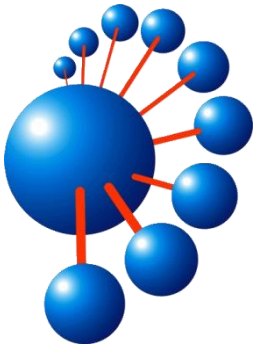


Proyecto
Sistemas Operativos
Shell Simple

Profesora: Cecilia Hernández
Integrantes: Karley Parada Haquin.
Diego Varas Moya.
Jeremías Torres Tapia.

Abril 4, 2017

Introducción

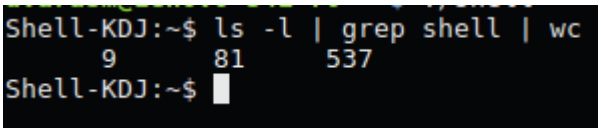
Se debe desarrollar un intérprete de comandos simples en Linux (Shell). La shell a implementar debe ser similar a la disponible en Linux. Para ello se deberá leer los comandos ingresados y ejecutarlos correctamente, incluyendo lectura de pipes (|).

Los pipes o tuberías son los encargados de llevar información de un comando a otro, es decir, la salida del primer comando será la entrada del comando siguiente y así sucesivamente.

En este trabajo se podrá ver un historial de comandos utilizados en la shell y ejecutarlos nuevamente si se requiere, además de obtener el tiempo de ejecución de los procesos.

Desarrollo

Nuestra Shell, proporciona un prompt para identificar el modo de espera de los comandos, luego de ejecutar se imprimirá nuevamente el prompt para esperar las siguientes instrucciones (en caso de presionar enter imprime nuevamente el prompt esperando nuevos comandos).



En el inicio del programa guardamos la línea completa de comandos, la cual en la función ***find_pipe()*** se divide por comando (al encontrar un |) y en ***parse_cmd()*** se divide los comandos de sus argumentos, a su vez se cuenta la cantidad de comandos ingresados, dependiendo de la cantidad existente.

Para ejecutar los comandos ingresados en la Shell, tenemos la función ***ejec()*** que hace una llamada a la función para ejecutar, según corresponda.

En el caso que no existiera un pipe, es decir, sólo se ejecuta un comando, tenemos la función ***ejecutar_cmd(),*** para la cual tenemos que crear un proceso mediante fork(), en el cual se hace que el proceso padre espere mientras que el proceso hijo, correspondiente a la ejecución del comando es efectuado.

En el caso que existiera un pipe, tenemos la función ***ejecutar_cmd2()***, para la cual creamos un pipe para comunicar el proceso hijo con el proceso padre, realiza la llamada ***pid1=fork()***, para crear los procesos, luego tenemos que si se logra la ejecución del fork(), en el proceso hijo se reemplaza la salida estándar, con el pipe que comunica los procesos mediante la llamada ***dup2(p[1], STDOUT_FILENO,*** a continuación se produce el cierre de los pipe y la ejecución del primer comando mediante la función ***execvp()***, a la cual se le pasa el comando principal y sus argumentos. En caso que no se logre ejecutar el comando, se desplegará un mensaje de error, luego el proceso padre está a la espera de que termine su proceso hijo (pid1==0) y a su vez tenemos que en el proceso padre, se crea otro proceso mediante la llamada ***pid2=fork()***, esto es necesario para poder ejecutar el segundo comando sin que se termine la ejecución de la Shell. En el segundo proceso hijo, tenemos que se reemplaza la entrada estándar, con el pipe de comunicación entre los procesos, ***dup2(p[0], STDIN_FILENO)***, luego se efectúa el cierre de los pipe y la ejecución del segundo comando, que se encontraba a continuación del pipe, de igual manera que en el proceso hijo anterior, en caso de que la ejecución del segundo comando no se haga efectiva, desplegará un mensaje de error. El proceso padre de este segundo proceso, está en espera (***wait()***) a que termine el segundo proceso hijo.

Para el caso de 2 pipe tenemos las funciones ***ejecutar_cmd3()***, que en general, actúa de la misma forma que la función ***ejecutar_cmd2()***, pero en esta nueva función tenemos un tercer proceso que se crea en el proceso padre del segundo proceso, y se realiza de la misma forma que el caso anterior.

Finalmente, a medida que los comandos sean ejecutados, éstos son guardados en un historial (***historial_cmd()***) junto a su medida de tiempo de ejecución la cual es paralelamente medida por ***getrusage()***.

En la consola se desplegará la información de los comandos ejecutados con su correspondiente tiempo de ejecución, acompañados por un id. Para llevar a cabo esta acción basta con escribir “historial” en la shell y si se desea reejecutar un comando ya utilizado, se debe escribir su id y presionar enter.

Para terminar con la ejecución de la Shell, sólo tenemos que escribir la palabra “exit”, esto nos lleva a la función ***terminar()*** y cierra la ejecución de nuestra mini shell.

Conclusión

En éste proyecto se pudo poner en práctica el uso de procesos concurrentes, además de familiarizarnos con algunas llamadas a sistema básicas de Linux, como fork() y exec(), para la creación de procesos, aprendimos acerca de las función pipe, que nos permitía el redireccionamiento de entradas y salidas.

Además se aprendió sobre otras funciones que pueden ser realizadas en la Shell de Linux, la capacidad de llevar a cabo trabajos en foreground y background, interactividad mayor con ciertos procesos, y como logra llevar a cabo múltiples pipe, etc, lo cual está muy por encima de lo logrado en este proyecto de mini shell.