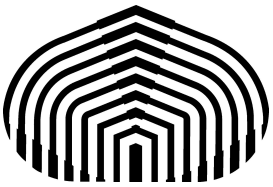

Universidad Simón Bolívar

Sistemas Operativos I

# *Proyecto II*

# *Terminal Esencial*

Nicolás Jaua | 15-10732

Marcos Lerones | 15-10778

*Sartenejas*

*07 de Junio de 2019*

# Introducción

En este proyecto se realizó un shell minimal que solo cuenta con 3 comandos: ls, grep y chmod. El comando ls se usa para listar los contenidos de un directorio, el comando grep se usa para obtener líneas de un archivo que contengan cierto patrón y el comando chmod se usa para cambiar los permisos de un archivo.

Además de implementar estas rutinas, también se crearon mecanismo para redireccionar el input y el output de un comando. Estos mecanismos son los operadores '<', '>' y '|'. El funcionamiento de estos es como su funcionamiento en el shell bash. El operador '<' hace que el comando que se encuentra a la izquierda del operador lea del archivo que se encuentra a la derecha del mismo en lugar de leer del *standard input*. El operador '>' hace que el comando que se encuentra a la izquierda sobreescriba el archivo que está a la derecha en lugar de escribir en el *standard output*. Por último, el operador '|' hace que el *standard output* del comando a su izquierda sea redirigido al *standard input* del comando a su derecha.

# Ejecución del programa

Para correr este programa se debe compilar con el uso del comando `'make'` (el makefile ya está incluido para crear todos los ejecutables necesarios). Luego, teniendo los 4 ejecutables en el directorio actual se debe correr usando

```
./tesh [script]
```

Si ningún script es especificado, se proporciona un programa interactivo que despliega un prompt para informar al usuario que puede usar la línea de comandos y va leyendo y ejecutando línea por línea hasta que el usuario introduzca el comando exit para terminar la ejecución.

# Implementación

Para la implementación de este programa se dividió el trabajo en dos grandes partes. La primera es la implementación del terminal en sí, que se encargue de los redireccionamientos y los llamados a los comandos, y la segunda parte son los comandos que son llamados por el terminal. Cada comando fue implementado en un archivo propio (ls.c, grep.c y chmod.c) y el terminal fue implementado en un archivo aparte (tesh.c). A continuación se explicará el funcionamiento y la implementación de cada uno de estos archivos por separado.

`tesh.c`:

Esta parte del proyecto se debe encargar de implementar los operadores de redireccionamiento y las llamadas a los distintos comandos. Para hacer esto, se tomó como referencia el shell bash. En bash, los operadores '<' y '>' tienen mayor precedencia que el operador '|' y además, los 3 asocian de izquierda a derecha. Esto significa que si, por ejemplo, tenemos algo como esto:

$$\texttt{[comando] < in}_1 \texttt{ < in}_2 \texttt{ < … < in}_k$$

entonces toma como entrada estándar del comando el archivo que se encuentre más a la derecha en la cadena (en este caso $\texttt{in}_k$).

Además, si tenemos algo como esto:

$$\texttt{cmd}_1 \texttt{ > out.txt | cmd}_2$$

donde $\texttt{cmd}_1$ y $\texttt{cmd}_2$ son comando válidos, la salida estándar de $\text{cmd}_1$ se redireccionará al archivo `out.txt` y nada llegará a la entrada estándar de $\texttt{cmd}_2$.

Las cadenas de pipes funcionan como uno esperaría, como una tubería de izquierda a derecha, donde el output de un comando se redirecciona hacia el input del siguiente.

Para hacer esto posible se tienen 2 funciones:

- `void execCommandLine(char *cmd)`
- `void execCommand(char *cmd, int inFD, int outFD)`

La función execCommand recibe el comando a ejecutar, un descriptor de archivo de entrada y un descriptor de archivo de salida. Lo que hace esta función es llamar a fork() y en el proceso hijo se encarga de encontrar los archivos de entrada y salida del comando (indicados con los operadores < y >). Si estos archivos existen, se abren y sus descriptores se guardan en inFD y outFD respectivamente (ignorando lo que tenían antes). Esto se hace así para lograr la mayor precedencia de los operadores < y > sobre el |. Luego de hacer esto, se utiliza la función dup2 para reemplazar la entrada y salida estándar del proceso por los archivos cuyos descriptores se encuentran en inFD y outFD respectivamente. Por último, se separa el comando por espacios (para obtener el comando y sus flags/argumentos) y se ejecuta mediante la llamada al sistema execvp(). Mientras tanto, el padre espera a que el proceso hijo termine con la función wait().

La función execCommandLine recibe una línea de comandos y la separa por el carácter | usando strtok() y guarda estos comandos en un arreglo. La función guarda una variable llamada inputFD que indica cual es el descriptor de archivo para la entrada del siguiente comando. La función recorre el arreglo de comandos y si el comando no es el último de la lista, crea un pipe y llama a execCommand con inFD igual a inputFD y outFD igual al descriptor de escritura del pipe. Luego a inputFD le asigna el descriptor de lectura del pipe y continúa el ciclo. Si el comando es el último de la lista, entonces igualmente se llama a execCommand con inFD igual a inputFD pero ahora, outFD será STDOUT_FILENO (el descriptor de la salida estándar).

`Funciones`:

Las tres funciones ls, grep y chmod atacan su problema de la misma forma. Toma los argumentos recibidos como strings en argv e identifica si son flags o no. Iteran sobre cada argumento primero revisando si el primer caracter es - (o + en caso de chmod), y en caso de ser así iteran sobre el resto de los caracteres, revisando que coincidan con los flags permitidos para esa función. Las tres funciones tienen un arreglo de enteros, dónde se marca como 0 si el flag no está presente o 1 si lo está. En caso de que no se encuentre - o + al comienzo del caracter se considera que este es el nombre de un archivo, en el caso de grep, in directorio, en el caso de la o alguno de los dos en el caso de chmod; estos serán guardados en un arreglo de

strings. El primer argumento distinto de un flag que se encuentre en grep será considerado el string a buscar.
Luego de identificar correctamente todos los argumentos y almacenar su información se comienza a iterar sobre los archivos o directorios, realizando el propósito de la función.

### ls.c:

En ls se hace uso de dirent para poder iterar sobre los archivos contenidos en un directorio. La información de cada archivo es accedida usando la función stat. Con esta se puede encontrar los datos que ofrecería ls -l en el bash, haciendo algunos cambios para que la información se presente de la misma forma. En el caso de los permisos de debe verificar bit por bit para ofrecerlo en el formato deseado, para el tiempo se debe convertir a fecha y hora y en el caso de él tamaño se debe calcular el correspondiente en kilobytes o megabytes.

### grep.c:

En el caso de grep no se iteran por defecto sobre los archivos, ya que se podría no recibir ningún archivo. En este caso se le piden inputs al usuario y los imprime en caso de encontrar el string deseado como substring en el input. En caso de haber archivos se itera sobre ellos y luego se itera sobre sus líneas, buscando correspondencia con el string dado. En ambos casos se usa strstr (o strcasestr en caso de no considerar las mayúsculas), que devolverá un apuntador al substring dónde comienza el string buscado en el input. Si la función se da sin el flag -v de imprimirá el la línea cuando la función devuelva el apuntador (número positivo) y en caso de tener -v se imprimirá si devuelve un número negativo.

### chmod.c:

Para chmod se usó el syscall chmod que ofrece C. En este caso para cada archivo se hacía uso de la función stat para conseguir los permisos que ya el archivo tiene, y sobre este aplicar operaciones bit a bit para apagar o encender los bits de lectura, escritura o ejecución. En caso de que los flags indiquen encender y apagar el mismo bit esto es ignorado.

# Conclusiones

El proyecto es completamente funcional en el estado actual. Cumple con las especificaciones de hacer un shell que pueda ejecutar ls, grep y chmod (con sus respectivos flags) y además redireccionar entrada y salida entre comandos. Para mejorarlo se pueden incorporar más comandos y hacerlo más robusto implementando un parser para el lenguaje del shell script que fue creado (muy similar al de bash). El proceso de investigación e implementación de este proyecto proporcionó un *insight* en cómo funcionan los intérpretes de línea comandos (CLI). Más aún, se pudo aprender más sobre el sistema de archivos de Linux y su estructura de directorios y su acercamiento al manejo de disco.