Programming Assignment #4 (Lab 4): IO Scheduling **Professor Hubertus Franke**
Class CSCI-GA.2250-001 Spring 2017

You are to implement different IO-schedulers in C or C++ and submit the **source** code, which we will compile and run. Please provide a Makefile so we can run on energon (and please at least test there as well).

In this lab you will implement/simulate the scheduling of IO operations. Applications submit their IO requests to the IO subsystem, where they are maintained in an IO-queue. The IO-scheduler then selects a request from the IO-queue and submits it to the disk. On completion another request can be taken from the IO-queue and submitted to the disk. The scheduling policies will allow for some optimization as to reduce disk head movement or overall wait time in the system. The schedulers to be implemented are FIFO (i), SSTF (j), SCAN (s), CSCAN (c), and FSCAN (f) (the letters in bracket define which parameter must be given in the –s program flag).
Note: ***when we say [C|F]SCAN we actually mean the LOOK variant algorithm*** version which does not scan the entire width of the of the disk. This **is** the common usage in the industry of the term SCAN. Also remember that when switching queues in FSCAN you always scan up first from the current position, then down until queue empty, this is just a convention for this lab.

Invocation is a follows:

./iosched –s<schedalgo> <inputfile>

The input file is structured as follows: Lines starting with '#' are comment lines and should be ignored.
Any other line describes an IO operation where the 1st integer is the time step at which the IO operation is issued and the 2nd integer is the track that is accesses. Since IO operation latencies are largely dictated by seek delay (i.e. moving the head to the correct track), we ignore rotational and transfer delays for simplicity. Move by one track takes one time unit. The inputs are well formed.

```
#io generator
#numio=32 maxtracks=512 lambda=10.000000
1 430
129 400
:
```

We assume that moving the head by one track will cost one time unit. As a result your simulation can/should be done using integers. The disk can only consume/process one IO request at a time. Everything else must be maintained in an IO queue and managed according to the scheduling policy. The initial direction of the SCAN algorithms is from 0-tracks to higher tracks. The head is initially positioned at track=0 at time=0. Note maxtrack is not an issue here (think why).

Each simulation should compute and printout the following computed results in the subsequent format.

Total_time:	total simulated time, i.e. until the last I/O request has completed.
Tot_movement:	total number of tracks the head had to be moved
Avg_turnaround:	average turnaround time per operation from time of submission to time of completion
Avg_waittime:	average wait time per operation, i.e. time from submission to issue of the IO request to start of disk operation
Max_waittime:	maximum wait time for any io operation.

```
printf("SUM: %d %d %.2lf %.2lf %d\n",
        total_time,
        tot_movement,
        avg_turnaround,
        avg_waittime,
        max_waittime);
```

Various sample input and outputs are provided on the website.
Please look at the sum results and identify what different characteristics the schedulers exhibit.

You can make the following assumptions:
- at most 1000 io operations will be tested, so its OK to first read all requests from the file before processing.

You don't have to use discrete event simulation. You can write a loop that increments simulation time by one and check whether any action is to be taken, i.e. whether an I/O completed or whether and I/O arrived and whether the track has to be moved by one (i.e. there is an active I/O ongoing).

Additional Information:

As usual, I provide some more detailed tracing information to help you overcome problems. Note your code only needs to provide the 'SUM line' and that's all we are going to test.

There are two parts to this information:
a) detailed execution trace
b) result line for each I/O

The execution trace contains 3 different operations (add a request to the IO-queue, issue an operation to the disk and finish a disk operation). Following is an example of tracking IO-op 21 through the times 2827..2892 from submission to completion.

```
2827:   21 add 204          //  21 is the IO-op # (starting with 0) and 204 is the track# requested
2827:   21 issue 204 269    //  21 is the IO-op #, 204 is the track# requested, 269 is the current track#
2892:   21 finish 65        //  21 is the IO-op #, 65 is total length of the io from request to completion
```

Finally, I provide a summary line for each io-request which consolidates the trace into a single line per IO, providing the following information:

- IO-op# ,
- its arrival to the system (same as inputfile)
- its disk start time
- its disk end time