Práctico 4 SO - Persistencia

Ejercicio 1. El disco Seagate Exos 7E8 de 8 TiB e interfaz 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.

Primero pasamos de RPM (rotaciones por minuto) a RPS (rotaciones por segundo).

→ 7200 RPM / 60s = 120 RPS.

→ Luego vemos que en 1 segundo se efectúan 120 rotaciones.

→ 1 / 120 ⇒ **0.008333333 s** (estos son los segundos que se tarda en hacer una rotación).

→ 0.008333333 s * 1000 **= 8.333333 ms**

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

Recordemos que el disco tiene 3 fases para finalmente leer un dato: $\mathbf{T_{seek}}$ (movimiento del cabezal o latencia de búsqueda) + $\mathbf{T_{rotation}}$ (esperar al sector) + $\mathbf{T_{transfer}}$ (leer el dato).
Y qué tasa de transferencia = $\mathbf{Size_{transfer} / T_{I/O.}}$

- $\mathbf{T_{seek}}$ = 4.16 ms.
- $\mathbf{T_{rotation}}$ = esto es un promedio debido a que el sector puede estar listo al momento de mover el cabezal o se pudo haber pasado, lo que implica tomar todo el tiempo de rotación para detenerlo en el sector. Por eso hacemos $\mathbf{T_{rotation}}$ **/ 2.**
  → $\mathbf{T_{rotation}}$ = 8.333333 / 2 = **4.167 ms**
- $\mathbf{T_{transfer}}$ = esto se calcula como la cantidad que quiero leer sobre la capacidad de lectura (en este caso es de 215 MiB/s).
  → $\mathbf{T_{transfer}}$ = 4096 bytes / (215 * 1048576) = 225443840
  → $\mathbf{T_{transfer}}$ = 4096 bytes / 225443840 bytes
  → $\mathbf{T_{transfer}}$ = 0.000018168 s
  → $\mathbf{T_{transfer}}$ = **0.018168 ms**

$\mathbf{T_{i/o}}$ **=** 4.16 ms + 4.167 + 0.018168 = **8.345168 ms**
Rate of I/O → $\mathbf{R_{I/O}}$ ⇒ $\mathbf{Size_{transfer} / T_{I/O}}$ ⇒ 4 KiB / 8.345168 ms = **0.479319289 ms** * 1000
→ $\mathbf{R_{I/O}}$ **= 479.319289 KiB/s.**

Respuesta final: **479.319289 KiB/s**

Ejercicio 2. Compute el tamaño de la FAT para:

a. Un disquete de doble cara doble densidad 360 KiB (~1982): FAT12, cluster de 512 bytes.

El tamaño de nuestro sistema de almacenamiento es de 360 KiB que es lo mismo que 360 * 1024 bytes. Por ende:

→ 360 * 1024 = 368640 bytes.

→ Cada cluster ocupa 512 bytes, luego entran en en el disquete:

→ 368640 / 512 = **720 clusters**.

Luego la FAT utiliza 12 bits por cada cluster.

→ 720 cluster * 12 bits = 8640 bits de longitud.

→ **1080 bytes** de longitud es el tamaño de la FAT.

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

El tamaño de nuestro sistema de almacenamiento es de 4 GiB. Por ende:
Recordemos que 1 GiB = 1.048.676 KiB.
→ Cada cluster es de 4096 bytes (4 KiB).
→ 4 GiB = 4 * 1.048.676 KiB = 4.194.304 KiB.
→ 4.194.304 KiB / 4 KiB = **1.048.676 clusters.**
Luego la FAT utiliza 16 bits por cada cluster.
→1.048.676 cluster * 16 bits = 16.777.216 bits de longitud.
→ 2.097.152 bytes de longitud es el tamaño de la FAT.
→ **2 MiB de longitud.**

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

El tamaño de nuestro sistema de almacenamiento es de 32 GiB. Por ende:
Recordemos que 1 GiB = 1.048.676 KiB.
→ Cada cluster es de 16384 bytes (16 KiB).
→ 32 GiB = 32 * 1.048.676 KiB = 33.557.632 KiB.
→ 33.557.632 KiB / 16 KiB = **2.097.352 clusters.**
Luego la FAT utiliza 16 bits por cada cluster.
→2.097.352 cluster * 32 bits = 67.115.264 bits de longitud.
→ 8.389.408 bytes de longitud es el tamaño de la FAT.
→ **8 MiB de longitud.**

Ejercicio 3. El sistema de archivos de xv6 es una estructura a 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.

El tamaño máximo de archivo en este caso es **bloques_directos + bloques_indirectos**.
→ Tamaño directo = cantidad_indice_bloque * tamaño_indice_bloque.
→ **Tamaño directo** = 12 bloques directos * 512 bytes = **6144 bytes**.
→ Índices por bloque (indirectos) = 512 bytes / 4 bytes (32 bits).
→ **Índices por bloque** = **128** índices por bloque.
→ **Tamaño indirecto** = 1 bloque * 128 índices * 512 bytes = **65.536 bytes**.

Sumamos ambas partes y obtenemos que un archivo puede pesar a lo máximo **71680 bytes**.

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

Los índices indirectos son una parte de la información de metadatos de un archivo. Almacenan una lista de bloques que contienen datos del archivo, es decir, el overhead son todos los bloques que no contienen datos.
Como tenemos un bloque indirecto, a lo sumo el overhead es:
→ Tenemos un solo bloque indirecto de 512 bytes.
→ bloques_indirectos / max_tamaño_archivo
→ 512 bytes / 71680 bytes

→ 0.5 KiB / 70 KiB
→ **0.007142857**.

Finalmente el overhead es del **0.714285714%**.

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

Supongamos ahora que tenemos índices de bloques de la mitad, es decir 16 bits.
→ Tamaño directo = cantidad_indice_bloque * tamaño_indice_bloque.
→ **Tamaño directo** = 12 bloques directos * 512 bytes = **6144 bytes**.
→ Índices por bloque (indirectos) = 512 bytes / 2 bytes (16 bits).
→ **Índices por bloque** = **256** índices por bloque.
→ **Tamaño indirecto** = 1 bloque * 256 índices * 512 bytes = **131072 bytes**.

Sumamos ambas partes y obtenemos que un archivo puede pesar a lo máximo **137216**.
Al recortar los índices de los bloques, se aprovecha mejor la capacidad de los bloques, brindando capacidad máxima de archivo superior.

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 qué número de bloque hay que leer para acceder al byte 451, al byte 6200 y al byte 71000.

Nota: bloques directos desde el 0-11. Bloques indirectos es el 12.

- Byte 451 → 451 / 512 bytes = 0 (en división entera). Hay que leer el bloque cero.
- Byte 6200 → 6200 / 512 bytes = 12 (corresponde a bloque indirecto). Luego 12 - 12 = 0, hay que leer la primera entrada del bloque indirecto.
- Byte 71000 → 71000 / 512 bytes = 138 (bloque indirecto). Luego 138 - 12 = 126, hay que leer la entrada 126 del bloque indirecto.

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) = 1 si 0 < x, 0 si x ≤ 0.

byte(x) =
- ☐ (x / tamaño_bloque) ≤ 11 → y = x / tamaño_bloque
- ☐ cc → y = (x / tamaño_bloque) - 12

Práctico 4 SO - Persistencia

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 está 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.

Veamos primero el tamaño directo:

→ 8 punteros a bloques directos = 8 * 4 KiB = 2^3 * 2^{12} bytes = 32768 bytes.

Ahora, veamos los índices por bloque:

→ 4 KiB / 4 b = 2^{12} bytes / 4 bytes = 1024 índices por bloque.

Veamos los bloques de indirección “n”:

→ Bloques indirectos = $2^{10 * 1}$.

→ Bloques doble indirectos = $2^{10 * 2}$.

→ Bloques triple indirectos = $2^{10 * 3}$.

Calculemos el tamaño de cada indirección:

→ **Región indirecta:** tamaño del bloque * punteros a bloques (bloques).

→ 4 KiB * 2^{10} = 2^{12} KiB.

→ **Región indirecta:** tamaño del bloque * punteros a bloques (bloques).

→ 4 KiB * 2^{20} = 2^{22} KiB.

→ **Región indirecta:** tamaño del bloque * punteros a bloques (bloques).

→ 4 KiB * 2^{30} = 2^{32} KiB.

Sumamos las 4 partes = Región directa + Región indirecta + Región doble indirecta + Región triple indirecta.

→ 2^{15} KiB bytes + 2^{12} KiB + 2^{22} KiB + 2^{32} KiB.

→ 4.299.198.464 KiB.

→ **4 TiB**.

Ejercicio 6. Considere el sistema de archivos de xv6. Indique qué 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.

Notemos que cada bloque es de 512 bytes. Y en xv6 tenemos 12 bloques directos.

- $bloques_{archivo}$ = 6000 bytes / 512 bytes = 11.71875.

  → $bloques_{archivo}$ = 12.

- $bloques\_directos_{archivo}$ = $bloques_{archivos}$ ‘mín’ $bloques_{directos.}$

  → $bloques\_directos_{archivos}$ = 12 ‘min’ 12.

  → **$bloques\_directos_{archivos}$ = 12.**

- $bloques\_indirectos_{archivo}$ = 0 bloques ‘max’ ($bloques_{archivo}$ - $bloques\_directos_{archivo}$).

  → $bloques\_indirectos_{archivo}$ = 0 ‘max’ 0.

  → **$bloques\_indirectos_{archivo}$ = 0**.

b. Se agregan al final 1000 bytes más.

bloques_nuevos = 1000 bytes / 512 bytes
→ bloques_nuevos = 1.9.
→ **bloques_nuevos = 2** + 1 bloque indirecto que contiene los índices.

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.

Se debería de recorrer todos los índices y chequear los bloques que estén ocupados, aunque nunca podremos decidir del todo ya que no sabemos si en verdad están ocupados o tienen basura.

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.

Si hay errores, resultan importantes. Veamos que hay 4 diferentes casos:
- Bloques marcados como usados realmente ocupados → caso correcto.
- Bloques marcados como libres realmente no ocupados → caso correcto.
- Bloques marcados como usados que realmente no están ocupados → se está usando un bloque que debería estar libre.
- Bloques marcados como libres que realmente están ocupados → pérdidas de datos o corrupción en el fs.

Ejercicio 9. Suponiendo a) no hay nada en la cache 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

read inode ‘/’ → read ‘/’ data → read inode ‘initrd.img’

b. /usr/games/moon-buggy

read inode ‘/’ → read ‘/’ data → read inode usr → read usr data → read inode games → read games data → read inode moon-buggy → reads moon-buggy data

Recordemos que en el inode están los metadatos y no los datos.

Práctico 4 SO - Persistencia

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.

Ventajas:

- **Eficiencia de acceso:** Al almacenar una parte del contenido directamente en el i-nodo, puede haber un acceso más eficiente a esa parte del archivo, ya que no es necesario seguir enlaces adicionales para recuperar la información.
- **Menos fragmentación:** Al tener parte del contenido directamente en el i-nodo, se reduce la probabilidad de fragmentación del sistema de archivos, ya que parte del archivo está ubicada de manera más contigua.

Desventajas:

- **Tamaño limitado:** Los i-nodos suelen tener un tamaño limitado, y si se reserva parte del espacio para el contenido, puede haber restricciones en el tamaño máximo de los archivos. Esto podría ser problemático para archivos grandes.
- **Desperdicio de espacio:** Para archivos pequeños, almacenar parte del contenido directamente en el i-nodo podría ser ineficiente y llevar al desperdicio de espacio, especialmente si el espacio reservado es más grande que el tamaño real del archivo.
- **Complejidad en la gestión:** La gestión de espacio y la lógica del sistema de archivos podrían volverse más complejas. La implementación de la lógica para manejar la asignación de espacio y la gestión de fragmentos podría ser más complicada.
- **Problemas de fragmentación externa:** Aunque se puede reducir la fragmentación interna, podría aumentar la fragmentación externa, ya que el espacio disponible en el sistema de archivos puede estar disperso alrededor debido a la asignación de i-nodos con espacio reservado.

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 inodo, y esto es algo que se puede calcular recorriendo el grafo de directorios. ¿Por qué se usa este campo?

El **conteo de enlaces** es un contador en el i-nodo que indica cuántas entradas de directorio están vinculadas a ese i-nodo, lo que es fundamental para el manejo de enlaces duros y la gestión eficiente de los recursos del sistema de archivos.

Un **enlace duro** (hard link) es una asociación adicional de un nombre de archivo a un i-nodo existente. Cada i-nodo en un sistema de archivos contiene información sobre un archivo, y los enlaces duros permiten que múltiples entradas de directorio apunten al mismo i-nodo, compartiendo así el mismo conjunto de datos en disco.Cuando se crea un enlace duro, se asigna un nuevo nombre de archivo a un i-nodo existente. Ambas entradas de directorio, la original y la nueva, apuntan al mismo i-nodo, lo que significa que ambos enlaces tienen acceso a la misma información del archivo, incluidos los permisos, el propietario, la fecha de creación, la fecha de modificación y el contenido del archivo

Aunque es cierto que el número de enlaces a un i-nodo puede calcularse recorriendo el grafo de directorios, el campo de conteo de enlaces dentro del i-nodo tiene su utilidad y no es redundante.

1. **Eficiencia:** calcular el número de enlaces recorriendo el grafo de directorios puede ser una operación costosa en términos de tiempo y recursos, especialmente en sistemas de archivos grandes. Mantener un campo de conteo de enlaces en el i-nodo permite acceder a esta información de manera rápida y eficiente.
2. **Optimización de la búsqueda:** el campo de conteo de enlaces proporciona una forma rápida de determinar si un i-nodo está en uso por múltiples archivos o si es un archivo único. Esto puede ser útil en operaciones como la eliminación de archivos, donde la liberación de recursos asociados al i-nodo puede depender de su número de enlaces.
3. **Garantía de integridad:** el campo de conteo de enlaces en el i-nodo ayuda a garantizar la consistencia del sistema de archivos. Si hubiera inconsistencias entre el campo de conteo de enlaces y la cantidad real de entradas de directorio que apuntan al i-nodo, podría indicar problemas en la integridad del sistema de archivos.
4. **Rendimiento de operaciones comunes:** al mantener un campo de conteo de enlaces, se optimizan operaciones comunes, como la creación y eliminación de enlaces, ya que se puede verificar rápidamente si el i-nodo ya tiene enlaces antes de realizar operaciones adicionales.

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.

- Chequear que ninguna cadena de la tabla sea apuntada por más de un archivo.
- Chequear que no exista un bucle entre cadenas.
- Chequear que todos los tamaños de archivo en el directorio nodo sea menor que la cantidad de clusters.
- Chequear que todos los archivos apunten a un cluster válido en la tabla.
- Chequear que el nombre del archivo sea válido.

b. UNIX.

- Chequear que no haya dos archivos distintos que tengan algún bloque en común.
- Chequear que todos los inodes apuntados por algún archivo estén marcados como ocupados y el resto como libres en el bitmap de inodes. Así como también, para los bloques.
- Chequear realmente que se corresponda la cantidad de archivos que apuntan a otro con el link counter.
- Chequear que ningún archivo tenga menos bloques que los necesarios para su tamaño.

Práctico 4 SO - Persistencia

Ejercicio 13. Explique por qué 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 autoreferencia.

Lo que pasa es que si dos directorios se hacen referencia uno con el otro, y por algún motivo quedan aislados, se genera una isla y los datos se terminan perdiendo, siendo imposible de acceder a ellos.

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

- Un **log file system** es aquel en que los datos y metadatos están escritos secuencialmente en un buffer circular llamado log.
- Un **journalist file system** es aquel que trackea un seguimiento de los cambios que no han sido escritos en la parte principal del filesystem guardando estos cambios en una estructura de datos llamada journal (log circular). De esta manera, ante un evento desafortunado, el filesystem puede volver atrás más rápido y restaurar lo que se ha corrompido. Depende de la implementación que datos/metadatos son trackeados por el fs.

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.

→ Capa del file system.
→ Capa de los bloques físicos.
→ Capa del dispositivo.

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. **V**
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. **F: todo es un archivo**
C. Es posible establecer enlaces duros (hard links) entre archivos de diferentes particiones. **F: inodos únicos por cada partición**
D. Los dispositivos de I/O se dividen en I/O mapped y memory mapped. **V**
E. Un disco duro en formato UFS2 puede no estar lleno y aun así no poder almacenar más archivos. **V: llenamos los inodos con archivos vacíos**
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). **F**
G. El uso de espacio en disco es siempre mayor o igual a la longitud del archivo. **F: truncate**
H. Un archivo borrado no se puede recuperar. **F**
I. Los sistemas de archivos como FAT o UFS no sufren de fragmentación externa. **V**

J. Los sistemas de archivos modernos como EXT4 o ZFS tienen problemas de escalabilidad en la cantidad de entradas de un directorio. **F**
K. La syscall truncate sirve para recortar el tamaño del archivo. **F**

Ejercicio 17. ¿Por qué 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.

Básicamente por rendimiento, en el bitmap se accede mucho más rápido a aquellos i-nodos que estén disponibles. Imaginemos tener que leer un bit de cada inodo libre, seria mucho mas lento.

Ejercicio 18. Los FS con asignación de espacio no-contiguo (FAT, UFS, etc.) suelen tener un desfragmentador 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.

→ Guardar el bloque de inicio/fin de un archivo.