Instituto Tecnológico de Costa Rica

Arquitectura de computadoras

Profesor:
Jaime Gutiérrez

Estudiantes:
Jason Castillo
Jean Carlo Umaña

Segundo semestre, 2013

Introducción

Este documento contiene información referente al desarrollo del proyecto “Intérprete de comandos”; se va explicar la manera en que se desarrolló el proyecto, las funcionalidades que se lograron implementar y las que no, y la metodología que se usó para la implementación.

También se incluye un análisis crítico de las decisiones tomadas, y usadas, para el desarrollo del proyecto, algunas recomendaciones que podrían ser de utilidad para personas que desarrollen el mismo proyecto (o alguno similar), conclusiones que se obtuvieron tras haber finalizado el proyecto y algunas referencias que fueron de gran utilidad a lo largo del desarrollo del proyecto.

Este proyecto consiste en la utilización de 6 comandos distintos, algo muy parecido a los comandos que se puede usar en la terminal de linux (de hecho, se ejecuta en la terminal de linux). Los comandos implementados funcionan con archivos que se encuentran en la misma carpeta en que se encuentre el código fuente de este proyecto, para funcionar se ingresa el comando que se quiere ejecutar (ya se explicaran) seguido de un espacio y el nombre del archivo sobre el cual se va a ejecutar; el primero es el de borrar, lo que hace es eliminar un archivo existente en la carpeta en que se esté trabajando; el comando mostrar lo que hace es imprimir en la terminal el contenido del archivo que se ingrese; el comando renombrar cambia el nombre de un archivo al que se indique; el comando copiar crea un archivo con el mismo contenido del indicado a otro con el nombre que se indique; el comando comparar va a verificar si el contenido de dos archivos es el mismo, y sino se va indicar en que parte es que son distintos; y el último comando es salir, que lo que hace únicamente es terminar la ejecución del programa.

Es importante mencionar que para ver los resultados de los comandos lo mejor es usar archivos de texto plano.

También para cada comando se ofrece la opción --ayuda, que sirve para poder entender el funcionamiento de los comandos durante su uso; y también, para los comandos de renombrar y borrar se ofrece la opción --forzado, esto permite poder ejecutar el comando directamente, sin que aparezca el mensaje "¿Seguro que desea realizar está acción", ya que en estos dos comandos primero se realiza la pregunta mencionada.

Algunos ejemplos de como funcionan los comandos:

JJCLI# borrar nombre-archivo
JJCLI# borrar nombre-archivo --forzado
JJCLI# borrar --ayuda
JJCLI# mostrar nombre-archivo
JJCLI# mostrar --ayuda
JJCLI# renombrar nombre-archivo otro-nombre
JJCLI# renombrar nombre-archivo otro-nombre --forzado
JJCLI# renombrar --ayuda
JJCLI# copiar nombre-archivo otro-nombre
JJCLI# copiar --ayuda
JJCLI# comparar nombre-archivo segundo-archivo
JJCLI# comparar --ayuda
JJCLI# salir

Análisis del problema

Cuando se dio inicio al desarrollo del proyecto lo primero que se hizo fue pensar en el nombre que iba a tener el prompt, se decidió llamarlo “JJCLI#”, esto por los nombres de los dos integrantes del proyecto, el significado de las siglas sería: Jason & Jean Command Line Interface. Lo siguiente fue decidir si estaría diseñado para la terminal minimizada o maximizada, se decidió que minimizada ya que así aparece cuando se abre, de igual manera si se usa con la terminal maximizada no presenta ningún inconveniente.

Con la parte estética lista se dio inicio al proyecto propiamente; se decidió empezar por el comando mostrar, ya que parecía el más sencillo y además de que iba a ser de utilidad en la parte de --ayuda de todos los comandos.

Para el comando mostrar se encontró información que fue de ayuda y también acertada para la implementación.(se incluye en las referencias); lo que se hace es utilizar el servicio stat (este proporciona información sobre el estado de un archivo) para poder averiguar el largo que tiene un archivo; una vez que se tiene el largo lo que se hace es usar el servicio para abrir un archivo, luego se usa el servicio de leer y se lee la cantidad que se guardó tras haber usado el servicio stat (esa cantidad queda en el registro eax) y se imprime, una vez que se imprimió se cierra el archivo.

Con el comando mostrar funcionando se pasó a implementar la manera de separar los argumentos cuando se ejecutada un comando, porque las pruebas se estaban realizando en líneas separadas, algo así:

```
mostrar
nombre-archivo
```

entonces se pasó a tomar todo el comando en una sola línea, para esto se realizan varios ciclos. Para hacer el proceso de separar los argumentos se usan

varios buffers, dos son temporales, cuatro son usados para guardar los argumentos (ya que es la cantidad máxima de argumentos que se pueden ingresar) y uno para guardar toda la línea que se ingresa; lo primero que se hace es recorrer caracter por caracter la línea ingresada, esto se hace por medio de un ciclo que se ejecuta hasta que se encuentra con un espacio, y cada caracter que se encuentra lo mueve a uno de los buffers temporales, una vez que se encontró un espacio entonces se pasa a otro buffer temporal el argumento, pero esta vez le quita el último caracter (ya que en el primer ciclo agarra el espacio, y esto daría problemas a la hora de ejecutar el comando) y una vez que se tiene el argumento, por así decirlo, "limpio" se mueve al buffer correspondiente que será utilizado para la ejecución del comando; para saber si hay o no argumentos que agarrar lo que se hace es comparar el caracter por el cual se va (actual, digamos) leyendo en la línea que se ingresó con 0, si es igual es porque ya no hay nada más ahí entonces para de leer, por el contrario seguiría leyendo; y para saber el número de un argumento (si es el primero, segundo...) se lleva un "contador" en el registro esi, entonces cada vez que se mueve un argumento al respectivo buffer se aumenta dicho contador, por ejemplo: una vez que se obtuvo el primer argumento y se colocó en el buffer donde debe ir, se aumenta el contador. De esta manera se tienen en lugares fijos los argumentos que se van a usar para ejecutar un comando (más adelante se explicará exactamente como funciona la ejecución de un comando).

Con mostrar listo y agarrando todo en una línea se pasó a implementar --ayuda, para esto se usa el comando mostrar; lo que se hace es mover al buffer que utiliza el comando mostrar para agarrar el nombre del archivo a mostrar, el nombre de un archivo (que está como una hilera, y el archivo debe estar en la carpeta en la que se encuentra el código) que contiene la descripción de la manera en que se usa el comando, y luego se ejecuta mostrar.

Luego de implementar --ayuda se pasó a buscar información sobre los demás comandos a implementar, y se encontró información muy valiosa y que sirvió en el desarrollo de los demás comandos (se agrega en las referencias); lo que contiene dicha información es como utilizar los servicios disponibles para nasm.

Sabiendo los servicios que se podían utilizar para los comandos restantes se prosiguió con la implementación de renombrar, lo primero que se hace es mover los argumentos ingresados a los buffers correspondientes de la manera que se explicó anteriormente, con eso en orden se utiliza el servicio 38, se deben acomodar los registros de esta manera:

```
mov  eax, 38
mov  ebx, nombre-archivo-original
mov  ecx, nuevo-nombre-archivo
int   80h
```

lo que se mueve al ebx y al ecx son los buffers en donde se guardaron los argumentos ingresados, y de esta manera se cambia el nombre de un archivo.

La manera de implementar el comando borrar es muy parecida, primero se acomodan los argumentos ingresados en sus respectivos buffers y después solamente se utiliza el servicio 10 con el buffer donde quedó el nombre del archivo, algo así:

```
mov  eax, 10
mov  ebx,  nombre-archivo
int  80h
```

de esta manera se elimina el archivo que se ingresó.

Para el comando copiar, al igual que los anteriores comandos, se usa un servicio, el 9, y al igual que los anteriores se usan los buffers donde se guardaron los argumentos ingresados, funciona así:

```
mov  eax, 9
mov  ebx, nombre-archivo
mov  ecx,  nombre-del-nuevo-archivo
int  80h
```

así todo el contenido que haya en nombre-archivo también va a quedar en nombre-del-nuevo-archivo.

Para el comando salir nada mas se termina la ejecución.

En los casos de los comandos explicados era relativamente sencilla la implementación, ya que es tener orden con los buffers usados y saber obtener bien los argumentos; el problema más grande se presentó con comparar, para implementarlo se intentó de esta manera:

usar 2 buffers que servirían para guardar el contenido de los dos archivos, y llevar un buffer que sirviera como contador para llevar el número de línea por la cual se va, una vez que se movía el contenido de los archivos a los buffers se iba ir comparando caracter por caracter y si en algún momento no fueran iguales entonces se iba a imprimir lo que hubiera en el buffer contador; a pesar de que se intentó esta idea no dio resultado, ya que se indican mal las líneas en donde hay diferencias (no se logró identificar la causa del error, se cree que es cuando la línea de uno de los archivos es más larga que la otra, entonces llega en un punto donde todo se “descoordina”).

Ahora se explicará como funciona en sí el programa; cuando se inicia la ejecución lo que se muestra es el prompt, una vez que el usuario indicó la acción que quiere realizar se separan todos los argumentos ingresados en buffers distintos de la manera que se explicó anteriormente, por ejemplo que se haya ingresado: `JJCLI# renombrar archivo1 vivalaliga --forzado`

en este caso renombrar quedaría en el buffer primer_argumento, archivo1 en

segundo_argumento, vivalaliga en tercer_argumento y --forzado en cuarto_argumento; lo siguiente que se hace es identificar el comando que se quiere realizar, para esto se compara lo que hay en primer_argumento con “mostrar”, “borrar”, “renombrar”... luego se compara lo que hay en segundo_argumento con “--ayuda” por si se pidiera información de como funciona el comando, pero en este caso no se indica, entonces pasa a hacer uso de el comando renombrar, pero primero se verifica si en el cuarto_argumento está --forzado (que en este caso sí), y si está entonces ejecuta el comando directamente, y sino estuviera entonces primer pregunta si desea realizar la acción; una vez que se ejecutó el comando ingresado entonces se regresa al prompt para esperar el siguiente comando a ejecutar.

También después de cada ejecución de un comando de llama a una subrutina que se encarga de limpiar todos los buffers, esto es para evitar problemas en la ejecución de más de un comando seguido, y para limpiar lo que se hace es mover 0 a todos los buffers.

En cuanto al análisis crítico, se cree que se realizó un buen trabajo, la implementación utilizada dio resultado a pesar de que no se logró uno de los comandos, pero los demás funcionan correctamente.

Los errores identificados fueron:

-Cuando se ingresa el comando incompleto se acepta, por ejemplo: most, borr, sali...

-Cuando se deja más de un espacio entre los argumentos se cae el programa.

-Mal funcionamiento del comando comparar.

Conclusiones y recomendaciones

Tras haber finalizado el proyecto se puede concluir que se aprendió sobre lo que se puede realizar con respecto a un archivo, además de que se encontró una gran utilidad en utilizar buffers en lugar de espacios fijos, ya que estos se pueden “reutilizar”; también que es importante llevar un orden en el código, por ejemplo utilizar subrutinas que se puedan llamar, esto permite realizar una misma acción en distintas partes del código sin tener que poner lo mismo dos o más veces.

Algunas recomendaciones:

-Investigar bastante, cuando se topa con un trabajo sobre el cual no se tiene información no queda de otra que buscarla.

-Iniciar con lo más fácil, esto permite ir avanzando poco a poco y permite formar una idea de como se puede desarrollar el trabajo más difícil.

-Pedir ayuda, no todo se puede hacer solo y puede que 10 minutos de hablar con alguien permite el avance del trabajo.

-Trabajo en equipo, permite avanzar más rápidamente, también se brindan más variedad de ideas.

-Empezar a realizar el trabajo ingresando argumentos en líneas separadas, una vez que ya funcione de esta manera intentar implementarlo para que funcione en una sola línea.

-Limpiar lo que se usa, esto va evitar problemas de que se llegue a resultados extraños.

-Descansar de vez en cuando, es importante porque permite refrescar la mente y así llegan nuevas ideas.

Referencias

Linux Syscall References. Kernelgrok.com. Recuperado en noviembre, 13, 2013 de http://syscalls.kernelgrok.com/

NASM- Linux Dynamic memory Allocation/Read File. tuicool.com. Recuperado en noviembre, 12, 2013 de http://www.tuicool.com/articles/jiuyea