DUQUESNE UNIVERSITY , PITTSBURGH

A PROJECT REPORT
ON

CPU SCHEDULING ALGORITHMS

SUBMITTED TO

Professor Dr. Donald Simon

BY

Rishav Shrivastava

Rajan Himanshu Jokhakar

Deepkumar Nimeshbhai Patel

October,2023

Report

The language chosen for this task is java, threads were implemented using an executable interface. Three threads were created to process the program: a file read thread, a CPU thread, and an io thread. The organization of the process was provided by the process management object in DPS. Java synchronization was done through an OS object accessed by all spawned threads. Overall, the assignment turned out to be an interesting challenge, and it was refreshing to take system-level principles from c and apply them in a different language and framework.

At the design level, thread objects handled all important aspects of execution by themselves, using PCB and OS only as containers for shared elements to simplify synchronization in critical sections. Critical section synchronization is handled by using synchronized blocks in unsynchronized methods to create mutex locks on the object being used. This will prevent two threads from using the object and keep the program running smoothly. Additionally, the critical total process counter, which increments as file reads create new PCBs, is set to volatile to ensure that it is stored in the JVM main memory instead of in one of its virtual CPU caches. This is done to ensure that the CPU and io only stop processing data when there are no more processes to work with. Admittedly, problems can still exist with this type of synchronization, especially when adding PCBs to the queue of ready devices. By just adding a new process to the queue in the file read thread and not interrupting the methods in the other two threads on this event, the processes can have shorter wait times than they were waiting for.

Analyzing the processing with the -debug argument shows that all algorithms are correctly implemented, with only a small change from one to the other with the test data. This disturbing result is probably due to the simplicity of the entry2. Txt and the speed at which the file processed all the data when read. The differences in results are likely due to the different speeds at which he creates new PCBs. This causes the wait time of processes to change, as they will not be queued - and therefore waiting - until after they are added by reading the file. A sampling of the results is available in the directory generated with the given sample input. More accurate and realistic data could have been produced with more diverse inputs, but such testing was not possible due to time constraints.

While CPU scheduling was implemented according to speciation requirements, io scheduling was left to a simple FIFO(FCFS) pipeline. At the design level, object-oriented principles were attempted with popularization. CPU clusters were given a general method of use for non-preemptive procedures - FCFS, pr - and another for preemptive ones - RR. This made it possible to keep the codebase small and make bugs much easier to spot and fix. The correctness of these implementations can be checked by adding debug to the end of the command line arguments, which provides detailed information about burst ordering within synchronized blocks.

After waiting for all threads to execute the initial thread will start processing data. Looking at the list of completed processes, the main collects data from the PCB objects and then calculates

statistics for the run. All numbers generated at the end of the execution report are of course subject to rounding errors.

RESULT:-

1. FIFO(FCFS)

```
Input File Name : input.txt
CPU Scheduling Alg : FIFO
CPU utilization : 74.77 %
Throughput : 0.158 processes / ms
Avg. Turnaround time : 170.67 ms
Avg. Waiting time in R queue : 0.0 ms
Responce time : 214.0 ms
(base) rishavshrivastava@Rishavs-MacBook-Pro JobAlgorithmTask %
```

2. PR

```
Input File Name : input.txt
CPU Scheduling Alg : PR
CPU utilization : 74.42 %
Throughput : 0.126 processes / ms
Avg. Turnaround time : 156.67 ms
Avg. Waiting time in R queue : 0.0 ms
Responce time : 215.0 ms
(base) rishavshrivastava@Rishavs-MacBook-Pro JobAlgorithmTask % java Assign5 -alg FIFO -input input.txt
```

3. RR(ROUND ROBIN)

```
Input File Name : input.txt
CPU Scheduling Alg : RR (60)
CPU utilization : 75.12 %
Throughput : 0.158 processes / ms
Avg. Turnaround time : 170.67 ms
Avg. Waiting time in R queue : 0.0 ms
Responce time : 213.0 ms
(base) rishavshrivastava@Rishavs-MacBook-Pro JobAlgorithmTask % java Assign5 -alg PR -input input.txt
```

4. Shortest job first (SJF) or shortest process next (SPN)

```
Input File Name : input.txt
CPU Scheduling Alg : SJF
CPU utilization : 75.12 %
Throughput : 0.147 processes / ms
Avg. Turnaround time : 157.86 ms
Avg. Waiting time in R queue : 0.0 ms
Responce time : 213.0 ms
(base) rishavshrivastava@Rishavs-MacBook-Pro JobAlgorithmTask % java Assign5 -alg RR -quantum 60 -input input.txt
```