CSCI 447 - Operating Systems, Spring 2024
Homework 5 : Disk Scheduling
Due Date: Check Canvas

~~Collaboration~~ Policy

Your homework submissions must be yours. You may chat with other students on a high-level about topics and concepts, but you cannot share, disseminate, co-author, or even view, other students' code. If any of this is unclear, please ask for further clarification.

1 Free Response Question

Upload a .pdf document to Canvas for the *Homework 5 - questions* submission.

1.1 Written Question : 5 points

None of the disk-scheduling disciplines, except FCFS, is truly fair (starvation may occur).

a. Explain why this assertion is true
b. Describe a way to modify algorithms such as SCAN to ensure fairness. Please do not merely answer, "modify SCAN so that it is C-SCAN."
c. Explain why fairness is an important goal in a time-sharing system
d. Give three or more examples of circumstances in which it is ipmortant that the operating system be unfair in serving I/O requests.

2 Coding Task : 2 points for setup, 29 points for the program

Your task is to implement a disk scheduling simulator, a program named *DiskSim* which runs the FCFS, SSTF, SCAN, LOOK, C-SCAN and C-LOOK disk scheduling algorithms. The simulation will either generate a random head starting position and direction, and 1,000 random requests among 10,000 cylinders to seek, else will receive as input user specified parameters. The program will output the order of the seek requests served, the sum of the head movements, and the count of head reversals, for all 6 algorithms.

2.1 Setup

1. Create and checkout a new branch on your gitlab. Name the branch *homework5*. **When creating a new branch, ALWAYS do that from the master branch.**
2. In the *homework5* branch, create a new directory, *homework5*. All work for this homework assignment should be done in the *homework5* branch and *homework5* directory.

2.2 The Disk, and program parameters

Your disk has 10,000 cylinders. Write a simulator program, *DiskSim*, in any language that can be compiled from the command line of a CS computer – for example C, C++, java, or python. The program must have two modes, *R*andom and a parameter mode. If the single command line argument is `R`, the program should:

- Select a random start head location
- Select a random direction for the head (towards higher or lower cylinder numbers) relative to the start location. At time 0, the head is MOVING in the direction specified.
- Generate 1,000 random cylinder requests

If `R` is not the **single** command line argument, the program should receive and process the following command line arguments (in this order):

- Starting location of disk head. This should be an integer.
- Direction of MOVING disk head at the start of the simulation, either `H` for Higher cylinder values relative to the starting location, or `L` lower cylinder values relative to the starting location.
- Cylinder requests (integer values), space separated, such as `34 432 2000`, else the argument of the form `R#`, where the `#` is an integer count specifying the count of random cylinder requests that should be generated. For example, `R5` and `R345` would generate 5, and 345 randomly generated cylinder requests.

For C-SCAN and C-LOOK, the head serves requests when moving from 0 to 9,999.

Three sample invocations are the following (these are for code that was written in C, but if you write your code in java or python, the invocation will be slightly different, and prepended with `python` or `java` instead of `./`:

```
./DiskSim R
```

```
./DiskSim 700 H R78
```

```
./DiskSim 322 L 67 544 32
```

In the first, the head's location and direction, as well as 1,000 seek requests, will be randomly generated. In the second, the head's starting location is 700, it is moving towards higher cylinder values, and 78 randomly generated seek requests will be made. In the third, last, invocation, the head begins at location 322, it is moving towards lower cylinder numbers, and the seek requests 67, 544, and 32 are processed.

2.3 Program output

Your program should output, to the screen, the following (in this order):

1. For each of FCFS, SSTF, SCAN, C-SCAN, LOOK, and C-LOOK:
   (a) The order of the cylinder seek numbers that were serviced

2. For each of FCFS, SSTF, SCAN, C-SCAN, LOOK, and C-LOOK:
   (a) The count of head direction changes
   (b) The number of cylinders over which the head moved

2.4 Duplicate requests, multiple service histories, request times, etc.

- Assume that all cylinder requests have been received by the OS at $t = 0$. For FCFS, they should be processed in the order left-to-right, but for all others, the OS should be assumed to be aware of ALL requests.
- If either of the random command line options are specified, your simulation should simulate the 6 disk scheduling algorithms on the SAME randomly generated request list.
- If there is more than one possible answer (sequence of cylinder requests), your program should predict only ONE of them (see below sample invocation).
- If there are duplicate cylinder requests (for example, cylinder 45 is requested twice), then FCFS should service all of them individually, EXCEPT when the duplicate requests are adjoining. For example, if requests are 10, 5, 10, then FCFS would service 10, 5, then 10, but if the requests are 10, 10, 5, then FCFS would service 10 then 5. For all algorithms other than FCFS, duplicate requests should be serviced only once.
- You should "count" the starting position of the head in the tally of cylinder locations. For example, if the cylinder head starts at 15, and proceed to 10, and then proceed to 12, then the head would move over a total of 8 cyldiners, because the sequence of cyldiner access would be `15,14,13,12,11,10,11,12`.

2.5 Sample invocation and output

To facilitate confirming counts "by hand," the below sample invocation is for a disk with only 6 cylinders (0,1,2,3,4,5). Make sure your implementation is for a 10,000 cylinder disk.

```
./DiskSim 3 H 4 1 5 2
== Service history ==
FCFS 4 1 5 2
SSTF 4 5 2 1
SCAN 4 5 2 1
C-SCAN 4 5 1 2
LOOK 4 5 2 1
C-LOOK 4 5 1 2
== Service stats ==
FCFS 3 12
SSTF 1 7
SCAN 1 7
C-SCAN 2 10
LOOK 1 7
C-LOOK 2 8
```

The sequences of cylinder movements for the above example are the following:

- FCFS: `3 4 3 2 1 2 3 4 5 4 3 2`
- SSTF: `3 4 5 4 3 2 1` (1 reversal) or `3 2 1 2 3 4 5` (2 reversals)

- SCAN: `3 4 5 4 3 2 1`
- C-SCAN: `3 4 5 4 3 2 1 0 1 2`
- LOOK: `3 4 5 4 3 2 1`
- C-LOOK: `3 4 5 4 3 2 1 2`

2.6 Files to submit

Please push to the homework5 branch, into the homework5 directory, the following :

- The *DiskSim.h* and *DiskSim.c*, or *DiskSim.java*, or *DiskSim.py* file(s)
- Any other custom .c and .h file(s), and/or .java and/or .py files, you needed to complete this task
- A *Makefile*, which if invoked will generate a *DiskSim* executable or *DiskSim.class* (if using C, C++, or java)
- A short report, 1 "page" maximum, a plain Text file, called *diskSchedulingAnalysis.txt*, that includes your findings on the disk scheduling algorithm that you would recommend for use by a disk controller. **Substantiate** your recommendation with your run-time results.

3 Rubric

Book question	5
Programming Task : setup (git), including branch, and .o nor executable file(s) have NOT been pushed to your git branch	2
Programming Task : program correctly identifies the order of requests served, distance traveled, and head direction changes for each scheduling algorithm	18
Programming Task : Makefile	1
Programming Task : program processes command line arguments, and correctly simulates the scheduler, including random options	4
Programming Task : quality of your code, including choice of data structures, efficiency, readability, commenting, etc.	1
Programming Task : *diskSchedulingAnalysis.txt* file, which includes (brief) discussion, explaining your selection of the optimal disk scheduling algorithm	5
Total	36 points