CPU Scheduling Algorithms

CSCE – 5640.003
FALL 2024

Professor Amar M. Maharjan
Department of Computer Science
University of North Texas

Shreya Sri Bearelly (11734229)
Divya Sree Dandu (11594287)
Ganesh G. (11700551)
Vikas Varala (11704793)

Overview:

This project is going to focus on the process scheduling and the research of the algorithms of process scheduling, which is important in operating systems for orderly and efficient handling of multiple processes. Process scheduling is the function of allocating the CPU to the various processes in a way most suited in the CPU time-utilization perspective.

On this project, a scheduler takes tasks from a file where each task has a name, priority, and its CPU burst time. The rationale for this is to deploy and evaluate the effectiveness of several scheduling algorithms outlined below as per different performance indicators. The efficiency of usage of this CPU relies on the implementation of the algorithms which this project focuses on discussing and evaluating. Included scheduling algorithms are First-Come, First-Served (FCFS), Shortest-Job-First (SJF) and Priority Scheduling, Round Robin and Priority with Round Robin.

Algorithms to be Implemented:

1. First-Come, First-Served (FCFS): The processes are arrived in the system and are queued according to their arrival time regardless of their CPU burst time or even priority.

2. Shortest Job First (SJF): Schedules processes according to the order of the burst time on CPU, in order to reduce the waiting and turn-around time.

3. Priority Scheduling: Tasks are planned depending on the characteristics of priorities where priority ends represent priority indexes.

4. Round-Robin (RR): Each process is allocated a fixed amount of time allocation or a ‘quantum’ through which all the processes get some of the CPU time in rotation.

5. Priority with Round-Robin: Tasks are being planned according to the priority level, and tasks having the same priority level are in turn rotated among the units.

Problem Statement:

The goal is to provide necessary modifications to the way the CPU time is allocated in order to increase its throughput, decrease response time and initiative wait time for processes in order to improve the system performance. The problem, which is a variation of those frequently met in operating systems, is how to distribute CPU time fairly between processes so that the overall processing capability may be limited, and the number of processes is large.

Problems Encountered in CPU Scheduling:

1. Resource Contention: When similar processes are competing for CPU time then there are contentions and this will slow down the CPU.
2. Process Starvation: In some algorithms, the low priority processes do not get CPU time at all if the high priority processes keep coming in incessantly.
3. Poor CPU Utilization: An inefficient scheduler might allow the CPU to remain dormant while tasks remain waiting or could allow certain tasks to continuously consume the CPU while the other tasks remain starved, a poor throughput.
4. Long Waiting and Turnaround Times: Certain scheduling methods may cause some process to stay for a longer time in queue and this will influence the system response time and performance.
It is the goal of this project, by employing various algorithms, to determine the optimal schemes that perform effectively under each workload to determine specific trade-off characteristics of each of the algorithms used.

Importance:

CPU scheduling is integral in operation systems meaning that there is much difference in the effectiveness of process execution. Due to the similar fashion in designing different algorithms, when and how they are implemented can be determined effectively, showing advantages and shortcomings of each algorithm to best schedule in different contexts. Organisational schedules are fundamental to optimise execution time and meeting high, demand response of Multi-user and real time systems. An optimized scheduling approach can provide significant benefits:

1. Reduced Waiting and Turnaround Times: Potential solutions for a system include managing how long processes remain in a queue, for the best efficiency to ensure the user gets the quickest and most efficient service, reducing a systems load.
2. Better CPU Utilization: The CPU can be utilized with more efficiency and the CPU usage is spent more times on the CPU than it is not.
3. Enhanced Scalability: One advantage of the latter is that environments that have more of them can induce a higher load onto the well-designed scheduler.
4. Process Fairness: Some scheduling algorithms (round-robin in particular) prevent such problems as starvation or monopolization of CPU by processes.
5. Adaptability: Thus, in comparing with other algorithms, this project is able to determine the most appropriate scheduling strategies compatible with different operating environments, including real-time, interactance, and background.

Detailed Breakdown of Each Algorithm:

1. First-Come, First-Served (FCFS):
Description: FCFS is the easiest to understand of the scheduling algorithms. Tasks are assigned to processes based on the first in, first out fashion without considering the duration that the CPU burst time or the priorities of the processes.
Pros: Easy to implement, it is fairly used in a basic sense where all the processes are given CPU time based on arrival.
Cons: Results in a system that has a long waiting time, especially if the processes included in the system have long burst times and requiring processing first (convoy effect).
2. Shortest Job First (SJF):
Description: SJF focuses on short CPU burst time first so as to minimize the average waiting time.
Pros: Normally, this leads to a better average wait time and a better average turnaround time.
Cons: Sometimes hard to be enforced if the CPU bursts are unpredictable, and the longer tasks can be starved for a long time.
3. Priority Scheduling:
Description: When tasks are assigned to the system, they are all assigned to be completed in order of priority, the higher the priority the higher the chance the CPU will work on the task than the lower priority tasks.
Pros: Increased opportunities for more critical tasks to be processed.
Cons: Can easily starve low priority processes, in case high priority tasks continue to arrive frequently.
4. Round-Robin (RR):
Description: To provide pipelining, time-sliced pre-emptive scheduling is use, in which processes are assigned a specific time quantum (e.g., 10ms) and the scheduler passes through the set of processes in a cyclic manner allocating each quantum of CPU time to the processes.
Pros: Scheduling was fairly-defined with respect to distributing CPU time; there will be no process, which turns out all the time waiting.
Cons: May result in more response times if there are many such processes and performance is highly sensitive to quantum length.
5. Priority with Round-Robin:
Description: Pretends both priority scheduling and round robinate. More critical processes are slotted at the top, while equal priority processes run on the round-robin method.
Pros: Enables processes that require CPU time to do so while at the same time protecting the processes with the same priority level.

Cons: Slightly more complex to implement; but, the implementation makes many high priority task to compete with the low priority task leading to extremely long waiting times for the low priority task.

Background:

Understanding CPU Scheduling Algorithms:

CPU scheduling is a process by which the CPU resources are allocated with processes depending on certain characteristics. Key algorithms include:

- FCFS: Burst time and priority do not affect the processing of the processes and they are processed as they arrive.
- SJF: It selects the task that has the shortest CPU burst time, thus low total average turnaround time.
- Priority Scheduling: Servicing the highest prioritized process first; the ties are counter preferenced by the FCFS.
- Round-Robin: Every task considers a time quantum; later, the sequent process is initiated, making an efficient distribution of the CPU time.
- Priority with Round-Robin: Works with priority scheduling algorithm for the real time processes which are prioritized while using Round-Robin for any process having equal priority with other but critical.

Example test case for all scheduling algorithms with 5 processors:

This is how we solved below examples to find out average wait time and also provided Gantt chart.

FCFS:

Processors	Priority	Burst time
P1	2	15
P2	1	10
P3	3	5
P4	4	20
P5	2	8

Turnaround time= completion time – arrival time (arrival time will be considered 0 because it is not mentioned in the above table)

Wait time = Turnaround time – burst time

Average wait time = wait time/ no. of processors

Processors	Priority	Burst time	Turnaround time	Wait time
P1	2	15	15	0
P2	1	10	25	15
P3	3	5	30	25
P4	4	20	50	30
P5	2	8	58	50

Average wait time = 0+15+25+30+50/5=120/5=24
Order of Execution: P1 → P2 → P3 → P4 → P5
| P1 | P2 | P3 | P4 | P5 |
0 15 25 30 50 58

SJF:

Processors	Priority	Burst time
P1	2	15
P2	1	10
P3	3	5
P4	4	20
P5	2	8

Turnaround time= completion time – arrival time (arrival time will be considered 0 because it is not mentioned in the above table)
Wait time = Turnaround time – burst time
Average wait time = wait time/ no. of processors

Processors	Priority	Burst time	Turnaround time	Wait time
P1	2	15	38	23
P2	1	10	23	13
P3	3	5	5	0
P4	4	20	58	38
P5	2	8	13	5

Average wait time = 23+13+0+38+5/5=79/5=15.8
Order of Execution: P3 → P5 → P2 → P1 → P4
| P3 | P5 | P2 | P1 | P4 |
0 5 13 23 38 58

Priority (highest Priority comes first):

Processors	Priority	Burst time
P1	2	15
P2	1	10
P3	3	5
P4	4	20
P5	2	8

Turnaround time= completion time – arrival time (arrival time will be considered 0 because it is not mentioned in the above table)
Wait time = Turnaround time – burst time
Average wait time = wait time/ no. of processors

Processors	Priority	Burst time	Turnaround time	Wait time
P1	2	15	48	33
P2	1	10	58	48
P3	3	5	25	20
P4	4	20	20	0
P5	2	8	33	25

Average wait time = 33+48+20+0+25/5=126/5=25.2
Order of Execution: P4 → P3 → P5 → P1 → P2
| P4 | P3 | P5 | P1 | P2 |
0 20 25 33 48 58

Round Robin (time quantum =10):

Processors	Priority	Burst time
P1	2	15
P2	1	10
P3	3	5
P4	4	20
P5	2	8

Turnaround time= completion time – arrival time (arrival time will be considered 0 because it is not mentioned in the above table)
Wait time = Turnaround time – burst time
Average wait time = wait time/ no. of processors

Processors	Priority	Burst time	Turnaround time	Wait time
P1	2	15	48	33
P2	1	10	20	10
P3	3	5	25	20
P4	4	20	58	38
P5	2	8	43	35

Average wait time = 33+10+20+38+25/5=136/5=27.2
Order of Execution: P1 → P2 → P3 → P4 → P5
| P1 | P2 | P3 | P4 | P5 | P1 | P4 |
0 10 20 25 35 43 48 58

Priority with Round Robin (highest priority comes first and time quantum as 10)

Processors	Priority	Burst time
P1	2	15
P2	1	10
P3	3	5
P4	4	20
P5	2	8

Turnaround time= completion time – arrival time (arrival time will be considered 0 because it is not mentioned in the above table)
Wait time = Turnaround time – burst time
Average wait time = wait time/ no. of processors

Processors	Priority	Burst time	Turnaround time	Wait time
P1	2	15	58	43
P2	1	10	43	33
P3	3	5	15	10
P4	4	20	53	33
P5	2	8	23	15

Average wait time = 43+33+10+33+15/5=135/5=27
Order of Execution: P4 → P3 → P5 → P1 → P2→ P4 → P1
| P4| P3| P5 | P1 | P2 | P4 | P1 |
0 10 15 23 33 43 53 58

Implementation:

Solution Overview:

Both scheduling algorithms were coded in C++ and C programming languages and examined in the Linux/Ubuntu environment for their performance level and compatibility with scheduling tools.

FCFS and SJF Algorithms:
It was Shreya Sri Bearelly working on them, these algorithms divide processes with the help of techniques such as arrival time and smallest burst time.

Priority and Priority with Round-Robin Algorithms:
Designed by Divya Sree Dandu, these rank processes based on the priority assigned with the Round-Robin variant it to deal with tie-holders.

Round-Robin and File Reading:
Implemented by Ganesh G. the Round-Robin is a scheduling algorithm, which serves each process in equal time quantum.

Testing and Analysis:
The test cases and performance analysis were provided by Vikas Varala.

Implementation Details:

Programming Language: C++ and C.
Operating System: Linux/Ubuntu.
Test Cases: Included different configuration with the process count and priority setting by using the custom test file.
5 files with 6 processes.
5 files with 10 processes.
5 files with 16 processes.
Example Test Input Files: schedule1Ex.txt, schedule2Ex.txt…. schedule15Ex.txt etc.

FCFS Implementation:

This program takes inputs in form of tasks (processes) from one or several files according to the command line input. Each of the tasks has a name, priority and the time duration within which it can be executed as a burst. The program uses the First-Come First-Served (FCFS) scheduling technique which implies that the tasks are completed in the order that they were received. It computes the waiting and turnaround times of every task, after which the program will show these values together with the mean waiting time. This algorithm used C++ as programming language for implementation.
Puts the waiting time for the current task waiting time to be equal the current clock time current time.
The time to complete the task turnaround time equals to the current clock current time plus the backup clock burst time.
Then adds the burst time of the task to the current time to have an update on the time for the next task.
The code is used to read tasks from files and is also implemented with First Come First Served scheduling of tasks, where tasks are done in the order they arrived. Following this, it shows the waiting time and turnaround time and the mean waiting time is also computed. FCFS is simplistic, and heavily penalizes tasks with large burst times ("convoy effect"). This program is once again useful for calculating such behavior with various task sets.

SJF Implementation:

A program uses a file to read tasks and each task contains the name, priority and burst time. It then used the Shortest Job First (SJF) scheduling algorithm regime in which the tasks with shorter burst times are processed first. The program gives waiting and turnaround time of each process and then shows the average waiting time. This algorithm used C as programming language for implementation.
This one sets waiting time as the cumulative burst time of previous tasks – the current time as of this instruction. Updating the set turnaround time by adding the burst of the task into waiting time. Adds burst time of the current task to current Time.
Tasks are issued from a file and SJF Scheduling is used, where tasks with comparatively smaller burst time are favored. After scheduling, it shows the waiting and turnaround time and the average of the waiting time that is actually spent. This

makes it useful for evaluating SJF scheduling, in which the average waiting time for different jobs which have different burst times are minimized.

Priority Implementation:

The program then processes the tasks that have been introduced in a file and each task has a name, priority and burst time. Then, it is allowed to process tasks with Priority Scheduling where many tasks with high priority values are processed. At this point, the program prints out the waiting as well as the turnaround time and general average waiting time. This algorithm used C++ as programming language for implementation.
Gives the waiting time of the first task as 0 and the turnaround times of the first task same as its burst time.
For each subsequent task Calculates the waiting time by summing the waiting time and burst time of the preceding task.
Computes the turnaround time for current working task as a sum of waiting time and burst time.
The first source code reads tasks from a file and simulates the Priority Scheduling algorithm by completing higher priority tasks before lower priority ones. The processed data shown to the client consist of average waiting times for comparison and analysis with samples. This is ideal for the assessment of priority scheduling with regards to waiting time indices.

Round Robin Implementation:

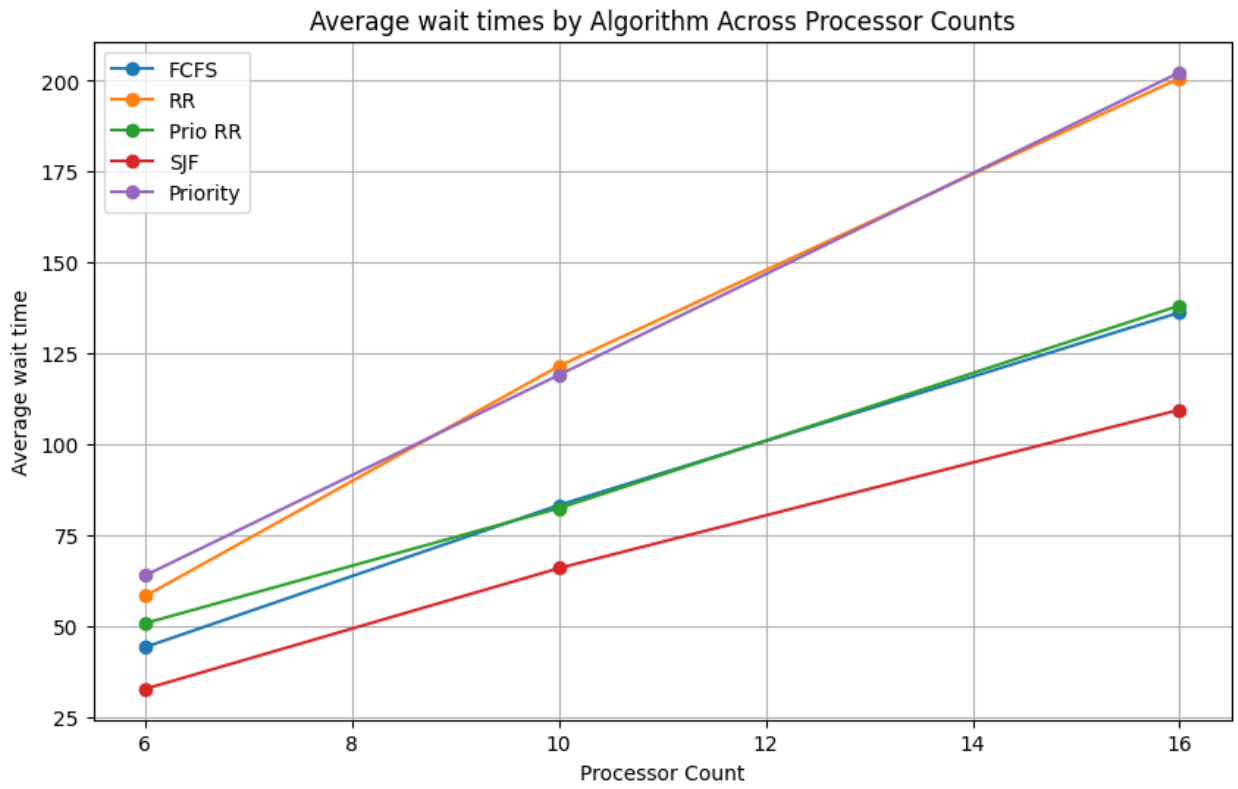
The program is instantiated with the tasks taken from file and each task has its name, its priority level and burst time. It then implements Round Robin scheduling algorithm with no regard to priority and with a time quantum of 10 milliseconds. The code then determines waiting and turnaround times for all tasks and finally presents the scheduling outcomes on average number of waiting time of the files. This algorithm used C++ as programming language for implementation.
Round-Robin Processing:
Spends a maximum of current time (10 milliseconds) processing each task in the queue at a time. If the task has more remaining time than current time it Subtracts current time from the remaining time and then adds current time to the current time. The task is then placed back on the queue to be synthesized again at some other time. If the task's remaining time is less than or equal to current time, Task is over in this round. The waiting time is determined as the current system time – the initial burst time, or as the current system time obtained at the moment when all the tasks are done, concerning the turnaround time.

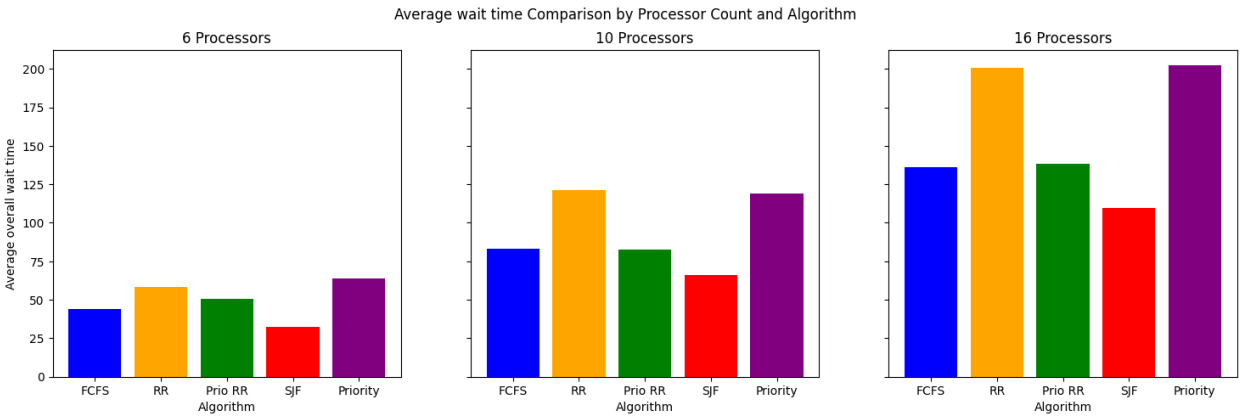
Scheduling tasks include reading scheduling from files and implementation of Round Robin scheduling with time slice of 10 milliseconds. It gives the result of each file with mean waiting time, which helps to analyze the round-robin process with the shown time quantum for scheduling.

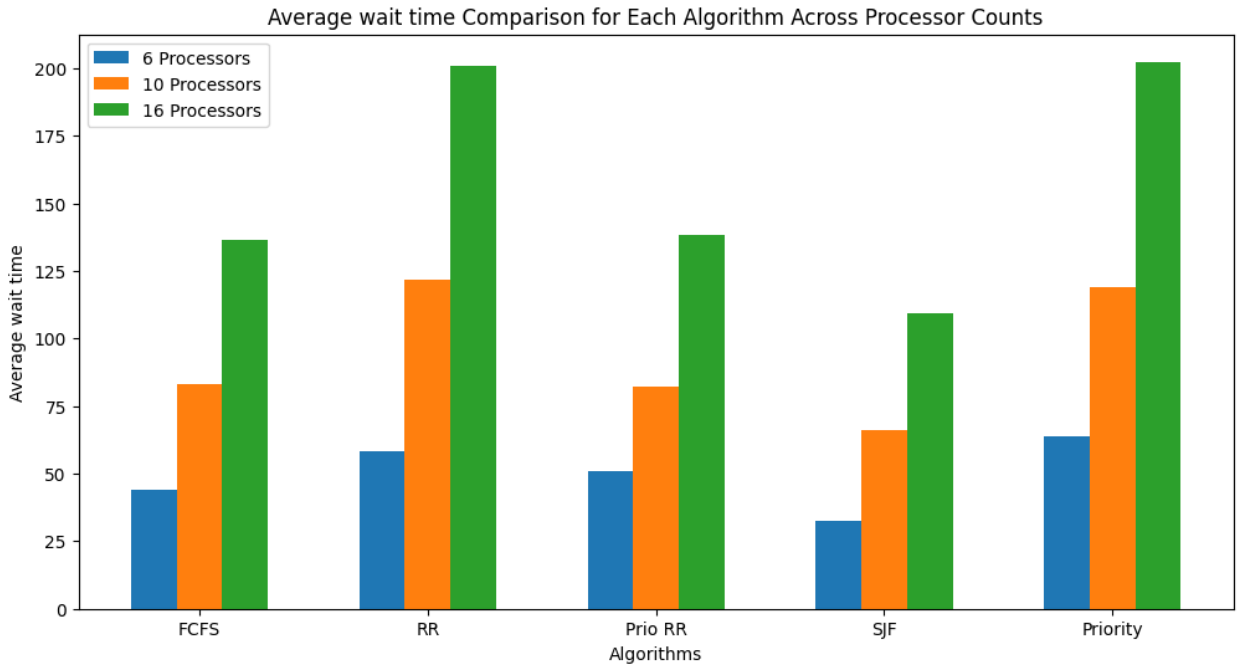
Priority with Round Robin Implementation:

The program accepts tasks from a file, by each task having a name, priority and burst time. It then uses a Priority with Round Robin scheduling algorithm; in this, I assign a priority value to the task then executes the task with high priority first; if the task has the same priority value, then the kernel proceeds in a round-robin method with a quantum time of 10 milliseconds. This algorithm used C++ as programming language for implementation.
Performs each task in the queue for a maximum of 10 milliseconds at a time. In case the task possesses more time than current time, the program takes away current time from remaining burst time and adds the same to time. The task is then re-enqueued. If time t for the task is less than or equal to current time, the task finishes in this round. Waiting time refers to current time minus the initial burst time, while turnaround time is set with current time after the completion of a particular task.
For each test file provided as a command-line argument, the output will be a table showing: The task name, its priority, the burst time of each task and its waiting and turnaround times. It also shows the result of each file, mean of waiting time etc, which makes it more convenient in analyzing the performance of this scheduling algorithm.

Experimental Results:
Outputs plotted in graphs:







Tools used to generate graphs:

a. We have used matplotlib to generate graphs.
b. We have used Numpy to generate graphs.
c. We have used Pandas to generate graphs.

Insights to be gained from graphs:

Here, when the processor count increases from 6,10,16., the wait times also tend to increase too which shows us that the high processor count will be resulting in a significant amount of workload. Yet, SJF and priority with RR has the least increase in wait times which can imply us that they scale relatively good when bumping up the processor count, and out of all the round robin has the largest increase which shows it can struggle with heavy tasks.

1. If our primary concern is “minimizing the overall wait times”, then it is proven that the SJF algorithm is the best choice to go for.
2. if a much more priority-based handling is needed, then priority scheduling with round robin can be a good choice because it achieves good amount of balance between the performance and wait times.

Algorithm	6 processors-Avg Wait time	10 processors-Avg Wait time	16 processors-Avg Wait time
FCFS	44.1668	83.24	136.2626
SJF	32.668	65.88	109.488
Priority	50.2	81.66	138.1
Round Robin	58.204	121.5	200.712
Priority with Round Robin	50.733	82.44	138.1622

Calculated overall average waiting time/ no of examples to find out 6,10,16 processors average waiting time.
Looking at the results we can see that for 6,10 and 16 Processors **SJF** has the lowest Average Wait Time. Considering our processors SJF works best out of all the 5 scheduling algorithms.

Results Overview:
In all the test cases the efficiency of each algorithm was computed based on the average waiting time and the average turnaround time. For each algorithm, written output was captured in tables.

Data Interpretation:

b) Interpretation of the Results
The study on the result shows that the average wait time increases as the number of processors increase from 6 to 16 which suggests that the increase in the number of processors is conducive to the extra load. However, the change in rate of increase in waiting times depends with the particular algorithm used.

For situations in which number of processors increases, the Round Robin (RR) algorithms exhibit the maximum raise in average wait time, whereas Prio RR reveals the lowest hike. This implies out of both algorithms, especially the Prio RR, may perform very good compared to their actual performance when the load increases, than the round robin which is very effective.

SJF (Shortest Job First): This algorithm demonstrates the lowest wait time throughout all offered processor amount, making it the most suitable one whenever the reduction of wait time is critical. Considering that the testing of SJF does not demonstrate much fluctuation with increases in the number of processors, it appears

well suited for use in environments where optimization of processor use is paramount and waiting times should be kept to a bare minimum.

Priority Scheduling: As it can also be seen, the wait times from this algorithm are one of most higher wait times. Unlike SJF in which wait times are the shortest for arriving tasks, Priority is reasonably highest in dealing with arriving tasks while still prioritizing the tasks to be processed as per waiting time alone is not always the priority. Here , the performance of the algorithm will gradually decline as the processor count increases which is mostly likely due to the most freq switches or maybe due to the wait for high priority tasks which are the factors that increase the overall wait time.

Round Robin (RR): While RR is equitable in giving out the time slice to be served by the processor to various tasks, its wait time escalates with the rise in the number of processors. This implies that RR is not so efficient under conditions of high loads, which means that it is better to use it for lighter or for balanced loading processes. Managing time slices is especially important in RR to avoid too long waits of processes under high number of processors.

Comparison of algorithm:

On the other hand, SJF is less likely to be fair in terms of allocations of tasks because those tasks are allocated by short job duration while the waiting time is also less in this scheduling technique because it is the best when waiting time is more important than fairness. So, priority scheduling stands in between of the first two as it serves tasks sorted by their priority but keeps waiting time maximums low. This makes it a good fit for use in workloads in which some-sort of priority is essential, but not intense efficiency is required. Therefore, with relation to the wait time, SJF should be adopted while Priority has the best overall performance of the various scheduling algorithms. Round Robin can be selected mainly for the equally distribution of the tasks, although the time slice has to be managed in order not to have long waits when using different processors as the number of them grows.

Conclusion:

The project was able to effectively teach and test different forms of CPU scheduling algorithms where it was proved that each form as its strength and weakness. From the outcomes determined, it was clear that the SJF and Priority management system with the Round-Robin policies offered the best results in most tests conducted, the Round-Robin in particular, offering fair ground as far as the CPU time and the Processes are concerned. Furthermore, the comparison highlighted the drawbacks of

the various CPU scheduling techniques; for example, SJF incurred the lowest waiting time at the cost of starvation of the longer processes; Round-Robin was fair, but there was an increased overhead as a result of the frequent context switches. In conclusion, the comparative analysis presented here is advantageous in identifying suitable scheduling algorithms related to particular system constraints and achieving a high efficiency/fairness trade-off as well as minimizing response time.

Accomplishments:
The project efficiently demonstrated and evaluated various CPU scheduling algorithms explaining merits and demerits of each. From the results obtained it was evident that both the SJF and Priority with Round-Robin policies provided the best performance in most cases and Round-Robin made it fair.

Future Work:
The future studies could incorporate more flexible scheduling algorithms which can learn from the working dynamics in a system and might advance the ability to handle variable loads and changing priorities.

References:
https://www.wiley.com/en-us/Operating+System+Concepts,+10th+Edition-p-9781119320913