1. Introducción

A lo largo de este trabajo tratamos de crear una serie de funciones que ayuden al manejo de una imagen en formato FAT16. Estas funciones permiten ver la información de la imagen, como ser su tamaño total, la cantidad de FATs que contiene, etc; obtener un listado de todos los archivos y subdirectiorios que se encuentran en un directorio dado; obtener el tamaño de un directorio (junto con todos sus subdirectorios) y por último extraer un archivo de la imagen.

El TP fue programado casi en su totalidad en lenguaje ensamblador, según lo requerido por la materia, tan sólo usamos el lenguaje C para la función principal, desde la cual llamamos según corresponda por lo pasado en la línea de comandos a las diferentes funciones en Assembler, y para imprimir por pantalla los datos obtenidos ya que esto es más sencilllo en C que en lenguaje ensamblador

Las cuatro funciones a programar son en esencia iguales al principio, por lo tanto describiremos ese funcionamiento en un apartado, luego nos dedicaremos a observar el comportamiento de cada una de ellas por separado en detalle

2. Arquitectura de las Pcs utilizadas

Este trabajo fue hecho y testeado en múltiples computadoras, tanto las del laboratorio de computación como las nuestras, en todos los casos se usó el sistema operativo Linux de 32 bits, habiendo usado Debian, Ubuntu 9 y Ubuntu 10 como distribuciones en el testeo. En cuanto a los procesadores usados, podemos nombrar el Intel Atom y el Intel Core 2 Duo, en ambos casos el TP corrió normalmente

3. Descripción de las funciones

3.1. Comienzo de las fuciones

Todas las funciones reciben un parámetro a la imagen, el trabajo de esta parte es colocar un puntero al boot, esto es efectuado esencialmente de la misma manera en las cuatro funciones, dada la imagen se llama a la función en C open, siempre con el flag read only ya que no tenemos intención de modificar a la imagen. Esta función nos devuelve un file descriptor, el cual nos almacenamos en memoria para no perderlo.

Luego debemos llamar a una simple función C statsize, que calcula cual es el tamaño del struct stat en la PC actual, esto lo hacemos para asegurar que no pasamos el parámetro incorrecto ya que después debemos llamar a la función malloc para reservar memoria igual al tamaño de ese struct. Habiendo hecho esto, podemos llamar a la función fstat, que dado un file descriptor y un stat buffer rellena este último con la información del archivo. Dado esto, podemos observar el tamaño de este mismo, moviendo el puntero hasta ese offset, y así sabemos cuánta memoria vamos a necesitar para el mapeo.

Luego de llamar al malloc para que reserve esa cantidad de memoria, podemos proceder al mapeo con mmap que nos va a permitir recorrer la imagen, este llamado lo hacemos con los flags de read only nuevamente ya que no necesitamos modificar la imagen. Después de llamar a esta función, el valor que nos devuelve nos indica la primera posición del mapeo, ésta es la posición de booteo, que es donde empezamos a recorrer la imagen propiamente dicha, guardando antes esta misma posición y el tamaño de la imagen en memoria para su posterior acceso.

3.2. Imprimir

El objetivo de esta función es simplemente imprimir información de la imagen, ésta está toda contenida en el boot y nos va a servir a la hora de recorrerla. Aquí al encontrarnos en el boot simplemente llamamos a la función C printboot que es la que se ocupa de imprimir en pantalla la información necesaria. Esta función utiliza un struct definido por nosotros llamado booty, que lo usamos para organizar toda la informacion que se encuentra en memoria de acuerdo con cómo la queremos presentar. Por lo tanto, debemos asegurarnos que cada uno de los observadores se corresponda con lo que designan, como cada uno está asignado a un parámetro de diferente tamaño, es importante agregarle a la struct el atributo packed, de otra manera no nos permitirá ver lo que queremos. Luego, simplemente

pasamos cada uno de los observadores a un printf con su correspondiente descripción para exhibir toda la información relevante sobre la imagen

3.3. Lsdir

Esta función tiene el objetivo de presentar el contenido de un directorio, algo similar a lo que hacen los comandos dir en MS-DOS y ls en Linux. Aquí, además de la imagen se debe pasar el path al directorio del cual se quiere obtener el contenido.

Luego de posicionarnos en el root según la form ya descripta, procedemos a entrar al root de la imagen, el directorio raíz desde donde vamos a recorrer hasta el path dado. Para esto, debemos encontrar la posición del root, esto lo hacemos calculando el tamaño del boot, de los sectores reservados aparte de éste en caso de que los hubiere y de las tablas de FAT, tanto en su tamaño como en su cantidad. Es importante recalcar que a medida que avanzamos vamos guardando toda información que creemos necesaria en memoria ya que la necesitaremos luego, como ser el tamaño de un cluster o la cantidad de entradas del root.

Una vez hallados en el root podemos empezar a recorrerlo en busca del primer directorio contenido en el path. Para saber cuando nos hallamos en el path nos valemos del puntero al string pasado, éste lo vamos acortando a medida que entramos en los directorios pedidos, y cuando se encuentra apuntando a Null (o sea, cuando apunta a un cero) es cuando llegamos a nuestro objetivo. Es por esto que lo movemos una posición al entrar al root, tanto para dejarlo apuntando al nombre del primer directorio, como en el caso de que simplemente se haya pedido un lsdir del root, poder responder a este llamado ya que nos encontramos en el directorio correcto.

Para recorrer el root (y los subsecuentes subdirectorios) vamos a movernos de a 32 bytes por este, cada uno de estos 32 bytes corresponde a una entrada del mismo. Primero comprobamos que el primer byte de la entrada no indique una a la cual no debamos prestar atención (0x20, 0xe5), además de verificar que este mismo no indique que el root no posee más entradas (0x00), en cuyo caso enviamos un mensaje por pantalla diciendo que no se pudo encontrar el directorio especificado. Luego nos movemos al byte de atributo, en el cual verificamos que no sea un archivo (los cuales no nos interesan en esta función) ni que no sea una entrada de formato largo (0x0f).

Habiendo constatado que es un directorio, hacemos un llamado a la función comparestring, que dados dos string, uno que corresponde al path y otro que corresponde al directorio en el cual nos encontramos, nos dice si efectivamente los nombres coinciden. Para esto debemos sortear las diferencias que existen en el formato en el cual ambos string están presentados, ya que uno rellena los espacios con ceros hasta completar los 11 caracteres que conforman el directorio, mientras que el path puede contener un punto que no se encuentra en el parámetro proveniente de la imagen. Esta comparación se realiza byte a byte, con especial atención a los casos límite, y devuelve 0 en caso de que los directorios no tengan el mismo nombre y 1 en caso de que sí; en este caso además quita del path el directorio que acaba de encontrar para permitir que el ciclo se efectúe de vuelta de la misma manera.

En caso de que el directorio sea el buscado, debemos movernos hacia ese directorio para seguir buscando el path, esto lo hacemos mirando el cluster donde tal directorio comienza. Luego, debemos ir a ese cluster, esto lo hacemos con los datos que teníamos grabados en memoria, como el tamaño de un cluster y la cantidad de entradas del root; con esta cuenta podemos posicionarnos en el directorio buscado.

Al constatar que el path ya ha sido recorrido en su totalidad, lo único que falta es imprimir por pantalla la información del contenido del directorio, esto lo hacemos de manera similar a como recorrimos los directorios anteriormente, obviando los casos que no nos interesan y distinguiendo entre si la entrada es un directorio o un archivo para llamar en ambos casos a la función C printdir, pero con un valor diferente que usamos como bool para ayudarnos a imprimir la información de esta entrada.

Con la función printdir lo único que hacemos es imprimir el nombre del directorio o archivo, y agregar el tamaño de éste en caso de que sea un archivo. Para distinguir los directorios, utilizamos el formato de DOS, encerrándolo entre corchetes. Todo esto lo hacemos recursivamente para cada entrada hasta constatar que no existen más en el directorio, dando por terminada la función

3.4. Sizedir

Esta función se comporta de manera muy similar al lsdir, ya que realiza las mismas acciones que la anterior hasta encontrar el directorio especificiado por el path; es decir, una vez hallada en el boot se desplaza hasta el root guardando en memoria todo lo que consideramos necesario, una vez allí comienza a recorrer el root de a 32 bytes, llamando al comparestring en el caso de encontrar un directorio y entrando en este moviéndose hasta el cluster correspondiente en caso de que esta función devuelva 1. Una vez en el directorio pasado por parámetro, corresponde calcular el tamaño de todo lo contenido por éste, esto lo hacemos llamando a la función calcsize que calcula el tamaño de un directorio. Esto se mediante un contador que debe ser un registro preservado en una llamada a función (o sea, esi, edi o ebx) ya que al ser recursiva debemos asegurar que no se pierda en subsecuentes llamados.

La función calcsize recorre cada entrada, si es un archivo suma su tamaño al contador, si es un directorio se llama a si misma con el nuevo directorio como parámetro para calcular su tamaño, y luego de que esta llamada recursiva termina mueve suma el tamaño devuelto al contador para proseguir hasta que no encuentra más entradas en el directorio.

Una vez que terminan las llamadas recursivas de calcsize, nos queda en eax el tamaño total del directorio, así entonces llamamos a la función C printsize que simplemente imprime por pantalla el tamaño obtenido

3.5. Extraer

En esta función nos colocamos en el boot de la manera ya descripta, y recorremos el root y el path indicado de manera similar al lsdir, solamente que en este caso entramos al comparestring tanto si es un archivo como si es un directorio, ya que estamos buscando un archivo. Una vez encontrado este mismo al haber agotado el path (en caso de que este exista, caso contrario devuelve que no pudo encontrarlo), procedemos a extraerlo.

Para esto primero recurrimos a una simple función en C que nos transforma el nombre del archivo provisto por la imagen (ya que el pasado por el path lo fuimos descartando al buscar el archivo mismo) en otro formato al quitarle los espacios y agregarle el punto donde corresponde. Luego, con este string llamamos a un nuevo open para obtener un nuevo filedescriptor, es muy importante que el open lo llamemos con el flag O_CREAT y O_RODWR para poder crearlo en caso de que no exista el archivo ya extraido en el directorio donde se ejecuta el main, y para poder escribir en él.

Una vez obtenido el file descriptor, podemos empezar a escribir en el con la función write, para esto es necesario tener en cuenta que la imagen tiene al archivo partido en clusteres cuyo tamaño ya determinamos y guardamos en memoria, o sea que no vamos a poder copiar todo el archivo en una sola pasada a menos que este ocupe un sólo cluster. En caso contrario, debemos recurrir a la tabla de FAT. Para efectuar el extraído recurrimos a la función write, que recibiendo como parámetros el file descriptor, un puntero a memoria y la cantidad de bytes escribe esta cantidad en el primero; acá pedimos que copie el tamaño del cluster en bytes salvo que lo que quede por copiar del archivo sea menor a este número.

Para encontrar el siguiente cluster, posicionándonos en la tabla de FAT nos movemos hasta el cluster actual, y nos fijamos qué información tiene, si posee un valor entre 0xfff8 y 0xffff ése es el último cluster del archivo y damos por terminado el copiado, en caso contrario nos posicionamos en el cluster siguiente y llamamos de nuevo el write hasta conseguir los valores en la FAT que nos indiquen que no hay más por copiar. Una vez terminado el copiado podemos proceder a hacer el munmap y cerrar los file descriptors

4. Manual del Usuario

Para compilar el código y obtener el binario main, tan sólo basta con ejectuar en la consola el comando make, el Makefile provisto se encarga de compilar y linkear todos los sources.

A la hora de ejecutar se debe tipear en consola el comando "./main" seguido de un espacio y luego el cuál de las funciones se quiere ejectutar, a continuación sus parámetros.

Los parámetros posibles son

- ./main -v IMAGEN para ejecutar la función imprimir
- ./main -l DIRECTORIO IMAGEN para ejecutar la función lsdir
- ./main -s DIRECTORIO IMAGEN para ejecutar la función imprimir
- ./main -e ARCHIVO IMAGEN para ejecutar la función extraer

Es importante recalcar que los paths para los directorios deben estar todos escritos en mayúscula y deben comenzar y terminar con / Los nombres de archivos también deben estar en mayúscula, pero no deben terminar en / bajo ningún concepto
Por lo tanto, si quisiéramos observar los contenidos en la imagen im1.bin del directorio dir3, que se encuentra dentro del directorio dir2, el cual se encuentra a su vez dentro del directorio dir1 el cual está en el root, se debe tipear
./main -l /DIR1/DIR2/DIR3/ im1.bin

Si se quiere pasar como parámetro el root, simplemente se debe pasar /

5. Conclusiones

Este trabajo nos sirvió para entender mejor cómo funciona la programación en Assembler, cómo se debe encarar un problema a ser programado en dicho lenguaje; también nos ayudó a entender mejor la interacción entre C y Assembler, qué es lo que conviene hacer en cada lenguaje al hacer un programa. Durante el trabajo nos encontramos con innumerables errores de programación, la mayoría de estos por negligencia nuestra, esto nos dio paso a una mayor familiarización con las herramientas de debuggeo.

6. Correcciones

Las correcciones para el recuperatorio de este TP se enfocaron en resolver los problemas que no habíamos notado con nuestras imágenes de prueba y que aparecieron en los tests de la cátedra. La corrección más grande que tuvimos que hacer fue en el traversedir para todas las funciones, donde no habíamos tenido en cuenta que los directorios podían ocupar más de un cluster. Por lo tanto, antes el listar un directorio simplemente lo iba a cortar en el medio si este era más grande que un cluster, o si estaba buscándolo es probable que no lo encontrara. Esto fue solucionado guardándonos en un registro el valor de la tabla de FAT que correspondía al cluster actual, una vez terminado de recorrer el cluster simplemente debíamos chequear si ese valor apuntaba a 0xFFFF o si apuntaba a un nuevo cluster, en cuyo caso bastaba con repetir el ciclo sobre este mismo.

Este cambio significó tambien una necesidad de separar el recorrido de los directorios entre el root y cualquier otro, ya que el root no se divide en clusters, por ende tuvimos que adaptar las funciones a estas dos formas de recorrer diferentes. Esto a veces causó que la distancia de los jumps fuese más larga que lo que soporta la instrucción, por eso en algunas ocasiones tuvimos que implementar el jump near para remediar este problema.

Otro inconveniente que surgió en el testing fue que el comparestring no daba resultados correctos en algunos casos límite; como esta función ya era demasiado difícil de modificar puesto que había sido parcheada de apuro en múltiples ocasiones para corregir los errores que surgían, decidimos simplemente rehacerla, antes llamando a una función (redostring) que convertía el string del directorio en un formato idéntico al del path; con lo cual el comparestring ya no debía considerar casos límite como espacios y puntos sino simplemente comparar caracter por caracter.

El extraer, además del ya mencionado cambio al traversedir, sufrió otros cambios conforme surgían errores. En primer lugar, cambiamos la forma de pedir el parámetro, ya que no necesitamos más que el archivo termine en /, sino que simplemente usamos la función redopath para agregársela automáticamente (la / es necesaria para que el comparestring devuelva el resultado correcto). También descartamos la función createname, y la suplantamos por la función removeslash, ya que todavía conservábamos el nombre del archivo del path, lo único que necesitábamos era quitar la / que habíamos agregado para el comparestring.

Al testear el extraer nos dimos cuenta que no estábamos saltando bien de cluster en cluster, y que además la llamada al write estaba mal porque si el tamaño era mayor que 2048 (o sea, un cluster), pedíamos que escribiera todo el tamaño del archivo en lugar de sólo ese cluster, lo cual causaba diferencias en el archivo extraído.

La recorrección del TP pasa satisfactoriamente todos los casos de testeo provistos por la cátedra excepto cuando se pide el tamaño del root de la imagen 4, en cuyo caso devuelve un resultado que es aproximadamente 20000 bytes al resultado de la cátedra. Reconocemos el error en ese caso, pero no podemos encontrar dónde éste yace ya que la imagen posee demasiados directorios y se nos hace imposible hacer un debuggeo correcto. Todos los otros directorios pedidos por la cátedra de esa imagen y los roots de las otras imágenes obtienen resultados satisfactorios, con lo cual estamos seguros que el error no es demasiado severo. Los otros casos de testeo, para las cuatro diferentes funciones y para todas las imágenes obtienen resultados correctos. Para la imagen 6 nos concentramos no tanto en que devolviera el mensaje de error debido, ya que a veces esto era imposible, sino más que si era posible que no rompiera con ninguna violación de segmento. En los casos en que fue posible, se devuelve el error acontecido