Sistema de archivo MINIX 3

Objetivos

El objetivo básico del laboratorio es comprender y reconocer las estructuras que forman parte del sistema de archivo MINIX 3 utilizando los vaciados hexadecimales obtenidos con el comando de Linux *dd* (*disk dump*) y mostrados con el comando *hexdump* (*hexadecimal dump*).

Durante el laboratorio al alumno se le presentará una imagen de un sistema de archivo MINIX 3 y deberá brindar un informe acerca de las características particulares presenta este sistema de archivos.

Descripción

El usuario promedio tiene un conocimiento limitado acerca de cómo realmente se administra los datos guardados en su computadora. Es cierto que no es necesario que esté enterado, le basta con saber que sus datos están guardados en archivos y estos en carpetas, y tanto unos como otros pueden ser copiados, renombrados o borrados. Sin embargo para llevar a cabo estas operaciones administrativas se necesita de cierta “infraestructura”, entrecomillo esta palabra por que al final de cuentas lo que está en el disco son bytes y nada más que bytes, sin embargo si se establece por consenso algún esquema que se toma como cierto, entonces podemos darle una interpretación a esta secuencia de bytes. Alguien puede entonces afirmar que hay tantos esquemas como interpretaciones existan, de alguna forma esto es cierto, sin embargo los esquemas que han sido eficientes o muy populares por distintos motivos, son los que han perdurado. Cada esquema toma un nombre particular por ejemplo los sistemas de archivos FAT12, FAT16, FAT32, ext2, ext3, ext4, MINIX 3, NTFS, ISO9660, etc. Los sistemas operativos pueden soportar uno o más sistemas de archivos, esto depende de cómo están diseñados. Cada sistema de archivo tiene una estructura específica y las llamadas al sistema son escritas de acuerdo al sistema de archivo. Como el título de este laboratorio lo indica el sistema de archivo que nos ocupa es MINIX 3.

Requisitos

Para llevar a cabo este laboratorio con éxito necesita tener dominio sobre los siguientes temas:

Little and big endian

http://es.wikipedia.org/wiki/Endianness

Comando *hexdump*

Manual en línea (`man hexdump`)

Comando *dd*

Manual en línea (`man dd`)

http://en.wikipedia.org/wiki/Dd_(Unix)

Sistema de archivo MINIX 3

http://inform.pucp.edu.pe/~inf232/Semestre-2007-2/Laboratorio-1/index.htm

(a pesar de que no se pedirá programa alguno en el laboratorio, es bueno mirar los programas encontrados en esta página para conocer las estructuras del superbloque y del inodo)

Plan general del laboratorio

A continuación crearemos la imagen de un disquete conteniendo el sistema de archivo MINIX sin contenido. Analizaremos su estructura. Luego se le enseñará montarlo como un dispositivo de bloques para que pueda ser accedido desde Linux. La intención es que lo puedan modificar grabando en él archivos. A continuación vuelvan a ver las estructuras y comprueben las modificaciones.

Las siguientes secuencias de comandos deberán ser ejecutadas desde una terminal, lo que está en negrita es lo que se debe de escribir en la línea de comando, mientras que el resto es la salida (si hubiese).

Creando un archivo del tamaño de un disquete

```
dd if=/dev/zero of=disk.img bs=512 count=2880
2880+0 records in
2880+0 records out
1474560 bytes (1.5 MB) copied, 0.0216361 s, 68.2 MB/s
```

Creando un sistema de archivo MINIX 3 sobre *disk.img*

```
/sbin/mkfs -t minix disk.img
480 nodos-i
1440 bloques
Primera zona de datos=19 (19)
Tamaño de zona=1024
Tamaño máximo=268966912
```

En este punto ya tenemos un sistema de archivos vacío, y por supuesto puede ser visualizado con *hexdump*

```
hexdump -C disk.img

00000000  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000400  e0 01 a0 05 01 00 01 00  13 00 00 00 00 1c 08 10  |................|
00000410  8f 13 01 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000420  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000800  03 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000810  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000830  00 00 00 00 00 00 00 00  00 00 00 00 fe ff ff ff  |................|
00000840  ff ff ff ff ff ff ff ff  ff ff ff ff ff ff ff ff  |................|
*
00000c00  03 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000c10  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000cb0  00 c0 ff ff ff ff ff ff  ff ff ff ff ff ff ff ff  |................|
00000cc0  ff ff ff ff ff ff ff ff  ff ff ff ff ff ff ff ff  |................|
*
00001000  ed 41 e8 03 40 00 00 00  c2 8b f3 4d e8 02 13 00  |.A..@......M....|
00001010  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00004c00  01 00 2e 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00004c10  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00004c20  01 00 2e 2e 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00004c30  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00168000
```

La primera columna es la dirección en hexadecimal.

Los asteriscos indican que las líneas que deberían estar allí, tienen el mismo contenido.

El *dump* que se ha hecho es por carácter y no por palabra.

Las estructuras más significativas se muestran a continuación.

Estructura del superbloque:

```
typedef struct {
  ino_t s_ninodes;   /* # usable inodes on the minor device */
  zone1_t  s_nzones; /* total device size, including bit maps etc */
  short s_imap_blocks;      /* # of blocks used by inode bit map */
  short s_zmap_blocks;      /* # of blocks used by zone bit map */
  zone1_t s_firstdatazone;  /* number of first data zone */
  short s_log_zone_size;    /* log2 of blocks/zone */
  short s_pad;       /* try to avoid compiler-dependent padding */
  off_t s_max_size;  /* maximum file size on this device */
  zone_t s_zones;           /* number of zones (replaces s_nzones in V2) */
  short s_magic;            /* magic number to recognize super-blocks */
  /* The following items are valid on disk only for V3 and above */
  /* The block size in bytes. Minimum MIN_BLOCK SIZE. SECTOR_SIZE
   * multiple. If V1 or V2 filesystem, this should be
   * initialised to STATIC_BLOCK_SIZE. Maximum MAX_BLOCK_SIZE.
   */
  short s_pad2;             /* try to avoid compiler-dependent padding */
  unsigned short s_block_size;     /* block size in bytes. */
  char s_disk_version;      /* filesystem format sub-version */
} super_block;
```

Estructura del inodo

```
typedef struct {
  short i_mode;         /* file type, protection, etc. */
  short i_nlinks;       /* how many links to this file */
  short i_uid;          /* user id of the file's owner */
  short i_gid;          /* group number */
  off_t i_size;         /* current file size in bytes */
  time_t i_atime;       /* time of last access (V2 only) */
  time_t i_mtime;       /* when was file data last changed */
  time_t i_ctime;       /* when was inode itself changed (V2 only)*/
  zone_t i_zone[V2_NR_TZONES];/*zone numbers for direct,ind,and dbl ind */
} inode;
```

Estructura de la entrada del directorio

```
typedef struct {
       ino_t d_inodo
       char   d_name[60]
} entry_dir
```

A continuación se dan la definición de algunos tipos para conocer su longitud en bytes:

```
typedef unsigned long  ino_t;
typedef unsigned short zone1_t;
typedef long           off_t;
typedef unsigned long  zone_t;
typedef unsigned long  bit_t;
typedef long           time_t;
typedef unsigned long  ino_t;

#define V2_NR_TZONES  10
```

Recuerde que el tamaño en bytes por tipo es como sigue:

```
char                1 byte
short               2 bytes
int, long           4 bytes
```

Los mapas de bits se interpretan por carácter (1 byte) y se sigue la siguiente interpretación:

Tomando la línea en la dirección 0x800 (a 2KB del inicio del sistema de archivo):

```
00000800  03 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
```

En principio sabemos que esta línea corresponde a la zona del mapa bits de inodos, por que por teoría éste debería encontrarse en la segundo bloque, y como el bloque según este sistema de archivos es 1KB, entonces corresponde a un offset de 0x800.

Tomamos el primer byte y lo volcamos como una representación de sus 8 bits:

```
    0           3
0 0 0 0     0 0 1 1
```

El bit menos significativo, el que está más a la derecha representa al inodo 0, el siguiente bit más significativo representa al inodo 1, y así sucesivamente. En nuestro ejemplo se observa que los bits que representan a los inodos 0 y 1 están en 1, lo que significa que están ocupados. El resto de inodos están libres. Este primer byte nos permite representar hasta el inodo 7. El inodo 8 será representado por el bit menos significativo del siguiente byte. También es importante hacer notar que el inodo 0 no existe (motivo por el cual en el mapa de bits siempre está en 1), pero se considera para facilitar el desplazamiento. Una pregunta interesante es ¿a quién corresponde el inodo 1? La respuesta es muy sencilla corresponde al directorio raíz.

Para el mapa de zonas se hace una interpretación análoga, con la diferencia de que el primer bit corresponde a la zona P – 1, donde P es la primera zona de datos y cuyo número se encuentra en el superbloque.

Análisis del superbloque.

El superbloque se debe encontrar a 1KB del inicio del sistema de archivos, es decir a 0x400

```
00000400  e0 01 a0 05 01 00 01 00  13 00 00 00 00 1c 08 10  |................|
```

Si seguimos la definición de las estructuras, los primeros 4 bytes corresponden al número de inodos que tiene el sistema, según nuestro caso será:

`e0 01 a0 05` que interpretándolo como *little endian* tenemos `05 a0 01 e0` = 94372320

que por la información que se tiene cuando se creó el sistema de archivos (480 inodos), no corresponde a ese valor. Pero si en lugar de tomar 4 bytes tomamos 2, tenemos

`e0 01` que interpretándolo como *little endian* tenemos `01 e0` = 480

La conclusión es que al crear el sistema de archivo desde Linux, parece que no crea un sistema de archivo V3 sino el V2 de MINIX. La pregunta entonces es, ¿cómo obtener una imagen de un sistema de archivo V3 de MINIX?

Una forma sencilla es a través de una máquina virtual. Adjunto a este archivo se le proporciona la imagen de un disquete con sistema de archivo V3 de MINIX 3 creada desde KVM. A continuación parte del *dump* hexadecimal:

```
00000000  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000400  00 03 00 00 00 00 01 00  01 00 10 00 00 00 00 00  |................|
00000410  ff ff ff 7f 68 01 00 00  5a 4d 00 00 00 10 00 00  |....h...ZM......|
00000420  00 00 00 00 10 00 00 00  00 00 00 00 00 00 00 00  |................|
00000430  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00002000  07 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00002010  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00003000  ff 1f 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00003010  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00004000  ff 41 02 00 02 00 02 00  c0 00 00 00 ec 73 f3 4d  |.A...........s.M|
00004010  e7 73 f3 4d e7 73 f3 4d  10 00 00 00 00 00 00 00  |.s.M.s.M........|
00004020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00004040  a4 81 01 00 00 00 00 00  f8 96 00 00 97 ba f3 4d  |...............M|
00004050  e0 73 f3 4d e0 73 f3 4d  11 00 00 00 12 00 00 00  |.s.M.s.M........|
00004060  13 00 00 00 14 00 00 00  15 00 00 00 16 00 00 00  |................|
00004070  17 00 00 00 19 00 00 00  00 00 00 00 00 00 00 00  |................|
00004080  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
```

Ahora si tomamos los 4 primeros bytes del super bloque (offset = 0x400) tenemos:

`00 03 00 00` que interpretándolo como little endian tenemos `00 00 03 00` = 768

Es decir el número de inodos de este sistema de archivos es 768.

Como puede observar el mapa de bits de inodos y el mapa de bits de zonas se muestra de forma diferente. La razón es muy sencilla, en este sistema de archivo se ha colocado un archivo y obviamente ambos mapas se han modificado.

TAREA – PARTE 1

Para el sistema de archivos V3 de MINIX, contenido en *disk1.img*, (adjunto con este archivo) responda y lleve a cabo las siguientes tareas:

1.- ¿En qué dirección inicia el mapa de bits de inodos?
2.- ¿En qué dirección inicia el mapa de bits de zonas?
3.- ¿En qué dirección inicia la tabla de inodos?
4.- Interprete todos los campos del super bloque.
5.- ¿Cómo se llama el archivo contenido la raíz de esta imagen? ¿Cuántos bloques ocupa?
6.- Interprete los valores del inodo para el directorio raíz y para el archivo que lo contiene.

Por último es interesante ver las modificaciones que sufre el sistema de archivos cuando se copian o borran archivos en su sistema. Para esta segunda parte necesita tener instalado el paquete **udisks2**. En caso de que no lo tenga, puede instalarlo desde *synaptic* o por línea de comando.

Una forma de modificar el sistema de archivos es desde la máquina virtual que lo creó. Pero desde Linux tenemos un mecanismo para representarlo como un dispositivo de bloques y montarlo sobre un directorio para luego acceder a él a través del explorador de archivos o desde una terminal. Usaremos esta última alternativa.

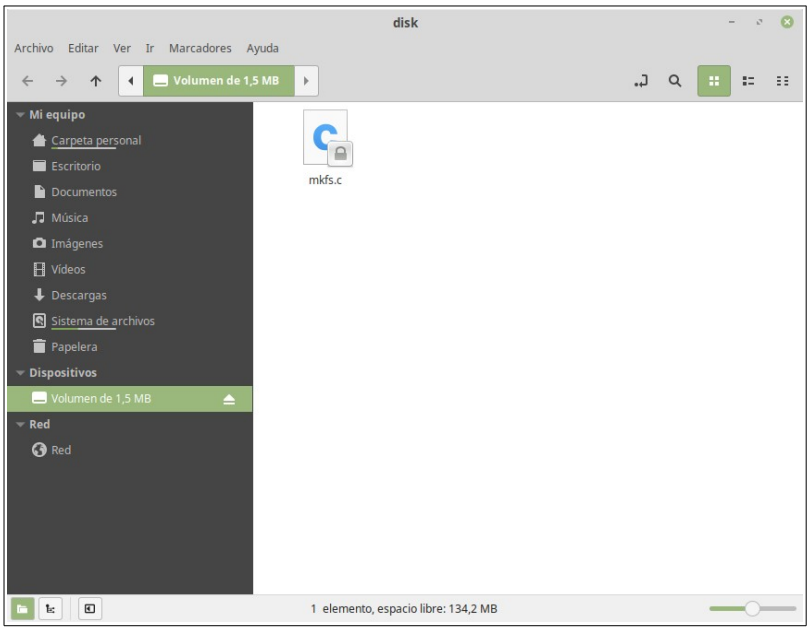
En el siguiente comando procederemos a **montar** nuestra imagen del disco en un directorio. Para ello es necesario dar a conocer la ruta absoluta donde se encuentra el archivo imagen. Asumiendo que se encuentra en */home/alulab/Descargas*, entonces

```
udisksctl loop-setup -f /home/alulab/Descargas/disk1.img
```

Como respuesta deberá aparecer en la terminal los siguiente:

```
Mapped file /home/alulab/Descargas/disk1.img as /dev/loop0.
```

También se abrirá (en Linux Mint 19) una ventana del navegador de archivos (Nemo) mostrando el directorio donde se ha montado la imagen.



Como resultado del comando anterior, el sistema ha creado un directorio con nombre ***disk*** (el nombre puede variar) en */media/alulab/* (asumiendo que el usuario es *alulab*) y **monta** el sistema de archivo sobre ese directorio. El directorio creado tiene como propietario al que ejecutó el comando (*alulab* en este caso), de forma que puede copiar y borrar archivos en esta imagen. Puede ver su contenido de forma acostumbrada:

```
ls -l /media/alulab/disk
```

En adelante cuando ingrese a este directorio, estará accediendo a este disco virtual, ahora usted puede grabar o borrar archivos contenidos en este directorio y lo que realmente estará haciendo es grabando y borrando archivos en el sistema de archivos V3 de MINIX. El objetivo es analizar como varía el sistema de archivo una vez que está en uso.

Cuando ya no se desee acceder más a esta imagen se debe **desmontarlo** para que los cambios tomen efecto.

```
udisksctl unmount -b /dev/loop0
```

TAREA – PARTE 2

1.- Agregue archivos y vea como se modifican los mapas de bits de inodos y los de zonas.

2.- Borre archivos y vea cómo se modifican los mapas de bits de inodos y de zonas. Además vea cómo queda la entrada de directorio de un archivo borrado.

3.- ¿Es posible recuperar un archivo borrado? ¿Qué condiciones son necesarias?

4.- Busque una forma de extraer un archivo (sin montarlo) desde la imagen de disco, sabiendo solamente el tamaño y dónde está ubicado (Sugerencia: emplee el comando *dd*).

Prof: Alejandro T. Bello Ruiz