Universidad
Nacional
de Córdoba

Cátedra de Sistemas Operativos II

Trabajo Práctico Nº I

Soriano, Juan

22 de abril del 2018

Índice

Introducción

En esta sección se encuentra la especificación de los requerimientos de la aplicación Sockets correspondiente al trabajo N° 1 de la Sistemas Operativos II.

Si partimos de la definición de un requerimiento, como las características que debe tener el sistema o las restricciones que debe satisfacer para la aceptación del cliente. La especificación de los requerimientos y de las restricciones está escrita de manera tal que el cliente pueda entender.

Propósito

El propósito del Trabajo Práctico nrº 1 de Sistemas Operativos II es el de diseñar e implementar un software que utilice la API de sockets del Sistema Operativo aplicando los conocimientos adquiridos durante la matería, así también como en Sistemas Operativos I e Ingeniería de Software.

Definiciones, Acrónimos y Abreviaturas

Login: Ingreso.

SSH: Secure SHell.

API : Interfaz de Programación de Aplicaciones

Referencias

1. Linux man pages: https://linux.die.net/man/
2. GNU, “5 Making The Best Use of C”, https://www.gnu.org/prep/standards/html_node/Writing-C.html
3. W. Richard Stevens, Stephen A. Rago, Advanced Programming in the UNIX Environment, 3rd Edition, Adison-Wesley

Descripción General

En la presente sección se explicarán los distintos aspectos del trabajo. Las funciones que ofrece el producto, la perspectiva del mismo, las carácterísticas de los usuarios, requisitos para futuras iteraciones, entre otras cosas.

Perspectiva del Producto

Según Wikipedia: “Un Socket designa un concepto abstracto por el cual dos programas (posiblemente situados en computadoras distintas) pueden intercambiar cualquier flujo de datos, generalmente de manera fiable y ordenada.

El término socket es también usado como el nombre de una interfaz de programación de aplicaciones (API) para la familia de protocolos de Internet TCP/IP, provista usualmente por el sistema operativo.

Los sockets de Internet constituyen el mecanismo para la entrega de paquetes de datos provenientes de la tarjeta de red a los procesos o hilos apropiados. Un socket queda definido por un par de direcciones IP local y remota, un protocolo de transporte y un par de números de puerto local y remoto.”

La idea del producto es emular el comportamiento que pueda tener una conexión mediante SSH entre un cliente o host y un servidor o server. Con un previo sistema de Login, el producto ofrece la posibilidad de ejecutar comandos en el servidor de manera remota. Estos comandos son ejecutados por el programa “baash”, programa en lenguaje C realizado durante el transcurso de la materia Sistemas Operativos I.

El presente producto se presenta en forma de dos programas en lenguaje C ejecutables. Uno que se ejecutará en el servidor y otro en el cliente.

Servidor: Se implementan las funcionalidades necesarias para la autenticación del cliente que realiza la conexión, manejo de múltiples clientes en simultáneo y transferencia de archivos.

Cliente: Se implementan las funcionalidades necesarias para establecer una conexión de forma segura (TCP) para el intercambio de mensajes y una conexión no orientada a la conexión (UDP) para la descarga de archivos.

Funciones del Producto

Utilizando Sockets, se podrán establecer múltiples conexiones TCP entre múltiples clientes y un único servidor. Una vez realizada la conexión, se lleva a cabo una etapa de autenticación en la cual el cliente de identifica e intenta loguearse al sistema. Si no es aceptado por el servidor, se le pedirá que vuelva a autenticarse.

Una vez llevada a cabo la etapa del login, el usuario recibe el prompt del programa “baash”, sobre el cual podrá ejecutar los comandos básicos de una consola, por ejemplo, “cd”, “ls”, “cat”, realizar pipes, entre otras funcionalidades que no serán cubiertas en este informe. Si el cliente así lo desea, puede iniciar una conexión insegura (UDP) para descargar un archivo proveyendo el nombre del mismo. En caso de que el archivo exista, el servidor lo enviará por protocolo UDP.

Características de los Usuarios

El producto se destina a usuarios con conocimientos básicos del manejo de sistemas operativos Linux por consola.

Restricciones

Las restricciones son muy bajas. Se requiere de una computadora con requerimientos de hardware mínimos capaz de correr el programa del servidor y, por otro lado y de la misma forma, una computadora con requerimientos de hardware mínimos que ejecute el programa del cliente. Debe haber conectividad de red entre ambas.

Suposiciones y Dependencias

El presente producto no presenta un sistema de seguridad robusto, por lo cual debe ser siempre utilizado en redes privadas y seguras. La performance del producto se encuentra atada a la calidad de la conexión entre cliente y servidor, por lo que si la calidad de la conexión es mala, los resultados obtenidos no serán de buen calibre.

Requisitos Futuros

Queda un gran abanico de posibles funcionalidades para ser agregadas, a continuación se listan algunas de ellas:

- Mejorar el sistema de autenticación con el fin de hacerlo más robusto y seguro.
- Solucionar problemas de concurrencia que ocurren cuando múltiples usuarios se encuentran conectados en simultáneo.

- Control de integridad del archivo descargado por medio de una función de hash como puede ser SHA1.

Requisitos Específicos

En la presente sección se explayará con mayor lujo de detalle las entradas y salidas del sistema como así también una descripción de las funciones y requerimientos funcionales del producto.

Interfaces Externas

A continuación se ofrece una breve descripción de las interfaces implementadas.

Interfaz de usuario

La interfaz que se ofrece al usuario es un Prompt en una consola. Esto se hizo para simplicidad del uso del programa por parte del usuario y para reducir el volumen de errores.

Sockets

Para la comunicación se utilizaron tanto Sockets TCP como UDP.

Funciones

En esta sección se listan los requerimientos funcionales de la aplicación para la versión 1 del release del producto con fecha 22 de Abril de 2018:

1. Conexión por medio de sockets TCP, ingresando la dirección IP y el número de puerto.
2. Manejo de multiusuario por parte del servidor, proveyendo la escucha a múltiples conexiones de distintos clientes en simultáneo.
3. Autenticación con usuario y contraseña del cliente que se realice del lado del servidor.
4. Verificación de la validez del usuario y la contraseña del cliente del lado del servidor. Que resulte en una notificación al cliente con la leyenda “Usuario y/o contraseña incorrectos”
5. Prompt del lado del cliente que le indique que hay conexión y le muestre quién se encuentra ejecutando comandos en dicho momento.
6. Ejecución de comandos sobre “baash” de forma remota por parte del cliente en el servidor, manteniendo una conexión segura.
7. Envío de la salida del programa “baash” desde el servidor hacia el cliente por medio de la conexión TCP.
8. Descarga de un archivo de forma remota que se encuentre en el servidor para el cliente ingresando el comando “descarga <nombre de archivo>” utilizando conexión no segura.

Requisitos de Rendimiento

En esta sección se listan los requerimientos funcionales de la aplicación para la versión 1 del release del producto con fecha 22 de Abril de 2018:

1. Uso de memoria mínimo y apropiado para que el programa pueda ser corrido en una placa de desarrollo con MMU, como puede ser una placa Raspberry Pi.
2. Tiempo de respuesta del prompt en el cliente menor a 1 segundo para mantener la ejecución del “baash” en tiempo real.

Restricciones de Diseño

En esta sección se listan las restricciones de diseño de la aplicación para la versión 1 del release del producto con fecha 22 de Abril de 2018:

1. Tanto el código del cliente como el del servidor serán desarrollados en lenguaje C.
2. Se utilizará el puerto 6020 como puerto para la conexión TCP, puerto en el cual se encontrará en todo momento escuchando el servidor.
3. El intérprete que se ejecutará del lado del servidor será el programa “baash”, realizado en Sistemas Operativos I.
4. El servidor estará montado en una placa de desarrollo que soporte sistemas operativos del tipo GNU/Linux.
5. Sintaxis de código del tipo GNU o Linux Kernel.

Atributos del Sistema

En esta sección se listan los atributos del sistema de la aplicación para la versión 1 del release del producto con fecha 22 de Abril de 2018:

1. Sistema portable, es decir, deberá ser capaz de correr en cualquier sistema operativo.
2. Sistema compilado con GCC y flags -Werror -Wall -pedantic
3. Se correrá la herramienta de análisis de código Cppcheck.

Implementación y Resultados

Primero se realiza make, a continuación se presenta el makefile:

```
CC=gcc
CFLAGS=-Werror -Wall -pedantic

all: build/client build/server

build/client: build/client.o
    $(CC) $(CFLAGS) build/client.o -pthread -o build/client

build/client.o: src/client/main.c
    $(CC) $(CFLAGS) -c src/client/main.c -o build/client.o

build/server: build/server.o
    $(CC) $(CFLAGS) build/server.o -o build/server

build/server.o: src/server/main.c
    $(CC) $(CFLAGS) -c src/server/main.c -o build/server.o

clean:
    rm build/client*
    rm build/server*
```

Como se puede ver, no arroja warnings de ningún tipo:

```
torce@torcedesk ~/Desktop/JuanSoriano/TP1 $ make
gcc -Werror -Wall -pedantic -c src/client/main.c -o build/client.o
gcc -Werror -Wall -pedantic build/client.o -pthread -o build/client
gcc -Werror -Wall -pedantic -c src/server/main.c -o build/server.o
gcc -Werror -Wall -pedantic build/server.o -o build/server
```

A continuación se procede a ejecuta la aplicación del lado del server con el comando ./server dentro de la carpeta build, nótese que el binario puede ser portado.

Ahora, en el cliente se ejecuta la aplicación del cliente con el comando ./cliente que de la misma forma que el binario del server, puede ser portado. En el cliente se presenta un pedido de ingreso del comando:

```
Ingrese connect usuario@numero_ip:nr_puerto
>>comando 123
Expresion invalida, ingrese nuevamente
Ingrese connect usuario@numero_ip:nr_puerto
>>
```

Como se puede ver, si se ingresa una expresión inválida, se pide que se la ingrese de nuevo.

```
>>connect usuario@127.0.0.1:6020
Conexion establecida
Por favor ingrese el password
>>contrasenia
Usuario y/o contrasena invalidos, por favor ingrese el usuario
>>
```

Cuando se ingresa una expresión válida pero una combinación usuario/contraseña inválidos, el servidor envía el mensaje “Usuario y/o contrasena invalidos, por favor ingrese el usuario” para realizar un nuevo intento.

```
Usuario y/o contrasena invalidos, por favor ingrese el usuario
>>admin
Por favor ingrese el password
>>admin
autenticacion exitosa
torce@torcedesk:/home/torce/Desktop/JuanSoriano/TP1/build$
```

Una vez autenticados exitosamente, se presenta el prompt. Ahora podemos ingresar comandos que serán ejecutados por el servidor en el programa “baash”.

```
torce@torcedesk:/home/torce/Desktop/JuanSoriano/TP1/build$ ls
baash
client
client.o
incomingFile
outgoingFile
server
server.o
torce@torcedesk:/home/torce/Desktop/JuanSoriano/TP1/build$ cd ..
torce@torcedesk:/home/torce/Desktop/JuanSoriano/TP1$ ls
build
cmake-build-debug
CMakeLists.txt
docs
Makefile
src
test
TODO.txt
```

Como se puede ver, los comandos se ejecutan correctamente.

```
torce@torcedesk:/home/torce/Desktop/JuanSoriano/TP1$ descarga outgoingFile
Descarga exitosa!
```

Y el archivo se descarga con el nombre “incomingFile”. Si el archivo no existe, lo indicará. Además, si se cierra la conexión del lado del servidor, la conexión del lado del cliente también se cerrará.

Conclusiones

A modo de conclusión se puede decir que si bien al trabajo le quedaron varios aspectos por mejorar, se logró el cometido que es el de utilizar los conceptos aprendidos en diversas materias de la carrera. Se aplicaron los conocimientos de comunicación por sockets, comunicación entre procesos y muchos otros.

Al implementarse el trabajo en lenguaje C, se presentan diversos problemas cuya resolución no es trivial. Sin embargo, se pudieron sortear los obstáculos y se logró un resultado satisfactorio.