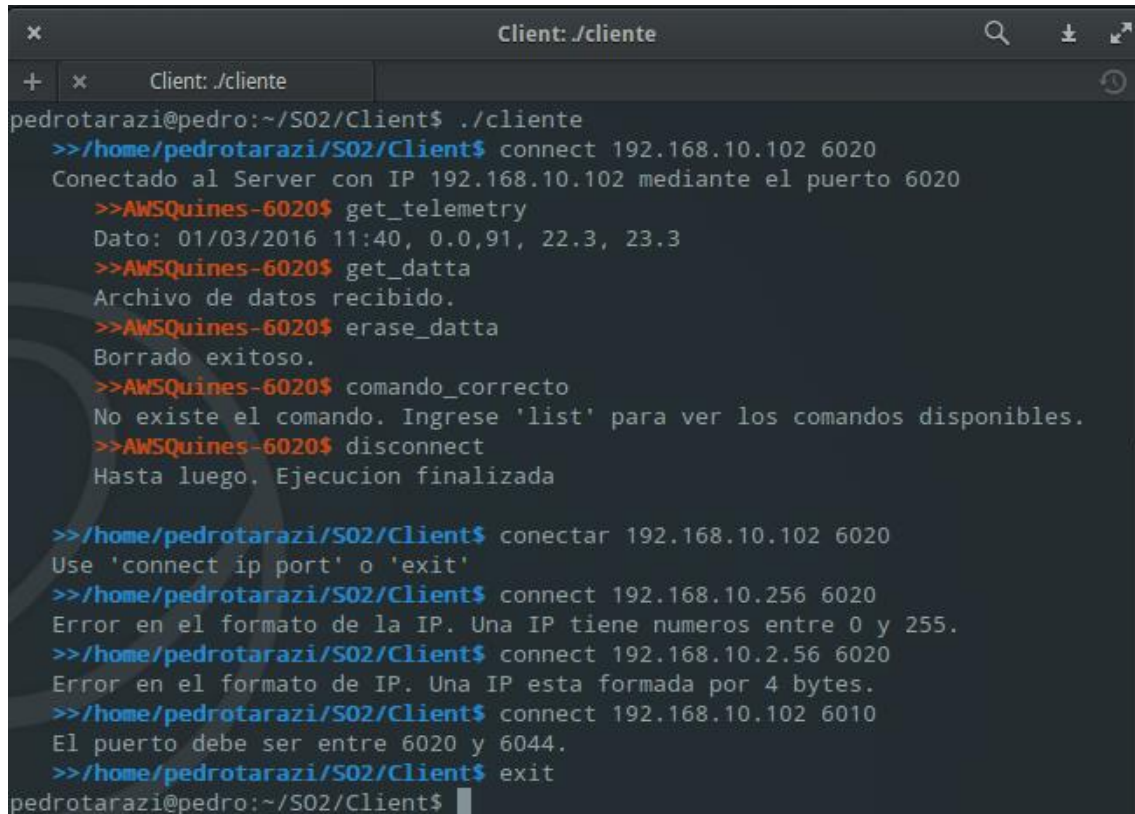
Imagen 13- El cliente ingresa "disconnect" y se desconecta.



```
x .server 6020
+ x .server 6020
pedrotarazi@pedro:~/S02/Server$ ./server 6020
>>$ Socket disponible: 6020
>>$ Cliente 'AWS_Quines' conectado...
>>$ Recibi: get_telemetry
>>$ 01/03/2016 11:40, 0.0,91, 22.3, 23.3
>>$ El ultimo dato sensado fue enviado.
>>$ Recibi: get_datta
>>$ Enviar archivo mediante conexion no segura.
>>$ Server UDP
>>$ Socket UDP disponible: 6020
>>$ Socket UDP cerrado.
>>$ Archivo enviado.
>>$ Recibi: erase_datta
>>$ Debo borrar datos.
>>$ Borrado exitoso.
>>$ Recibi: comando_correcto
>>$ Recibi: disconnect
>>$ Como recibi 'disconnect', termino la ejecucion.

>>$ Socket disponible: 6020
```

Imagen 14- Cuando el server recibe "disconnect", le da el OK al cliente para que termine la ejecución, y deja el socket disponible para que otro cliente se pueda conectar.



```
Client: ./cliente
pedrotarazi@pedro:~/S02/Client$ ./cliente
>>/home/pedrotarazi/S02/Client$ connect 192.168.10.102 6020
Conectado al Server con IP 192.168.10.102 mediante el puerto 6020
>>AWSQuines-6020$ get_telemetry
Dato: 01/03/2016 11:40, 0.0,91, 22.3, 23.3
>>AWSQuines-6020$ get_datta
Archivo de datos recibido.
>>AWSQuines-6020$ erase_datta
Borrado exitoso.
>>AWSQuines-6020$ comando_correcto
No existe el comando. Ingrese 'list' para ver los comandos disponibles.
>>AWSQuines-6020$ disconnect
Hasta luego. Ejecucion finalizada

>>/home/pedrotarazi/S02/Client$ conectar 192.168.10.102 6020
Use 'connect ip port' o 'exit'
>>/home/pedrotarazi/S02/Client$ connect 192.168.10.256 6020
Error en el formato de la IP. Una IP tiene numeros entre 0 y 255.
>>/home/pedrotarazi/S02/Client$ connect 192.168.10.2.56 6020
Error en el formato de IP. Una IP esta formada por 4 bytes.
>>/home/pedrotarazi/S02/Client$ connect 192.168.10.102 6010
El puerto debe ser entre 6020 y 6044.
>>/home/pedrotarazi/S02/Client$ exit
pedrotarazi@pedro:~/S02/Client$
```

*Imagen 15-* Cuando el cliente se desconectó, aparece el prompt principal esperando una nueva conexión o la finalización del programa. Se muestra como es el control de errores, ya que no se aceptan IPs con formatos inválidos o puertos fuera del rango especificado en requerimientos. El ingreso del comando “exit” finaliza la ejecución del programa.

## **CONCLUSION**

En general, podemos concluir que es factible construir nosotros mismos los programas que nos permitan comunicarnos con otros programas y enviar información mutuamente. La forma de realizarlo es mediante Sockets.

Se comprendió el funcionamiento básico de esta herramienta, la cual es de gran utilidad a la hora de conectar dos procesos que pueden estar en una misma maquina o en distintas computadoras. Los sockets solo pueden conectarse con sockets de su mismo tipo y familia, por lo que hay que tener muy claro esto a la hora de codificar. Esta herramienta permite realizar varias conexiones simultaneas, aunque en el presente trabajo se realizó la conexión entre un cliente y un servidor. Los mensajes y señales que se envían entre ambos son de gran importancia para poder tener una comunicación fluida y sin problemas.

Para finalizar, podemos aclarar que al principio costó entender cómo funcionaba la comunicación, pero con el correr de las semanas, y con mucha lectura, se logró una programación más amena y satisfactoria.