Resumen PN – ISO

P4 – Procesos

Algoritmos

FCFS: Cuando hay que elegir un proceso para ejecutar, se selecciona el más viejo. No favorece a ningún tipo de procesos, pero en principio podríamos decir que los CPU Bound terminan al comenzar su primera ráfaga, mientras que los I/O Bound no.

SJF: Política no apropiativa que selecciona el proceso con la ráfaga más corta. Calculo basado en la ejecución previa. Procesos cortos se colocan delante de procesos largos. Los procesos largos pueden sufrir inanición.

RR: Política basada en un reloj. Cuando un proceso es expulsado de la CPU es colocado al final de la RQ y se selecciona otro (FIFO circular). Existe un contador que indica las unidades de CPU en las que el proceso se ejecutó. Cuando el mismo llega a 0, el proceso es expulsado. El contador puede ser global o local (PCB).

- TV: El contador se inicializa en Q cada vez que un proceso es asignado a la CPU. Es el más utilizado.
- TF: El contador se inicializa en Q cuando su valor es 0. Se puede ver como un valor de Q compartido entre los procesos.

STRF: Version preemptive de SJF. Selecciona el proceso al cual le resta menos tiempo de ejecución en su siguiente ráfaga. Favorece a procesos I/O Bound.

Prioridades: Cada proceso tiene un valor que representa su prioridad. Se selecciona el proceso de mayor prioridad de los que se encuentran en la RQ. Existe una RQ por cada nivel de prioridad. Puede ser apropiativo o no. Los procesos de baja prioridad pueden sufrir inanición. Solución: permitir a un proceso cambiar su prioridad durante su ciclo de vida (Aging o Penalty).

Tiempos:

- Retorno: tiempo que transcurre entre que el proceso llega al sistema hasta que completa su ejecución (fin – (inicio+1)).
- Espera: tiempo que el proceso se encuentra en el sistema esperando, es decir el tiempo que pasa sin ejecutarse (retorno – CPU).

Criterios de desempate:

- Orden de llegada de los procesos.
- PID de los procesos.

P5 – Memoria

Traducción de direcciones

Dirección lógica a física:

N pag	= dir lógica DIV tam pag
Desplazamiento	= dir lógica MOD tam pag
Inicio marco	= inicio marco en el que está N pag
Dir física	= inicio marco + desplazamiento

Dirección física a lógica:

N marco	= dir física DIV tam marco
Desplazamiento	= dir física MOD tam marco
N pag	= N pag que esté en N marco
Dir lógica	= (N pag * tam pag) + desplazamiento

Ejemplo:

Tamaño página = 512 bytes

Tamaño dirección = 1 byte

Tamaño proceso = 2000 bytes

Tabla de páginas del proceso	
# Página	# Marco/Frame
0	3
1	5
2	2
3	6

Memoria Principal (MP)	
# Marco	Bytes inicio..fin
0	0..511
1	512..1023
2	1024..1535
3	1536..2047
4	2048..2559
5	2560..3071

Dir Lógica	DIV (512) N° Pag	MOD (512) Desplazamiento	Marco (base)	Dir. Física
35	0	35	M 3 (1536)	1536 + 35 = 1571
512	1	0	M 5 (2560)	2560 + 0 = 2560
2051	4	3	error	
0	0	0	M 3 (1536)	1536 + 0 = 1536
1325	2	301	M 2 (1024)	1024 + 301 = 1325
602	1	90	M 5 (2560)	2560 + 90 = 2650

Dir Fisica	DIV (512) N° Marco	MOD (512) Desplazamiento	Página (base)	Dir. Logica
509	0	509	error	
1500	2	476	P 2 (1024)	1024 + 476 =1500
0	0	0	error	
3215	6	143	P 3 (3072)	3072 + 143 = 3215
2014	3	478	P 0 (0)	0 + 478 = 478
2000	3	464	P 0 (0)	0 + 464 = 464

Direcciones

Ejemplo:

- Se dispone de un espacio de direcciones virtuales de 32 bits.
- Cada dirección referencia 1 byte.
- El tamaño de página es de 512 KiB.
- Se dispone de 256 MiB de memoria real.
- El tamaño de cada entrada en la tabla de páginas es 2 Kb.

Cantidad de direcciones:

$$2^{32} = 4.294.967.296$$

Tamaño máximo de un proceso:

$$\text{cant direcciones} * \text{tam referencia} = 2^{32} * 1 \text{ byte} = 2^{32} \text{ bytes} = 4\text{GiB}$$

Número máximo de páginas que puede tener un proceso:

$$\text{tam max proceso} / \text{tam pag} = 4194304 \text{ KiB} / 512 \text{ KiB} = 8192 \text{ páginas}$$

Número de marcos que puede haber:

$$\text{memoria} / \text{tam pag} = 262144 \text{ KiB} / 512 \text{ KiB} = 512 \text{ frames}$$

Tamaño máximo de la tabla de cada proceso:

$$\text{cant max pag} * \text{tam entrada pag} = 8192 * 2 \text{ Kb} = 16384 \text{ Kb}$$

Reparto de marcos (asignación fija)

Marcos = 40

Reparto equitativo:

Equitativo			
Marcos de proceso: total de marcos / cant procesos			
Proceso	Paginas	Marcos	
1	15	10	
2	20	10	
3	20	10	
4	8	10	

Reparto proporcional:

Proporcional			
Marcos de proceso: pag de proceso / total de pag * total de marcos			
Proceso	Paginas	Marcos	
1	15	9	15 / 63 * 40
2	20	13	20 / 63 * 40
3	20	13	20 / 63 * 40
4	8	5	8 / 63 * 40

Algoritmos

Óptimo: Selecciona la página cuya próxima referencia se encuentra más lejana a la actual.

FIFO: La página más vieja en la memoria es reemplazada.

FIFO segunda chance: Se usa el bit de referencia. Cuando la página se carga en memoria, el bit R se pone a 0. Cuando la página es referenciada el bit R se pone en 1. La víctima se busca en orden FIFO. Se selecciona la primera página cuyo bit R esté en 0. Mientras se busca la víctima cada bit R que tiene el valor 1 de las que podrían haber sido seleccionadas (no todas las que tengan el bit en 1) se cambia a 0.

LRU: Reemplaza la página que no fue referenciada por más tiempo.

Ejemplo: P5EJ22

FIFO segunda chance:

SC	1	2	15	4	6	2	1	5	6	10	4	6	7	9	1	6	12	11	12	2	3	1	8	1	13	14	15	3	8
1	1	1	1	1	1	1	1*	1	1	1	4	4	4	4	4	4	12	12	12*	12*	12*	12*	12	12	12	12	15	15	15
2		2	2	2	2	2*	2*	2	2	2	2	2	7	7	7	7	7	11	11	11	11	11	8	8	8	8	8	3	3
3			15	15	15	15	15	5	5	5	5	5	5	9	9	9	9	9	9	2	2	2	2	2	13	13	13	13	8
4				4	4	4	4	4	4	10	10	10	10	10	1	1	1	1	1	1	3	3	3	3	3	14	14	14	14
5					6	6	6	6	6*	6	6	6*	6*	6*	6*	6*	6	6	6	6	6	1	1	1*	1*	1*	1	1	1
PF=22	X	X	X	X	X			X		X	X		X	X	X		X	X		X	X	X	X		X	X	X	X	X
Qeue	1	1	1	1	1	1	1*	~~1~~	4	~~4~~	~~6~~	2	~~2~~	~~5~~	~~10~~	6*	~~6~~	9	9	~~9~~	~~1~~	~~6~~	~~12~~	2	~~2~~	~~3~~	~~1~~	~~8~~	~~13~~
		2	2	2	2	2*	2*	~~2~~	6*	6*	2	5	5	10	6*	4	~~4~~	1	1	1	6	12*	~~11~~	3	3	1*	~~12~~	13	14
			15	15	15	15	15	~~15~~	1	1	5	10	10	6*	4	7	7	6	6	6	12*	11	2	1*	1*	12	8	14	1
				4	4	4	4	4	2	2	10	6*	6*	4	7	9	9	12	12*	12*	11	2	3	12	12	8	13	1	15
					6	6	6	6	5	5	6	4	4	7	9	1	1	11	11	11	2	3	1	8	8	13	14	15	3
								1		10	4		7	9	1		6			2	3	1	12		13	14	1	3	8
								2									12						8				15		
								5																					

LRU:

LRU	1	2	15	4	6	2	1	5	6	10	4	6	7	9	1	6	12	11	12	2	3	1	8	1	13	14	15	3	8
1	1	1	1	1	1	1	1	1	1	1	1	1	7	7	7	7	7	11	11	11	11	11	8	8	8	8	8	3	3
2		2	2	2	2	2	2	2	2	2	4	4	4	4	4	4	12	12	12	12	12	12	12	12	13	13	13	13	13
3			15	15	15	15	15	5	5	5	5	5	5	9	9	9	9	9	9	2	2	2	2	2	2	14	14	14	14
4				4	4	4	4	4	4	10	10	10	10	10	1	1	1	1	1	1	3	3	3	3	3	3	15	15	15
5					6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	1	1	1	1	1	1	1	8
PF=22	X	X	X	X	X			X		X	X		X	X	X		X	X		X	X	X	X		X	X	X	X	X

Descarga asincrónica

El sistema operativo reserva uno o varios marcos para la descarga asincrónica de páginas. Cuando es necesario descargar una página modificada:

- La página que provocó el fallo se coloca en un frame designado a la descarga asincrónica.
- El SO envía la orden de descargar asincrónicamente la página modificada mientras continúa la ejecución de otro proceso.

- El frame de descarga asincrónica pasa a ser el que contenía a la página víctima que ya se descargó correctamente.

Ejemplos (no están corregidos, pero creo que están bien):

1) Suponga un SO con administración de la memoria virtual mediante paginación por demanda. Si la cantidad de marcos disponibles es 4 y se debe reservar un marco para la descarga asincrónica de páginas, complete el gráfico correspondiente a la asignación de páginas a marcos e indique la cantidad de fallos de pagina producidos para la siguiente secuencia de paginas :

{1, 2, 4, 2, 1^M, 3, 4^M, 1^M, 6, 2^M, 1^M, 4^M, 7, 5, 3}

con el siguiente algoritmo: FIFO CON SEGUNDA CHANCE

	1	2	4	2	1M	3	4M	1M	6	2M	1M	4M	7	5	3
1	1	1	1	1	1M*	1M	1M						7	7	7
2		2	2	2*	2*	2	2	1M	1M	1M	1M*	1M	1M	1M	3
3			4	4	4	3	3	3	6	6	6	4M	4M	4M	4M
4							4	4	4	2M	2M	2M	2M	5	5
PF	X	X	X			X	X	X	X	X		X	X	X	X
Q	~~1M*~~	~~2*~~	~~4~~	~~1M~~	~~2~~	~~3~~	~~4~~	~~1M*~~	~~6~~	~~2M~~	~~1M~~	~~4M~~	7	5	3

1. Dado un sistema cuya memoria es administrada a través del sistema de memoria virtual. Siendo la siguiente la lista de referencias a páginas: 1 2 3 1M 3M 4 2 5 2 7 2M 3 4 5 1 6 8 9. Simule el reemplazo de páginas para un algoritmo LRU con 4 frames, de los cuales se reserva 1 para la descarga asincrónica y calcule la cantidad de Page Faults.

	1	2	3	1M	3M	4	2	5	2	7	2M	3	4	5	1	6	8	9
1	1	1	1	1M	1M	1M	1M	5	5	5	5	3	3	3	1	1	1	9
2		2	2	2	2	4	4	4	4	7	7	7	4	4	4	6	6	6
3			3	3	3M	3M	3M	3M						5	5	5	8	5
4							2	2	2	2	2M	2M	2M	2M				
PF	X	X	X			X	X	X		X		X	X	X	X	X	X	X

P6 – Entrada Salida

HDD

Capacidad:

platos * caras por plato * pistas (cilindros) por cara * sectores por pista * tam sector

Ejemplo:

- Supongamos un disco con 6 platos, 2 caras útiles, 1500 pistas por cara y 700 sectores por pista de 256 bytes cada uno
- Si queremos calcular la capacidad total del disco, hacemos:

*tamaño_disco = #caras * #pistas_cara * #sectores_pista * tamaño_sector*

(6 * 2) * 1500 * 700 * 256 bytes = 3225600000 bytes = 3,00407 GiB(Gibibytes)

Ocupación:

Ejemplo:

- Supongamos un disco con 6 platos, 2 caras útiles, 1500 pistas por cara y 700 sectores por pista de 256 bytes cada uno
- Si queremos cuantas caras ocupará un archivo de 513 Mibytesalmacenado de manera contigua a partir del primer sector de la primer pista de una cara determinada:
  - Calculamos la capacidad de 1 cara:
    1500 * 700 * 256 bytes = 268800000 bytes
  - Dividimos el tamaño del archivo por la capacidad de una cara:
    513 MiB= 537919488 bytes
    537919488 / 268800000 = 2,00118 → 3 caras

Tiempos:

- 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. Si no se conoce, se asume media vuelta.
- Transfer time (transferencia): tiempo de transferencia del sector (bloque) del disco a la memoria.

Tiempo de acceso:

- Secuencial:

  seek + latency + (transfer bloque * cant bloques)

- Aleatorio:

  (seek + latency + transfer bloque) * cant bloques

Ejemplo:

- Supongamos un disco con 6 platos, 2 caras útiles, 1500 pistas por cara y 700 sectores por pista de 256 bytes cada uno. El disco gira a 12600 RPM , tiene un tiempo de posicionamiento (*seek*) de 2 milisegundos y una velocidad de transferencia de 15 Mib/s (Mebibits por segundo)
- Si queremos saber cuantos milisegundos se tardarían en transferir un archivo **almacenado de manera contigua y aleatoria** de 4500 sectores

- Calculamos los datos que faltan:
  - Latencia:
    12600 vueltas → 1' = 60 s = 60000 ms
    0,5 vueltas → x = 2,3809 ms
  - Transferencia:
    15 Mibits → 1 s = 1000 ms
    256 bytes → x

    Unificamos unidades:
    15728640 bits → 1000 ms
    2048 bits → x = 0,1302 ms

- Datos obtenidos:
  - Seek time: 2 ms
  - Latency time: 2,3809 ms
  - Tiempo transferencia bloque: 0,1302 ms
  - #bloques: 4500 → eventualmente se tienen que calcular
- Resultados:
  - Almacenamiento secuencial:
    seek + latency + tiempo_transferencia_bloque * #bloques
    2 + 2,3809 + 0,1302 * 4500 = 590,2809 ms
  - Almacenamiento aleatorio:
    (seek + latency + tiempo_transferencia_bloque) * #bloques
    (2 + 2,3809 + 0,1302) * 4500 = 20299,95 ms

Ejemplo:

- 7 platos con 2 caras utilizables cada uno.
- 1100 cilindros
- 300 sectores por pista, donde cada sector de es 512 bytes.
- Seek Time de 10 ms
- 9000 RPM.
- Velocidad de Transferencia de 10 MiB/s (Mebibyte por segundos).

a) Calcule la capacidad total del disco.

7 * 2 * 1100 * 300 * 512 bytes = 2365440000 bytes

b) ¿Cuantos sectores ocuparía un archivo de tamaño de 3 MiB (Mebibytes)?

tam_archivo = 3 * 2^{20} bytes ; tam_sector = 512 bytes

3 * 2^{20} = 3145728 bytes / 512 bytes = 6144 sectores

c) Calcule el tiempo de transferencia real de un archivo de 15 MiB (Mebibytes). grabado en el disco de manera **secuencial** (todos sus bloques almacenados de manera **consecutiva**)

Fórmula de Secuencial: seek + lantecy + (transfer_time * #block)

latencia = tiempo que tarda en girar
si no nos dan la latencia hay que calcular media vuelta

latencia: 9000 RPM 9000 vuetas -> 1' = 60" = 60000 ms
0,5 vueltas -> x = 3,33 ms

latencia: 3,33 ms

Transferencia: Tiempo que tarda en transferir cada bloque
10 MiB/s
10 MiB -> 1 s = 1000 ms
512 Bytes (bloque o sector) -> x

Unficamos Unidades:
10 MiB -> 10 * 2^{20} Bytes = 10 * 1024 * 1024 Bytes = 10485760 Bytes
Entonces:

10485760 Bytes -> 1000 ms
512 Bytes -> x = **0,0488 ms <- Tiempo de Transferencia de 1 Bloque**

Archivo = 15 MiB -> bloques? **sector = bloque = 512 Bytes**
15 MiB = 15 * 1024 * 1024 Bytes = 15728640 Bytes
Cantidad de bloques que ocupa el archivo = 15728640 / 512 Bytes = 30720 bloques

10 ms + 3,33 ms + (0,0448 ms * 30720 bloques) = **1389.586 ms <- Tiempo de Transferencia Secuencial**

Algoritmos

El seek time es el parámetro que más influye en el tiempo de acceso al disco. El objetivo es minimizar el movimiento de la cabeza.

Ejemplo:

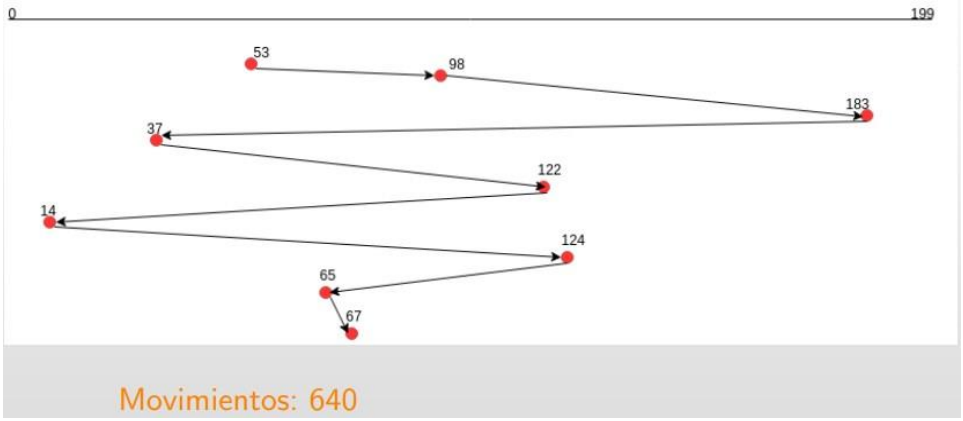
Cantidad de pistas: 200.

Requerimientos: {98, 183, 37, 122, 14, 124, 65, 67}

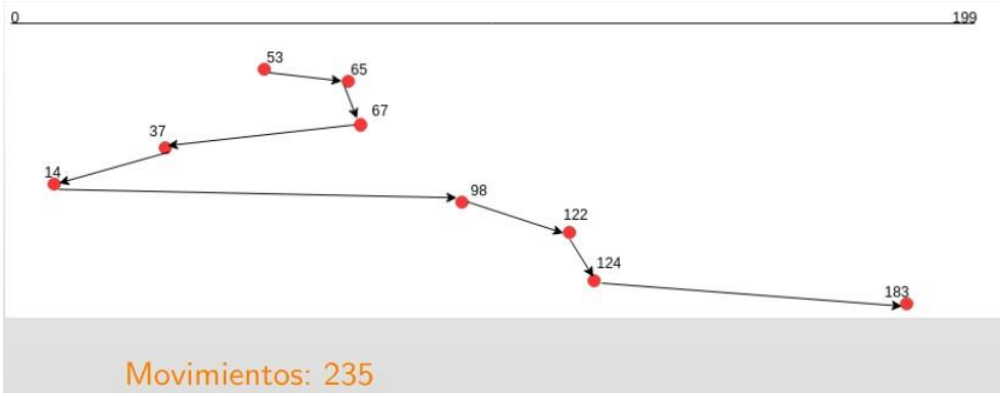
Viene de: 61.

Ubicación actual: 53.

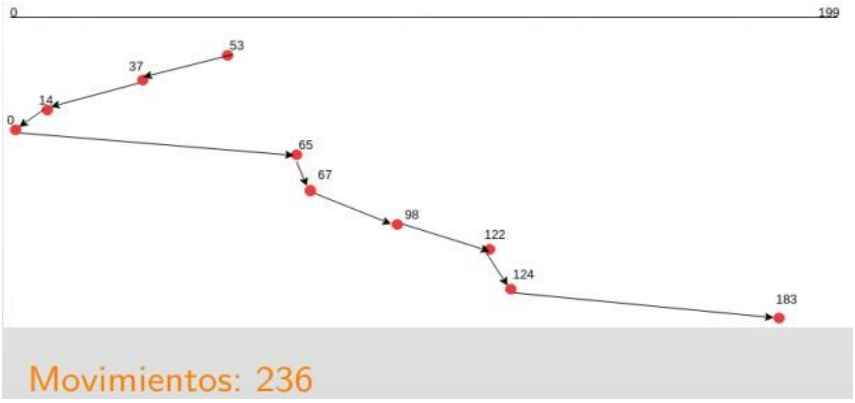
FCFS: atiende los requerimientos por orden de llegada.



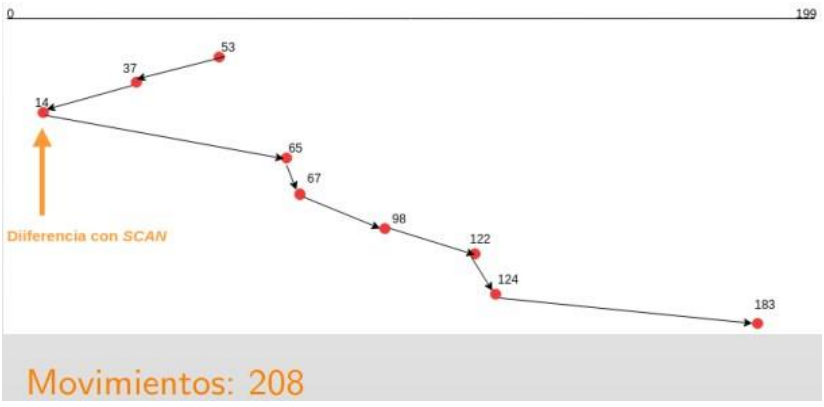
SSTF: selecciona el requerimiento que requiere el menor movimiento del cabezal.



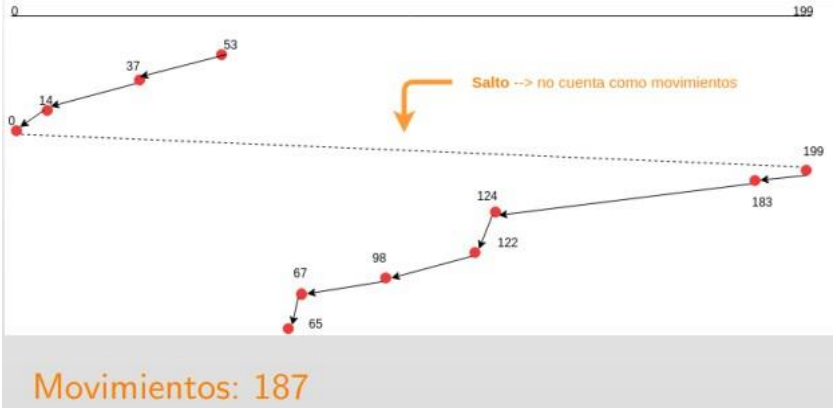
SCAN: barre el disco en una dirección atendiendo los requerimientos pendientes en esa ruta hasta llegar a la última pista del disco y cambia la dirección. Es importante saber en qué pista se está y de que pista se viene para determinar el sentido del cabezal.



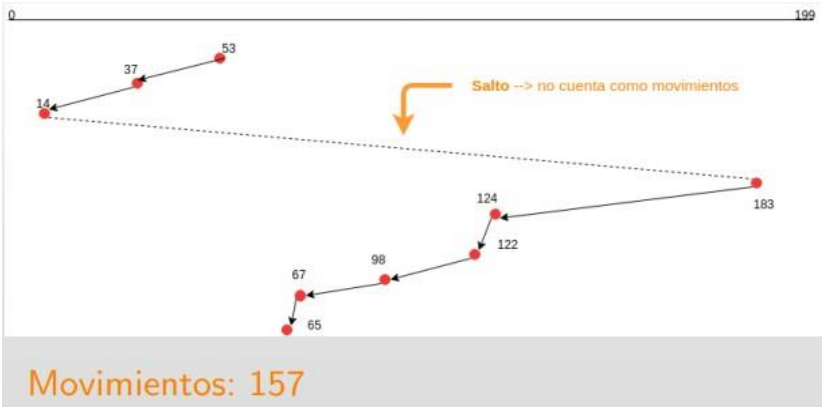
LOOK: se comporta igual que el SCAN, pero no llega hasta la última pista del disco sobre la dirección actual, sino que llega hasta el último requerimiento de la dirección actual. Es importante saber en qué pista se está y de que pista se viene para determinar el sentido del cabezal.



C-SCAN: se comporta igual que el SCAN, pero restringe la atención en un solo sentido. Al llegar a la última pista del disco en el sentido actual, vuelve a la pista del otro extremo (salto -> no se cuentan los movimientos) y sigue barriendo en el mismo sentido.



C-LOOK: se comporta igual que el LOOK, pero restringe la atención en un solo sentido. Al llegar a la última pista de los requerimientos en el sentido actual, vuelve a la primera pista más lejana del otro extremo (salto -> no se cuentan los movimientos) y sigue barriendo en el mismo sentido.



Existen requerimientos especiales que deben atenderse con urgencia. Los fallos de página indican que tienen mayor prioridad con respecto a los requerimientos convencionales, por lo tanto, deben ser atendidos inmediatamente después del requerimiento que se está atendiendo actualmente. La lógica de atención de múltiples PF se maneja según el algoritmo de planificación. En todos los algoritmos, los movimientos utilizados para atender estos requerimientos especiales deben ser contados.

Ejemplo: {10, 40PF, 70PF, 10}

- FCFS: primero se atiende al 40PF y luego al 70PF.
- SSTF: si estoy en la pista 65, primero atiendo al 70PF y luego al 40PF.

Una vez que no existan más requerimientos por PF en la cola, se procede:

- FCFS: en orden FCFS.
- SSTF: en orden SSTF.
- SCAN: con el sentido que determina la atención de los últimos dos requerimientos. Puede cambiar de sentido.
- C-SCAN: con el sentido original.
- LOOK: del mismo modo en que lo hace el SCAN.
- C-LOOK: del mismo modo en que lo hace el C-SCAN.

Ejemplo:

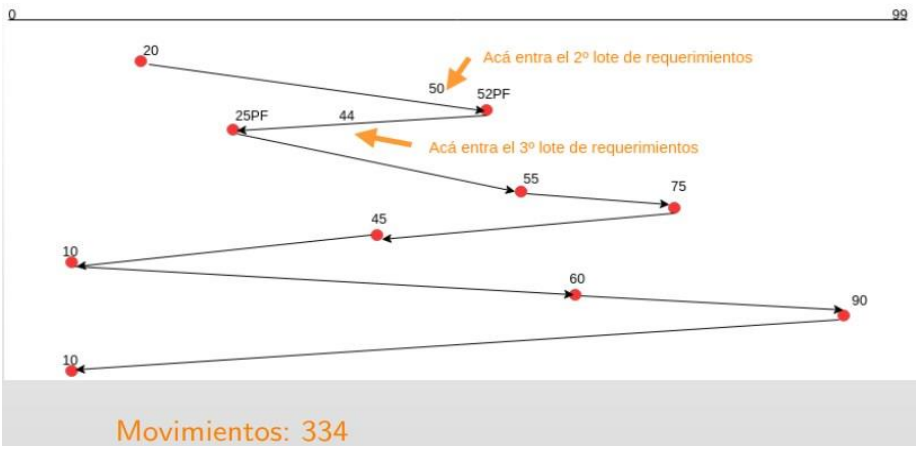
Cantidad de pistas: 100.

Requerimientos: {55, 75, 52PF, 45, 10}. Luego de 30 movimientos {25PF, 60}. Luego de 10 movimientos más {90, 10}.

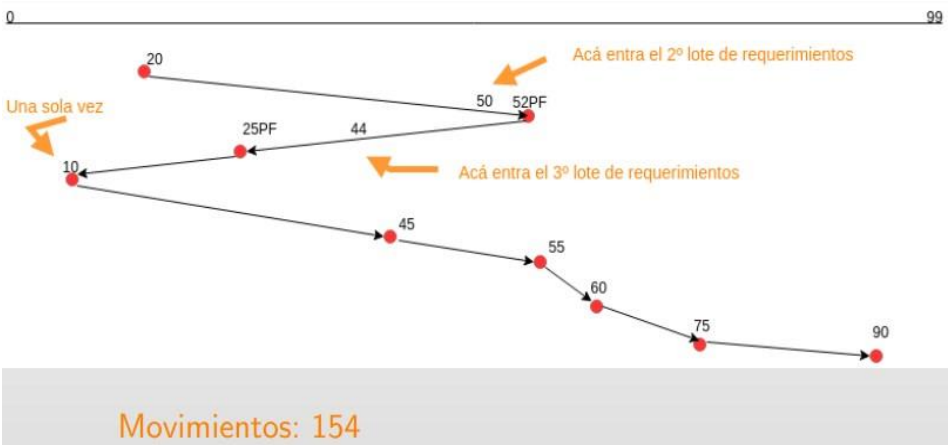
Viene de: 15.

Ubicación actual: 20.

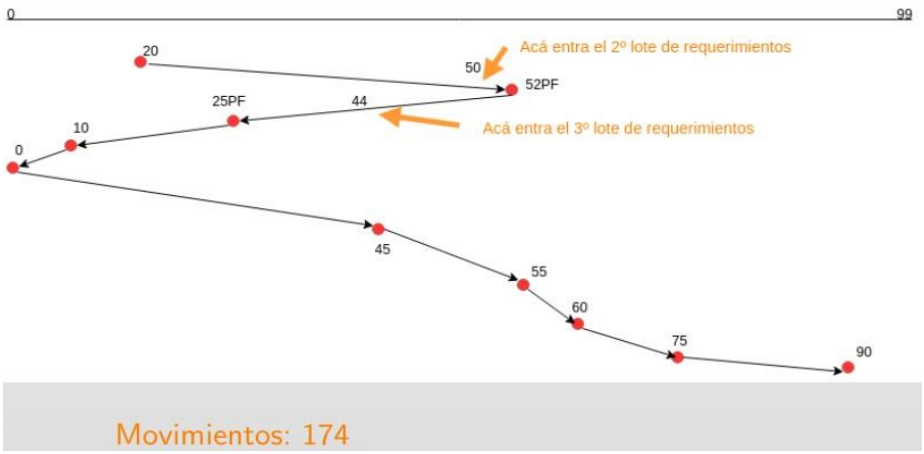
FCFS:



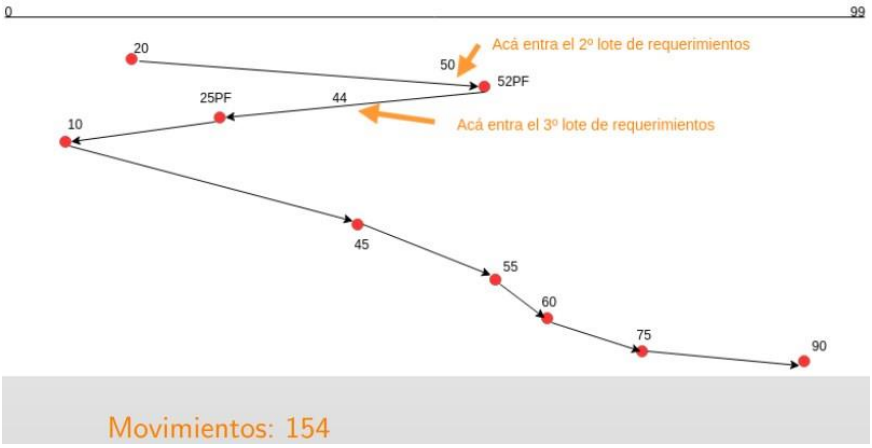
SSTF:



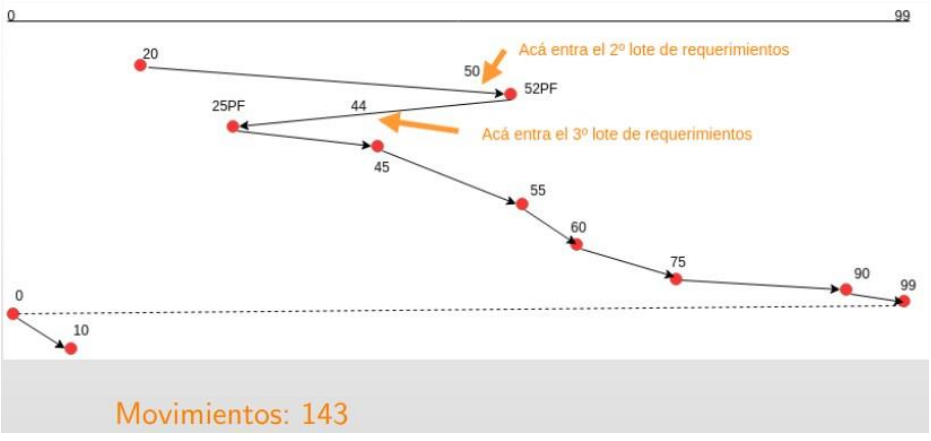
SCAN:



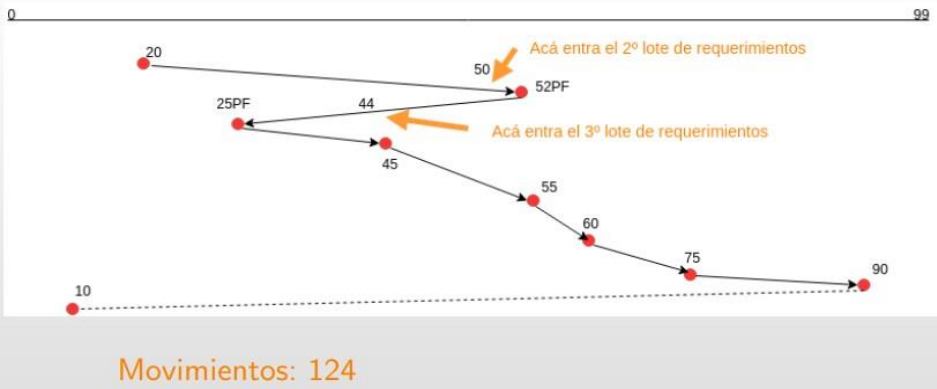
LOOK:



C-SCAN:



C-LOOK:



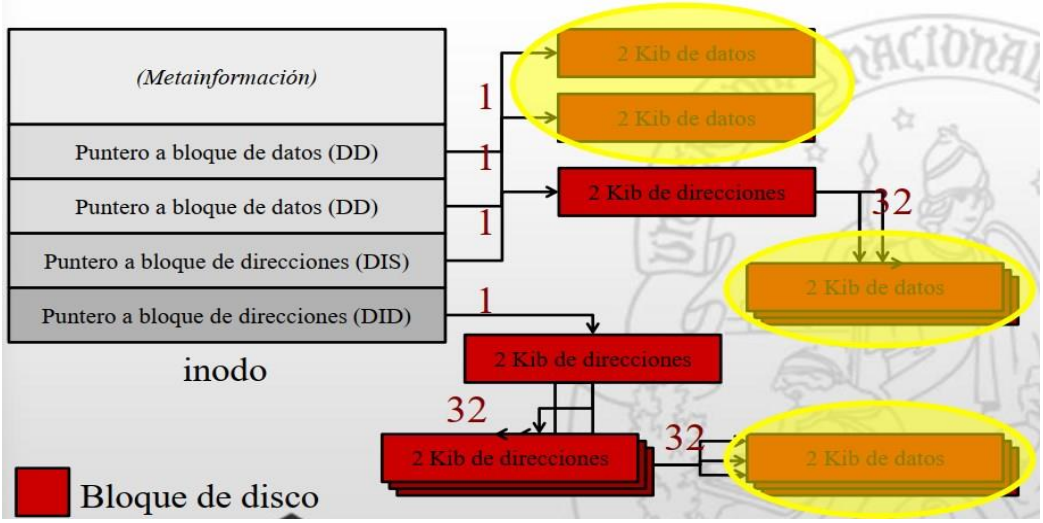
Inodos

Inodo: Estructura auxiliar que permite referenciar los archivos y acceder a ellos. Contiene metainformación del archivo y punteros a los bloques de datos en el disco que conforman el archivo. Se identifican con un número.

Estructura imaginaria (cada dirección es de 64 bits):

4 direcciones a los bloques de datos:

- 2 de direccionamiento directo (DD).
- 1 de direccionamiento indirecto simple (DIS).
- 1 de direccionamiento indirecto doble (DID).



Tamaño del inodo:

tam dirección * direcciones = 64 bits * 4 = 256 bits

Cantidad de inodos que puede tener un bloque de disco (de 2 Kib):

tam bloque / tam inodo = 2 Kib / 256 bits = 8 inodos

Direcciones por bloque de direcciones:

tam bloque / tam dirección = 2 Kib / 64 bits = 32 direcciones

Tamaño máximo de archivo:

2 punteros a bloques de datos (DD) +

1 puntero a bloque de direcciones que apuntan a bloques de datos (DIS) +

1 puntero a bloque de direcciones que apuntan a bloques de direcciones que apuntan a bloques de datos (DID) =

2 * 2 Kib (DD) +

1 * 32 * 2 Kib (DIS) +

1 * 32 * 32 * 2 Kib (DID) =

4 Kib + 64 Kib + 2048 Kib = 2116 Kib = 264.5 KiB