Ejercicio 1. El disco Seagate Exos 7E8 de 8 TiB e intefaz SAS tiene una velocidad de rotación de 7200 RPM, 4.16 ms de latencia de búsqueda y 215 MiB/s de tasa de transferencia.

(a) Indicar cuantos *ms* tarda en dar una vuelta completa.

(b) Indicar la tasa de transferencia de lectura al azar de bloques de 4096 bytes.

Ejercicio 2. Compute el tamaño de la FAT para:

(a) Un diskette de doble cara doble densidad 360 KiB (∼1982): FAT12, cluster[1] de 512 bytes.

(b) Un disco duro de 4 GiB (∼1998): FAT16, cluster de 4096 bytes.

(c) Un pendrive 32 GiB (∼2014): FAT32, cluster de 16384 bytes.

Ejercicio 3. El sistema de archivos de `xv6` es una estructura *à la* Fast-Filesystem for UNIX (UFS), con parámetros: bloque de 512 bytes, 12 bloques directos, 1 bloque indirecto, índices de bloque de 32 bits.

(a) Calcule el tamaño máximo de un archivo.

(b) Calcule el tamaño de la *sobrecarga* para un archivo de tamaño máximo.

(c) ¿Se podrían codificar los números de bloque con menos bits? ¿Qué otros efectos produciría utilizar la mínima cantidad de bits?

Ejercicio 4. Para un sistema de archivos `xv6`: bloque de 512 bytes, 12 bloques directos, 1 bloque indirecto, índices de bloque de 32 bits.

(a) Indique que número de bloque hay que leer para acceder al byte 451, al byte 6200 y al byte 71000.

(b) Dar la expresión matemática que indica a partir del byte que número de bloque hay que acceder. Se puede usar división por casos, ejemplo $positivo(x) = \mathbf{1}\ si\ 0 < x, \mathbf{0}\ si\ x \leq 0$.

Ejercicio 5. Considere un disco de 16 TiB (2^{44} bytes) con bloques de 4 KiB (2^{12} bytes) e índices de bloque de 32 bits. Suponga que el sistema de archivos esta organizado con i-nodos de hasta 3 niveles de indirección y hay 8 punteros a bloques directos. ¿Cuál es el máximo tamaño de archivo soportado por este sistema de archivos? Justifique su respuesta.

Ejercicio 6. Considere el sistema de archivos de `xv6`.
Indique que bloques directos o indirectos hay que crear. Suponga que el i-nodo ya está creado en disco. Puede usar esquemas.

(a) Se crea un archivo nuevo y se agregan 6000 bytes

(b) Se agregan al final 1000 bytes más.

Ejercicio 7. Supongamos que se borró todo el **mapa de bits** con los bloques en uso de un filesystem tipo UNIX. Explique el procedimiento para recuperarlo.

[1] *Cluster* es la terminología Microsoft para *block* o bloque.

Ejercicio 8. Un programa que revisa la estructura del filesystem (`fsck`) con el método del ejercicio anterior construyó la siguiente tabla de bloques que aparecen en uso *vs.* bloques que se indican como libres en el bitmap del disco.

```
In use: 1 0 1 0 0 1 0 1 1 0 1 0 0 1 0
Free:   0 0 0 1 1 1 0 0 0 1 0 1 1 0 1
```

¿Hay errores? De ser así ¿resultan importantes?
Ayuda: analice los 4 casos posibles y discuta cada uno.

Ejercicio 9. Suponiendo a) no hay nada en la caché de bloques de disco, b) cada consulta a un i-nodo genera exactamente una lectura de bloque, c) cada directorio entra en un bloque, d) cada archivo también entra en un bloque. Describa la secuencia de lecturas de bloque para acceder a los archivos:

(a) `/initrd.img`

(b) `/usr/games/moon-buggy`

Ejercicio 10. Un novato en diseño e implementación de sistemas de archivos sugiere que la primera parte del contenido de cada archivo UNIX se almacene en el **i-nodo mismo**. ¿Es una buena idea? Explique.

Ejercicio 11. El campo *link count* dentro de un i-nodo resulta redundante. Todo lo que dice es cuantas entradas de directorios apuntan a ese i-nodo, y esto es algo que se puede calcular recorriendo el grafo de directorios. ¿Por qué se usa este campo?

Ejercicio 12. Ya vimos que las estructuras de datos en disco de los FS mantienen **información redundante** que debería ser **consistente**. Piense en **más pruebas de consistencia** de debería realizar un *filesystem checker* en un sistema de archivos tipo:

(a) FAT.

(b) UNIX.

Ejercicio 13. Explique porque no se pueden realizar *enlaces duros* de directorios en un sistema de archivos UNIX.

```
$ mkdir dir1
$ ln dir1 dir2
ln: ‘dir1’: hard link not allowed for directory
```

Ayuda: pensar en el campo *link count* y en dos directorios que se autoreferencian.

Ejercicio 14. Explique la diferencia entre un *log-filesystem* y un *journaling-filesystem*.

Ejercicio 15. Enumere y explique brevemente las tres (3) capas en las que se estructura un sistema de archivos. Dar dos (2) ejemplos que muestran la conveniencia de esta división.

Ejercicio 16. ¿Verdadero o Falso? Explique.

(a) La entrada/salida programada (por interrupciones) y el DMA liberan a la CPU de hacer *pooling* a los dispositivos.

(b) Los sistemas de archivos tipo UNIX tienen soporte especial para que cuando un directorio crece en cantidad de archivos y no entra más en un bloque, pida más bloques.

(c) Es posible establecer enlaces duros (*hardlinks*) entre archivos de diferentes particiones.

(d) Los dispositivos de I/O se dividen en *I/O mapped* y *memory mapped*.

(e) Un disco duro en formato UFS[2] puede no estar lleno y aún asi no poder almacenar más archivos.

[2]Sistema de archivos del UNIX tradicional.

(f) El tamaño de un archivo no es un atributo realmente necesario en los i-nodos, se puede calcular a partir de los bloques ocupados (FAT o UFS).

(g) El uso de espacio en disco es siempre mayor o igual a la longitud del archivo.

(h) Un archivo borrado no se puede recuperar.

(i) Los sistemas de archivos como FAT o UFS no sufren de fragmentación externa.

(j) Los sistemas de archivos modernos como EXT4 o ZFS tienen problemas de escalabilidad en la cantidad de entradas de un directorio.

(k) La *syscall* `truncate` sirve para recortar el tamaño del archivo.

Ejercicio 17. ¿Porqué resulta conveniente en una estructura tipo UFS tener un **bitmap de i-nodos libres**? Exactamente lo mismo se puede conseguir utilizando un bit especial en la tabla de i-nodos que indique si está libre o no. Discuta.

Ejercicio 18. Los FS con asignación de espacio no-contiguo (FAT, UFS, etc.) suelen tener un **defragmentador** para que todos los bloques de un archivo estén **contiguos** y el *seek-time* no impacte tanto. ¿Cómo se podría mejorar la estructura de datos que mantiene la disposición o secuencia de bloques de FAT o UFS para aprovechar que siempre se intenta que la secuencia bloques sea una secuencia creciente?[3]

20201125

[3]Esta idea ya la implementan `NTFS` desde 1993 y `ext4` desde 2008.