ADVANCED OPERATING SYSTEM PROJECT

PROCESS SCHEDULING SIMULATION

FANTASTIC FOUR

TEAM NUMBER : 3

MENTOR NAME
RISHABH MALIK

PREPARED DATE
Nov 25, 2021

SUBMISSION. DATE
Nov 27, 2021

MEMBER NAME	ROLL NUMBER
NITIN KUMAR	2021202020
PRATIK HAWARE	2021201001
SAI TEJA GUDALA	2021201040
SAI VARUN REDDY BHAVANAM	2021201026

THIS IS A GROUP BASED PROJECT SUBMITTED UNDER FOR THE COURSE ADVANCED OPERATING SYSTEM

UNDERTAKEN DURING FIRST SEMESTER OF POST-GRADUATION (M.TECH) PROGRAMME IN

INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY, HYDERABAD.

1. The project has been done on our own, no copy or version of this has been taken from any source.

Introduction

A program is just a bunch of instructions (and maybe some static data) which sits on the disk. A process is defined as a program which is currently executing. A processor (CPU core) can run a single program at a time by executing its instructions one by one. A typical computer has few physical CPU cores (ranging from 2 to 8). However, at a time, a few dozen or even a few hundreds of processes may be running in a computer system. How is this possible when a single CPU can run only a single process at a time? The answer is Process Scheduling.

Problem Description

We have a few physical CPU cores but we want to provide the illusion of a nearly endless supply of CPUs. The OS creates this illusion by virtualizing the CPU. By running one process, then stopping it and running another, and so forth, the OS can promote the illusion that many virtual CPUs exist when in fact there is only one physical CPU (or a few). This basic technique is known as time sharing. The cost of this technique is performance. Each process will run more slowly if the CPU must be shared.

All processes are not equal. Some processes need to be executed as fast as possible. Example: Braking mechanism in self driving cars. Some processes need to finish their execution before a deadline. Example: A robot may be expected to respond to stimuli within 2 seconds, Multiplayer games. Interactive processes need CPU time regularly or they may appear unresponsive. Compilations of large programs can be allowed to take more time than usual as they are not usually considered time critical. All the above requirements can be satisfied by designing a well thought scheduling policy. A scheduling policy can be implemented by choosing the right scheduling algorithms.

Scheduling algorithms are thus very critical to any computer system. They have a huge impact on the system performance.

Solution Approach

We decided to implement 8 scheduling algorithms in C++. Our programs read input from a CSV file and write their output to another CSV file. The performance of different scheduling algorithms

can be compared by scheduling metrics. A metric is just something we use to measure something. We used the following scheduling metrics:

1. Turn Around Time (TAT): TAT = T(Completion) - T(Arrival)
2. Response Time (RT) : Time after arrival when the process is scheduled by the CPU for the first time
3. Waiting Time (WT): Time for which the process is ready and waiting for its turn to be scheduled by the CPU.
4. Completion Time (CT): Process completion time.

We ran our algorithms on the same input and recorded these statistics for each process. Further, we changed the parameters of our algorithms to see their effect on the above 4 metrics in the following ways

1. In round robin, we change the time slice for each process
2. In priority scheduling, we saw the effects of giving high priority to processes with low priority numbers and vice versa.
3. In FCFS (First Come First Serve), we explored the convoy effect

These statistics were then visualized using Line graphs, Bar graphs and Gantt charts. Python was used for data visualization. Our Python scripts take as input the CSV files generated by our C++ code.

Work Distribution

Each team member took responsibility for implementing 2 process scheduling algorithms.

Sai Varun Reddy Bhavanam

Shortest Job First (Non-Pre-emptive) (SJF):

1. Out of all the available processes during scheduling, the process with least burst time is considered.
2. In case of a tie, one of the tied processes with lowest process ID is considered.
3. Since this is a non-preemptive version, the process is run until completion or when I/O request is to be handled by the process.
4. It doesn't take into account the waiting time of processes, instead it only considers burst times of the processes. So, if a process with larger burst time comes early, it will continue to wait until all the available processes complete execution.

Alternatives:

A. We can use the algorithm which takes into account waiting time as well. Highest Response Ratio next (HRRN) is one such algorithm which will schedule the processes with **highest response ratio** and response ratio is calculated as

$$response\ ratio = \frac{waiting\ time\ of\ a\ process\ so\ far + estimated\ run\ time}{estimated\ run\ time} = 1 + \frac{waiting\ time\ of\ a\ process\ so\ far}{estimated\ run\ time}$$

B. This will give priority to both waiting time of process and estimated run time (burst time)

Shortest Job First (Pre-emptive) or Shortest Remaining Time First (SRTF):

1. Out of all the available processes during scheduling, the process with least remaining time is considered.
2. In case of a tie, one of the tied processes with lowest process ID is considered.
3. Since this is a preemptive version, if a process which has **strictly shorter remaining time** than the present process comes into queue, the present running process is preempted and the newly arrived shorter process is scheduled.

Advantages:

C. **Theoretically,** this is the best algorithm (optimal algorithm)
D. It acts as benchmark for other scheduling algorithms when comparing performances

Drawbacks:

A. Burst times of processes are not known beforehand and hence it is difficult to implement this algorithm practically.
B. **SRTF** has more context switches and scheduling overheads than **SJF** as processes are more frequently preempted in **SRTF**
C. This algorithm (both preemptive and non-preemptive versions) can cause starvation to processes with higher burst times.

Alternatives:

A. We can predict the burst times of processes using various techniques
    a. **Static techniques**
        i. Using this we can predict the **total burst time**
        ii. It is predicted by using
            1. **Size of process**
            2. **Type of process**
                a. Foreground processes are given more priority than background processes
    b. **Dynamic techniques**
        i. Using this we can predict the **next CPU burst**
        ii. It is predicted by using

1. **Simple averaging**
2. **Exponential averaging**

Sai Teja Gudala

First Come First Serve Scheduling:

1. FCFS schedules the processes according to their arrival time and is a non preemptive method, so the process will run till the completion once it gets started.
2. In this algorithm, processes which request the CPU first get the CPU allocation first.
3. The processes in the ready queue are sorted based on the arrival times and the process which has the least arrival time executes first.
4. A process which arrives first is selected for scheduling among all other processes in the ready queue.
5. In case of conflict, a process with the minimum process id is selected. In this way all the Processes are selected from the ready queue.
6. After the selection of a process, the process is executed completely and then the CPU takes the next process from the ready queue. In this way the processes get executed from the ready queue.

Drawback:

Whenever there is a process with large burst time before the process with small burst time, the latter process can get CPU time only after the former process has finished. This property leads to a situation called Convoy Effect (Shown in Results).

The Convoy effect occurs when a process is allocated to the CPU and will never release the CPU until it finishes executing, so the waiting time of the processes after this process is high, i.e short processes which are at the back of the queue have to wait for the long process at the front to finish.

Round Robin Scheduling:

1. Round Robin is a preemptive scheduling algorithm.
2. Each process is assigned a fixed time (Time Quantum/Time Slice) in a cyclic way**.**To implement Round Robin scheduling, we keep the ready queue as a FIFO queue of processes. The CPU scheduler picks the first process from the ready queue and executes it for 1-time quantum, and then changes the process.
3. If the process burst time is less than 1-time quantum then it will implicitly get the other process from the ready queue and remove it from the ready queue, else it will be added at the end of the queue to complete its execution.
4. The Program is tested for various values of Time quanta, to understand it's working under different conditions.
5. Advantage of this process is that it gives very less average response time.

6. The processes also do not suffer from starvation, as all processes get a fair share of CPU time for their execution.

Drawback:

The only drawback in this algorithm is, it is difficult to find the perfect time quantum because if the time quantum is high then response time is higher and if the time quantum is low then the context switches are higher.

Pratik Haware

MultiLevel Queue

1. Normally, we have a single ready queue for processes. When the processor is available, it selects a process from this queue
2. However, not all processes are equal. Some system processes or real time processes may need to be executed as soon as possible. They should have minimum turnaround time. If these processes are not executed quickly, they may cause significant damage to human life and property. Example: Braking system in self driving cars
3. Interactive processes should regularly get some CPU time or else they will appear unresponsive to the user.
4. Background processes which are not critical and don't require user interaction (also called Batch processes) can be scheduled when there are no critical processes in the system
5. Hence, it makes sense to have separate ready queues for system, interactive and batch processes.
6. Higher priority should be given to the system process queue. When this queue is empty, then we can start to schedule the interactive process queue. Finally, batch process queue can be scheduled when the higher priority queues are empty
7. System processes queue is scheduled using highest priority first algorithm
8. Round robin scheduling is used for interactive process queue so that all processes get equal time slices and no process seems unresponsive to the user.
9. First Come First Serve (FCFS) is used for batch process queue because their turnaround time is not critical

Drawbacks:

1. Some processes may get starved for CPU time if the queue for high priority processes do not get empty. In above implementation, batch processes will never be scheduled if system and interactive processes keep on coming
2. It is inflexible

Alternatives:

Multilevel Feedback Queue

MultiLevel Feedback Queue (MLFQ)

1. The aim of MLFQ is to optimize turnaround time and minimize response time. The OS doesn't know in advance how long the job will run. Hence Shortest Job First (SJF) cannot be used practically
2. MLFQ has a number of distinct queues, each assigned a different priority level. At any given time, a job that is ready to run is on a single queue. MLFQ uses priorities to decide which job should run at a given time: a job with higher priority (i.e., a job on a higher queue) is chosen to run.
3. The key to MLFQ scheduling therefore lies in how the scheduler sets priorities. Rather than giving a fixed priority to each job, MLFQ varies the priority of a job based on its observed behavior
4. If, for example, a job repeatedly relinquishes the CPU while waiting for input from the keyboard, MLFQ will keep its priority high, as this is how an interactive process might behave.
5. If, instead, a job uses the CPU intensively for long periods of time, MLFQ will reduce its priority. In this way, MLFQ will try to learn about processes as they run, and thus use the history of the job to predict its future behavior
6. When a job enters the system, it is placed at the highest priority (the topmost queue).
7. If a job uses up an entire time slice while running, its priority is reduced (i.e., it moves down one queue).
8. Processes in a queue will be scheduled only if the queues above it are empty.

Drawbacks:

1. It is the most complicated scheduling algorithm because it requires some means of selecting values for all the parameters to define the best scheduler

Nitin Kumar

Priority Scheduling (Non Preemptive):

1. Each Process has a priority associated with it.
2. A Process with highest priority is selected for scheduling among all other processes in the ready queue.
3. In case of Conflict (Two processes having same priority) then the process with minimum arrival time is selected.
4. If there is (Two processes with same priority and same arrival time) then the process with minimum process ID is selected.
5. Once the process occupies the CPU it will not leave it until it's completion or till it needs some IO operation even if a new process comes which has higher priority.

6. So there can be a case that process which needs CPU instantly (as it has more critical task to do) has to wait, like internal system chipset errors, memory corruption problems, parity errors and high-level errors needing immediate attention which has to be serviced immediately will have to wait for a process to complete which has lower priority.
7. Time Complexity: O(nlog(n)), where n is the number of processes which are to be scheduled.

Advantages of priority:

A. Provides a mechanism to prioritise processes based on their role and not just on other metrics like arrival time and burst time, we can give relative importance to process and scheduling can be based on that.
B. Easy to implement.

Drawback of priority:

A. Inability to allow a higher priority task to execute if CPU is occupied by a lower Priority job until completion of lower priority job.
B. Starvation, Low priority process has to wait indefinitely.
C. In case of system crash all low priority processes are lost.

Alternative:

Use of Preemptive version of algorithm.

Priority Scheduling (Preemptive):

1. Each Process has a priority associated with it.
2. A Process with highest priority is selected for scheduling among all other processes in the ready queue.
3. In case of Conflict (Two processes having same priority) then the process with minimum arrival time is selected.
4. If there is (Two processes with same priority and same arrival time) then the process with minimum process ID is selected.
5. Once the process occupies the CPU it is allowed to preempt the CPU if a new process comes which has higher priority.
6. In case of Preemptive Priority Scheduling algorithm a process may not occupy the CPU until it's execution completes in one go.
7. Time Complexity: O(nlog(n)), where n is the number of processes which are to be scheduled.

Advantages of priority:

A. Provides a mechanism to prioritise processes based on their role and not just on other metrics like arrival time and burst time, we can give relative importance to process and scheduling can be based on that.
B. Easy to implement.

Drawback of priority:

A. Priority Algorithm is unfair for I/O bound processes as they are given less priority.
B. Starvation, Low priority process has to wait indefinitely.

C. In case of system crash all low priority processes are lost.

Alternative:

Use of Completely Fair Scheduler algorithm as it does elegant handling of both I/O and CPU bound processes and is also more practical and implemented in LINUX Based Operating system.

Problems Faced and Shortcomings

1. It was initially difficult to plot graphs using matplotlib in Python. It is important to select the right backend or allow Python to select it for you. We took a lot of time to realize that we can ask Python to do this job for us
2. Deciding which algorithms to use in Multi level queue can Multilevel feedback queue. There are no right or wrong answers. It depends on your policies.
3. Faced Problem in making all the code work together and in a synchronous manner.

Learnings

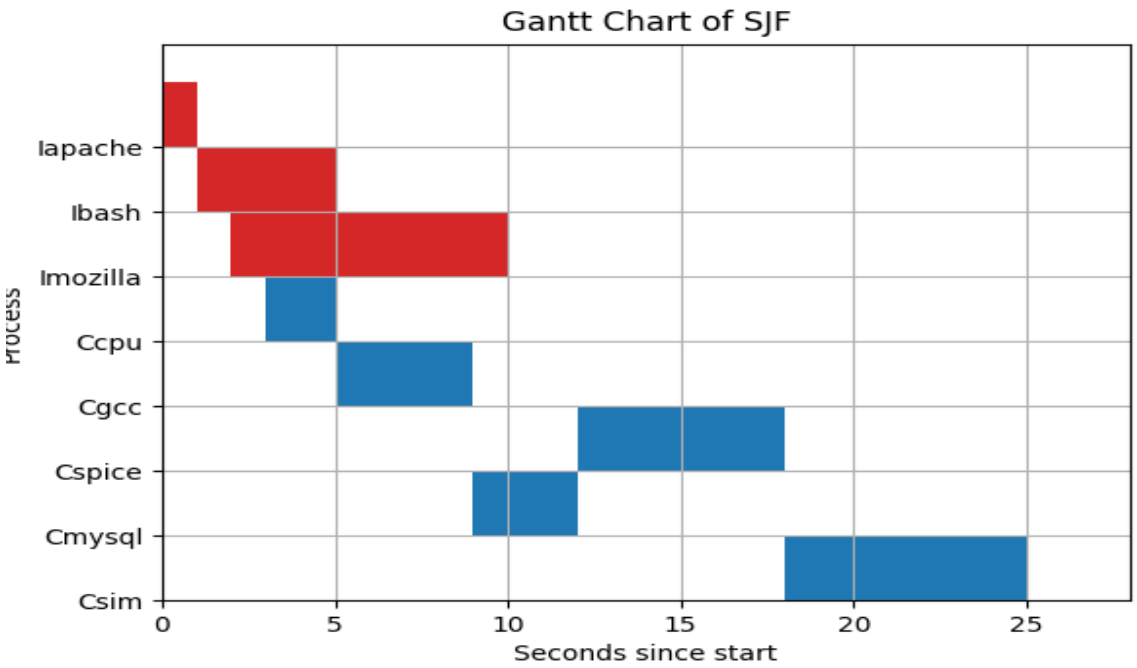
1. We learned the details of process scheduling algorithms and the data structures required to support them.
2. We learned to implement the process scheduling algorithms in C++
3. We learned how different scheduling algorithms impact the scheduling metrics (turnaround time, completion time, response time, waiting time) of processes. This shows how important scheduling policies are for system performance.
4. Further, we also learned how changing the parameters of scheduling algorithms (Example: Round Robin time slice) affects the scheduling metrics. Thus along with selecting the right scheduling algorithms, tuning the parameters of the algorithms is also equally important
5. We learned to visualize statistics by plotting them using Python and the matplotlib library
6. We learned how to work together and the technologies which make coding together easy (Git, Github, Discord)

Results

Shortest Job First

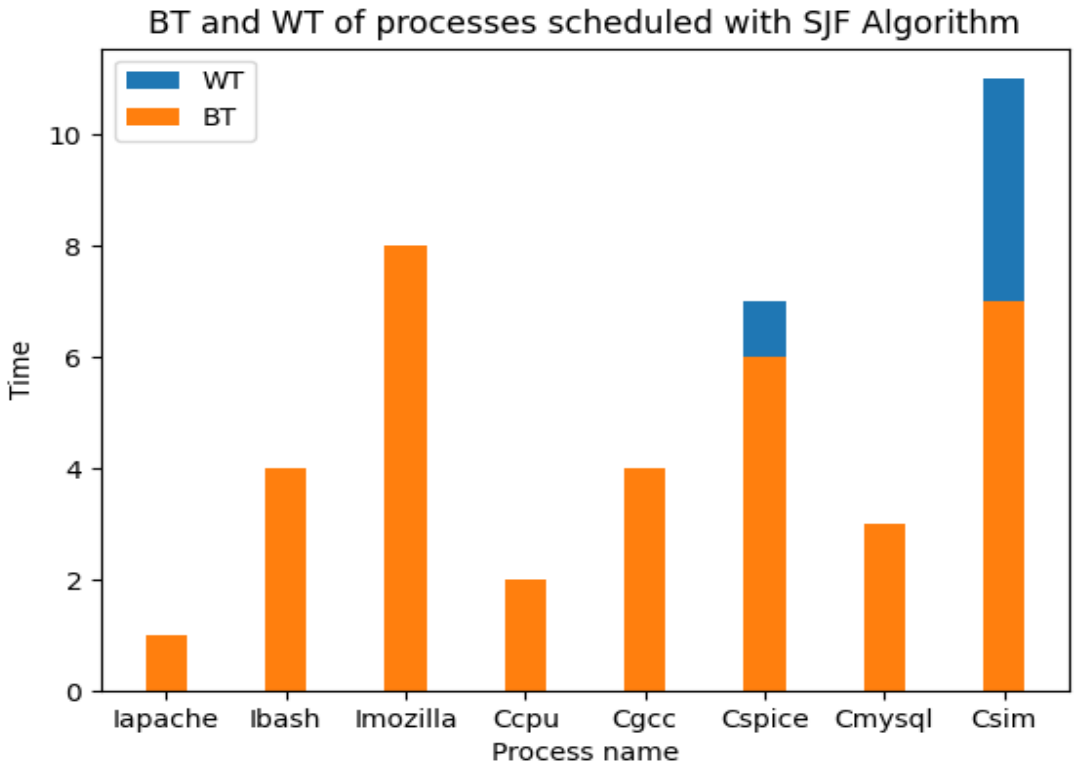
Process ID	Process Name	IO/CPU Bound	Arrival Time	Burst Time
0	Iapache	IO-BOund	0	1
1	Ibash	IO-BOund	1	4
2	Imozilla	IO-BOund	2	8
3	Ccpu	CPU-Bound	3	2
4	Cgcc	CPU-Bound	4	4
5	Cspice	CPU-Bound	5	6
6	Cmysql	CPU-Bound	6	3
7	Csim	CPU-Bound	7	7

Note: All the Analysis is done for the input value given above.It is assumed that all I/O are handled parallely.



Graph depicts the Gantt chart for the processes scheduled using shortest job first (SJF) algorithm.

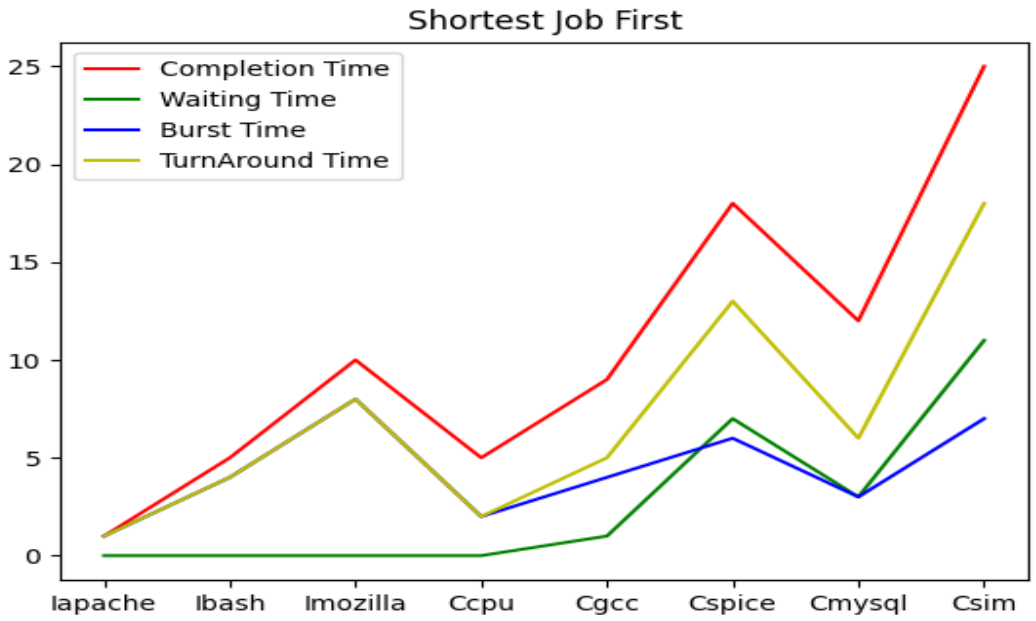
Red depicts I/O bound processes and blue depicts CPU bound processes. Each Blue slice depicts the share of CPU time taken by the respective process.



Bar graph depicts the burst time and waiting time of processes scheduled using the shortest job first (SJF) algorithm.

Claim: SJF **may** cause starvation to processes with higher burst time.

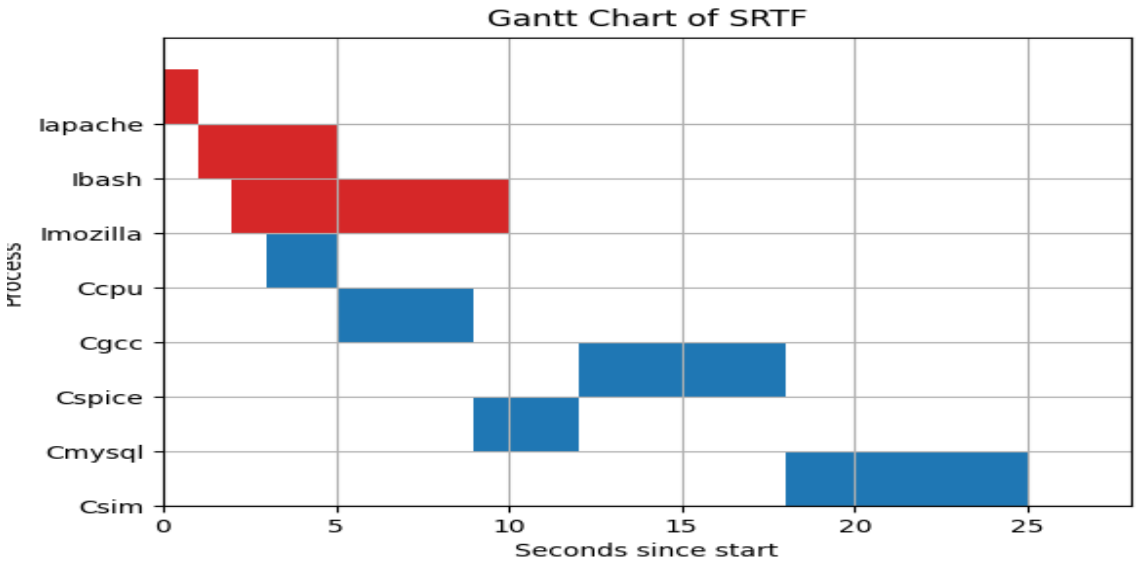
Process **Csim** has arrived last and has higher burst time than most of the processes and hence it has higher waiting time which supports our claim.



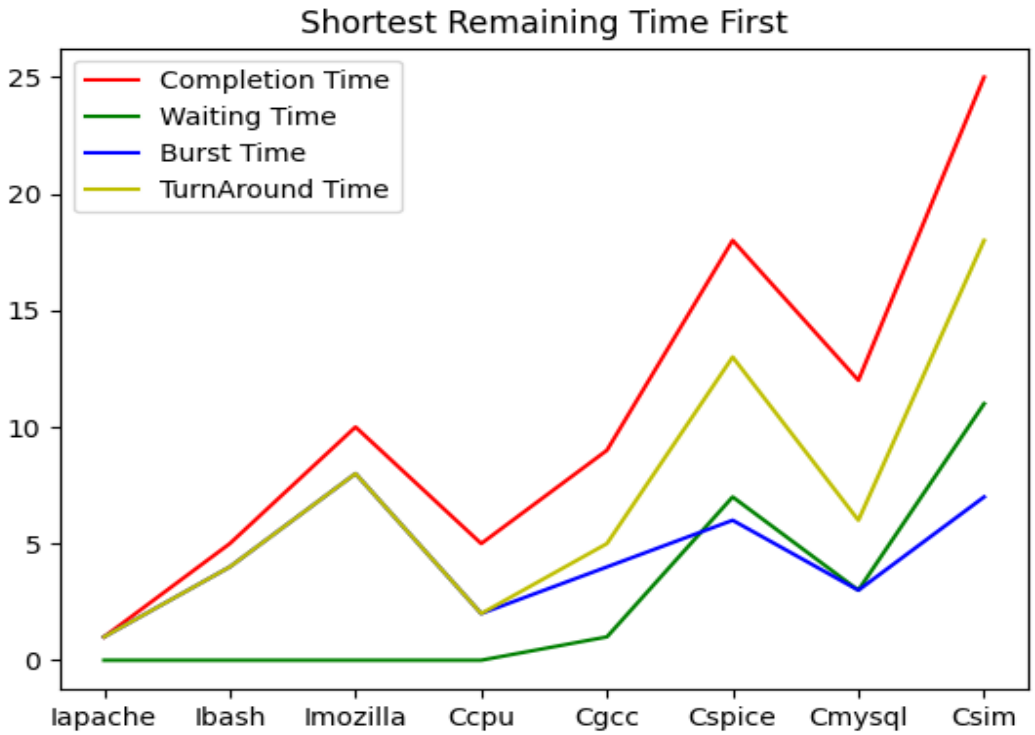
Graph depicts the burst time, waiting time, completion time and turnaround time of processes scheduled using the shortest job first (SJF) algorithm.

Processes Cspice and Csim have the highest burst times out of all CPU-bound processes. Hence from the graph it is evident that they have the highest waiting times.

Shortest Remaining Time First

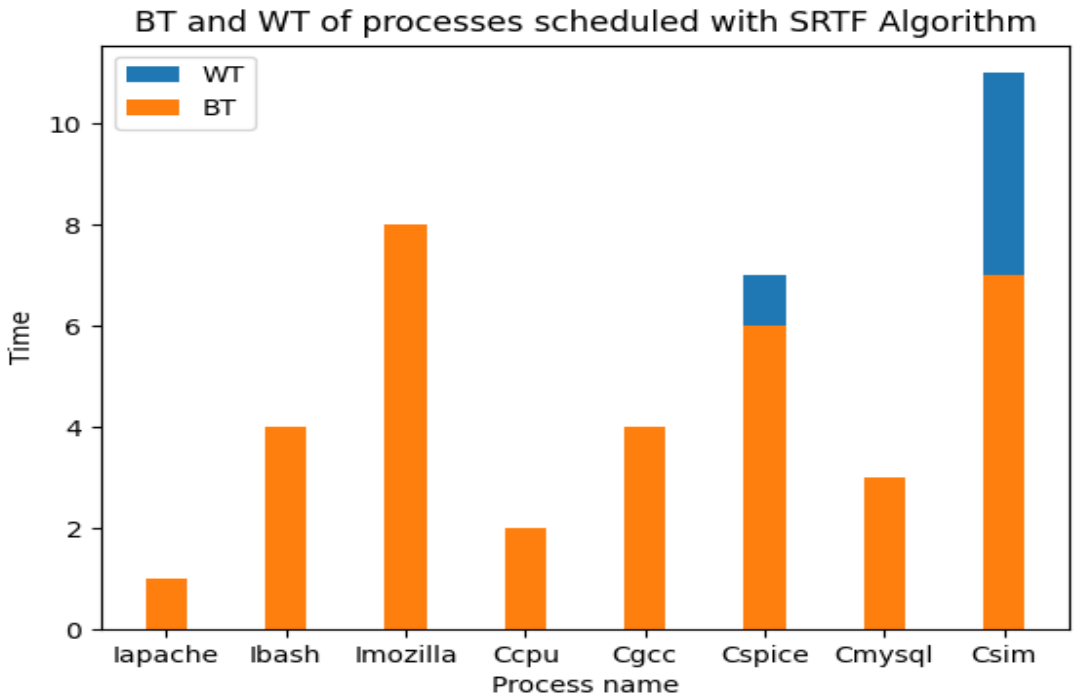


Graph depicts the Gantt chart for the processes scheduled using shortest remaining time first (SRTF) algorithm.



Graph depicts the burst time, waiting time, completion time and turnaround time of processes scheduled using the shortest remaining time first (SRTF) algorithm.

Process **Csim** has the highest burst time out of all the CPU-bound processes. Hence from the graph it is evident that it has the highest waiting time.

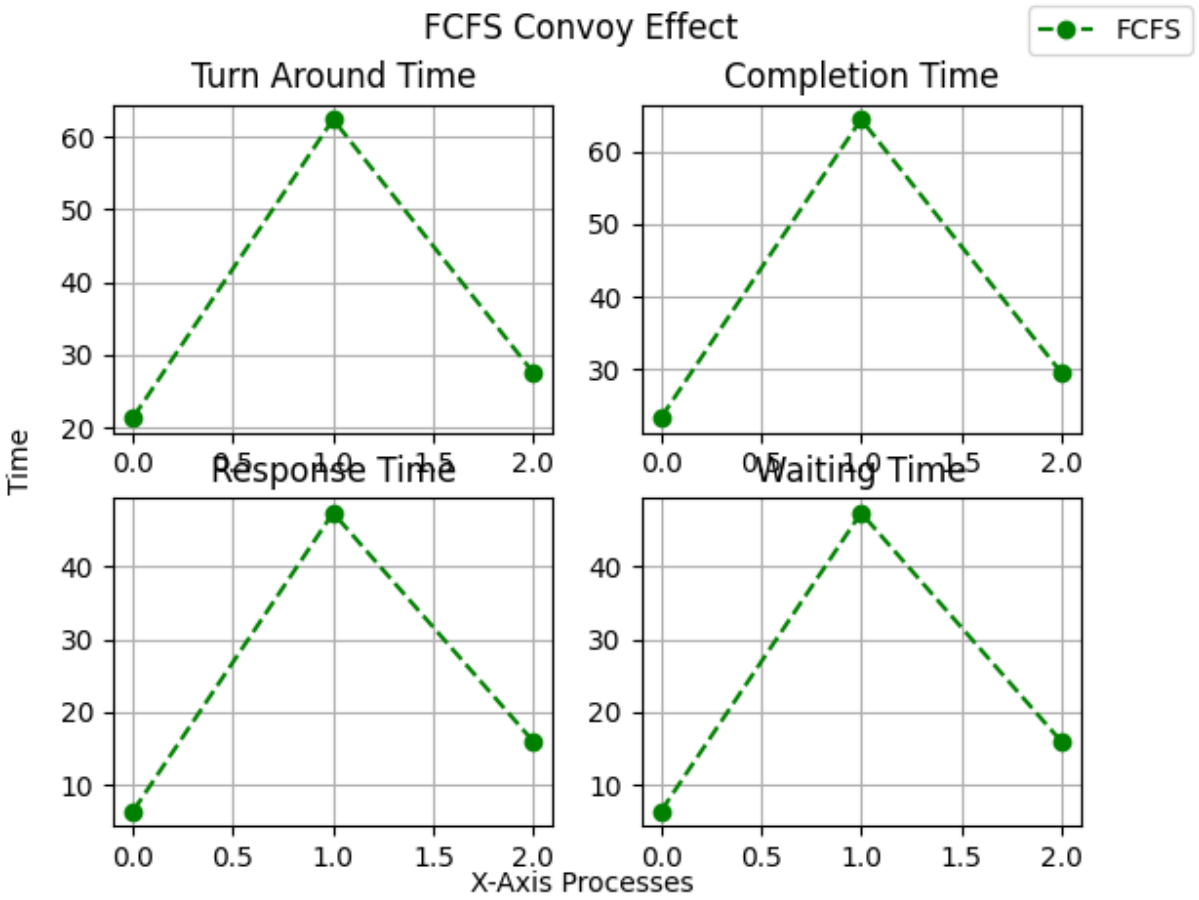


Bar graph depicts the burst time and waiting time of processes scheduled using the shortest remaining time first (SRTF) algorithm.

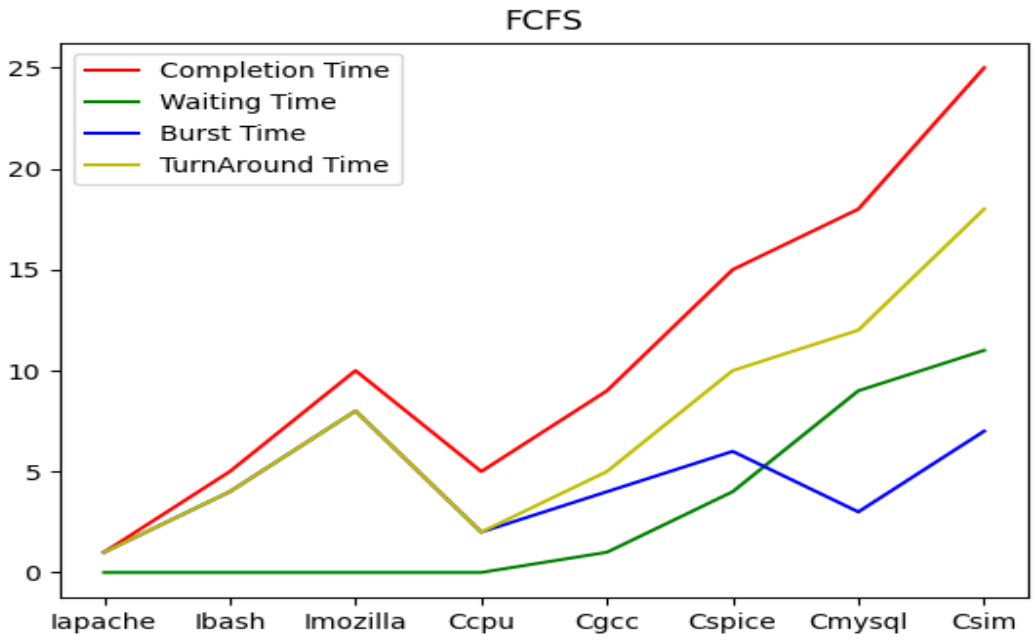
Claim: SRTF **may** cause starvation to processes with higher burst time.

Process **Csim** has the highest burst time out of all the processes and hence it has the highest waiting time which supports our claim.

First Come First Serve

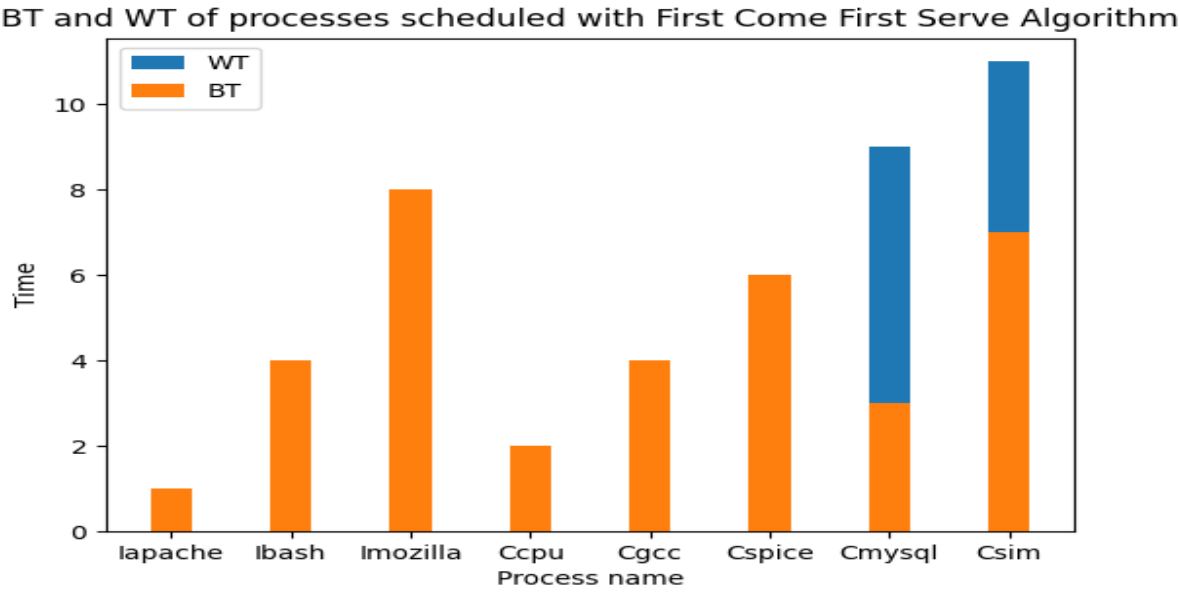


Observation: FCFS shows the **convoy effect** for the case where the arrival time of the larger processes are lower than the shorter processes as shown in the figure. The graph clearly shows that the average response time,waiting time, turn around time, and completion time of the test case-2 (X-axis : 1.0) are higher compared to the other test cases. The test case-1(X-axis : 0.0) is the case where the shorter processes arrive before the larger processes, so it has lower average rt,wt,tat,ct compared to other cases and the test case-3(X-axis: 2.0) is the case where the shorter process is followed by a larger process so the average values of this case are higher than case-1 and lower than case-2.



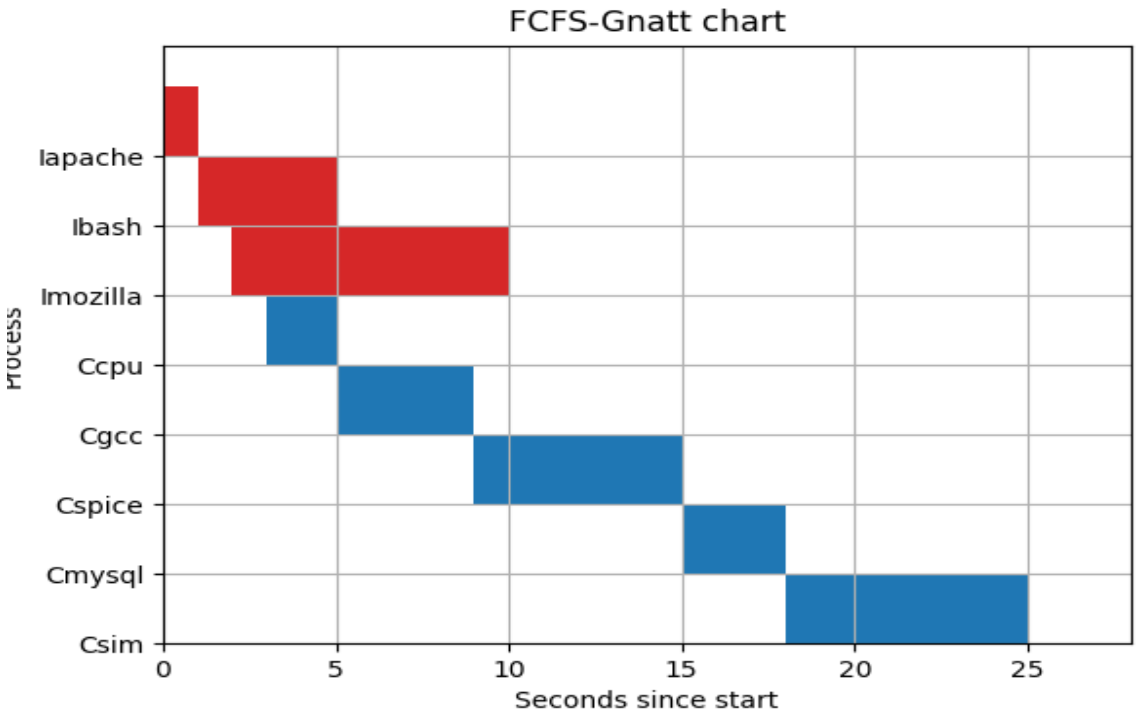
Graph depicts the burst time, waiting time, completion time and turnaround time of processes scheduled using the First Come First Serve (FCFS) algorithm.

Processes **Cmysql and Csim** have a higher waiting time(so is the completion and response time) compared to all the processes because the burst time of the previous processes is higher.



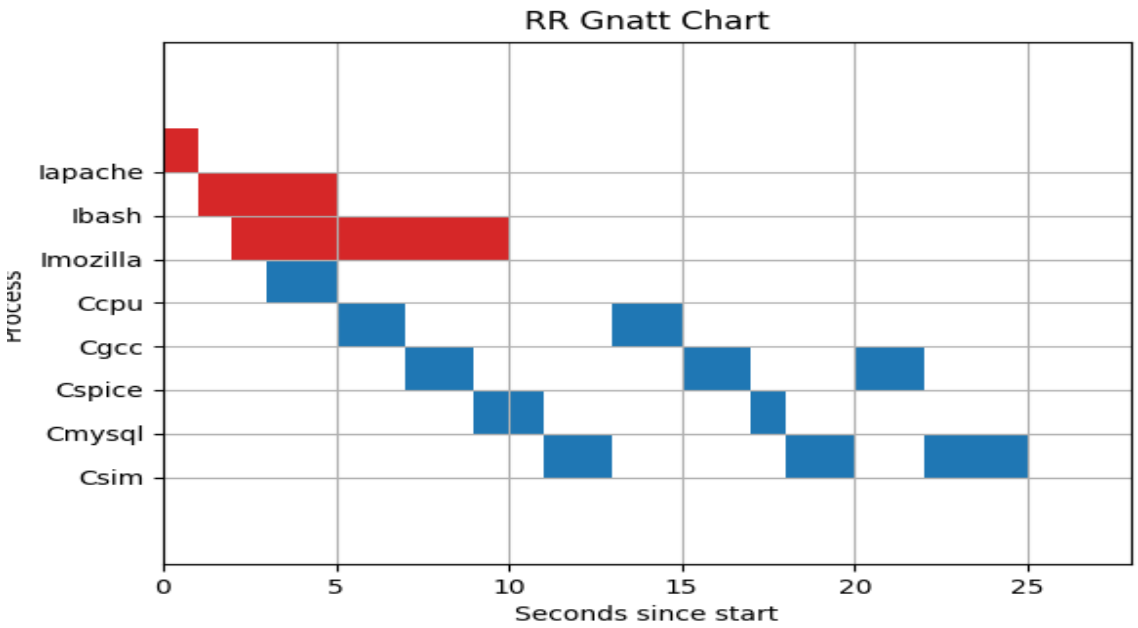
Bar graph depicts the burst time and waiting time of processes scheduled using the First Come First Serve (FCFS) algorithm.

Processes **Cmysql and Csim** have a higher waiting time as shown in the bar graph compared to all the processes because the burst time of the previous processes is higher.



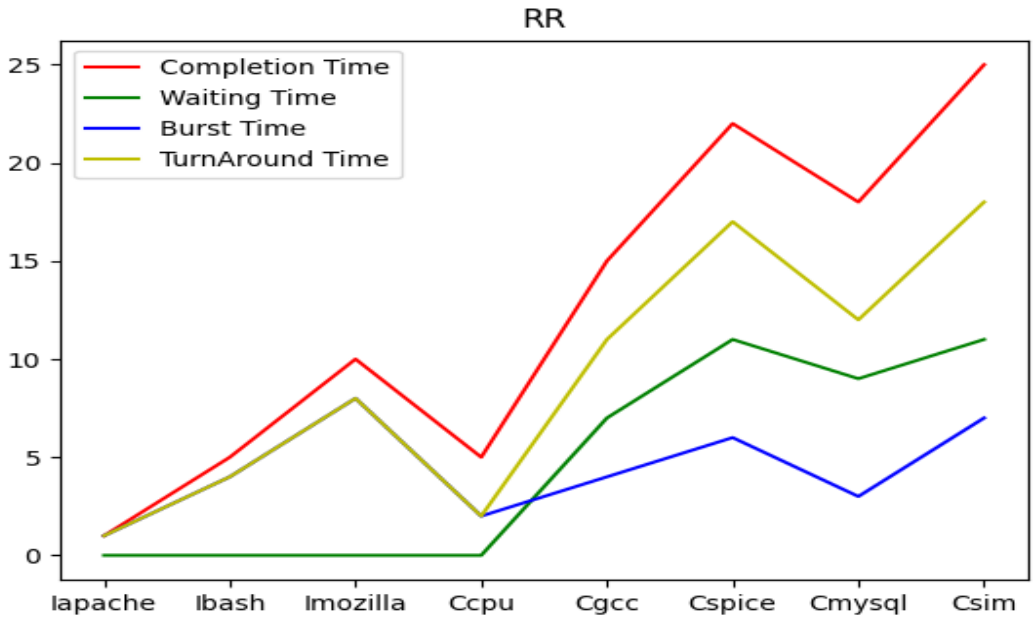
Graph depicts the Gantt chart for the processes scheduled using the First Come First Serve (FCFS) algorithm.

Round Robin



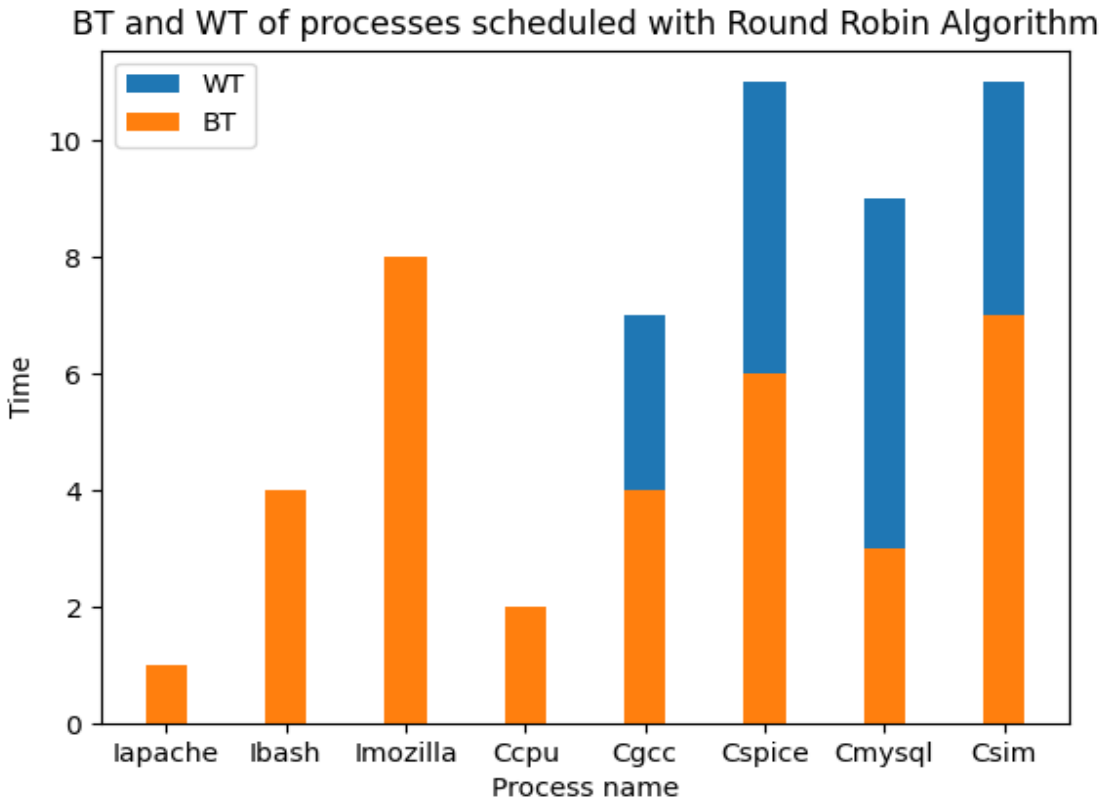
Graph depicts the Gantt chart for the processes scheduled using the Round Robin(RR) algorithm.

The graph clearly shows the time quantum of the processes is 2 and the processes are switched after each time quantum of 2 seconds.



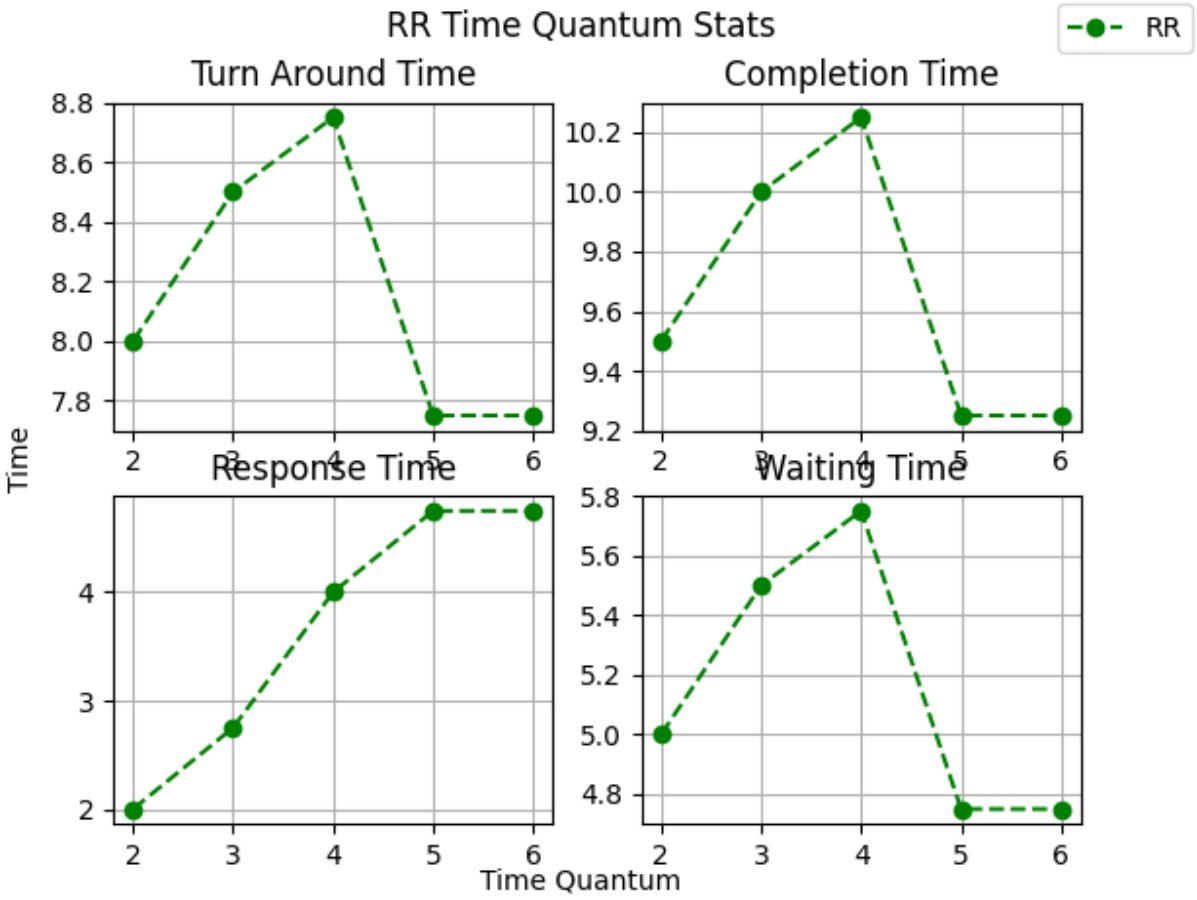
Graph depicts the burst time, waiting time, completion time and turnaround time of processes scheduled using the Round Robin(RR) algorithm.

Process **Csim** has the highest completion out of all the processes because it has the highest burst time but the time quantum is 2 seconds, so the context switches are high with many processes having larger burst time and the time quantum is lesser.



Bar graph depicts the burst time and waiting time of processes scheduled using the Round Robin(RR) algorithm.

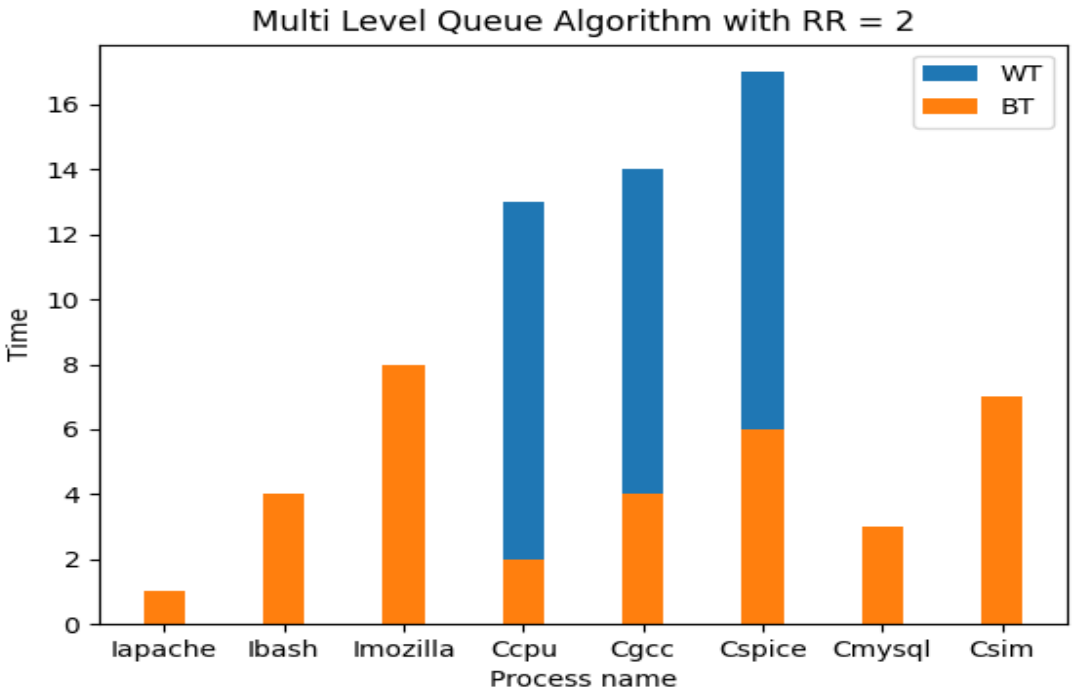
Almost all the processes have higher waiting time because the time quantum is less compared to their burst times.



Observation: The graph clearly depicts that the average waiting time, completion time, and turn around time for the same set of processes decreases with increase in the time slice time. The higher the time quantum, the lower are the average waiting time, completion time and turnaround time. We note that as the time slice increases the Round Robin behaves as FCFS and hence we can see this effect on the response time. The response time is getting increased with the increase in time-slice (Third Graph).

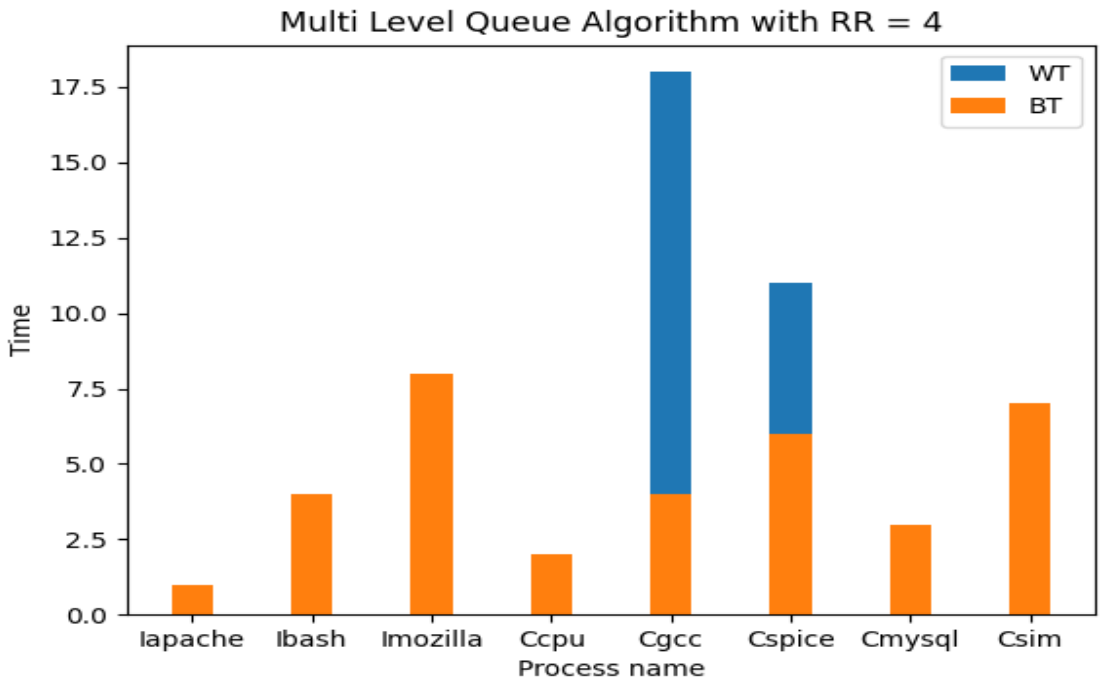
Multi Level Queue

1. System process queue uses priority scheduling algorithm
2. Interactive process queue uses Round Robin scheduling algorithm
3. Batch process queue uses FCFS
4. We have changed time quanta of Round Robin algorithm to generate different statistics and observe its effect on turnaround time (TAT), waiting time(WT), completion time (CT)
5. Processes in the above input are randomly assigned as system, interactive or batch

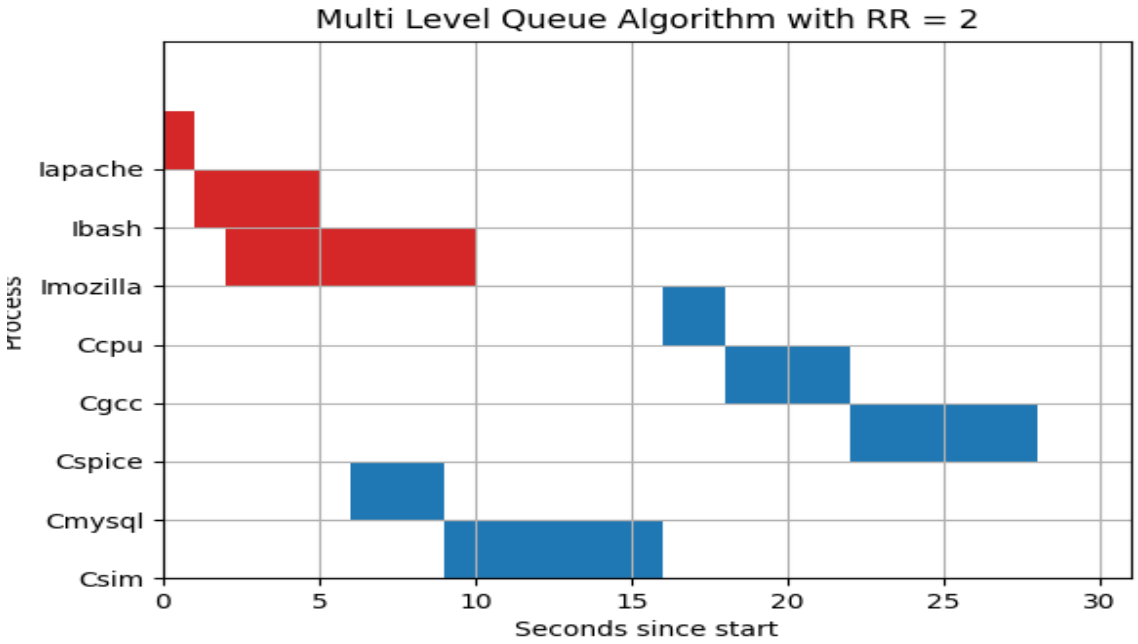


Graph of waiting time and burst time of processes when time quanta for Round Robin is 2.

Conclusion: Average waiting time in round robin is high. In the graph it can be seen that waiting time is very high compared to burst time for some processes. Hence, the below graph agrees with this theory

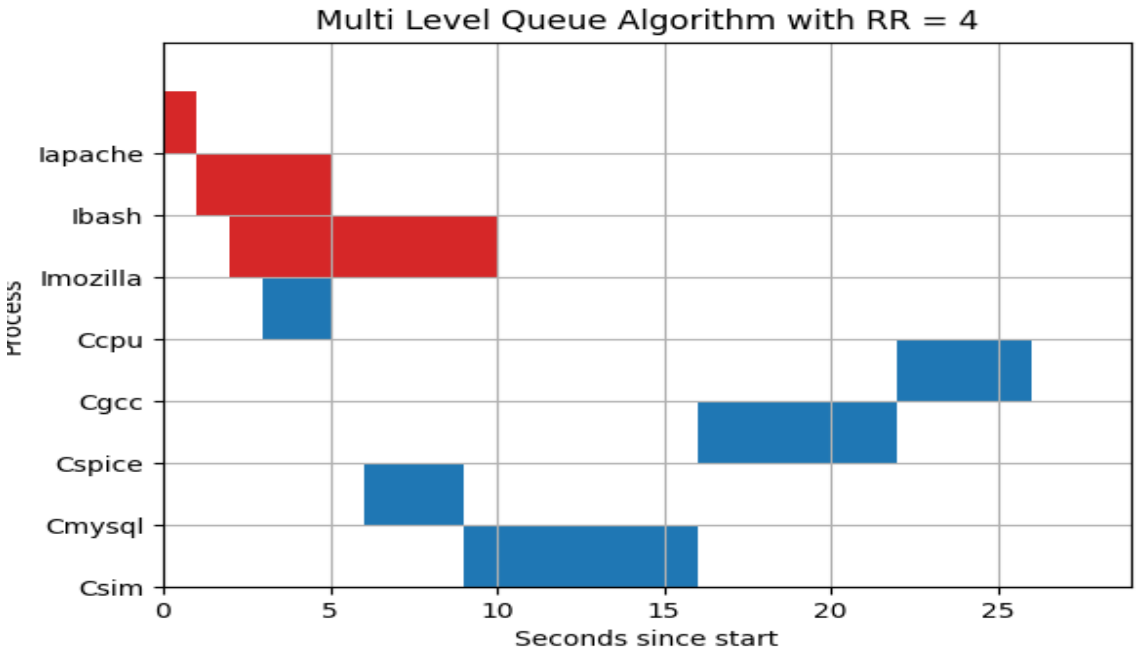


Graph of waiting time and burst time of processes when time quanta for Round Robin is 4.
(You can see how it differs from the above graph)

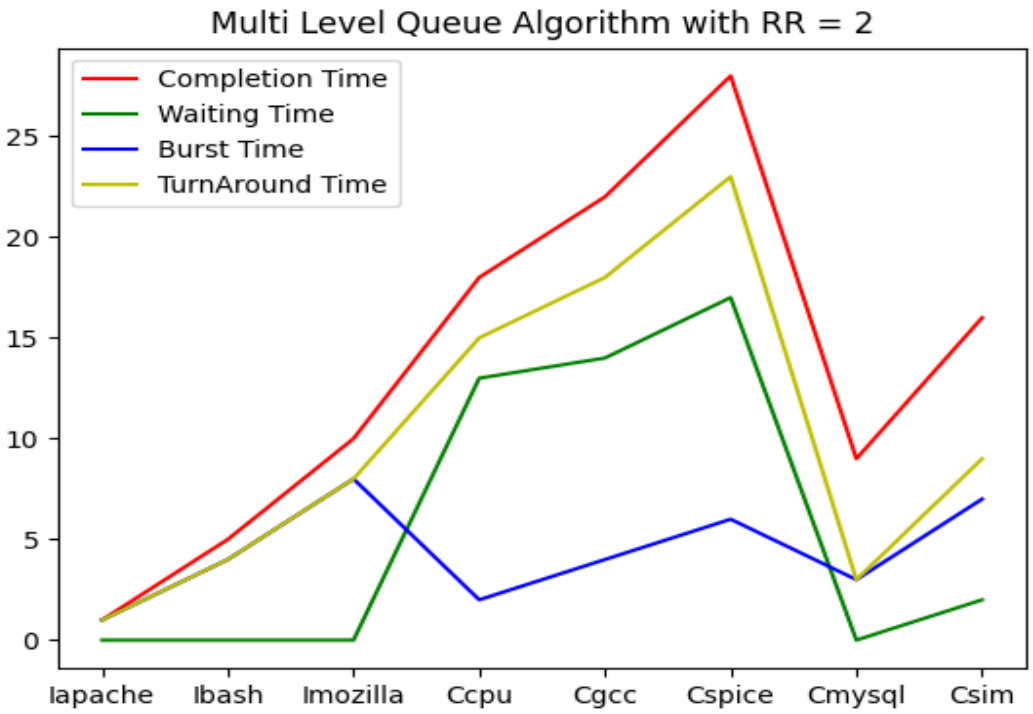


Gantt chart for CPU bursts when Round Robin quantum is set to 2.

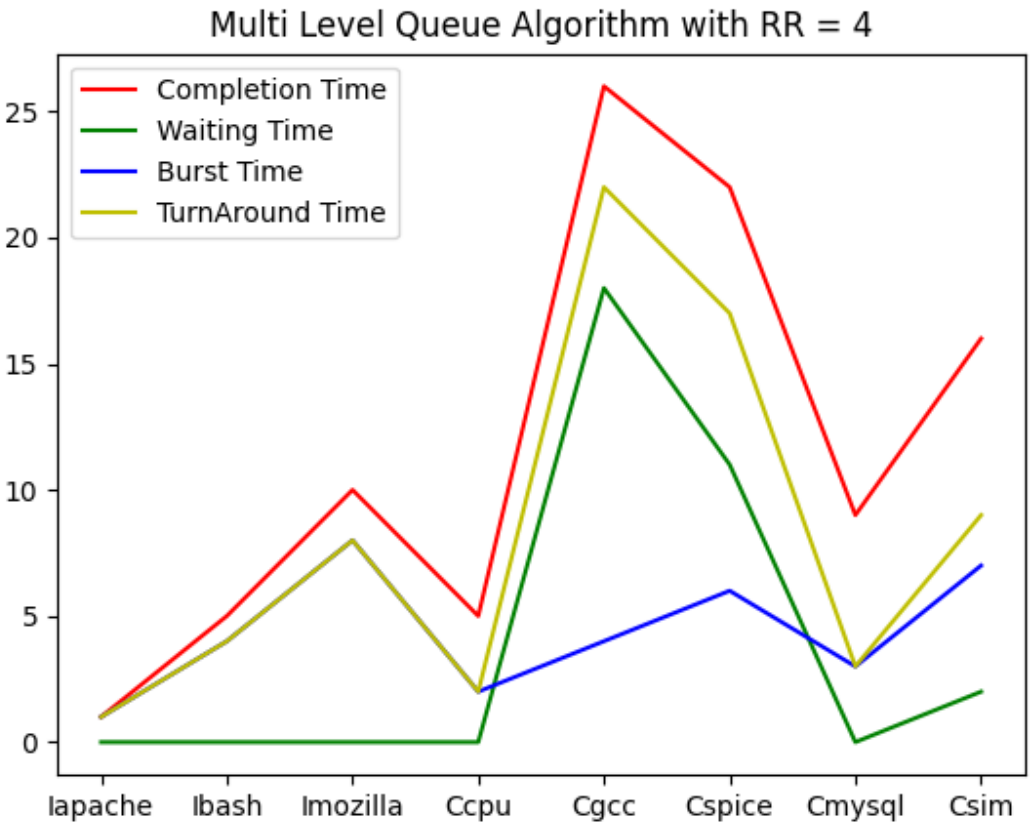
Conclusion: It tells us about the times when CPU is allocated to the process.



Gantt chart for the same set of processes but Round robin quantum is set to 4. (The differences are evident.)



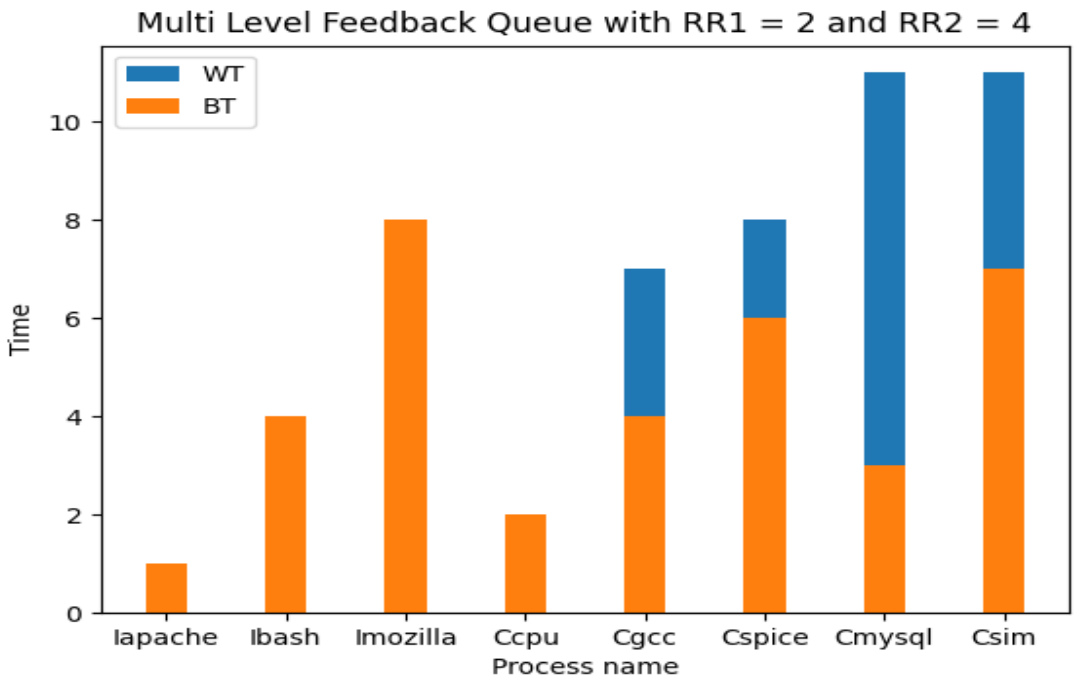
Graph shows all the scheduling metrics for Multi Level Queue with round robin quantum 2.



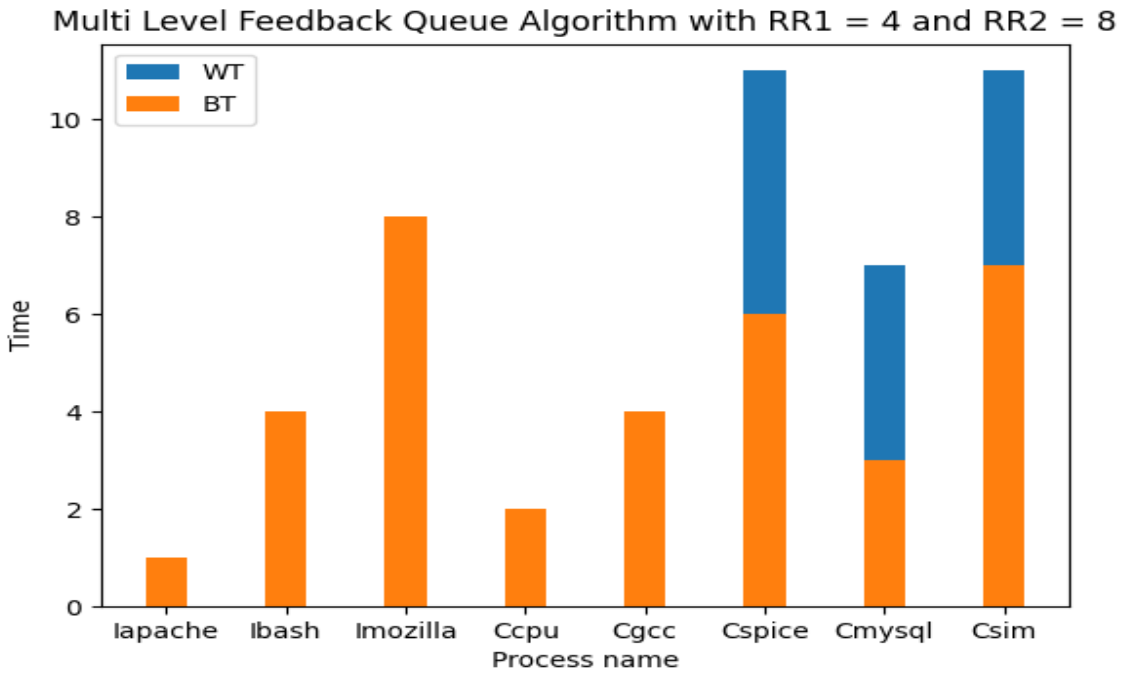
Graph shows all scheduling metrics when round robin quantum is 4. (These graphs can be compared to see the effect of increasing time quantum on the scheduling metrics)

Multilevel Feedback Queue

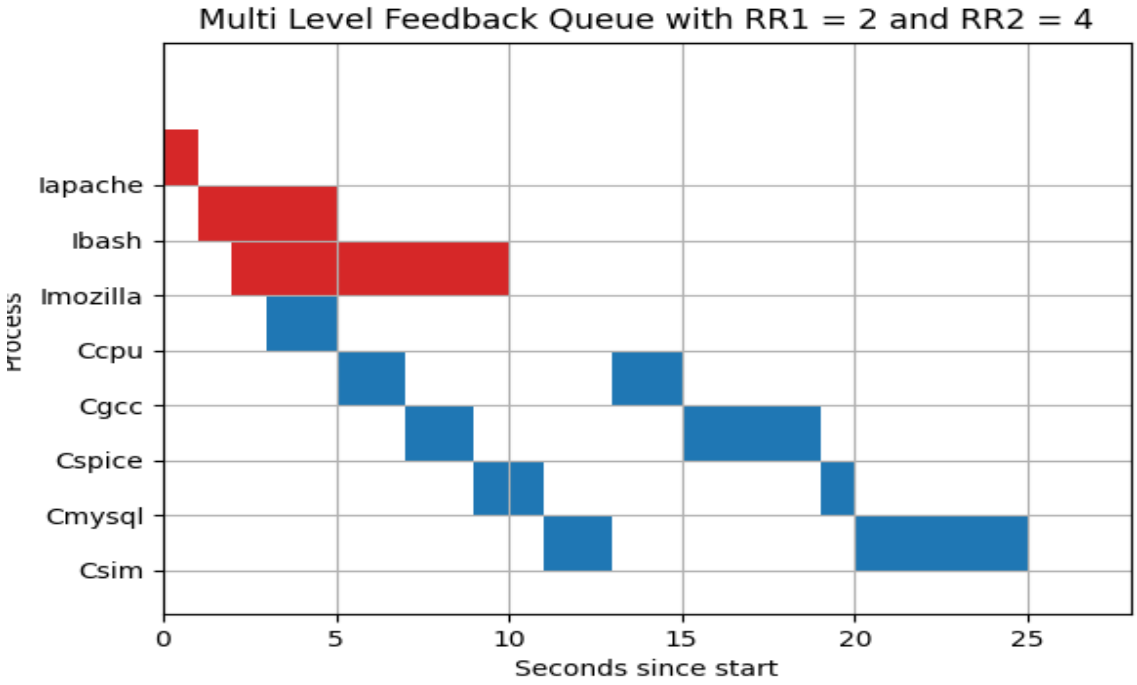
1. 3 queues are implemented for the Multi Level Feedback Queue.
2. Round Robin scheduling is used for the first 2 queues are FCFS (first come first serve) is used for the last queue.
3. We have changed time quanta of Round Robin algorithm to generate different statistics and observe its effect on turnaround time (TAT), waiting time(WT), completion time (CT)



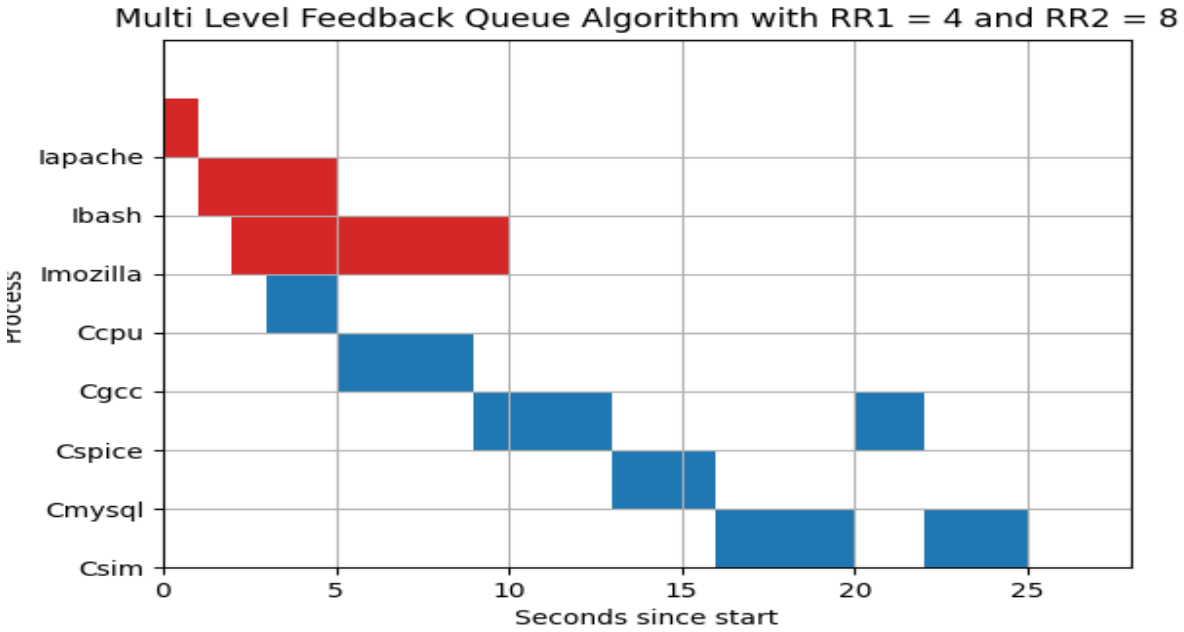
Graph shows the ratio of waiting times and burst times of processes when time quantum for 1st queue is 2 and the time quantum for second queue is 4.



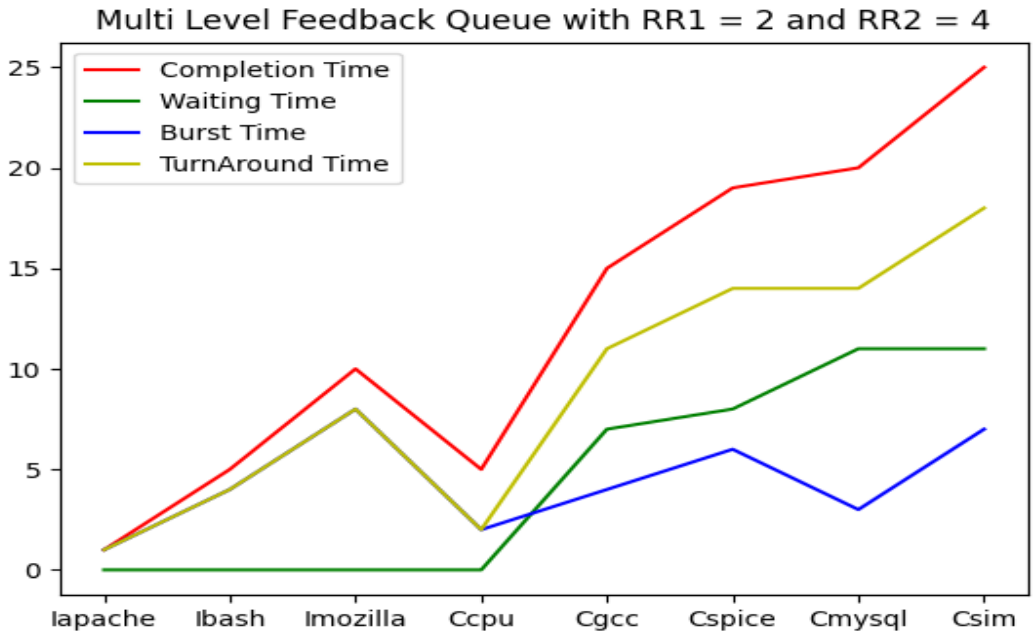
Graph generated when MultiLevel Feedback queue is run on the same set of processes with time quantum for 1st queue as 4 and time quantum for 2nd queue as 8.



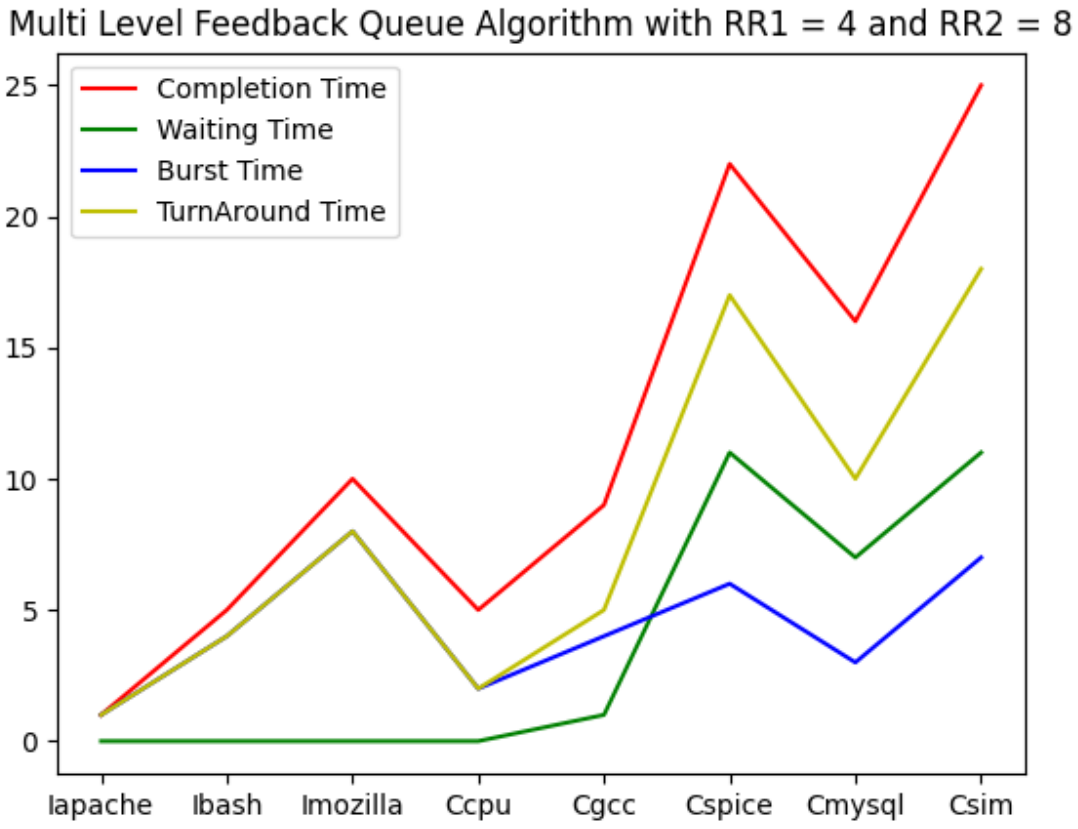
Gantt chart for CPU bursts when Round Robin quantum is set to 2 for first queue and 4 for second queue. (It tells us about the times when CPU is allocated to the process)



Gantt chart for the same set of processes but Round robin quantum is set to 4 for the first queue and 8 for the second queue.(The differences are evident.)



Graph shows all the scheduling metrics for MultiLevel Feedback Queue when time quantum for 1st queue is 2 and time quantum for 2nd queue is 4.

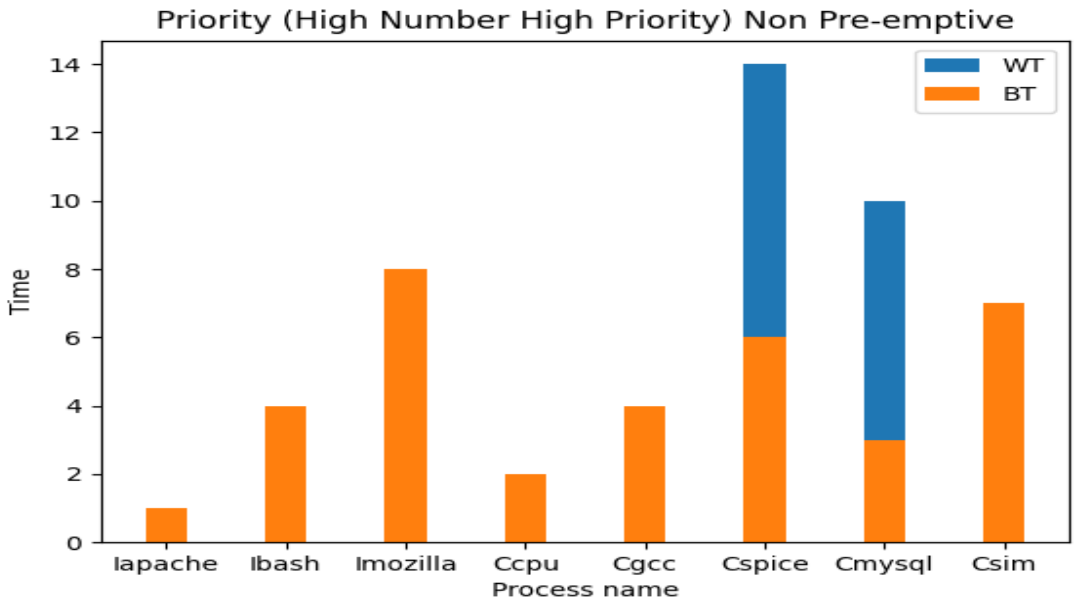


Graph again shows all the scheduling metrics for Multi Level Feedback Queue on the same set of processes. Time quantum for the 1st queue is 4 and the time quantum for the 2nd queue is 8.

Priority Scheduling (Non Preemptive & Preemptive):

Process Name	IO/CPU Bound	Priority	Arrival Time	Burst Time
Iapache	IO-BOund	8	0	1
Ibash	IO-BOund	7	1	4
Imozilla	IO-BOund	7	2	8
Ccpu	CPU-Bound	3	1	2
Cgcc	CPU-Bound	4	2	4
Cspice	CPU-Bound	1	3	6
Cmysql	CPU-Bound	7	4	3
Csim	CPU-Bound	8	5	7

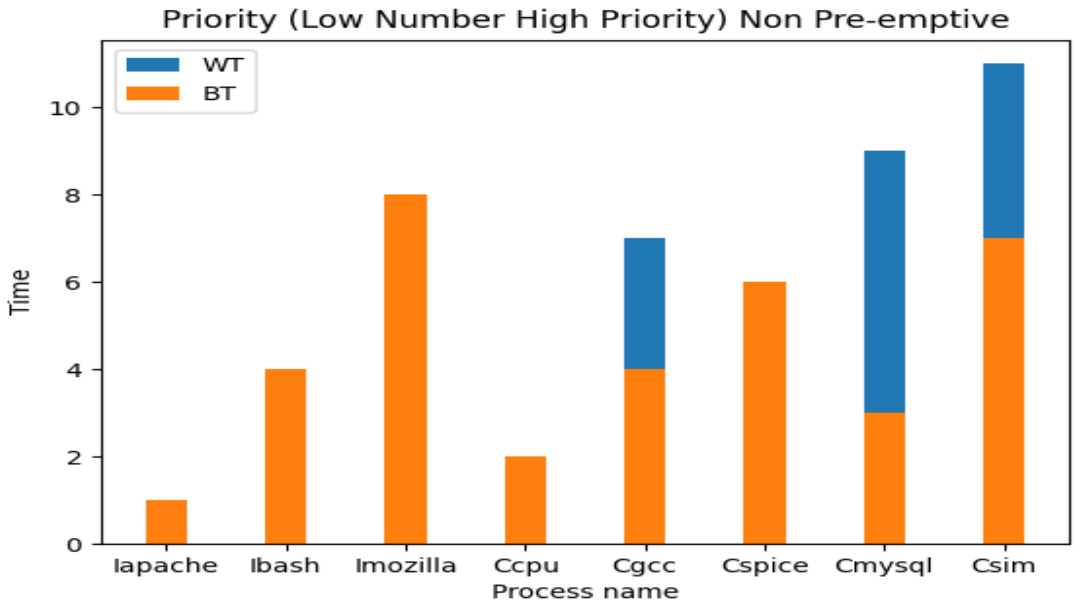
Note: All the Analysis for Priority Algorithm is done for this input (Arrival Time and Burst Time is same as other algorithms, but priority is added)



Comparison for waiting & burst time and among processes (High Number means high Priority) Non-Preemptive.

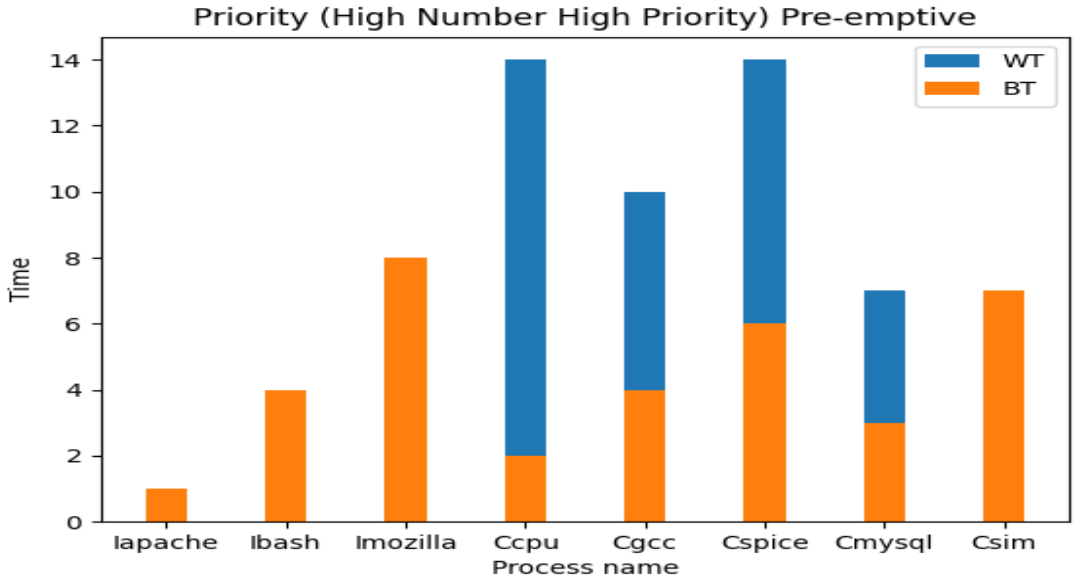
Conclusion: Since Cspice had the lowest number(lowest priority) its waiting time is large, CCpu also had a lower number but it has very less waiting time as it is Non-Preemptive algo and Ccpu was one of the first processes to arrive for scheduling.

.



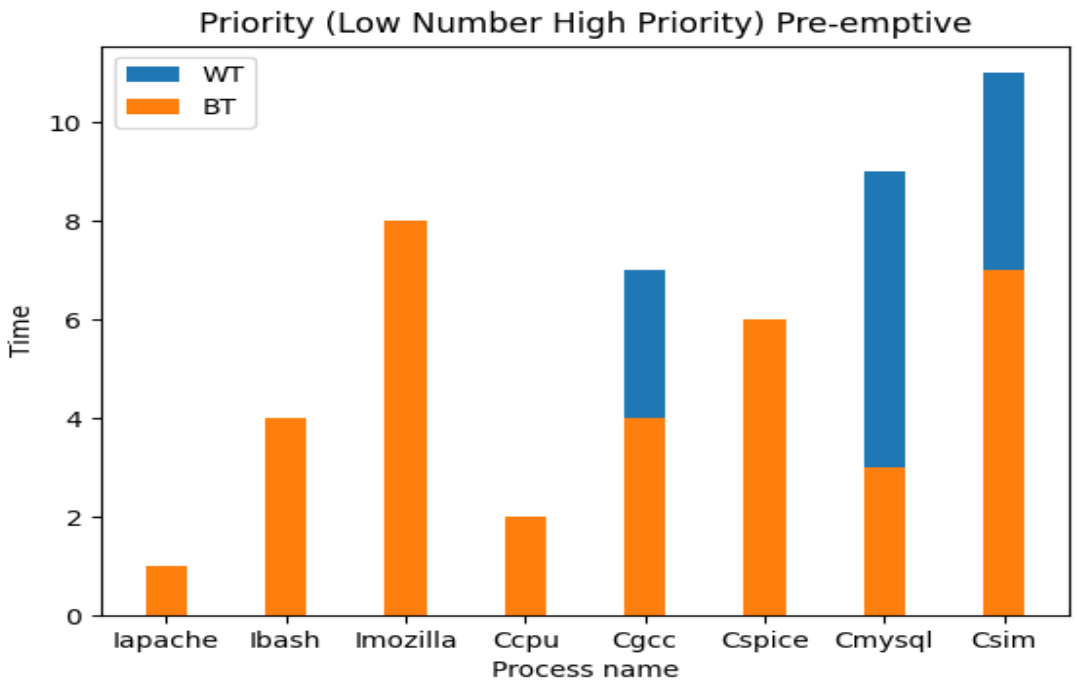
Comparison for waiting & burst time and among processes (Low Number means high Priority) Non-Preemptive.

Conclusion: Since Cspice had the lowest number(Highest priority) its waiting time is 0, But CCpu has a relatively higher number but still it has 0 waiting time as it is Non-Preemptive algo and Ccpu was one of the first processes to arrive for scheduling.



Comparison for waiting & burst time and among processes
(High Number means high Priority)Preemptive.

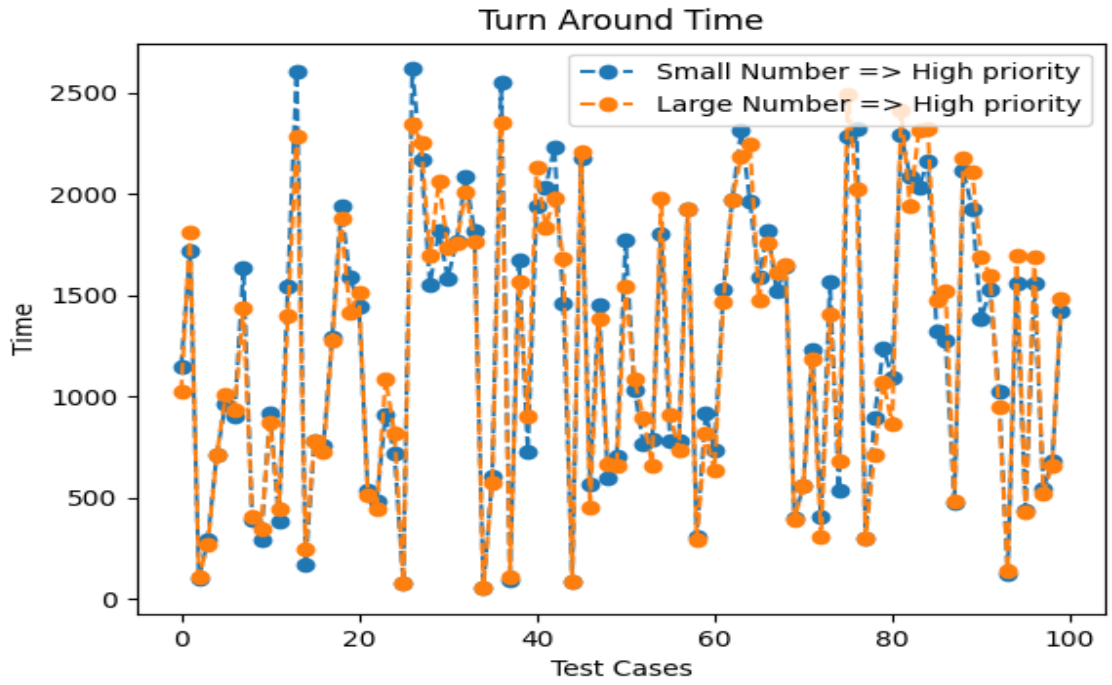
Conclusion: Now a lot of issues are resolved, those processes who had high priority (High number) are given preference over others and the waiting time is accordingly obtained.



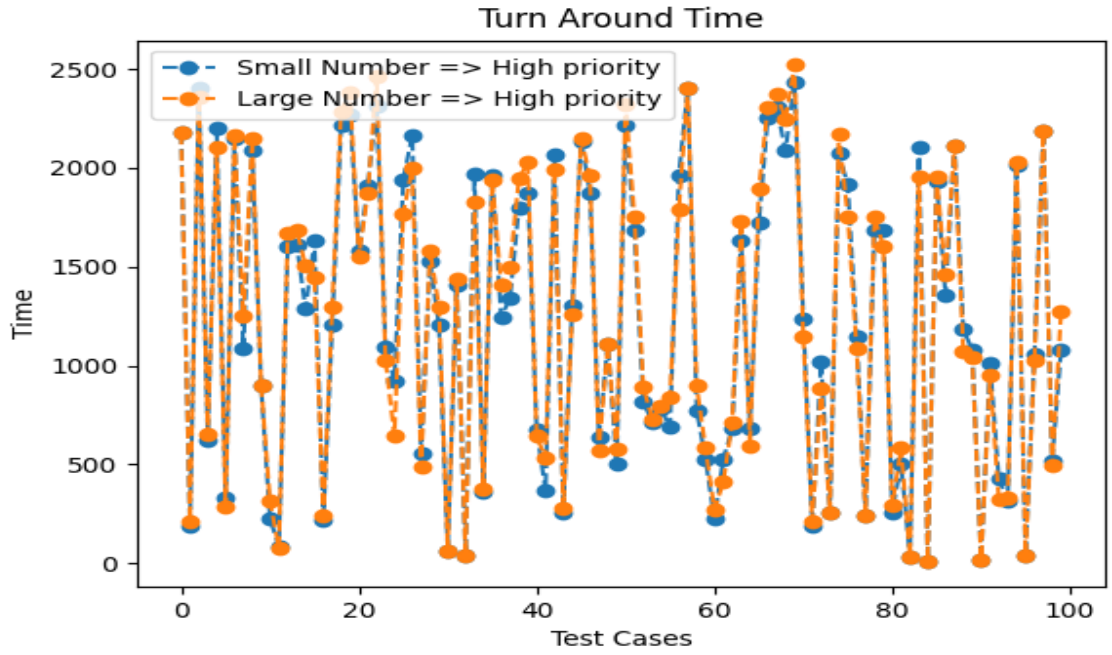
Comparison for waiting & burst time and among processes
(Low Number means high Priority)Preemptive.

Conclusion: Those processes who had high priority (Low number) are given preference over others and the waiting time is accordingly obtained.

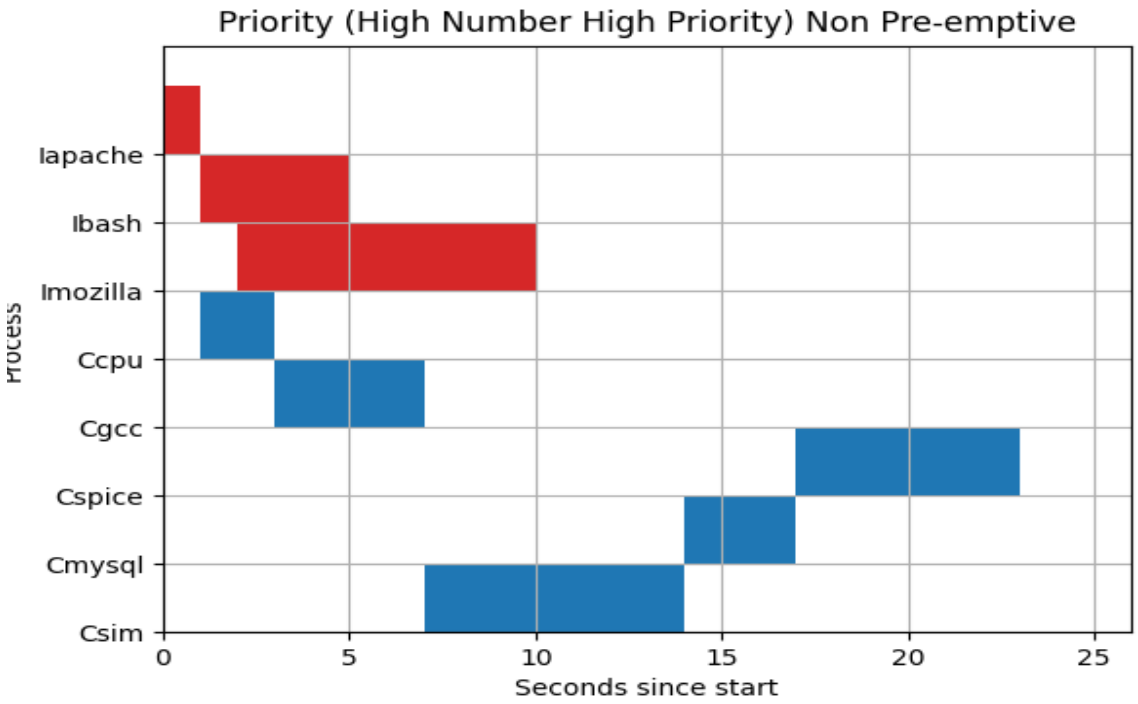
Can we come to a conclusion that scheduling processes with low numbers as high priority gives better performance(i.e. Low average Turnaround Time or Waiting Time) ? or vice versa.
Answer: No.
Proof: Consider the following Image for Turn Around time and Waiting TIme. Blue Dots represents TAT or WT of corresponding test cases which had small number high priority and Orange Dots represents TAT(or WT) of corresponding test cases which had large number as high priority as we can see Number of times Blue dots are above Orange dots remains the same and nullifies each other's effect, this is evident from the set of 100 random test cases also, but if we further extended the situation for very large amount of test cases this would have been more clear moreover, it was verified computationally as well and the difference in corresponding sum of all TAT(or WT) was in the range (-0.0009 to 0.000011) summed over 1000 random test cases.



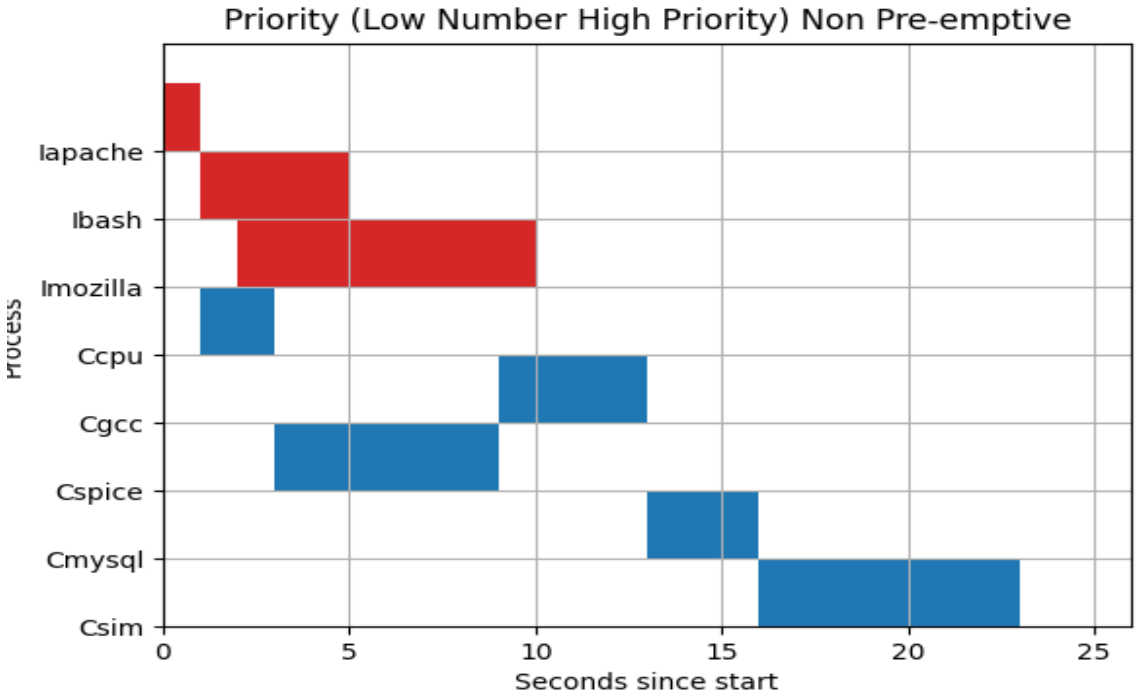
Comparison for TAT for 100 test cases, to test what changes occur by changing Hyper parameters.(Non-Preemptive version)



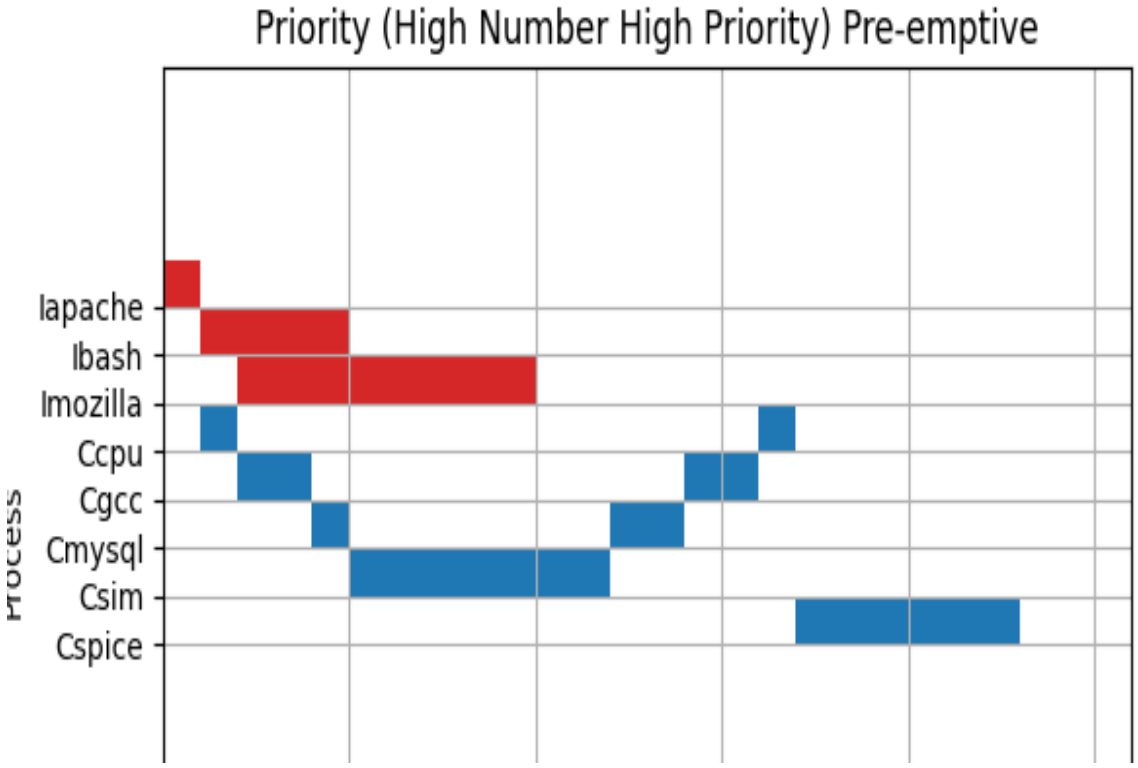
Comparison for TAT for 100 test cases, to test what changes occur by changing Hyper parameters.(Preemptive version)



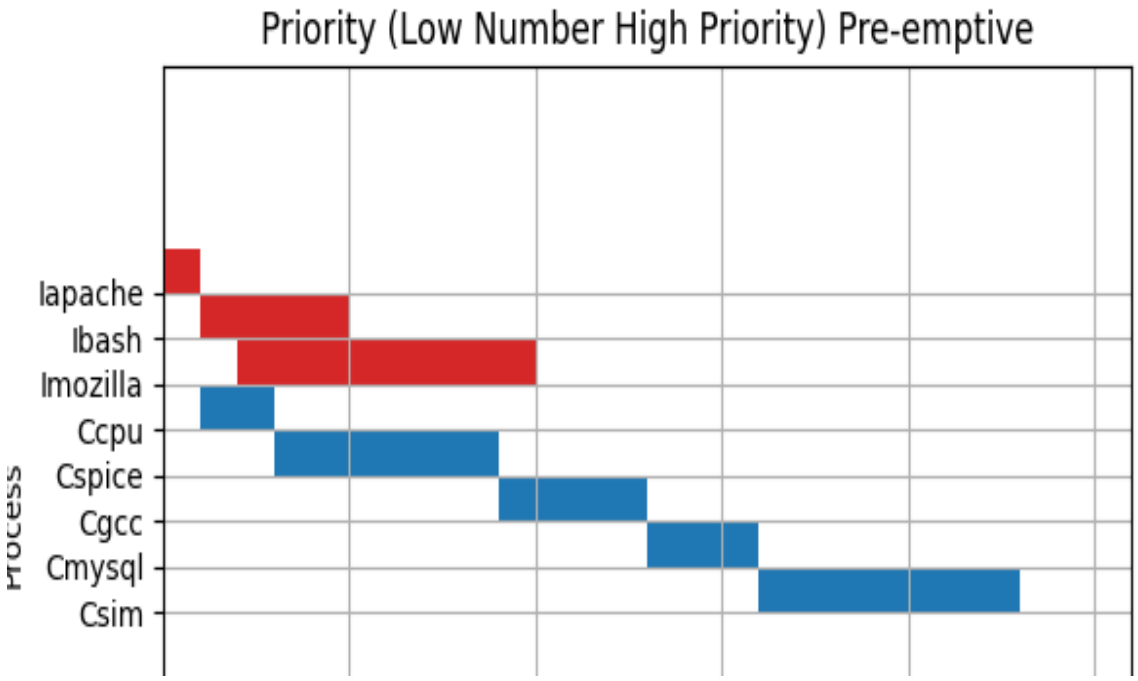
Gantt Chart for Non-Pre-emptive Priority Scheduling with metric High Number as High Priority.



Gantt Chart for Non-Pre-emptive Priority Scheduling with metric Low Number as High Priority.

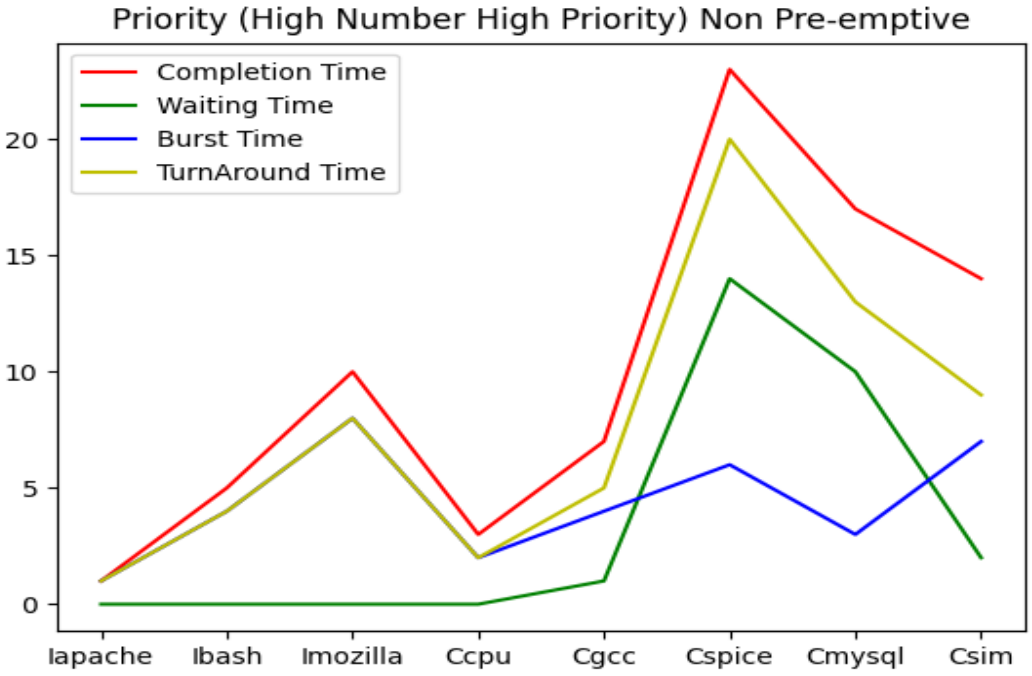


Gantt Chart for Preemptive Priority Scheduling with metric High Number as High Priority.



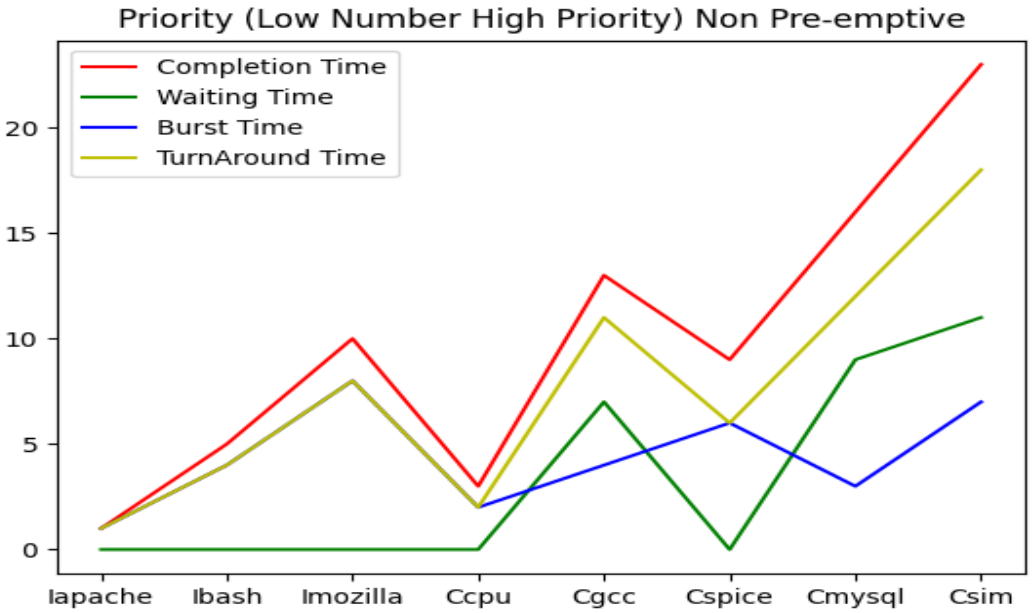
Gantt Chart for Preemptive Priority Scheduling with metric Low Number as High Priority.

Conclusion:There is a share of CPU time based on priority of process.



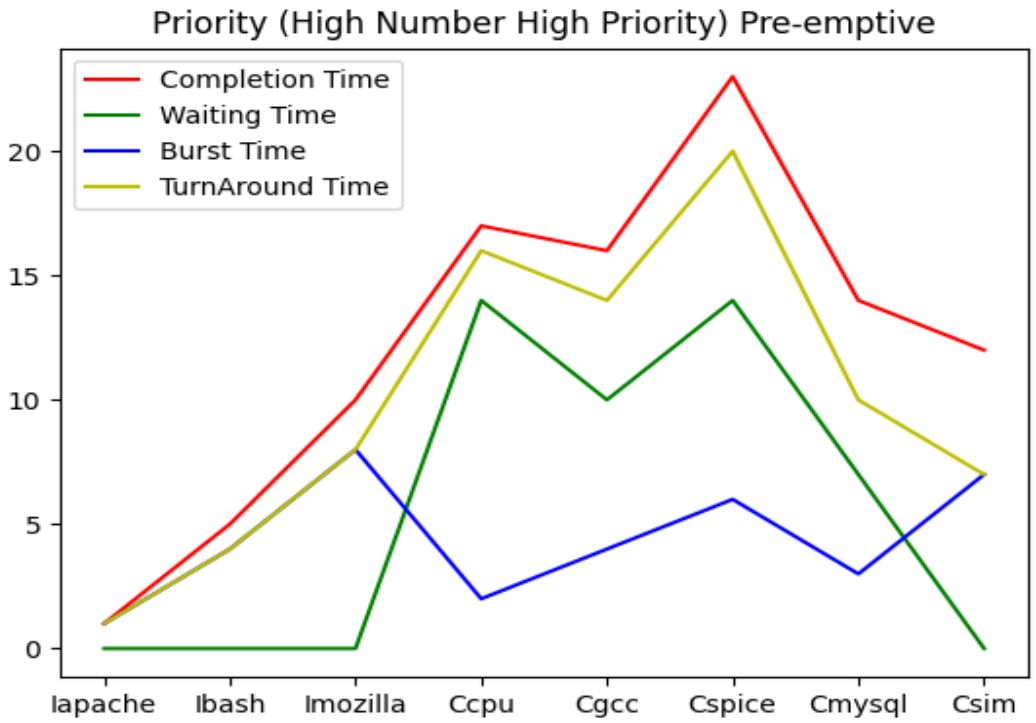
Line Graph for Non-Preemptive Priority Scheduling with metric High Number as High Priority.

Conclusion: Since Cspice had the lowest number(lowest priority) its waiting time is large, CCpu also had a lower number but it has very less waiting time as it is Non-Preemptive algo and Ccpu was one of the first processes to arrive for scheduling.



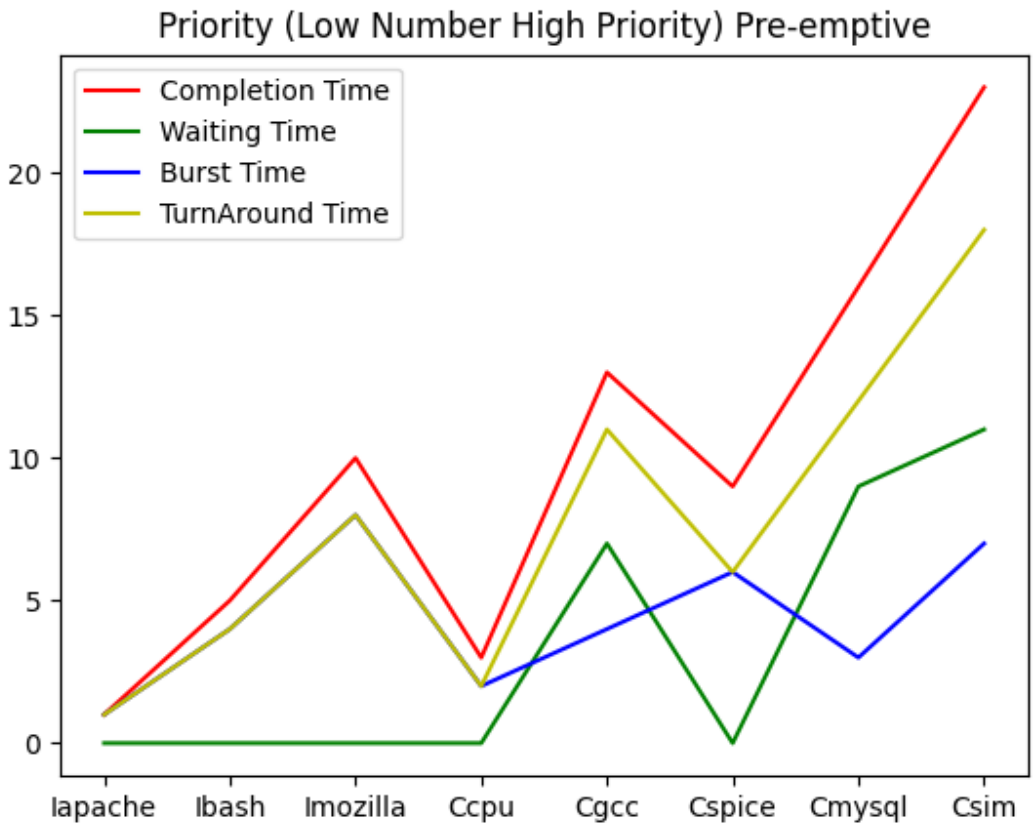
Line Graph for Non-Preemptive Priority Scheduling with metric Low Number as High Priority.

Conclusion: Since Cspice had the lowest number(Highest priority) its waiting time is 0, But CCpu has a relatively higher number but still it has 0 waiting time as it is Non-Preemptive algo and Ccpu was one of the first processes to arrive for scheduling.



Line Graph for Preemptive Priority Scheduling with metric High Number as High Priority.

Conclusion: Now a lot of issues are resolved, those processes who had high priority (High number) are given preference over others and the waiting time is accordingly obtained and since it is a preemptive version whenever a process with higher priority comes then lower priority process is preempted.

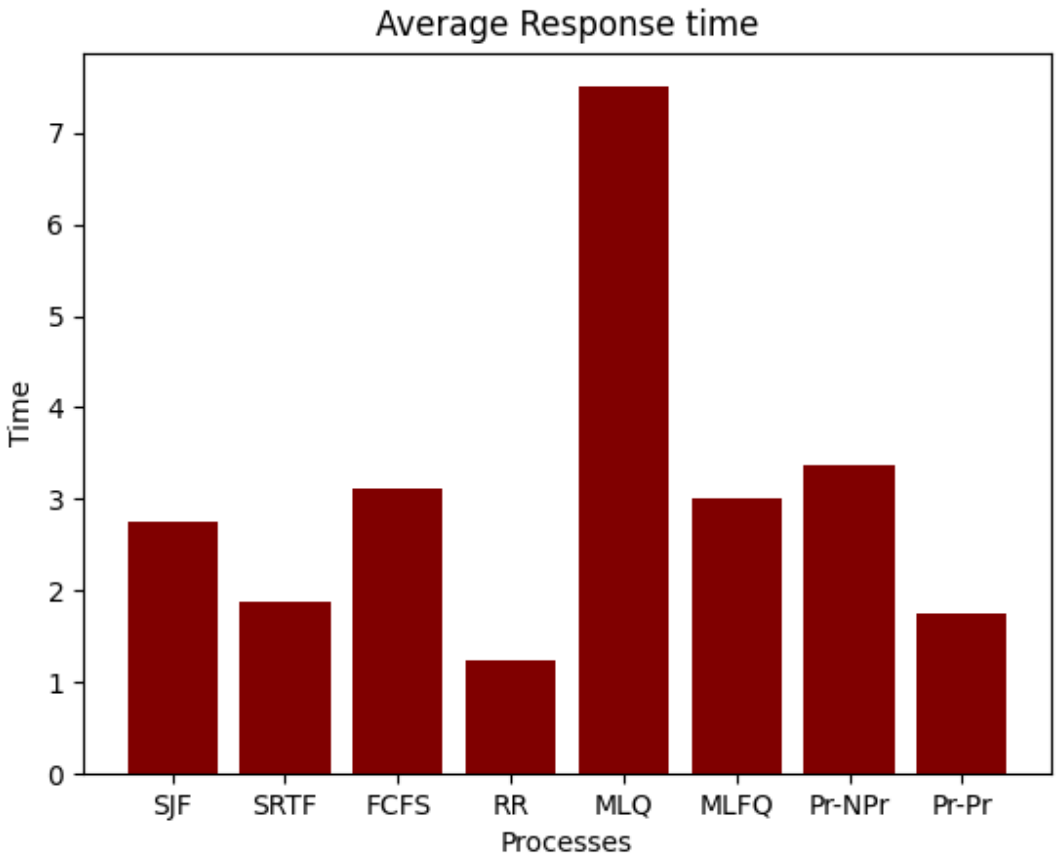


Line Graph for Preemptive Priority Scheduling with metric Low Number as High Priority.

Conclusion: Those processes who had high priority (Low number) are given preference over others and the waiting time is accordingly obtained.

Inter-Process Comparison

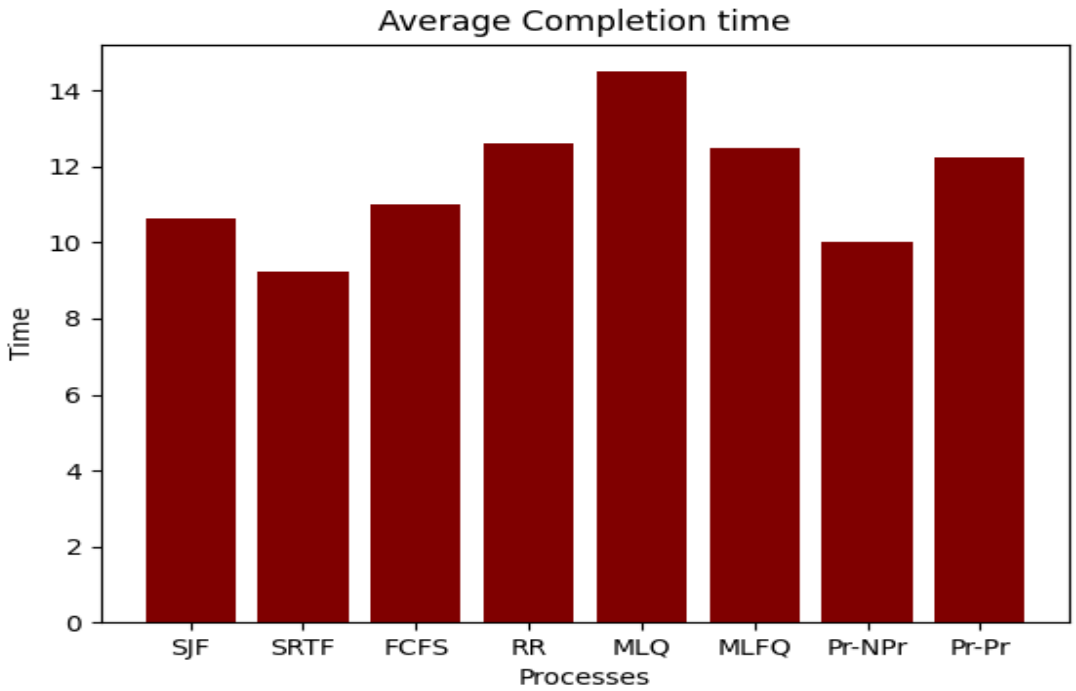
Note: Input for SRTF has slightly changed in order to show the difference between SJF and SRTF scheduling algorithm.



Bar Graph Showing Comparison between Response time Obtained using various Process Scheduling algorithms.

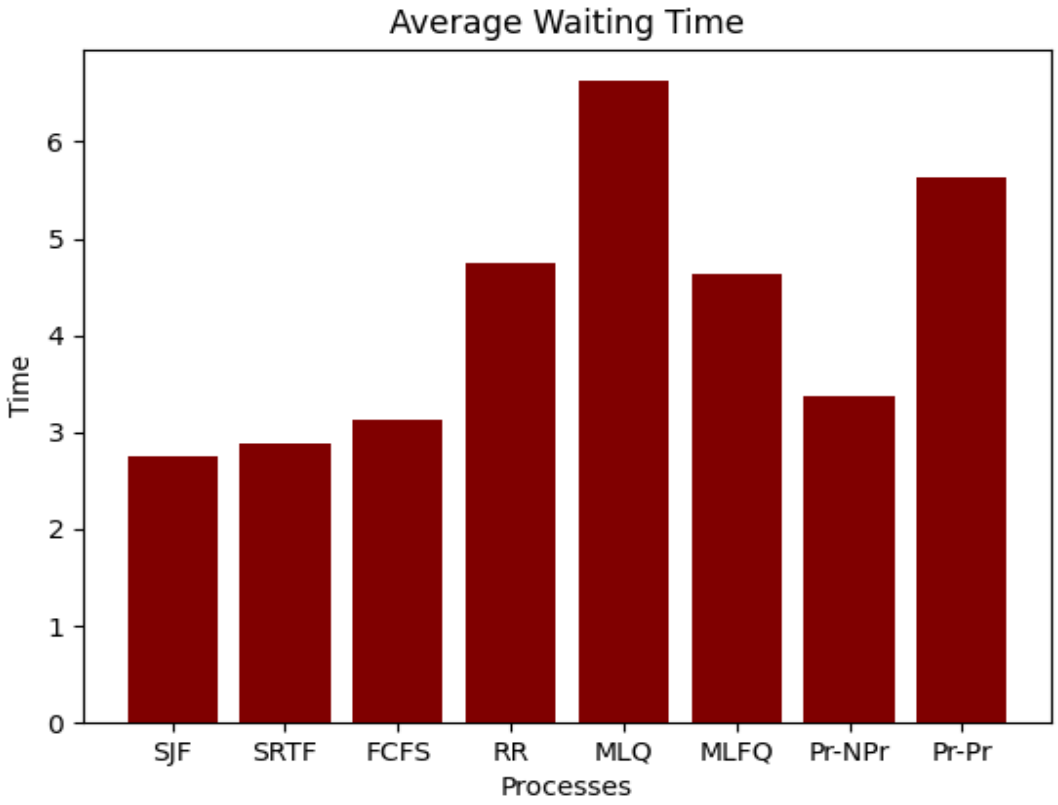
1. SJF and SRTF, even though they are theoretically the best algorithms, they will starve processes with higher bursts and hence they have higher response times than Round Robin but they are next best to Round Robin. SRTF being the preemptive version will have better response time as preemption is allowed whereas in SJF, preemption is not allowed, and hence it will not be that responsive.

2. In RR, the average response time is less compared to all other algorithms because it allocates CPU to all the processes without any starvation i.e all the processes get fair allocation of CPU.
3. In Multi Level Queue (MLQ), processes in lower queues get CPU time only after the upper queues are empty. Hence, if processes keep arriving in the upper queue then processes in lower queues will be starved for CPU time. This explains the high average response time in the above graph.
4. In the MultiLevel Feedback Queue (MLFQ), processes are first kept in the upper queue. If they are still not finished with their execution, then they are moved in lower queues. Thus, processes get their first CPU time fairly earlier compared to Multi Level Queue. Hence Average Response Time is much better compared to Multi Level Queue as seen in the above graph
5. Preemptive priority as well as Non-Preemptive Priority has very less waiting time, but that may not be the case always, it can happen that a process which causes large average response time for other processes has high priority and due to it the graph could have been entirely different,although Priority scheduling algorithm is not resigned to give less response time, for that we have round robin. And we can also verify that it is indeed the case and round robin has given very less response time.



Bar Graph Showing Comparison of Average Completion Time Obtained using various Process Scheduling algorithms.

1. As SJF & SRTF are theoretically the best algorithms (optimal algorithms) . We can see that they have very less completion times as their waiting time is minimum and SRTF being the preemptive version of SJF has even better performance.
2. In RR, the average completion time is higher because the time quantum taken for the given processes is 2 seconds, which is lower compared to the burst times of the processes, so there are many context switches.
3. In FCFS, the average completion time is higher because the burst time of the processes are higher and the waiting time of some of the processes are higher because of the other process.
4. In Multi Level Queue, processes in lower queues get CPU time only after the upper queues are empty. Hence, if processes keep arriving in the upper queue then processes in lower queues will be starved for CPU time. This explains the high average completion time in the above graph.
5. In the MultiLevel Feedback Queue (MLFQ), processes are first kept in the upper queue. If they are still not finished with their execution, then they are moved in lower queues. Thus, processes start executing fairly earlier compared to Multi Level Queue. Hence Average Completion Time is better compared to Multi Level Queue as seen in the above graph. If processes have large burst times compared to round robin time quantum, then they are relegated to lower queues which causes an increase in their completion time.
6. Preemptive priority as well as Non-Preemptive Priority has Average completion time which is comparable to others for this case, and which makes sense as the algorithm is designed to give priorities to processes and not for having minimum Average completion time.

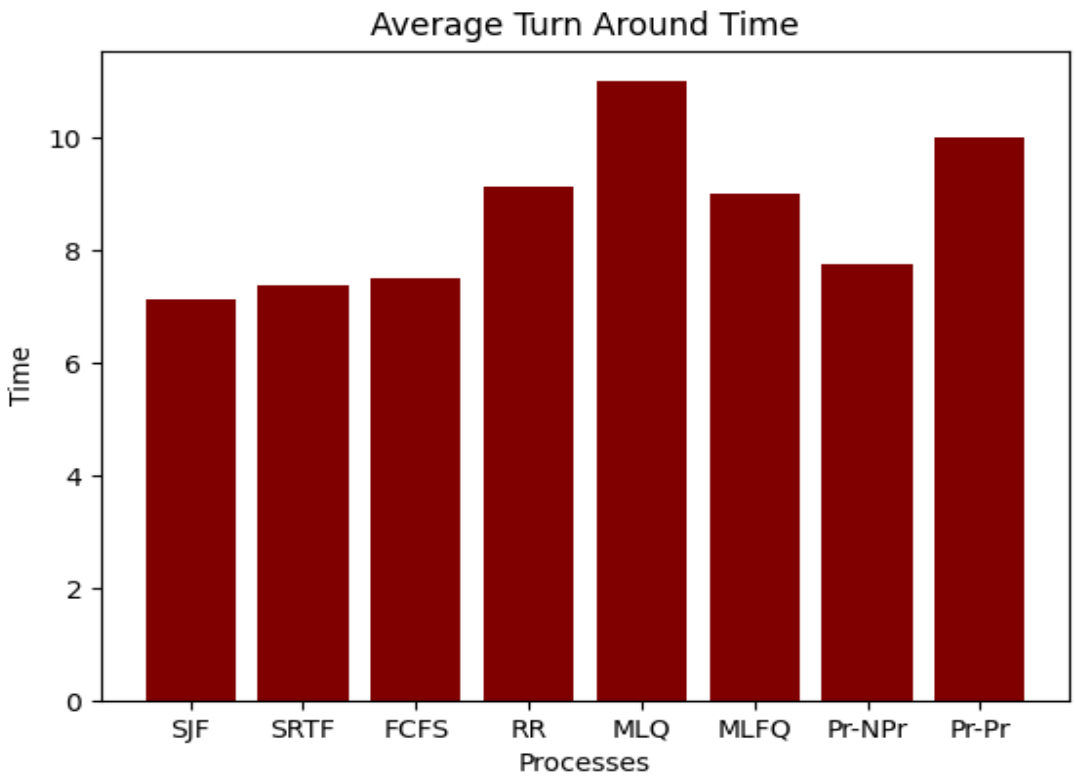


Bar Graph Showing Comparison of Average Waiting time Obtained if scheduled using different Process Scheduling algorithms.

1. SJF and SRTF are the closest we can get to optimal scheduling algorithms. As the definition itself states to give preference to processes with lower remaining time, they will have the least waiting time because more often than not, all the processes that are scheduled will have different burst times and it is only logical to complete processes with lower bursts first to increase efficiency and throughput which in turn decreases **average** waiting time.
2. In RR, the average waiting time is high because the time quantum taken for the given processes is 2 seconds, which is lower compared to the burst times of the processes, so there are many context switches and the processes have to wait more time to get the CPU allocated.
3. In FCFS, the average waiting time is high because the smaller process has to wait more time till the larger process, before it, gets executed.
4. In Multi Level Queue, processes in lower queues get CPU time only after the upper queues are empty. Hence, if processes keep arriving in the upper queue then processes

in lower queues will be starved for CPU time. This explains the high Average Waiting Time in above graph.

5. In the MultiLevel Feedback Queue (MLFQ), processes are first kept in the upper queue. If they are still not finished with their execution, then they are moved in lower queues. Thus, processes start executing fairly earlier compared to Multi Level Queue. Hence Average Waiting Time is better compared to Multi Level Queue as seen in the above graph. It can be observed that Average Waiting Time of MLFQ and RR is similar. This is because all queues in MLFQ except the last queue use Round Robin scheduling.
6. Preemptive priority has very large waiting time but Non-Preemptive Priority has lesser waiting time for this case, it is because the process which is causing large waiting time has lower priority and hence scheduled at last and hence causing a large waiting time.



Bar Graph Showing Comparison of Average Turnaround Time Obtained using various Process Scheduling algorithms.

1. Once again, SJF and SRTF are the closest we can get to optimal scheduling algorithms. Hence, in most of the cases, processes with smaller bursts complete faster than other algorithms and hence maintain lower **average** turnaround time.
2. In RR, the average turnaround time is higher because the time quantum taken for the given processes is 2 seconds, which is lower compared to the burst times of the processes, so there are many context switches and the time taken for the process to complete is higher..
3. In Multi Level Queue, processes in lower queues get CPU time only after the upper queues are empty. Hence, if processes keep arriving in the upper queue then processes in lower queues will be starved for CPU time. This explains the high Average Turnaround Time in above graph.
4. In the MultiLevel Feedback Queue (MLFQ), processes are first kept in the upper queue. If they are still not finished with their execution, then they are moved in lower queues. Thus, processes start executing fairly earlier compared to Multi Level Queue. Hence Average Turnaround Time is better compared to Multi Level Queue as seen in the above graph.
5. Preemptive priority as well as Non-Preemptive Priority has Average turnaround time which is comparable to others for this case, and which makes sense as the algorithm is designed to give priorities to processes and not for having minimum Average Turnaround time.

Conclusion

1. Scheduling Algorithms have a huge impact on the performance of computer systems. They can be considered as one of the most important parts.
2. Different scheduling algorithms give different performances. They all have their strengths and weaknesses
3. Two systems with the same hardware can have completely different performances based on the scheduling algorithms used. This enables us to use the same system for different tasks (within reasonable limits). All we have to do is change the scheduling policies and implement them using scheduling algorithms. This can save a lot of cost.
4. Although we have O(1) scheduling algorithm we do not use it as our requirement is not a fast scheduling algorithm rather our requirement is a scheduling algorithm which considers many other parameters , one such algorithm is CFS (Completely fair scheduler) introduced in kernel 2.6 for Linux based systems.