Practica 6 PN

Administración de E/S

1)

Característica	Dispositivos de bloque	Dispositivos de flujo
Tipo de acceso	Acceso aleatorio: permiten leer/escribir bloques específicos	Acceso secuencial: los datos se procesan en orden
Unidades de datos	Operan con bloques de datos de tamaño fijo	Operan con flujos continuos de datos, sin estructura fija
Ejemplos comunes	Discos duros, SSD, unidades USB, particiones de disco	Teclados, mouse, micrófonos, cámaras de video
Buffering y caché	Suelen usar sistemas de caché para mejorar el rendimiento	No suelen utilizar caché; los datos se procesan en tiempo real
Uso típico	Almacenamiento masivo y persistente	Transmisión de datos en tiempo real o I/O
Acceso por el SO	Requieren sistemas de archivos para organizar y acceder a los datos	Utilizan interfaces simples como flujos de I/O
Control de errores	Mayor control en la transferencia y manipulación de datos	Menor control, dependen de la rapidez del hardware

2)

E/S programada:

La CPU tiene control directo sobre la I/O

- Controla el estado
- Comandos para leer y escribir
- Transfiere datos

Cpu espera que el componente de I/O complete la operación

Se desperdician ciclos de CPU

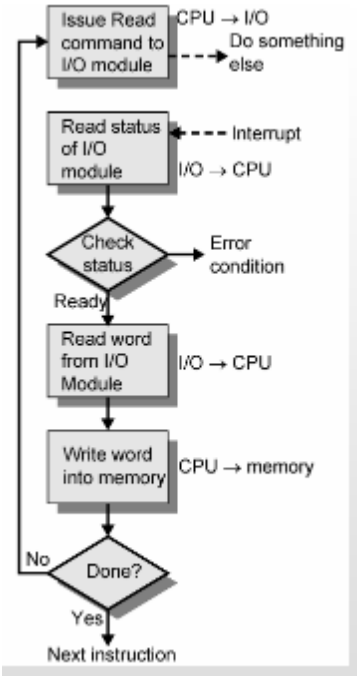
En la I/O Programada, es necesario hacer polling del dispositivo para determinar el estado del mismo

- Listo para recibir comandos
- Ocupado
- Error

Ciclo de “Busy-wait” para realizar la I/O
Puede ser muy costoso si la espera es muy larga

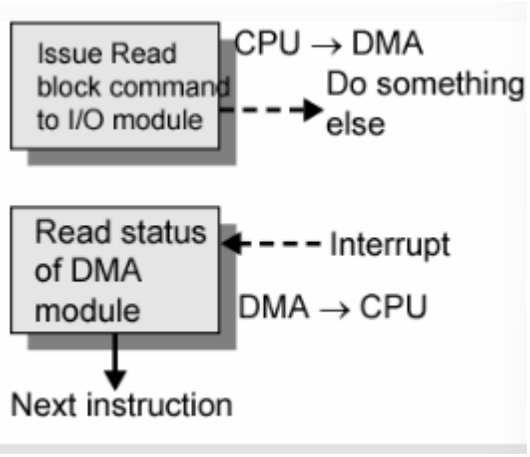
E/S manejada por interrupciones:

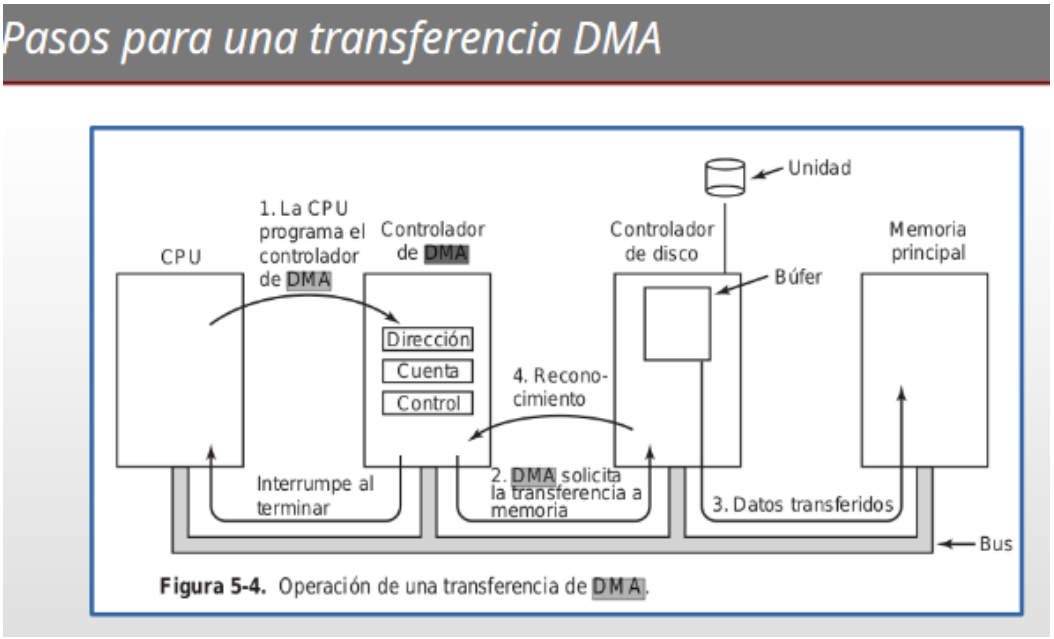
- Soluciona el problema de la espera de la CPU
- La CPU no repite el chequeo sobre el dispositivo
- El procesador continúa la ejecución de instrucciones
- El componente de I/O envía una interrupción cuando termina



DMA:

- Un componente de DMA controla el intercambio de datos entre memoria principal y el dispositivo
- El procesador es interrumpido luego de que el bloque entero fue transferido.





3)

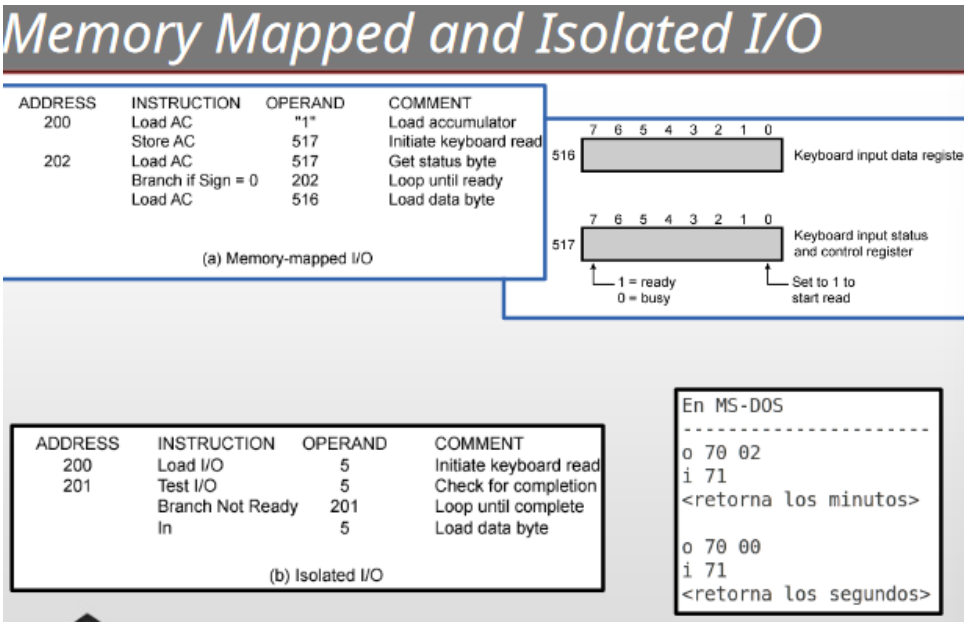
E/S mapeada:

- Dispositivos y memoria comparten el espacio de direcciones
- I/O es como escribir/leer en la memoria

- No hay instrucciones especiales para I/O
    - Ya se dispone de muchas instrucciones para la memoria

E/S aislada:

- Espacio separado de direcciones
- Se necesitan líneas de I/O. Puertos de E/S
- Instrucciones especiales
    - Conjunto Limitado



4)

- Generalidad:
    - Es deseable manejar todos los dispositivos de I/O de una manera uniforme, estandarizada
    - Ocultar la mayoría de los detalles del dispositivo en las rutinas de niveles más “bajos” para que los procesos vean a los dispositivos, en términos de operaciones comunes como: read, write, open, close, lock, unlock
- Interfaz uniforme
- Eficiencia

    - Los dispositivos de I/O pueden resultar extremadamente lentos respecto a la memoria y la CPU
    - El uso de la multi-programación permite que un proceso espere por la finalización de su I/O mientras que otro proceso se ejecuta
- Planificación
    - Organización de los requerimientos a los dispositivos
    - Ej: planificación de requerimientos a disco para minimizar tiempos
- Buffering - Almacenamiento de los datos en memoria mientras se transfieren
    - Solucionar problemas de velocidad entre los dispositivos
    - Solucionar problemas de tamaño y/o forma de los datos entre los dispositivos
- Caching - Mantener en memoria copia de los datos de reciente acceso para mejorar la performance
- Spooling - Administrar la cola de requerimientos de un dispositivo
    - Algunos dispositivos de acceso exclusivo, no pueden atender distintos requerimientos al mismo tiempo:
        - Por ejem: impresora
    - Spooling es un mecanismo para coordinar el acceso concurrente al mismo dispositivo
- Reserva de Dispositivos: Acceso exclusivo
- Manejo de Errores:
    - El S.O. debe administrar errores ocurridos (lectura de un disco, dispositivo no disponible, errores de escritura)
    - La mayoría retorna un número de error o código cuando la I/O falla
    - Logs de errores
- Formas de realizar I/O
    - Bloqueante: El proceso se suspende hasta que el requerimiento de I/O se completa
        - Fácil de usar y entender

        - No es suficiente bajo algunas necesidades
    - No bloqueante: El requerimiento de I/O retorna cuando es posible
        - Ejemplo: Interfaz de usuario que recibe input desde el teclado/mouse y se muestra screen.
        - Ejemplo: Aplicación de video que lee frames desde un archivo mientras va mostrándolo en pantalla.

5)

Son la interfaz entre el SO y el Hardware

Forman parte del espacio de memoria del Kernel

En general se cargan como módulos

Contienen el código dependiente del dispositivo. Manejan un tipo de dispositivo, se encargan de traducir los requerimientos abstractos en los comandos del dispositivo

Escribe sobre los registros del controlador

Acceso a la memoria mapeada

Encola requerimientos

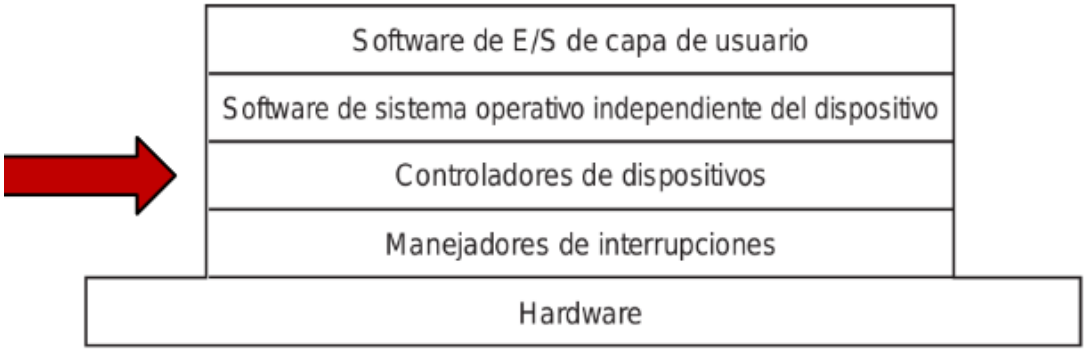
Comúnmente las interrupciones generadas por los dispositivos son atendidas por funciones provistas por el driver

Los fabricantes del HW implementan el driver en función de una API especificada por el SO

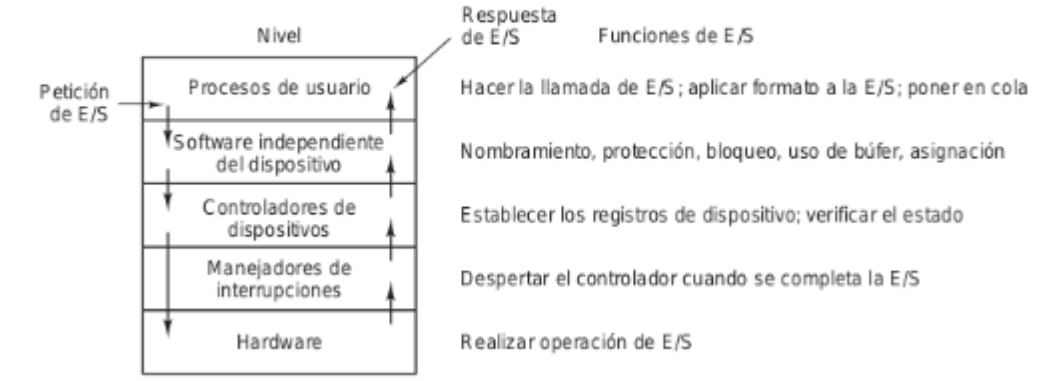
open(), close(), read(), etc

Para agregar nuevo HW sólo basta indicar el driver correspondiente sin necesidad de cambios en el Kernel

6)



7)



- Consideremos la lectura sobre un archivo en un disco:
  - Determinar el dispositivo que almacena los datos
    - Traducir el nombre del archivo en la representación del dispositivo.
  - Traducir requerimiento abstracto en bloques de disco (Filesystem)
  - Realizar la lectura física de los datos (bloques) en la memoria

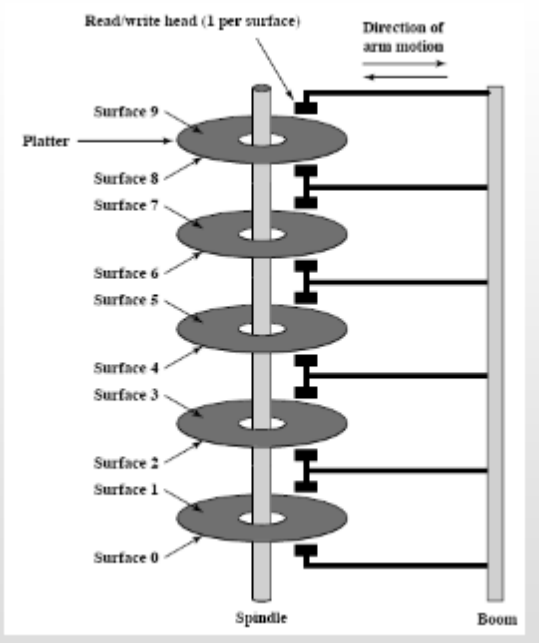
- Marcar los datos como disponibles al proceso que realizo el requerimiento
  - Desbloquearlo
- Retornar el control al proceso

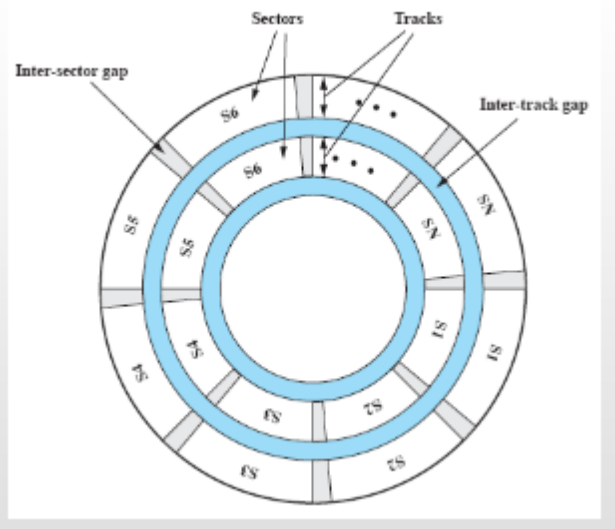
9)

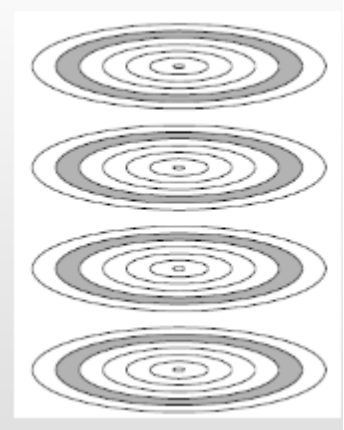
1. **Operaciones comunes de E/S**:
   a. `read`, `write`, `open`, `close`, `lock`, `unlock`.
2. **Planificación:**
   a. Organización de los requerimientos a dispositivos, como la planificación de acceso a discos.
3. **Buffering**:
   a. Almacenamiento temporal de datos en memoria mientras se transfieren entre dispositivos para resolver problemas de velocidad o tamaño.
4. **Caching**:
   a. Mantener en memoria datos recientemente accedidos para mejorar el rendimiento.
5. **Spooling**:
   a. Mecanismo para administrar la cola de requerimientos a un dispositivo de acceso exclusivo, como una impresora.
6. **Reserva de dispositivos**:
   a. Asegurar acceso exclusivo a un dispositivo específico.
7. **Manejo de errores**:
   a. Administración de errores en operaciones de E/S, como problemas de lectura/escritura y generación de códigos o logs de error.
8. **Formas de realizar E/S**:
   a. **Bloqueante**: Suspensión del proceso hasta que se complete la operación de E/S.
   b. **No bloqueante**: La operación de E/S retorna de inmediato, permitiendo que el proceso continúe (ej., aplicaciones de video o interfaces gráficas).

Administración de Discos

10)







11)

- Seek time (posicionamiento): tiempo que tarda en posicionarse la cabeza en el cilindro
- Latency time (latencia): tiempo que sucede desde que la cabeza se posiciona en el cilindro hasta que el sector en cuestión pasa por debajo de la misma
- Transfer time (transferencia): tiempo de transferencia del sector (bloque) del disco a la memoria

12)

A)

tamaño disco = #caras * #pistas cara * #sectores pista * tamaño sector

Tamaño = (7 ∗ 2) * 1100 ∗ 300 ∗ 512

2.365.440.000 bytes

2.365.440.000 / 1024 = 2.310.000 MIB

2.310.000 / 1024 = 2.255,85 KIB

2.255,85 / 1024 = 2.20 GIB

B)

Mib = 2^20

2^20 × 3 = 3.145.728

3.145.728 × 1024 = 3.221.225.472 bytes

3.221.225.472 / 512 = 6.291.456

Ocuparía 6.291.456 de sectores

c)

Sec = seek + latency + (tiempo transferencia bloque * #bloques)

Latencia = 60000 ∗ 1/2 / 9000 = 3.33 ms

transferencia = 1000 ∗ 1/2 / 10 MIB

Transformo los 10 mib a bytes...

10.485.760

transferencia = 1000 ∗ 512 / 10.485.760 =

0.04888

Bloques = 15 mib / 512

Paso a bytes =

15.728.640

bloques = 15.728.640 / 512 = 30.720

10 + 3.33 + (0.0488∗ 30720) = 1,511.466

D)

Sec = (seek + latency + tiempo transferencia bloque) * #bloques)

Latencia = 60000 ∗ 1/2 / 9000 = 3.33 ms

transferencia = 1000 ∗ 1/2 / 10 MIB

Transformo los 10 mib a bytes...

10.485.760

transferencia = 1000 * 512 / 10.485.760 =

0.0488

Bloques = 16 mib / 512

Paso a bytes =

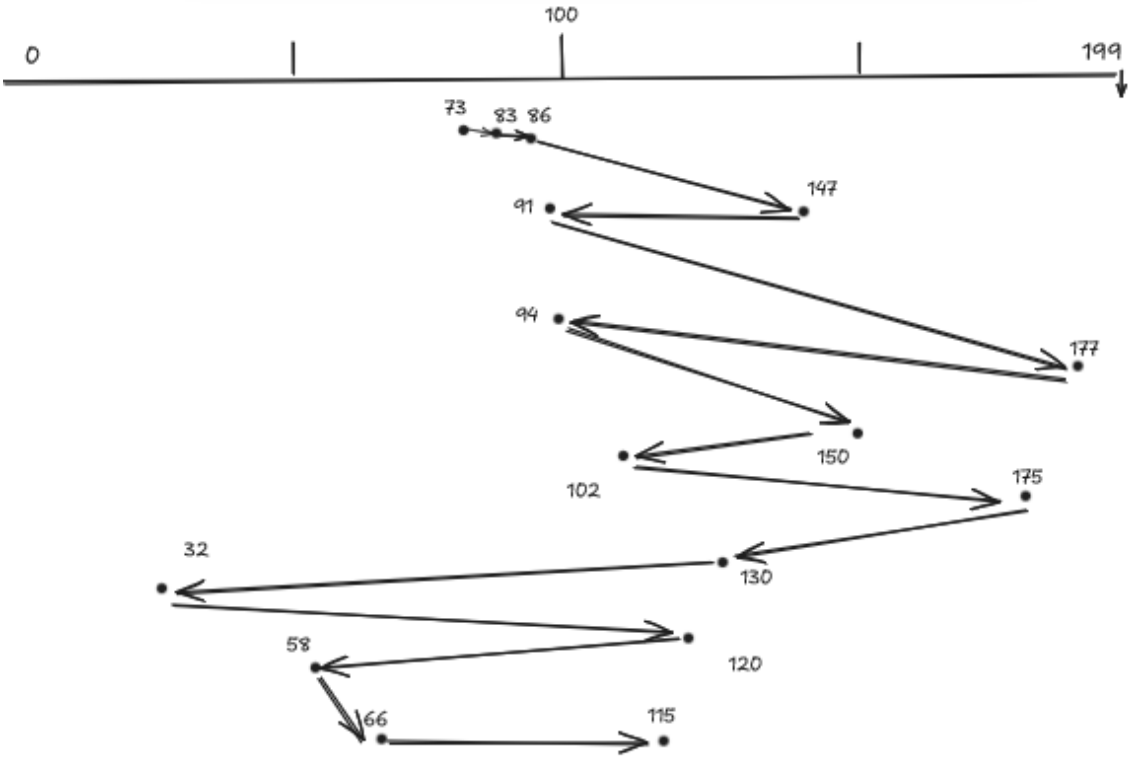
16.777.216

bloques = 16.777.216 / 512 = 32.768

(10 + 3.33 + 0.0488)* 32.768 = 438,396.5184 ms

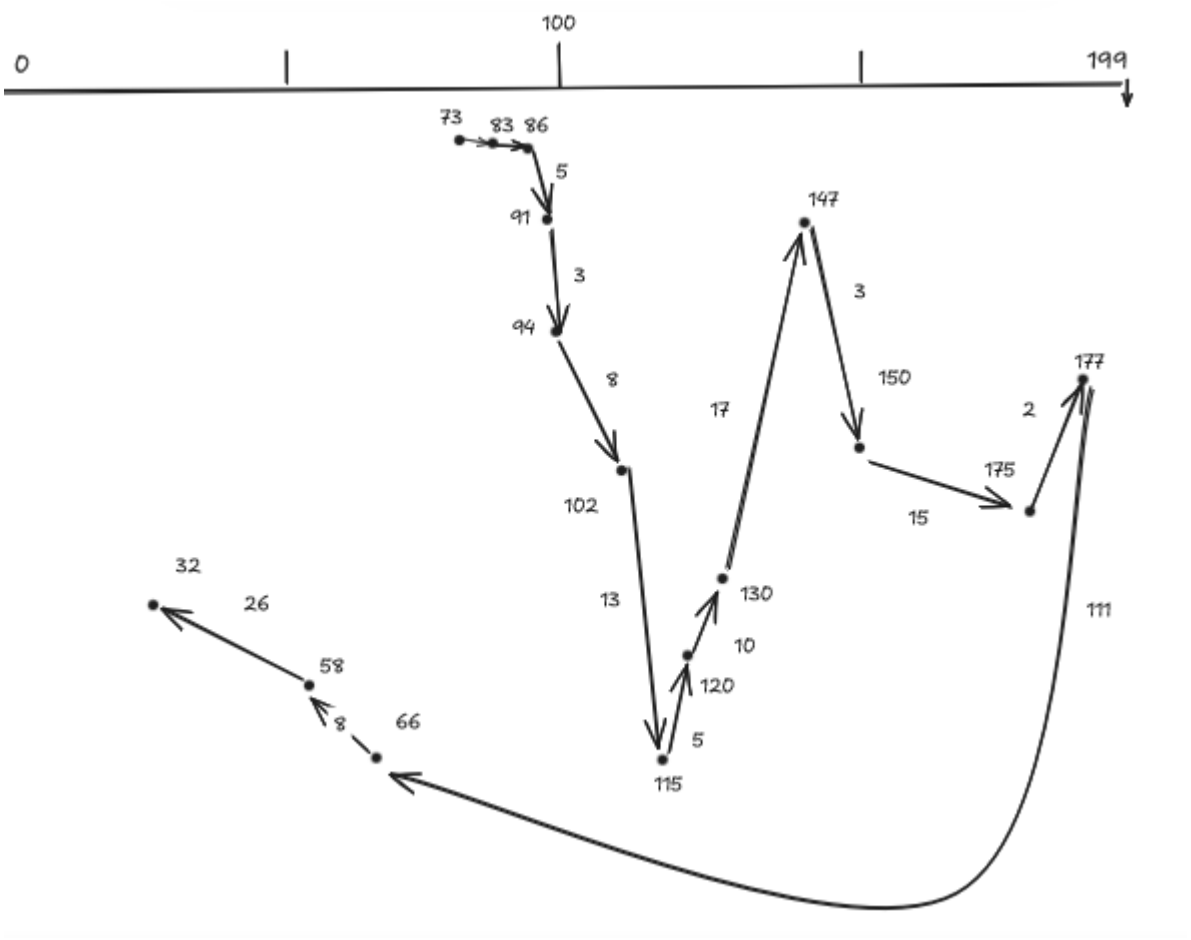
13)

A)



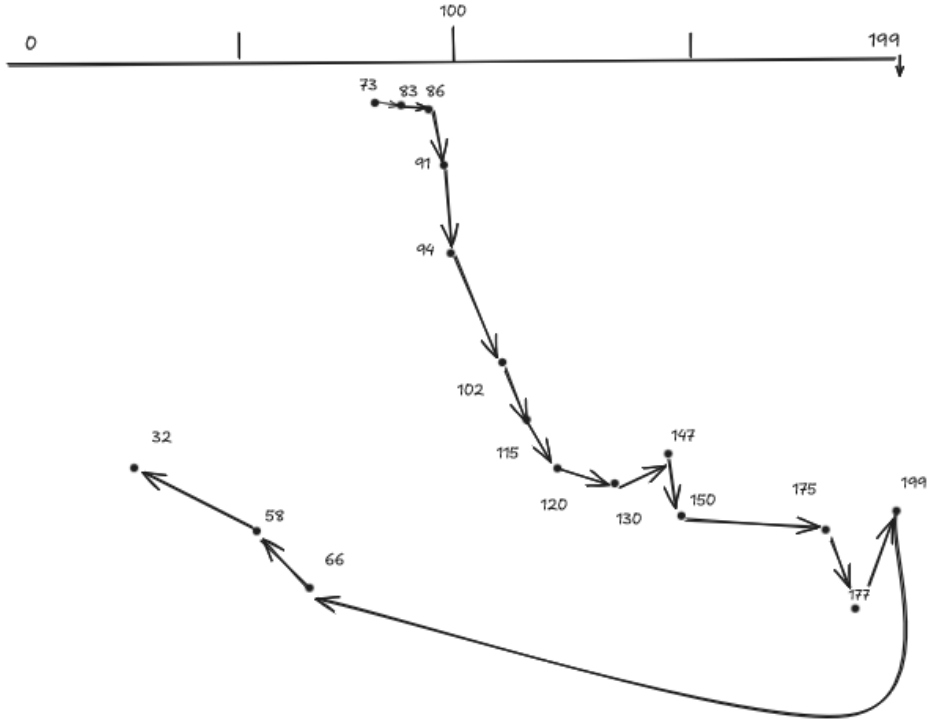
movimientos = 816

b)



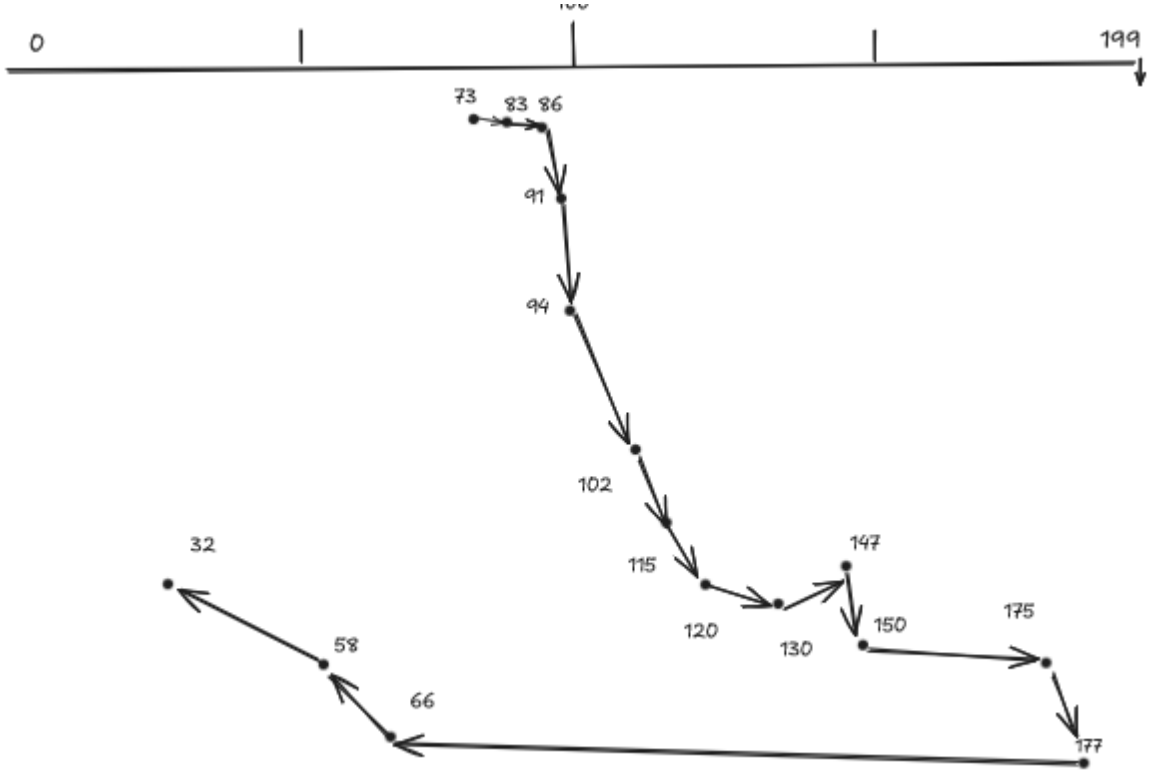
movimientos = 236

c)



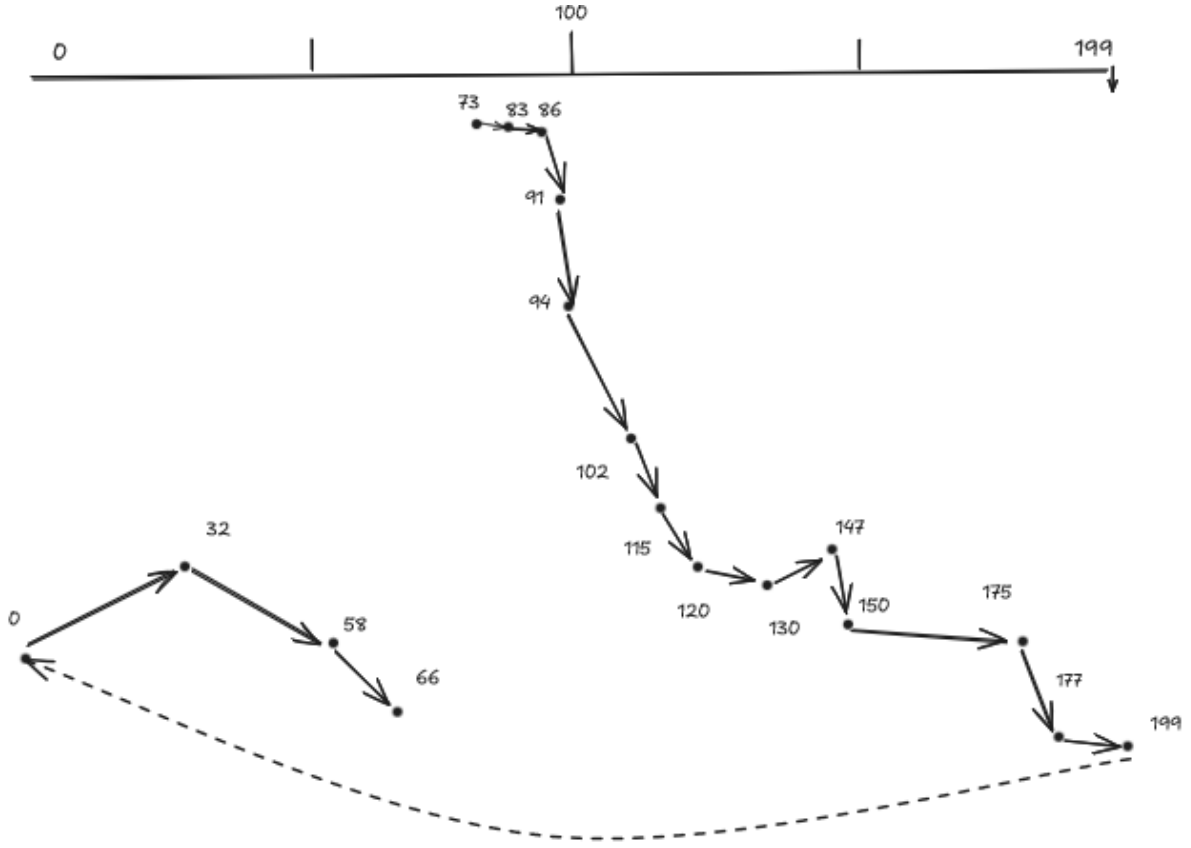
Movimientos = 116 + 167 = 283

d)

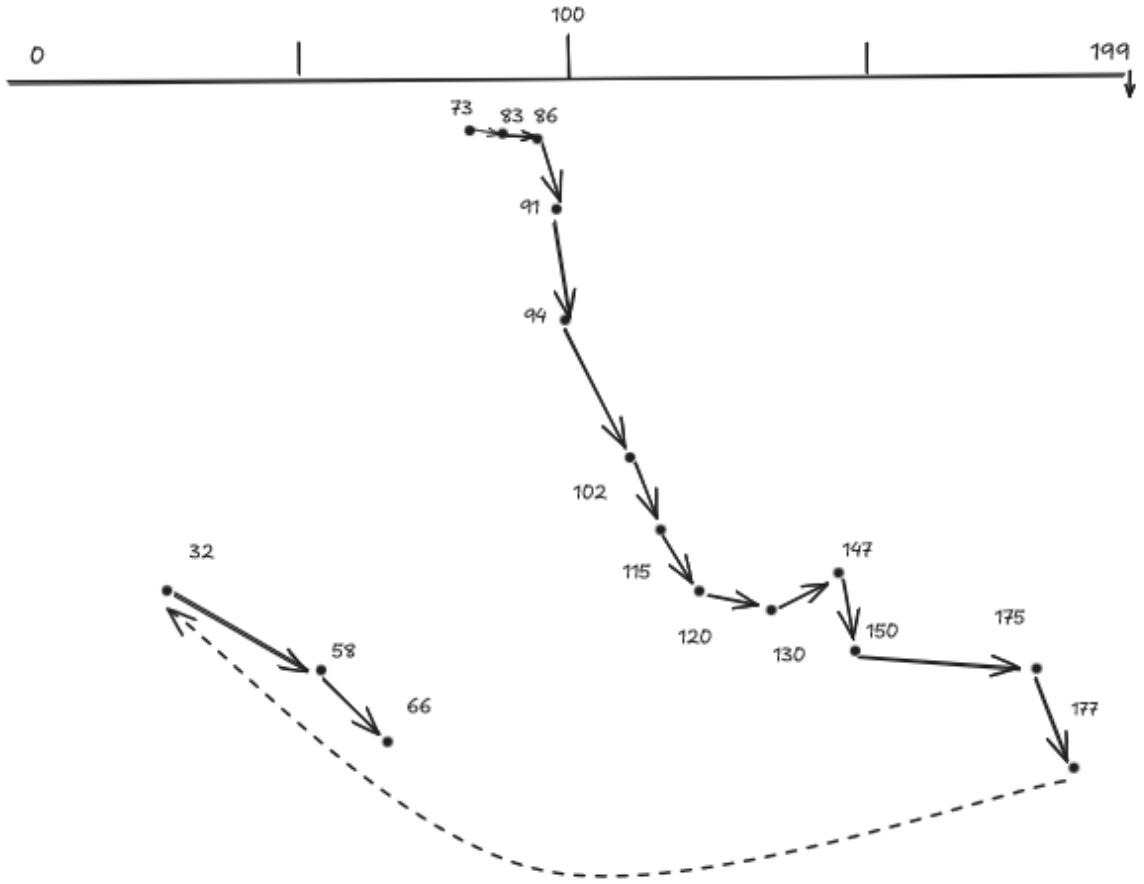


movimientos = 94 + 145 = 239

e)


0
100
199
73
83
86
91
94
102
115
120
130
147
150
175
177
199
0
32
58
66

f)



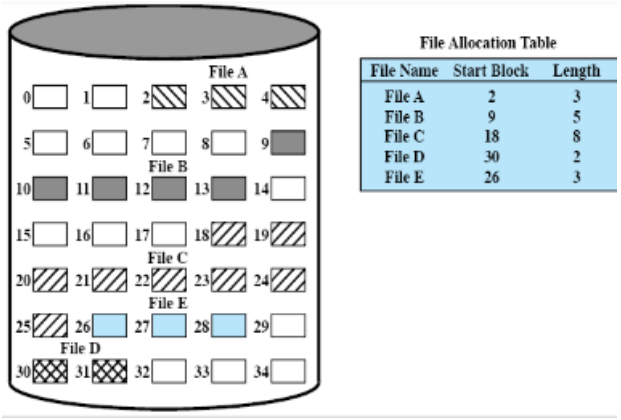
Todos menos el FCFS , ya que si siguen apareciendo requerimientos antes de cambiar el sentido, puede ocurrir que nunca sean atendidos aquellos que estén en el sentido siguiente. Y en el caso del SSTF si siguen apareciendo requerimientos cerca de el lector y lo alejan de aquellos requerimientos que no fueron atendidos. Aquellos que estén mas lejos nunca serán atendidos.

Administración de archivos

17)

A)

Continua:



File Allocation Table

File Name	Start Block	Length
File A	2	3
File B	9	5
File C	18	8
File D	30	2
File E	26	3

- Conjunto continuo de bloques son utilizados
- Se requiere una pre-asignación
  - Se debe conocer el tamaño del archivo durante su creación
- File Allocation Table (FAT) es simple
  - Solo una entrada que incluye bloque de inicio y longitud
- El archivo puede ser leído con una única operación
- Puede existir fragmentación externa
  - Compactación
- Problemas de la técnica
  - Encontrar bloques libres continuos en el disco
  - Incremento del tamaño de un archivo

Enlazada:

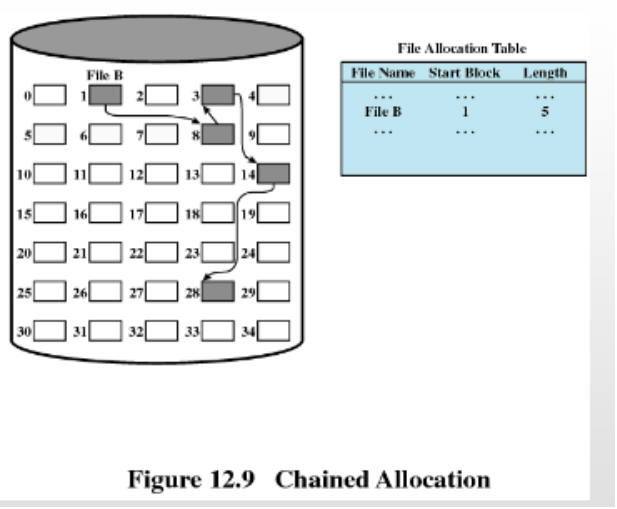


Figure 12.9 Chained Allocation

- Asignación en base a bloques individuales
- Cada bloque tiene un puntero al próximo bloque del archivo
- File allocation table
  - Única entrada por archivo: Bloque de inicio y tamaño
- No hay fragmentación externa
- Útil para acceso secuencial (no random)
- Los archivos pueden crecer bajo demanda
- No se requieren bloques contiguos
- Se pueden consolidar los bloques de un mismo archivo para garantizar cercanía de los bloques de un mismo dispositivo

Indexada:

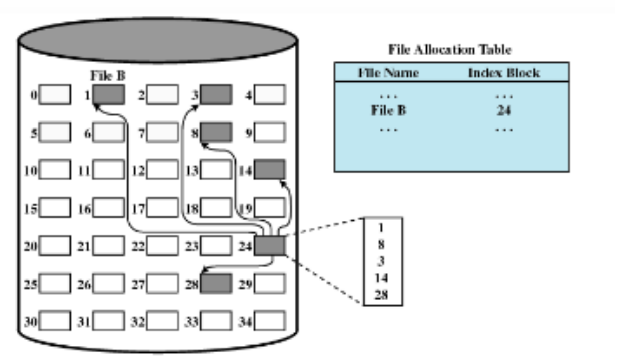


Figure 12.11 Indexed Allocation with Block Portions

- La FAT contiene un puntero al bloque índice
- El bloque índice no contiene datos propios del archivo, sino que contiene un índice a los bloques que lo componen
- Asignación en base a bloques individuales
- No se produce Fragmentación Externa
- El acceso “random” a un archivo es eficiente
- File Allocation Table
  - Única entrada con la direccón del bloque de índices (index node / i-node)
- Variante: asignación por secciones
  - A cada entrada del bloque índice se agrega el campo longitud
  - El índice apunta al primer bloque de un conjunto almacenado de manera contigua
- Variante: niveles de indirección
  - Existen bloques directos de datos
  - Otros bloques son considerados como bloque índices (apuntan a varios
    bloques de datos)
  - Puede haber varios niveles de indirección

B)

Asignación Contigua

Ventajas:

- **Acceso rápido:** Los archivos pueden ser accedidos rápidamente mediante cálculos aritméticos simples, ya que todos los bloques están en ubicaciones consecutivas.
- **Fácil implementación:** Simple de gestionar, especialmente para archivos pequeños o de tamaño fijo.
- **Menor sobrecarga:** No requiere estructuras adicionales para localizar bloques.

Desventajas:

- **Fragmentación externa:** El espacio libre puede quedar dividido en fragmentos pequeños, dificultando la asignación de nuevos archivos.
- **Dificultad para crecer:** Si el archivo necesita más espacio, no siempre hay bloques contiguos disponibles.
- **Reubicación costosa:** Ampliar o mover archivos puede ser lento, ya que implica copiar datos.

Asignación Enlazada

Ventajas:

- **Sin fragmentación externa:** Los bloques no necesitan ser contiguos.
- **Flexibilidad:** Es más fácil gestionar archivos dinámicos o en crecimiento.
- **Eficiencia en el uso de espacio:** Aprovecha mejor el almacenamiento disponible.

Desventajas:

- **Acceso lento:** Requiere seguir punteros, lo que aumenta la latencia de lectura/escritura.
- **Mayor sobrecarga:** Los punteros ocupan espacio adicional.
- **Propenso a errores:** Si un puntero se corrompe, parte del archivo puede perderse.

Asignación Indexada

Ventajas:

- **Acceso directo:** Los índices permiten encontrar bloques rápidamente sin necesidad de recorrerlos en orden.
- **Versatilidad:** Maneja bien archivos grandes y pequeños.
- **Flexibilidad para expansión:** Los índices pueden referirse a bloques dispersos.

Desventajas:

- **Espacio extra para el índice:** Requiere almacenamiento adicional para las tablas de índices.
- **Complejidad de implementación:** Es más complicado de desarrollar y mantener.
- **Limitación de tamaño:** Si el índice es pequeño, podría no soportar archivos grandes sin una estructura de índice adicional (por ejemplo, índices multinivel).

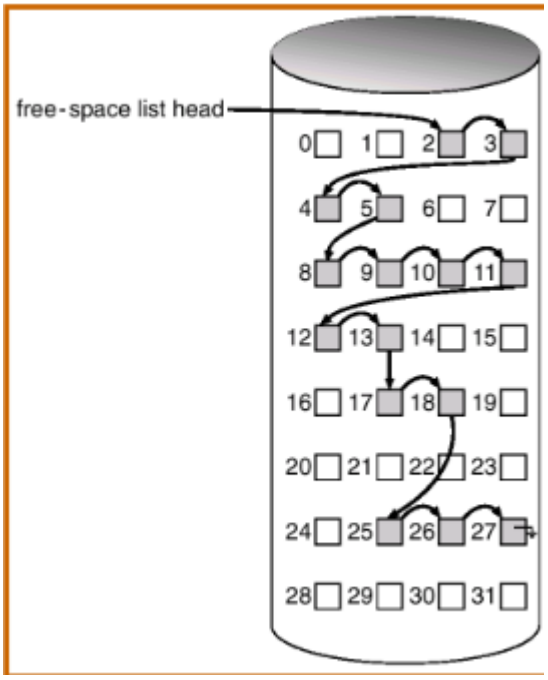
18)

Tabla de bits:

- Tabla (vector) con 1 bit por cada bloqueo de disco
- Cada entrada:
  - 0 = Bloque libre 1 = bloque en uso
- Ventaja
  - Fácil de encontrar un bloque o grupo de bloques libres
- Desventaja
  - Tamaño del vector en memoria
    - Tamaño de disco bytes / tamaño bloque en sistema

Lista ligada:

- Se tiene un puntero al primer bloque libre
- Cada bloque libre tiene un puntero al siguiente bloque libre
- Ineficiente para la búsqueda de bloques libres
    - Hay que realizar varias operaciones de E/S para obtener un grupo libre
- Problemas con la pérdida de un enlace
- Difícil encontrar bloques libres consecutivos

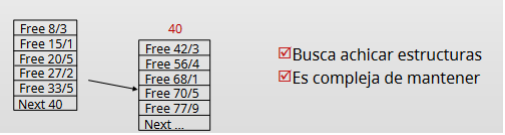


Indexación:

- Variante de “bloques libres encadenados”
- El primer bloque libre contiene las direcciones de N bloques libres
- Las N-1 primeras direcciones son bloques libres
- La N-ésima dirección referencia otro bloque con N direcciones de bloques libres

Recuento:

- Variante de Indexación
- Esta estrategia considera las situaciones de que varios bloques contiguos pueden ser solicitados o liberados a la vez (en especial con asignación contigua)
- En lugar de tener N direcciones libres (índice) se tiene:
  - La dirección del primer bloque libre
  - Los N bloques libres contiguos que le siguen. (# bloque, N siguientes bloques libres)



B)

Tabla de Bits

Ventajas:

- **Acceso rápido:** Permite localizar bloques libres eficientemente usando operaciones de bit a nivel de hardware.
- **Compacta:** Requiere poco espacio, ya que cada bloque se representa por un único bit.
- **Facilidad de implementación:** Es sencilla de interpretar y manipular.

Desventajas:

- **Búsqueda costosa:** Si hay muchos bloques, localizar espacios contiguos puede ser lento.
- **Ineficiencia en discos grandes:** Con discos de gran tamaño, recorrer la tabla completa puede ser poco práctico.

Lista Ligada

Ventajas:

- **Sin necesidad de compactación:** Los bloques libres se enlazan, por lo que no hay problemas de fragmentación externa.
- **Flexibilidad:** Facilita la gestión de bloques dispersos y permite un uso eficiente del espacio disponible.
- **Búsqueda eficiente:** Cada nodo de la lista contiene información específica del bloque.

Desventajas:

- **Sobrecarga adicional:** Requiere almacenar punteros para enlazar los bloques libres, ocupando espacio adicional.
- **Acceso secuencial:** Puede ser lento para localizar grandes áreas contiguas de espacio libre.
- **Propenso a errores:** Un puntero corrupto puede perder parte de la lista.

Agrupamiento

Ventajas:

- **Optimización de búsqueda:** Agrupa bloques libres, lo que permite localizar rápidamente grandes áreas de espacio libre.
- **Menor sobrecarga:** Sólo almacena punteros al siguiente grupo, reduciendo la cantidad de información adicional.
- **Escalable:** Maneja bien discos de gran tamaño.

Desventajas:

- **Complejidad adicional:** La implementación puede ser más complicada que otros métodos.
- **Menos eficiente para bloques individuales:** Recuperar bloques libres dispersos puede ser más costoso.

Recuento

Ventajas:

- **Simplicidad:** Almacena la dirección inicial y el número de bloques libres consecutivos, reduciendo la cantidad de información que gestionar.
- **Rápido para espacios contiguos:** Es eficiente si los bloques libres están agrupados.

- **Bajo consumo de memoria:** Requiere menos almacenamiento que otros métodos.

Desventajas:

- **Fragmentación externa:** Puede dificultar el uso del espacio si los bloques libres están dispersos.
- **Menos flexible:** No maneja bien espacios no contiguos.
- **Mayor complejidad al crecer:** Si se eliminan bloques en el medio, la estructura debe actualizarse frecuentemente.

19)

Bloque de disco = 1Kib = 1024 bytes

Dirección = 32 bits = 4 bytes

1024 / 4 = 256 direcciones en 1 bloque

10 X 1024 = 10240

Direcciones directas

cada bloque tiene 1024 bytes

Entonces 1024 × 10 = 10240

Direccionamiento indirecto simple

- 1 dirección apunta a un bloque que contiene **256 direcciones**.
- Tamaño aportado:

  256×1024= 262.144 bytes

Direccionamiento indirecto doble:

- 1 dirección apunta a un bloque que contiene **256 direcciones**.
- Cada una de estas direcciones apunta a otro bloque con **256 direcciones**.
- Tamaño aportado:

  256×256×1024=67.108.864 bytes.

Direccionamiento indirecto triple:

- 1 dirección apunta a un bloque que contiene **256 direcciones**.
- Cada una de estas direcciones apunta a otro bloque con **256 direcciones**.
- Cada una de estas, a su vez, apunta a otro bloque con **256 direcciones**.
- Tamaño aportado:

  256×256×256×1024=17.179.869.184 bytes.

Tamaño maximo=10.240+262.144+67.108.864+17.179.869.184=
17.247.250.432bytes.

17.247.250.432 / 2^30 = 16.06 Gib