Simon Fraser University
School of Computing Science
CMPT 300: Assignment #4

Disk Scheduling Algorithms

Reminder: The rules of academic conduct apply as described in the course outline. All coding is to be done **alone or in pairs**. We will be using electronic software to compare all assignments handed in by the class to each other, as well as to assignments handed in for previous terms that this, or similar, assignment may have been given in.

Be sure that this assignment compiles on the Linux computers in the CSIL lab in using the gcc compiler. You can access the Linux server remotely as detailed on the course discussion forum.

What to Hand In:

1. Every C source file you completed for this assignment.
2. A Makefile which will compile your program creating an executable named **DSSimul**.
3. Sample output showing that your program works.

Simulating Disk Scheduling Algorithms

In this assignment you will write a program that compares the performance of the disk scheduling algorithms SSTF, Scan, and C-Scan to the FCFS algorithm.

The disk drive will be modelled as consisting of 200 concentric tracks, numbered 0 through 199. A disk scheduling algorithm will accept a request list as a sequence of m unique integers in the range [0, 199], where each integer t is a request to seek to track t. The output of each scheduling algorithm is an ordering in which the m requests are serviced.

Your program will do the following:

- If the user provides a command-line argument, the argument will be a comma-delimited list of unique track numbers (with at least three track numbers, no spaces), each representing a request to seek to the corresponding track.
  - For example, your program could be called as
    `./DSSimul 49,20,34,55,23,123,154,12,0,101,188`
- If the user *does not* provide a command-line argument, your program will generate a request list of at least 50 unique random integers in the range [0, 199], each representing a request to seek to the corresponding track.

- Implement *two* of the following algorithms: SSTF, Scan, or C-Scan
  - Use the first integer in the list as the starting track number for the r/w head at the beginning of each algorithm.
    - For example, if the first integer in the list was 64, then you will start the r/w head at track 64, with the head moving in the *left* (i.e., decreasing track number) direction in the case of the Scan/C-Scan algorithm.
  - Each implementation will take the sequence of requests and return the ordered sequence that the requests will be serviced in, as well as the total number of tracks traversed by the r/w arm for that algorithm.
    - For the Scan or C-Scan algorithms, you should *not* use the "Look" modification to the algorithm, as described in class.
    - For the C-Scan algorithm, you should count the "rewind" of the disk head when counting the total number of tracks traversed by the r/w arm.
- Compare the fairness of the algorithms in terms of the delays experienced by any request as a result of being processed out of FCFS order.
  - For example, FCFS processes a given sequence of tracks (1, 5, 3, 2, 6, 4) in the given order and thus no track experiences any delay or speedup. SSTF processes the same sequence in the order (1, 2, 3, 4, 5, 6). Tracks 2 and 4 are processed earlier than under FCFS. The preferential treatment is at the expense of track 5, which is delayed by 3 steps, and track 6, which is delayed by 1 step.
- Determine and output:
  - The original request list of track numbers your program started with, either from the command-line, or the randomly generated one.
  - The ordered sequence that requests will be serviced in by each algorithm you have implemented.
  - The total number of tracks traversed by the r/w arm for each algorithm.
  - The longest delay experienced by any track in comparison to FCFS for each algorithm.
  - For each algorithm, the average delay of all tracks that have been processed later than under FCFS. In the above example, track 5 was delayed by 3 steps and track 6 by 1 step. The average delay is $(3 + 1)/2 = 2$.