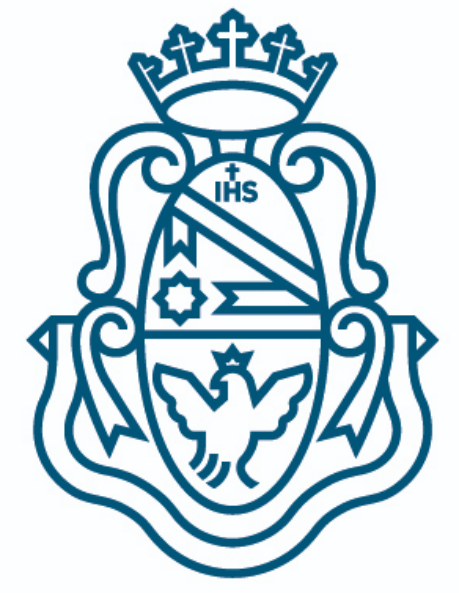
Universidad Nacional de Córdoba
Facultad de Ciencias Exactas, Físicas y Naturales



TRABAJO PRÁCTICO N°1

Sistemas Operativos II

Sockets de internet en sistemas tipo UNIX

Gerardo Collante

Profesor
Ing. Agustín Martina

2 de marzo de 2020

Índice

1. Introducción

1.1. Definición de objetivos

El objetivo de este trabajo práctico consiste en que el programador cree un sistema cliente-servidor donde el cliente se simulará que es un satélite y el servidor una estación terrena. Ambos intercambian archivos e información a través de sockets TCP y UDP.

Para llevar a cabo esta tarea fue entregada una lista de requerimientos con la cual el programador debe pensar como cumplirlos, esto se conoce como *análisis funcional* y es una parte vital del trabajo, ya que usualmente el cliente (persona física que solicita el sistema) no posee los conocimientos técnicos para realizar la propuesta.

1.2. Definición de objetivo

Consiste en crear un sistema cliente-servidor donde el cliente se simulará que es un satélite y el servidor una estación terrena. Ambos intercambian archivos e información a través de sockets TCP y UDP.

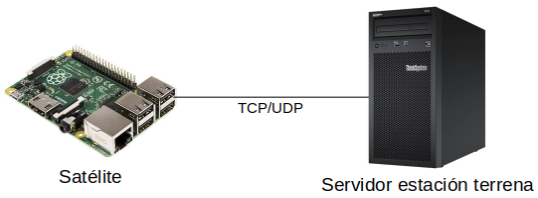


Figura 1: Diagrama simple del modelo del proyecto.

2. Desarrollo

2.1. Cliente

Se utilizará como hardware una *RaspberryPi 3B+* con sistema operativo *Raspbian Stretch.*

2.2. Servidor

El hardware será una PC con sistema operativo *Linux Ubuntu 18.04.4 LTS.*

2.3. Conexión

Ambas dispositivos estarán conectados a través de cable de red *Fast Ethernet* para simular una red de internet.

2.4. Funcionamiento general del algoritmo

El cliente siempre está corriendo esperando conexiones de servidores TCP o UDP. Una vez que llega una conexión posee un `handler` que se encarga de la misma según su tipo.

En cambio el servidor sólo inicia conexiones una vez que el usuario ingresa los comandos correspondientes.

2.4.1. `login()`

Se pensó en utilizar *hashes* para almacenamiento del `userpass`, debido a que es la forma que usualmente se guardan las contraseñas.

```
unsigned int string_hash(void *string){
    /* This is the djb2 string hash function */

    unsigned int result = 5381;
    unsigned char *p;

    p = (unsigned char *) string;

    while (*p != '\0') {
            result = (result << 5) + result + *p;
            ++p;
    }

    return result;
}
```

De esta forma se guardo en variables `long unsigned int` la contraseña del `login()`, además se validaron los requerimientos solicitados como cantidad de intentos.

Además se aprovechó esta funcionalidad para sustituir a `strcmp()` ya que comparando las variables en un `switch` fácilmente se podían distinguir los comandos ingresados por el usuario.

2.4.2. `update firmware.bin`

Se establece una conexión TCP entre cliente y servidor para la transferencia del archivo. Una vez completada, el cliente se reinicia.

En el cliente fue creado un servicio que corre un *script* que al iniciar el sistema se encarga de verificar si existe un nuevo *bin*, en caso que exista se borra el anterior, se cambia el nombre, se actualiza la versión del *firmware* en un archivo de texto, se le dan permisos de ejecución (`chmod()`) y se corre nuevamente.

2.4.3. `start scanning`

Se establece una conexión TCP entre el cliente y servidor donde se realiza la transferencia de la imagen, luego se cierra el socket.

2.4.4. `obtener telemetria`

El servidor crea un socket UDP para la conexión. Una vez que el cliente es contactado, envía a través de mensajes UDP los datos requeridos. El servidor se encarga de mostrarlos en consola para el usuario.

2.5. `Makefile`

Se utilizó una cantidad de tiempo considerable en la creación de un archivo `Makefile` bien configurado para las diferentes pruebas a las que se sometió el sistema puedan ser ejecutadas con rapidez.

2.5.1. Organización de carpetas

Se optó por que cada carpeta del proyecto tuviera un tipo de archivo para mejorar la organización. Entonces en cada carpeta se guardó:

- `bin`: archivos binarios.
- `src`: código fuente y archivos *.o*.
- `include`: *headers*.

2.5.2. Compilación cruzada

Al principio todo el desarrollo se hizo internamente en una PC, una vez finalizado esto se tenía que migrar al dispositivo cliente.

Sin embargo, para cumplir con los requerimientos de actualización de *firmware* era necesario realizar compilación cruzada, que se define como el acto de compilar código para un sistema informático (a menudo conocido como el objetivo) en un sistema diferente, llamado host.

Fue necesario descargar el *toolkit ARM RaspberryPi* y realizar varias pruebas hasta que se logró su correcto funcionamiento. Esto puede ser revisado en el `Makefile`.

3. Conclusión

Si bien las consignas estaban claras y el desarrollo estaba estructurado algunos pasos tomaron un tiempo considerable, sumado a los problemas que surgieron del ensamblaje final de cada uno de ellos.

Pero fueron adquiridos conocimientos valiosos como una correcta estructura del proyecto para que el código fuente sea mantenible, además de la documentación del mismo. También se usó *Git* de manera local para los repositorios para la implementación de nuevas *features* a través de ramas.

En síntesis a pesar de ser extenso el proyecto se obtuvieron habilidades en diversas herramientas que serán de utilidad en el futuro maximizando recursos y tiempo.