Jesse Mayer

COEN-177

Assignment #5

The goal of my experiment would be to investigate the performance of the disk scheduling algorithms for specific disk requests. The algorithms I would focus on would be First-Come-First Served (FCFS), Shortest Seek Time First (SSTF), SCAN (elevator algorithm), and a random disk scheduling algorithm. FCFS processes requests in the order in which they arrived and is easy to implement. SSTF processes requests with the shortest seek time from the current head position but may starve some requests. SCAN proceeds from one end of the disk, processes requests as it goes, and reverses direction once it gets to the other end of the disk. The random disk scheduling algorithm would for each step select a random request to process next. The analysis of these algorithms will not provide a definitive answer to what is the best disk scheduling algorithm since that relies entirely on what requests are being made. Instead this analysis should highlight some of the situations when one algorithm is more advantageous than another for a set of specific disk requests. There should not be any situations where the random scheduling algorithm is the best option. Instead this algorithm should act as a baseline to compare the other algorithms to as it will generally act as a worst case scenario of sorts. In order to provide a breadth of taste cases disk requests will be generated randomly using the following pseudocode:

For (int i = 0; i < Num\_Reqs; i++) {

Req[i] = rand() % (Num\_Tracks – 1);

}

where Num-Reqs is the number of disk requests for this specific test run and will be varied to better highlight how these scheduling algorithms perform for different quantities of requests. Req[] is the array of all the requested tracks on the disk starting from track(). The scheduling algorithms will be run for a series of different disc request sets with three different timing cases. In the first case all the requests are already queued at the starting time (time = 0). For the second case requests are generated dynamically while the given scheduling algorithm is running with no requests queued at starting time. The last case involved some requests already queued at the starting time and others generated dynamically while the algorithm is running. This is an important distinction because the scheduling algorithms will behave differently depending on the timing of when the requests are queued. The efficiency of a given scheduling algorithm will be determined by recording how long a given scheduling algorithm runs for with the following pseudocode:

start = clock();

Scheduling\_algorithm(Req[], Num\_Reqs);

end = clock();

run\_time = ((double) (end - start)) / CLOCKS\_PER\_SEC;

where run\_time is the difference in number of clock cycles between when the given scheduling algorithm started and ended, divided by the CLOCKS\_PER\_SEC to calculate the run time in seconds. The shorter the run time the greater that scheduling algorithm’s efficiency is. Over a large enough series of test runs with variable requests, number of requests, and timing of requests being queued we should get a more in depth understanding of the specific cases in which one disk scheduling algorithm runs more efficiently compared to the others. As previously mentioned, which algorithm is better largely depends on the specifics of the requests being made, so the best one to use will always depend on the specific situation it is being used for and should be selected accordingly.