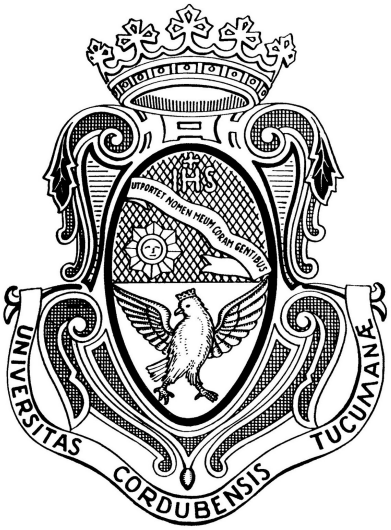
UNIVERSIDAD NACIONAL DE CÓRDOBA

FACULTAD DE CIENCIAS EXACTAS FÍSICAS Y NATURALES



SISTEMAS OPERATIVOS II

TRABAJO PRÁCTICO Nº 1: Sockets en sistemas tipo UNIX

Integrantes:

- Cazajous Miguel A. - 34980294

Córdoba - Argentina

24 de noviembre de 2018

Índice

1. Introducción:

1.1. Propósito:

El propósito en este práctico, es el desarrollo de una aplicación que nos permita obtener información de un servidor mediante la ejecución de diferentes comandos Unix, como así también el permiso de descargar archivos desde ese servidor.

1.2. Ambito del sistema:

La aplicación desarrollada está pensada para ser implementada bajo un Sistema Operativo GNU/Linux a través del uso de la terminal de este, por lo que cualquier computadora que disponga de una distribución GNU/Linux debería ser capaz de ejecutar el programa sin inconvenientes. Lo anterior expuesto hace referencia, tanto a la aplicación del lado del cliente como del servidor.

1.3. Definiciones, acrónimos y abreviaturas:

1. Prompt: Símbolo que se muestra en una terminal (normalmente es una barra vertical u horizontal parpadeante) para indicar que se está listo para recibir órdenes.

2. GNU/Linux: Sistema Operativo que tiene sus orígenes en Unix y debe su nombre al uso del kernel de Linux y el uso de Herramientas GNU.

3. Git: Sistema de control de versiones. Lleva un registro de todos los cambios efectuados en los archivos y permite volver a versiones anteriores fácilmente.

4. Software: Referente a programas, aplicaciones.

5. Hardware: Componentes físicos, computadora, placa de desarrollo, etc.

6. Sistema Operativo: Software principal residente en un hardware que permite la gestión de sus partes y su interacción con aplicaciones de usuario. Es una capa intermedia entre aplicaciones de usuario y el hardware de la máquina.

7. UDP: Conexión no segura.

8. TCP: Conexión segura.

9. Socket: Interfaz de programación de aplicaciones que permite la comunicación entre programas.

10. Ethernet: Estándar de transmisión de datos.

1.4. Referencias:

[1] William Stallings. Sistemas Operativos 5º edicion. 2005. ISBN: 84-205-44692-0.

[2] Stephen Brennan. LSH (LibStephen Shell). URL: https://github.com/brenns10.

[3] Wikipedia. Socket de Internet. URL: https://es.wikipedia.org/wiki/Socket_de_Internet.

[4] Mendez Gonzalo. Especificacion de Requisitos segun el estandar de IEEE 830. URL: https://www.fdi.ucm.es/profesor/gmendez/docs/is0809/ieee830.pdf.

[5] Lucid Software Inc. LucidChart. URL: https://www.lucidchart.com.

[6] GeeksForGeeks. File transfer using UDP. URL: https://www.geeksforgeeks.org/c-program-for-file-transfer-using-udp/.

1.5. Descripción general del documento:

Este documento se compone de 7 secciones principales, siendo la primera la introducción, donde se explica brevemente el fin del proyecto como así también sus requerimientos de forma muy general. En la segunda sección la descripción general del sistema con el fin de conocer las principales funciones que debe ser capaz de realizar, conocer a quién va dirigido, además de restricciones y supuestos que puedan afectar el desarrollo del mismo. En la sección 3 se definen de manera detallada los requerimientos del sistema a desarrollar, los que definen el comportamiento del sistema como así también otros requerimientos que puedan ser deseados considerando el uso que el sistema va a tener. En la cuarta sección se presenta el diseño del sistema que dará solución a los requerimientos antes enumerados. En la quinta sección se discuten los resultados de implementación del diseño elaborado en la etapa anterior donde se muestra como el sistema opera. En la sección 6 se elabora una breve conclusión acerca de la experiencia en la elaboración del proyecto. Finalmente en la última sección se agrega información alternativa que pueda ser de interés.

2. Descripción general:

2.1. Perspectiva del producto:

La aplicación deberá proveer un método rápido y sencillo de acceso a la información de manera remota a un servidor con el fin de conocer acerca de el. Se dispone, además un mecanismo de seguridad que permite confidencialidad en los datos.

2.2. Funciones del producto:

El sistema deberá ser capaz de:

1. Establecer una conexión entre un cliente y el servidor donde se encuentra la información.

2. Desplegar un menú de opciones y un prompt en el lado del cliente que permita operar con el servidor mediante comandos.

3. El servidor debe ser capaz de interpretar los comandos y tomar decisiones adecuadas a cada uno de ellos.

2.3. Características de los usuarios:

La aplicación está destinada a usuarios autorizados a obtener información del servidor. En cuanto a conocimientos técnicos, los usuarios deben tener una leve noción de como manejarse en un entorno GNU/Linux lo que incluye un manejo básico de comandos por consola.

2.4. Restricciones:

Como restricción de lenguage de programación se tiene que la aplicación debe ser en su totalidad elaborada bajo lenguage C mediante el uso de Sockets. Además como se mencionó el entorno donde corre la aplicación debe ser cualquier distribución basada en GNU/Linux.
En cuanto a hardware no se requieren mayores prestaciones. Cualquier computadora que pueda correr un GNU/Linux con fluidez es apta.
Del lado del servidor cualquier plataforma de desarrollo que sea compatible con GNU/Linux es igualmente válida.

2.5. Suposiciones y dependencias:

Un cambio en el sistema operativo donde se desea que corra la aplicación requeriría de cambios sustanciales en la funcionalidad del prodcuto. El hardware como se mencionó tiene un gran rango de variación aceptable que no generaría ningún inconveniente en el uso de la aplicación.

2.6. Requisitos futuros:

El proyecto tiene un fin específico y no está pensado para abarcar más requerimientos en el futuro, de todas maneras sería deseable que con actualizaciones mínimas del producto este pueda ser usado para extraer información de otros servidores, o se permita el ingreso de nuevos clientes dándoles permiso si estos desean logearse.

3. Requisitos específicos:

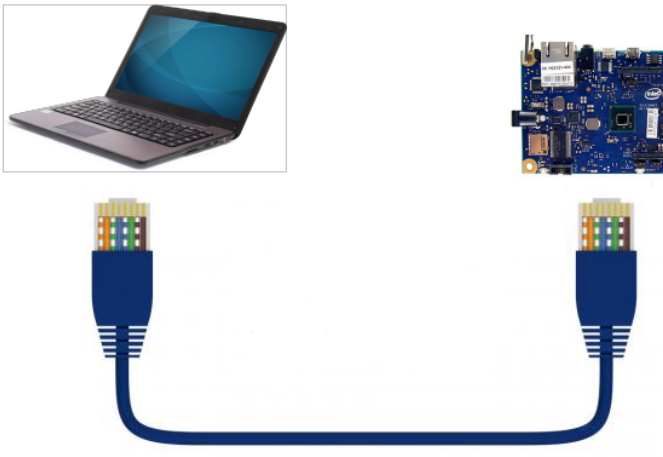
3.1. Interfaces externas:

Interfaz de software:

La interfaz de software no será gráfica, será simplemente una interfaz basada en ingreso de comandos desde la terminal.
El prompt y un menú simple de comandos es más que suficiente para el fin de la aplicación, una interfaz gráfica para estas operaciones sería un gasto inútil considerando que el usuario final del proyecto debe tener conocimientos en GNU/Linux, el manejo de la terminal no debería ser mayor inconveniente.

Interfaz de hardware y comunicación:

La interfaz de hardware y comunicación consta de una computadora que opera bajo GNU/Linux como cliente y una plataforma de desarrollo compatible que será el servidor donde está la información, donde ambas están conectadas entre sí mediante un cable Ethernet.



3.2. Funciones:

3.3. Requisitos de rendimiento:

La aplicación es interactiva y se espera que el tiempo de respuesta sea el mínimo posible para cualquier petición que se realice, siendo deseable que se obtengan resultados en milésimas de segundo para operaciones que no incluyen descarga de datos. Para esta última se considera que una espera de 2 segundos como máximo considerando archivos tipo texto es un tiempo límite de aceptación.

La cantidad de conexiones entrantes que el servidor pueda tener no se ha fijado, aunque el mismo sería capaz de soportar varias conexiones sin problemas, además al ser operaciones de lectura la integridad de los datos no se pierde si ocurren operaciones concurrentes.

3.4. Restricciones de diseño:

Las únicas restricciones de diseño que se tienen al momento de elaborar este documento es la restricción de lenguage de programación que se debe usar junto con el uso de Sockets para su implementación y la presentación mediante archivos de compilación Makefile.

3.5. Atributos del sistema:

El sistema de seguridad será implementado mediante un login de usuario y contraseña con el formato descrito en puntos anteriores, además el servidor cuenta con un permite identificar que comandos son incorrectos o inválidos.

El sistema es fácilmente portable a cualquier computadora y plataforma de desarrollo que cumpla con los requisitos.

La mantenibilidad del softwre es sencilla ya que el mismo lleva consigo un control de versiones implementado mediante la herramienta Git.

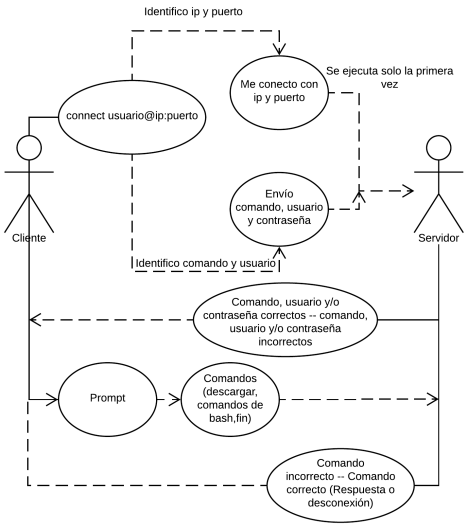
Factores que pueden afectar principalmente la fiabilidad del software están en un fallo del SO tanto en la computadora como en la plataforma de desarrollo que correrá el sistema servidor y también un fallo entre la conexión de las anteriores mencionadas.

4. Otros requisitos:

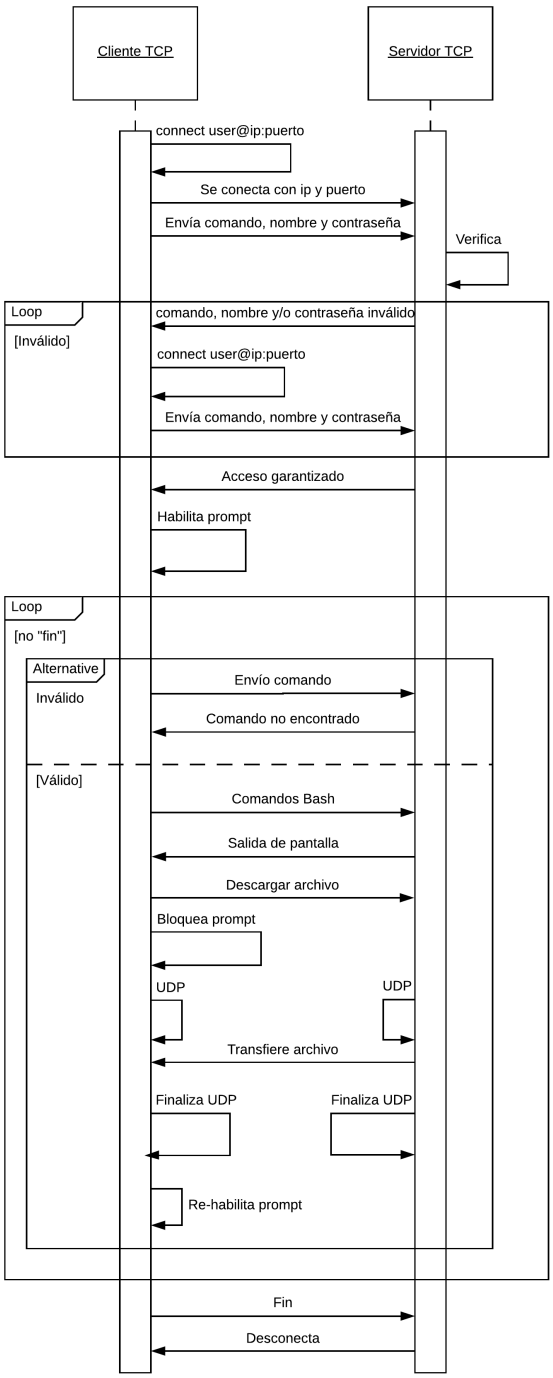
No se tienen otros requisitos presentes que no hayan sido ya mencionados en las secciones anteriores.

5. Diseño de solución:

5.1. Representación de los requerimientos mediante: casos de uso: (Click en la imagen para abrir)



5.2. Diagrama de secuencia: (Click en la imagen para abrir)



6. Implementación y resultados:

Los resultados que se muestran a continuación por cuestiones de disponibilidad de mostrarán en una misma computadora, es decir servidor y cliente comparten el mismo host.

Inicialmente tenemos la interfaz de cliente arriba a la izquierda, al lado los archivos de lado del cliente y debajo el servidor. Siendo este último no observable cuando esté corriendo en Raspberry Pi.

Figura 1: Interfaz Cliente

A continuación se muestran dos mensajes de error al ingresar un puerto erroneo y no ingresar un número de puerto en absoluto.

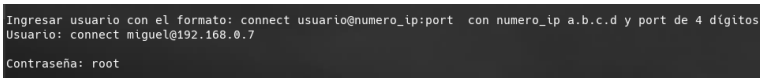


Figura 2: Número de puerto incorrecto.

Figura 3: Número de puerto no ingresado.

Nótese que al ingresar ":" seguido de NULL considera como número de puerto incorrecto. Para el caso de una contraseña, usuario y/o comando incorrecto se muestra el siguiente mensaje.

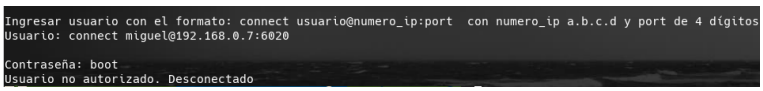


Figura 4: Usuario no autorizado.

Si el usuario es autorizado el servidor lo da a saber con el siguiente mensaje. El usuario puede acceder al servidor y se le habilita un prompt para el ingreso de comandos.

Figura 5: Usuario autorizado.

Mediante el ingreso de "?" obtenemos una ayuda de los comandos que pueden ingresarse.

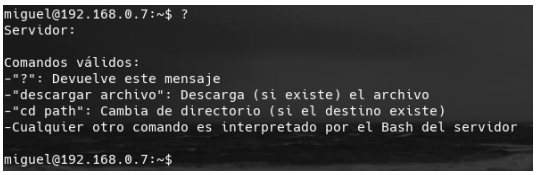


Figura 6: Ayuda.

Mostramos a continuación 3 comandos que según la ayuda serán interpretados por el bash, teniendo salidas para comandos tipo "ls -ls", "df -h" y un mensaje vacío para mensajes que el bash no puede interpretar.

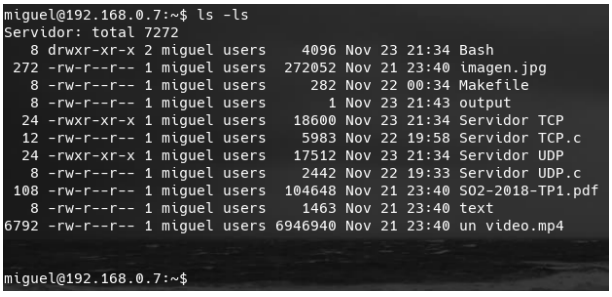


Figura 7: Salida ls -ls

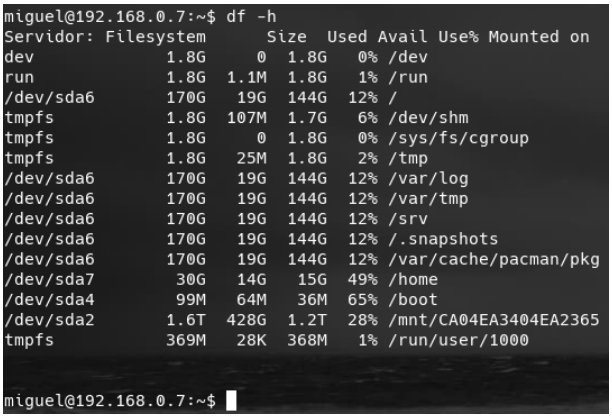


Figura 8: Salida df -h

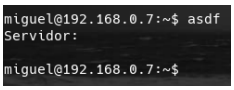


Figura 9: Salida comando aleatorio

Vemos a continuación una serie de capturas que nos permite ver el comportamiento del comando “cd” ayudándonos con el comando “pwd”.

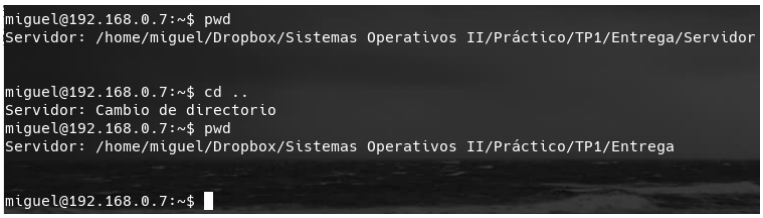


Figura 10: Cambio a directorio anterior.

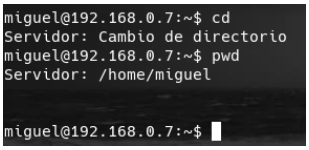


Figura 11: Cambio a directorio raiz.

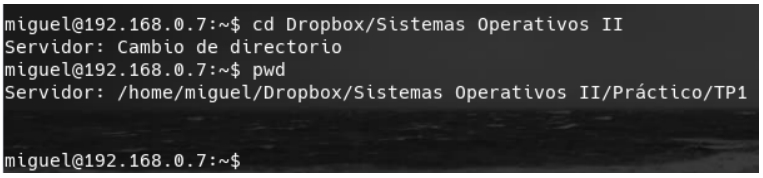


Figura 12: Cambio a directorio con espacios.

En la primera captura se observaba el directorio del cliente con solo los archivos del programa. Vamos a ver como luego de la ejecución del comando "descarga" tenemos nuevos archivos que han sido transferidos desde el servidor.

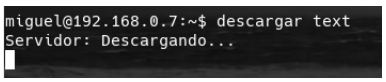


Figura 13: Descarga un archivo de texto

Figura 14: Descarga un archivo pdf

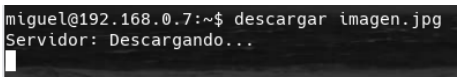


Figura 15: Descarga una imagen

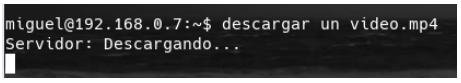


Figura 16: Descarga un video

Utilizando un “file manager” vemos que las miniaturas de los archivos nos indican que no se han corrompido.

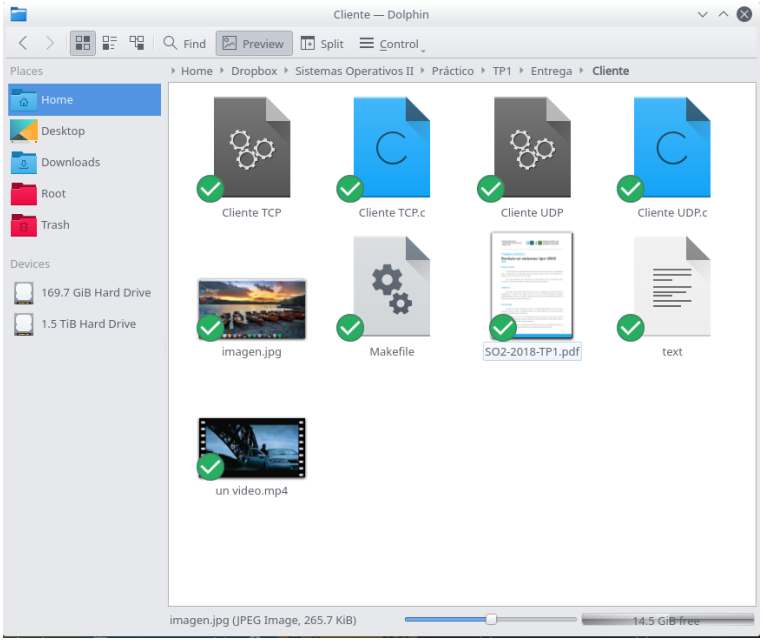


Figura 17: Miniaturas de archivos descargados

Aquí no se han hecho capturas de todos los comandos que bash puede interpretar, pero se han provado comandos como “cp”, “mv”, entre otros y funcionan sin inconveniente. Un comando que también funciona pero no debería ser usado es “rm” ya que no se han implementado medidas de prevención ante la ejecución de ese comando con cualquier archivo en el servidor.

7. Conclusiones:

Mediante la realización de este trabajo se pudo comprender como funciona una comunicación por socket mediante dos procesos.

Se lograron adquirir nuevas prácticas en la programacón C al utilizar estructuras nunca antes vistas como las encargadas de la comunicación por Sockets y se retomaron otras como la manipulación de archivos.

Se encontraron dificultades al momento de guardar el archivo en la transferencia por UDP, pues el cliente lo recibía corrompido. El problema se pudo solucionar aunque quizás no de la manera más óptima, la cual requeriría más pruebas.

El bloqueo de ingreso por teclado durante la descarga de un archivo fue solucionado, nuevamente, de una manera rápida pero no las más óptima.

8. Apéndices:

8.1. Sockets:

Un socket representa un concepto abstracto por el cual dos procesos intercambian información.

Están los Sockets Unix y los Sockets Internet, estos últimos son los que son de interés para nuestro caso.

Un Socket es una API (Interfaz de programación de aplicaciones) para la familia de protocolos TCP/IP que provee el sistema operativo. Esto quiere decir que son un conjunto de funciones listas para ser utilizadas como una capa de abstracción ya que no presenta detalles de su implementación.

Se define un socket por medio de un par de direcciones IP (origen y destino), un protocolo de transporte (TCP o UDP) y un par de números de puertos (origen y destino).

Los sockets permiten implementar una arquitectura cliente-servidor. El servidor está a la espera de que un cliente inicie la comunicación.

Protocolo TCP y UDP:

1. TCP:
   a) Orientado a conexión.
   b) Se garantiza la transmisión de todos los bytes sin erroes ni omisiones.
   c) Se garantiza que todo byte llegará a su destino en el mismo orden en que se ha transmitido.
2. UDP:
   a) No orientado a conexión.
   b) Solo garantiza que si un mensaje llega, llegue bien. En ningún caso se garantiza que llegue o que lleguen todos los mensajes en el mismo orden que se mandaron.
   Es adecuado para el envío de datos frecuentes.

Orígenes:

Sus orígenes se atribuyen a la universidad de Berkeley en una variante del sistema operativo Unix conocida como BSD Unix. Al notar que las computadoras iban a necesitar un medio sencillo y eficaz para la comunicación entre procesos se dio origen a la primera implementación de sockets.