INSTITUTO TECNOLÓGICO DE COSTA RICA

Escuela de Computación

Arquitectura de Computadores

Grupo 20

II Tarea programada en ensamblador

CLI - Interfaz de linea de comandos

Integrantes:

Daniel Solís Méndez

Melvin Alonso Elizondo Pérez

Nombre del profesor:

Jaime Gutiérrez Alfaro

II Semestre 2013

Introducción:

El siguiente proyecto fue motivado por la Escuela de Computación del InstitutoTecnológico de Costa Rica como una iniciativa clara para incrementar el conocimiento de los estudiantes de dicha carrera en el curso de Arquitectura de Computadores.

El objetivo es utilizar algunos de los conceptos y técnicas de programación estudiados durante el curso, por ejemplo: estructuras de control, uso de la pila, manejo de la memoria de forma indirecta (a través de índices, como se hace en los arreglos en lenguajes de alto nivel); así como investigar con más detalle otras herramientas, servicios y técnicas para resolver adecuadamente el proyecto.

La siguiente documentación hace referencia a una tarea programada que debió ser desarrollada utilizando el lenguaje de programación ensamblador para Intel de 32bit.

Se intenta con este documento crear un registro de todo el proceso por el cual pasa el proyecto mostrando problemas, soluciones y recomendaciones para el mismo.

Análisis del problema:

El desarrollo del proyecto se nos facilito en gran medida si comparamos este con la primer tarea programada, por la experiencia adquirida, y la práctica que habiamos ejercido resolviendo el problema anterior.

Primeramente iniciamos reuniendo ideas del pasado proyecto programado que nos podrian servir en el actual. Nos dimos cuenta que ya teniamos una idea bastante clara de la implementacion de varias de las funcionalidades que se requerian para una Interfaz de linea de comandos.

Ya se había manejado un poco archivos, por medio de las matrices de puntos cargadas desde archivos, por lo que conociamos las bases del funcionamiento para manejar archivos.

Se empezó a cuestionar la utilizacion de funciones de C para facilitarnos la implementación de este proceso, pero llegamos a la conclusión de que no nos ayudaría en gran medida usarlas, por lo que las omitimos de la lista de herramientas.

Para inicar se contruyó la estructura base para el prompt, que fuera un ciclo hasta que se digitara la intruccion "salir" en la consola.

Una ves obtenido esto iniciamos los pasos para reconocer cada ves que era digitado un comando de los disponibles: copiar,comparar,borrar, renombrar, mostrar. Esto lo llevamos a cabo leyendo el comando en un buffer de memoria, y posteriormente analizando, con la ayuda de un registro para indexar el buffer, los caracteres que habian sido ingresados por el usuario, y comparandolos con los necesarios para que se formaran las distintas instrucciones. Ejm:

En el caso del comando "mostrar"

```
cmp byte[buffer+0] , 'm'
    jne ErrorComando
cmp byte[buffer+1] , 'o'
    jne ErrorComando
cmp byte[buffer+2] , 's'
```

```
jne ErrorComando
cmp byte[buffer+3] , 't'
jne ErrorComando
cmp byte[buffer+4] , 'r'
jne ErrorComando
cmp byte[buffer+5] , 'a'
jne ErrorComando
cmp byte[buffer+6] , 'r'
```

(ErrorComando seria una etiqueta que senala una direccion en memoria donde se ejecutan las instrucciones para mostrar un texto en pantalla que indica que se digito un comando no valido).

Para hacer mas agil el proceso se nos ocurrio chequear la primera letra del comando para ver si esta era una letra posiblemente "valida". Asi verificamos si era una "m"(mostrar), "b"(borrar), "r"(renombrar), y para la "c" generamos una "subrutina" que realizara la disticion entre copiar y comparar, ya que difieren en la tercer letra, y asi podriamos concluir directamente si no era una instrucción válida con solo analizar la primera letra, sin la necesidad de proceguir verificando el resto de letras de cada comando.

Posterior a cada nombre de comando debia seguir un espacio y sus parametros espeficicos, que en todos los casos eran nombres de archivos.

Para leer los nombres de los archivos se ingenió un ciclo que leia los caracteres después del espacio del nombre de comando y hasta que se encontrara un 10 que representa un enter.Asi, lograbamos extraer los nombres y depositarlos en otros buffer que nos iban a servir mas tarde para realizar la determinada operación del comando seleccionado.

Una tarea sencilla pero que requirió bastante tiempo, fue investigar sobre los servicios del kernel de linux, con lo que pudimos dar con la gran facilidad de muchos servicios del sistema que servian para copiar, borrar, renombrar, entre muchas otras funciones mas.

Con estos numeros de servicios y su descripcion pudimos llevar a cado de maneja muy sencilla el objetivo de la mayoria de comandos.

Para el parametro opcional de ayuda de cada uno de los comandos, nos encargamos en crear un archivo de texto plano con una breve descripcion de ayuda acerca de cada comando, para leer estos archivos e imprimirlos en pantalla cuando fuera necesario. Este proceso era el mismo necesario para mostrar un archivo de texto en la consola.

Mostrar

En el caso de mostrar el metodo de mostrarlo en pantalla se basaba en los siguientes pasos:

-Primero debiamos verificar si el archivo existia. Intentabamos abrirlo y si ocurria un error era porque el archivo no existia.

-Una ves confirmada su existencia, procediamos a leer el contenido del archivo en un buffer con un tamano bastante grande(para prevenir que el archivo que se deseaba mostrar fuera de longitud considerable).

-Y por ultimo, haciamos una llamada al servicio de impresion en pantalla con el texto previamente leido en el buffer.

Eliminar

Los pasos para eliminar eran bastante similares a los de mostrar solo que al final se eliminaba el archivo.

Este comando poseia un atributo opcional de --forzado que lo que introducia era omitir la pregunta de confirmacion de si se deseaba borrar. (De una manera similar se debía comprobar caracter por caracter para ver si el parametro --forzado habia sido digitado de manera correcta).

Si este habia sido escrito, simplemente nos saltabamos la pregunta de confirmacion haciendo un jump a una etiqueta q no incluia la pregunta y solo se dirigia directamente a intentar eliminar el archivo.

Renombrar

Para la solución de este comando, tras unas cuantas horas de investigación, dimos con un servicio del sistema "rename" que recibía de parámetros el nombre viejo y el nuevo y automáticamente cambiaba el nombre a un archivo en el sistema.

Antes de hacer la llamada al servicio "rename" se preparaban los parámetros de este. Se leía el nombre viejo en un buffer y el nuevo también en un buffer, luego estos buffer eran usados como los argumentos.

Copiar

Se encontró un servicio del kernel de linux llamado sys_link que era el encargado de realizar una copia de archivos. Este también recibía el nombre del archivo que se deseaba copiar y el nombre en el que se copiaría. Los nombres eran leídos de la misma manera

Comparar

Para compara se toma la decisión de subir el archivo entero a un buffer, ya que esto facilita el manejo por medio de un índice para moverse por el mismo y no cargar línea por línea, lo que probablemente se puede volver más engorroso y complicado, luego de esto se seleccionan dos registros totalmente reservados para mantenerlos como índices de los dos buffers que contiene la información del archivo, de estos buffers se mueven a dos registros de un byte bl, dl, los cuales se comparan entre si, esto si son iguales lo lleva a una subrutina que incrementa los índices en los registros para compara el próximo carácter.

Antes de verificar si son iguales se verifica que no sean 10, esto significa un enter o un cambio de línea esto hace que se utilicen dos nuevas subrutinas las cuales lo que hacen es mover el índice hasta el “enter” o 10 más cercano del otro documento, seguido de esto se agrega la línea donde se dio el cambio esto puede significar que los archivos muestran modificación o que simplemente cambia de línea, para esto se lleva un contador de la línea actual que se incrementa solo si los dos archivos han llegado al “enter”, luego de esto se verifica si son diferentes los caracteres entonces se procede a imprimir la línea donde son diferentes, y crear otras dos subrutinas auxiliares para mover los dos índices al “enter mas cercano hacia adelante” de los dos buffer.

Por último se verifica que alguno de los dos documentos no llegue a 0, esto significa que en ese punto el buffer no contiene más información que comparar, entonces se imprime la línea donde son diferentes y además el nombre del archivo el cual ha terminado y no contiene más información que comparar con ayuda de otras subrutinas correspondientes a cada caso.

Logs:

Esta parte de la tarea demando gran tiempo de investigación sobre el funcionamiento de un modo del servicio open llamado O_APPEND. Se encontró que para este fin se debía pasar el código 02000 que es el indicador de append, pero esto no funcionaba el archivo no se abria de manera correcta y generaba un error. Se decidió usar otro código hexadecimal que se vio en un ejemplo de internet que habre un archivo como lectura y escritura y, además, en modo append. Este código nos sirvió de maravilla y nos facilito la tarea de agregar lineas de texto a un archivo.

En los logs se guardan eventos importantes de la ejecucion del programa, como lo son algunos errores generados en el uso. Dentro de estos errores se incluye:

Cuando un comando es digitado de manera incorrecta o es totalmente un comando invalido.

Cuando se intenta mostrar, borrar, renombrar, copiar o comparar archivos que no existen(donde esta situación aplica).

Conclusiones:

El proyecto logra finalizar exitosamente, cumpliendo con la funcionalidad requerida y siendo, en lo que cabe, agradable al usuario. Se sacan grandes retroalimentaciones que nos servirán de aquí en adelante en nuestra carrera y vida profesional.

Este proyecto nos dio una perspectiva más clara de como es el manejo de archivos en un bajo nivel, qué argumentos se usan, cómo y para qué.

Es una experiencia muy agradable, ya que en ensamblador, muchas veces toca sacar ese lado creativo para hacer que algo funcione como se espera.

Ahora que se conoce mejor lo que pasa después de compilar un programa en alto nivel, se puede tomar conciencia de ciertas acciones para mejorarlas o corregirlas.

En general fue un eslabón que nos ayudó a aprender más acerca de la programación y nos deja conocimiento que será de gran ayuda para el resto de nuestra vida.

Recomendaciones:

- Creer que se puede lograr, no tenerle miedo solo por lo que dicen las personas.

- Investigar en internet, especialmente en http://stackoverflow.com/. Esta página cuenta con mucha información importante para el desarrollo de cualquier aplicación y ensamblador y nasm no es la excepción.

- Dividir el trabajo en partes pequeñas y fáciles de obtener, ir haciéndolo de manera progresiva y uniendo las partes alcanzadas.

- Llevar el código y la documentación de la mano, es mucho mejor para no acumular trabajo al final, y para comprender mejor lo que estamos haciendo y hemos hecho.

- Tomar decisiones en equipo, es mejor así se pueden dialogar ideas y escoger la mejor.

- Cada vez que se visita una página web en busca de información, y se da en el punto, inmediatamente añadir la dirección web a las referencias del trabajo.

- Lograr la funcionalidad del proyecto primero, y luego revisar excepción y restricciones del juego/aplicación.

Referencias:

Delete a file. Rosetta Code. Recuperado el 12 de noviembre del 2013 desde http://rosettacode.org/wiki/Delete_a_file

Create a file. Rosetta Code. Recuperado el 12 de noviembre del 2013 desde http://rosettacode.org/wiki/Create_a_file#X86_Assembly

Syscalls. Kernelgrok. Recuperado el 12 de noviembre del 2013 desde http://syscalls.kernelgrok.com/

Linux Programmer's Manual (27 de enero del 2013). Man7. Recuperado el 12 de noviembre del 2013 desde http://man7.org/linux/man-pages/man2/link.2.html

How to open a file in assembler and modify it? (29 de noviembre del 2011). StackOverflow. Recuperado el 17 de noviembre del 2013 desde http://stackoverflow.com/questions/8312290/how-to-open-a-file-in-assembler-and-modify-it

Reading from a file in assembly(27 de julio del 2010). StackOverflow. Recuperado el 17 de noviembre del 2013 desde http://stackoverflow.com/questions/3347747/reading-from-a-file-in-assembly

Archive(1 de noviembre del 2012). Max Bruning's weblog. Recuperado el 19 de noviembre del 2013 desde http://mbruning.blogspot.com/2012_11_01_archive.html

Linux x86 Shellcoding(29 de abril del 2011). 0xcd80.wordpress. Recuperado el 19 de noviembre del 2013 desde http://0xcd80.wordpress.com/2011/04/29/linux-x86-shellcoding-103/

Anexos

Bitacora:

6/11/2013

Se hizo una pequeña reunión para discutir ideas para el desarrollo del proyecto.

8/11/2013

Se empieza a definir la estructura base del programa.

12/11/2013

Se implementan los comandos de mostrar, borrar, copiar.

Renombrar se tiene implementado pero no funcional.

Se hacen las funcionalidades para los atributos --ayuda, creando las descripciones de cada uno en archivos de texto

17/11/2013

Se corrige el problema de renombrar descrito a continuación.

Un problema que nos surgió por descuido fue el digitar mal el numero de servicio de renombrar, por lo que este no funcionaba, parecía no hacer nada, y nos llevo bastante tiempo encontrar donde estaba el error.

Lo descubrimos tras ver de nuevo una tabla con los servicios del kernel y sus números, y revisar nuestra "tabla" de servicios. Así nos dimos cuenta que el numero del servicio en nuestro código estaba mal. Lo corregimos, volvimos a probar el código, y todo funcionando como se esperaba.

Limpiar buffer

El hecho de limpiar los buffer fue un problema que nos surgio, ya que al utilizar los mismos buffer tras cada comando, estos permanecian "sucios" con los caracteres usados en el ciclo anterior, por lo que el programa no operaba como debia debido al exceso de caracteres.

Con el fin de solucionar estos utilizamos algunos ciclos, con los buffer que nos daban problemas, para dejarlos de una forma que no nos afectaran al funcionamiento correcto del programa.

Se recorria cada buffer hasta encontrar su fin, sustituyendo cada caracter encontrado por un 0 que indica el final de un string o cadena de caracteres.

19/11/2013

Se logra generar un archivo de logs que llevara el registro de los eventos mas importantes en la ejecucion.

Se presenta un problema de manejo de la pila, tras hacer un call si se hace un pop se saca la direccion de regreso por lo que da un segmentation fault. Se debe tener cuidado con ello. Logramos descubrirlo rapido porque lo habiamos experimentado antes.