



Universidad
Nacional
de Córdoba

Cátedra de Sistemas Operativos II

Trabajo Práctico N° I

Sepulveda Federico Nicolas
22 de Abril del 2018

Índice

| | |
|--|-----------|
| Introducción | 3 |
| Propósito | 3 |
| Ámbito del Sistema | 3 |
| Definiciones, Acrónimos y Abreviaturas | 3 |
| Referencias | 3 |
| Descripción General del Documento | 4 |
| Descripción General | 4 |
| Perspectiva del Producto | 4 |
| Funciones del Producto | 4 |
| Características de los Usuarios | 4 |
| Restricciones | 5 |
| Suposiciones y Dependencias | 5 |
| Requisitos Futuros | 5 |
| Requisitos Específicos | 5 |
| Interfaces Externas | 5 |
| Funciones | 5 |
| Requisitos de Rendimiento | 5 |
| Restricciones de Diseño | 6 |
| Atributos del Sistema | 6 |
| Otros Requisitos | 6 |
| Diseño de solución | 7 |
| Implementación y Resultados | 9 |
| Conclusiones | 11 |
| Apéndices | 11 |
| Apéndice A | 11 |
| Apéndice B | 11 |

Introducción

En este trabajo se pretende diseñar, implementar y testear un software, que permita realizar la conexión, control y transferencia de datos entre uno o más cliente con un con un servidor que posee el baash desarrollado en Sistemas Operativos I

Propósito

El propósito de este trabajo práctico es instruirse en el desarrollo y la implementación de lo aprendido en las clases de la asignatura Sistemas Operativos II, sobre Sockets TCP y UDP.

Ámbito del Sistema

A fines de llevar a cabo su implementación, se provee de una plataforma de desarrollo, Raspberry Pi, sobre la cual desarrollar el prototipo de ingeniería.

Definiciones, Acrónimos y Abreviaturas

A lo largo de este informe se hará referencia a las siguientes abreviaciones:

TCP. Abreviación por sus siglas en inglés de Transmission Control Protocol.

UDP. Abreviación por sus siglas en inglés de User Datagram Protocol.

SCP. Abreviación por sus siglas en inglés de Secure Copy Protocol.

SSH. Abreviación por sus siglas en inglés de Secure Shell.

SO. Abreviación de Sistema Operativo.

Referencias

[1] GNU, “5 Making The Best Use of C”, https://www.gnu.org/prep/standards/html_node/Writing-C.html, [Marzo 2018]

[2] Trovalds, L., Et al., <https://github.com/torvalds/linux/tree/master/Documentation/process>, [Marzo, 2018]

[3] W. Richard Stevens, Stephen A. Rago, Advanced Programming in the UNIX Environment, 3rd Edition, Adison-Wesley, [Marzo, 2018]

[4] Doxygen, <https://www.doxygen.org/> [Abril, 2018]

[5] Makefile documentation, Germán Póo-Caamaño, <http://calcifer.org/documentos/make> [Abril, 2018]

Descripción General del Documento

A continuación se describe el producto a desarrollar, seguido de su correspondiente diseño e implementación.

Descripción General

Perspectiva del Producto

Con el motivo de implementar sockets TCP y UDP, en la cátedra de Sistemas Operativos II se pidió una aplicación que utilice la arquitectura cliente servidor orientada a conexión. De esta manera un cliente puede conectarse a un servidor en el cual está corriendo la aplicación `bash`. Esta aplicación ejecuta comandos como la terminal que trae por defecto los SO basados en linux. El cliente recibe todo lo que el servidor imprime en pantalla como una conexión SSH. Además el cliente puede solicitar al servidor la descarga de archivos, esta transferencia hace uso no orientado a conexión.

Funciones del Producto

El programa que corre en el Cliente permite conectarse al Servidor, de forma segura (orientado a conexión), utilizando un puerto fijo (6020) y tomar un número de un puerto libre (aleatorio) de su sistema operativo.

Además, brinda un sistema de autenticación (usuario y password) de acceso al servidor.

Características de los Usuarios

El cliente se conecta al Servidor mediante la introducción del siguiente comando:

- `connect usuario@numero_ip:port:`

Luego el programa le solicitará el password de dicho usuario para loguearse en el servidor como tal.

Al llegar esta información al servidor, de no existir el usuario o de haber ingresado un password incorrecto, se notifica al cliente, con leyenda "nombre de usuario y/o contraseña incorrecto". En caso de haber sido positivo, el servidor le envía un mensaje de autorización.

Una vez autorizado, el cliente muestra un menú de opciones a ejecutar.



Restricciones

El Servidor solo acepta los usuarios que tiene registrados. Del lado del Cliente, se restringe su uso brindando el comando que se debe ingresar para realizar la conexión.

Suposiciones y Dependencias

Para el desarrollo de este trabajo se tomó las siguientes suposiciones:

- Existe dos usuarios uno llamado 'usuario' con password: 'pass' y el otro 'sepulveda' con password: '12345'.
- Este trabajo correrá en un Sistema Operativo Linux.
- El cliente y el servidor se encuentran en la misma red.

Requisitos Futuros

- El cliente y servidor pueden estar en distintas redes
- Soportar el cambio de sesión.

Requisitos Específicos

Interfaces Externas

Este trabajo ofrece para la comunicación con el usuario un prompt (proporcionado por el servidor) mediante el cual el cliente puede realizar las peticiones al servidor.

Funciones

La aplicación proporciona un sistema de logueo para usuarios. Una vez autorizado, el cliente muestra un mensaje de conexión exitosa. Luego el cliente puede ingresar comandos del baash.


El comando:

- descarga nombre_archivo

Es el único comando implementado fuera del baash, descarga el archivo solicitado.

Requisitos de Rendimiento

La transferencia (descarga) de archivos se realiza con utilizando conexión no segura (sin conexión).



Durante la operación de transferencia del archivo, se bloquea 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.

Además, la aplicación provee un mecanismo de control y manejo de errores por parte del servidor con comunicación al cliente.

Restricciones de Diseño

Todos los procesos deben ser mono-thread.

Atributos del Sistema

El sistema debe tener una arquitectura compatible con GNU/Linux

Otros Requisitos

La aplicación debe ser desarrollada en lenguaje C.

Diseño de solución

Se realizó un diagrama de caso de usos, el cual se muestra en la Figura 1. El mismo contempla los casos de uso que están al alcance del usuario, y como la aplicación interactúa con el baash.

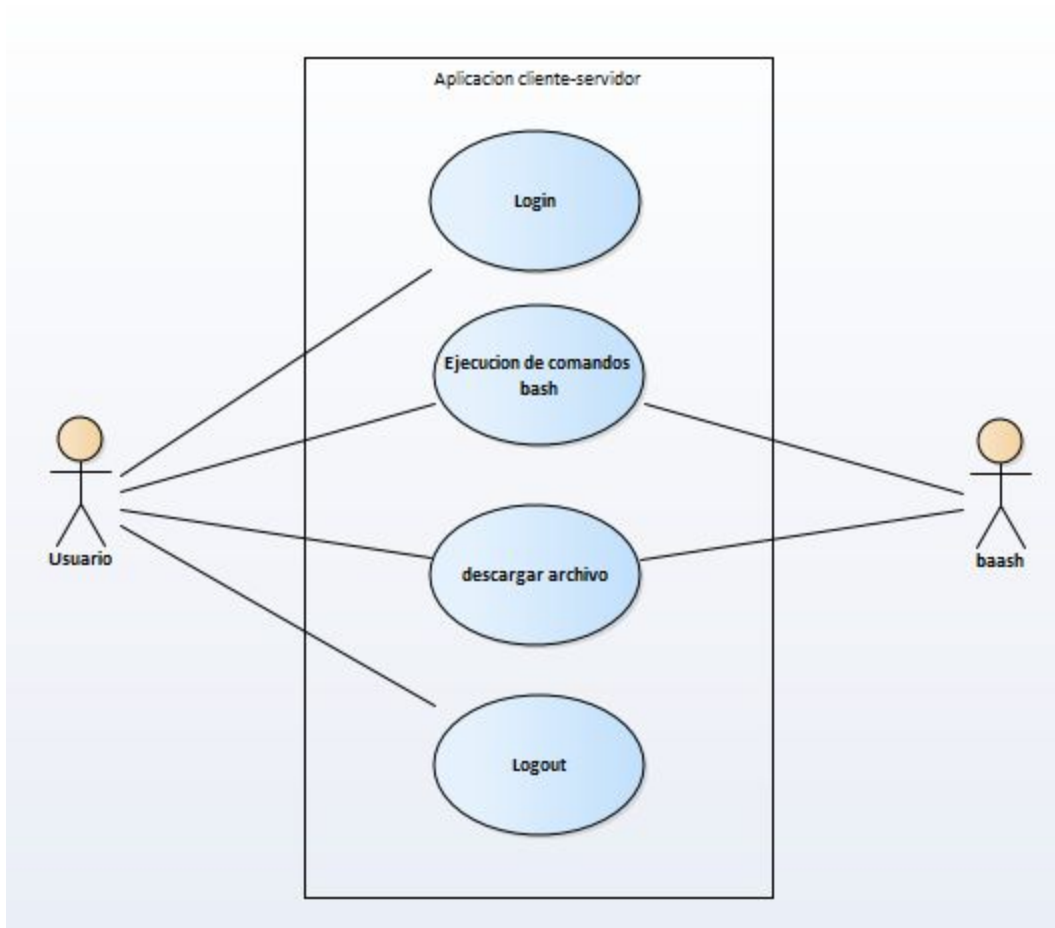


Figura 1. Diagrama de Caso de Usos.

En la Figura 2 se presenta un diagrama de secuencia, el cual intenta generalizar el proceso de envíos de comandos por parte del usuario, y se enfoca más al proceso de autenticación de usuario y validación de comandos.

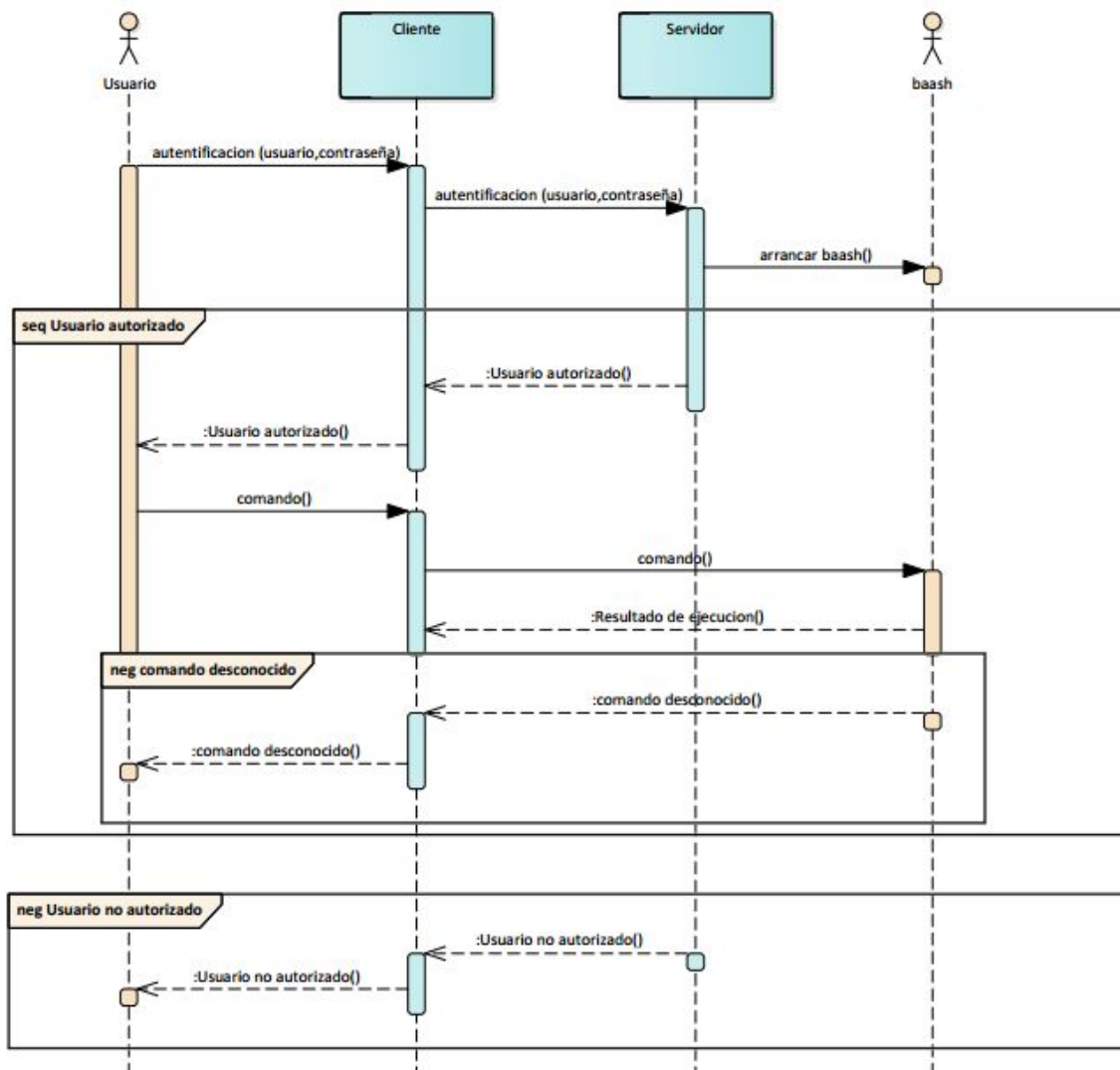


Figura 2. Diagrama de Secuencia.

Implementación y Resultados

El diseño se llevó a cabo desarrollando un software programado en lenguaje C_{[1][2]}, para el cual se realizó una documentación utilizando la herramienta Doxygen_[4], ver Apéndice A.

Para la implementación se utilizó una placa a disposición Raspberry Pi 3 B+, la cual tiene un sistema operativo Raspbian Jessie 4.14.34+.

Se desarrolló un archivo Makefile_[4] para la compilación de la aplicación, ver Apéndice B.

La transferencia de la aplicación a la placa se hizo usando SCP, estableciendo una conexión remota utilizando el protocolo SSH.

El usuario de la RPI es pi y la contraseña raspberry.

Se muestra en la Figura 3 a continuación una prueba de logueo del usuario.

```
federico@Bangho-PC ~/Escritorio/so2/tp1/tp1_cliente_baash $ ./tp1_cliente_baash
Para establecer conexion con el servidor debe ingresar: connect usuario@numero_ip:port
Cliente: connect usuario@10.42.0.63:6020
USUARIO usuario. Conectado a server de direccion: 10.42.0.63 en puerto: 6020.
Usuario: usuario
Ingrese la clave: pass
Ingreso con exito
usuario@raspberrypi:/home/pi/so2/tp1/tp1_servidor_baash$
```

Figura 3. Logueo de usuario.

En caso de que el usuario introduzca un comando desconocido, el cliente muestra un menú de opciones válidas.

```
usuario@raspberrypi:/home/pi/so2/tp1/tp1_servidor_baash$ desconocido
No existe el comando
usuario@raspberrypi:/home/pi/so2/tp1/tp1_servidor_baash$
```

Figura 4. Menú de opciones válidas.

En las siguientes figuras se muestran capturas de pantalla de las pruebas de los comandos introducidos por el usuario.

```
usuario@raspberrypi:/home/pi/so2/tp1/tp1_servidor_baash$ ls
armar_path_relativo.c
armar_path_relativo.h
baash
baash.c
baash.h
buscar_en_PATH.c
buscar_en_PATH.h
cantidad_de_retrocesos.c
cantidad_de_retrocesos.h
index.c
Makefile
parsear_entrada.c
parsear_entrada.h
pipe_argumentos.c
pipe_argumentos.h
tp1_servidor_baash
tp1_servidor_baash.c
usuarios.csv

usuario@raspberrypi:/home/pi/so2/tp1/tp1_servidor_baash$
```

Figura 5. Comando ls.

Con motivos de probar el comando descarga se agregó el directorio para_enviar dentro de la carpeta del proyecto. Este directorio contiene archivos de distintos tipo como imágenes, documentos, etc. En este caso se eligió hacer la descarga de la imagen rpi.png

```
usuario@raspberrypi:/home/pi/so2/tp1/para_enviar$ ls
fechas
materias.pdf
mediciones.csv
rpi2.jpg
rpi.png
tp1.odt

usuario@raspberrypi:/home/pi/so2/tp1/para_enviar$ descarga rpi.png
Preparando archivo rpi.png....

CREANDO ARCHIVO rpi.png

ARCHIVO rpi.png
CREADO
usuario@raspberrypi:/home/pi/so2/tp1/para_enviar$
```

Figura 6. Comando descargar.

Conclusiones

Se logró implementar la aplicación desarrollada, en el hardware disponible, incorporando los conocimientos básicos sobre Sockets TCP y UDP.

Además, se incorporó conocimientos sobre documentación utilizando la herramienta Doxygen.

Apéndices

Apéndice A

La documentación obtenida utilizando la herramienta Doxygen se encuentra en el directorio /documentacion. Se debe correr el archivo index.html en cualquier navegador disponible.

Apéndice B

Se desarrollaron los siguientes Makefile.

Makefile para el cliente

```
.SUFFIXES: .o .c
```

```
.c.o:
```

```
    $(CC) -c $(CFLAGS) $<
```

```
CC = gcc
```

```
SRC = tp1_cliente_baash.c
```

```
OBJ = tp1_cliente_baash.o
```

```
BIN = tp1_cliente_baash
```

```
CFLAGS = -Wall -pedantic
```

```
# Reglas explicitas
```

```
all:
```



```
$(CC) $(CFLAGS) $(SRC) -o $(BIN)
```

```
clean:
```

```
$(RM) $(OBJ) $(BIN)
```

```
Makefile para el servidor
```

```
.SUFFIXES: .o .c
```

```
.c.o:
```

```
$(CC) -c $(CFLAGS) $<
```

```
CC = gcc
```

```
SRC = tp1_servidor_baash.c parsear_entrada.c parsear_entrada.h buscar_en_PATH.c buscar_en_PATH.h  
cantidad_de_retrocesos.c cantidad_de_retrocesos.h armar_path_relativo.c armar_path_relativo.h  
pipe_argumentos.c pipe_argumentos.h baash.c baash.h
```

```
OBJ = tp1_servidor_baash.o parsear_entrada.o buscar_en_PATH.o cantidad_de_retrocesos.o  
armar_path_relativo.o pipe_argumentos.o baash.o
```

```
BIN = tp1_servidor_baash
```

```
CFLAGS = -Wall -pedantic
```

```
# Reglas explicitas
```

```
all:
```

```
$(CC) $(CFLAGS) $(SRC) -o $(BIN)
```

```
clean:
```


```
$(RM) $(OBJ) $(BIN)
```

```
# Reglas implicitas
```

```
parsear_entrada.o: parsear_entrada.c parsear_entrada.h
```

```
buscar_en_PATH.o: buscar_en_PATH.c buscar_en_PATH.h
```

```
cantidad_de_retrocesos.o: cantidad_de_retrocesos.c cantidad_de_retrocesos.h
```



armar_path_relativo.o: armar_path_relativo.c armar_path_relativo.h

pipe_argumentos.o: pipe_argumentos.c pipe_argumentos.h

baash.o: baash.c baash.h