Universidad
Nacional
de Córdoba

Cátedra de Sistemas Operativos II

Trabajo Práctico

Nº I

Kleiner Matias

11 de abril del 2019

Introducción

Propósito

En el presente documento se mostrará el proceso de diseño y codificación de un programa que implementa un sistema de comunicación entre procesos con sockets. Se dispone de un programa que atiende solicitudes de conexiones de los satélites (de ahora en mas, estacion). Además, otro programa simula el comportamiento de un satélite brindando funciones de telemetría, scanning y actualización del firmware.

Ámbito del Sistema

La aplicación cliente se ejecuta en un sistema embebido (Raspberry PI) la cual a su vez tiene como sistema operativo Linux Ubuntu Mate.

La aplicación servidor en cambio, se ejecuta en una computadora de propósito general con sistema operativo Linux Manjaro.

Referencias

[1]. Michael Kerrisk, The Linux Programming Interface, 2010, Adison-Wesley

[2]. Imagen del satelite Goes16,

”https://cdn.star.nesdis.noaa.gov/GOES16/ABI/FD/13/20190811400 GOES16-

ABI-FD-13-10848x10848.jpg.zip”, Online; accessed 20 Marzo 2019

[3]. Making The Best Use of C, https://www.gnu.org/prep/standards/html node/Writing-

C.html, Online; accessed 20 Marzo 2019

[4]. L.Trovalds, Et al., https://github.com/torvalds/linux/tree/master/Documentation/process,

Online; accessed 20 Marzo 2019,

Descripción General

Perspectiva del Producto

Este trabajo práctico se realizó para implementar un sistema cliente-servidor por medio de sockets UNIX e Internet. La función principal del sistema es ofrecer, mediante el servidor, una consola de comandos que implementa programas básicos que ejecutara un satélite.

Funciones del Producto

Las funciones que ofrece el producto son:

- Conexión de un cliente a un servidor mediante sockets.
- Ingresar comandos desde el servidor para que los clientes conectados atiendan dichas consultas.
- Ejecución de los comandos ingresados por el servidor, desde el satélite.

Características de los Usuarios

El usuario solamente necesita estar mínimamente familiarizado con la terminal de Linux, pudiendo observar los resultados directamente en su terminal.

Restricciones

No se aceptan conexiones simultáneas de satélites. No se garantiza la recepción de la ejecución “obtener telemetría” ya que los datos viajan por UDP y no se hace un control sobre el mismo.

Suposiciones y Dependencias

- Se supone que la PC en la que correrá el cliente y la placa de desarrollo en la que correrá el servidor, ambas tienen instalada una distribución estándar de Linux.
- Se debe disponer de un cable ethernet para la comunicación entre el embebido y la computadora de propósito general.

Requisitos Específicos

Interfaces Externas

Para ejecutar el servidor en la placa de desarrollo, es necesario (además de contar con una) disponer de una forma de interacción con la misma, ya sea conectando un teclado USB y un monitor HDMI, o accediendo mediante SSH desde una computadora (ambas, la computadora y la placa, deben estar conectadas en la misma red). El cliente se puede ejecutar en cualquier PC con alguna distribución de Linux. Además, tanto el cliente como el servidor deben estar en la misma red para interactuar.

Funciones

Servidor:

- A través de un puerto especificado (6028), permitir autenticar a un usuario usando usuario y password.
- Permitir el ingreso de funciones (**update firmware.bin**, **start scanning**, **obtener telemetría**).

Cliente:

- **update firmware.bin** envía un archivo binario al satélite con una actualización del software (”firmware”) cliente, subido el archivo, se debe reiniciar el satélite con la nueva actualización.
- **start scanning** inicia un escaneo de toda la cara de la Tierra una sección del planeta. Cada escaneo se debe enviar a la estación terrena, que va a estar midiendo el tiempo que le lleva el escaneo de todo el disco (desde que envía el comando, hasta que recibe el último datagrama). Este proceso debe ser lo más óptimo posible. El tamaño de cada escaneo está determinado por el tamaño máximo de un datagrama.

- **obtener telemetría** cada satélite le envía a la estación terrena la siguiente información:

    Id del satélite

    Uptime del satélite

    Versión del software

    Consumo de memoria y CPU

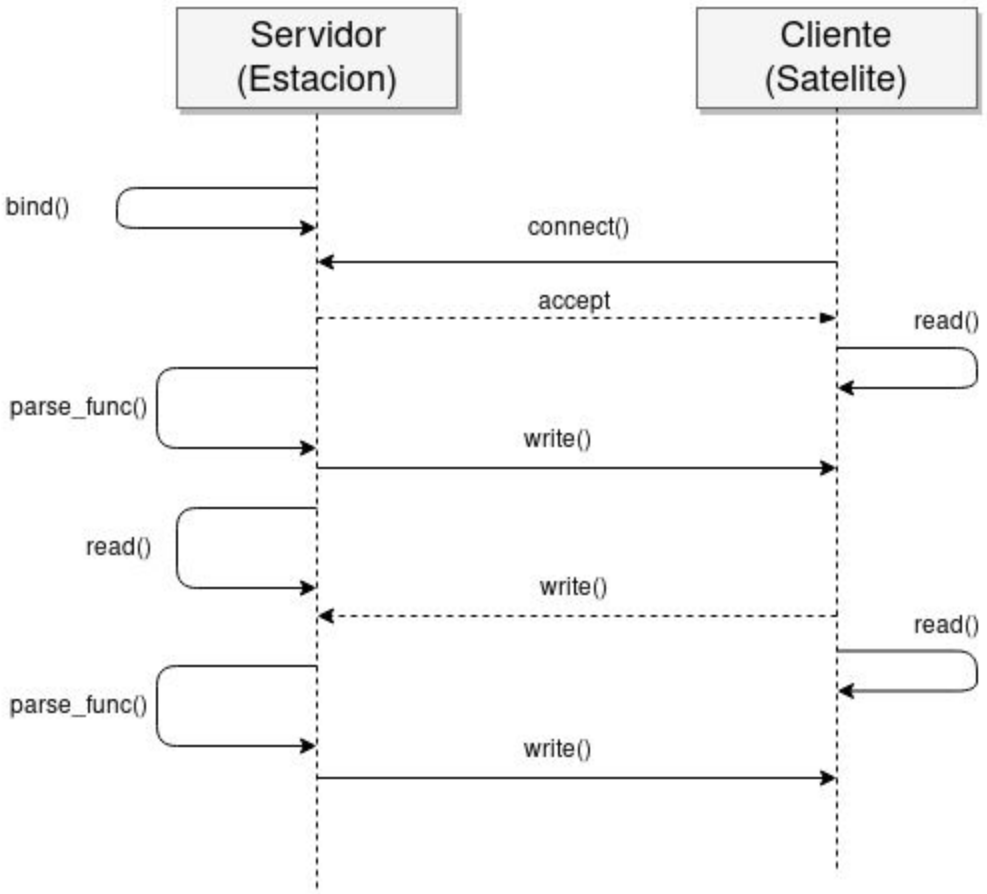
Restricciones de Diseño

- Ambos programas deben ser escritos en C siguiendo un estilo de programación.
- Se debe utilizar la API de sockets.
- Se debe utilizar sockets de internet de forma segura (orientado a conexión).
- El puerto del servidor es fijo.
- El cliente debe tomar un número de puerto libre de su sistema operativo.
- Debe implementarse un sistema de autenticación (usuario y password).
- La autenticación debe realizarse del lado del servidor.
- La transferencia del archivo debe realizarse utilizando conexión segura (orientado a la conexión)
- Durante la transferencia se debe bloquear la interfaz de usuario del programa.
- Todos los procesos deben ser mono-thread.
- No debe usarse ningún sistema de base de datos.
- La implementación del cliente debe ser en una placa de desarrollo.
- La implementación del servidor debe ser una computadora cualquiera.
- La placa de desarrollo debe soportar MMU y poseer una arquitectura compatible con GNU/Linux.
- Se debe utilizar Cppcheck y compilar con las flags -Werror -Wall y -pedantic.

Diseño de solución

Para diseñar el software, se comenzó con un programa básico tanto para el cliente como para el servidor, en donde solamente se conectan mediante un socket UNIX y, una vez establecida la conexión, el cliente imprime los mensajes que el servidor envía. Además, ambos programas fueron ejecutados en la misma PC en dos terminales distintas, para reducir las posibles complicaciones de usar la placa de desarrollo.

El diagrama de secuencia ejemplifica el diseño básico del sistema:



Luego, se realizó el proceso de autenticación utilizando, en el servidor un vector multidimensional que tiene guardados los datos de todos los usuarios que están registrados en el sistema.

Si los datos son correctos, se informa al cliente que se conectó, de lo contrario, se debe reingresar la contraseña hasta 3 intentos. Luego de los tres intento, debe volver a ingresar el usuario.

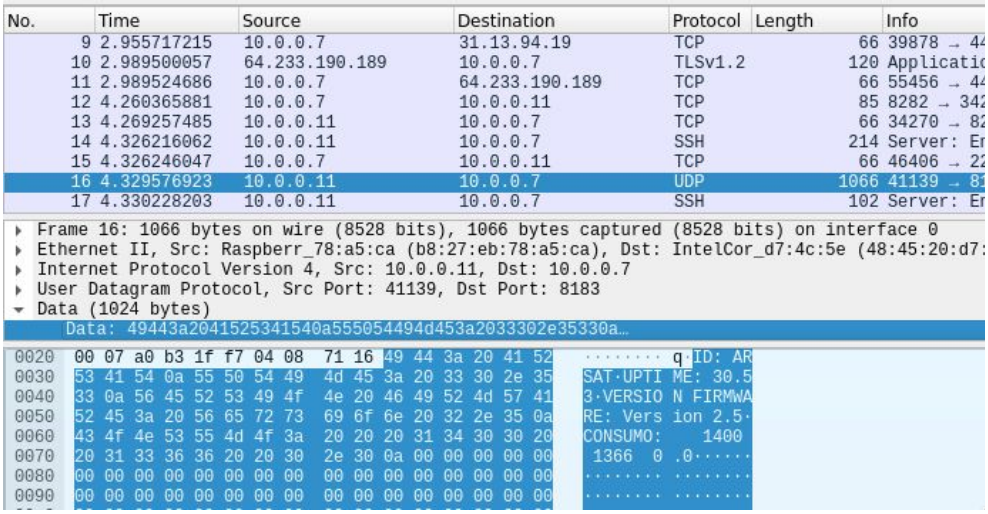
El cliente puede ingresar 3 tipos de comandos, por lo que se debe hacer algo diferente dependiendo de la cadena enviada:

- **update firmware.bin**
- **start scanning**
- **obtener telemetría**

Durante todo el proceso, tanto el cliente como el servidor deberán ejecutar varias veces los comandos read y write. Es necesario asegurar que ninguno de los dos programas se bloqueará, por lo que se busca que a cada write le corresponda un read del otro lado. También, es preciso evitar dos write o read seguidos, porque esto puede llevar a bloqueos.

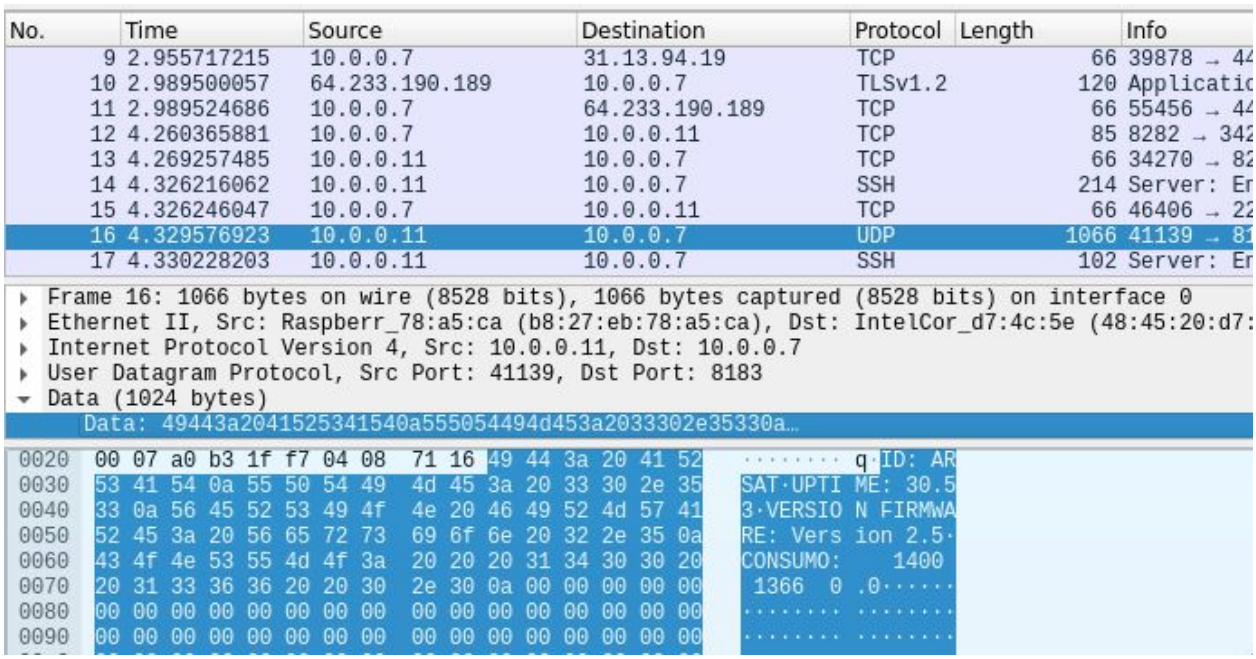
Implementación y Resultados

A continuación, se muestran los resultados de la implementación de las distintas funcionalidades del sistema.



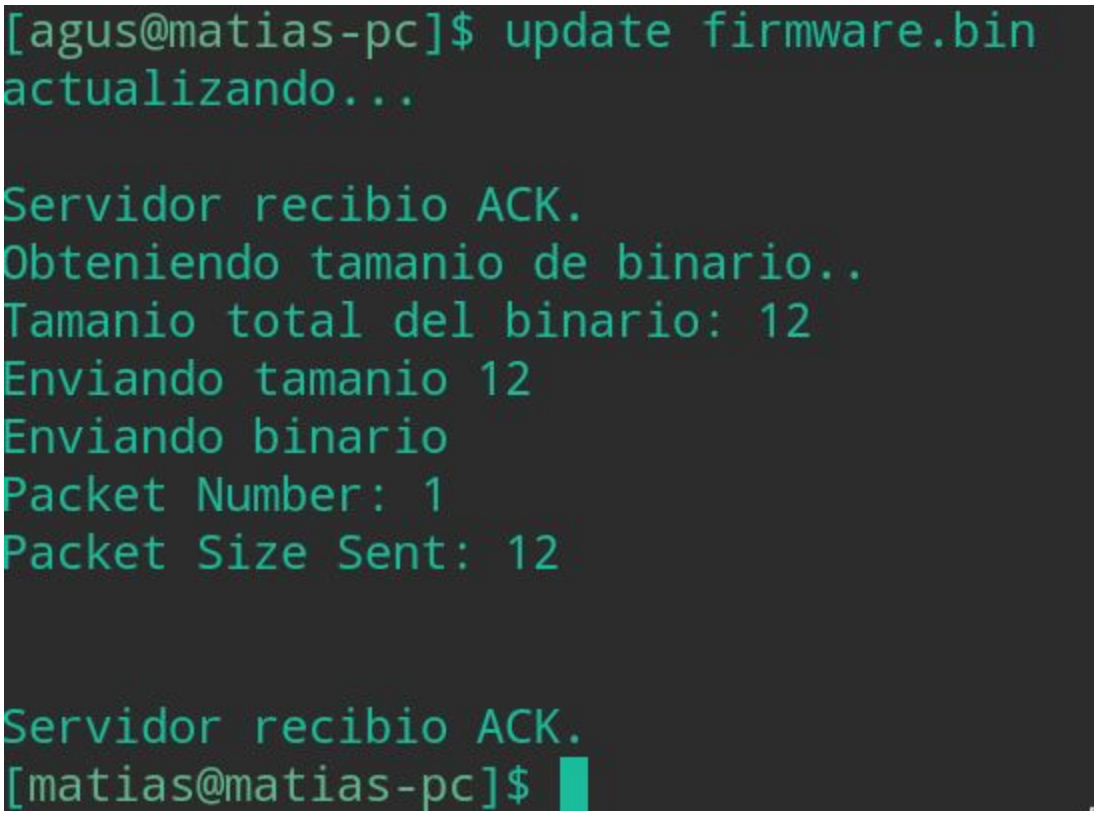
Captura de “obtener telemetría”

Como se puede observar, obtener telemetría se envía a través de UDP tal como requiere la consigna.



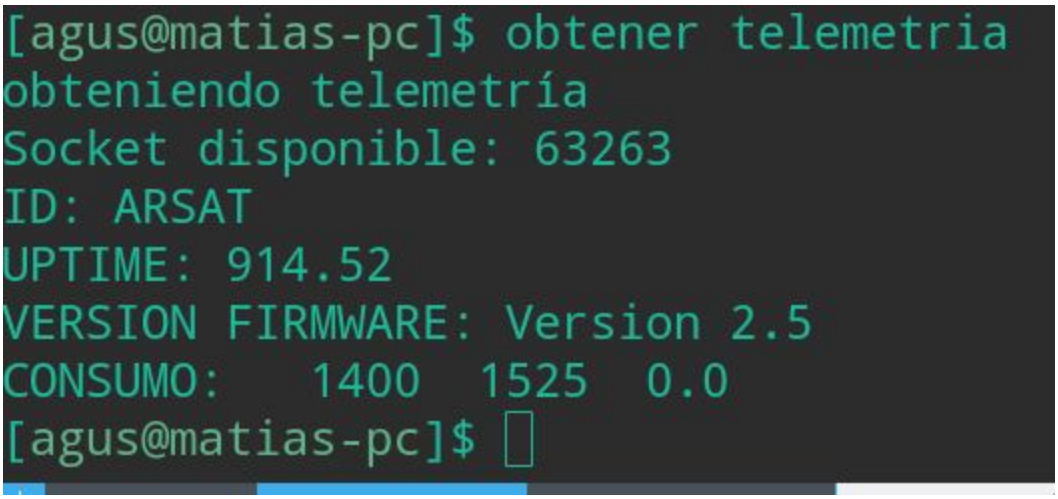
Captura de “start scanning”

Como se puede observar, start scanning se envía a través de mensajes TCP. A su vez, la fragmentación la hace la propia aplicación y no el protocolo TCP.



Captura de “update firmware”

Luego de ejecutar update firmware, el cliente se reinicia (por eso se desconecta el servidor y hay un cambio en el promp). Al realizar la consulta de “obtener telemetría” nuevamente, se puede ver que la versión del firmware fue actualizada en la aplicación cliente.



Captura de “obtener telemetría”

La información que recibe el servidor del cliente luego de ejecutar “obtener telemetria” corresponde a ID, UPTIME, VERSION FIRMWARE y CONSUMO (MEM y CPU), tal como se pide en la consigna.

Conclusiones

Al realizar este trabajo práctico se pudo aprender sobre la abstracción de los sockets y como son utilizados para la comunicación entre procesos, llevando a cabo en la práctico lo que se había visto solo de manera conceptual. También se obtuvieron conocimientos de cómo operan distintas llamadas al sistema del kernel de Linux (read, write, etc.) y cómo se utilizan buffers (line buffered, no buffered y block buffered) para escribir datos en los distintos streams (stdin, stdout, stderr)

Se tuvo el problema de que el código compilaba con todas las flags en la notebook y, una vez llevado el programa a la Raspberry PI que corría un sistema operativo Ubuntu Mate, el mismo presentaba errores de compilación al hacer el make (debido a la diferencia en el estándar C90 y C99).