Laboratorio de Sistemas de Operación I
Septiembre-Diciembre 2014
Proyecto II

Objetivo

Familiarización con las llamadas al sistema para la creación y acceso a las estructuras del sistema de archivos y para el envío y recepción de información entre procesos.

Introducción al Problema

Los intérpretes de comandos son parte fundamental de un sistema de operación. A través de ellos se solicita la ejecución de órdenes suministradas por el usuario por medio de una interfaz alfanumérica. También se denomina *shell*. 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 solo sistema de operación tenga varios intérpretes de comandos o "*shell*", es más, la visión lógica de un sistema operativo cambia según el intérprete de comandos usado. Ejemplo: bash o korn sobre LINUX.

Los *shells* comenzaron a evolucionar hacia interfaces gráficas de usuario (GUI) ampliamente usadas en los sistemas modernos, no obstante, una significativa minoría de usuarios de computadoras prefieren usar directamente los *shells*, la mayoría de ellos porque les parece que éstos proporcionan un entorno de mejor productividad. Los interpretadores de comandos son usados principalmente por programadores y administradores de sistemas, especialmente en sistemas de operación basados en Unix; en entornos científicos y de ingeniería; y por un subconjunto más pequeño de usuarios domésticos avanzados.

En su forma más simple, un interpretador de órdenes muestra un **prompt** al usuario, éste *escribe* una orden en el teclado y finaliza la orden (normalmente con la tecla Intro), y la computadora ejecuta la orden, proporcionando una salida de texto.

Las órdenes dadas al interpretador son con frecuencia de la forma

```
[haz_algo] [como] [a_estos_archivos]
```

o

```
[haz_algo] [como] < [archivo_de_entrada] >
[archivo_de_salida]
```

Descripción general del problema

Para este proyecto Ud. debe implementar un *Shell* que permita realizar, sobre una jerarquía de directorios, algunas operaciones básicas sobre los archivos allí contenidos. Para cada directorio se debe crear un proceso que sea responsable de las operaciones sobre los archivos. La comunicación entre los procesos será a través de pipes no nominales.

Algunas restricciones aplican:

1. Se debe crear un proceso por cada uno de los subdirectorios dentro de la jerarquía dada, la raíz suministrada inclusive.

2. Su *shell* siempre estará *parado* en el directorio raíz del sistema. Y las referencias a archivos (dentro del FS) deben ser absolutas (ruta desde esa raíz). En otras palabras, no existe un comando que permita cambiar el directorio actual.

3. Sólo el proceso asociado al directorio que sea la raíz del árbol podrá realizar interacción con el usuario, es decir, ningún otro proceso debe imprimir por consola ni leer valores del teclado. Note que este proceso también será responsable de las operaciones sobre los archivos que él contenga.

4. Puede asumir que los archivos dentro de la jerarquía serán todos archivos de texto (subconjunto de los archivos regulares).

5. Sólo el proceso asociado a un directorio puede realizar operaciones sobre los archivos contenidos en ese directorio.

6. Un proceso sólo podrá enviar y recibir información a o desde su padre o hijos.

7. El *shell* sólo puede ejecutar un orden a la vez. En otras palabras, un comando no comienza a ejecutarse hasta que el anterior haya terminado.

8. No es necesario (para la versión básica) el reconocimiento ni manejo de las redirecciones.

9. Los comandos que el *shell* podrá reconocer deben ser implementados como comando internos, es decir, no podrá hacerse uso de los comando de Unix (no puede hacer uso de las funciones exec, system ni ninguna función similar para ejecutar comando externos desde su shell).

10. Ninguno de los comando debe manejar las expresiones regulares (ejemplo, *.c, ?asa.txt, etc).

Sobre los comando a reconocer

1. ls <path>: debe mostrar la información del archivo (sea regular o directorio) que daría en el sistema Unix el comando “ls –l”

2. cat <path>: debe mostrar por consola la información contenida dentro del archivo en caso de que éste exista y sea un archivo regular, y debe dar un error en caso contrario.

3. cp <path1> <path2>: copia el archivo regular referenciado por path1 al archivo referenciado por path2. Si path2 existe su contenido será sobreescrito, en caso contrario debe ser creado; en ambos casos, al completarse la orden deberá tener el mismo contenido que el archivo fuente. Si path2 es un directorio, se debe crear un archivo dentro de él con el mismo nombre y mismo contenido que el archivo referenciado por path1. Si path1 es un directorio se debe reportar el error.

4. mv <path1> <path2>: renombra el archivo path1 con el nombre path2. Si path1 es un directorio se debe reportar el error. Note que esta opción puede mover un archivo regular de un directorio a otro pero no podrá mover directorios.

5. find <cadena>: este comando deberá imprimir en consola la lista de todos los archivos (desde la raíz) que contengan la cadena en su nombre. Si cadena es **a**, se imprimirán todos los archivos (ruta absoluta) que tengan una **a** minúscula en su nombre, si un directorio tiene como nombre **algo** los archivos en él contenidos cumplirán la condición. En otras palabras, se considera como parte del nombre de un archivo su ruta absoluta.

6. rm <path1>: eliminará el archivo regular referenciado por path1. Si path1 es un directorio se debe dar un mensaje de error.

7. mkdir <path1>: creará el directorio. Si path1 es "/sopi/pepe", debe crear el directorio **pepe** dentro del directorio **sopi** que debe estar ubicado en la raíz del sistema. Si path1 existe debe dar un mensaje de error. En caso de éxito, este comando debe generar el proceso que será responsable de realizar las operaciones sobre el nuevo directorio, las estructuras que almacenan información de comunicaciones entre padres e hijos deben ser actualizadas (en los procesos pertinentes).

8. rmdir <path1>: debe eliminar el directorio del sistema. Si path1 es un archivo regular o directorio no vacío debe dar un mensaje de error. En caso de éxito, se debe eliminar el proceso asociado al directorio borrado y las estructuras que almacenan información de comunicaciones entre padres e hijos deben ser actualizadas (en los procesos pertinentes).

9. quit: el shell debe salir, todos los procesos creados deben ser finalizados. Es la única forma de salir del sistema, la combinación de teclas cntrl-c y afines no deben "matar" el shell.

Notas:

- Recuerde que los archivos son referenciados en forma absoluta dentro del Shell.
- Los mensajes de error deben ser apropiados al tipo de error detectado.
- Los cambios sobre la estructura de directorios deben poder verse:
  - dentro del sistema, con direcciones absolutas desde la raíz de su shell.
  - fuera del sistema, con direcciones absolutas desde la raíz de Unix.

Implementación

El ejecutable de su proyecto debe invocarse de la siguiente forma

fssh <directorio>

Donde <directorio> es la ruta (absoluta o relativa) del directorio que será la raíz del sistema de archivos que el programa fssh reconocerá. En otras palabras, si la invocación es:

fssh /home/yudith

significa que los archivos que estén fuera de él (/home, /home/yudith, etc) no serán visibles, es decir, para el sistema no existen. Adicionalmente dentro del sistema, /home/yudith será "conocido" como / (la raiz del sistema de archivos).

Versión extendida (Puntos extras)

Esta versión se diferencia de la básica en que ésta si reconoce tres de los direccionamientos básicos (>, <, |).

Ud debe definir e implementar un mecanismo que permita la ejecución de comandos con estos direccionamientos respetando las restricciones básicas del planteamiento. Recuerde que su shell siempre esta "parado" en el directorio raíz y que las operaciones de I/O sobre un directorio deben ser realizadas **directamente** por el proceso asociado al directorio correspondiente. Toda comunicación entre procesos debe realizarse a través de pipes no nominales.

Recomendaciones

Para la realización del proyecto se les sugiere que siga las siguientes recomendaciones:

- Entender bien el problema antes de diseñar su propuesta.
- Lea la información dada por las páginas de manual del sistema sobre las funciones que debe usar.
- Antes de comenzar a implementar, diseñe su propuesta tomando en cuenta todas las especificaciones dadas.
- Al terminar la implementación de cada módulo verifique que todas las rutinas funcionan correctamente (de la forma como ud lo planificó)
- Implemente en forma incremental: genere un plan sobre el orden en que debe implementar los módulos diseñados. Por ejemplo (esto es sólo un ejemplo):
    - Genere el árbol de procesos con todas sus estructuras de datos asociadas
    - Implemente un módulo de comunicación basado en pipes
    - Implemente un módulo con …
    - Etc.
- Recuerde que sólo el proceso asociado a la raíz puede interactuar con el usuario. Ningún otro proceso puede imprimir en consola o leer de teclado en absolutamente ningún momento.
- Trabaje en forma modular, estructurada y en orden. No repita un mismo código en diferentes funciones. No hacer esto, muy probablemente es equivalente a implementar un módulo de generación de errores difíciles de detectar y reparar.
- Verifique si hay comandos de su shell que puedan ser implementados como secuencias de otros comandos del shell.

Entrega

La entrega del proyecto está pautada para el día martes de semana 11