README

What does this program do?

In this program, we have tried to compare different disk scheduling algorithms(Random, First-in first-out(FIFO), Shortest service time first(SSFT), SCAN, C-SCAN) in which I have taken 1000 processes in my process queue, and for each process in the process queue, I have found the response time with respect to a particular algorithm, after obtaining the data of response time I have noted minimum response time, maximum response time, average response time, the standard deviation of response time and throughput of the algorithm for the generated random input.

Working of the program:

Firstly I generated 1000 requests in my process queue with the help of the Random function then I'm calling my function get_process_number() which will return the index of the process in the process queue with respect to the disk scheduling algorithm and the current position of the read and write head then I founded the response time for the current process with the help of the calc_time() which considers current process details(platter number, track number, sector number) read-write head position details(track number, sector number) and disk scheduling algorithm to calculate the response time.

How to run the program:

1. Open the terminal. Navigate to the directory where the file is located. The filename Is a file.c
2. Enter: gcc file.c -o File -lm (It will generate a code executable file.)
3. Enter: ./File r N Ts

where
r: rotational speed of the disk (revolution per minute).
N: sector size in bytes. Ts: average seek time in ms

Command: In server side terminal: gcc file_name.c -o file_name -lm ./file_name 7500 512 4

Working:

First, I used the Random function to generate 1000 requests in my process queue, then I called my function to get process number(), which returns the index of the process in the process queue based on the disc scheduling algorithm and the current position of the read and write heads. Finally, I used the calc time() function to calculate the response time for the current process, which takes into account current process details (platter number, track number, sector number, etc.). The results of the data collection during simulations are mentioned below:

Parameters r, N, Ts	Scheduling Algorithm	Throughput Requests/s	Min response time in ms	Max response time in ms	Average of response time in ms	Standard deviation of response time in ms
7500, 512, 4ms	Random	63	8	19.6	15.72	2.40
15000, 512, 4ms	Random	102	4	11.8	9.71	1.36
7500, 512, 4ms	FIFO	63	8	19.6	15.73	2.39
15000, 512, 4ms	FIFO	102	4	11.8	9.78	1.40
7500, 512, 4ms	SSTF	116	8	12.4	8.58	0.65
15000, 512, 4ms	SSTF	229	4	8.2	4.35	0.60
7500, 512, 4ms	SCAN	116	8	12.4	8.61	0.63
15000, 512, 4ms	SCAN	229	4	8.2	4.35	0.59
7500, 512, 4ms	C-SCAN	116	8	12	8.59	0.64
15000, 512, 4ms	C-SCAN	229	4	8.4	4.35	0.60

(NOTE: for obtaining the given data I have taken the starting position of my read-write head at 12th track and 10th sector)

Conclusion:

1. If we compare Random, FIFO, SSFT, SCAN, and C-SCAN disk scheduling algorithms, we find that Random and FIFO are very much slower in comparison to SST, SCAN, and C-SCAN. Due to the obvious fact that random is processing the request in random order and thus not being able to exploit the fact that there can be many other processes that can be near the current read-write head position so they should be processed first Same with FIFO as it always processes the next request irrespective of the fact that how far is the next request is.
2. The fact that SSFT, SCAN, and C-SCAN give good timings because they always process the request in either a linear fashion in space or the process those requests

which are near to the current read-write head which eventually makes them a faster disk scheduling algorithm in comparison to FIFO and Random.