

BRFS

BOOT-ROOT Filesystem

Versión 0.2

Contenido

Contenido	ii
1 Introducción	3
1.1 Sobre BRFS	3
1.2 Definiciones	3
1.3 Ventajas y desventajas	3
2 Estructura y funcionamiento	5
2.1 Punteros	5
2.2 Directorios	5
2.3 Directorio raíz	6
2.4 Organización en disco	6
3 Subformato	9
3.1 Etiquetado de índices (Index labeling)	9
3.2 Tamaño de punteros	9
3.3 Tamaño de atributos	9
3.4 BPB y sector de arranque	9
3.5 Nomenclatura	10
4 Límite teórico	11
4.1 Sector máximo	11
4.2 Punto de inflexión	12

1.1 Sobre BRFS

El sistema de archivos BRFS se basa en la lectura y escritura del disco por sectores. Está pensado para implementar en sistemas operativos que no necesitan un complicado sistema de archivos ni tampoco manejar grandes tamaños. Sin embargo, es posible que BRFS se pueda extender tanto que resulte un sistema de archivos competente.

1.2 Definiciones

A lo largo de este documento nos referiremos a determinadas palabras clave sobre BRFS con una fuente ancha.

Palabras clave:

- **Elemento** : Se refiere tanto a ficheros, como a directorios.
- **Subformato** : Configuración del disco que define varios datos que podrían variar entre sistemas (punteros, metadatos, etc).

1.3 Ventajas y desventajas

Ningún sistema es perfecto, y por supuesto BRFS tampoco. Pero intenta ser lo mejor posible, no respecto a otros sistemas de archivos; sino respecto a sí mismo, buscando un sistema de archivos que cubra todas las necesidades fácilmente de forma simple.

Ventajas	Desventajas
Tamaño máximo de los elementos indeterminado.	Aunque no ha sido sometido a pruebas de velocidad, es probable que la alta fragmentación de los elementos en el disco provoque una ligera caída de velocidad, aunque no se ha notado retraso al leer archivos muy fragmentados.
Optimización del espacio en el disco, debido a la alta fragmentación de los elementos.	
Personalización del Subformato en función de las necesidades del sistema. Y fomentando la universalidad definiendo los parámetros del	El tamaño máximo del disco está determinado por el tamaño de los punteros: <ul style="list-style-type: none">• 16 bits : ~ 32 KiB

subformato en el sector de arranque, para que otro sistema los interprete adecuadamente.	<ul style="list-style-type: none"> • 32 bits : ~ 1 TiB • 64 bits : ~ 8 ZiB • ... Ver: 4 Límite teórico
No necesita clústers, ni tablas de archivos.	

2 *Estructura y funcionamiento*

2.1 Punteros

Pueden ser punteros finales ó punteros regulares.

Los elementos están divididos en sectores (512 bytes), pero los últimos bytes de cada uno representan un puntero al siguiente sector que comprende al propio elemento.

El tamaño de los datos del elemento en cada sector se ve reducido en función del tamaño del puntero.

Por ejemplo:

- Con un puntero de 16 bits (2 bytes), los datos de un archivo de 512 bytes, no ocupa 1 sector, sino que ocupa 510 bytes en un sector y los 2 bytes que faltan en otro sector, porque los últimos 2 bytes de cada sector funcionan como puntero a otro sector.
- Con un puntero de 32 bits (4 bytes), los datos de un archivo de 512 bytes, no ocupa 1 sector, sino que ocupa 508 bytes en un sector y 4 bytes en otro sector.
- ...

Todos punteros deben usar el formato [LBA \(Logical Block Addressing\)](#), y no otro.

Si un puntero final apunta a 0x01, no significa que el siguiente sector sea el 0x01, sino que el siguiente sector a leer es el posterior.

2.2 Directorios

Un directorio es un elemento que almacena una lista de los elementos que contiene, siguiendo el siguiente formato:

identificador - tipo (1 byte) - puntero - atributos

- Identificador Es una cadena de texto que define el nombre del elemento.
- Tipo Define si el elemento es un archivo o una carpeta. Puede ser **0x1c** (archivo)[*File separator*] ó **0x1d** (carpeta)[*Group separator*].

- **Puntero** Apunta al primer sector que contiene los datos del elemento. El tamaño de los punteros está definido en el subformato. Este es un puntero regular, no un puntero final.
- **Atributos** Contiene información sobre los atributos del elemento, distribuidos en bits. El tamaño de los atributos está definido en el subformato.

2.3 Directorio raíz

Se llama BOOT-ROOT al sistema de archivos porque el directorio raíz (ROOT) sólo puede estar instantáneamente después del sector de arranque (BOOT). Así, independientemente de que el disco contenga el sistema operativo o cualquier otra estructura de datos, se puede saber con certeza dónde está la raíz y establecer el directorio actual en ese sector.

La carpeta raíz (o directorio raíz) puede tener un tamaño inicial indeterminado, pero se recomienda no almacenar muchos elementos en este porque se fragmentará muy lejos y es posible que ello retrase la lectura y/o escritura del disco.

2.4 Organización en disco

En el siguiente ejemplo veremos la distribución en disco del siguiente árbol:

```

ROOT
├── a.txt
├── FOLDER
└── b.txt

```

Con el siguiente Subformato:

- Etiquetado de los índices: ASCII.
- Tamaño de los punteros: 16 bits.
- Tamaño de los atributos: 0 bytes.

0x0000 →	BOOT
0x0001 →	ROOT

0x0003 →	a.txt
0x0004 →	FOLDER
0x0005 →	*a.txt
0x0006 →	b.txt
	⋮

[illegible][illegible]


7

Buenos días.

- 0x01 means the next sector.

-

Buenas noches.



Good Morning

3 *Subformato*

3.1 Etiquetado de índices (*Index labeling*)

Se refiere a la codificación de caracteres que se usa en los `identificadores` las entradas de los Directorios. Siempre debe ser distinto de 0.

Generalmente se usa este estándar:

- 0x01 [ASCII](#)
- 0x02 [UTF-8](#)

3.2 Tamaño de punteros

Indica cuántos bytes ocupan los punteros, tanto los `finales` como los `regulares`.

3.3 Tamaño de atributos

Indica cuántos bytes ocupan los atributos, que siempre aparecerán a continuación de los punteros `regulares`.

Cada atributo se representa en 1 bit, por esto los atributos no suelen ocupar más de 1 byte (8 bits). Con 1 byte muchas veces incluso sobran atributos.

3.4 BPB y sector de arranque

El [BPB \(Bios Parameter Block\)](#), es una estructura de datos en el sector de arranque de un disco, que describe los parámetros físicos del mismo. Lo primero que debe aparecer es una instrucción **JMP** y un **NOP**. El BPB ocupa 33 bytes, el EBPB (Extended BPB) ocupa los siguientes 26 bytes.

En BRFS se utiliza un espacio extra a continuación del EBPB y tan sólo ocupa 3 bytes, define el etiquetado de índices, el tamaño de los punteros, y el tamaño de los atributos, respectivamente.

Ejemplo (en [NASM](#)) de un sector de arranque simple con el BPB arreglado para BRFS:

```
org 0x7c00
bits 16
```

```

BPB:
    jmp START
    nop

    db 'FreeBAS '           ; OEM, 8 bytes
    dw 512                   ; bytes per sector
    db 1                     ; sectors per cluster
    dw 1                     ; reserved sector count
    db 0                     ; number of FATs
    dw 0                     ; root directory entries
    dw 2880                  ; total sectors in drive
    db 0xF0                  ; media byte
    dw 0                     ; sectors per fat
    dw 18                    ; sectors per track
    dw 2                     ; number of heads
    dd 0                     ; hidden sector count
    dd 0                     ; number of sectors huge
EBPB: ; -----
    db 0                     ; drive number
    db 0                     ; reserved
    db 0x69                  ; signature, maybe 0 or 29 or 28
    dd 0                     ; volume ID
    db 'NO LABEL '          ; volume label, 11 bytes
    db 'BRFS '              ; file system type, 8 bytes
BRFS: ; -----
    db 0x01 ; LABELING           ; 00: Not BRFS, 01: ASCII, 02: UTF-8
    db 0x02 ; Pointers size (in bytes)
    db 0x00 ; Attributes size (in bytes)

START:

    jmp $ ; HANG

times 510-($-$$) db 0 ; Fill with NUL
db 0x55,0xaa ; Boot signature

```

3.5 Nomenclatura

BRFS-16a significa puntero de 16 bits y etiquetado de índices ASCII. **BRFS-32u1** significa punteros de 32 bits, etiquetado UTF-8 y 1 byte de atributos. Si el tamaño de los atributos es 0, no se especifica. **BRFS-64a2** significa puntero de 64 bits, etiquetado ASCII y 2 bytes de atributos.

4 *Límite teórico*

4.1 Sector máximo

Como vimos antes, el tamaño máximo de un disco al que se puede acceder con BRFS está determinado por el tamaño de los punteros.

Ejemplo: si el tamaño de los punteros es 16 bits, el sector máximo que se puede representar con 2 bytes es 0xFFFF.

16 bits:

0xFFFF → 65.535	Sólo se puede acceder a 65.535 sectores con 16 bits.
$65.535 * 512 = \mathbf{33.553.920 \text{ bytes}}$	En bytes: 33.553.920
$33.553.920 / 1024 = 32.767,5 \text{ KiB} / 1024 = \mathbf{31,9995117188 \text{ MiB}}$	Aproximadamente: 32 MiB

Esto significa que con un puntero de 16 bits sólo podemos acceder a 32 MiB ($\approx 33.5 \text{ MB}$) de disco, aunque el tamaño del volumen sea 4 GB, 8 GB, 1 TB, ... Sólo se podrá acceder a 32 MiB, inutilizando el resto del disco.

La parte buena de este problema, es que el tamaño máximo crece exponencialmente y con 64 bits ya cubrimos un tamaño de disco considerable.

32 bits:

0xFFFFFFFF → 4.294.967.295	Sector máximo: 4.294.967.295
$4.294.967.295 * 512 = \mathbf{2.199.023.255.040 \text{ bytes}}$	Tamaño máximo: 2.199.023.255.040 Bytes
$2.199.023.255.040 / 1024 / 1024 = \mathbf{2.097.151,99951171875 \text{ MiB}}$	Tamaño máximo: $\sim \mathbf{2.097.152 \text{ MiB}}$
$2.097.151,99951171875 / 1024 = \mathbf{2.047,99999952316284179687 \text{ GiB} / 1024 = \mathbf{1.99999999953433871269 \text{ TiB}}$	Tamaño máximo: $\sim \mathbf{2 \text{ TiB}}$

Con 32 bits ya se puede alcanzar la cifra de 2 Tebibytes [**TiB**] ($\approx 2,20 \text{ Terabytes [TB]}$).

64 bits:

0xFFFFFFFFFFFFFFFF → 18.446.744.073.709.551.615	Sector máximo: 18.446.744.073.709.551.615
18.446.744.073.709.551.615 * 512 = 9.444.732.965.739.290.426.880 bytes	Tamaño máximo: 9.444.732.965.739.290.426.880 Bytes
9.444.732.965.739.290.426.880 / 1024 / 1024 = 9.007.199.254.740.991,99951171875 MiB	Tamaño máximo: 9.007.199.254.740.991,99951171875 MiB
9.007.199.254.740.991,99951171875 / 1024 / 1024 = 8.589.934.591,99999999953433871269 TiB / 1024 = 8,388,607.999999999954525264 PiB	Tamaño máximo: ~ 8,388,608 PiB
8,388,607.99999999999954525264 / 1024 = 8,191.999999999999955591 Exbibytes / 1024 = 7.99999999999999956 Zebibytes	Tamaño máximo: ~ 8 ZiB

Con 64 bits se alcanza la inimaginable cifra de 8 Zebibytes [**ZiB**] ($\approx 9,4$ Zettabytes [**ZB**]).

Para hacerse una idea (<http://highscalability.com/blog/2012/9/11/how-big-is-a-petabyte-exabyte-zettabyte-or-a-yottabyte.html>):

- *2 Petabytes*: Todas las bibliotecas de investigación académica de EE. UU.
- *5 Exabytes*: Todas las palabras dichas por seres humanos.
- Una investigación de la Universidad de California en San Diego informa que en 2008, los estadounidenses consumieron **3,6 zettabytes** de información.

4.2 Punto de inflexión

Como ya hemos aclarado antes, los datos que se pueden almacenar sobre un archivo en cada sector se ve reducido por el tamaño del puntero. ¿Pero qué sucedería si el puntero fuera demasiado grande?

Supongamos que tenemos un archivo que simplemente contiene la palabra “Hola”, ocupando 4 bytes. Podemos utilizar un puntero de 16 bits (o 32, o incluso 64) y el archivo sólo ocuparía un único sector. Pero...

¿Y si el puntero ocupase 511 bytes? Entonces tendríamos un problema, porque el archivo sólo dispondría de **1 byte** por cada sector, de modo que como el archivo ocupa 4 bytes, estaría repartido entre **4 sectores**.

El problema del punto de inflexión es que debe haber un tamaño de puntero determinado para el cuál, ya no merece la pena utilizar BRFS en lugar de otro sistema de archivos. Pero no debería ser un problema dado que con 32 bits de puntero ya se cubre un tamaño de disco que la mayoría de nosotros no vamos a ver.

Probablemente nunca se alcanzará un tamaño de puntero que ocupe la mitad del sector.