Resolución - Guía de ejercicios

1.1 Montaje de un sistema de archivos.

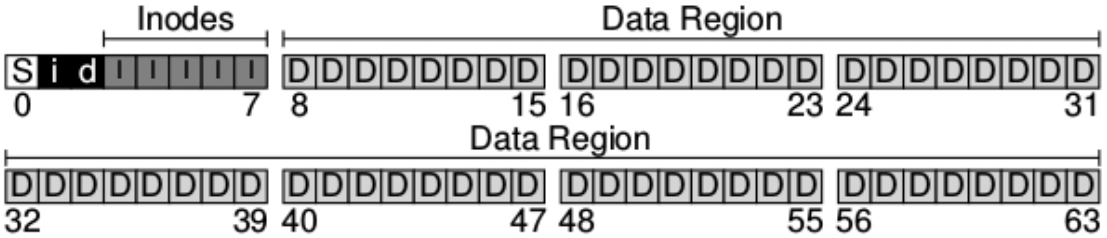
Sea un disco que posee 512 bloques de 4kb y un sistema operativo cuyos inodos son de 256 bytes, definir un sistema de archivos FFS.

★ Justifique en base a que, se toman todas las decisiones para el armado del FFS.

Nota: en la clase de consultas, se dijo que el FFS se refería al VSFS, por los datos que se dan. Así que lo resolví con esa lógica.

Para definir un VSFS, necesito:

- 1 superbloque (con la información del VSFS)
- 1 bitmap de bloques de datos
- 1 bitmap de bloques de inodos
- Bloques de datos
- Bloques de inodos



Lo que hay que hacer es calcular la cantidad de bloques ocupados por datos y los bloques ocupados por inodos.

Como cada inodo ocupa 256 bytes, si cada bloque tiene 4 KB, voy a tener 4096/256 = 16 inodos por bloque.

El tamaño total en bytes de los bloques es 512 x 4096 = 2097152 = 2^21.

Tomamos que un inodo cubre hasta 4 KB, así que para el tamaño total del disco, voy a necesitar 2^21/2^12 (4096) = 2^9 = 512 inodos.

¿Cuántos bloques necesito para almacenar 512 inodos?

512/16 = 32 bloques de inodos

Entonces, de datos voy a tener 512 - 32 - 3 = 477 bloques.

Finalmente:

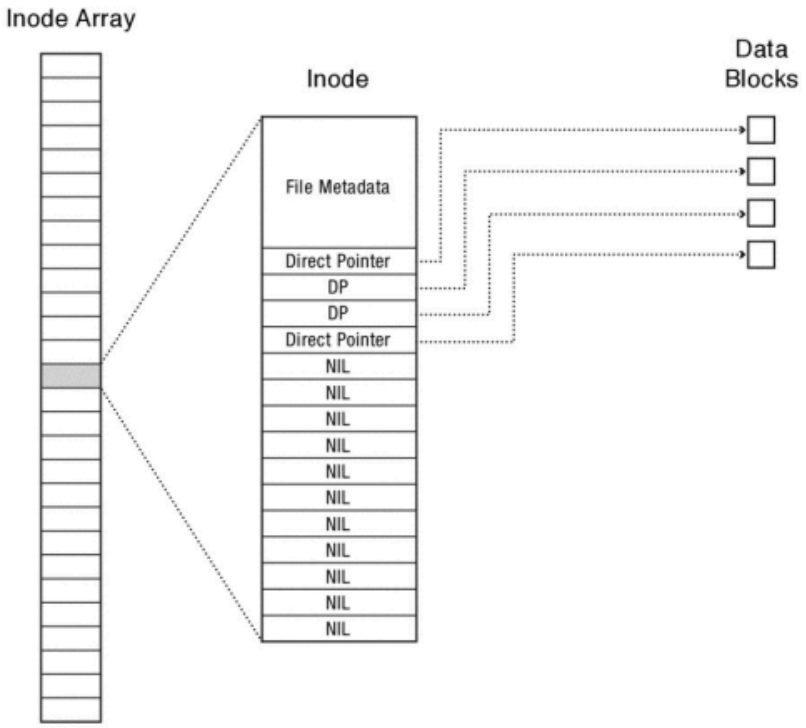
- 1 superbloque
- 1 bitmap de datos
- 1 bitmap de inodos
- 32 bloques de inodos
- 477 bloques de datos

1.2 Capacidad máxima de un inodo.

Suponiendo que BigFS es una variante de FSS, el clásico sistema de archivos de unix, el cual posee 12 referencias inodos directas, 1 indirecta, 1 doble indirecta, una triple indirecta y una cuádruple indirecta. Asumiendo bloques de 4 Kb, 8 byte por puntero a bloques.

★ ¿Cuál es el máximo tamaño de archivo que se soporta? Expresar en Terabytes.

◆ Dibujar la distribución de los punteros y los bloques de punteros.



Cantidad de punteros en total:

4096 bytes/8 bytes/puntero = 512 punteros

REFERENCIAS DIRECTAS: apuntan directamente a un bloque de datos.

Tamaño direccionable: 12 * 4 KB = 48 KB

REFERENCIAS INDIRECTAS: apuntan a un bloque de punteros, donde cada puntero apunta a un bloque de datos.

512 punteros * 4 KB = 2048 KB

REFERENCIA DOBLE INDIRECTA:

bloque punteros -> bloque punteros -> bloque de datos

512 * 512 * 4 KB = 1.048.576 KB

REFERENCIA TRIPLE INDIRECTA:

bloque punteros -> bloque punteros -> bloque punteros -> bloque de datos

512 * 512 * 512 * 4 KB = 536.870.912 KB

REFERENCIA CUÁDRUPLE INDIRECTA:

bloque punteros -> bloque punteros -> bloque punteros -> bloque punteros -> bloque de datos

512 * 512 * 512 * 512 * 4 KB = 274.877.906.944 KB

Ahora, sumo todos los tamaños: 275.415.828.528 KB
Paso a Tera bytes (divido por 1024 = 2^10 3 veces):

274.877.906.944/1024 = 268.960.770,046875 MB

268.960.770,046875/1024 = 262.657,0019989013671875 GB

262.657,0019989013671875/1024 = **256,5 TB (redondeando)**

1.3 Accesos a inodos y datablocks.

Partiendo de un filesystem vacío se ejecutan una serie de comandos en la shell en el órden indicado.

Para los comandos en **negrita**, indicar la cantidad de accesos a inodos y accesos a bloques de datos que el sistema operativo debe realizar.

Notas:

- Asuma que no existe ningún mecanismo de caché ú optimización, y el sistema operativo siempre tiene que acceder a todos los inodos y bloques de datos necesarios en cada comando.

```
# mkdir /dir    /dir/s
# echo 'hola' > /dir/x
# echo 'mundo' > /dir/s/y
#
# ls /dir/x     Inodos: ---    Blq. datos: ---
# cat /dir/s/y      Inodos: ---    Blq. datos: ---
#
# touch /dir/h /dir/y
# ln /dir/x /dir/h
# ln -s /dir/s/y /dir/y
# rm /dir/x


#
# cat /dir/h    Inodos: ---    Blq. datos: ---
# cat /dir/y    Inodos: ---    Blq. datos: ---
```

Primera parte: comandos ls /dir/x y cat /dir/s/y

ls /dir/x - Accesos a inodos y bloques

inodo /
bloque /
inodo dir
bloque dir
inodo x (ls solo accede al inodo)

Resultado: 3 inodos - 2 bloques

cat /dir/s/y - Accesos a inodos y bloques

inodo /
bloque /
inodo dir
bloque dir
inodo s
bloque s
inodo y
bloque y (cat accede al bloque de datos)

Resultado: 4 inodos - 4 bloques

Segunda parte: comandos cat /dir/h y cat /dir/y

cat /dir/h - Accesos a inodos y bloques

Observación: **hard link** entre /dir/x y /dir/h, es decir, apuntan al mismo inodo, por lo tanto, si se eliminara alguno de los dos, si accedemos al que sigue en pie podemos acceder. En este caso, /dir/x se removió.

inodo /
bloque /
inodo dir
bloque dir
inodo h
bloque h (cat accede al bloque de datos)

Resultado: 3 inodos - 3 bloques

cat /dir/y - Accesos a inodos y bloques

Observación: **soft link** entre /dir/s/y y /dir/y, esto significa que /dir/y apunta a /dir/s/y, por lo tanto, si accedemos al bloque de datos de /dir/y, obtendremos el path de /dir/s/y.

inodo /
bloque /
inodo dir
bloque dir
inodo y
bloque y (accedemos a /dir/s/y)
inodo /
bloque /
inodo dir
bloque dir
inodo s
bloque s
inodo y
bloque y

Resultado: 7 inodos - 7 bloques

¿Cuándo necesito un soft link y cuándo un hard link?

Característica	Hard link	Soft Link (Symlink)
El inodo apunta a:	Los datos del archivo	Un bloque con la ruta del archivo
Relacion con el archivo	Mismo inodo	Archivo separado
Funcionamiento tras borrar el destino	Sigue funcionando	Se rompe
Directorios	No se permite	Si se permite
Diferentes particiones	No se permite	Si se permite
Tamaño	No ocupa espacio extra	Ocupa como un archivo pequeño

1.4 Idem anterior (variante de parcial).

a. El superbloque de un sistema de archivos indica que el (3) inodo correspondiente al directorio raíz es el #43. En la siguiente secuencia de comandos, y siempre partiendo de ese directorio raíz, se pide indicar la cantidad de inodos y bloques de datos a los que se precisa acceder (leer) para resolver la ruta dada a `cat(1)` o `stat(1)`.

```
# mkdir /dir    /dir/s    /dir/s/w
# touch /dir/x /dir/s/y
#
# stat /dir/s/w/x     Inodos: ---    Blq. datos: ---
# stat /dir/s/y       Inodos: ---    Blq. datos: ---
#
# touch /dir/y
# ln /dir/s/x /dir/h
# ln -s /dir/s/y /dir/y
#
# cat /dir/h    Inodos: ---    Blq. datos: ---
# cat /dir/y    Inodos: ---    Blq. datos: ---
```

Ayuda: Todos los directorios ocupan un bloque. La idea es que describan como `stat` llega a los archivos.

★ Justifique los pasos que el sistema operativo necesita hacer para completar cada comando, y que resulten en la cantidad de accesos que previamente ha indicado.

◆ Dibuje como queda el file system en cada caso.

Primera parte: comandos stat /dir/s/w/x y stat /dir/s/y

stat: solo accede al inodo.

stat /dir/s/w/x - Accesos a inodos y bloques

inodo /
bloque /
inodo dir
bloque dir
inodo s
bloque s
inodo w
bloque w
cuando intenta acceder al inodo de x tirará error, porque x en esa ubicación no existe, así que ahí termina.

Resultado: 4 inodos - 4 bloques

stat /dir/s/y - Accesos a inodos y bloques

inodo /
bloque /
inodo dir
bloque dir
inodo s
bloque s
inodo y

Resultado: 4 inodos - 3 bloques

Segunda parte: comandos cat /dir/h y cat /dir/y. Misma lógica que la segunda parte del ej. anterior, un hard link y un soft link.

Observación:

En **ln /dir/s/x /dir/h**, ni el archivo /dir/s/x ni el /dir/h existe, por lo tanto la creación del hard link devolverá un error. Pero el recorrido del comando cat en /dir/h se realizará de todas formas, hasta llegar al h que no existe.

cat /dir/h - Accesos a inodos y bloques

inodo /
bloque /

inodo dir
bloque dir
cuando intente acceder al inodo de h, no lo encontrará y dará error.

Resultado: 2 inodos - 2 bloques

cat /dir/y - Accesos a inodos y bloques

Observación: en duda si el archivo se crea con touch, si hago un soft link éste se crea correctamente.

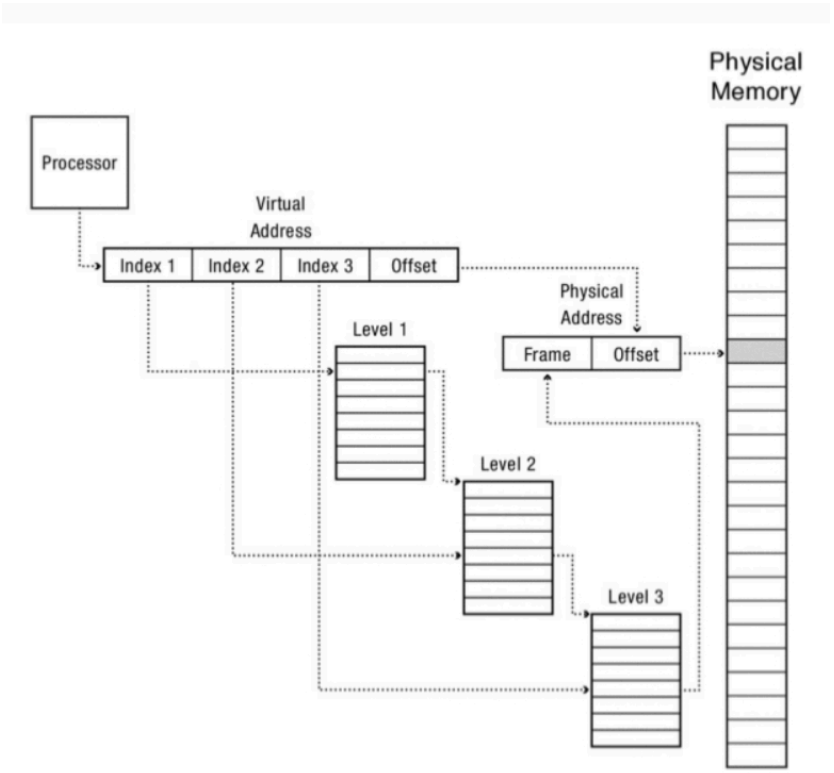
2.1 Cantidad de direcciones.

Explicar el mecanismo de address translation memoria virtual paginada de tres niveles de indirección de 32 bits. Indique la cantidad de direcciones de memoria que provee, una virtual address:

7 bits	7 bits	6 bits	12 bits

◆ Dibuje la distribución de las páginas y como se conectan:

- Page Directory.
- Page Table #1.
- Page Table #2.
- Page Frame.



La virtual address tiene:

7 bits	7 bits	6 bits	12 bits

La primera posición es para el Page Directory (Level #1), las 2 siguientes son para las Page Tables (Level #2 y #3), y la última es el OFFSET.

Cada nivel tiene la capacidad de 2^cantidad de bits. Entonces

Capacidad Level #1 = 2^7
Capacidad Level #2 = 2^7
Capacidad Level #1 = 2^6

Y además, por el OFFSET tenemos 2^12 combinaciones más.

Finalmente, la cantidad de memoria será

2^7 * 2^7 * 2^6 * 2^12 = **2^32 direcciones posibles**

2.2 Capacidad de memoria en Kbytes.

Cual es la cantidad de Kbytes que se pueden almacenar en un esquema de memoria virtual de 48 bits con 4 niveles de indirección, en la cuál una dirección de memoria se describe como sigue: 9 bits para el directorio de página, 9 bits para cada tabla de páginas y 12 bits para el offset.

★ Desarrolle las cuentas que se deben realizar para encontrar la cantidad de Kbytes que se pueden almacenar.

◆ ¿Cuánta memoria es en Gbytes? ¿Y en Tbytes?.

9 bits	9 bits	9 bits	9 bits	12 bits

Primeros 9 bits para Page Directory, los siguientes 3 para las Page Tables y por último los 12 bits del OFFSET.

La cantidad de memoria será:

(2^9)^4 * 2^12 = 2^48 bytes

Paso a KB, MB, GB y TB dividiendo por 2^10 = 1024.

2^48/2^10 = 2^38 KB

2^38/2^10 = 2^28 MB
2^28/2^10 = 2^18 GB
2^18/2^10 = 2^8 TB

2.3 Traducción de direcciones.

Suponga a las virtuales address con las siguientes características:

- 4 bit para el segment number.
- 12 bit para el page number.
- 16 bit para el offset.

Segment table	Page Table A	Page Table B
0 Page B	0 ZASPEH	0 F000
1 Page A	1 DEAD	1 D8BF
X Invalid	2 BEEF	2 3333
	3 BA11	X Invalid

Traducir las siguientes direcciones virtuales a físicas: [0000000], [20022002], [10222002], [00015555]

★ Desarrolle como se realizan las traducciones y cual número de cada dirección indica las distintas tablas.

◆ Dibujar como se acceden a las distintas tablas y páginas.

Segment Number	Page Number	OFFSET
0	000	0000

Segment = 0 -> Page B
Page number = 0 -> FRAME = F000

Physical Address

FRAME	OFFSET
F000	0000

Segment Number	Page Number	OFFSET
2	002	2002

Segment = 2 -> Invalid!

Segment Number	Page Number	OFFSET
1	022	2002

Segment = 1 -> Page A
Page number = 022 -> Invalid!

Segment Number	Page Number	OFFSET
0	001	5555

Segment = 0 -> Page B
Page number = 1 -> FRAME = DEAD

Physical Address

FRAME	OFFSET
DEAD	5555

2.4 Accesos a memoria.

Considere un sistema x86 de memoria virtual paginada de dos niveles con un espacio de direcciones de 32 bits, donde cada página tiene un tamaño de 4096 bytes. Un entero ocupa 4 bytes y se tiene un array de 50,000 enteros que comienza en la dirección virtual 0x01FBD000.

El arreglo se recorre completamente, accediendo a cada elemento una vez. En este proceso, **¿a cuántas páginas distintas** (no la cantidad total de accesos) necesita acceder el sistema operativo para conseguir esto? Recuerde contar las tablas de páginas intermedias, no solo las páginas que contienen los elementos del array.

★ Desarrolle como se realizan los accesos y cuantos enteros se ocupan por página.

◆ dibuje como se realizan los accesos a las distintas páginas. (Realizar un dibujo conceptual que muestre como se usa la memoria).

Cada página tiene 4096 = 2^12 bytes. Tomamos que cada dirección de memoria apunta a un byte individual.

¿Cuántos enteros entran en 1 página?

4096/4 = 1024 int por página

Si son 50K enteros y sé que entran 1024 por página, ¿cuántas páginas físicas necesito?

50.000/1024 = 48.82 = 49 páginas físicas (redondeamos)

¿Cómo puedo saber dónde se ubica la Page #1?

Si el tamaño del OFFSET es según los bytes de la página:

4096 = 2^12 -> 12 bits para el OFFSET

Como las direcciones son de 32 bits, me quedan 20 bits que reparto equitativamente entre el Page Directory y la Page Table.

Page Directory	Page Table	OFFSET
10 bits	10 bits	12 bits

La dirección de inicio del array de enteros es 0x01FBD0000, convierto a binario:

0000	0001	1111	1011	1101	0000	0000	0000

Separo en 10, 10, 12

Page Directory	Page Table	OFFSET
0000000111	1110111101	000000000000
7	957	

Voy desde el 957 al 957 + 49 - 1 = 1005.

Recapitulando: ¿cuántas páginas necesito?

- Hay una sola página Page Directory (1)
- Como la Page Table puede contener desde 0 a 2^10 direcciones diferentes (0 a 1023) y comienzo en la 957 y voy hasta la 1005, es decir, entra todo en 1 sola, tengo una página más (1).
- Las 49 páginas físicas que ocupa el array de enteros.

En total 49 + 1 + 1 = 51 páginas en total.

2.5 Idem anterior (variante de parcial).

Dado un espacio de direcciones virtuales con direcciones de 8 bits y páginas de 16 bytes, asume un array de 12 enteros (cada uno de 4 bytes) comenzando en la dirección virtual 100. Calcula el patrón de aciertos y fallos en la TLB cuando se accede a todos los elementos del array en un bucle. Asume que inicialmente, la TLB está vacía.

Pasos:

1. Determina el VPN y el desplazamiento para cada elemento del array.
2. Identifica si ocurre un acierto o un fallo en la TLB para cada acceso.

★ Desarrolle como se realizan los accesos y cuantos enteros se ocupan por página.
◆ dibuje como se realizan los accesos a las distintas páginas. (Realizar un dibujo conceptual que muestre como se usa la memoria).

Hay que saber cómo se distribuyen los bits en las direcciones de 8 bits entre VPN y OFFSET.

Como las páginas tienen 16 bytes = 2^4 -> 4 bits de OFFSET

Como las direcciones tienen 8 bits, 8 - 4 (bits del OFFSET) = 4 bits de VPN.

VPN	OFFSET
4 bits	4 bits

Índice	Dirección	VPN - OFFSET	HIT/MISS
A[0]	100	0110 0100	MISS
A[1]	104	0110 1000	HIT
A[2]	108	0110 1100	HIT
A[3]	112	0111 0000	MISS
A[4]	116	0111 0100	HIT
A[5]	120	0111 1000	HIT
A[6]	124	0111 1100	HIT
A[7]	128	1000 0000	MISS
A[8]	132	1000 0100	HIT

A[9]	136	1000 1000	HIT
A[10]	140	1000 1100	HIT
A[11]	144	1001 0000	MISS

Importante: el primero es un MISS porque al inicio la TLB está vacía, entonces debe realizar la traducción completa.

3.1 Ejecución de un proceso MLFQ.

Sea 1 proceso, cuyo tiempo de ejecución total es de 40 ms, el *time slice* por cola es de 2 ms/c pero el mismo se incrementa en 5 ms por cola.

★ ¿Cuántas veces se interrumpe y en qué cola termina su ejecución?

◆ Dibuje las colas detalladamente, mostrando los pasos y cómo el proceso va perdiendo prioridad mientras se ejecuta.

Q4					2ms
Q3				17ms	
Q2			12ms		
Q1		7ms			
Q0	2ms				
Acumulado	2ms	9ms	21ms	38ms	40ms

Se interrumpe 4 veces y termina en la cola Q4 (la quinta).