CSE 2431 **HOMEWORK 4** **SPRING 2025**

Instructor: Luan Duong, Ph.D.

Electronic Submission: 11:59 pm, Wednesday, April 16, 2025

Point: 25 points (5% of total grade) + Extra 5 points (1% of total grade)

Question 1: Disk Scheduling [15 points]

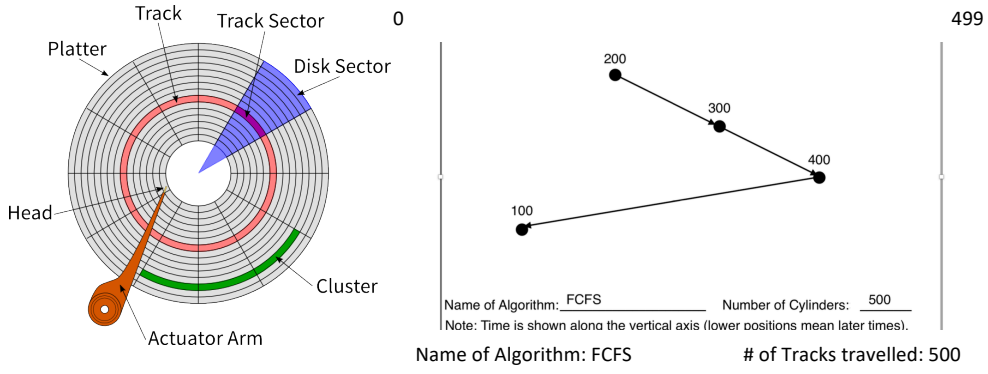
A) For each of the following scheduling algorithms: **SSTF**, **SCAN**, **C-SCAN**: please describe how each of them works in 1 to 2 sentences. The description should be precise enough to distinguish each algorithm from the others. [3 points]

SSTF keywords: head moves to the nearest track/cylinder from the disk head's current position. Simple and reduces disk arm motion, compared to the FCFS algorihtm. [1 pt]

SCAN: Act like an elevator in real life. Follow the direction the disk head is currently moving, reading all the tracks/cylinders in this direction, go to the end, and then reverse direction and does likewise. [1 pt]

C-SCAN: Acting like a "one-way" elevator, meaning that it only read all the tracks/cylinders following its current direction to the end, bounces back the start, and "scan through" again the whole series of tracks. [1 pt]

B) Disk Scheduling [12 points]: Recall that the tracks (or so-called cylinders) of the hard drive is as described in Figure 1(a). The first track (track 0) is the outer-most track, and the inner most track is the highest track that a hard disk has. For each request, the disk head needs positioning itself onto the requested track, following some order (given by the algorithm).



Name of Algorithm: FCFS # of Tracks travelled: 500

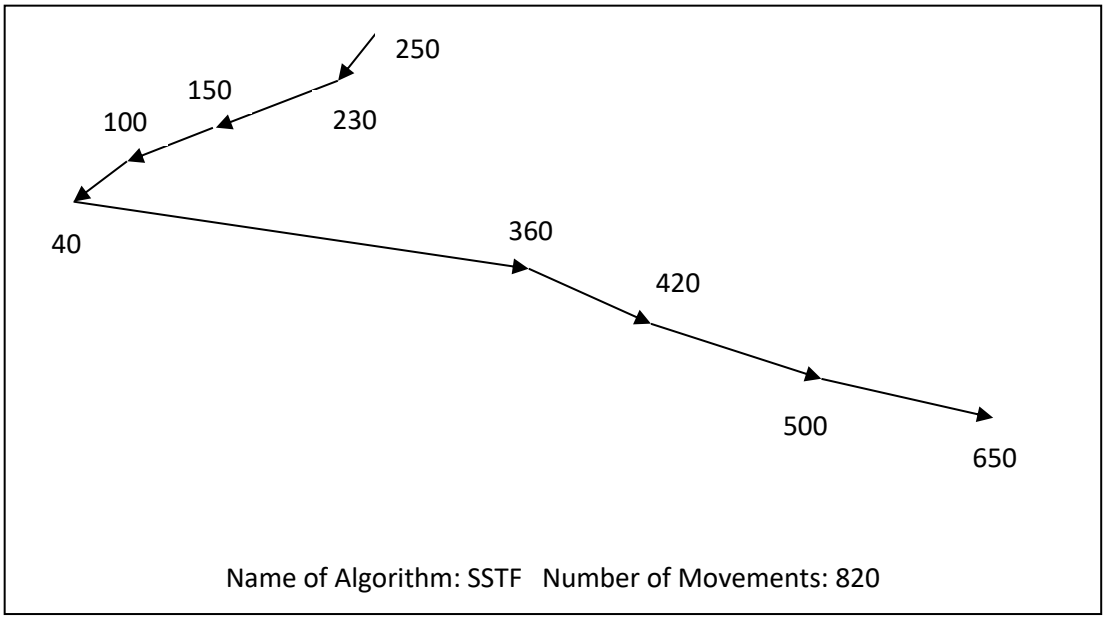
Figure 1(a): Hard Disk illustration Figure 1(b): Example of FIFO Algorithm

Figure 1(b) shows track seeking for a hard drive with 500 tracks, numbered 0 to 499, in which the FCFS algorithm is used. The figure shows service for the following hard disk track requests: 200, 300, 400, 100. Time flows topdown. We assume that no other track requests arrive while this algorithm executes.

In this question 1B, we have a **different hard disk with 700 tracks**, numbered 0 to. 699. The disk head **is at track 250** (**moving toward track 699**), and we have the following queue of hard-disk track requests: **40, 100, 150, 230, 360, 420, 500, 650**.

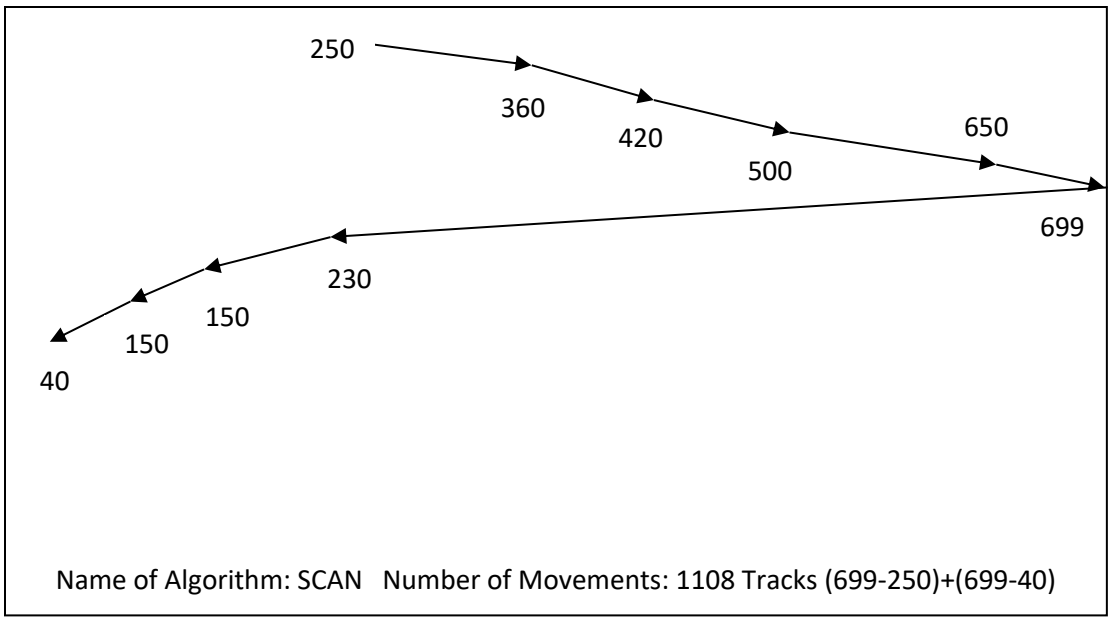
Please draw digrams (as in Figure 1b) showing the movement of the head over time for SSTF and SCAN algorithm. Also, please calculate the total movement of the disk head (in # of tracks). Assume that no other track requests arrive while this algorithm executes.

1. SSTF [5 points] + Total movement of the disk head [1 point]



Name of Algorithm: SSTF Number of Movements: 820

2. SCAN [5 points] + Total movement of the disk head [1 point]



Name of Algorithm: SCAN Number of Movements: 1108 Tracks (699-250)+(699-40)

Question 2: RAIDs **[4 points]**

Match each RAID level with its primary disadvantage. Each disadvantage should be used only once. Choose the best option for each:

C – RAID-0	a. Wastes disk capacity
A – RAID-1	b. Complicated calculation of data and parity location
D – RAID-4	c. Failure of one disk causes loss of data
B – RAID-5	d. Parity disk is performance bottleneck

Question 3: Deadlocks **[6 points]**

Suppose a system has four processes, P = {P1, P2, P3, P4} and three resource types, R =. {R1, R2, R3}. In addition, assume we have two instances of R1, two instances of R2, and two instances of R3. The current state of the system is defined by the following requests and assignments:

- P1 requests an instance of R1;
- P2 requests an instance of R2;

- P3 has been assigned an instance of R1 and P3 requests an instance of R2 and an instance of R3;
- P4 has been assigned an instance of R3 and requests an instance of R1.

Please draw a resource-allocation graph to the above requests and assignments. Is there a deadlock in this situation? You need to explicity say “Yes” or “No”. If you conclude there is no deadlock, your justification must include a sequence of execution for the processes showing that all processes can execute to completion. If you conclude there is deadlock, describe when deadlock could happen? (Note that in this situation, there is only one conclusion: deadlock or not deadlock)

There is NO deadlock – No Cycle in RAG

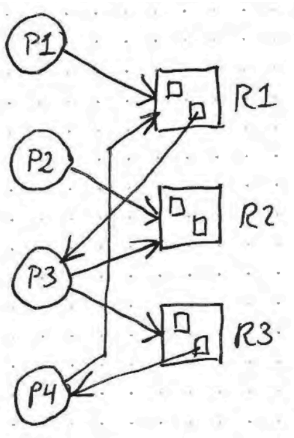
We can allocate an instance of R2 to P2, let P2 finish.

Then allocate one of R2 and one of R3 to P3, since it has already been allocated R1, then P3 can run to completion.

Then assign one of R1 to P1 and let it run to completion.

Then when P3 finishes, it releases R1 and R3, so P4 can be allocated with one R1 and one R3 to run and completes.

Other possible solution will also be accepted.



Question 4: iNode **[Extra 5 points] [Referred to lecture note, Lecture_Topic_8b, slide 62-68]**

Consider a file system (UNIX-based) with 12 direct pointers, 1 indirect pointer, 1 double-indirect pointer, and 1 triple indirect pointer in **the i-node**. Disk blocks' size is 8K bytes, and each pointer pointing to a disk block needs 4 bytes.

a. With this design, what is the largest supported file? You do not need to calculate the final numerical value, but you need to show the full expression. (The multiplication expression) [2 points]

Number of ptrs/block = 8K/4 = 2048

(12 * 8KB) + (2048 * 8KB) + (2048 * 2048 * 8KB) + (2048 * 2048 * 2048 * 8KB)

b. How many disk reads required to **read block 14** of the file named **/a**? Assume that nothing relevant is in the file-cache (e.g., no i-nodes and no data blocks) and that the root directory is only one-block long. Please describe each disk read in details. (i.e. Read [content] from where to get where) [3 points]

Block 14 of the file will be in the indirect block (there are only 12 direct pointers).

A total of 5 reads are required, as follows:

1 - Read i-node 2 and get the first data pointer

2 - Read the data associated with the root directory; find file "a" and its inode number

3 - Read a's i-node and get the address of the indirect block

4 - Read the indirect block to get its 2nd pointer (ptr for block 14)

5 - Read block 14