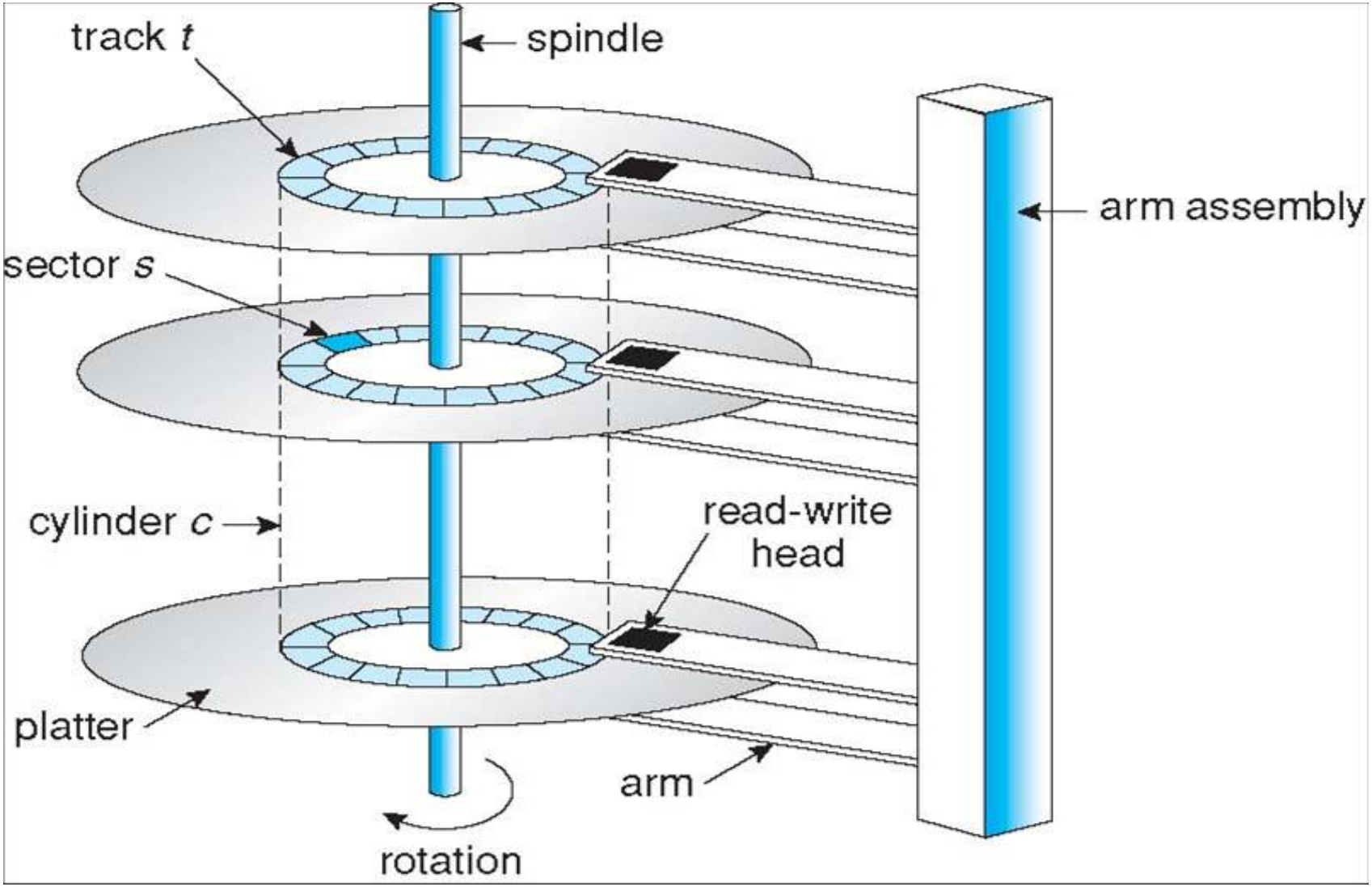
Entrada/Salida

Nano Semelman ⇒ Maximiliano Geier ⇒ Maximiliano Sacco

DC - FCEyN - UBA

Sistemas Operativos, 2c-2014

Repaso de discos



Repaso de discos

Un disco tiene p pistas de s sectores cada una, y h cabezas. El disco gira a R RPM y para moverse entre dos pistas adyacentes le toma t_a y el *seek-time* promedio es t_{seek}. Calcular los siguientes parámetros:

1. T_t: Tiempo de latencia media
2. T_s: Tiempo de lectura de 1 sector
3. T_{min}: Tiempo mínimo para leer el disco completo
4. Tiempo promedio para leer un sector al azar en el disco.

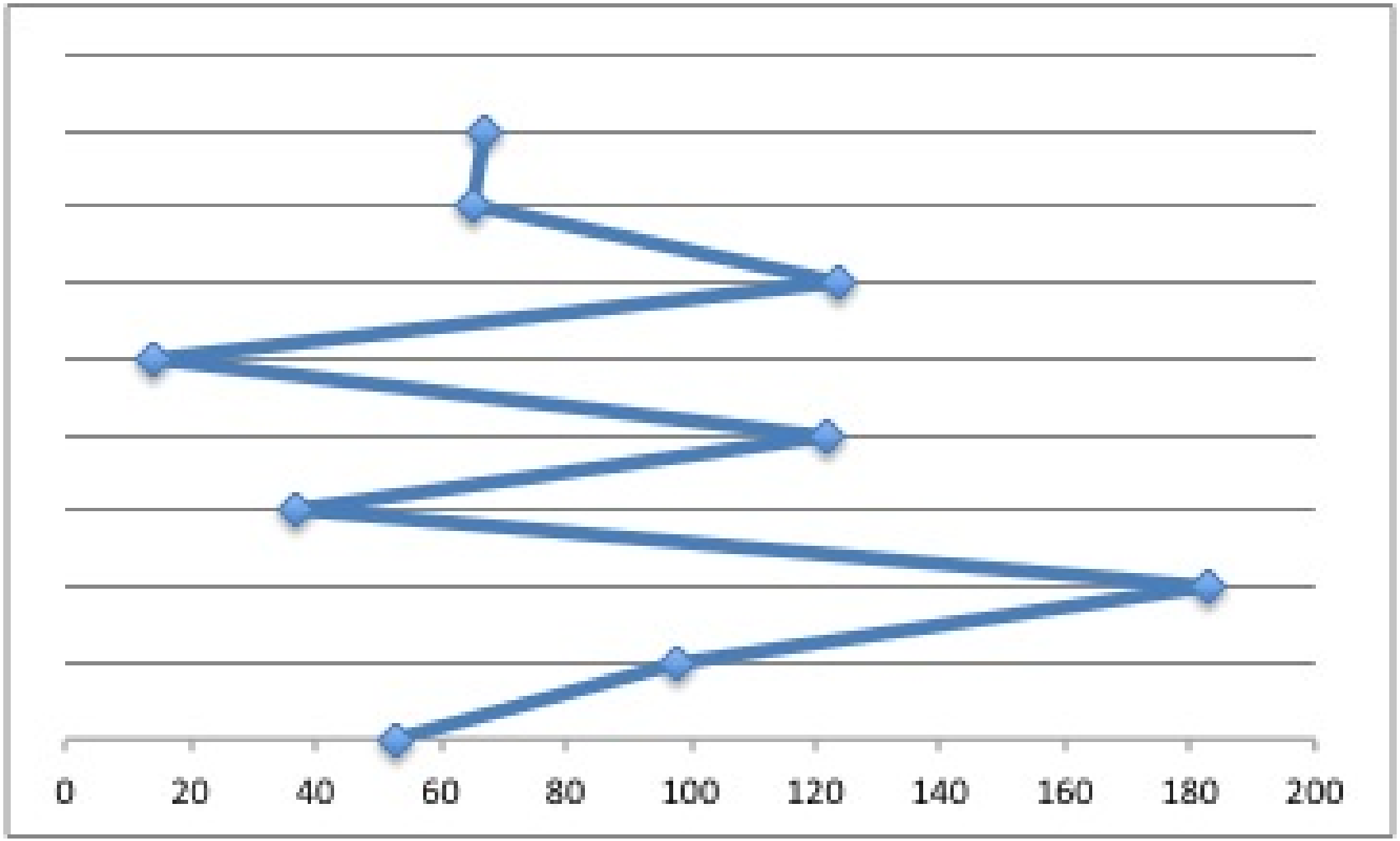
Algoritmos de planificación de discos

Llegan pedidos al driver de disco para los cilindros 98, 183, 37, 122, 14, 124, 65 y 67 en ese orden. El *seek-time* (tiempo para mover un cabezal de una pista a su adyacente) es de 6*msec* por cilindro. Inicialmente, el cabezal del disco se encuentra en el cilindro 53. ¿Cuánto tardará en atenderse las peticiones para cado uno de los siguientes algoritmos de planificación?[1]

1. First-Come, First-Served.
2. SSTF (Shortest Seek Time First)
3. SCAN (Algoritmo del ascensor).
4. SCAN-C

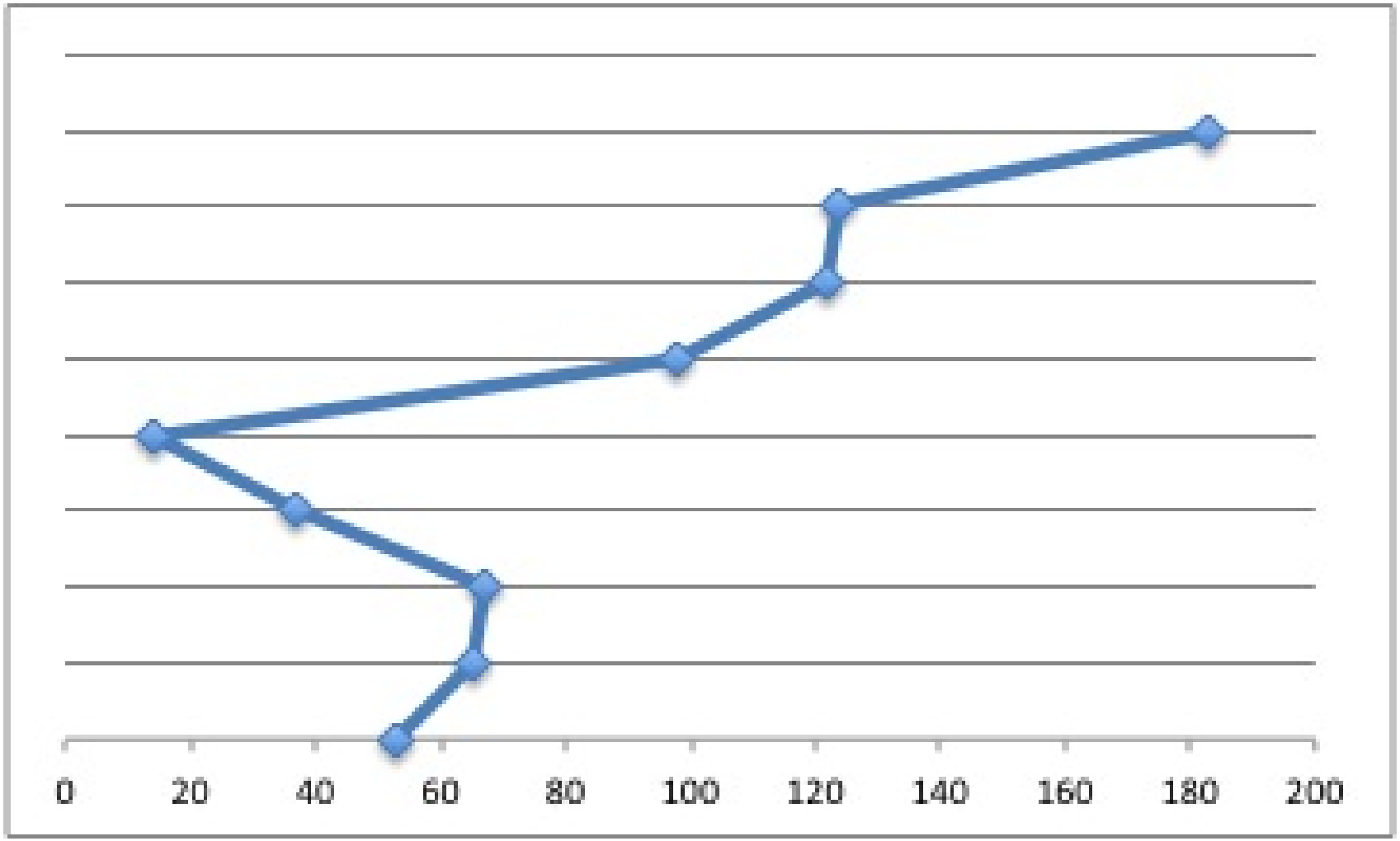
[1] Ejercicio tomado del Tanenbaum.

First Came First Serve



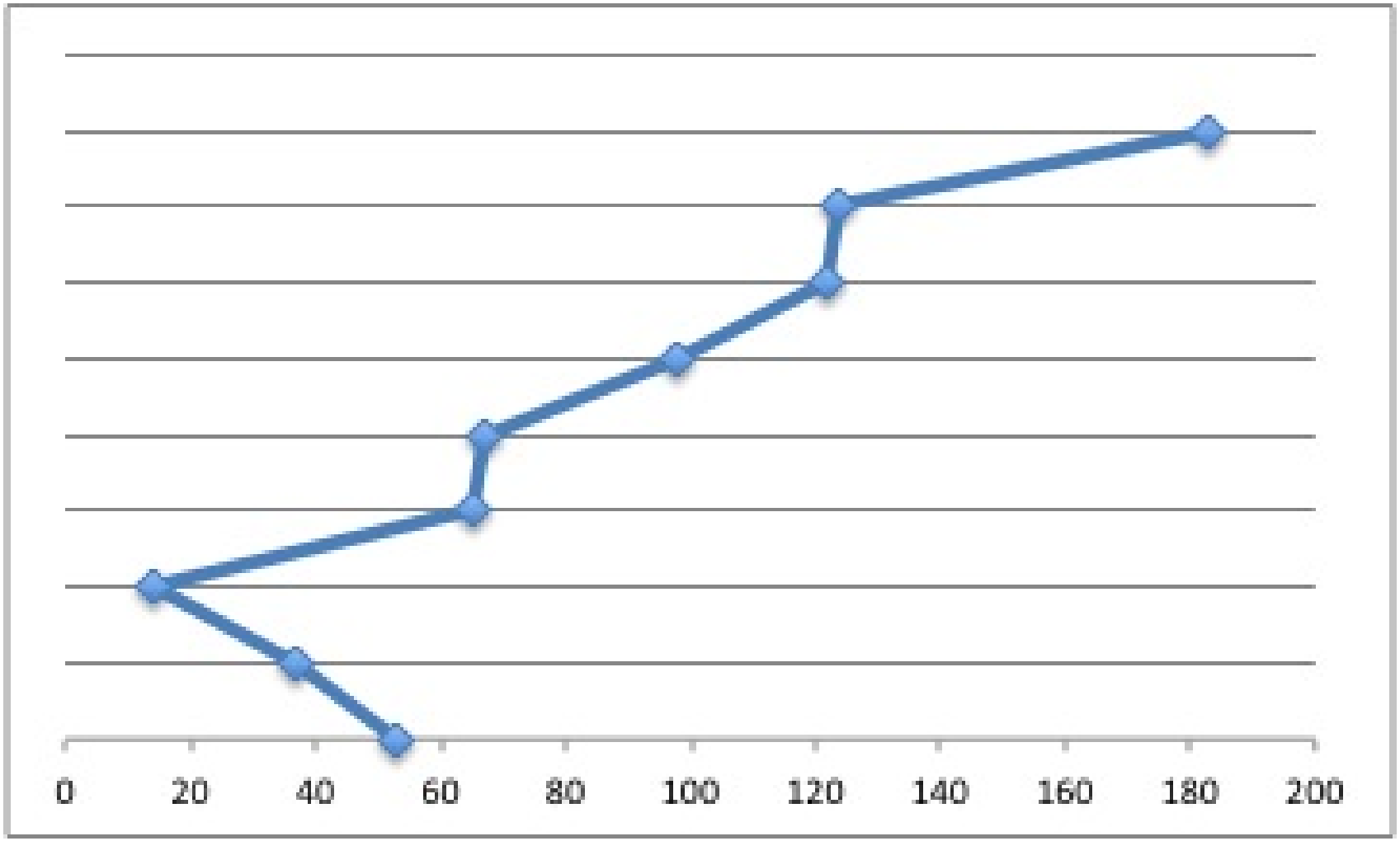
Total de pistas atravesadas: 640

Shortest Seek Time First



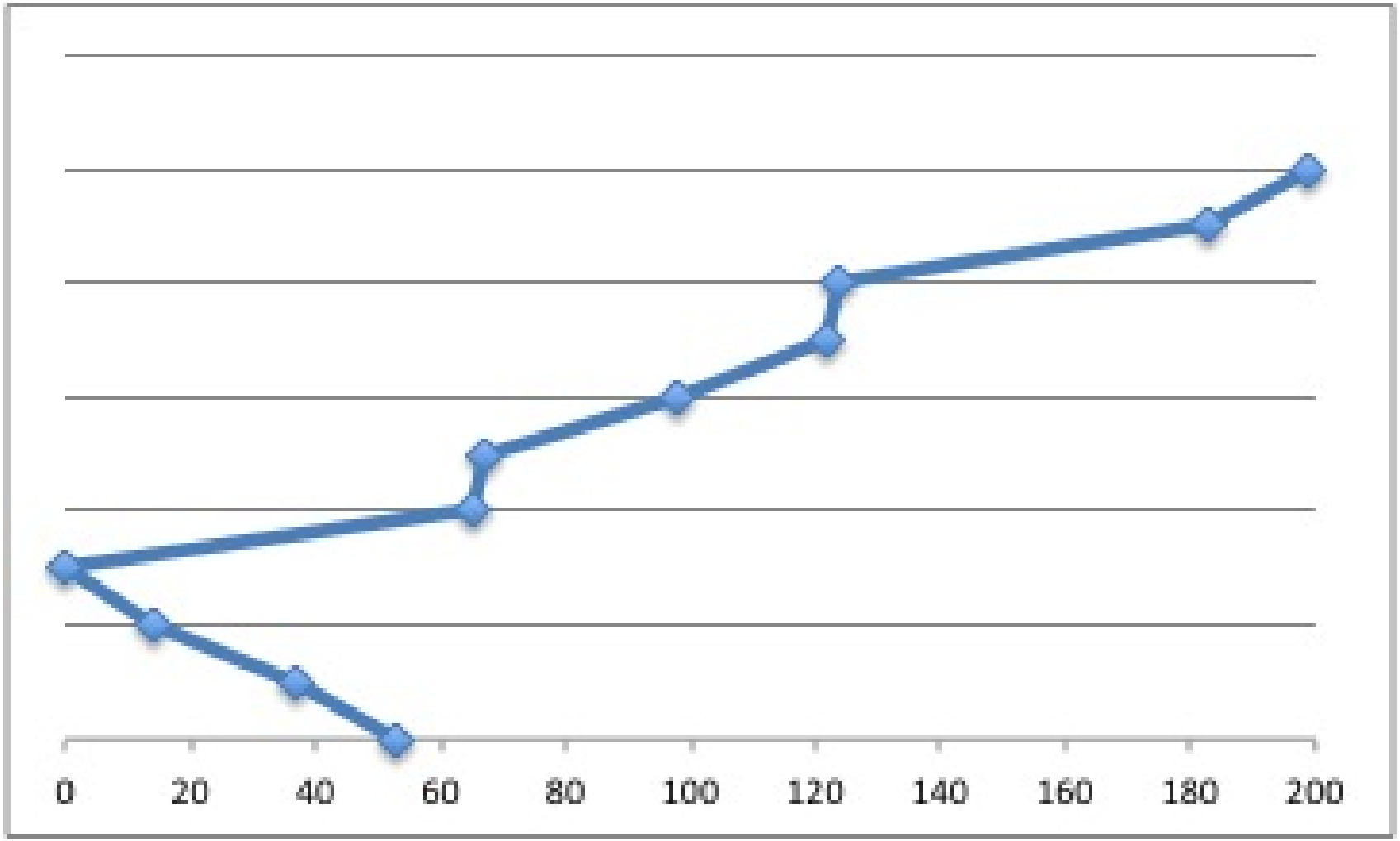
Total de pistas atravesadas: 236

Optimo



Total de pistas atravesadas: 208

SCAN



Total de pistas atravesadas con borde: 253

Total de pistas atravesadas sin borde: 223

Resumen

- FCFS
  - Facil de programar
  - Intrinsecamente justo
  - Peor tiempo promedio
- SSTF
  - Goloso, Muy Bueno tiempo promedio
  - Starvation
  - Alta varianza en tiempo de respuesta
- SACN
  - Algoritmo del ascensor
  - Buen tiempo de promedio (con y sin borde)
  - Alta varianza en tiempo de respuesta (se arregla con SCANC)

Para pensar...

Los algoritmos de scheduling de accesos a disco que vemos en la materia asumen que el mayor costo de la operación de E/S está dado por el *seek time* (tiempo que toma mover el cabezal de una pista a otra pista consecutiva). ¿Qué implicaría si los discos mejoraran de modo tal que dicho tiempo se reduzca en varios órdenes de magnitud mientras que la velocidad de rotación no mejore de la misma manera?

Interfaz de E/S

Un SO provee la siguiente API para operar con un dispositivo de E/S.

int open(int device_id) int close(int device_id)	Abre el dispositivo Cierra el dispositivo
int read(int device_id, int* data) int write(int device_id, int* data)	Lee el dispositivo device_id Escribe el valor en el dispositivo device_id

Todas las operaciones retornan la constante IO_OK si fueron exitosas o la constante IO_ERROR si ocurrió algún error.

Programación de un Driver

Para ser cargado como un Driver válido por el sistema operativo, este debe implementar los siguientes procedimientos:

Función	Invocación
int driver_init()	Durante la carga del SO
int driver_open()	Al solicitarse un open
int driver_close()	Al solicitarse un close
int driver_read(int *data)	Al solicitarse un read
int driver_write(int *data)	Al solicitarse un write
int driver_remove()	Durante la descarga del SO

Syscalls para Drivers

Para la programación de un driver, se dispone de las siguientes syscalls:

- *void OUT(int IO_address, int data)*
  Escribe data en el registro de E/S
- int IN(int IO_address)
  Retorna el valor almacenado en el registro de E/S

Un Puerto Serie
Envío

Un puerto serie de 1 bit posee 3 registros de E/S mapeados en las siguientes direcciones de memoria: SP_CTRL, SP_STATUS y SP_DATA.

Para enviar un bit por el puerto serie el Driver debe:

1. Escribir el dato a transferir en el bit menos significativo del registro SP_DATA.
2. Escribir la constante SP_SEND en en el registro SP_CTRL para ordenar al dispositivo su envío.
3. Cuando el valor del registro SP_STATUS pasa de valer la constante SP_NOT_READ a valer la constante SP_READY, podemos estar seguros que el dato ha sido transferido por el puerto serie.

Para recibir un bit por el puerto serie el Driver debe:

1. Escribir la constante SP_RECEIVE en en el registro SP_CTRL para ordenar al dispositivo la recepción de un bit.
2. Cuando el valor del registro SP_STATUS pasa de valer la constante SP_NOT_READ a valer la constante SP_READY, podemos estar seguros que un bit ha sido recibido por el puerto serie.
3. Leer el dato recibido en el bit menos significativo del registro SP_DATA.

Escribir un Driver para transferir de a 3 bits usando el puerto serie.
Usar busy-waiting.

Modificar el Driver del ejercicio anterior para aprovechar una nueva versión del puerto serie con soporte para interrupciones. El puerto serie está conectado a la interrupción número 3. El puerto serie interrumpe al procesador por alguno de los siguientes motivos:

- Terminó de transferir un bit.
- Terminó de recibir un nuevo bit.

Syscalls para manejar interrupciones

Para poder manejar interrupciones, necesitamos dos syscalls adicionales:

- int request_irq(int irq, void* handler)
  Permite asociar el procedimiento handler a la interrupción irq.
  Retorna IRQ_ERROR si ya está asociada a otro handler
- int free_irq(int irq)
  Libera la interrupción irq del procedimiento asociado