Universidad Simón Bolívar.
Departamento de Computación y Tecnología de la Información.
Taller de Sistemas de operación I.
Enero-Marzo 2018.
Proyecto II.

Terminal esencial

Los intérpretes de órdenes, también llamados *shells*, son parte fundamental de un sistema de operación. A través de ellos se solicita la ejecución de órdenes suministradas por el usuario desde un terminal. Suelen incorporar características tales como control de procesos, redirección de entrada/salida, archivos, comunicaciones y un lenguaje de órdenes para escribir programas por lotes o *scripts*.

Es posible que un sistema de operación soporte varios intérpretes de órdenes o *shells*. Los usuarios escogen cuál utilizar según sus preferencias (*cshell*, *bash*, *korn*, etc.). Los *shells* comenzaron a evolucionar hacia interfaces gráficas de usuario (GUI) ampliamente usadas en los sistemas modernos. No obstante, una significativa minoría de usuarios prefieren usar directamente los *shells*, la mayoría de ellos porque les parece que éstos proporcionan un entorno de mejor productividad. Los interpretadores de órdenes son usados principalmente por programadores y administradores de sistemas, especialmente en sistemas de operación basados en Unix; en entornos científicos y de ingeniería.

Reconociendo la complejidad que adquieren la mayoría de los *shells*, se pide que haga un *shell* minimalista con tan solo tres comandos.

Requerimientos del programa

Un interpretador de órdenes muestra un *prompt* al usuario (un símbolo que indica que está listo para reconocer comandos). El usuario puede, entonces, escribir una orden que será ejecutada y generará una salida de texto. Las órdenes dadas al interpretador son de una de las siguientes formas:

- `[haz_algo] [como] [a_estos_archivos]`
- `[haz_algo] [como] < [archivo_de_entrada]`
- `[haz_algo] [como] [a_estos_archivos] > [archivo_de_salida]`
- `[haz_algo] [como] < [archivo_de_entrada] > [archivo_de_salida]`
- `[haz_algo] [como] [a_estos_archivos] | [haz_al_resultado] [como]`
- `[haz_algo] [como] | [haz_al_resultado] [como] > [archivo_de_salida]`

Donde

- < indica alimentación del contenido del archivo en sustitución de la entrada estándar.
- > indica redireccionamiento de la salida estándar al archivo.
- | indica alimentación de la salida estándar del programa precedente al programa sucesor en sustitución de la entrada estándar. Puede haber varios seguidos en un *pipeline*, por ejemplo:
  `ls|grep a|grep b>resultantes.txt`

Para este proyecto se les pide *implementar un shell* que permita realizar algunas operaciones básicas sobre archivos. Debe poder llamarse a su shell desde la consola de Linux de la siguiente manera:

`tesh [script]`

dónde `script` es un archivo que contiene una lista de comandos a ejecutar. En estos casos, no se debe proporcionar un prompt; solo deben ejecutarse las instrucciones del script.

Comandos a reconocer:

- **ls**: comando para mostrar la información del archivo (sea regular o directorio) que daría el comando de Unix `ls -l` (se recomienda leer la página correspondiente del manual de Unix). Debe obtener la información a través de `stat.h` o `dirent.h`. El comando reconocerá las siguientes opciones:
  - `-i`: (index) imprime el número del inodo.
  - `-G`: (group) omite el grupo del dueño del archivo.
  - `-g`: (group member) omite el nombre del dueño del archivo.
  - `-h`: (human-readable) imprime el tamaño del archivo en kilobytes (seguido de K) ó, si ocupa más de 800K, megabytes (seguido de M).
  - `-R`: (recursive) imprime la información de los archivos contenidos en los directorios. A diferencia del comando de Unix, si se encuentra un subdirectorio dentro de los mismos, no deberá obtener sus contenidos.
- **grep**: comando para buscar coincidencias de un string en los archivos de entrada. A diferencia del comando de Unix, este comando solo busca que el string esté contenido en la línea, no cualquier expresión regular. Reconocerá siguientes opciones:
  - `-i`: (ignore) ignorar mayúsculas y minúsculas.
  - `-v`: (invert) invierte el sentido de coincidencia para seleccionar líneas del archivo que no hacen match.
- **chmod**: realiza los cambios en el `struct stat` correspondiente al archivo. Debe reconocer las siguientes opciones:
  - `+r`: otorgar permiso de lectura.
  - `-r`: quitar permiso de lectura.
  - `+w`: otorgar permiso de escritura.
  - `-w`: quitar permiso de escritura.
  - `+x`: otorgar permiso de ejecución.
  - `-x`: quitar permiso de ejecución.

Entrega

Debe entregar su código en un archivo `.tar.gz` ó `.tgz` antes del viernes, 7 de junio de 2019, día en el cual se verificará que el programa corre en las computadoras de la universidad. Si se usa cualquier otro formato de compresión, su código no será revisado. Debe entregar su proyecto mediante un correo electrónico a su profesor/preparador y usar la página del curso (aula virtual-moodle).

Su código será probado en máquinas del LDC, así que se le recomienda que haga uso del comando `ssh` para que pueda probar su solución/soluciones. Obviamente, el ambiente de desarrollo y pruebas es software libre y Linux, será inaceptable para este proyecto entregar soluciones en Windows.

Informe

Debe incluir un informe (es importante la *buena ortografía* y el uso del *buen uso* del español). En general, y en el informe deben incluir al menos:

- Introducción.
- Contenido:
    - Cómo correr su programa
    - Cómo diseñaron y estructuraron su solución, y por qué
    - Cualquier decisión de implementación tomada o dificultad encontrada
- Conclusiones
    - Qué estado tiene el proyecto
    - Cómo mejorarían su implementación
- Fuentes consultadas

No se soportará ningún tipo de plagio o copia de ideas. Es importante el pensamiento crítico y las ideas que se desarrollen en el informe. Recuerde identificar a su equipo en el informe.

Evaluación

Se asignarán:

- 6 puntos por código
    - 1 punto por su uso correcto de `stat.h`
    - 1 punto por la estructura modular e intuitiva del código
    - 1 punto por la escalabilidad y robustez de su código
    - 2 puntos por tener todos los comandos
    - 1 punto por respetar las convenciones de C en Unix (Makefile, parámetros, etc)
- 5 puntos por ejecución.
    - 1 punto por manejar *scripts*
    - 0,5 punto por procesar archivos correctamente con `grep`
    - 0,5 punto por procesar la entrada estándar correctamente con `grep`
    - 1 punto por impresión correcta con `ls`
    - 0,5 punto por manejar archivos de salida
    - 0,5 punto por operar correctamente `chmod`
    - 1 punto por manejar *pipelines*
- 4 puntos por su informe