Objectives:

- Simulate some disk scheduling algorithms
- Gain hands on experience with the physical structure of HDDs; sectors, tracks, cylinders etc.
- Exercise with Seek Time, Rotational Latency, and Transfer Time Calculations
- Practice with data layout; Logical Block Numbers (LBNs) and Physical Sector Numbers (PSNs)

1 Project Description

In this project, you will write a program called **disksched**, which simulates FCFS and SSTF disk scheduling algorithms. The program will be invoked as follows:

```
./disksched <inputfile> <outputfile> <algorithm> [limit]
```

- **`<inputfile>`** is the name of a text file including the information about the disk requests to be scheduled. Each line of the input file will include information about a different disk request and it will have the following format:

  ```
  <arrival-time> <LBN> <request-size>
  ```

  For example, the following can be a sample input file:

  ```
  0.011413 657728 16
  0.011565 31244784 64
  0.011721 21868784 16
  0.016187 34567136 64
  0.018256 11813968 32
  0.020103 32234560 16
  ```

  Each line of the input file represents a different request, starts without any space, ends with a newline, and there is a single space (' ') in between each field in a line. Note that the example input file provided above is just provided for illustration purposes. Your program will be tested using input files containing much larger number of requests. Here is more information about the content of the input file:

  – All three fields are guaranteed to appear in the input file
  – `<arrival-time>` is a positive real number in **seconds**
  – `<LBN>` stands for the Logical Block Number and it is a positive integer representing the starting location of the disk request
  – `<request-size>` is a positive integer describing the number of **sectors** to be transferred
  – Requests are sorted by their arrival times in increasing order
  – Requests have different arrival times

- **<outputfile>** is the name of the text file including the result of the simulation. You will receive the name as a command line argument, create the file and fill its content. Each line of the output file will contain the simulation result of a different request previously provided in the input file and the results will have the following format:

```
<arrival-time> <finish-time> <waiting-time> <PSN> <cylinder> <surface> <sector-offset>
```

  The results in the output file should appear in a sorted (increasing) order with respect to their `<finish-time>`. Each request output should start without any space and should end with a newline, where there is a single space (' ') in between each field in a line. Here is more information about the content of the output file:

  - `<finish-time>` is a positive real number in **seconds** representing the ending time of the request with respect to its `<arrival-time>`
  - `<waiting-time>` is a non-negative real number in **seconds** representing the time that the request waited in the disk queue
  - `<PSN>` stands for the beginning of the Physical Sector Number that the disk head is pointing right after the completion of the request
  - `<cylinder>` stands for the cylinder number that the request is served from. `<cylinder>` is a number between 0 and **CYLINDERS**-1. **We will make sure that requests will not span on multiple cylinders**
  - `<surface>` stands for the surface number that the request is served from. Surface is a number between 0 and **TRACKSPERCYLINDER**-1
  - `<sector-offset>` stands for the beginning of the sector offset that the disk head is pointing right after the completion of the request. Sector offset is a number between 0 and **SECTORSPERTRACK**-1

- **<algorithm>** can be one of the followings:

  - FCFS
  - SSTF

  FCFS will simulate the First Come First Served (FCFS) disk scheduling algorithm and SSTF will simulate the Shortest Seek Time First (SSTF) disk scheduling algorithm. **While simulating SSTF, if there are multiple requests in the disk queue with the same seek distance to the disk head, then choose the one with the earliest arrival time.**

- **[limit]** is an **optional** command-line argument (square brackets around it point out that it is an **optional** argument). If this argument is not provided, then you will simulate the whole input file until its end; otherwise, you will only simulate the first `[limit]` requests. For instance, if limit is given as 10, then you will only simulate the first 10 requests and you program will terminate without simulating the rest of the input file. `[limit]` is a positive integer less that or equal to the number of lines in the input file.

  For example, if your program is invoked as:

```
./diskched in.txt out.txt FCFS
```

then it will read the input file "`in.txt`", compute the finish time and waiting time of each disk request provided in this input file by simulating the FCFS algorithm, and write the corresponding <arrival-time> <finish-time> <waiting-time> <PSN> <cylinder> <surface> <sector-offset> values to the output file "`out.txt`". Note that input and output file names do not have to be `in.txt` and `out.txt`, different names can be passed to the program. Assume that the content of `in.txt` is as the sample input file provided above. Then for this invocation, the content of the output file should be as follows:

```
0.011413 0.018133 0.000000 5261840 3288 5 40.000000
0.011565 0.025093 0.006568 249958336 156223 7 136.000000
0.011721 0.029173 0.013372 174950288 109343 7 88.000000
0.016187 0.035173 0.012986 276537152 172835 5 152.000000
0.018256 0.040933 0.016917 94511776 59069 6 176.000000
0.020103 0.050053 0.020830 257876496 161172 6 96.000000
```

For the same input file content, if your program is invoked as:

```
./diskched in.txt out.txt FCFS 3
```

then the content of the output file should be as follows:

```
0.011413 0.018133 0.000000 5261840 3288 5 40.000000
0.011565 0.025093 0.006568 249958336 156223 7 136.000000
0.011721 0.029173 0.013372 174950288 109343 7 88.000000
```

For the same input file content, if your program is invoked as:

```
./diskched in.txt out.txt SSTF
```

then the content of the output file should be as follows:

```
0.011413 0.018133 0.000000 5261840 3288 5 40.000000
0.011721 0.025093 0.006412 174950288 109343 7 88.000000
0.011565 0.030613 0.013528 249958336 156223 7 136.000000
0.020103 0.034933 0.010510 257876496 161172 6 96.000000
0.016187 0.040693 0.018746 276537152 172835 5 152.000000
0.018256 0.046453 0.022437 94511776 59069 6 176.000000
```

For the same input file content, if your program is invoked as:

```
./diskched in.txt out.txt SSTF 3
```

then the content of the output file should be as follows:

```
0.011413 0.018133 0.000000 5261840 3288 5 40.000000
0.011721 0.025093 0.006412 174950288 109343 7 88.000000
0.011565 0.030613 0.013528 249958336 156223 7 136.000000
```

2 Project Details

2.1 Constants

You can assume the following constants:

- SECTORSPERTRACK 200
- TRACKSPERCYLINDER 8
- CYLINDERS 500000
- RPM 10000
- PHYSICALSECTORSIZE 512 (Bytes)
- LOGICALBLOCKSIZE 4096 (Bytes)
- TRACKTOTRACKSEEK 2 (milliseconds)
- FULLSEEK 16 (milliseconds)
- TRANSFERRATE 1 (Gb/s)

2.2 Disk Layout

Based on the constant values above, the structure of the disk is as in Figure 1.

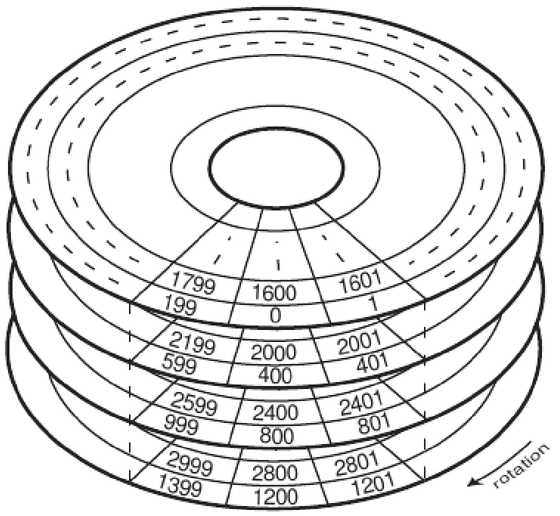

Figure 1: HDD Structure

There are 4 platters in the disk and both surfaces of each platter can be used to store sectors. Therefore, there are total of 8 surfaces and 8 head, one for each surface. This means that there are also 8 tracks per cylinder. There are 500000 tracks on each surface, which also means that there are 500000 cylinders in the disk. Each track can store 200 sectors, and sectors are stored to the disks as in the figure such that sectors 0 to 199 are stored consecutively on surface 0, sectors 200-399 are stored on surface 1, sectors 400-599 are stored on surface 2 and so on. Based on these values, you can calculate the capacity of this disk as follows:

CAPACITY = SECTORSPERTRACK * TRACKSPERCYLINDER * CYLINDERS * 512B $\approx$ 134GB

Since the logical block size (4KB) is larger than the physical sector size (512B), one logical block will span multiple ($\frac{4KB}{512B} = 8$ in this case) consecutive sectors. Given an LBN, you can easily calculate the PSN first (just multiply by 8) and then using this PSN, you can calculate the corresponding cylinder, surface, and sector offset values.

2.3 <finish-time> Calculation

In order to calculate the `<finish-time>` of the requests, you need know how much a request waits in the disk queue and what is the service time of the disk for that request. `<waiting-time>` of the requests can be calculated easily depending on the time when they will be started to be serviced based on the disk scheduling algorithm and their `<arrival-time>`. Service time is the time that the disk spends to service the request and it is calculated as follows:

Service Time = Seek Time + Rotational Latency + Transfer Time

2.3.1 Seek Time Calculation

Before calculating the seek time, you need to know the distance of seek to be performed based on the current head position (which cylinder it is located on) and the target cylinder where the request to be served is located. After finding this distance, you can calculate the seek time as follows:

$$SeekTime(D) = \begin{cases} 0 & D = 0 \\ \frac{FULLSEEK - TRACKTOTRACKSEEK}{CYLINDERS} * D + TRACKTOTRACKSEEK & D > 0 \end{cases} \quad (1)$$

Based on Equation 1, if the seek distance D is 0, then the seek time will be 0; otherwise, it will be calculated using the linear function:

$$\frac{FULLSEEK - TRACKTOTRACKSEEK}{CYLINDERS} * D + TRACKTOTRACKSEEK = 0.000028 * D + 2$$

Note that a linear seek curve is not realistic but it simplifies the problem.

2.3.2 Rotational Latency Calculation

Before calculating the rotational latency, you need to know the current head position (which sector offset it is located on) and the target sector offset where the request will be started to be serviced. Pay attention to the rotation direction of the disk given in Figure 1. After finding the number of sectors to be travelled, you can calculate the rotational latency using the SECTORSPERTRACK and RPM values. **One important issue here is that during the seek, the disk continues to rotate and you need to take this rotation also into account.** Note that after the seek operation, the head may end up being in the middle of a sector, therefore, the current head position (which sector offset it is located on) can be a real number.

2.3.3 Transfer Time Calculation

Transfer time of a request can be calculated using TRANSFERRATE and `<request-size>` info.

Pay attention to the unit of the service time calculated at the end since the finish time and waiting time values to be written to the output file should be in seconds.

2.4 Assumptions

You can make the following assumptions:

- Before the simulation starts, the head points to the beginning of the physical sector 0
- Controller overhead and head switch times are zero
- Requests do not span on multiple cylinders
- Disk does not rotate if it is not servicing a request. It can immediately stop after finishing the last request in the queue until a new request is started to be serviced

2.5 Tips

For consistency of the fractional parts of the values in the output file, use only the data types **`double`** (for timing and sector offset variables) and **`int`** (for all other variables) in your program.

3 Development

You will develop your programs in a Unix environment using the C programming language. *gcc* will be used as the compiler. You will be provided a Makefile and your program should compile without any errors/warnings using this Makefile. Black-box testing will be applied. The output file created by your program will be compared to the correct output file.

What to Submit

You will only submit the source code of your `disksched.c` program.