**Prashant Yadav**

**CS 4348.002 – Program 5 Report**

**Problem statement –**

The assigned task was to simulate disk scheduling process for following scheduling policies.

* First in first out eviction policy(FIFO)
* Shortest Service time first(SSTF)
* SCAN
* CSCAN

Number of I/O requests will vary from 500 to 1000 and for each number of I/O requests there must be 1000 simulation and seek time will be average over all these simulations. It is a Monte Carlo simulation. Here is configuration of the disk drive.

**HDD details**: 64KB sectors, 5,000 tracks, 12,000 sectors per track, 6Gb/s transfer rate, 10,000 RPM, Average seek time of 2 Ms, a snapback feature which will move the head back to track zero in 1.5 Ms.

**Approach to solution –**

I went through slides and book to understand each type of scheduling policy. In order to perform read or write operation at a location in disk drive head must move from one track sector combination to another track sector combination.

*float ttime = abs(tr-totr)\*tseek; //Track seek time*

*int isec = sec - ((int)(ttime/sseek)%12000); //Sectors moved with in that time*

*if(isec<0){*

*isec+=12000;*

*}*

*return ttime + abs(tosec - isec)\*sseek; //Total time*

There are following policies to select next I/O request that must be served.

1. **FIFO–** In case of FIFO scheduling policy processor servers I/O requests on first come first server basis. It performs bad in real time scenario.
2. **Shortest Service time first(SSTF) –** In this scheduling policy processor evaluates which is the next location where can be reached in least time. It prioritizes I/O request based on the shortest service time first. It is better than FIFO but almost like SCAN.
3. **SCAN –** In this scheduling policy head moves in increasing order of track and after reaching at the end comes back in decreasing order of track. Practically this performs better than other three scheduling policies.
4. **CSCAN -** In this scheduling policy head moves in increasing order of track and than it comes back to beginning and goes again in increasing order. It performs better than FIFO but bas compare to SSTF and SCAN.

**Solution description –**

After understanding each type of scheduling policies, I implemented them in C. I used rand() method in C to generate random values and implemented a method to check whether generated value is unique or not. In this simulation number of I/O request has been varied from 500-1000 and for each number of I.O requests there are 1000 simulations.

*for(i=0;i<1000;i++){*

*generate\_io\_data(); //Generate random I/O Requests*

*//print\_data();*

*seek\_time[length-500][0]+=(float)(fifo\_disk\_scheduling()); //Simulate for FIFO*

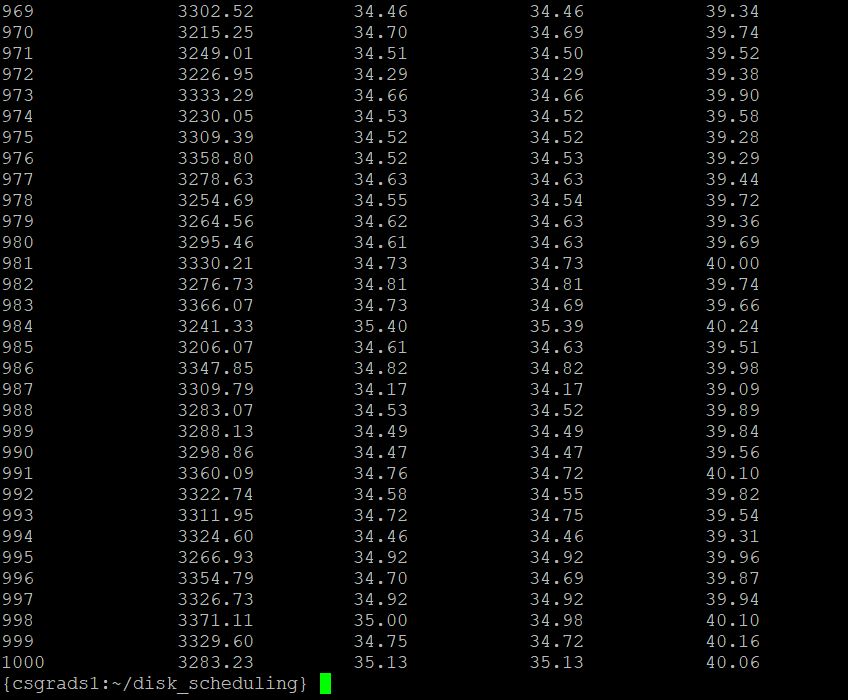
*seek\_time[length-500][1]+=(float)(sstf\_disk\_scheduling()); //Simulate for SSTF*

*seek\_time[length-500][2]+=(float)(scan\_disk\_scheduling()); //Simulate for SCAN*

*seek\_time[length-500][3]+=(float)(csan\_disk\_scheduling()); //Simulate for CSCAN*

}

Above snippet of code shows simulation from 1 to 1000 for each number of I.O request. Seek time for each simulation is maintained in seek\_time matrix and later we normalize it and covert it to second. Here is snapshot for code execution.



–

In above code we can see that FIFO performs worst whereas SSTF and SCAN are kind of similar where SCAN takes edge over SSTF in most of the simulations. CSCAN performs better than FIFO but not good enough as SSTF and SCAN.

For given HDD configuration it is SCAN scheduling policy which out performs rest of the scheduling policies.

Here is the plot of seek time over number of I/O for each type of scheduling policy(see next page).

Figure 1