Asignación III: Conceptos de Dispositivos y Planificación de Disco

Jose David Ruano Burbano

Noviembre de 2025

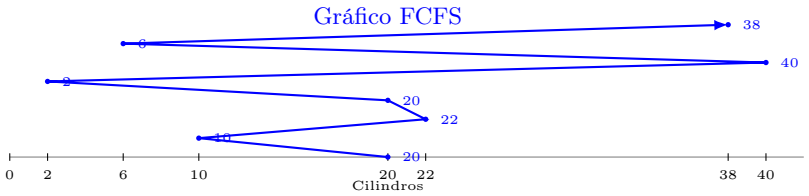
1. Revisión Conceptual (20 %)

A continuación, se presentan las respuestas a las preguntas teóricas de la asignación.

1.1. Pregunta 1

1.1.1. (a) First-Come, First-Served (FCFS)

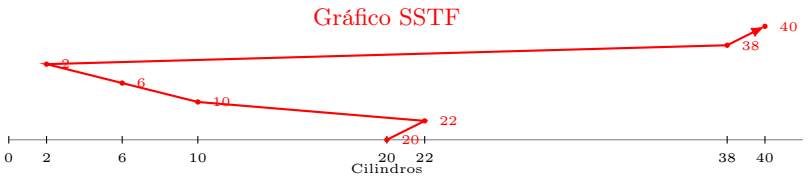
- **Ruta:** $20 \rightarrow 10 \rightarrow 22 \rightarrow 20 \rightarrow 2 \rightarrow 40 \rightarrow 6 \rightarrow 38$
- **Cálculo de cilindros:**
  - Movimiento 1 (20 a 10): $20 - 10 = 10$ cilindros
  - Movimiento 2 (10 a 22): $22 - 10 = 12$ cilindros
  - Movimiento 3 (22 a 20): $22 - 20 = 2$ cilindros
  - Movimiento 4 (20 a 2): $20 - 2 = 18$ cilindros
  - Movimiento 5 (2 a 40): $40 - 2 = 38$ cilindros
  - Movimiento 6 (40 a 6): $40 - 6 = 34$ cilindros
  - Movimiento 7 (6 a 38): $38 - 6 = 32$ cilindros
- **Total Cilindros:** $10 + 12 + 2 + 18 + 38 + 34 + 32 =$ **146** cilindros
- **Tiempo de Búsqueda:** 146 cilindros $\times$ 6 mseg/cilindro = **876** mseg



1.1.2. (b) Closest Cylinder Next (SSTF - Shortest Seek Time First)

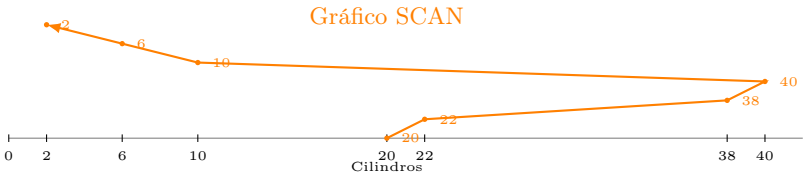
- **Cola de solicitudes:** {10, 22, 20, 2, 40, 6, 38}. **Inicio:** 20.
- El cabezal está en 20 y hay una solicitud en 20, así que la atiende (0 movimiento).
- Desde 20, el más cercano es 22 (distancia 2).
- Desde 22, el más cercano es 10 (distancia 12).
- Desde 10, el más cercano es 6 (distancia 4).
- Desde 6, el más cercano es 2 (distancia 4).
- Desde 2, el más cercano es 38 (distancia 36).
- Desde 38, el único que queda es 40 (distancia 2).
- **Ruta:** $20 \rightarrow 22 \rightarrow 10 \rightarrow 6 \rightarrow 2 \rightarrow 38 \rightarrow 40$

- **Cálculo de cilindros:**
  - Movimiento 1 (20 a 22): $22 - 20 = 2$ cilindros
  - Movimiento 2 (22 a 10): $22 - 10 = 12$ cilindros
  - Movimiento 3 (10 a 6): $10 - 6 = 4$ cilindros
  - Movimiento 4 (6 a 2): $6 - 2 = 4$ cilindros
  - Movimiento 5 (2 a 38): $38 - 2 = 36$ cilindros
  - Movimiento 6 (38 a 40): $40 - 38 = 2$ cilindros
- **Total Cilindros:** $2 + 12 + 4 + 4 + 36 + 2 =$ **60** cilindros
- **Tiempo de Búsqueda:** 60 cilindros $\times$ 6 mseg/cilindro = **360** mseg



1.1.3. (c) Algoritmo de Elevador (SCAN)

- **Cola:** {10, 22, 20, 2, 40, 6, 38}. **Inicio:** 20 (subiendo).
- **Ruta (subiendo):** Atiende 20, luego 22, 38, y 40. Al ser 40 la última solicitud en esa dirección, invierte el sentido.
- **Ruta (bajando):** Atiende 10, 6, y 2.
- **Ruta Completa:** $20 \rightarrow 22 \rightarrow 38 \rightarrow 40 \rightarrow 10 \rightarrow 6 \rightarrow 2$
- **Cálculo de cilindros:**
  - Movimiento 1 (20 a 22): $22 - 20 = 2$ cilindros
  - Movimiento 2 (22 a 38): $38 - 22 = 16$ cilindros
  - Movimiento 3 (38 a 40): $40 - 38 = 2$ cilindros
  - Movimiento 4 (40 a 10): $40 - 10 = 30$ cilindros
  - Movimiento 5 (10 a 6): $10 - 6 = 4$ cilindros
  - Movimiento 6 (6 a 2): $6 - 2 = 4$ cilindros
- **Total Cilindros:** $2 + 16 + 2 + 30 + 4 + 4 =$ **58** cilindros
- **Tiempo de Búsqueda:** 58 cilindros $\times$ 6 mseg/cilindro = **348** mseg



1.2. Pregunta 2

- **Posición Inicial:** 2150
- **Posición Anterior:** 1805 (Esto implica que el cabezal se está moviendo "hacia arriba", del 1805 al 2150).
- **Cola de Solicitudes:** {2069, 1212, 2296, 2800, 544, 1618, 356, 1523, 4965, 3681}
- **Límites del Disco:** 0 a 4999

1.2.1. (a) FCFS (First-Come, First-Served)

- **Ruta:** 2150 → 2069 → 1212 → 2296 → 2800 → 544 → 1618 → 356 → 1523 → 4965 → 3681
- **Cálculo de cilindros:**
  - $2150 - 2069 = 81$
  - $2069 - 1212 = 857$
  - $2296 - 1212 = 1084$
  - $2800 - 2296 = 504$
  - $2800 - 544 = 2256$
  - $1618 - 544 = 1074$
  - $1618 - 356 = 1262$
  - $1523 - 356 = 1167$
  - $4965 - 1523 = 3442$
  - $4965 - 3681 = 1284$
- **Total Cilindros:** $81 + 857 + 1084 + 504 + 2256 + 1074 + 1262 + 1167 + 3442 + 1284 =$ **13011** cilindros

1.2.2. (b) SCAN (Elevador)

El cabezal inicia en 2150 y se mueve hacia arriba (hasta el cilindro 4999).

- **Solicitudes (subiendo):** {2296, 2800, 3681, 4965}
- **Solicitudes (esperando para bajar):** {2069, 1212, 544, 1618, 356, 1523}
- **Ruta (subiendo):** 2150 → 2296 → 2800 → 3681 → 4965 → **4999** (fin del disco)
- **Ruta (bajando):** 4999 → 2069 → 1618 → 1523 → 1212 → 544 → **356** (última solicitud)
- **Cálculo de cilindros:** Se puede calcular como la suma de los dos grandes tramos:
  - Tramo 1 (Subida): $4999 - 2150 = 2849$
  - Tramo 2 (Bajada): $4999 - 356 = 4643$
- **Total Cilindros:** $2849 + 4643 =$ **7492** cilindros

1.2.3. (c) C-SCAN (SCAN Circular)

El cabezal inicia en 2150 y se mueve hacia arriba (hasta 4999). Luego, salta a 0 y sigue subiendo.

- **Ruta (tramo 1, subiendo):** 2150 → 2296 → 2800 → 3681 → 4965 → **4999**
- **Ruta (salto):** El cabezal se mueve de 4999 al cilindro **0**
- **Ruta (tramo 2, subiendo):** 0 → 356 → 544 → 1212 → 1523 → 1618 → **2069** (última solicitud)
- **Cálculo de cilindros:**
  - Tramo 1 (Subida): $4999 - 2150 = 2849$
  - Tramo 2 (Salto al inicio): $4999 - 0 = 4999$
  - Tramo 3 (Subida desde 0): $2069 - 0 = 2069$
- **Total Cilindros:** $2849 + 4999 + 2069 =$ **9917** cilindros

1.3. Pregunta 3

La principal ventaja del algoritmo modificado (C-SCAN) sobre el SCAN normal es la **equidad en el tiempo de espera**.

Para explicarlo podriamos decir que el SCAN normal es como un elevador por ejemplo. Si acabas de perder el elevador (que va "hacia arriba"), tienes que esperar a que llegue hasta el último piso y luego baje hasta tu piso. Las solicitudes que están más lejos, en la dirección opuesta, Tendran un peor tiempo de espera.

En cambio, C-SCAN es como un elevador que **solo va hacia arriba**. Cuando llega al último piso (final del disco), es como si se "teletransportaraïnmediatamente al primer piso (inicio del disco) y vuelve a subir, sin atender solicitudes en la bajada.

Aunque esto puede parecer que hace un movimiento no muy importante en realidad el salto de 4999 a 0, asegura que **ninguna solicitud tenga que esperar demasiado**. El tiempo de espera máximo es mucho más predecible. Nadie tiene que esperar un viaje completo de ida y vuelta; lo máximo que se espera es un ciclo completo en una sola dirección. Esto proporciona un tiempo de respuesta más justo y consistente para todas las solicitudes.

2. Implementación Práctica (60 %)

Para cumplir con la parte práctica de la asignación, se desarrolló un script en Python (‘simulacion.py‘) que modela el comportamiento de un controlador de disco.

2.1. Explicación de la Simulación

El programa implementa los algoritmos FCFS, SCAN y C-SCAN. Utiliza las siguientes librerías clave:

- **sys:** Para leer argumentos desde la línea de comandos (específicamente, la posición inicial del cabezal).
- **random:** Para generar la serie de 1,000 solicitudes de cilindros aleatorias, simulando una carga de trabajo real e impredecible.
- **matplotlib:** Para generar los gráficos estáticos solicitados en la parte de visualización.

El script calcula el movimiento total del cabezal para cada algoritmo y reporta los resultados tanto en la terminal como en archivos de imagen.

2.2. Instrucciones de Uso

Para ejecutar la simulación, se deben seguir los siguientes pasos en un entorno de terminal.

2.2.1. Dependencias

El proyecto usa librerías externas, por lo que se recomienda enfáticamente usar un entorno virtual (‘venv‘) para no instalarlas globalmente.

Crear y activar el entorno (una sola vez):

```
# Crear el entorno en una carpeta llamada .venv
python -m venv .venv

# Activar en Windows (PowerShell/CMD)
.\.venv\Scripts\activate

# Activar en Mac/Linux
source .venv/bin/activate
```

Instalar dependencias: Con el entorno activo, instale las librerías desde el archivo ‘requirements.txt‘:

```
pip install -r requirements.txt
```

2.2.2. Ejecución del Programa

El script se ejecuta con ‘python‘, pasando la posición inicial del cabezal como único argumento. Por ejemplo, para una posición inicial de 2150:

```
python simulation.py 2150
```

2.2.3. Modificación de Dirección (Opcional)

Por defecto, y para cumplir con la tarea, los algoritmos SCAN y C-SCAN se ejecutan con dirección 'up'. Si se desea probar la dirección 'down', se debe modificar manualmente el archivo 'simulacion.py' en la función 'main', antes de ejecutarlo:

```
Ejemplo de modificación para ejecutar C-SCAN en "down"

... en la funcion main() ...
scan_total, scan_path = run_scan(start_position, random_requests)
cscan_total, cscan_path = run_cscan(start_position,
                                    random_requests,
                                    direction="down") # <--- AÑADIR ESTO
```

3. Visualización de Datos (20 %)

La simulación genera dos archivos de imagen ('.png') que cumplen con los requisitos de visualización: una comparación de rendimiento y un gráfico del movimiento del cabezal.

3.1. Ejemplo de Ejecución

A continuación, se muestra una salida de terminal de ejemplo (los números de movimiento total serán diferentes en cada ejecución debido a la naturaleza aleatoria de las solicitudes).

```
(.venv) > python simulation.py 2150

Simulacion de Planificacion de Disco
Disco de 5000 cilindros (0-4999).
1000 solicitudes aleatorias.
Posicion inicial del cabezal: 2150

--- Resultados de Movimiento Total ---
FCFS:   2,451,889 cilindros
SCAN:   7,848 cilindros
C-SCAN: 9,781 cilindros

Generando graficos de visualizacion...
Graficos 'performance_comparison.png' y
'head_movement_comparison.png' guardados.
```

3.2. Gráficos Generados

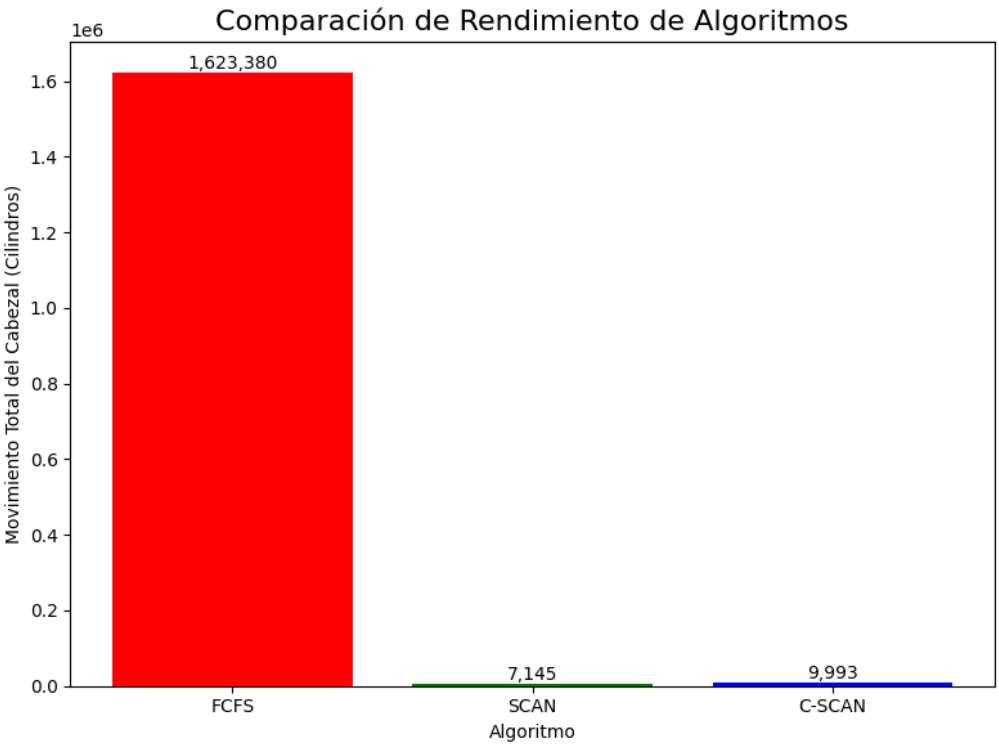


Figura 1: Comparación de Rendimiento: Movimiento total del cabezal para 1,000 solicitudes aleatorias. Se observa la drástica ineficiencia de FCFS.

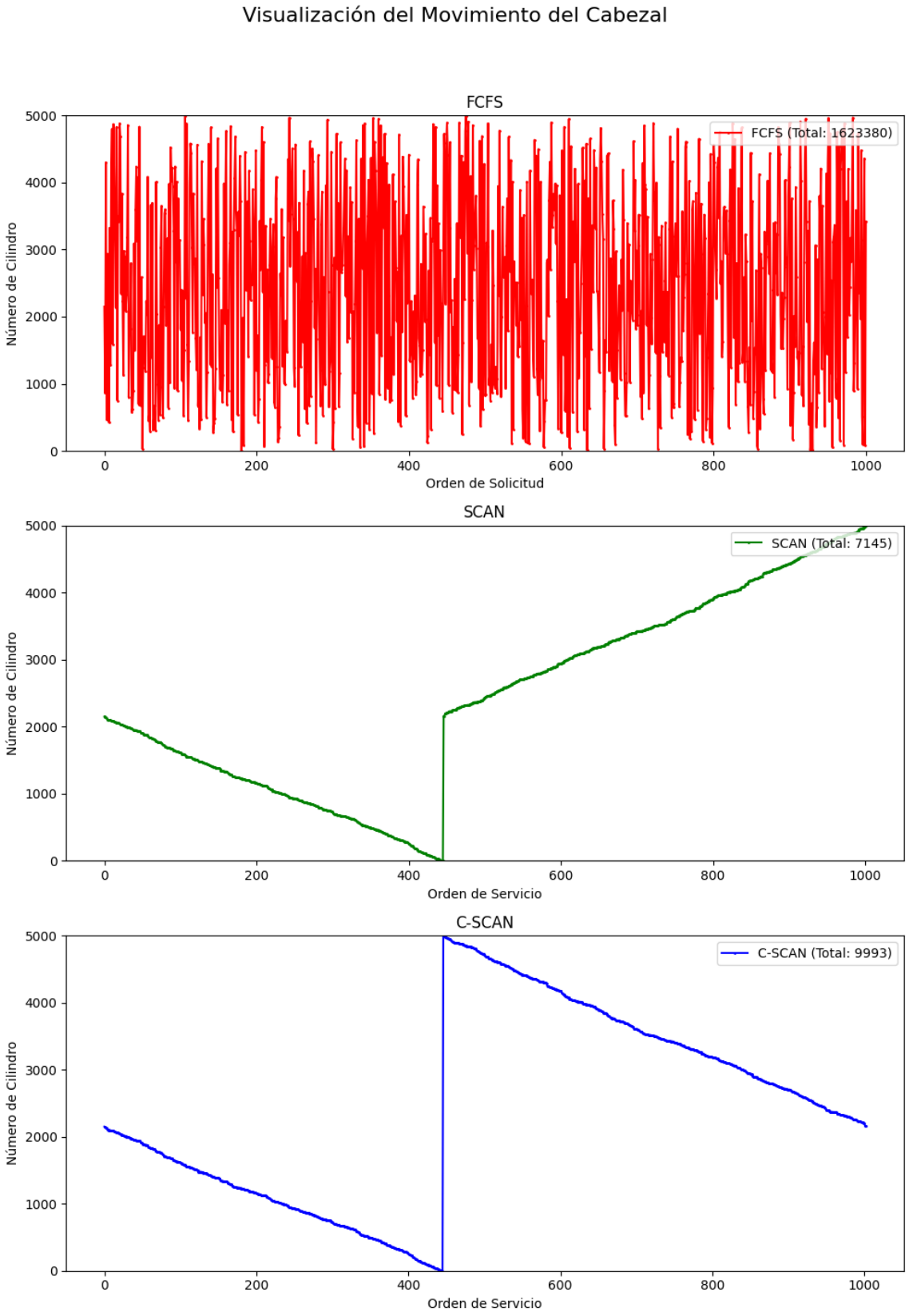


Figura 2: Visualización del Movimiento del Cabezal para cada algoritmo. Nótese el comportamiento errático de FCFS, el barrido de SCAN y el "salto"de C-SCAN.

4. Conclusiones

Del trabajo teórico y la simulación práctica, podemos extraer conclusiones claras. En general, el algoritmo **FCFS (First-Come, First-Served)** es el más simple de implementar y es "justo.[en] el sentido de que atiende todo en orden, sin riesgo de inanición. Sin embargo, como demuestra la simulación en la **Figura 1**, su rendimiento es extremadamente pobre para cargas de trabajo aleatorias. El movimiento errático y excesivo del cabezal (visible en la **Figura 2**) degrada drásticamente la capacidad de respuesta de todo el sistema.

Para mejorar esto, un algoritmo como **SSTF (Shortest Seek Time First)** ofrece un rendimiento mucho mejor al priorizar la solicitud más cercana. No obstante, introduce el grave problema de la inanición, donde las solicitudes lejanas podrían ser ignoradas indefinidamente.

Aquí es donde los algoritmos de .[el]evador"demuestran su superioridad. Tanto **SCAN** como **C-SCAN** no solo ofrecen un rendimiento excelente (como se ve en la barra diminuta de la **Figura 1**), sino que también resuelven el problema de la inanición. La diferencia clave, como se discutió en la Pregunta 3, es la uniformidad del tiempo de espera: SCAN es injusto con las solicitudes en los extremos, mientras que C-SCAN proporciona un tiempo de respuesta más predecible para todas las solicitudes, a costa de un pequeño "saltoïmproductivo.

En resumen, la simulación confirma que para cargas de trabajo pesadas, FCFS es una opción inviable. Los algoritmos de elevador (SCAN y C-SCAN) son superiores porque imponen un orden al caos, reduciendo el movimiento total del cabezal en órdenes de magnitud y mejorando drásticamente la capacidad de respuesta del sistema.

5. Código Fuente (Repositorio)

El código fuente completo de Python, junto con el archivo ‘requirements.txt‘, está disponible en el siguiente repositorio de GitHub:

https://github.com/Jdavidruanob/Assignment-III