Sistemas Operativos

Ejercicio Entregable 1

Tal y como hemos estudiado con anterioridad, una shell provee una interfaz que permite al usuario comunicarse con el *kernel*. La estructura de esta interfaz requiere un interesante trabajo de ingeniería puesto que se deben tener en cuenta aspectos tales como la comunicación entre procesos, redirecciones, tareas en *background* o *foreground*, estructuras de control, entre otras características.

Sin duda alguna, las *shells* más utilizadas en la actualidad son aquellas basadas en la *Bourne shell*, también conocida como **sh**. Este tipo de *shells* tienen la estructura que hemos estudiado en esta materia, donde las redirecciones se hacen con los símbolos $>$ y $<$, los pipes se representan con |, los procesos se ejecutan en background con **&**, etc. Sin embargo, el alumno debe entender que esta es solo una de las tantas interfaces que se pueden diseñar para interactuar con el sistema operativo.

Con el objetivo de ilustrar este concepto, el presente trabajo propone el desarrollo de una *shell*, limitada, con una interfaz distinta a la estudiada. También se planteará una peculiar pre-condición: la *shell* debe ser desarrollada utilizando **bash**.

Fecha de entrega: 04/05

Modo de entrega: A través de la plataforma (se debe subir un archivo *.tar.gz* a la actividad que se creará para tal fin. El mismo debe contener la implementación de la shell junto con lo que se especifica en el punto 2)

1. Desarrollo de una *shell*:

   a) En base al comportamiento típico y conocido como **REPL** y los conceptos vistos en la Práctica 1 desarrolle una *shell* teniendo en cuenta las siguientes pautas:

      i) El formato de la *shell* debería respetar la siguiente estructura:

```
while true; do

  #Imprimir prompt y leer lo ingresado por usuario
      read -p <prompt> line

      #Validaciones
      #Obtener comando y sus parametros
  #Ejecutar built-in de esta shell y esperar si es
      necesario
  #O ejecutar built-in de bash
  #O ejecutar comando externo con el criterio
      descrito mas abajo

done
```

II) El *prompt* del usuario, por defecto, debe ser el siguiente: **@nombre_usuario_logueado>**

III) Debe ser capaz de ejecutar algunos comandos buint-in de bash que no se especifiquen más abajo, como el comando *cd*.

IV) Esta *shell* no debe soportar ni *pipes*, ni *sustitución de comandos*, ni *redirecciones*, ni la creación de *subshells* ni tampoco ejecución *background (&)*.

b) La shell deberá disponer de los siguientes comandos *built-in* (funciones del script):

I) **ls**: Tiene el mismo comportamiento que un ls de bash, a diferencia que cuando se ejecuta un ls en bash, este muestra los resultados en formato de columnas, mientras que el ls que se debe implementar deberá mostrar los resultados en una única columna (un resultado bajo el otro) Deberá ser capaz de recibir el parámetro -l: El resultado de la ejecución de este comando con el parámetro -l devolverá un listado similar al de un ls -l tradicional, con la salvedad de que los permisos serán mostrados utilizando notación octal. Por ejemplo:

ls -l de bash	**ls -l del intérprete a desarrollar**
-rw-r–r– 1 root root 2389 feb 18 12:28 bind.keys	-644 1 root root 2389 feb 18 12:28 bind.keys
-rw-r–r– 1 root root 237 feb 18 12:28 db.0	-644 1 root root 237 feb 18 12:28 db.0
-rw-r–r– 1 root root 271 feb 18 12:28 db.127	-r644 1 root root 271 feb 18 12:28 db.127
drwxr-xr-x 2 bind bind 4096 abr 14 09:28 local	d755 2 bind bind 4096 abr 14 09:28 local
-rw-rw-r– 1 root bind 673 abr 14 09:21 named.conf	-664 1 root bind 673 abr 14 09:21 named.conf

II) **sl**: Devuelve el mismo resultado que un ls, pero en orden inverso (la primer línea la mostrará última).

III) **cat**: Tiene el mismo comportamiento que un cat de bash.

IV) **tac**: Devuelve el mismo resultado que un cat, pero en orden inverso (la primer línea la mostrará última).

V) **prompt**: Cambia el *prompt* de la shell. Recibe un string como parámetro:

- ***largo***: el prompt se convierte en **@YoSoy-*nombre_usuario*>**
- ***uid***: el prompt se convierte en **@*uid_usuario*>**
- Caso contrario el prompt toma la forma por defecto.

VI) **quiensoy**: Imprime información acerca de quien esta logueado. Recibe SOLO un string como parámetro, debiendo chequearse:

- Si la cantidad de parámetros no es exactamente uno imprime "Cantidad de argumentos incorrecta. Solo es posible recibir uno solo."
- Recibe ***+h***: imprime **Yo soy nombre_usuario y estoy en la maquina "nombre_host"**
- Recibe ***+inos***: imprime **Yo soy nombre_usuario y tengo UID=uid_usuario**
- Caso contrario imprime **Yo soy nombre_usuario**

VII) **pwd**: Tiene el mismo comportamiento que un pwd de bash.

VIII) **mkdir**: Crea un directorio dentro del FileSystem. Como primer parámetro siempre debe recibir el *path* del directorio a crear y a continuación cualquier otro parámetro soportado por el comando *mkdir* de bash. También tendrá que tener en cuenta lo siguiente:

- Si el usuario que lo ejecuta no tiene permiso para escribir en la estructura del FileSystem, deberá devolver el mensaje de error "No tenés permiso!", sin mostrar otro error en la pantalla.

- Si al momento de realizar la operación hay algun error se debera visualizar en pantalla ”Error!, Directorio/s no creado/s” y nada más.

c) En **bash** y *shells* similares, todo comando ingresado es buscado en los directorios configurados en la variable de entorno **PATH**. El valor de esta variable consiste en directorios separados por el caracter **“:”**. Por ejemplo:

```
$ echo $PATH
/usr/local/bin:/usr/bin:/bin
```

Entonces, si se ingresa el comando **cat**, bash primero comprobará si el comando es *built-in*. Si no lo es, lo buscará en **/usr/local/bin**, si no lo encuentra lo buscará en **/usr/bin** y finalmente en **/bin**.

Implemente una solución similar para la shell que se está desarrollando. Utilice una variable de entorno con nombre **RUTA** que separe cada directorio con el caracter **“;”**, y que busque el comando ingresado de atrás hacia delante. Por ejemplo:

```
$ echo $RUTA
/bin;/usr/bin;/usr/local/bin
```

Entonces, si el usuario ingresa **tree**, primero se deberá comprobar si es un comando interno (una funcion bash en nuestro caso). Si no lo es, se lo buscará en **/usr/local/bin/**, luego en **/usr/bin** y finalmente en **/bin**. En caso de que no se lo encuentre, la shell deberá imprimir “No se pudo encontrar el programa ‘tree”’.

2. Cree un directorio llamado **externalSO** a la misma altura que la shell para luego agregar esa ruta a la variable **RUTA**. El mismo debe contener lo siguiente:
   - Un script llamado **serverNC**: que implementa el servidor del ejercicio uno de la primer práctica.
   - Un script llamado **clientNC**: que implemente el cliente del ejercicio uno de la primer práctica.
   - Un script llamado **scanner**: que implemente el ejercicio tres de la primer práctica pero recibiendo la lista hosts por parámetros.
3. Cree un usuario en el sistema y asociele la *shell* creada en el punto 1 con el fin de que la misma sea su intérprete de comandos.
4. Verifique que todos los comandos *built-in* implementados, algunos *built-in* de bash (*cd*) y los scripts alojados en **externalSO** funcionen.
5. ¿Qué sucede si algunos de los comandos *built-in* de este interprete, como el *ls*, no son ejecutados en *background*?

TIPS:

- Para crear una *subshell* puede utilizar **&**.
- Investigue las variables internas de *bash*. Una que le va a ser de utilidad es **$!**.
- Investigue los comandos internos de *bash*. Uno que le va a ser de utilidad es **wait**.