Administración de E/S

1 – Dispositivos orientados a bloques

Proporcionan acceso a los dispositivos de hardware con un buffer y ofrecen cierta abstracción de sus características específicas. A diferencia de los dispositivos de caracteres, los dispositivos de bloque siempre permitirán al programador leer o escribir un bloque de cualquier tamaño (incluyendo caracteres individuales/bytes) y cualquier alineación. El **inconveniente** es que, debido a que los dispositivos de bloque están almacenados en un buffer, el programador no sabe cuánto pasará antes de que los datos escritos pasen de los búferes del Kernel al dispositivo real, o incluso en qué orden llegarán dos escrituras separadas al dispositivo físico. Además, si el mismo hardware expone tanto los dispositivos de caracteres como los de bloques, existe el riesgo de que los datos se corrompan debido a que los clientes que utilizan el dispositivo de caracteres no son conscientes de los cambios realizados en los búferes del dispositivo de bloques.

Ejemplo de archivos de dispositivos orientados a bloques

Nombre del archivo	Significado
`fd0`	1. disquetes
`hda`	IDE-disco duro o IDE-CD-ROM-Lector conectado al 1. enchufe Master
`hdb`	IDE-disco duro o IDE-CD-ROM-Lector conectado al 1. enchufe Slave
`hda1`	1. primera partición de disco del primer IDE-disco duro
`hda15`	15. partición lógica del 1. disco duro IDE
`ad0`	IDE-disco duro conectado al 1. enchufe Master (FreeBSD)
`ad1`	IDE-disco duro conectado al 1. enchufe Slave (FreeBSD)
`ad0s1`	1. Slice del primer disco duro IDE (FreeBSD)

OTROS LINKS DE INTERÉS:

- Dispositivos orientados a caracteres
- Dispositivos orientados a sockets
- Archivos de dispositivos virtuales

LOS DISPOSITIVOS ORIENTADOS A BLOQUES ALMACENAN LA INFORMACIÓN EN BLOQUES, EN GENERAL DE TAMAÑO FIJO, HACIENDO LAS TRANSFERENCIAS DE UN BLOQUE A LA VEZ.

2 – Dispositivos orientados a flujos

Transfieren los datos como una serie de bytes. En general, los dispositivos de almacenamiento secundario son orientados a bloque, y el resto son orientados a flujos. Las cintas son un ejemplo de dispositivo orientado a bloques y las impresoras son un ejemplo de dispositivo orientado a flujo.

3 – E/S programada | Polling

Para hacer la operación de E/S entre el procesador y el módulo de E/S, el procesador ejecuta un programa que controla toda la operación de E/S (programación, transferencia y finalización). Las etapas de la transferencia son la **sincronización** (el procesador está constantemente el estado del periférico consultado el registro de estado del módulo de E/S, es un bucle que se ejecuta continuamente hasta que detecta el cambio de estado e indica que el periférico está preparado), el **intercambio del dato** (si es una operación de entrada, el procesador lee el registro de datos del módulo de E/S para recoger el dato enviado por el periférico, y lo guarda en memoria; si es una operación de salida, el procesador toma de la memoria el dato que queremos enviar al periférico y lo escribe en el registro de datos del módulo de E/S). No es habitual gestionar operaciones de E/S con múltiples dispositivos utilizando E/S programada.

Fuente: Universidad Abierta de Cataluña

4 – E/S con interrupciones

Esta técnica pretende evitar que el procesador tenga que estar haciendo trabajo improductivo mientras espera a que el periférico esté preparado para hacer una nueva operación de E/S y pueda aprovechar este tiempo para ejecutar otros programas. Para ello, se debe disponer de una línea especial que tiene que formar parte del conjunto de líneas de control del bus de sistema, denominada **línea de petición de interrupción (INT).** El módulo de E/S avisa al procesador mediante esta línea e indica que está preparado para hacer la transferencia.

Fuente: Universidad Abierta de Cataluña

5 – Acceso Directo a Memoria

El procesador programa la transferencia de un bloque de datos entre el periférico y la memoria encargando a un **controlador de DMA** conectado al bus del sistema hacer toda la transferencia, o en versiones más evolucionadas un canal o procesador de E/S. Una vez acabada, este nuevo elemento avisa al procesador. De esta manera, el procesador puede dedicar todo el tiempo que dura la transferencia del bloque a otras tareas. La **nueva problemática** es que hay dos dispositivos (CPU y CDMA) que tienen que acceder de manera concurrente a la memoria. Dos soluciones posibles son las **memorias multipuerta** y la **conexión por robo de ciclo.** En este último, la CPU le cede el bus al CDMA para que éste realice el intercambio en memoria, **pero** la cesión del bus no es inmediata, y se alarga el tiempo de ejecución de las instrucciones.

Nota: en CADC vimos la transferencia por ráfaga (transfiere todo de una)

En la tabla que hay a continuación vemos cómo quedan repartidas las responsabilidades en una transferencia de E/S según la técnica de E/S que utilicemos.

	Programación	Sincronización	Intercambio	Finalización
E/S programada	Procesador	Procesador	Procesador	Procesador
E/S por interrupciones	Procesador	Módulo de E/S	Procesador	Procesador
E/S por DMA	Procesador	DMA	DMA bloquea el procesador	Procesador
Canales de E/S	Procesador/Canal de E/S	Canal de E/S	Canal de E/S bloquea el procesador	Procesador

6 – E/S mapeada

Se usa el mismo bus de direcciones para memoria y dispositivos de E/S, y las instrucciones de la CPU usadas para acceder a la memoria son también usadas para acceder a los dispositivos.

No hay instrucciones específicas de E/S. La CPU puede manipular datos de entrada y salida que residen en registros de interfaces con la misma instrucción que se utiliza para manipular palabras de memoria: cada interfase se organiza como un conjunto de registros que responden a peticiones de lectura y escritura en el espacio de direccionamiento normal. Las direcciones asignadas para estos registros no pueden utilizarse para palabras de memoria, lo cual reducen el rango de direcciones de memoria disponible.

Fuente: Wikipedia en español, Monografias.com

7 – E/S Aislada

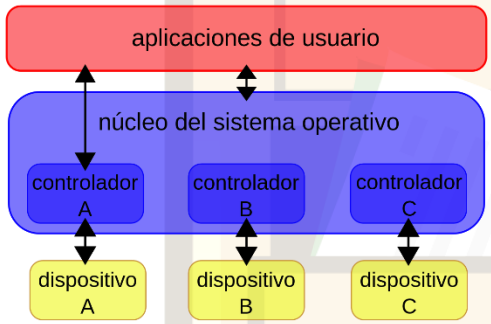
La CPU tiene instrucciones distintas de E/S, cada una se asocia con la dirección de un registro de interfase. Cuando la CPU recupera y decodifica el código de operación de una instrucción de E/S, coloca la dirección asociada con la instrucción dentro de las líneas de dirección comunes. Este método **separa** la memoria y las direcciones de E/S, para que los valores de la dirección de memoria no se afecten con la asignación de dirección de interfaces (espacios independientes).

8 – Drivers

Es un programa informático que opera o controla un tipo particular de dispositivo que está conectado a la computadora. Proporciona una interfaz de software a los dispositivos de hardware, permitiendo que el SO y los programas informáticos accedan a las funciones de hardware sin necesidad de conocer detalles precisos sobre el mismo. Un driver se comunica con el dispositivo mediante el bus al que se encuentra conectado el hardware. Cuando un programa invoca una rutina en el driver, éste emite comandos al dispositivo. Una vez que el dispositivo envía datos al driver, éste puede invocar rutinas en el programa de llamada original.

Los drivers dependen del hardware y del SO. Por lo general, proporcionan el manejo de interrupciones necesario para cualquier interfaz de hardware asíncrona.

LINK DE INTERÉS SOBRE DRIVERS EN LINUX: 02-MiIntroduc_a_Drivers0095.pdf



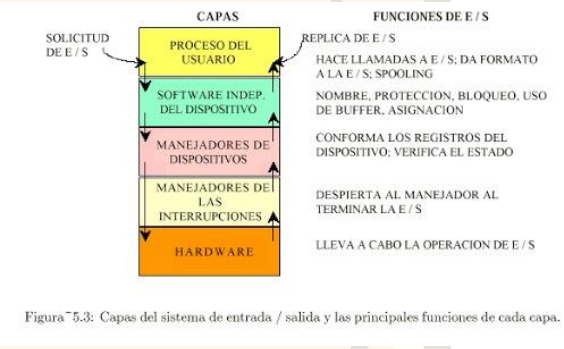
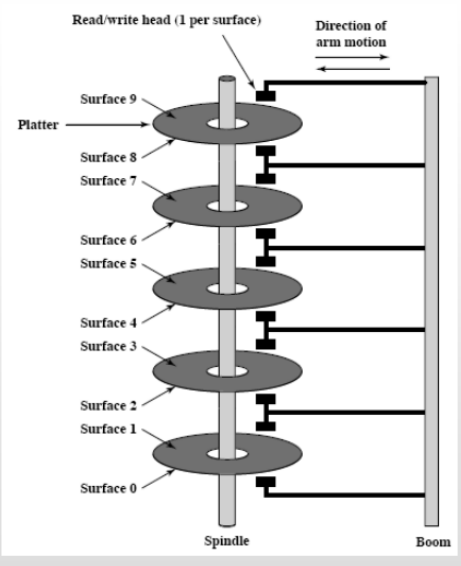
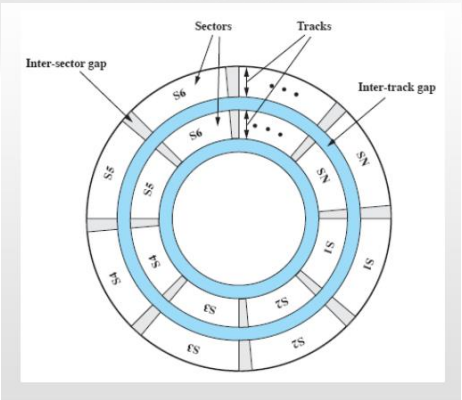


Figura 5.3: Capas del sistema de entrada / salida y las principales funciones de cada capa.

10 – Discos HDD

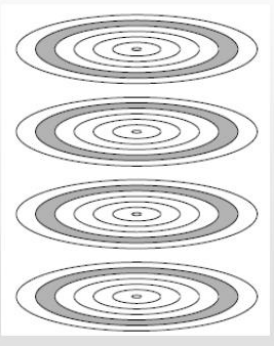


Está compuesto por **platos**, cada uno puede tener una o dos **caras**. Un cabezal con un brazo móvil puede leer y escribir en las caras. Cada cara está dividida en **pistas** (radios), que a su vez se dividen en **sectores**.



A un conjunto de varios sectores de una misma pista se lo denomina **cluster.**

También está el **cilindro**, que es el conjunto de las enésimas pistas de todas las caras. A modo de ejemplo, todas las segundas pistas forman un cilindro, o también, todas las terceras pistas.



11 – Tiempos en la obtención de datos

- **Seek Time**: en posicionarse la cabeza en el cilindro.
- **Latencia**: desde que la cabeza se posiciona en el cilindro hasta que el sector en cuestión pasa por debajo de la misma.
- **Transfer:** transferencia del sector (bloque) de disco a memoria

✓ *Datazo: si la latencia se desconoce, suponemos que es lo que tarda en dar media vuelta. En general, se conoce las rpm (1' = 60000 ms).*

Ejercicios de Discos

12 – Capacidad y Tiempo de Acceso

Se tienen 7 platos con 2 caras utilizables cada uno, 1100 cilindros, 300 sectores por pista, donde cada sector es de 512 bytes, el Seek Time es de 10 ms, el disco gira a 9000 rpm, y la velocidad de transferencia es de 10 MiB/s (Mebibytes por segundo; un Mebibyte = 2^{20} bytes).

$$Capacidad = \#Caras * \#Pistas * \#Sectores * (Espacio\ Sector)$$

$$Capacidad = 14 * 1100 * 300 * 512\ bytes = 2.365.440.000\ bytes \approx 2{,}2\ GiB$$

- ¿Cuántos sectores ocuparía un archivo de tamaño de 3 MiB?

$$\#Sectores\ Ocupados = \frac{Espacio\ Archivo}{Espacio\ Sector} = \frac{3\ x\ 2^{20}\ bytes}{2^9\ bytes} = 3\ x\ 2^{11} = 6144$$

- ¿Cuánto tarda en leerse un archivo de 15 MiB grabado de manera secuencial?

$$T = seek + latency + (T_{transf\ bloque} * \#bloques)$$

$$latency = \frac{0{,}5\ rev}{9000\ rev/min} * \frac{60\ s}{1\ min} * \frac{1000\ ms}{1\ s} = 3{,}3333\ ms$$

$$transfer = \frac{Espacio\ Archivo}{V_{transf}} = \frac{15\ MiB}{10\ MiB/s} = 1{,}5\ s = 1500\ ms$$

Lo anterior está permitido porque 15 MiB es múltiplo de 512 bytes (son 6144 x 5 bloques)

$$T = 10 + 3{,}3333 + 1500\ [ms] = 1513{,}33\ ms \approx 1{,}51\ s$$

- ¿Cuánto tarda en leerse un archivo de 16 MiB grabado de manera aleatoria?

$$T = \left(seek + latency + T_{transf\ bloque}\right) * \#bloques$$

$$T_{transf\ bloque} = \frac{512\ bytes}{10\ MiB/s} = \frac{2^9\ bytes}{10\ x\ 2^{20}\ bytes/s} * \frac{1000\ ms}{1\ s} = \frac{1}{20{,}48}\ ms$$

$$\#bloques = \frac{Espacio\ Archivo}{Espacio\ Sector} = \frac{2^4\ x\ 2^{20}\ bytes}{2^9\ bytes} = 2^{15} = 32768$$

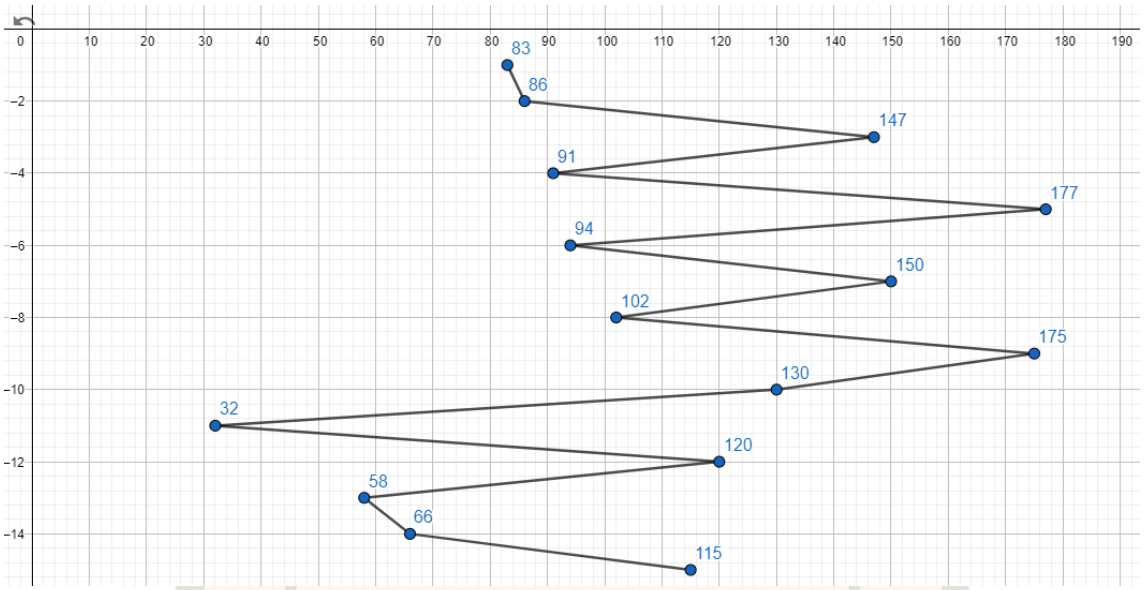
$$T = (10 + 3{,}3333 + 20{,}48^{-1}) * 32768\ [ms] \approx 438506\ ms \approx 7{,}31\ min$$

13 – Movimientos de Head

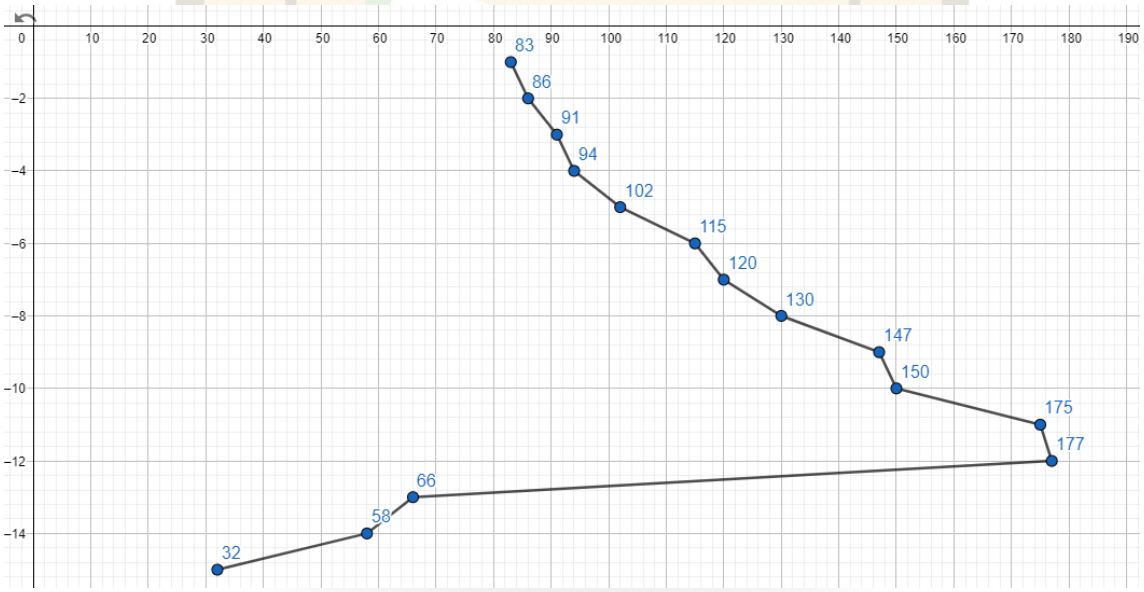
Las pistas van del 0 al 199, estoy en el track 83 y vine del 75 (izq). La cola de requerimientos:

$$86, 147, 91, 177, 94, 150, 102, 175, 130, 32, 120, 58, 66, 115$$

- FCFS: se tienen 64 + 56 + 86 + 83 + 56 + 48 + 73 + 143 + 88 + 62 + 57 = 816 mov

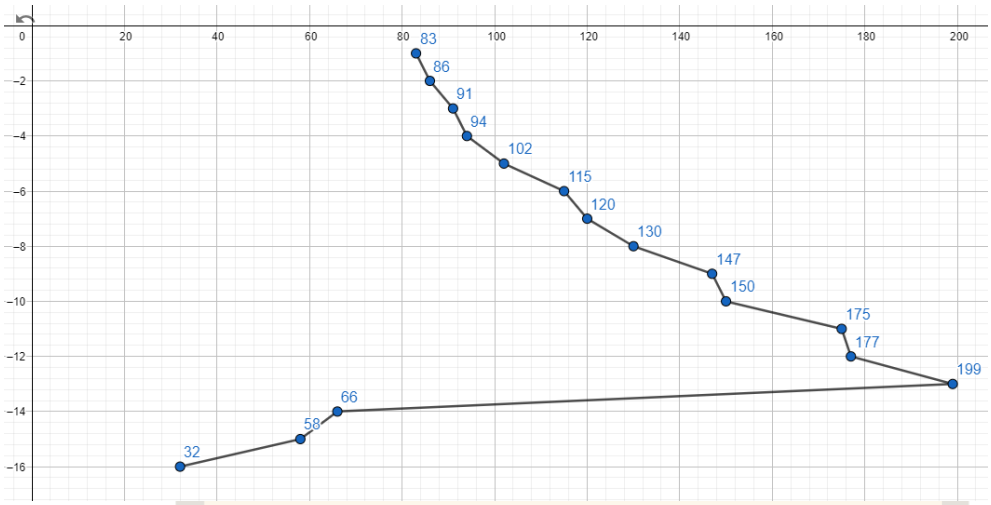


- SSTF: se tienen (177 – 83) + (177 – 32) = 94 + 145 = 239 movimientos (70% menos)

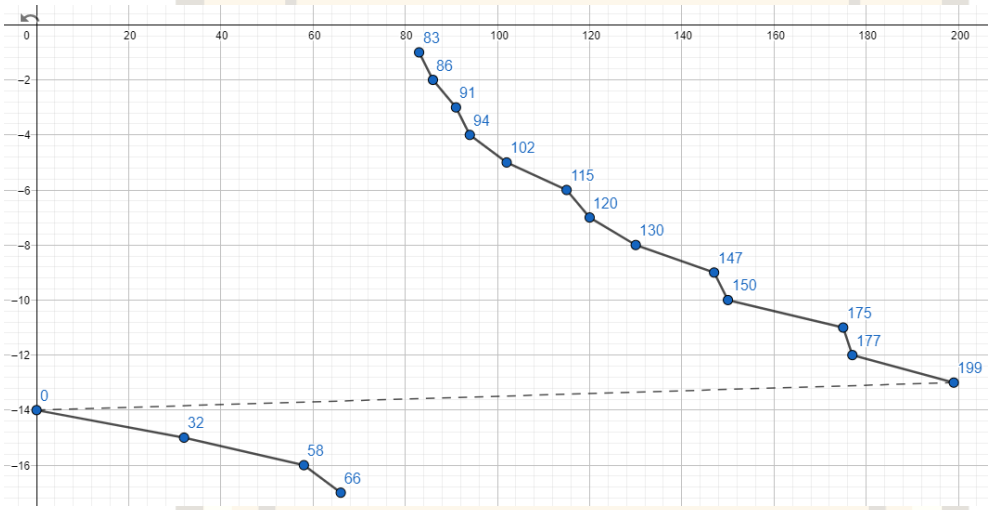


- LOOK: viene desde la izquierda (en este caso), y se va posicionando en los tracks que vaya encontrando en ese sentido, hasta llegar al último requerido (menor o igual a 199). Luego, invierte su sentido y se repite el procedimiento. Para este ejercicio particular, el algoritmo LOOK coincide con el SSTF, con 239 movimientos (70% menos que FCFS).

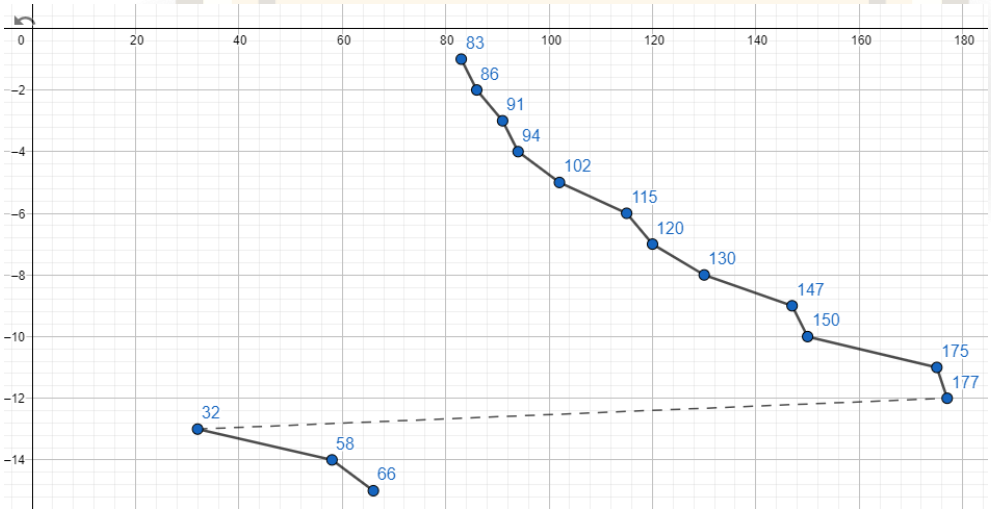
- SCAN: se tienen (199 – 83) + (199 – 32) = 116 + 167 = 283 movimientos



- C-SCAN: se tienen (199 – 83) + 66 = 182 movimientos (78% menos que en FCFS)



- C-LOOK: se tienen (177 – 83) + (66 – 32) = 94 + 34 = 128 mov (85% menos)



14 – Inanición

Se puede producir en el algoritmo SSTF, debido a que, de manera similar a como ocurría con SJF en la práctica de procesos, puede haber requerimientos de pistas muy lejanas a la posición del head, y si siguen llegando tracks *cercanos*, entonces la pista lejana puede sufrir *starvation*. También puede haber en los circulares, si está cerca pero en el sentido contrario al del head.

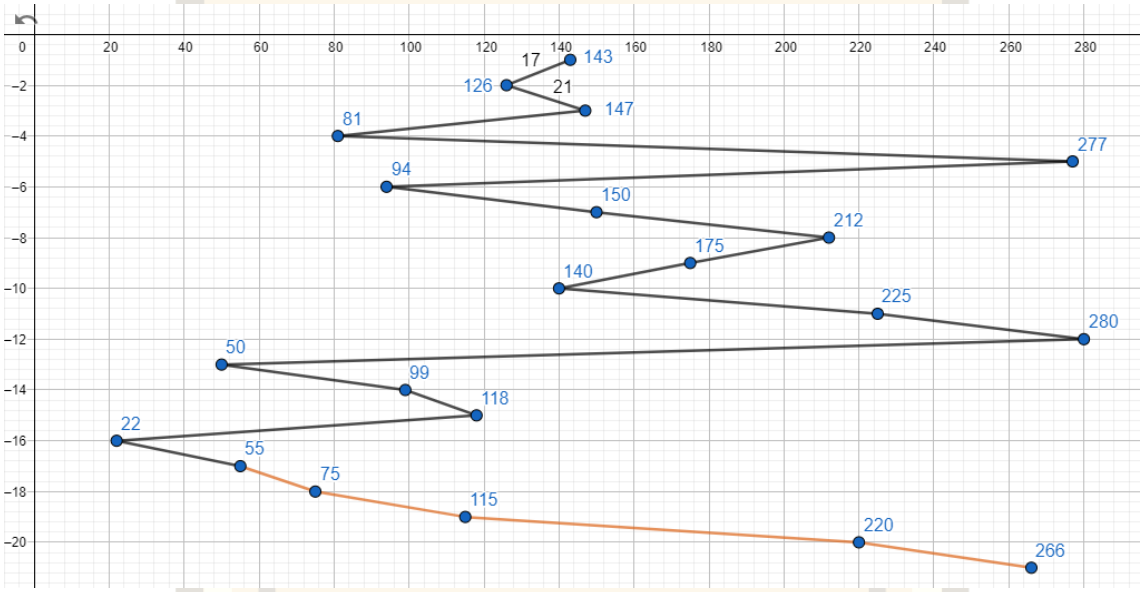
15 – Incorporación de requerimientos

Las pistas van del 0 al 299, estoy en el track 143 y vine del 125. La cola de requerimientos:

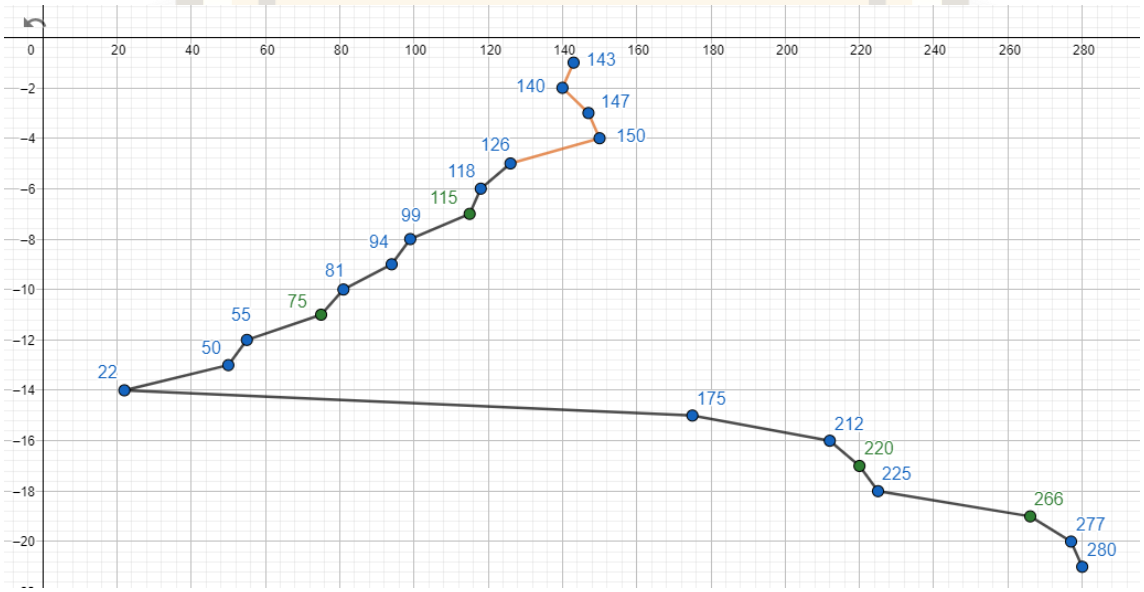
$$126, 147, 81, 277, 94, 150, 212, 175, 140, 225, 280, 50, 99, 118, 22, 55$$

Pero después de 30 movimientos, llegan estos requerimientos: $75, 115, 220, 266$

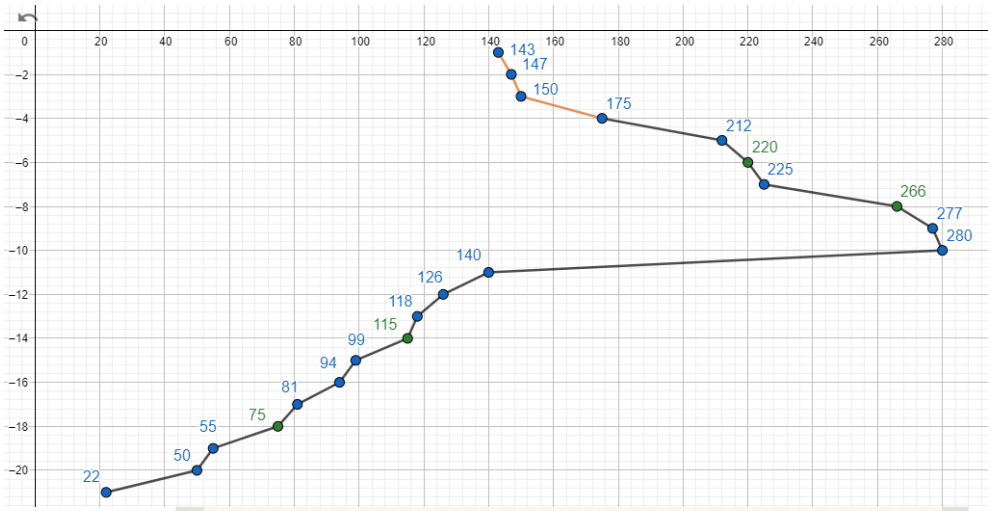
- FCFS: 17 + 21 + 66 + 196 + 183 + 118 + 72 + 140 + 230 + 68 + 96 + 244 = 1451 mov



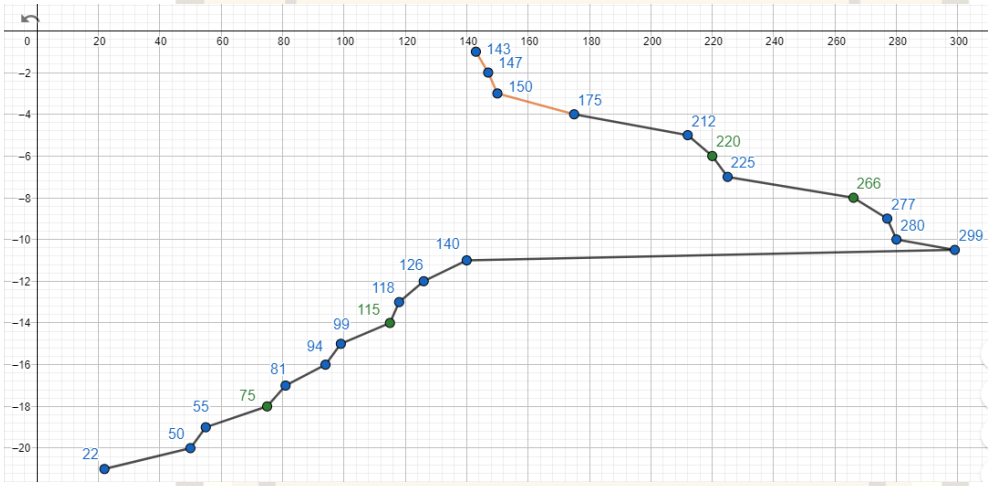
- SSTF: se tienen 3 + 10 + 128 + 258 = 399 movimientos (73% menos que en FCFS)



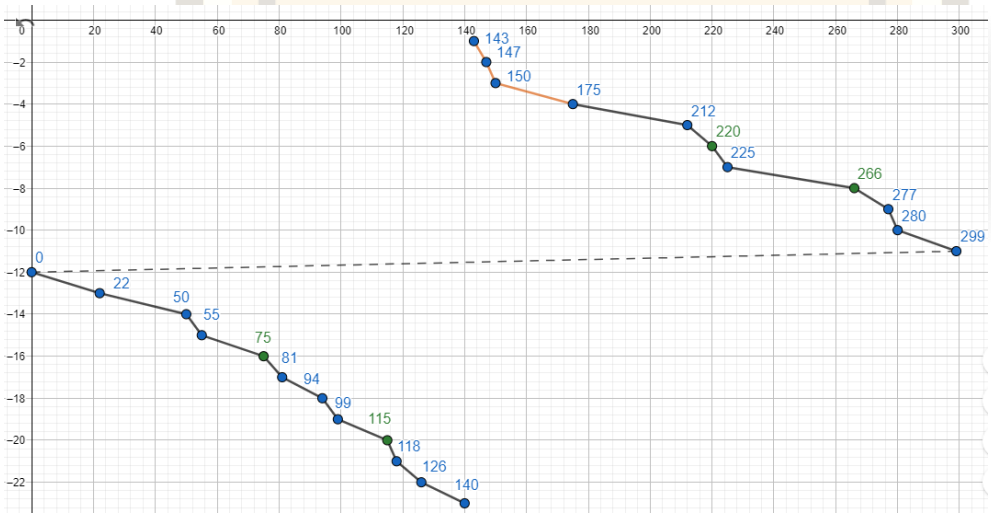
- LOOK: se tienen (280 – 143) + (280 – 22) = 137 + 258 = 395 mov (73% menos)



- SCAN: se tienen (299 – 143) + (299 – 22) = 156 + 277 = 433 mov (71% menos)



- C-SCAN: se tienen (299 – 143) + 140 = 296 movimientos (80% menos que en FCFS)



- C-LOOK: similar al anterior, pero no están los nodos 299 y 0, sino que directamente se produce el salto desde el 280 al 22, habiendo un total de (280 – 143) + (140 – 22) = 137 + 118 = 255 movimientos, que es un 82% menos que lo necesario en FCFS.

16 – Atención de Page Faults

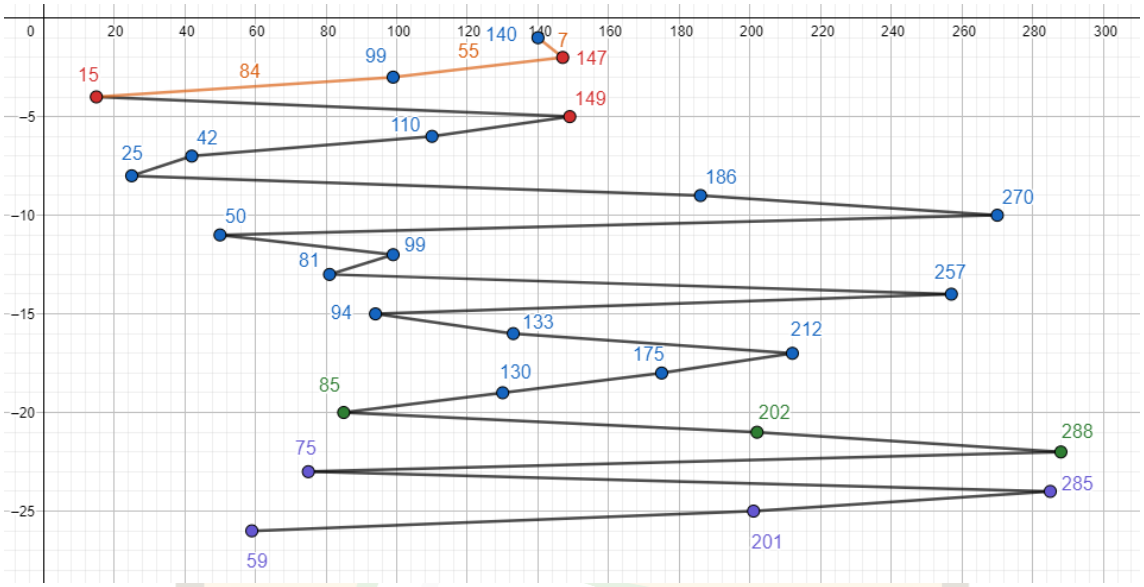
Las pistas van del 0 al 299, estoy en el track 140 y vine del 135. La cola de requerimientos:

$$99, 110, 42, 25, 186, 270, 50, 99, 147^{PF}, 81, 257, 94, 133, 212, 175, 130$$

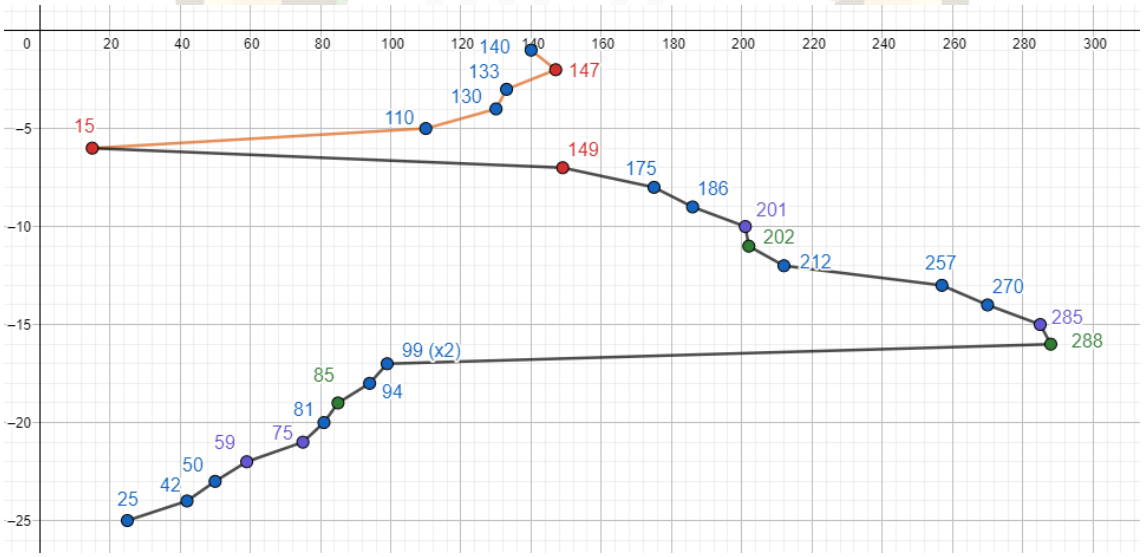
Pero después de 30 movimientos, llegan estos requerimientos: $85, 15^{PF}, 202, 288$

Y a los (30+40=70) movimientos, llegan estos requerimientos: $75, 149^{PF}, 285, 201, 59$

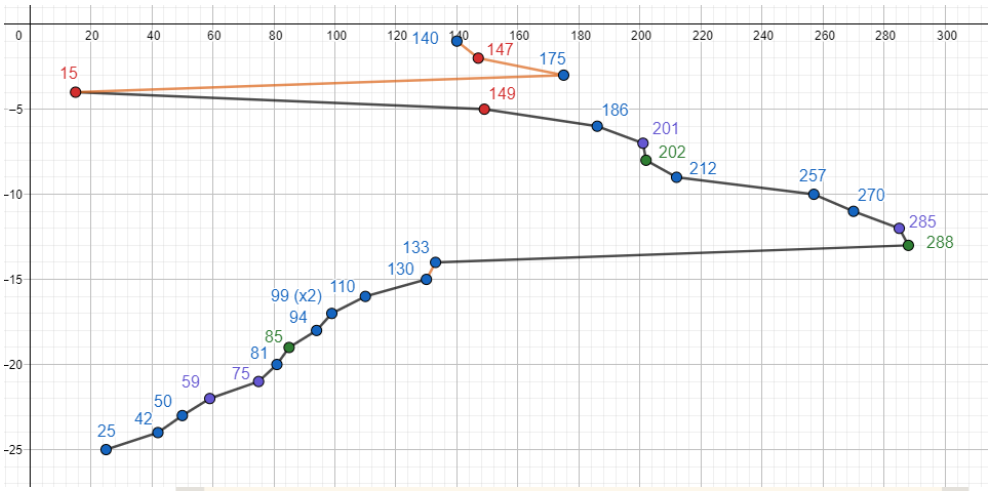
- FCFS: se tienen 146 + 258 + 465 + 67 + 339 + 245 + 416 + 436 = 2372 movimientos



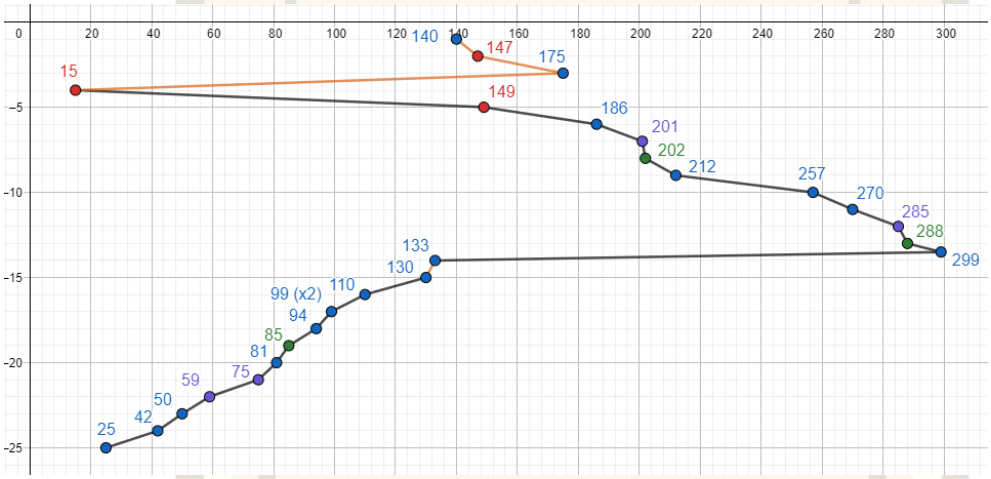
- SSTF: se tienen 7 + 132 + 273 + 263 = 675 movimientos (72% menos que en FCFS)



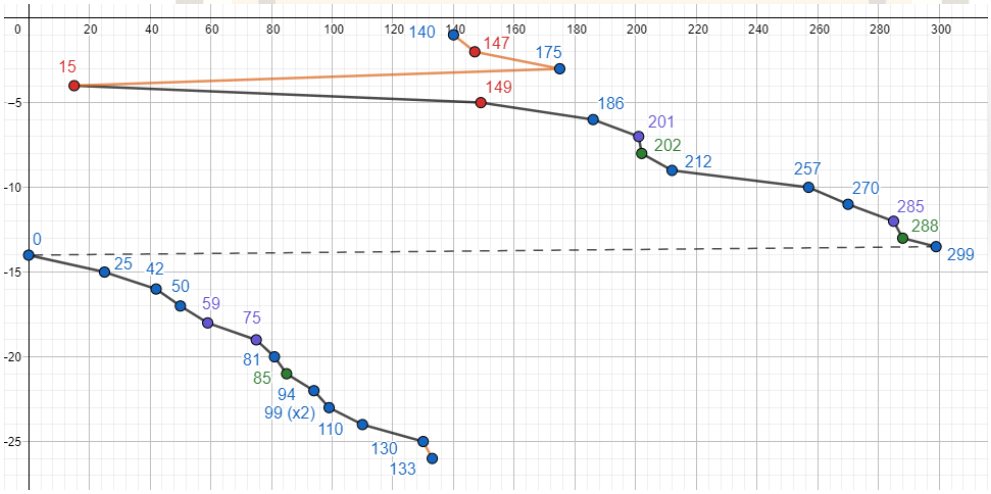
- LOOK: se tienen 35 + 160 + 273 + 263 = 731 movimientos (70% menos que FCFS)



- SCAN: se tienen 35 + 160 + 284 + 274 = 753 movimientos (69% menos que FCFS)



- C-SCAN: se tienen 35 + 160 + 284 + 133 = 612 movimientos (75% menos)



- C-LOOK: similar al anterior, sólo que el salto se realiza entre el 288 y el 25, quedando entonces 35 + 160 + 273 + 108 = 576 movimientos (76% menos que en FCFS)

Notar que, aunque los circulares tienen que continuar con el sentido izquierda-derecha, a pesar de los PF, afortunadamente en este caso coincide en la primera parte con los no circulares.

Administración de Archivos

17 – Asignación contigua

Los datos son dispuestos en forma contigua. Para mantener la información, es necesario saber en qué bloque comienza y la cantidad de bloques que tiene el archivo. Tiene 3 ventajas:

- Acceso directo a los datos, sólo necesito la dirección de inicio y un desplazamiento.
- Acceso más rápido en discos, se efectúan menos movimientos mecánicos del cabezal.
- Recuperación de datos perdidos en algún borrado es más sencilla y probable.

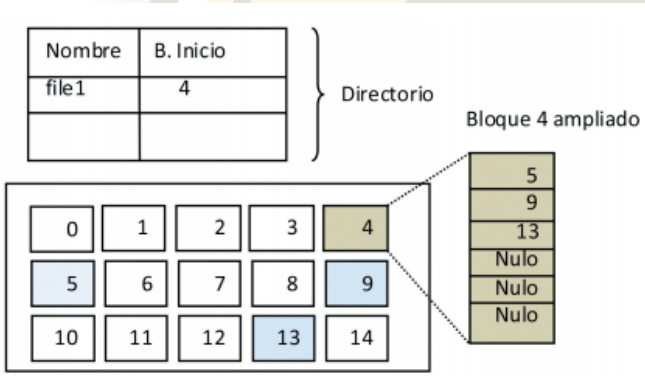
Como desventaja, se tiene que un archivo sólo puede crecer hasta el inicio de su vecino, ya que cualquier otro hueco que esté libre no estará contiguo a él. Además, se provoca **fragmentación externa** debido a que puede que no sea posible satisfacer una solución de espacio a pesar de existir el espacio en forma de "huecos" y que sumados satisfagan la petición.

18 – Asignación enlazada

Los bloques de datos forman una lista encadenada, es decir, cada bloque contiene un puntero al próximo bloque, lo que permite que los bloques puedan estar en cualquier parte del equipo de almacenamiento. Es necesaria una referencia al primer y último bloque de datos en el bloque de control de archivo. Como ventaja, los archivos pueden crecer mientras existan bloques libres, y se elimina el problema de la fragmentación externa. **Pero** no permite el acceso directo, ya que para acceder a un bloque **n** hay que recorrer los **n-1** bloques que le preceden, la dispersión de los bloques implica, en general, un **acceso más lento**, y además se gasta espacio adicional ya que cada bloque contiene un campo dedicado al puntero, inútiles desde el *pov* del usuario.

19 – Asignación Indexada

Se mantiene una tabla en donde cada entrada referencia a un bloque de datos. Como ventaja, no se provoca fragmentación externa, permite el acceso aleatorio, y los archivos pueden crecer mientras haya espacio y forma de hacer referencia a sus bloques. La desventaja es que el contenido de un archivo, en general, está disperso, lo cual implica **accesos lentos** comparado con la asignación contigua, pero no es muy grave viendo las desventajas de dicha asignación.



Fuente: Universidad Cooperativa de Colombia

20 – Adm del Espacio Libre

- **Mapa de bits:** cada bloque se representa por un bit. Si el bloque está libre se representa con un 1, y en caso contrario con un cero. La ventaja de este método es que es muy simple, **pero** debe mantener la **estructura completa en memoria principal.**

- **Lista enlazada:** se enlazan todos los bloques del disco, almacenando un puntero al primer bloque libre en un lugar especial del disco. Sin embargo, este esquema no es eficiente ya que para recorrer la lista se requiere mucho tiempo de E/S.

- **Recuento**: en general, muchos bloques contiguos se asignan o liberan simultáneamente, como en el algoritmo de asignación contigua, así que en lugar de almacenar la dirección de **n** bloques, sólo se almacena la dirección del primer bloque libre y del número de los **n** bloques contiguos que le siguen. Cada entrada de lista es una dirección y una cuenta.

21 – Contenido de un inodo

Estructura con 10 direcciones DD, 1 DIS, 1 DID y 1 DIT. Cada bloque es de 1 Kib (Kibibit). Cada dirección para referenciar un bloque es de 32 bits. Entonces:

$$\#Direcciones\ en\ bloque = \frac{Espacio\ Bloque}{Espacio\ Dirección} = \frac{2^{10}\ bits}{2^5\ bits} = 2^5 = 32$$

El resultado anterior nos sirve para el cálculo del tamaño máximo de archivo, específicamente para los direccionamientos indirectos. Las 10 direcciones DD referencian 10 x 1 Kib de datos. La DIS referencia 32 x 1 Kib de datos. La DID referencia 32 x 32 x 1 Kib de datos. Finalmente, la DIT referencia 32 x 32 x 32 x 1 Kib de datos. En resumen, tenemos:

$$Espacio\ Archivo = (10 + 32 + 32^2 + 32^3)\ x\ 1\ Kib = 33834\ Kib \approx 33\ MiB$$

22 – Localización de un inodo

$$nro\ bloque = \frac{nro\ inodo - 1}{\#inodos\ x\ bloque} + bloque\ de\ comienzo\ de\ la\ lista\ de\ inodos$$

Si el bloque 2 está en el comienzo de la lista, y hay 8 inodos por bloque, entonces:

- **Inodo 8**: se encuentra en el bloque (7 / 8) + 2 = 0 + 2 = 2
- **Inodo 9**: se encuentra en el bloque (8 / 8) + 2 = 1 + 2 = 3

¿Dónde estarían para bloque de disco de 16 nodos?

- **Inodos 8 y 9**: se encuentran en el bloque 0 + 2 = 2, porque 8 y 9 < 16

$$offset = (nro\ nodo - 1)\ mod\ (\#inodos\ x\ bloque) * medida\ de\ inodo\ del\ disco$$

Si cada inodo del disco ocupa 64 bytes, y hay 8 inodos por bloque, tenemos:

- **Inodo 8**: comienza en el offset (7 mod 8) * 64 bytes = byte 448
- **Inodo 6**: comienza en el offset (5 mod 8) * 64 bytes = byte 320

Si fueran inodos de 128 bytes, y 24 inodos por bloque, ¿dónde empezarían?

- **Inodo 8**: comienza en el offset (7 mod 24) * 128 bytes = byte 896
- **Inodo 6**: comienza en el offset (5 mod 24) * 128 bytes = byte 640

23 – Laboratorio de E/S

El programa **hdparm** es una utilidad de línea de comandos de GNU/Linux y Windows para ver y ajustar los parámetros del hardware de los discos IDE (principalmente) y SATA, como el caché de disco, el modo de descanso, el control de energía, la gestión acústica y los ajustes DMA.

En mi caso *(a nadie le importa)*, el disco tiene 16383 cilindros, 16 cabezas y 63 sectores. En SSD aparece un error de identificación porque son dispositivos NVMe, lo cual no es compatible.