OPERATING SYSTEMS AND CONCURRENCY (G52OSC) COURSEWORK

Prepared for

Dr. Yasir Hafeez

University Of Nottingham Malaysia

December 2023

Group Members of PGB02:

Name	**Student ID**	**OWA**	**Course Programme**
Chong En Lua	20417309	hcycl5	CS
Joseph Emmanuel Huan Qin Chay	20580363	hcyjc11	CS
Jonathan James Rawding	20510088	hfyjk6	CS
Jie Zhan Keh	20375609	hfyjk6	CSAI

Table of Contents

1.0 Glossary

The table below shows the commonly used keywords and terms throughout the report.

Arrival Time	Time at which the process arrives and is ready for execution.
Burst Time	Amount of time required by process to complete execution.
Completion Time	Time at which the process finishes its execution.
Turnaround Time	Total time taken by a process from arrival to completion.
Waiting Time	Total time a process spends waiting in the ready queue before execution.
Response Time	Time elapsed between starting a process and receiving first response.
Average Turnaround Time	Mean value for turnaround time for all processes.
Average Waiting Time	Mean value for waiting time for all processes.
Average Response Time	Mean value for response time of all processes.
Execution Blocks	Segments of time where a process is executed.
Idle blocks	Periods of time when no process is executed.
Time Quantum	Fixed time duration allocation to each process in Round Robin and Multi-Level Feedback Queue.

2.0 Introduction

CPU management and process scheduling is very important in the field of Operating Systems and Concurrency. The importance lies in understanding how computers ensure optimal utilization of processes within a computer. This coursework aims to implement, but also to investigate 3 major scheduling algorithms: FCFS (First Come First Serve), RR (Round Robin), and Multi-Level Feedback Queue (MLFQ). MLFQ uses 2 queues. For MLFQ the first queue is the Round Robin queue, and the second queue is the First Come first serve queue.

Scheduling algorithms are important in determining the order of which processes are processed whilst also including the average waiting time and average turnaround time. The choice of which scheduling algorithm to use could potentially impact the system's responsiveness, throughput, and overall resource allocation in the computer. Therefore, through exploring FCFS, RR, and MLFQ, the group aims to gain a deeper insight into each of the scheduling algorithm and their respective strengths and weaknesses.

First Come First Serve (FCFS) requires only one queue and is the most straightforward scheduling algorithm to implement as the algorithm processes each process in terms of their arrival time and burst time. Its simplicity allows it to act as a foundation for implementing other scheduling principles.

Next, is the Round Robin (RR) scheduling algorithm where it also only requires one queue but operates using the RR principle whereby processes are dequeued and enqueued (preemption) according to the time quantum and burst time. This process ensures fairness in allowing all processes to be processed within a pre-defined time quantum, therefore each process receives regular opportunities for execution and processing.

The Multi-Level Feedback Queue (MLFQ) it stands out as the most sophisticated scheduling approach. It utilizes two queues: Round Robin as the first queue and First Come First Serve as the second queue. In this algorithm, the round robin queue has higher priority and therefore all processes initiate their execution on the round robin queue. As each process completes their time

quantum on the round robin queue, it is then demoted to the first come first serve queue to execute and complete.

The implementation of all 3 scheduling algorithms is in C programming language. Through the extensive implementation and investigation of each scheduling algorithm, the group aims to analyze and compare the performance of each scheduling algorithms through various scenarios.

3.0 Methodology

The program is fully implemented in the C programming language. Upon execution of the code, it allows the user to enter all the necessary inputs that the chosen scheduler requires. The inputs are number of tasks, arrival time, burst time. Specifically for Round Robin and Multi-Level Feedback Queue, an additional input of time quantum is required. This provides the user to test different workloads with different parameters.

The code uses a queue linked list data structure to implement all three scheduling algorithms. The queue linked list was chosen as the primary queue structure because of its linear time complexity of O (1) but also the advantage of having a dynamic memory allocation. Both enqueuing and dequeuing requires linear time complexity which is favorable for the scheduling algorithms.

Moving on to the scheduling algorithms, all three algorithms use a time-based loop. Where each algorithm operates by every unit of time. The loop continues to loop per unit time until all processes have been completed. This imitates the CPU scheduling process and allows for better understanding of the behavioral aspect of scheduling algorithms. Upon completion of the loop, and all processes are completed, a table will be displayed. The contents of the table include Process ID (in order of completion time), Arrival Time, Burst Time, Completion Time, Turnaround Time, Wait Time, Completion Time, and Response Time. In addition to the table, there are also 2 tables to show the initial queue and the process start and end times. Following, also displayed is a Gantt chart shown in units of time, and number of execution blocks and idle blocks.

As all 3 schedulers are combined into one code file, the user can freely choose the scheduling algorithm they want to use, each scheduler runs independently. By running one scheduler at a time, the program runs faster and allows for ease of comparisons.
Keywords and Terms

4.0 Briefing

In this report, the group will discuss and answer the four questions listed below.

1) Which scheduler is preferred for each workload and why?
2) Determine the average waiting time and the average turnaround time for different workloads when running under different schedulers.
3) What is the influence of different time quantum settings in RR on the average waiting time and on the interactivity of the system?
4) How would you tune the MLFQ scheduler? What time quantum works best?

Each of the questions will be thoroughly investigated by implementing a series of tests utilizing the coding framework program of the project. The produced outputs will then be used to provide assertion and credence to all our assessments and studies as well as providing deeper discussions and explanations for them. The subsequent questions, 2, 3, and 4, will extend and utilize the foundational data and insights provided from question 1. This approach will further ensure the coherence in the discussions in the report.

5.2 Schedulers

5.2.1 First Come First Served Scheduler

First Come First Serve is a non-preemptive process scheduling algorithm. The FCFS policy implementation is managed by a First in First Out (FIFO) queue, meaning that processes are executed based on their arrival times. When a process arrives and is in the ready queue, the Process Control Block (PCB) is connected to tail end of the queue, therefore when the CPU is available the first process is at the head of the queue and is executed.

The user first enters the number of processes, arrival time, and the burst time for each process. Then the scheduling calculation will begin in the function `**solveFCFS**`.

Based on the code snippet provided in Appendix A under the method `**solveFCFS**`, the processes are dispatched according to their arrival time, with the earliest arriving process receiving the allocation from CPU time first. The runtime is used to simulate the global time throughout the entire program.

The function mainly utilizes 3 main queues, namely the waiting, running and completed queues. Waiting queue is loaded and sorted in ascending order of earliest arrival times using the bubble sort algorithm. The running queue is used for the process that is currently using the CPU while the completed queue is for processes that have completed the execution and dequeued from the running queue.

Each process in the scheduler is represented by a Process Control Blocks which holds information on the process. Its information will be updated overtime in respect to the processing situations.

The waiting and running queues are looped through until both queues are empty, ensuring that all the processes are completed. At each iteration, the algorithm checks for CPU idle time periods where no process is running. If the CPU is idle and there are pending processes, and idle block is recorded in the Gantt Chart.

Next, when the process from the waiting queue is ready to run, it is moved to the running queue, whereby no preemptions will be performed according to the nature of FCFS. This is evident based on the decrement of the remaining burst time (`**rbt**`) until it reaches zero.

For each running process, the code tracks whether the specific process has begun its execution (`**responded**` flag) and records the start time for that process. As each process completes, it is moved to the completed queue whereby the completion (`**ct**`), turnaround (`**tat**`) and waiting (`**wt**`)

times are calculated. The turnaround time is the total time taken from the arrival to completion, and the waiting times will be the difference between the turnaround and burst time (`**bt**`).

Inclusively, the program's global time (`**runtime**`) will be incremented upon each loop's iteration which will demonstrate the progression of time in the environment.

Finally, when waiting and running queues are finally both empty, the function returns the `**SolutionResult**` structure which contains the `**completedQueue**` and the Gantt Chart array (to be used for further visualization and analysis of the scheduling process).

5.2.1 Round Robin Scheduler

The function which is responsible for the scheduling algorithm for Round Robin is `**solveRR**` as seen in the source code in Appendix A. Similarly, like the `**solveFCFS**` function, it utilizes the waiting, running and completed queues. The process works in much likeness to the FCFS algorithm; however this implementation assigns a given fixed quantum time for each of the process execution enabling context switching after each quantum period has expired, ensuring that no process will use the CPU for more than the allowed period.

The main difference between this and the FCFS algorithm is using the variable `**quantumLeft**` to keep track of the remaining space allowed for execution before preemption. In each cycle, when a process is running and has reached its maximum quantum space but is still incomplete, it will be given another quantum in the next cycle and so on until its execution is fully complete. The `**quantumLeft**` variable ensures that each process will not be given extra time slice than the set quantum.

Processes that are incomplete, but have no more remaining quantum will be dequeued form the `**runningQueue**` and enqueued to the tail / rear / back of the queue again and wait for the next available quantum. At the same time the remaining quantum will reset to start afresh in the next loop for the next upcoming process. Otherwise, if the process's remaining burst time (`**rbt**`) has reached zero within its cycle, it will be moved to the `**completedQueue**`

This written algorithm emphasizes on a time-shared environment, ensuring that each process gets an equal opportunity to use the CPU.

Finally, upon completion of the entire workload, the `**SolutionResult**` structure is returned with the `**completedQueue**` and the `**GanttChart** ` with its size.

5.2.1 Multi-Level Feedback Queue Scheduler

This algorithm is designed to optimize both the waiting and turnaround times by dynamically placing processes in the appropriate priority queues based on their allocated burst times and the given quantum via demotion.

This algorithm is executed by the function `**solveMLFQ**` found in the source code in Appendix A utilizes a waiting queue, 2 running queues (RR running queue and FCFS running queue) and a completed queue. The RR queue serves as the higher priority queue while the FCFS serves as the lower priority queue.

The function works in similar fashion with inclusive functionalities found in both the `**solveFCFS**` and `**solveRR**` where each new process arrival from the `**waitingQueue**` will be added to the `**runningRRQueue**` as they become available, checks if any idle CPU times have been found, and begins to process tasks in the RR queue first. Tasks with still a non-zero remaining burst time (`**rbt**`) but already has its remaining quantum slice (`**quantumLeft**`) at zero will be demoted to the `**runningFCFSQueue**`. As shown in the function, the `**incomplete**` process is dequeued from the `**runningRRQueue**` and enqueued to the `**runningFCFSQueue**`.

The `**runningFCFSQueue**` is only executed when there is no longer any presence of tasks left in the `**runningRRQueue**`, making sure that all the higher priority tasks be executed first.

During each cycle, the function assesses whether a process has completed its execution based on the remaining burst time (`**rbt**`). If a process finishes, it is moved to the completed queue, and its completion (`**ct**`), turnaround (`**tat**`), and waiting (`**wt**`) times are calculated, contributing to the performance metrics of the scheduling algorithm.

Finally, the completed queue and Gantt chart is output from the function once the waiting and 2 running queues have been emptied.

6.0 Study and Discussions

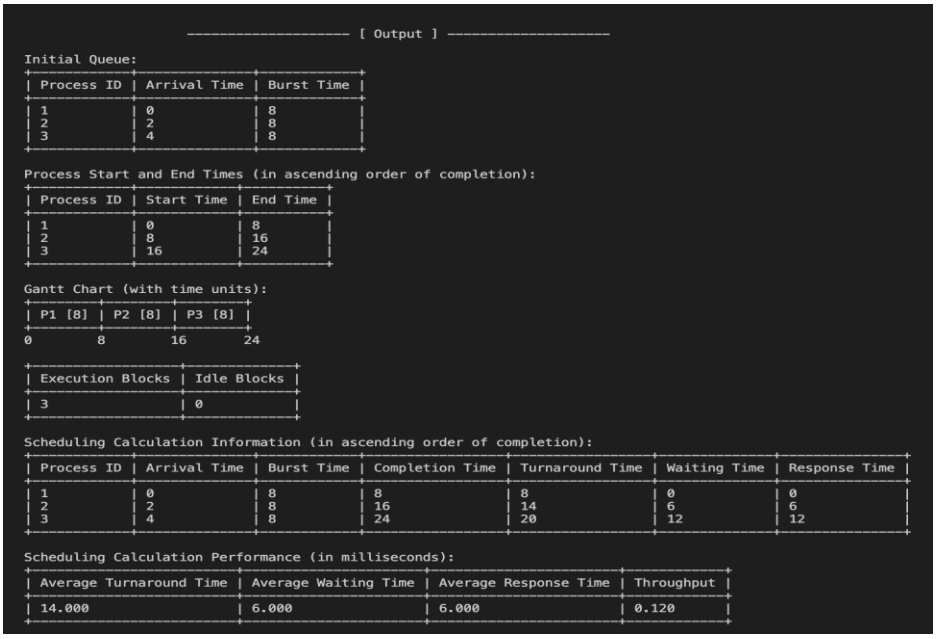
6.1 Scheduler Preferences and Averages for Waiting and Turnaround Times for different Workloads.

6.1.1 Workload 1: Processes with Fixed Burst Time

In this workload scenario, the first analysis of the scheduling algorithm is using a fixed/ constant burst time of 8 units of time. The analysis will apply to the 3 scheduling algorithms: First Come First Serve (FCFS), Round Robin (RR), and Multi-Level Feedback Queue (MLFQ). By making the burst time constant, it ensures a fair comparison to observe the differences in performance of each scheduling algorithm. Therefore, the focus of performance will be based on two metrics: average turnaround time and average waiting time. And overall, concentrates on how each scheduler handles a set of tasks.

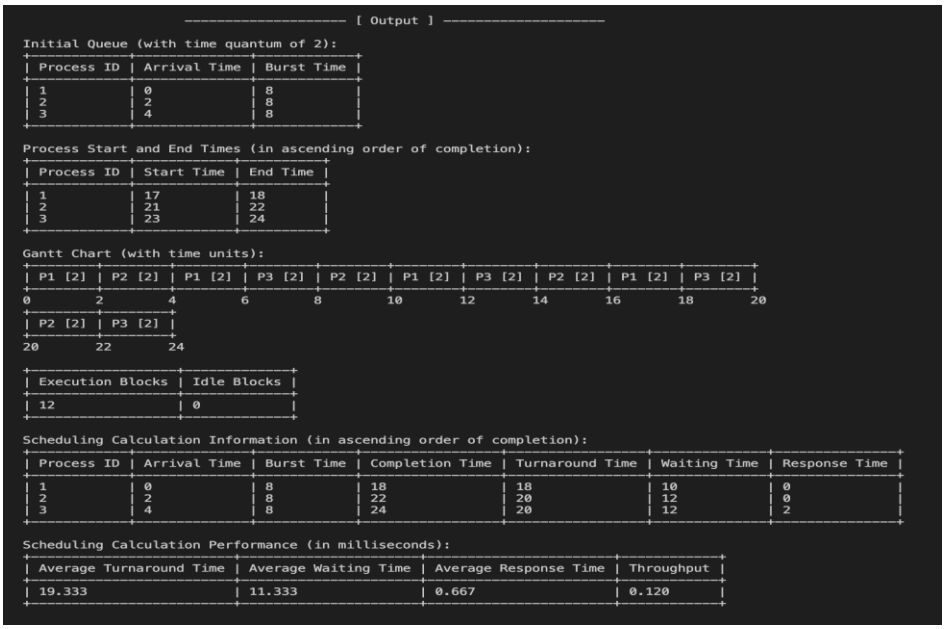
The task sets to be used is [0, 2, 4], [0, 3, 6], and [0, 4, 8]. As for Round Robin (RR) and Multi-Level Feedback Queue (MLFQ), it will be using a time quantum of 3. The time quantum of 3 was considered to aid with balancing the allocation of tasks and ensuring that no tasks are executed for too long, allowing all processes to be executed in a timely manner.

The time quantum of 2 was also used for RR and MLFQ to help for testing purposes.



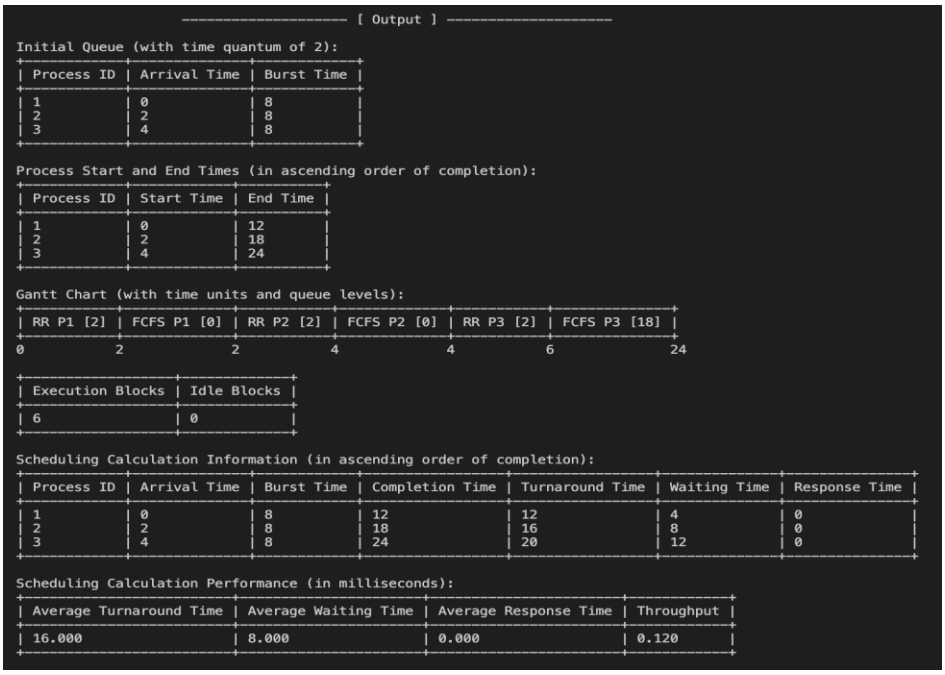
```
                    ---------- [ Output ] ----------

Initial Queue:
+------------+--------------+------------+
| Process ID | Arrival Time | Burst Time |
+------------+--------------+------------+
| 1          | 0            | 8          |
| 2          | 2            | 8          |
| 3          | 4            | 8          |
+------------+--------------+------------+

Process Start and End Times (in ascending order of completion):
+------------+------------+----------+
| Process ID | Start Time | End Time |
+------------+------------+----------+
| 1          | 0          | 8        |
| 2          | 8          | 16       |
| 3          | 16         | 24       |
+------------+------------+----------+

Gantt Chart (with time units):
+---------+---------+---------+
| P1 [8]  | P2 [8]  | P3 [8]  |
+---------+---------+---------+
0         8         16        24

+-----------------+--------------+
| Execution Blocks | Idle Blocks |
+-----------------+--------------+
| 3                | 0           |
+-----------------+--------------+

Scheduling Calculation Information (in ascending order of completion):
+------------+--------------+------------+-----------------+-----------------+--------------+---------------+
| Process ID | Arrival Time | Burst Time | Completion Time | Turnaround Time | Waiting Time | Response Time |
+------------+--------------+------------+-----------------+-----------------+--------------+---------------+
| 1          | 0            | 8          | 8               | 8               | 0            | 0             |
| 2          | 2            | 8          | 16              | 14              | 6            | 6             |
| 3          | 4            | 8          | 24              | 20              | 12           | 12            |
+------------+--------------+------------+-----------------+-----------------+--------------+---------------+

Scheduling Calculation Performance (in milliseconds):
+-------------------------+-----------------------+-------------------------+------------+
| Average Turnaround Time | Average Waiting Time  | Average Response Time   | Throughput |
+-------------------------+-----------------------+-------------------------+------------+
| 14.000                  | 6.000                 | 6.000                   | 0.120      |
+-------------------------+-----------------------+-------------------------+------------+
```

Figure 6.1: FCFS, Task Set [0, 2, 4] with fixed burst time of 8



```
                    ---------- [ Output ] ----------

Initial Queue (with time quantum of 2):
+------------+--------------+------------+
| Process ID | Arrival Time | Burst Time |
+------------+--------------+------------+
| 1          | 0            | 8          |
| 2          | 2            | 8          |
| 3          | 4            | 8          |
+------------+--------------+------------+

Process Start and End Times (in ascending order of completion):
+------------+------------+----------+
| Process ID | Start Time | End Time |
+------------+------------+----------+
| 1          | 17         | 18       |
| 2          | 21         | 22       |
| 3          | 23         | 24       |
+------------+------------+----------+

Gantt Chart (with time units):
+---------+---------+---------+---------+---------+---------+---------+---------+---------+---------+
| P1 [2]  | P2 [2]  | P1 [2]  | P3 [2]  | P2 [2]  | P1 [2]  | P3 [2]  | P2 [2]  | P1 [2]  | P3 [2]  |
+---------+---------+---------+---------+---------+---------+---------+---------+---------+---------+
0         2         4         6         8         10        12        14        16        18        20
+---------+---------+
| P2 [2]  | P3 [2]  |
+---------+---------+
20        22        24

+-----------------+--------------+
| Execution Blocks | Idle Blocks |
+-----------------+--------------+
| 12               | 0           |
+-----------------+--------------+

Scheduling Calculation Information (in ascending order of completion):
+------------+--------------+------------+-----------------+-----------------+--------------+---------------+
| Process ID | Arrival Time | Burst Time | Completion Time | Turnaround Time | Waiting Time | Response Time |
+------------+--------------+------------+-----------------+-----------------+--------------+---------------+
| 1          | 0            | 8          | 18              | 18              | 10           | 0             |
| 2          | 2            | 8          | 22              | 20              | 12           | 0             |
| 3          | 4            | 8          | 24              | 20              | 12           | 2             |
+------------+--------------+------------+-----------------+-----------------+--------------+---------------+

Scheduling Calculation Performance (in milliseconds):
+-------------------------+-----------------------+-------------------------+------------+
| Average Turnaround Time | Average Waiting Time  | Average Response Time   | Throughput |
+-------------------------+-----------------------+-------------------------+------------+
| 19.333                  | 11.333                | 0.667                   | 0.120      |
+-------------------------+-----------------------+-------------------------+------------+
```

Figure 6.2: RR, Task Set [0, 2, 4] with fixed burst time of 8, time quantum of 2



```
                    ---------- [ Output ] ----------

Initial Queue (with time quantum of 2):
+------------+--------------+------------+
| Process ID | Arrival Time | Burst Time |
+------------+--------------+------------+
| 1          | 0            | 8          |
| 2          | 2            | 8          |
| 3          | 4            | 8          |
+------------+--------------+------------+

Process Start and End Times (in ascending order of completion):
+------------+------------+----------+
| Process ID | Start Time | End Time |
+------------+------------+----------+
| 1          | 0          | 12       |
| 2          | 2          | 18       |
| 3          | 4          | 24       |
+------------+------------+----------+

Gantt Chart (with time units and queue levels):
+-----------+-------------+-----------+-------------+-----------+-------------+
| RR P1 [2] | FCFS P1 [0] | RR P2 [2] | FCFS P2 [0] | RR P3 [2] | FCFS P3 [18] |
+-----------+-------------+-----------+-------------+-----------+-------------+
0           2             2           4             4           6             24

+------------------+--------------+
| Execution Blocks | Idle Blocks |
+------------------+--------------+
| 6                | 0            |
+------------------+--------------+

Scheduling Calculation Information (in ascending order of completion):
+------------+--------------+------------+-----------------+-----------------+--------------+---------------+
| Process ID | Arrival Time | Burst Time | Completion Time | Turnaround Time | Waiting Time | Response Time |
+------------+--------------+------------+-----------------+-----------------+--------------+---------------+
| 1          | 0            | 8          | 12              | 12              | 4            | 0             |
| 2          | 2            | 8          | 18              | 16              | 8            | 0             |
| 3          | 4            | 8          | 24              | 20              | 12           | 0             |
+------------+--------------+------------+-----------------+-----------------+--------------+---------------+

Scheduling Calculation Performance (in milliseconds):
+------------------------+----------------------+------------------------+------------+
| Average Turnaround Time | Average Waiting Time | Average Response Time | Throughput |
+------------------------+----------------------+------------------------+------------+
| 16.000                 | 8.000                | 0.000                  | 0.120      |
+------------------------+----------------------+------------------------+------------+
```

Figure 6.3: MLFQ, Task Set [0, 2, 4] with fixed burst time of 8, time quantum of 2

Figures 6.1, 6.2, and 6.3 are images of the output when using task set [0, 2, 4]. The output of the code includes initial queue, process start and end times, Gantt chart, execution blocks and idle blocks, scheduling calculation information, and scheduling calculation performance.

Below are tables and charts that display the full analysis of the workload.

Average Waiting Time (Fixed Burst Time of 8)					
Task Set / Scheduler	FCFS	RR (Time Quantum = 2)	RR (Time Quantum = 3)	MLFQ (Time Quantum = 2)	MLFQ (Time Quantum = 3)
[0, 2, 4]	6.000	11.333	11.000	8.000	9.000
[0, 3, 6]	5.000	9.000	10.000	7.000	8.000
[0, 4, 8]	4.000	7.333	7.000	6.000	7.000

Figure 6.4: Table of average waiting time with fixed burst time of 8

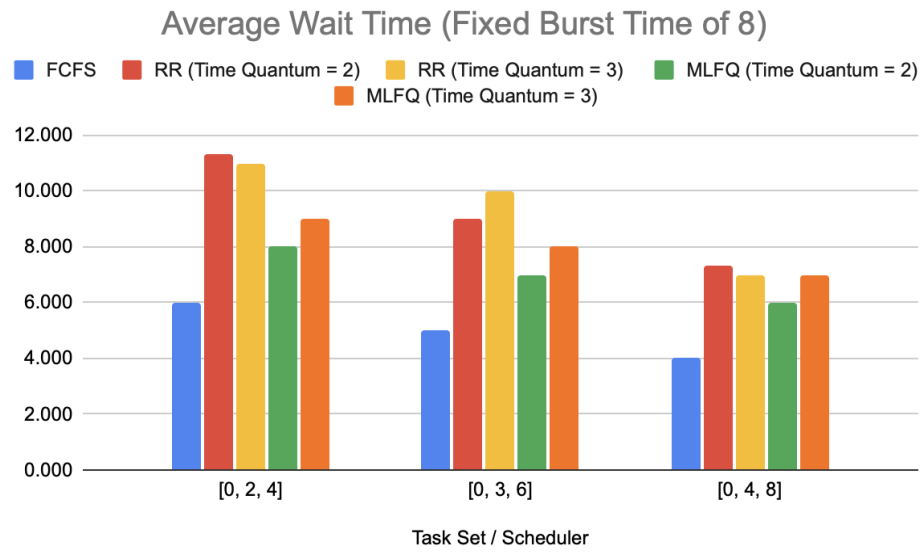


Figure 6.5: Bar chart of the results for average waiting time

Average Turn Around Time (Fixed Burst Time of 8)					
Task Set / Scheduler	FCFS	RR (Time Quantum = 2)	RR (Time Quantum = 3)	MLFQ (Time Quantum = 2)	MLFQ (Time Quantum = 3)
[0, 2, 4]	14.000	19.333	19.000	16.333	17.333
[0, 3, 6]	13.000	17.000	18.000	15.000	16.000
[0, 4, 8]	12.000	15.333	15.000	14.000	15.000

Figure 6.6: Table of average turnaround time with fixed burst time of 8

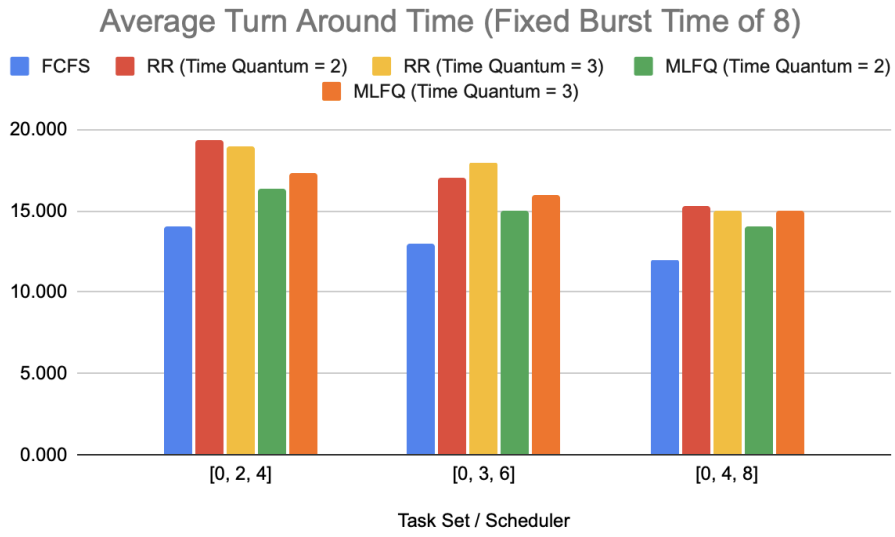


Figure 6.7: Bar chart displaying results for average turnaround time

Observations For Average Waiting Time

Through the data collected, First Come First Serve (FCFS) reveals a clear relationship between the difference in arrival times and waiting time. As the difference between arrival times increase, the waiting time increases as well. This is due to the FCFS principle where tasks are prioritized based on arrival time and tasks with longer burst time could potentially lead to increased waiting time for the remaining tasks.

Looking at Round Robin (RR), RR with a time quantum of 2 showed higher waiting times compared to RR with a time quantum of 3. Due to the nature of RR, a smaller time quantum in comparison to its burst time relates to more frequent content switching and additional overhead.

The Multi-Level Feedback Queue (MLFQ) surpasses FCFS and RR in terms of average waiting time and average turnaround time for all task sets. Looking at MLFQ dynamic and versatile nature, using both RR and FCFS queues, it can adapt to varying task sets. Therefore, its stable average waiting times and average turnaround times makes its an advantageous choice for task sets with varying arrival times.

Conclusively, MLFQ with time quantum of 2 demonstrated improved results in comparison to FCFS and RR.

Observations For Average Turnaround Time

First Come First Serve (FCFS) showed higher turnaround times compared to the other scheduling algorithms. This result aligns with the FCFS principle as processes are executed based on arrival times. Where processes with longer burst time will delay the turnaround time for the subsequent processes.

Looking at Round Robin (RR), with time quantum of 3, it resulted in improved turnaround times when compared with time quantum of 2. The assumption that the shorter the time quantum, the more content switches occur and therefore an increased overhead which leads to greater turnaround times. Through all 3 task sets, RR with time quantum of 3 does generally show that it has higher turnaround times than MLFQ.

Multi-level Feedback Queue (MLFQ) revealed improved turnaround times across all tasks when compared to FCFS and RR. This can be attributed to MLFQ dynamic and adaptable nature by using 2 different queues. MLFQ with a time quantum of 3 demonstrated higher turnaround times compared to MLFQ with a time quantum of 2, this indicates that a smaller time quantum potentially contributes to better turnaround times for specific task sets.

Overall, MLFQ with a time quantum of 2 is the most favorable scheduling algorithm. It provides better turnaround times when compared to FCFS and RR. The consistency and ability to adapt to the task sets and providing favorable results makes MLFQ the scheduler to be chosen.

Conclusion for Workload 1

Observing the performance metrics for the First Come First Serve (FCFS), Round Robin (RR), and Multi-Level Feedback Queue (MLFQ) scheduling algorithms, the data and task set tests have provided valuable insights into the effectiveness of each scheduler under different task sets. The metrics focused on average waiting time and average turnaround time.

From the task sets used, when all processes have the same burst time, FCFS emerges as the clear performer. This occurs because each task has the same burst time and therefore it automatically ensures a fair distribution of units of time to each process. Because of the nature of FCFS, by following the FCFS principle, the sequential execution process means that FCFS is able to perform well in this workload. In addition, FCFS is also easy and simple to implement, it is straightforward and not as complicated when compared to RR and MLFQ.

6.1.2 Workload 2: Processes with Increasing Burst Time

In this workload scenario, as for testing increasing burst time, the arrival time will be constant 0. However, the burst time will be increasing with a constant difference of 3, 4, and 5. The time quantum for RR and MLFQ will be kept at 3. The time quantum of 3 was considered to aid with balancing the allocation of tasks and ensuring that no tasks are executed for too long, allowing all processes to be executed in a timely manner. Therefore, the task set will be [3, 6, 9], [4, 8, 12], and [5, 10, 15].

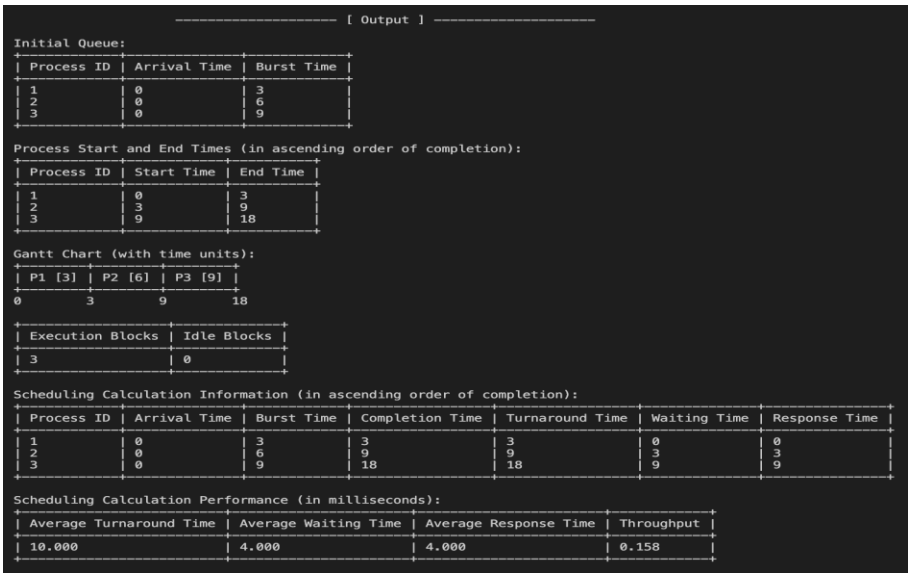


Figure 6.8: FCFS, Task Set [3, 6, 9] with fixed arrival time of 0

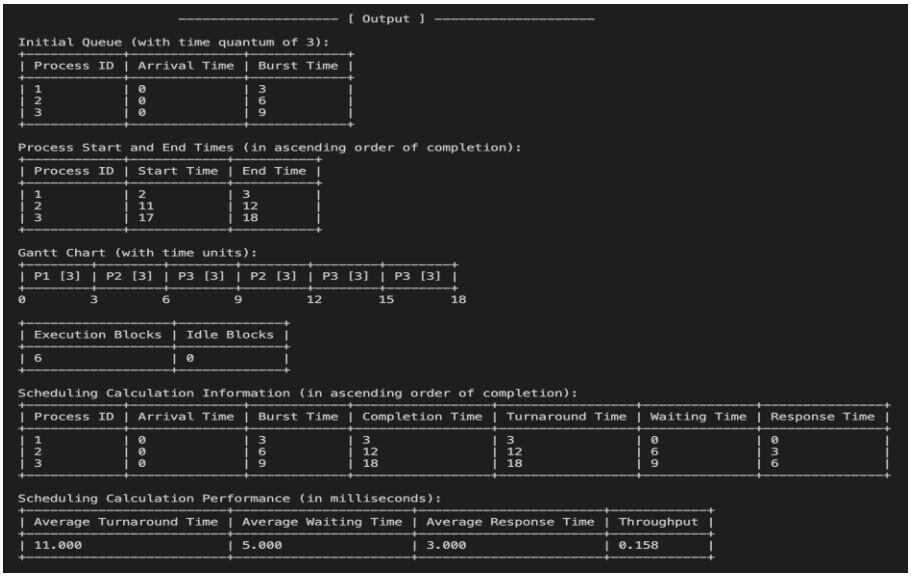


Figure 6.9: RR, Task Set [3, 6, 9] with fixed arrival time of 0, time quantum of 3

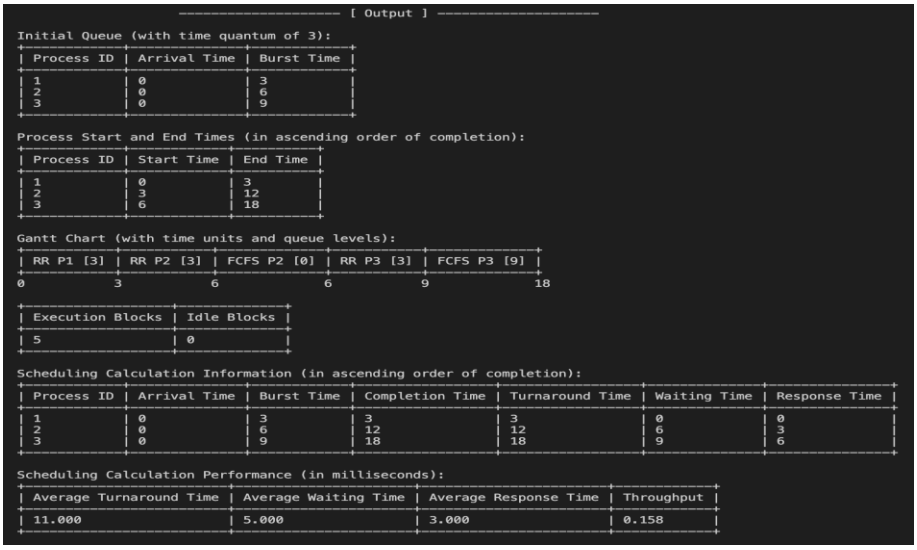


Figure 6.10: MLFQ, Task Set [3, 6, 9] with fixed arrival time of 0, time quantum of 3

Figures 6.8, 6.9, 7.0 are images of the output when using task set [3, 6, 9]. The output of the code includes initial queue, process start and end times, Gantt chart, execution blocks and idle blocks, scheduling calculation information, and scheduling calculation performance.

Below are tables and charts that display the full analysis of the workload.

Average Waiting Time (Fixed Arrival Time of 0)			
Task Set / Scheduler	FCFS	RR (Time Quantum = 3)	MLFQ (Time Quantum = 3)
[3, 6, 9]	4.000	5.000	5.000
[4, 8, 12]	5.333	9.333	8.333
[5, 10, 15]	6.667	11.667	9.667

Figure 6.11: Table of average waiting time with fixed arrival time of 0

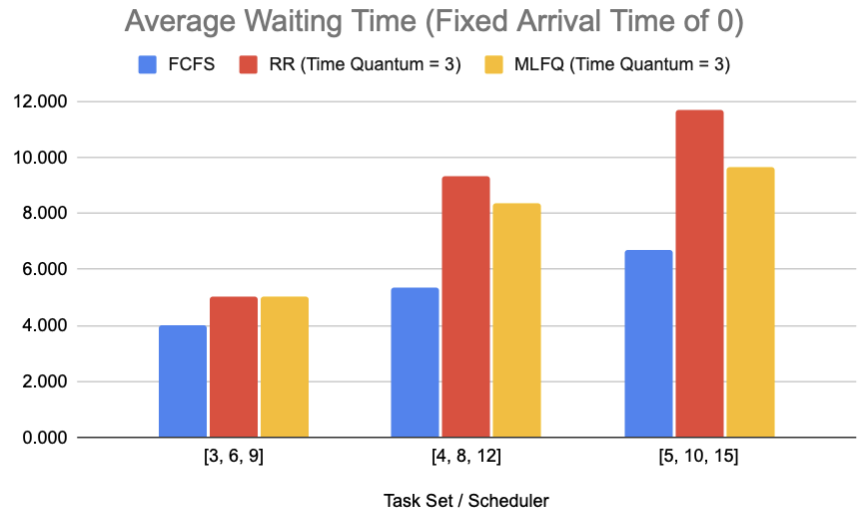


Figure 6.12: Bar chart of results for average waiting time

Average Turnaround Time (Fixed Arrival Time of 0)			
Task Set / Scheduler	FCFS	RR (Time Quantum = 3)	MLFQ (Time Quantum = 3)
[3, 6, 9]	10.000	11.000	11.000
[4, 8, 12]	13.333	17.333	16.333
[5, 10, 15]	16.667	21.667	19.667

Figure 6.13: Table of average turnaround time with fixed arrival time of 0

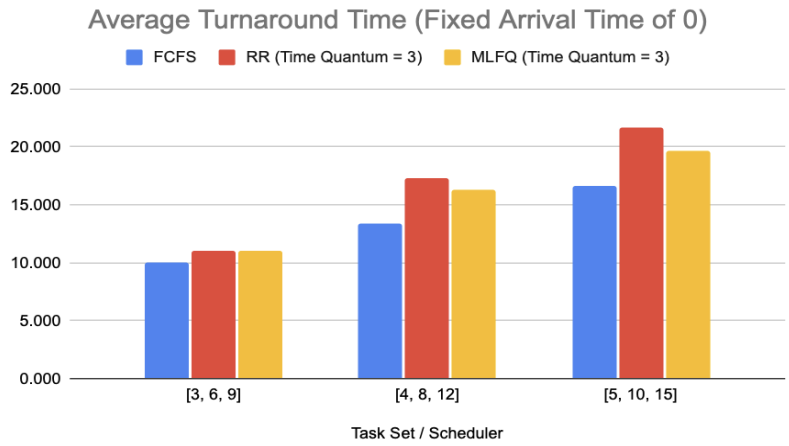


Figure 6.14: Bar chart displaying results for average turnaround time

Observations On Average Waiting Time

First Come First Serve (FCFS) demonstrated lower waiting times for shorter burst times. However, this does also mean that when burst time increase, the wait times increase as well. This is due to the nature of the FCFS principle whereby processes are executed based on arrival time. And processes are executed in a sequential manner.

Looking at Round Robin (RR), it showed consistent waiting times across all task sets. The time quantum of 3 units ensures that all processes are given the opportunity to execute. But looking at the table and the bar chart, it underperforms against FCFS and MLFQ.

As for Multi-Level Feedback Queue (MLFQ), it can adapt to the different task sets. It showed competitive results when compared to FCFS, this applies to task sets with longer burst times. The adaptive nature of MLFQ with 2 queues and priority, contributes to the adaptability to varying burst times.

Overall, FCFS scheduler outperforms RR and MLFQ. This is mainly due to the nature of RR and MLFQ where it includes a time quantum and depends on processes being completed.

Observations on Average Turnaround Time

Directing attention to the average turnaround time, FCFS results higher turnaround times when burst time increases. This is because in FCFS, processes are executed in order or arrival time.

Round Robin (RR) provides fair and consistent turnaround times with a time quantum of 3. And for Multi-Level Feedback Queue (MLFQ), it demonstrated good results because of the implementation of 2 queues. The results from RR with time quantum of 2 was also fair and consistent, but it also meant more content switching due to lower time quantum.

Overall, the results in average turnaround time across the 3 schedulers showed the importance in efficiency of the process execution. FCFS has shown to prioritized smaller tasks by executing them first according to the FCFS principle, this characteristic of FCFS is advantageous in this case where quicker completion of smaller tasks is of main importance. Whilst RR resulted in a fair and consistent outcome, by ensuring that all processes can execute on the ready queue. When turning the attention to MLFQ, leveraging both RR and FCFS, it shows fairly promising results by having the advantage of being able to adjust. However, the FCFS scheduler is the preferred option as it can give the smaller tasks priority first and to allow them to complete before moving on to the subsequent processes.

Conclusion on Workload 2

Results from FCFS demonstrated lower wait times for tasks with shorter burst time, aligning with its principle of prioritizing processes based on arrival time. However, when burst time increased, it showed that the average waiting times also increased and therefore a possible drawback to its performance.

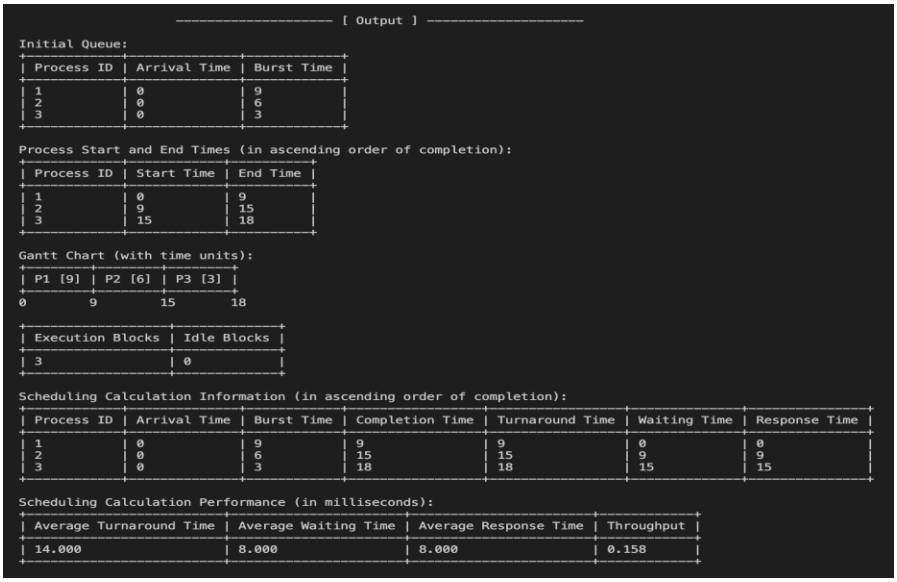
Round Robin with time quantum of 3 displayed consistent results in line with the RR principle. Although it clearly did not perform as well when compared to FCFS and MLFQ, it allowed for a fair execution process.

With the adaptable nature of MLFQ, it was able to demonstrate competitive results compared to FCFS, especially in task sets with longer burst times.

But however, overall, FCFS emerges as the best performing scheduler by prioritizing smaller tasks and executing them in a sequential process where quick completion is a priority.

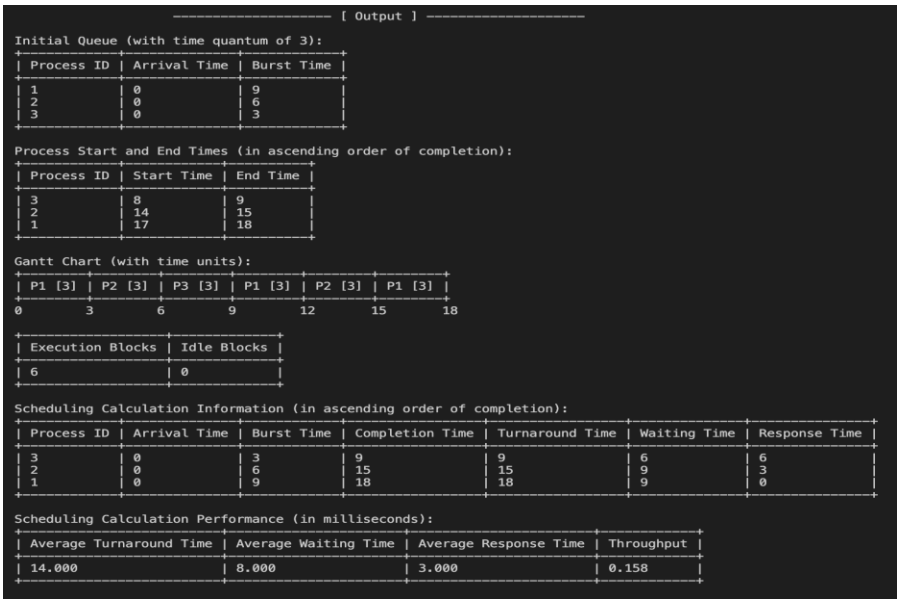
6.1.3 Workload 3: Processes with Decreasing Burst Times

In this workload scenario, similarly, to testing increasing burst time, the arrival time will be constant 0. However, the burst time will be decreasing with a constant difference of 3, 4, and 5. The time quantum for RR and MLFQ will be kept at 3. Therefore, the task set will be [9, 6, 3], [12, 8, 4], [15, 10, 5].



```
---------- [ Output ] ----------
Initial Queue:
+------------+--------------+------------+
| Process ID | Arrival Time | Burst Time |
+------------+--------------+------------+
| 1          | 0            | 9          |
| 2          | 0            | 6          |
| 3          | 0            | 3          |
+------------+--------------+------------+

Process Start and End Times (in ascending order of completion):
+------------+------------+----------+
| Process ID | Start Time | End Time |
+------------+------------+----------+
| 1          | 0          | 9        |
| 2          | 9          | 15       |
| 3          | 15         | 18       |
+------------+------------+----------+

Gantt Chart (with time units):
+---------+---------+---------+
| P1 [9]  | P2 [6]  | P3 [3]  |
+---------+---------+---------+
0         9         15        18

+------------------+--------------+
| Execution Blocks | Idle Blocks  |
+------------------+--------------+
| 3                | 0            |
+------------------+--------------+

Scheduling Calculation Information (in ascending order of completion):
+------------+--------------+------------+-----------------+-----------------+--------------+---------------+
| Process ID | Arrival Time | Burst Time | Completion Time | Turnaround Time | Waiting Time | Response Time |
+------------+--------------+------------+-----------------+-----------------+--------------+---------------+
| 1          | 0            | 9          | 9               | 9               | 0            | 0             |
| 2          | 0            | 6          | 15              | 15              | 9            | 9             |
| 3          | 0            | 3          | 18              | 18              | 15           | 15            |
+------------+--------------+------------+-----------------+-----------------+--------------+---------------+

Scheduling Calculation Performance (in milliseconds):
+------------------------+----------------------+------------------------+------------+
| Average Turnaround Time | Average Waiting Time | Average Response Time | Throughput |
+------------------------+----------------------+------------------------+------------+
| 14.000                 | 8.000                | 8.000                  | 0.158      |
+------------------------+----------------------+------------------------+------------+
```

Figure 6.15: FCFS, Task Set [9, 6, 3] with fixed arrival time of 0



```
---------- [ Output ] ----------
Initial Queue (with time quantum of 3):
+------------+--------------+------------+
| Process ID | Arrival Time | Burst Time |
+------------+--------------+------------+
| 1          | 0            | 9          |
| 2          | 0            | 6          |
| 3          | 0            | 3          |
+------------+--------------+------------+

Process Start and End Times (in ascending order of completion):
+------------+------------+----------+
| Process ID | Start Time | End Time |
+------------+------------+----------+
| 3          | 8          | 9        |
| 2          | 14         | 15       |
| 1          | 17         | 18       |
+------------+------------+----------+

Gantt Chart (with time units):
+---------+---------+---------+---------+---------+---------+
| P1 [3]  | P2 [3]  | P3 [3]  | P1 [3]  | P2 [3]  | P1 [3]  |
+---------+---------+---------+---------+---------+---------+
0         3         6         9         12        15        18

+------------------+--------------+
| Execution Blocks | Idle Blocks  |
+------------------+--------------+
| 6                | 0            |
+------------------+--------------+

Scheduling Calculation Information (in ascending order of completion):
+------------+--------------+------------+-----------------+-----------------+--------------+---------------+
| Process ID | Arrival Time | Burst Time | Completion Time | Turnaround Time | Waiting Time | Response Time |
+------------+--------------+------------+-----------------+-----------------+--------------+---------------+
| 3          | 0            | 3          | 9               | 9               | 6            | 6             |
| 2          | 0            | 6          | 15              | 15              | 9            | 3             |
| 1          | 0            | 9          | 18              | 18              | 9            | 0             |
+------------+--------------+------------+-----------------+-----------------+--------------+---------------+

Scheduling Calculation Performance (in milliseconds):
+------------------------+----------------------+------------------------+------------+
| Average Turnaround Time | Average Waiting Time | Average Response Time | Throughput |
+------------------------+----------------------+------------------------+------------+
| 14.000                 | 8.000                | 3.000                  | 0.158      |
+------------------------+----------------------+------------------------+------------+
```

Figure 6.16: RR, Task Set [9, 6, 3] with fixed arrival time of 0, time quantum of 3

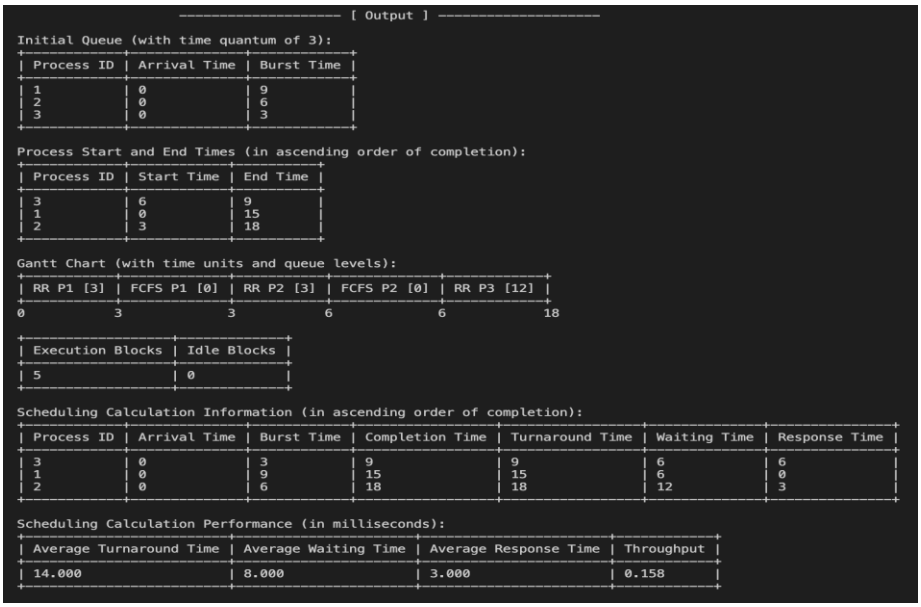


Figure 6.17: MLFQ, Task Set [9, 6, 3] with fixed arrival time of 0, time quantum of 3

Figures 6.15, 6.16, 6.17 are images of the output when using task set [9, 6, 3]. The output of the code includes initial queue, process start and end times, Gantt chart, execution blocks and idle blocks, scheduling calculation information, and scheduling calculation performance.

Below are tables and charts that display the full analysis of the workload.

Average Waiting Time (Fixed Arrival Time of 0)			
Task Set / Scheduler	FCFS	RR (Time Quantum = 3)	MLFQ (Time Quantum = 3)
[9, 6, 3]	8.000	8.000	8.000
[12 ,8, 4]	10.667	12.333	13.667
[15, 10, 5]	13.333	14.667	16.333

Figure 6.18: Table displaying average waiting time with fixed arrival time of 0

Average Turnaround Time (Fixed Arrival Time of 0)			
Task Set / Scheduler	FCFS	RR (Time Quantum = 3)	MLFQ (Time Quantum = 3)

[9, 6, 3]	14.000	14.000	14.000
[12 ,8, 4]	18.667	20.333	21.667
[15, 10, 5]	23.333	24.667	26.333

Figure 6.19: Table displaying average turnaround time with fixed arrival time of 0

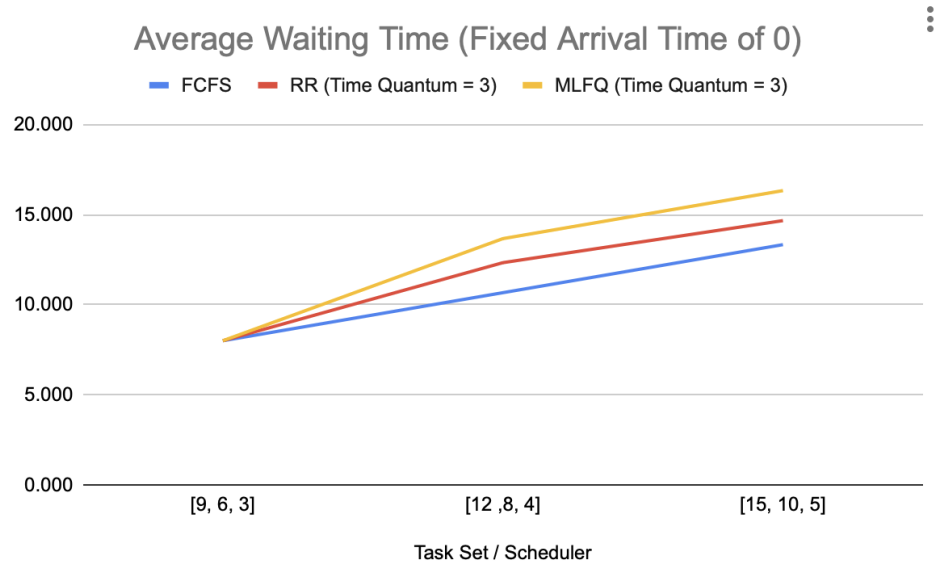


Figure 6.20: Line graph displaying the results of average waiting time, fixed arrival time of 0

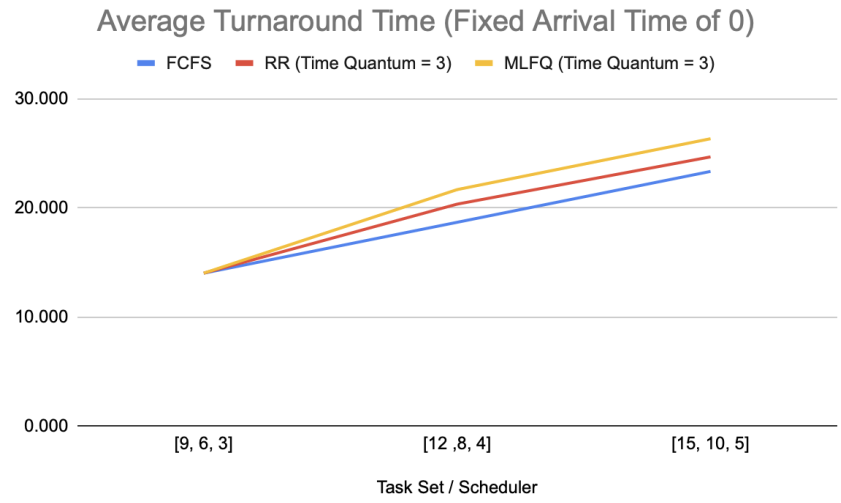


Figure 6.21: Line graph displaying the results of average turnaround time, fixed arrival time of 0

Observations On Average Waiting Time And Average Turnaround Time

First Come First Serve (FCFS) revealed lower waiting times for shorter burst time in the task set. But as burst times increase, FCFS results in higher waiting times naturally due to the FCFS principle of sequential execution.

On the other hand, Round Robin (RR) and Multi-Level Feedback Queue (MLFQ) generally reduced average waiting times and turnaround times. This occurs because the time quantum allows for content switching between processes and the adaptability of MLFQ utilizing 2 queues. Therefore, the data shows that when the workload is balanced/ similar burst time differences, RR and MLFQ can perform well and better than FCFS.

Furthermore, for processes with longer burst time, MLFQ's performance decreases whilst RR performance increases. This is because once the process finishes the time quantum on the RR queue in MLFQ, it is demoted to FCFS queue to finish, and therefore follows the basic FCFS principle of sequence execution. Whereas for RR, it continues to allocate the same amount of time quantum for every process and continuously preempts processes to execute subsequent processes until they finish.

To conclude, RR and MLFQ performs better than FCFS for certain workloads. This is mainly attributed to the time quantum and content switching adaptability of both RR and MLFQ to quickly expedite the execution of processes with shorter burst times. But ultimately, when the burst time decreases, RR performs best as operating in cycles and time quantum allows the process to be completed earlier.

Conclusion for workload 3

Results from FCFS showed low average waiting times for the task set with shorter burst times. But keeping in mind that as the burst time increased, so does the waiting times. And so, this does highlight the limitation of FCFS when it handles longer tasks.

Whilst on the other hand, RR and MLFQ both showed a general decrease in average waiting times and average turnaround times. This means that the improvement in the decrease can be attributed to the adaptability in both schedulers. For RR, content switching between processes within a fixed time quantum and MLFQ utilizing both the RR and FCFS queue. And therefore, under the task sets, both RR and MLFQ surpassed the performance of FCFS, and is the preferred scheduler.

6.1.4 Workload 4: Processes with Short Burst Times and High Arrival Rate

In this workload scenario, processes arrive frequently due to a high arrival rate, and each process has a short burst time. The tests below show examples of frequent arrival and also short burst time. Testing and investigating which scheduler operates best with this type of workload.

Average Waiting Time (Processes with short burst time and high arrival rate)				
Arrival Time	Burst Time	FCFS	RR (Time Quantum = 2)	MLFQ (Time Quantum = 2)
[1, 2, 3, 4, 5]	[2, 3, 1, 2, 3]	2.200	2.800	2.800
[1, 3, 5, 7, 9]	[3, 1, 1, 2, 2]	0.200	0.200	0.200
[1 , 1, 2, 2, 3]	[1, 2, 1, 2, 1]	2.000	2.000	2.000

Figure 6.22: Table of average waiting time for processes with short burst time and high arrival rate

Average Turnaround Time (Processes with short burst time and high arrival rate)				
Arrival Time	Burst Time	FCFS	RR (Time Quantum = 2)	MLFQ (Time Quantum = 2)
[1, 2, 3, 4, 5]	[2, 3, 1, 2, 3]	4.400	5.000	5.000
[1, 3, 5, 7, 9]	[3, 1, 1, 2, 2]	2.000	2.000	2.000
[1 , 1, 2, 2, 3]	[1, 2, 1, 2, 1]	3.400	3.400	3.400

Figure 6.23: Table of average turnaround time for processes with short burst time and high arrival rate

Observations on Average Waiting Time and Turnaround Time

First Come First Serve (FCFS), although the table shows that FCFS performed better than RR and MLFQ. In general, FCFS tends to result in higher average waiting times and turnaround times. This is because the processes are executed in order of their arrival time.

Round robin (RR) provides consistent average waiting times and average turnaround times throughout by ensuring fair allocation of time to every process.
Multi-Level Feedback Queue (MLFQ) also demonstrated consistent average waiting times and average turnaround times. This is mainly from the adaptive nature of MLFQ and ability to adjust to different scenarios.

Therefore, to conclude, it can be drawn from the data that both RR and MLFQ are the effective options. When in scenarios where processes have short burst time and high arrival rate, both RR and MLFQ are the optimal schedulers. They provide fairness and the ability to adapt to changing burst times of different processes.

In contrary, it can also be concluded that for processes with short burst time and low arrival rate, FCFS with its simple implementation and straightforward principle, would integrate well and be the optimal choice.

However, if a scenario of processes with long burst time and low arrival rate was to arise, FCFS would still remain as the optimal choice, because the sequential order of execution would potentially lower waiting times.

Conclusion for workload 4

FCFS showcased occasional performance in the task sets, it resulted in higher average waiting times and higher average turnaround times. This can be seen from the FCFS sequential execution nature and characteristics.

On the other hand, RR and MLFQ provided consistent results and balanced average waiting times and average turnaround times across the board. This is attributed to the nature of RR and its ability to content switch according to time quantum’s.

MLFQ demonstrated consistency in results, it showcased its ability to adapt to the different task sets. And therefore, both RR and MLFQ are the effective schedulers to use for processes with short burst time and high arrival rate.

6.2 Influence of Quantum Settings on Average Waiting Time in Round Robin

6.2.1 Fixed Burst Times

A set of workload tasks of 3 with fixed arrival times 0, 2, 4 will be used throughout the test's cases for this topic of study.

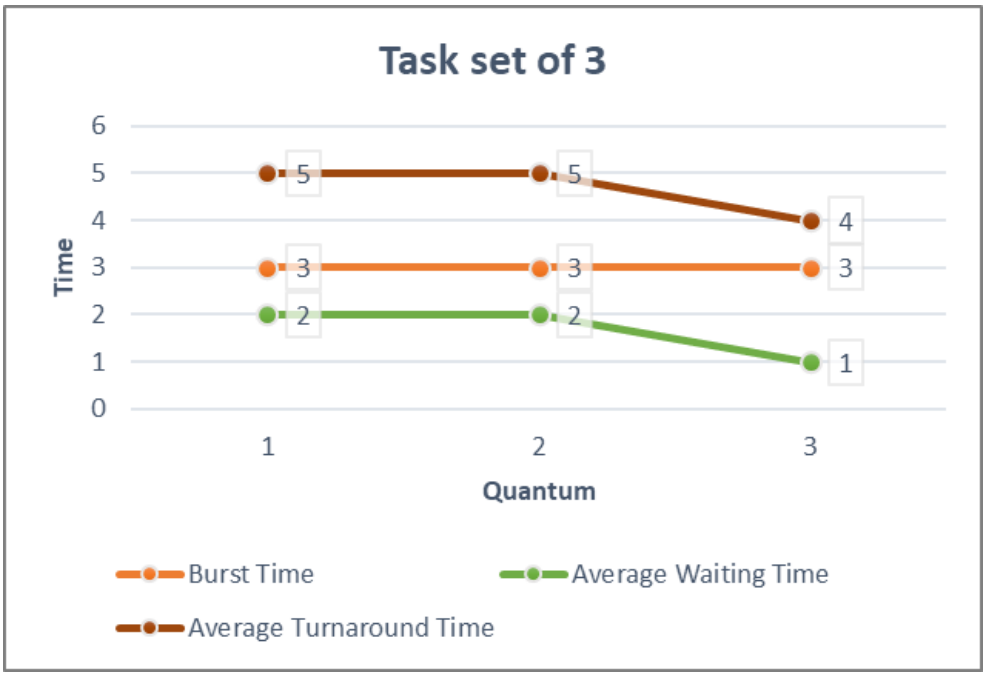


Figure 6.24 Workload Set of 3 with Fixed Burst Time of 3

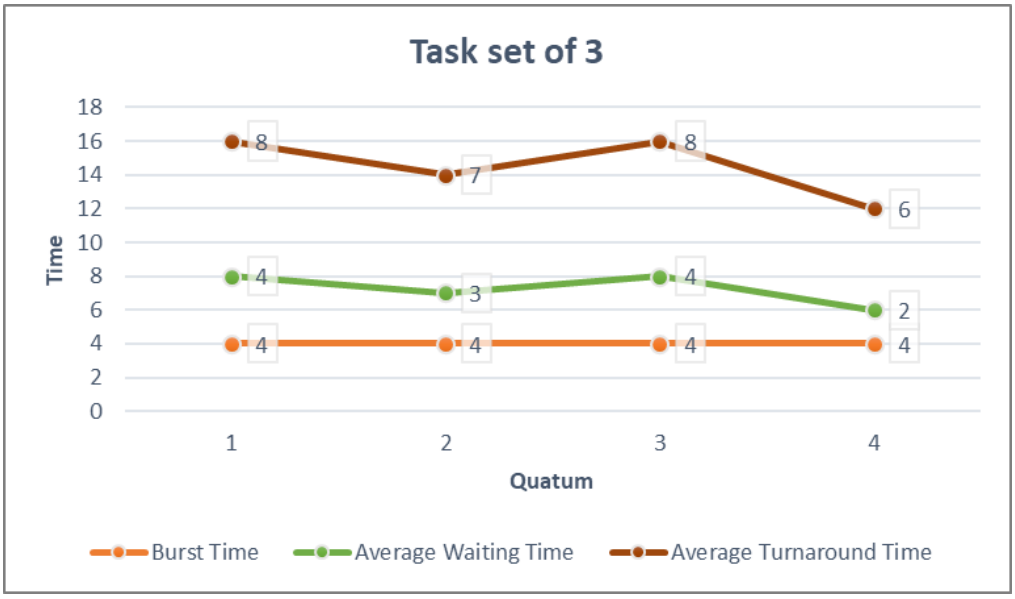


Figure 6.25 Workload Set of 3 with Fixed Burst Time of 4

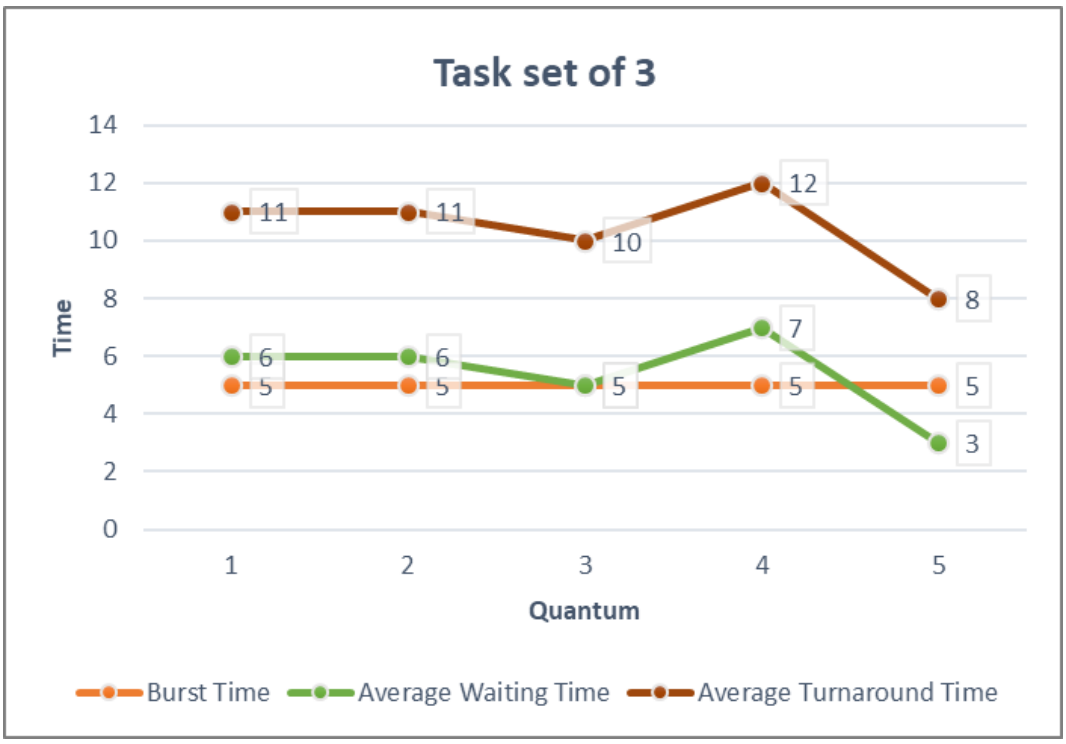


Figure 6.26 Workload Set of 3 with Fixed Burst Time of 5

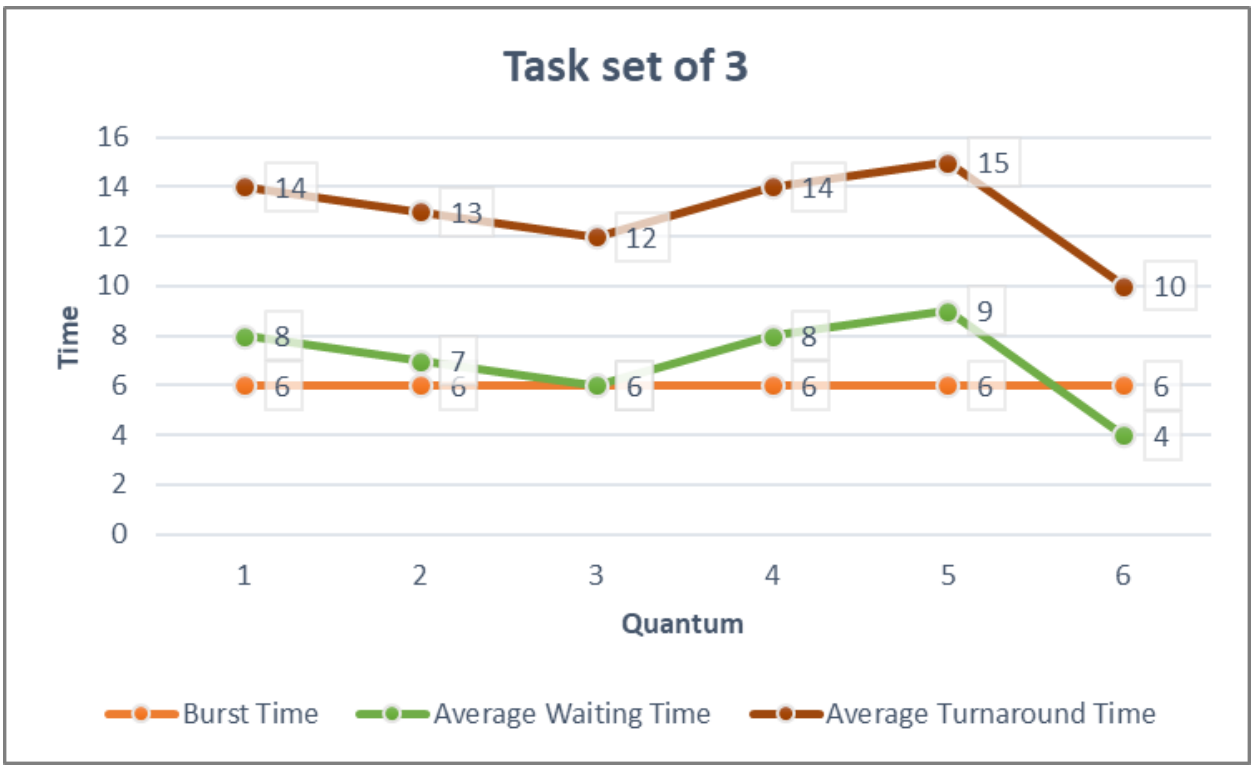


Figure 6.27 Workload Set of 3 with Fixed Burst Time of 6

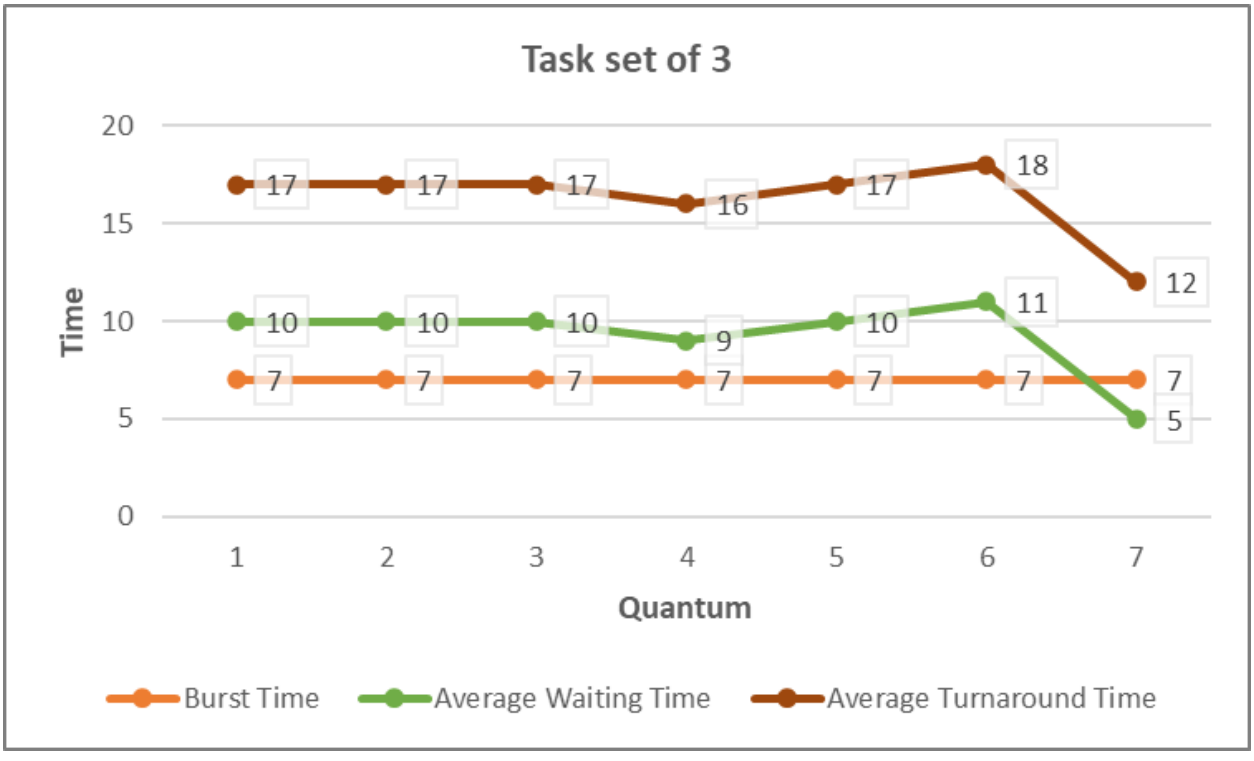


Figure 6.28 Workload Set of 3 with Fixed Burst Time of 7

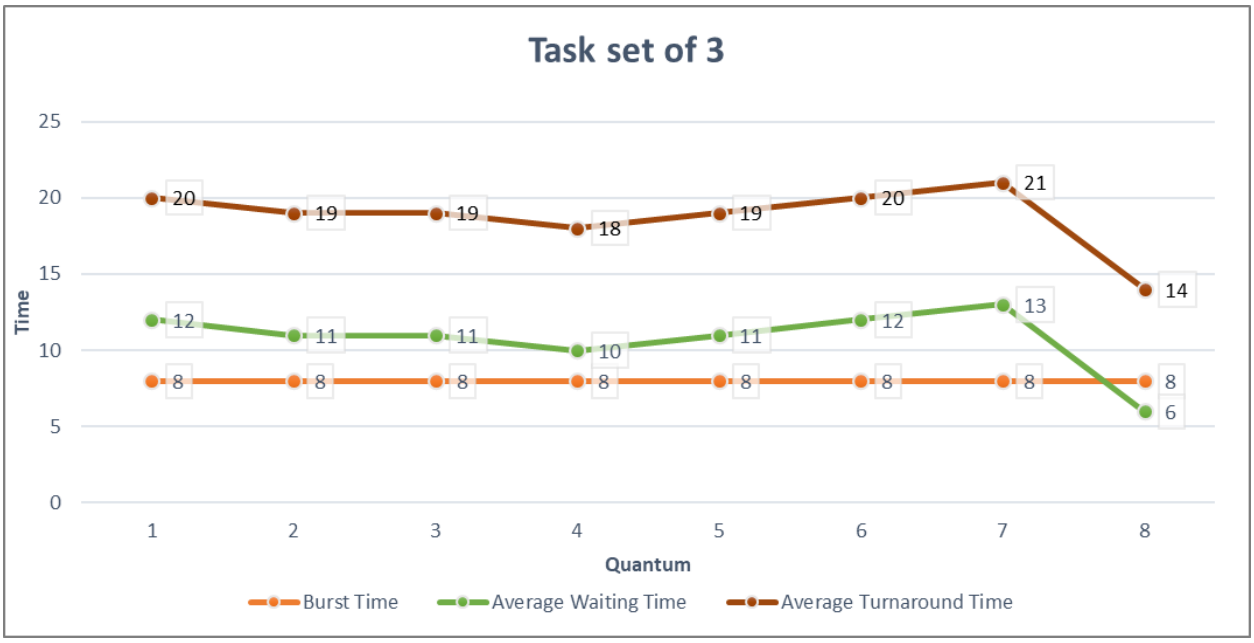


Figure 6.29 Workload Set of 3 with Fixed Burst Time of 8

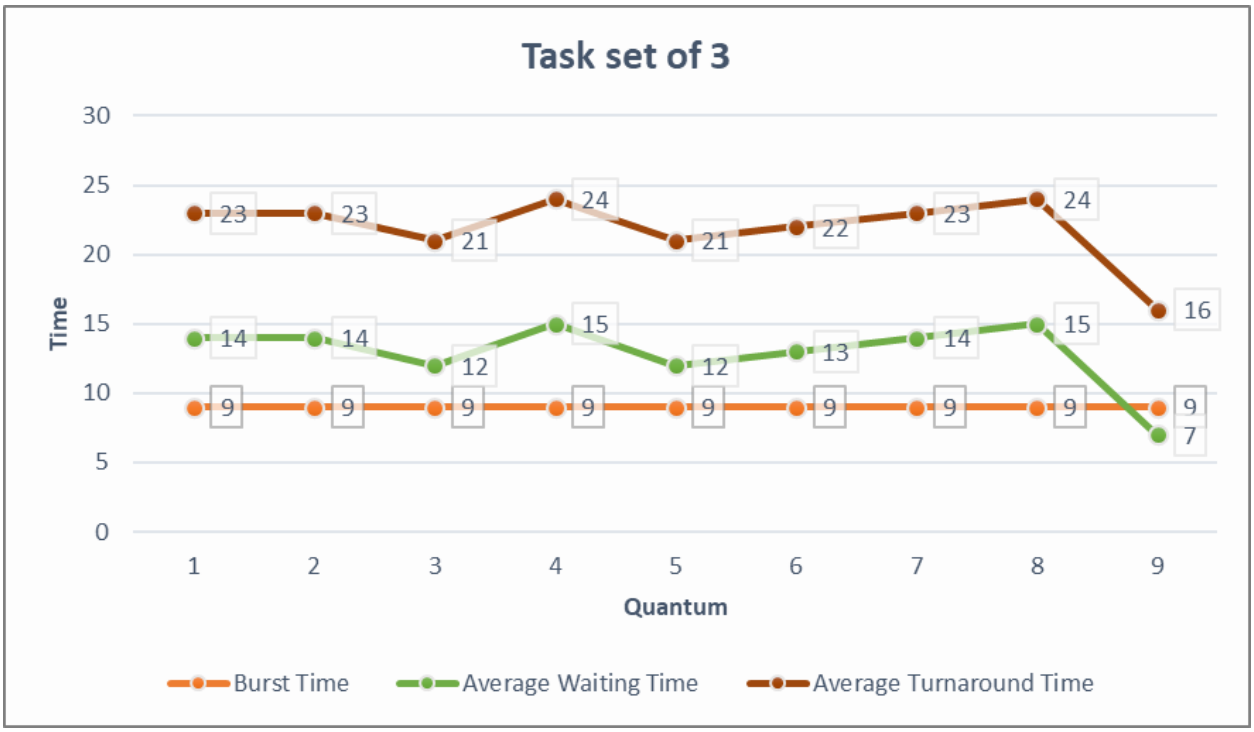


Figure 6.30 Workload Set of 3 with Fixed Burst Time of 9

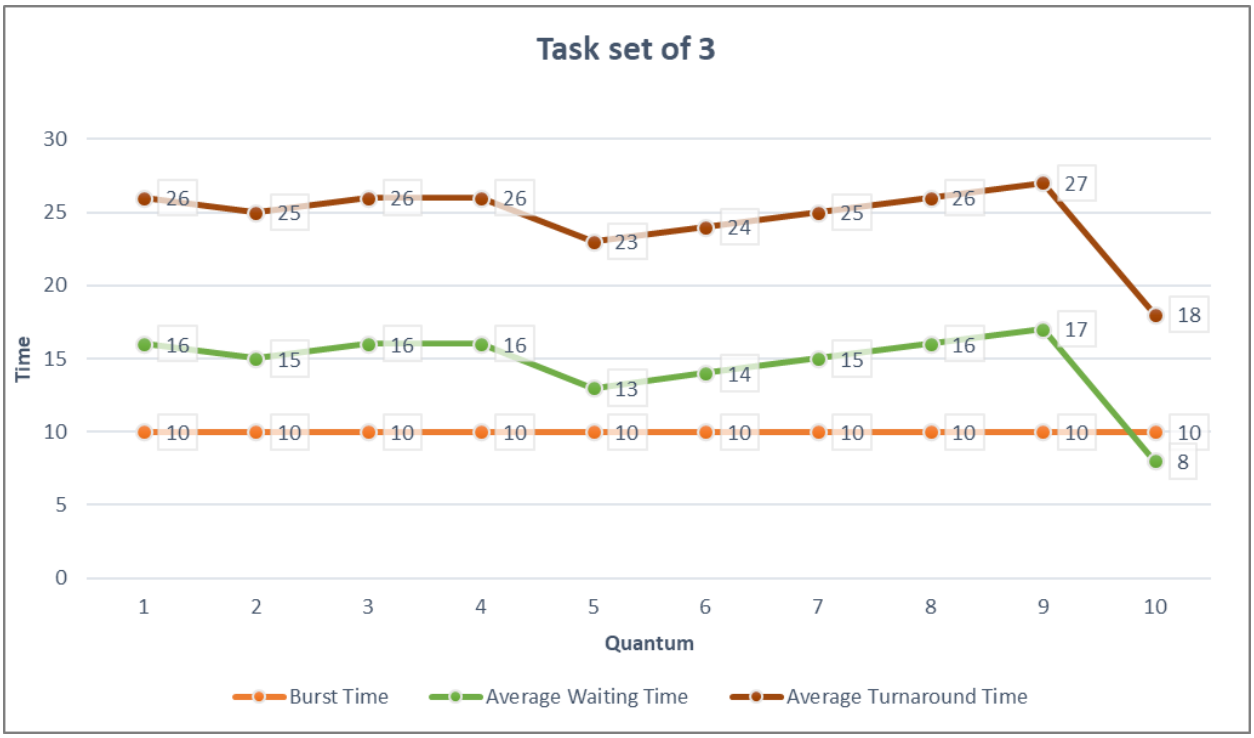


Figure 6.31 Workload Set of 3 with Fixed Burst Time of 10

Based on the chart produced using round robin with a constant workload set of three processes throughout, with an identical burst time for each test set displayed individually by each of the graphs presented in Figures 6.24, 6.25, 6.26, 6.27, 6.28, 6.29, 6.30, 6.31. The quantum times for each of the processes increment in a sequential order until it matches with its respective burst time. This experiment was repeated with different burst times to observe the general patterns from the results of the **average waiting time** and **average turnaround time**.

Results strongly indicate a few key factors in depicting the relationship between the quantum time and both the average waiting time and average turnaround time for various fixed burst times. The graphs reveal the generic pattern where both the average times increase with burst time. However, this trend is not linear and expresses slight individual characteristics for each of average time metrics.

There is a tendency for the initial phases of values for average waiting times and average turnaround times to decrease or stabilize as the quantum time increases before reaching the give-take half burst time as demonstrated by the provided figures. This decrease can be attributed to the

reduction in context switching frequency. On the contrary this would become less pronounced after the half-burst time whereby the average moves in the opposite direction from the initial half. This is introduced when the tasks exceed their quantum time by a rather fractional amount, which requires a slight slice of quantum to complete a small remaining amount of execution, overall leading to the increase of waiting times from subsequent processes. Example taken from Figure (the 4th figure), shows the waiting times averages at 14, 13, 12, and turnaround time averages at 8, 7, 6 in sequential order of quantum. These averages begin a steep upwards of 12, 14, 15 and 6, 8, 9 respectively when the half-burst time equates to the quantum.

Further observations from the generic pattern shown in the graphs clearly indicate that the ceiling value of half the burst time results in the second most efficient quantum time. Whereas given a quantum time that is equivalent to the given burst time, produces the lowest average waiting time and average turnaround time, being the most efficient time slice for any given burst time.

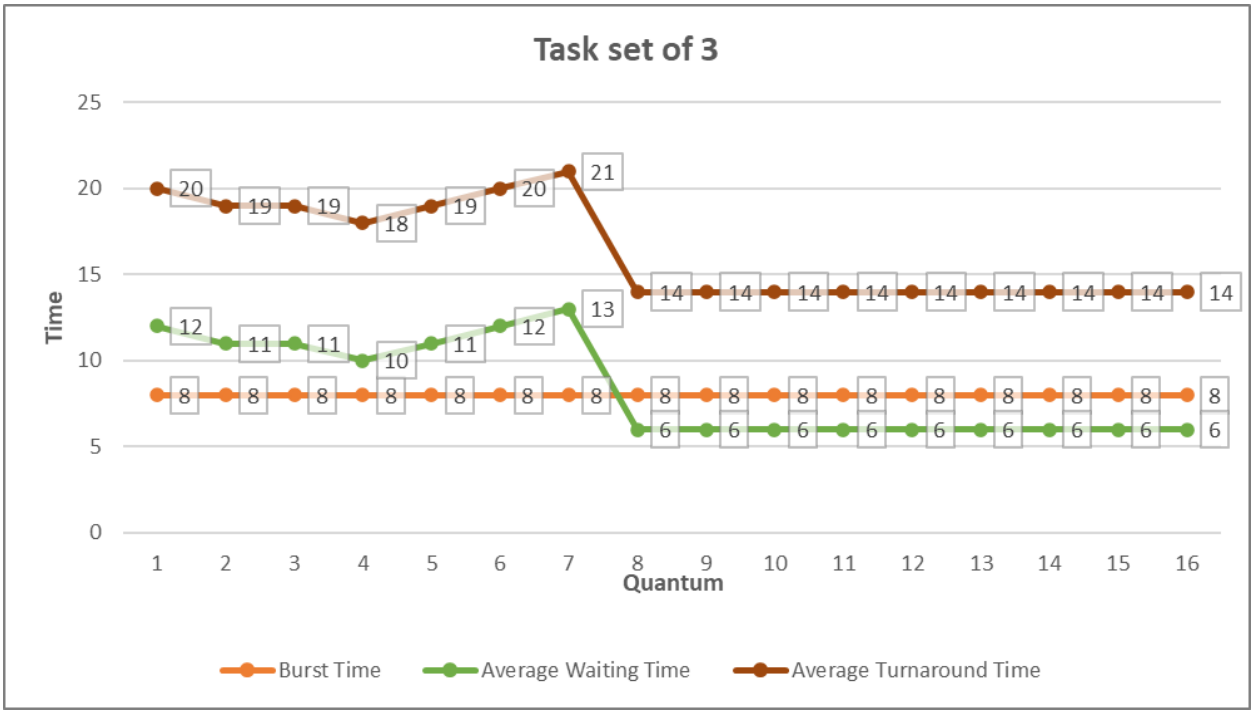


Figure 6.32 Workload Set of 3 with Fixed Burst Time of 8 with Quantum surpassing its equivalent of burst time

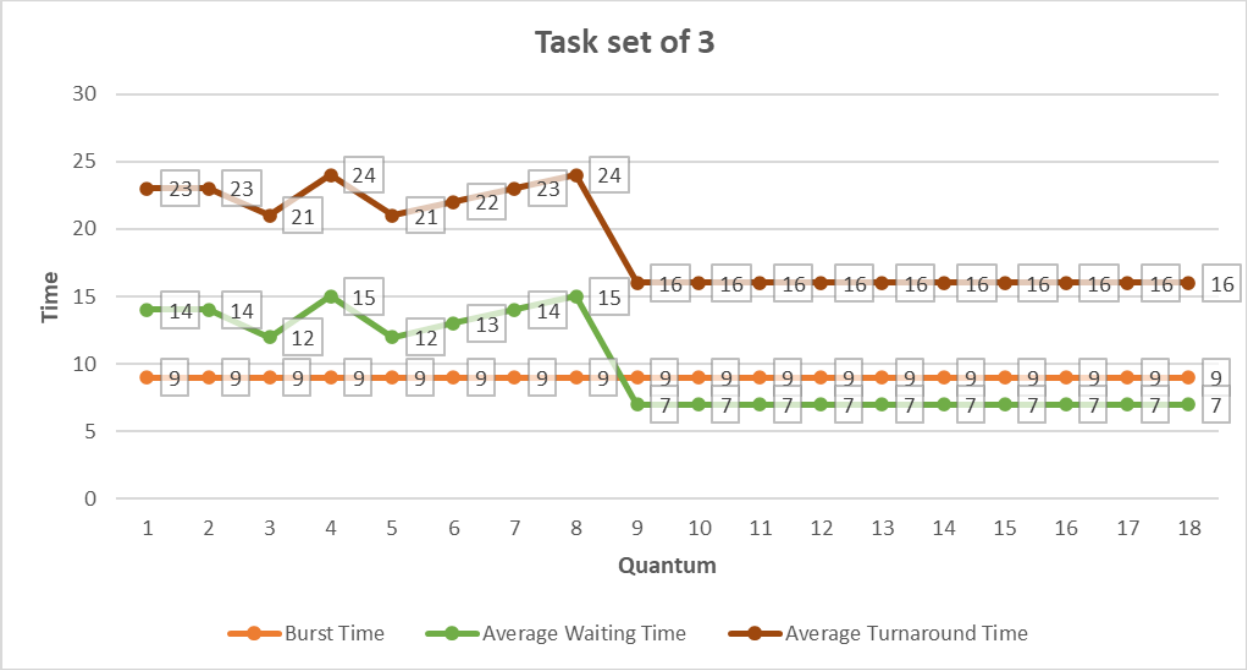


Figure 6.33 Workload Set of 3 with Fixed Burst Time of 9 with Quantum surpassing its equivalent of burst time

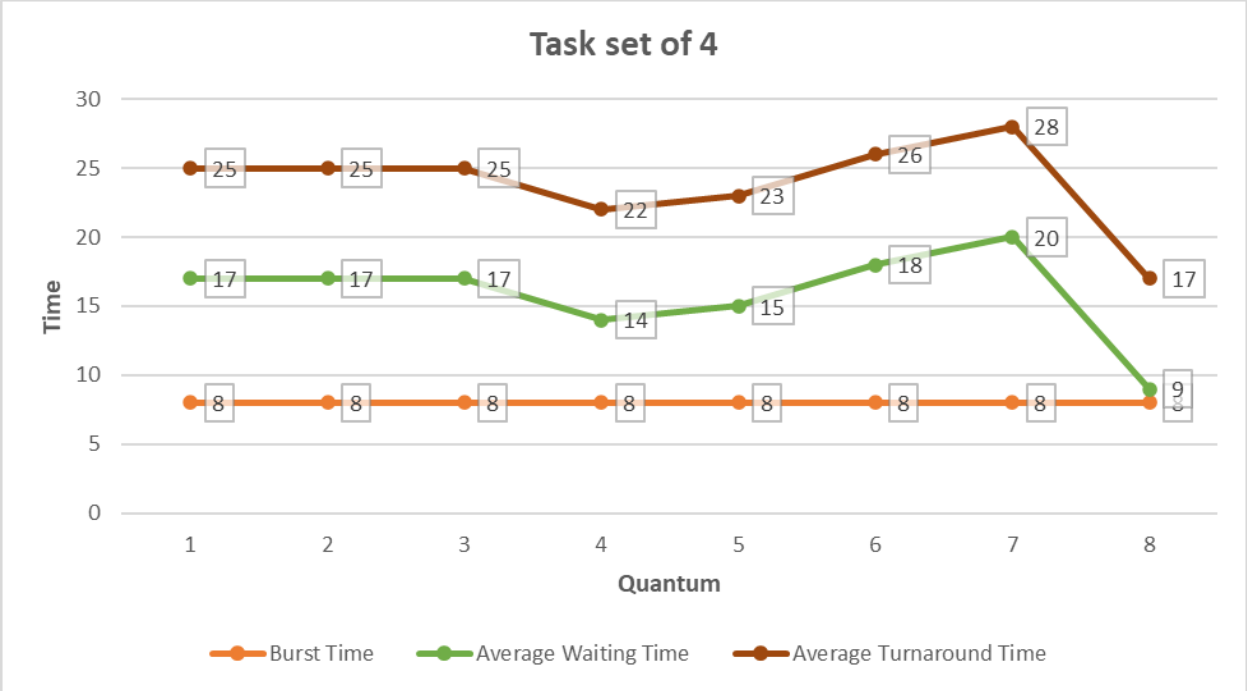


Figure 6.34 Workload Set of 4 with Fixed Burst Time of 8 and Arrival Times of [0, 2, 4, 6]

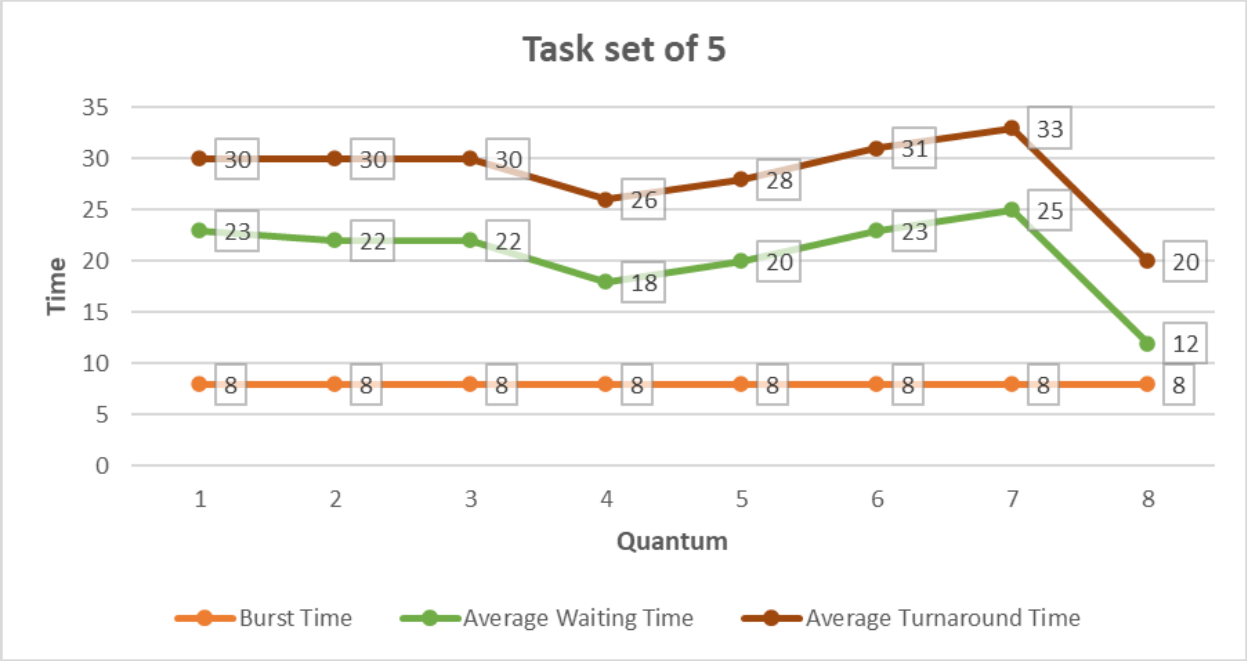


Figure 6.35 Workload Set of 5 with Fixed Burst Time of 8 and Arrival Times of [0, 2, 4, 6]

Figures 6.32 and 6.33 further prove that with the plateau effect seen on the averages for waiting and turnaround times when the number of quantum times surpasses the equivalent of the burst time, indicates a threshold beyond with the increase in the quantum time does not yield any significant improvements in these averages. Instead, the RR scheduling begins to behave similarly to the FCFS algorithm as tasks are likely to complete within a single quantum, consequently diminishing the system's overhead.

In addition to that, the consistency of results across task sets with varying numbers of processes, as seen in Figures 6.34 and 6.35, of 4 and 5 remains firm on previous observation.

6.2.2 Incrementing Burst Times

	Burst Time of Process		
Difference in Burst Time	Process 1	Process 2	Process 3
1	8	9	10
2	8	10	12
3	8	11	14
4	8	12	16
5	8	13	18
6	8	14	20
7	8	15	22
8	8	16	24

Figure 6.36 Workload Sets of 3 with Consistent Incrementing Burst Times

Figure 6.36 presents eight distinct task sets, each comprising three processes with varying burst times. The burst time for each process in each set increases arithmetically, where there is a fixed incremental difference between the burst times of subsequent processes within each set. Specifically, the burst time of Process 1 is consistently set at 8 units, while Process 2 and Process 3 have burst times that progressively increase by a constant difference, creating a uniform series of increments across the task sets. This approach ensures that each set differs by a consistent interval, which allows for systematic analysis for the study and discussion.

	Difference in Burst Time							
Quantum	1	2	3	4	5	6	7	8
1	12.333	13.000	13.667	14.333	15.000	15.667	16.333	17.000
2	12.333	12.667	13.667	14.000	15.000	15.333	16.333	16.667
3	11.333	12.667	13.000	13.333	14.667	15.000	15.333	16.667
4	11.667	12.000	12.333	12.667	14.333	14.667	15.000	15.333
5	11.333	11.667	13.667	14.000	14.333	14.667	15.000	17.000
6	12.333	12.667	13.000	13.333	15.667	16.000	16.333	16.667
7	13.333	13.667	14.000	14.333	14.667	15.000	17.667	18.000
8	9.000	9.333	9.667	10.000	10.333	10.667	11.000	11.333
9	6.333	9.667	10.000	10.333	10.667	11.000	11.333	11.667
10	6.333	6.667	10.333	10.667	11.000	11.333	11.333	12.000
11	6.333	6.667	7.000	11.000	11.333	11.667	12.000	12.333
12	6.333	6.667	7.000	7.333	11.667	12.333	12.333	12.667
13	6.333	6.667	7.000	7.333	7.667	12.333	12.333	13.000
14	6.333	6.667	7.000	7.333	7.667	8.000	13.000	13.333
15	6.333	6.667	7.000	7.333	7.667	8.000	8.333	13.667
16	6.333	6.667	7.000	7.333	7.667	8.000	8.333	8.667
17	6.333	6.667	7.000	7.333	7.667	8.000	8.333	8.667
18	6.333	6.667	7.000	7.333	7.667	8.000	8.333	8.667
19	6.333	6.667	7.000	7.333	7.667	8.000	8.333	8.667
20	6.333	6.667	7.000	7.333	7.667	8.000	8.333	8.667

Figure 6.37 Results for Average Waiting Times for Workload Sets of Incrementing Burst Times and Quantum

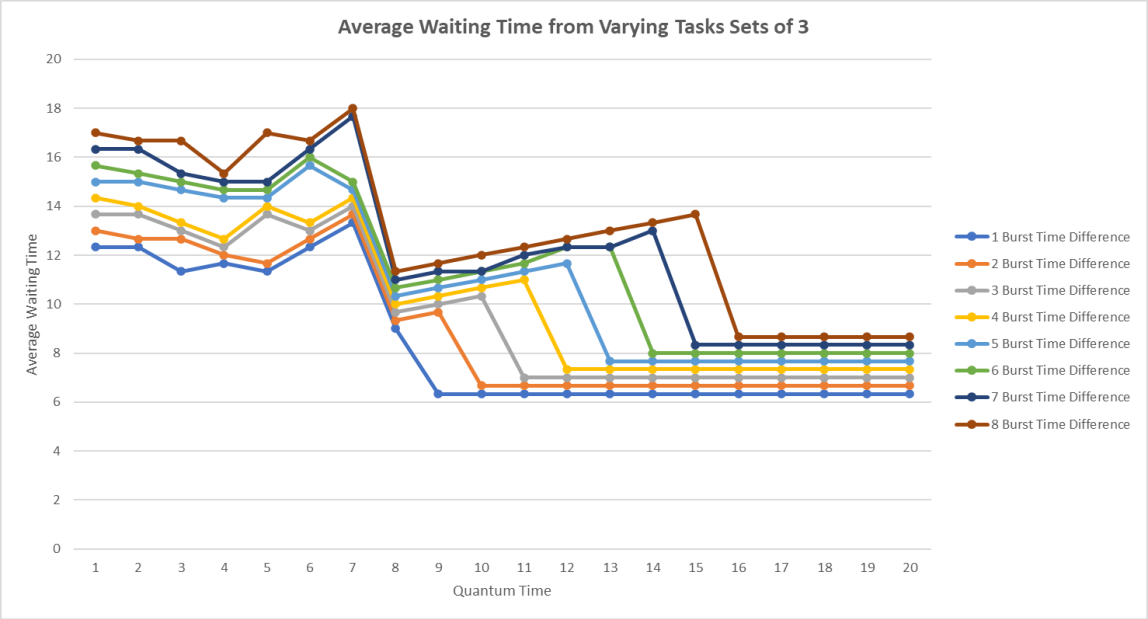


Figure 6.38 Graph for Average Waiting Times for Workload Sets of Incrementing Burst Times and Quantum

	Difference in Burst Time							
Quantum	1	2	3	4	5	6	7	8
1	21.333	23.000	24.667	26.333	28.000	29.667	31.333	33.000
2	21.333	22.667	24.000	26.000	28.000	29.333	31.333	32.667
3	20.333	22.667	23.333	25.333	27.667	29.000	30.333	21.667
4	20.667	22.000	24.667	24.667	27.333	28.667	30.000	31.333
5	20.333	21.667	24.667	26.000	27.333	28.667	30.000	33.000
6	21.333	22.667	24.000	25.333	28.667	30.000	31.333	32.667
7	23.333	23.667	25.000	26.333	27.667	29.000	32.667	34.000
8	18.000	19.333	20.667	22.000	23.333	24.667	26.000	27.333
9	15.333	19.667	21.000	22.333	23.667	25.000	26.333	27.667
10	15.333	16.667	21.333	22.667	24.000	25.333	26.333	28.000
11	15.333	16.667	18.000	23.000	24.333	25.667	27.000	28.333
12	15.333	16.667	18.000	19.333	24.667	26.000	27.333	28.667
13	15.333	16.667	18.000	19.333	20.667	26.333	27.333	29.000
14	15.333	16.667	18.000	19.333	20.667	22.000	28.000	29.333
15	15.333	16.667	18.000	19.333	20.667	22.000	23.333	29.667
16	15.333	16.667	18.000	19.333	20.667	22.000	23.333	24.667
17	15.333	16.667	18.000	19.333	20.667	22.000	23.333	24.667
18	15.333	16.667	18.000	19.333	20.667	22.000	23.333	24.667
19	15.333	16.667	18.000	19.333	20.667	22.000	23.333	24.667
20	15.333	16.667	18.000	19.333	20.667	22.000	23.333	24.667

Figure 6.39 Graph for Average Turnaround Times for Workload Sets of Incrementing Burst Times and Quantum

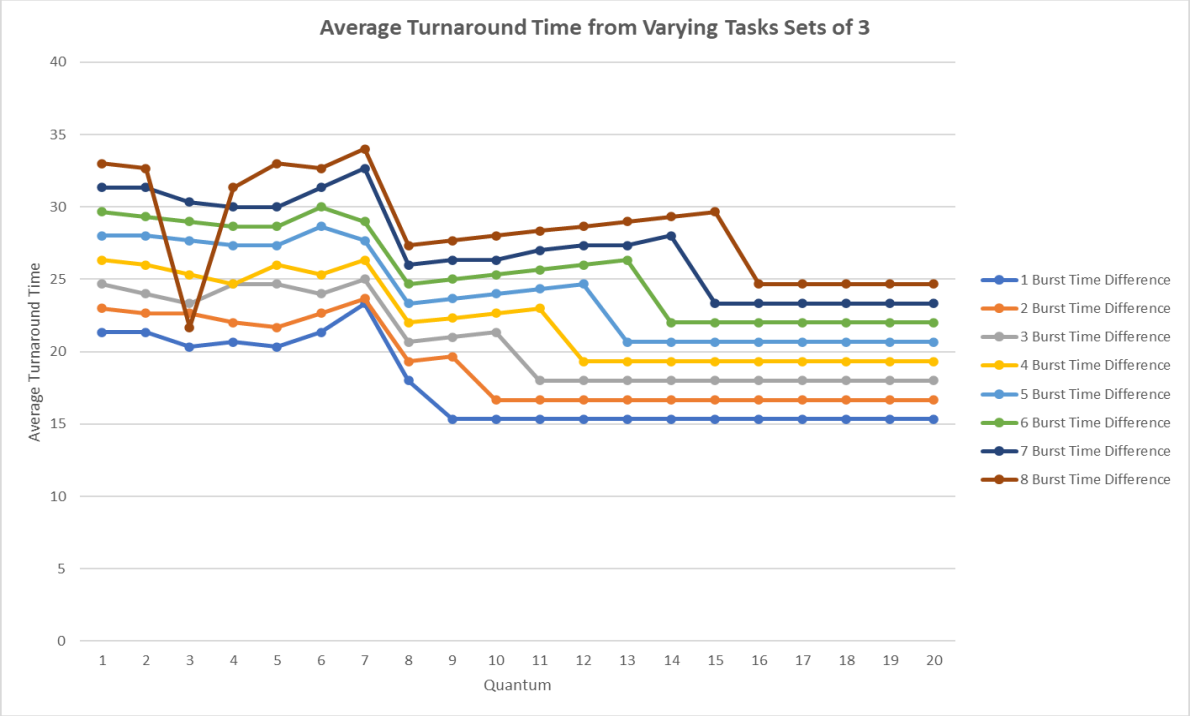


Figure 6.40 Graph for Average Turnaround Times for Workload Sets of Incrementing Burst Times and Quantum

Figure 6.37, 6.38, 6.39, 6.40 above illustrate the final outcomes after conducting a total of 8 different workload sets given in Figure 6.36 with incrementing burst times over the span of 20-time quanta values.

Difference in Burst Time	Average Waiting Time	Average Turnaround Time
1	6.333	15.333
2	6.667	16.667
3	7.000	18.000
4	7.333	19.333
5	7.667	20.667
6	8.000	22.000
7	8.333	23.333
8	8.667	24.667

Figure 6.41 Processing Results for the workload tasks with First Come First Served Algorithm

The cells highlighted in green within the tables in Figure 6.37 and 6.39 shows that both the averages of waiting and turnaround times begin to exhibit convergence towards the results characteristic of the First Come, First Served algorithm as when the same workload is being carried

out in the FCFS algorithm with results displayed in Figure 6.41. This was particularly evident when the quantum time allocated equaled or exceeded the given burst time for the second task from the last. The table above illustrates the results yielded from the First Come First Served algorithm indeed matches the values in the cells that were marked in green.

	Burst Times					
No of Tasks	Process 1	Process 2	Process 3	Process 4	Process 5	Process 6
4	8	10	12	14	-	-
5	8	10	12	14	16	-
6	8	10	12	14	16	18

Figure 6.42 Workload Sets of 4, 5, and 6 with Burst Time Difference of 2

	Burst Times					
No of Tasks	Process 1	Process 2	Process 3	Process 4	Process 5	Process 6
4	8	11	14	17	-	-
5	8	11	14	17	20	-
6	8	11	14	17	20	23

Figure 6.43 Workload Sets of 4, 5, and 6 with Burst Time Difference of 3

The tables presented in Figures 6.42 and 6.43 are prepared task sets for an investigation into the hypothesis that when the quantum time is equal to or greater than the burst time of the second last task in any given set, the resulting average waiting and turnaround times will align with those when generated by the First Come, First Served (FCFS) algorithm.

	Quantum																			
No of Tasks	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
4	21.250	20.500	20.750	19.000	18.500	20.000	21.500	17.000	17.750	13.500	13.750	11.000	11.000	11.000	11.000	11.000	11.000	11.000	11.000	11.000
5	30.800	29.600	29.800	28.000	27.000	28.000	28.600	25.600	26.800	22.000	22.600	18.400	18.600	16.000	16.000	16.000	16.000	16.000	16.000	16.000
6	41.667	40	40.167	37.667	37.5	37.667	38	38	36.667	31.667	32.667	27.667	28.167	24	24.167	21.667	21.667	21.667	21.667	21.667

Figure 6.44 Average Waiting Time Results for Datasets of Figure 6.42

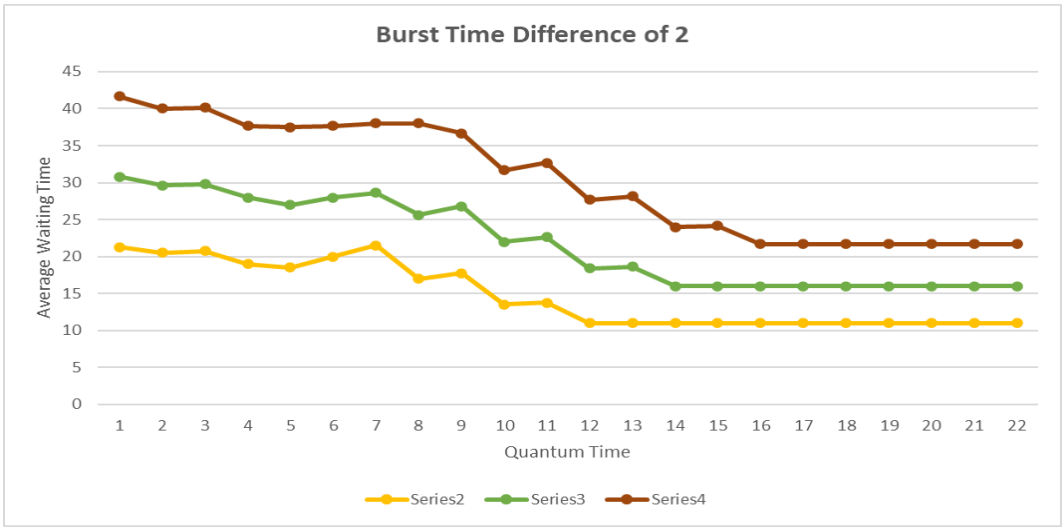


Figure 6.45 Average Waiting Time Graph Results for Datasets of Figure 6.42

	Quantum																					
No of Tasks	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
4	32.250	31.500	31.750	30.000	29.500	31.000	32.500	28.000	28.750	24.500	24.750	22.000	22.000	22.000	22.000	22.000	22.000	22.000	22.000	22.000	22.000	22.000
5	42.800	41.600	41.800	40.000	39.000	40.000	40.600	37.600	38.800	34.000	34.600	30.400	30.600	28.000	28.000	28.000	28.000	28.000	28.000	28.000	28.000	28.000
6	54.667	53.000	43.167	50.667	50.500	50.667	51.000	48.000	49.667	44.667	45.667	40.667	41.667	37.000	37.167	34.667	34.667	34.667	34.667	34.667	34.667	34.667

Figure 6.46 Average Turnaround Time Results for Datasets of Figure 6.42

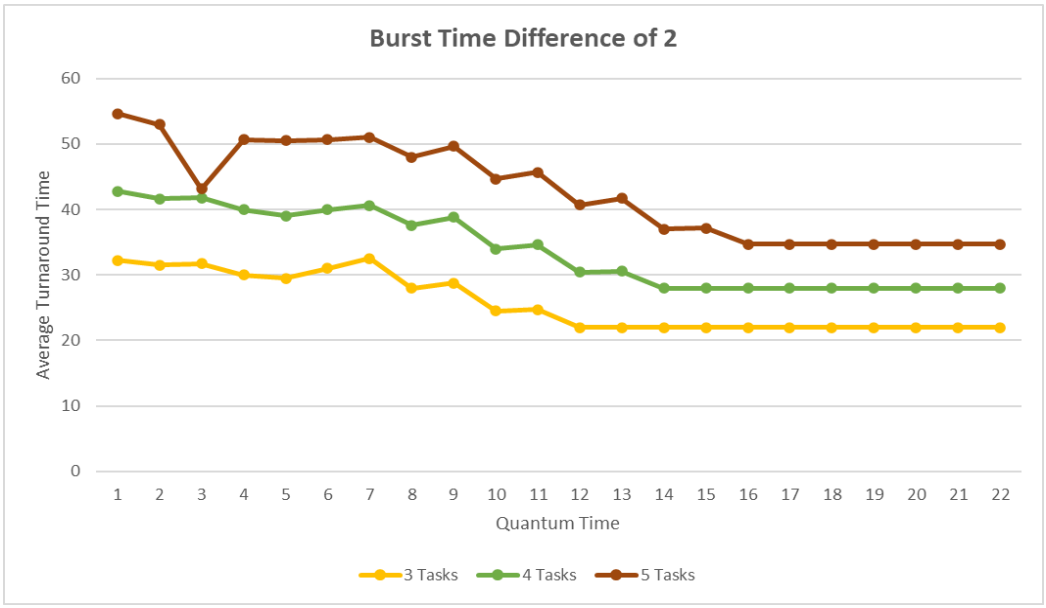


Figure 6.47 Average Turnaround Time Graph Results for Datasets of Figure 6.42

	Quantum																					
No of Tasks	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
4	23.250	23.000	22.500	21.000	22.000	22.500	22.500	18.000	18.750	19.500	14.750	15.000	15.250	12.000	12.000	12.000	12.000	12.000	12.000	12.000	12.000	12.000
5	34.800	34.400	33.600	32.400	33.000	32.400	32.000	29.200	28.800	30.000	24.600	25.200	25.800	20.800	21.000	21.200	18.000	18.000	18.000	18.000	18.000	18.000
6	48.333	47.667	46.500	45.000	45.833	45.000	43.667	42.333	41.500	41.667	36.000	37.000	38.000	32.000	32.500	33.000	27.833	28.000	28.167	25.000	25.000	25.000

Figure 6.48 Average Waiting Time Results for Datasets of Figure 6.43

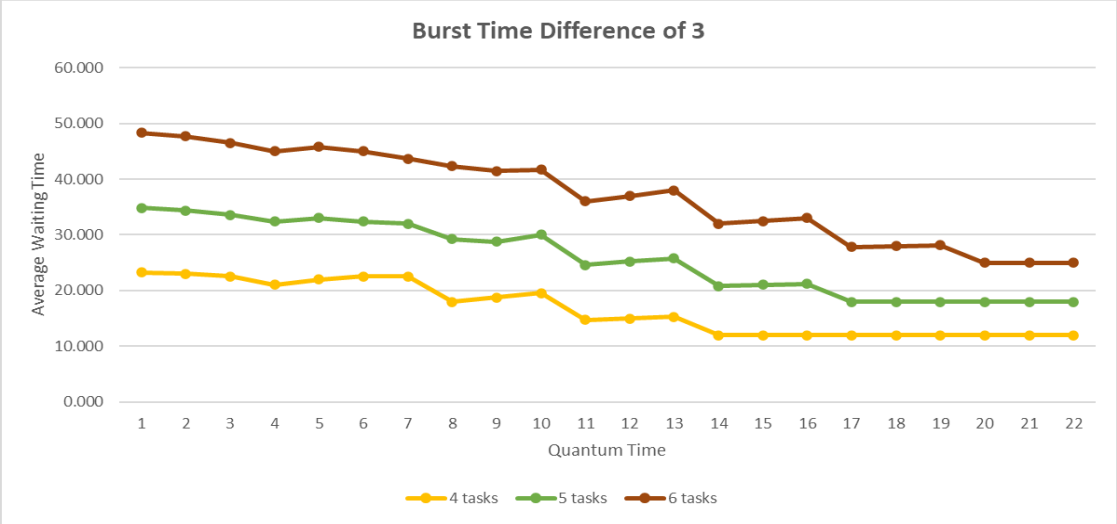


Figure 6.49 Average Waiting Time Graph Results for Datasets of Figure 6.43

	Quantum																					
No of Tasks	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
4	35.750	35.500	35.000	33.500	34.500	35.000	35.000	30.500	31.250	32.000	24.750	27.500	27.750	24.500	24.500	24.500	24.500	24.500	24.500	24.500	24.500	24.500
5	48.800	48.400	47.600	46.400	47.000	46.400	46.000	43.200	42.800	44.000	38.600	39.200	39.800	34.800	35.000	35.200	32.000	32.000	32.000	32.000	32.000	32.000
6	63.833	63.167	62.000	60.500	61.333	60.500	59.167	57.833	57.000	57.167	51.500	52.500	53.500	47.500	48.000	48.500	43.333	43.500	43.667	40.500	40.500	40.500

Figure 6.50 Average Turnaround Time Results for Datasets of Figure 6.43

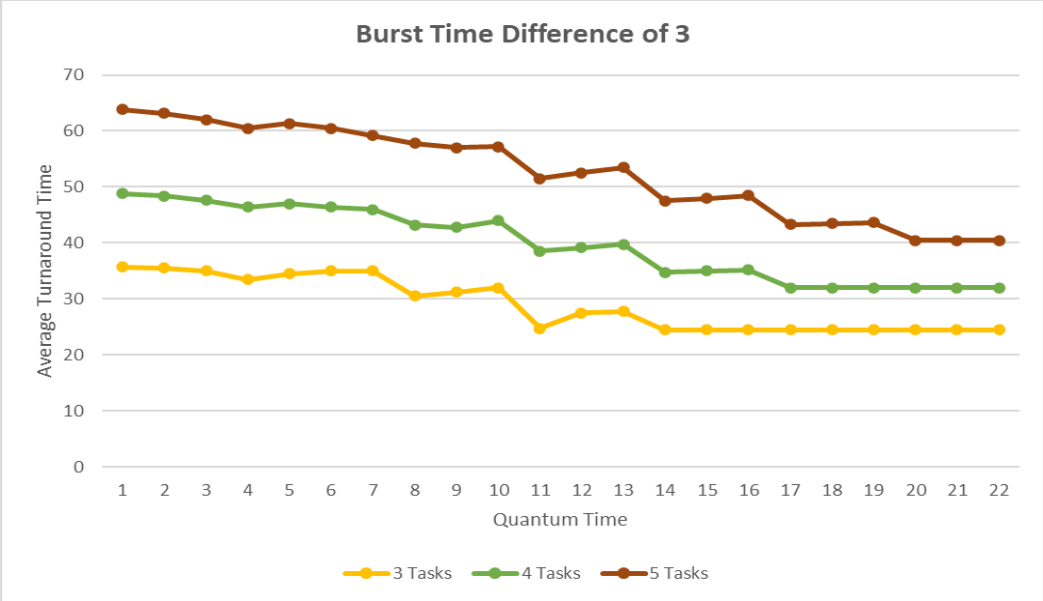


Figure 6.51 Average Turnaround Time Graph Results for Datasets of Figure 6.43

The Figures 6.44 to 6.51 were further used to justify that the average waiting time and average turnaround are equivalent to the result outcomes produced using the First Come First Served

algorithm. Further workload sets of 4, 5, and 6 tasks for each burst time difference of 2 and 3 respectively were used to give grounds showing that whenever the quantum time slice begins to be equivalent to the second last task onwards mirrors the same averages for both waiting and turnaround times as produced by First Come, First Served algorithm. Specifically, from the process workload set of 6 tasks with the burst time difference of 3, when the quantum is set to 20, any processing beginning from the 5th task appears to have the averages for the waiting and turnaround stabilized.

With the explanations above, it is strongly recommended that the First Come, First Served scheduling algorithm be used as it is much more optimal in performance since it involves much less context switching compared to when given a fixed time slice allocated for each process execution in round robin. Moreover, all the lowest average waiting and turnaround times are shown to be the result whenever the time slice allocated is larger than the second last task of any given workload.

6.2.3 Best Quantum for Lowest Averages Waiting and Turnaround Times

	Difference in Burst Time													
Quantum	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	12.333	13.000	13.667	14.333	15.000	15.667	16.333	17.000	17.667	18.333	19.000	19.667	20.333	21.000
2	12.333	12.667	13.667	14.000	15.000	15.333	16.333	16.667	17.667	18.000	19.000	19.333	20.333	20.667
3	11.333	12.667	13.000	13.333	14.667	15.000	15.333	16.667	17.000	17.333	18.667	19.000	19.333	20.667
4	11.667	12.000	12.333	12.667	14.333	14.667	15.000	15.333	17.000	17.333	17.667	18.000	19.667	20.000
5	11.333	11.667	13.667	14.000	14.333	14.667	15.000	17.000	17.333	17.667	18.000	18.333	20.333	20.667
6	12.333	12.667	13.000	13.333	15.667	16.000	16.333	16.667	17.000	17.333	19.667	20.000	20.333	20.667
7	13.333	13.667	14.000	14.333	14.667	15.000	17.667	18.000	18.333	18.667	19.000	19.333	19.667	22.333
8	9.000	9.333	9.667	10.000	10.333	10.667	11.000	11.333	14.333	14.667	15.000	15.333	15.667	16.000
9	6.333	9.667	10.000	10.333	10.667	11.000	11.333	11.667	12.000	12.333	15.667	16.000	16.333	16.667
10	6.333	6.667	10.333	10.667	11.000	11.333	11.333	12.000	12.333	12.667	13.000	13.333	17.000	17.333
11	6.333	6.667	7.000	11.000	11.333	11.667	12.000	12.333	12.667	13.000	13.333	13.667	14.000	14.333
12	6.333	6.667	7.000	7.333	11.667	12.333	12.333	12.667	13.000	13.333	13.667	14.000	14.333	14.667
13	6.333	6.667	7.000	7.333	7.667	12.333	12.333	13.000	13.333	13.667	14.000	14.333	14.667	15.000
14	6.333	6.667	7.000	7.333	7.667	8.000	13.000	13.333	13.667	14.000	14.333	14.667	15.000	15.333
15	6.333	6.667	7.000	7.333	7.667	8.000	8.333	13.667	14.000	14.333	14.667	15.000	15.333	15.667
16	6.333	6.667	7.000	7.333	7.667	8.000	8.333	8.667	14.333	14.667	15.000	15.333	15.667	16.000
17	6.333	6.667	7.000	7.333	7.667	8.000	8.333	8.667	9.000	15.000	15.333	15.667	16.000	16.333
18	6.333	6.667	7.000	7.333	7.667	8.000	8.333	8.667	9.000	9.333	15.667	16.000	16.333	16.667
19	6.333	6.667	7.000	7.333	7.667	8.000	8.333	8.667	9.000	9.333	9.667	16.333	16.667	17.000
20	6.333	6.667	7.000	7.333	7.667	8.000	8.333	8.667	9.000	9.333	9.667	10.000	17.000	17.333

Figure 6.52 Extended Average Waiting Time for Datasets from Figure 6.36

	Difference in Burst Time													
Quantum	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	21.333	23.000	24.667	26.333	28.000	29.667	31.333	33.000	34.667	36.333	38.000	39.667	40.333	43.000
2	21.333	22.667	24.000	26.000	28.000	29.333	31.333	32.667	34.667	36.000	38.000	39.333	40.333	42.667
3	20.333	22.667	23.333	25.333	27.667	29.000	30.333	21.667	34.000	35.333	37.667	39.000	40.333	42.667
4	20.667	22.000	24.667	24.667	27.333	28.667	30.000	31.333	34.000	35.333	36.667	38.000	40.667	42.000
5	20.333	21.667	24.667	26.000	27.333	28.667	30.000	33.000	34.333	35.667	37.000	38.333	41.333	42.667
6	21.333	22.667	24.000	25.333	28.667	30.000	31.333	32.667	34.000	35.333	38.667	40.000	41.333	42.667
7	23.333	23.667	25.000	26.333	27.667	29.000	32.667	34.000	25.333	36.667	38.000	39.333	40.667	44.333
8	18.000	19.333	20.667	22.000	23.333	24.667	26.000	27.333	31.333	32.667	34.000	35.333	36.667	38.000
9	15.333	19.667	21.000	22.333	23.667	25.000	26.333	27.667	29.000	30.333	34.667	36.000	37.333	38.667
10	15.333	16.667	21.333	22.667	24.000	25.333	26.333	28.000	29.333	30.667	32.000	33.333	38.000	39.333
11	15.333	16.667	18.000	23.000	24.333	25.667	27.000	28.333	29.667	31.000	32.333	33.667	35.000	36.333
12	15.333	16.667	18.000	19.333	24.667	26.000	27.333	28.667	30.000	31.333	32.667	34.000	35.333	36.667
13	15.333	16.667	18.000	19.333	20.667	26.333	27.333	29.000	30.333	31.667	33.000	34.333	35.667	37.000
14	15.333	16.667	18.000	19.333	20.667	22.000	28.000	29.333	30.667	32.000	33.333	34.667	36.000	37.333
15	15.333	16.667	18.000	19.333	20.667	22.000	23.333	29.667	31.000	32.333	33.667	35.000	36.333	37.667
16	15.333	16.667	18.000	19.333	20.667	22.000	23.333	24.667	31.333	32.667	34.000	35.333	36.667	38.000
17	15.333	16.667	18.000	19.333	20.667	22.000	23.333	24.667	26.000	33.000	34.333	35.667	37.000	38.333
18	15.333	16.667	18.000	19.333	20.667	22.000	23.333	24.667	26.000	27.333	34.667	36.000	37.333	38.667
19	15.333	16.667	18.000	19.333	20.667	22.000	23.333	24.667	26.000	27.333	28.667	36.333	37.667	39.000
20	15.333	16.667	18.000	19.333	20.667	22.000	23.333	24.667	26.000	27.333	28.667	30.000	38.000	39.333

Figure 6.53 Extended Average Turnaround Time for Datasets from Figure 6.36

Figures 6.52 and 6.53 utilize the same process sets that were used in Figures 6.37 and 6.39 extended from 8 to 14 differences in burst time increment to further establish the discovery of this topic.

In the previous event of discovering the effect of incrementing burst times has on different quantum settings, it was observed that the quantum times corresponding to the green marked cells has produced averages of waiting and turnaround times that are conforming to those generated by the First Come First Served algorithm. In search of the best quantum, these values are therefore rather extrinsic to the objective and have been excluded from the study, as they do not enhance the efficiency of the Round Robin algorithm when compared to First Come First Served. The next optimal quantum times are the cells highlighted in blue which as signified in the tables respectively, the time slices which are associated with the minimal average waiting time and average turnaround time.

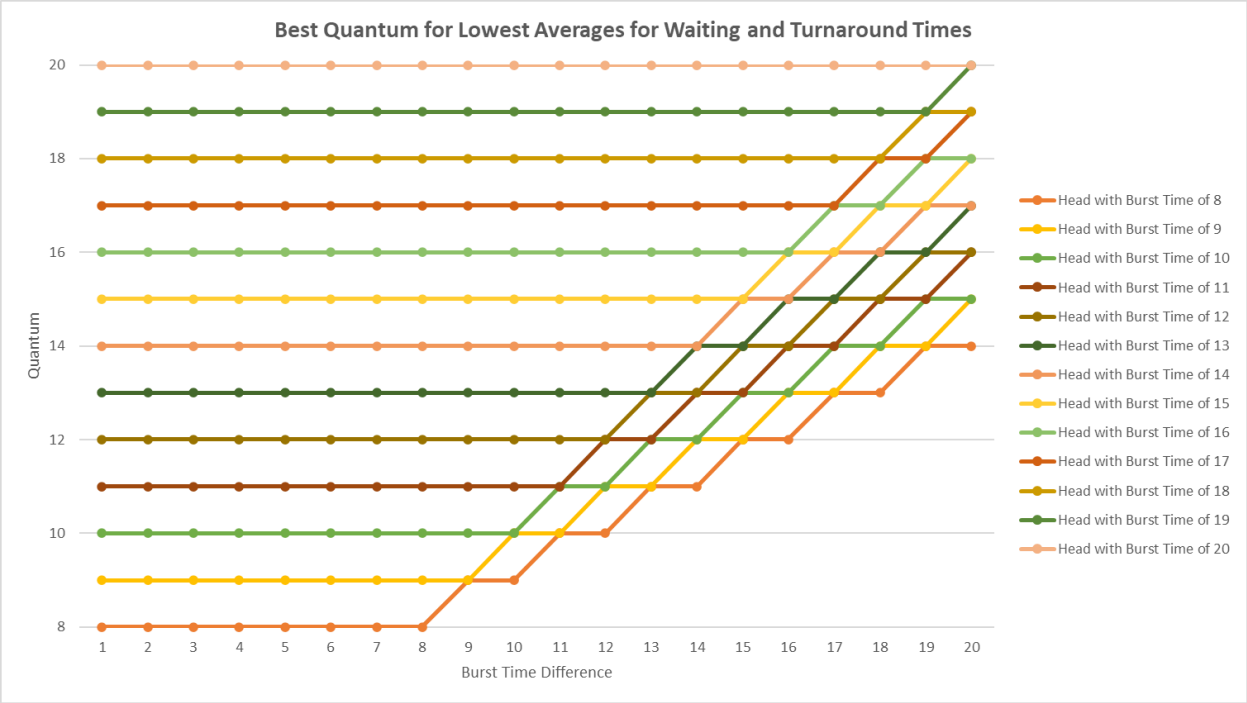


Figure 6.53 Best Quantum for Lowest Averages for Waiting and Turnaround Times

Finding the best setting for the quantum time utilizing a range of processes sets with different head burst time helps to make sure that the chosen quantum works well for both quick and shorter tasks.

The results as depicted in the graph in Figure 6.54, indicate a trend where the optimal quantum increases with the increase in burst time difference. As taken from the graph, it demonstrates that the optimal quantum time is consistently at a time lower than or the equivalent to its identical difference in burst time (as seen with the plateau line at the bottom of the graph). A positive correlation is shown between the difference in burst time and the optimal quantum time, whereby as the difference in burst time increases, the optimal quantum time also increases. For example, in the series of the head process with a burst time of 8, when the interval in the burst time is 1 unit, the optimal quantum is shown to be 8 milliseconds. However, as the burst time difference extends to 20 units, the optimal quantum time ascends to 14 milliseconds. Moreover, the examination on the series of the head process with the burst time of 15 displays a more distinct increase in its optimal quantum time with the increase in burst time difference. It begins with a one-to-one proportionate optimal quantum-to-burst-time-difference of 15 to 15, but then steeps more and rises to a 18 quantum time for a 20 unit burst time. This climb indicates that tasks with longer initial

burst times benefit from a comparatively larger quantum time to adequate and optimize the trade-off which occurs between throughput and the system interactivity.

Given the situation of Head with Burst time of 10 from the graph, it indicates that setting the quantum to 10 units produces the best balance between minimizing context switches and keeping the wait and turnaround times low inclusive of having the difference between each burst time to be either equal or lesser than 10. This is due to when the quantum time is set to match burst time of 10 or lower, it allows the process to run to completion in under one time slice without being preempted. This alignment avoids the need for the process to re-enter the queue, eliminating any overhead that is associated with context switching.

Given a sample workload set of burst times 8, 16, 24 milliseconds. The optimal quantum time would be 8 milliseconds where each of the processes are multiples of the quantum time where no underutilization and no context switching.

Another sample workload set of burst times 8, 19, 30, milliseconds have the quantum time of 10 milliseconds as its optimal. Its efficiency is demonstrated by the completion steps of the processes. The first process completes in less than one quantum, remaining on 2 units of unutilized quantum time. While the second process requires two quanta to complete creating a single context switch with a remainder of 1 unit of idle time in the second quantum. The third process requires three quanta and creates two context switches. This quantum time results in 3 context switches and 6 quanta and minimal CPU underutilization. Compared to a quantum time of 12 milliseconds, where the first task completes with a remainder of 4 units, second task completes within two quanta with a remainder of 5 milliseconds and third process completes with 3 quanta but with the remainder of 6 milliseconds. Although the context switch amounts are the same, the underutilization is substantially higher than that of 10 milliseconds quantum time.

6.3 Influence of Quantum Settings on System Interactivity in Round Robin

Selection of the quantum time slice in the Round Robin scheduling algorithm determines the amount of times context switching performs during the execution of any given workload, which is part of determining the average waiting time.

Opting for shorter quantum time often enhances the system's interactive execution on tasks. This is due to enabling a set expiration timeline for each of the task to get processed before having the system to transition to other processes. This prevents a single process from taking the entire CPU allocated resources. However, constant switching of context may introduce considerable overhead especially when the given burst times are of great length.

Longer quantum time counters the drawbacks of shorter quantum times. Likewise, as the quantum time begins to match the burst times of the processes, the effect of non-preemption takes place as the processing time for each of the processes lengthens causing the other upcoming processes to wait in the queue for longer times.

Utilizing a well-balanced quantum time which provides the right support for minimizing the systems process transitions while still providing a satisfactory response time for interactive processes will ensure the interactivity of the system. This can be done by setting a benchmark for both context switching and response time.

6.4 Tuning the Multi-Level Feedback Queue Scheduler

6.4.1 Integrating Intermediary Queues

The Multi-Level Feedback Queue currently utilizes only two queues namely the Round Robin as the high priority queue and the First Come, First Served as the lower priority queue. With only 2 queues, the scheduler can be further fine-tuned, allowing for more enhanced performance that can be adjusted based on the specific needs of the system and the characteristics of the workload sets.

In a Multi-Level Feedback Queue system, the effective distribution of tasks across multiple priority levels is important to maintain efficiency. This can be done especially by introducing additional Round Robin queues. Expanding the scheduler to include more of these queues with each given a unique quantum time setting, will provide the scheduler to handle the given processes better. Assigning processes with lower times to the higher priority queues with increase in quantum time periods in the top-down order. When incomplete processes from the first RR queue moves to the second RR queue, it prevents processes that only slightly exceed the first RR queue's quantum from being heavily penalized compared to the absence of the intermediary queue like moving directly to a First Come First Served queue, where the processes will wait unnecessarily long before getting CPU time allocation.

On the other hand, adding an FCFS queue as the intermediary between RR and FCFS may not be as optimal considering that it is non-preemptive. This will extend the waiting periods for subsequent processes that may be much shorter. Moreover, this has much greater effect in breaking the concept of MLFQ schedulers which are usually designed to prioritize processes based on their execution requirements whereby environments with a much diverse range of tasks sets may lead to longer-running processes to take up even more of the CPU allocation before allowing subsequent ones to utilize. This causes greater risks of starvation of processes in the lower-priority queue, as they may be neglected or delayed excessively. Hence, it is more justifiable to use RR as the intermediary layer within the MLFQ scheduler.

6.4.2 Priority Aging

Preventing the starvation of long-waiting tasks in the FCFS queue can be mitigated using a priority aging or task promotion mechanism, which is used to ensure that tasks that are waiting too long, which then causes starvation, get an equal chance to be completed without further delaying.

The continuous arrival of higher priority tasks causes starvation when they are being neglected during the scheduling process. Promoting these tasks that have already waited for a certain extended period prevents these tasks from being starved and ensures that they eventually get CPU allocation time.

Tasks which get delegated to a lower-priority queue can be subjected getting promoted back to a higher-priority queue. This is usually based on how long the task has been waiting. The longer a task waits in the FCFS queue, the higher its priority becomes. This gradual increase in priority is usually implemented through a threshold, whereby the task's waiting time will be monitored, and upon reaching this threshold, its priority increases. The idea is to stop any task from being left out over too long periods of time, which can happen in the FCFS queue if lots of new tasks continuously enter.

The system actively pinpoints tasks whose remaining burst time aligns with the quantum duration of an available RR higher priority queue, promoting them to the respective required queue in accordance with the alignment with the time quantum duration.

In practice, if the MLFQ consists of two high priority queues of RR and one lower priority queue of FCFS, where the first RR queue has been allotted with 8 milliseconds of quantum and the second with 14 milliseconds. Should anytime a task within the FCFS queue has a remnant of 8 milliseconds of burst time, it would be strategically advantageous to elevate this task to the 8-millisecond RR queue.

6.4.3 Optimal Quantum Setting

Careful selection on the quantum time for the RR queue that it should be long enough to reduce frequent context switching but at the same time short enough to ensure the interactivity of the system.

Shorter time quantum in the RR queue typically leads to an increase in the number of contexts switching. The higher overhead contributes to better interactivity and responsiveness of the system. This configuration often benefits jobs which typically require faster and more frequent CPU access, but in turn burdens the system's resources. On the other hand, longer time quantum is associated with fewer context switches, which reduces the overhead and potentially increases the system throughput. This configuration is often more suitable for jobs where each task mainly benefits from longer uninterrupted or non-preempted processing time like CPU-bound jobs, but in return with diminished interactivity.

While the time quantum is only assigned to the RR queue, it is important to note that it also indirectly influences the FCFS queue as well during task transfers. Therefore, the duration a task remains in the RR queue before potentially being transferred to the FCFS queue is affected by the key factor of the set time quantum.

Regarding the connection time quantum has between the RR and FCFS queue during task transfer, establishing an optimal time quantum for the MLFQ scheduler with a single RR and FCFS queue is crucial. Monitoring CPU utilization can assist in the proper setting for the right time quantum, as it indicates how effectively the time of the processor is being handled. Having high CPU utilization indicates that the time quantum is set with the right appropriation, whereas low CPU utilization indicates the excessive overhead produced, suggesting that the time quantum should be increased. However, if the CPU utilization percentage does not present a similar pattern and fluctuates indicates that the selected quantum was not effectively balancing across the workload processes. Therefore, monitoring the correlation between the CPU utilization and quantum setting can reveal the appropriation of the given time quantum for improved performance.

7.0 Conclusion

All three scheduling algorithms worked relatively well for most of the workloads. First Come First Serve (FCFS) is straightforward and simple to implement. It executes processes based on their arrival times and therefore has a low overhead. Round Robin (RR) uses the fundamentals of FCFS but follows the RR principle. It requires a time quantum and operates in a cycle, once a process finishes the time quantum it is preempted, and the next process is executed. RR is suitable for scenarios where balancing is important and allocating units of time fairly to each process in the workload, it prevents starvation and promotes adaptability. Moving on to Multi-Level Feedback Queue (MLFQ), it is the only dynamic scheduling algorithm of the three, it uses two queues with priority, where the first queue is RR and the second is FCFS. All processes start on the RR queue. It is suitable for environments that have a mix of short and long processes as it can provide responsiveness to interactive processes while not neglecting longer processes.

Although FCFS and RR displayed favorable results for most of the workloads, MLFQ has clearly demonstrated the flexibility of having both an RR and FCFS queue. RR works best for workloads that have similar burst times and FCFS works best for workloads that have uniform processes. Therefore, for workloads that have both short and long processes, MLFQ would be able to perform significantly better than RR and FCFS as it utilizes both their properties in one scheduler to achieve the most favorable result. Given MLFQ versatility, it would definitely be chosen as the preferred scheduler if given a random workload where all processes have polarizing differences from one another. It can still provide balanced results whilst maintaining its structure.

While the multiple workload tests have shown the strengths and weaknesses of each scheduling algorithm, MLFQ has been determined as the optimal scheduler of the three as it offers an unrivaled performance profile.

Appendix A: Program Source Code

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>

/**
 * Process Control Block.
 *
 * PCB::label The process ID.
 * PCB::responded Whether the process has responded to the CPU.
 * PCB::at Arrival time.
 * PCB::bt Burst time.
 * PCB::ct Completion time.
 * PCB::rbt Remaining burst time.
 * PCB::rt Response time.
 * PCB::wt Waiting time.
 * PCB::tat Turnaround time.
 * PCB::st Start time.
 * PCB::et End time.
 */
struct PCB {
    int label, responded;
    float at, bt, ct, rbt, rt, wt, tat, st, et;
};

/**
 * A process node in the process queue.
 */
struct ProcessNode {
    struct PCB current;
    struct ProcessNode* next;
};

/**
 * A queue of processes.
 */
struct Queue {
    struct ProcessNode* front;
    struct ProcessNode* rear;
};

/**
 * A block in the Gantt Chart.
 *
 * The Gantt Chart is formatted in such a way where each block leading column will be
the start time of the process,
 * and the trailing column will be the start time of the next process block. Only the
trailing column of the final block
 * will be the end time of the final process.
 */
```

```c
struct GanttNode {
    int label;
    float st, et;
    char* queueLevel;
    struct GanttNode* next;
};

/**
 * The result of the scheduling algorithm.
 */
struct SolutionResult {
    struct Queue* queue;
    struct GanttNode* ganttChart;
    int ganttChartSize;
};

/**
 * Manipulates the process and its nodes.
 */
struct ProcessNode* createProcess(int label, float arrivalTime, float burstTime);
void updateProcessTime(struct Queue* queue, float runtime);

/**
 * Solves the scheduling problem using the given algorithm.
 */
struct SolutionResult solveFCFS(struct Queue* waitingQueue, float* runtime);
struct SolutionResult solveRR(struct Queue* waitingQueue, float quantum, float*
runtime);
struct SolutionResult solveMLFQ(struct Queue* waitingQueue, float quantum, float*
runtime);

/**
 * Manipulates the queue of processes.
 */
struct Queue* createQueue();
struct Queue* duplicateQueue(struct Queue* queue);
void sortQueueByArrivalTime(struct Queue* queue, int processCount);
void enqueue(struct Queue* q, struct PCB data);
struct ProcessNode* peek(struct Queue* q);
struct PCB dequeue(struct Queue* q);
void freeQueue(struct Queue* q);
void emptyQueue(struct Queue* queue);
int isEmpty(struct Queue* q);

/**
 * Used for dynamic table formatting and other micro console visual enhancements.
 */
int intLength(int value);
int getInput(char* input, const char* prompt, int (*validateFunc)(char*), const char*
errorMsg);
int isValidAlgorithm(char* input);
int isValidProcessCount(char* input);
int isValidTime(char* input);
int isValidQuantum(char* input);
```

```c
struct Queue* createQueue() {
    struct Queue* q = (struct Queue*)malloc(sizeof(struct Queue));
    if (q) {
        q->front = q->rear = NULL;
    }
    return q;
}

struct Queue* duplicateQueue(struct Queue* queue) {
    struct Queue* newQueue = createQueue();
    struct ProcessNode* current = queue->front;

    while (current != NULL) {
        struct PCB newPCB = current->current;
        enqueue(newQueue, newPCB);
        current = current->next;
    }

    return newQueue;
}

void sortQueueByArrivalTime(struct Queue* queue, int processCount) {
    if (queue == NULL || isEmpty(queue)) return;

    struct ProcessNode** tempArray = (struct ProcessNode**) calloc(processCount,
sizeof(struct ProcessNode*));
    if (tempArray == NULL) {
        fprintf(stderr, "Memory allocation failed.\n");
        exit(EXIT_FAILURE);
    }

    // Dequeue all elements into an array
    for (int i = 0; i < processCount; i++) {
        tempArray[i] = (struct ProcessNode*) malloc(sizeof(struct ProcessNode));
        if (tempArray[i] == NULL) {
            fprintf(stderr, "Memory allocation failed.\n");
            exit(EXIT_FAILURE);
        }

        tempArray[i]->current = dequeue(queue);
        tempArray[i]->next = NULL;
    }

    // Perform bubble sort on the array
    for (int i = 0; i < processCount - 1; i++) {
        for (int j = 0; j < processCount - i - 1; j++) {
            if (tempArray[j]->current.at > tempArray[j + 1]->current.at) {
                // Swap tempArray[j] and tempArray[j + 1]
                struct ProcessNode* temp = tempArray[j];
                tempArray[j] = tempArray[j + 1];
                tempArray[j + 1] = temp;
            }
        }
```

```
    }

    // Re-enqueue the elements into the queue in sorted order
    for (int i = 0; i < processCount; i++) {
        enqueue(queue, tempArray[i]->current);
        free(tempArray[i]);
    }

    free(tempArray);
}

// Enqueue a PCB into a queue
void enqueue(struct Queue* q, struct PCB data) {
    struct ProcessNode* newNode = (struct ProcessNode*)malloc(sizeof(struct
ProcessNode));
    if (newNode) {
        newNode->current = data;
        newNode->next = NULL;

        if (q->rear == NULL) { // If the queue is empty
            q->front = q->rear = newNode;
        } else {
            q->rear->next = newNode;
            q->rear = newNode;
        }
    }
}

struct ProcessNode* peek(struct Queue* q) {
    return q->front;
}

struct PCB dequeue(struct Queue* q) {
    if (q->front == NULL) {
        return (struct PCB){0}; // Return an empty PCB if the queue is empty
    }

    struct ProcessNode* temp = q->front;
    struct PCB data = temp->current;

    q->front = q->front->next;
    if (q->front == NULL) {
        q->rear = NULL;
    }
    free(temp);
    return data;
}

// Check if a queue is empty
int isEmpty(struct Queue* q) {
    return q->front == NULL;
}

void freeQueue(struct Queue* q) {
```

```
    emptyQueue(q);
    free(q);
}

void emptyQueue(struct Queue* queue) {
    while (!isEmpty(queue)) {
        dequeue(queue);
    }
}

struct ProcessNode* createProcess(int label, float arrivalTime, float burstTime) {
    struct PCB pcb;
    pcb.label = label;
    pcb.at = arrivalTime;
    pcb.bt = burstTime;
    pcb.ct = 0;
    pcb.rbt = burstTime;
    pcb.rt = 0;
    pcb.wt = 0;
    pcb.tat = 0;
    pcb.responded = 0;

    struct ProcessNode* node = (struct ProcessNode*)malloc(sizeof(struct
ProcessNode));
    if (!node) {
        fprintf(stderr, "Failed to allocate memory for new process node.\n");
        exit(EXIT_FAILURE);
    }

    node->current = pcb;
    node->next = NULL;
    return node;
}

void updateProcessTime(struct Queue* queue, float runtime) {
    // Check if the queue is empty
    if (queue == NULL || queue->front == NULL) {
        return;
    }

    // Iterate through each node in the queue
    struct ProcessNode* node = queue->front;
    while (node != NULL) {
        struct PCB* current = &(node->current);

        // Only increment waiting time for processes that have arrived
        if (runtime >= current->at) {
            if (current->responded == 0) {
                current->rt += 1;
            }
            current->wt += 1;
        }

        node = node->next;
```

```
    }
}

struct SolutionResult solveFCFS(struct Queue* waitingQueue, float* runtime) {
    struct Queue* runningQueue = createQueue();
    struct Queue* completedQueue = createQueue();

    struct GanttNode* ganttChart = NULL;
    int ganttChartSize = 0;
    int begin = 0; // Track whether the processing has started;

    float lastRuntime = *runtime - 1;

    while (!isEmpty(waitingQueue) || !isEmpty(runningQueue)) {
        // Check for CPU idle time
        if (begin == 1 && isEmpty(runningQueue) && !isEmpty(waitingQueue) &&
lastRuntime >= 0) {
            float arrivalTime = peek(waitingQueue)->current.at;

            if (ganttChartSize > 0 && ganttChart[ganttChartSize - 1].label == -1) {
                // Update the previous block's et, since it's also an idle block
                ganttChart[ganttChartSize - 1].et = arrivalTime;
            } else {
                // Create a new idle block as CPU is idle until the next process
arrives.
                ganttChart = realloc(ganttChart, sizeof(struct GanttNode) *
(ganttChartSize + 1));
                ganttChart[ganttChartSize].label = -1; // Label for idle time
                ganttChart[ganttChartSize].st = lastRuntime;
                ganttChart[ganttChartSize].et = arrivalTime;
                ganttChart[ganttChartSize].queueLevel = NULL;
                ganttChartSize++;
            }
        }

        // If a process is running, continue
        if (!isEmpty(runningQueue)) {
            struct PCB* current = &peek(runningQueue)->current;

            // Process starts
            if (current->rbt == current->bt) {
                ganttChart = realloc(ganttChart, sizeof(struct GanttNode) *
(ganttChartSize + 1));
                ganttChart[ganttChartSize].label = current->label;
                ganttChart[ganttChartSize].st = lastRuntime;
                ganttChart[ganttChartSize].queueLevel = NULL;
            }

            // Process starts or continues
            if (current->responded == 0) {
                current->responded = 1; // Mark as responded
                current->rt = lastRuntime - current->at;

                updateProcessTime(runningQueue, *runtime);
```

```
                current->st = lastRuntime;
            }

            // Update the process time
            current->rbt -= 1;

            // Process ends
            if (current->rbt == 0) {
                current->et = *runtime;
                ganttChart[ganttChartSize].et = *runtime;
                ganttChartSize++;

                struct PCB completedProcess = dequeue(runningQueue);
                completedProcess.ct = *runtime;
                completedProcess.tat = completedProcess.ct - completedProcess.at;
                completedProcess.wt = completedProcess.tat - completedProcess.bt;

                enqueue(completedQueue, completedProcess);
            }
        }

        // Check for new arrivals
        while (!isEmpty(waitingQueue) && peek(waitingQueue)->current.at == *runtime) {
            struct PCB arrivedProcess = dequeue(waitingQueue);
            arrivedProcess.st = lastRuntime;
            enqueue(runningQueue, arrivedProcess);
            begin = 1;
        }

        // Increment time
        (*runtime)++;
        lastRuntime++;

        if (isEmpty(waitingQueue) && isEmpty(runningQueue)) {
            break;
        }
    }

    return (struct SolutionResult){completedQueue, ganttChart, ganttChartSize};
}

struct SolutionResult solveRR(struct Queue* waitingQueue, float quantum, float*
runtime) {
    struct Queue* runningQueue = createQueue();
    struct Queue* completedQueue = createQueue();
    float quantumLeft = quantum;

    struct GanttNode* ganttChart = NULL;
    int ganttChartSize = 0;
    int begin = 0; // Track whether the processing has started;

    float lastRuntime = *runtime - 1;
```

```
    while (!isEmpty(waitingQueue) || !isEmpty(runningQueue)) {
        // Check for CPU idle time
        if (begin == 1 && isEmpty(runningQueue) && !isEmpty(waitingQueue) &&
lastRuntime >= 0) {
            float arrivalTime = peek(waitingQueue)->current.at;

            if (ganttChartSize > 0 && ganttChart[ganttChartSize - 1].label == -1) {
                // Update the previous block's et, since it's also an idle block
                ganttChart[ganttChartSize - 1].et = arrivalTime;
            } else {
                // Create a new idle block as CPU is idle until the next process
arrives.
                ganttChart = realloc(ganttChart, sizeof(struct GanttNode) *
(ganttChartSize + 1));
                ganttChart[ganttChartSize].label = -1; // Label for idle time
                ganttChart[ganttChartSize].st = lastRuntime;
                ganttChart[ganttChartSize].et = arrivalTime;
                ganttChart[ganttChartSize].queueLevel = NULL;
                ganttChartSize++;
            }
        }

        if (!isEmpty(runningQueue)) {
            struct PCB* current = &peek(runningQueue)->current;

            // Processing restarts its time slice from the allocated quantum.
            if (quantumLeft == quantum) {
                ganttChart = realloc(ganttChart, sizeof(struct GanttNode) *
(ganttChartSize + 1));
                ganttChart[ganttChartSize].label = current->label;
                ganttChart[ganttChartSize].st = lastRuntime;
                ganttChart[ganttChartSize].queueLevel = NULL;
                ganttChartSize++;
            }

            if (current->responded == 0) {
                current->responded = 1;
                current->st = lastRuntime;
            }

            current->rbt -= 1;
            quantumLeft -= 1;

            updateProcessTime(runningQueue, *runtime);

            // Process ends
            if (current->rbt == 0) {
                current->et = *runtime;
                current->ct = *runtime;
                current->tat = current->ct - current->at;
                current->wt = current->tat - current->bt;

                enqueue(completedQueue, *current);
                dequeue(runningQueue);
```

```
                // Reset quantum for the next process
                quantumLeft = quantum;

                ganttChart[ganttChartSize - 1].et = *runtime;
            }
        }

        while (!isEmpty(waitingQueue) && peek(waitingQueue)->current.at == *runtime) {
            struct PCB arrived = dequeue(waitingQueue);

            // Set the start time for the process if it's the first time being
processed
            if (arrived.responded == 0) {
                arrived.st = *runtime;
            }
            enqueue(runningQueue, arrived);
            begin = 1;
        }

        if (quantumLeft == 0) {
            struct PCB* current = &peek(runningQueue)->current;
            current->et = *runtime;

            struct PCB incomplete = dequeue(runningQueue);
            enqueue(runningQueue, incomplete);

            quantumLeft = quantum;

            ganttChart[ganttChartSize - 1].et = *runtime;
        }

        (*runtime)++;
        lastRuntime++;

        if (isEmpty(waitingQueue) && isEmpty(runningQueue)) {
            break;
        }
    }

    return (struct SolutionResult){completedQueue, ganttChart, ganttChartSize};
}

struct SolutionResult solveMLFQ(struct Queue* waitingQueue, float quantum, float*
runtime) {
    struct Queue* runningRRQueue = createQueue(); // high priority queue
    struct Queue* runningFCFSQueue = createQueue(); // low priority queue
    struct Queue* completedQueue = createQueue();
    float quantumLeft = quantum;

    struct GanttNode* ganttChart = NULL;
    int ganttChartSize = 0;
    int begin = 0; // Track whether the processing has started;
```

```
    float lastRuntime = *runtime - 1;

    while (!isEmpty(waitingQueue) || !isEmpty(runningRRQueue)
|| !isEmpty(runningFCFSQueue)) {
        // Check for CPU idle time
        if (begin == 1 && isEmpty(runningRRQueue) && isEmpty(runningFCFSQueue)
&& !isEmpty(waitingQueue) && lastRuntime >= 0) {
            float at = peek(waitingQueue)->current.at;

            if (ganttChartSize > 0 && ganttChart[ganttChartSize - 1].label == -1) {
                // Update the previous block's end time, since it's also an idle
block.
                ganttChart[ganttChartSize - 1].et = at;
            } else {
                // Create a new idle block as CPU is idle until the next process
arrives.
                ganttChart = realloc(ganttChart, sizeof(struct GanttNode) *
(ganttChartSize + 1));
                ganttChart[ganttChartSize].label = -1; // Label for idle time
                ganttChart[ganttChartSize].st = lastRuntime;
                ganttChart[ganttChartSize].et = at;
                ganttChart[ganttChartSize].queueLevel = NULL;
                ganttChartSize++;
            }
        }

        if (!isEmpty(runningRRQueue)) {
            struct PCB* current = &peek(runningRRQueue)->current;

            // Processing restarts its time slice from the allocated quantum.
            if (current->responded == 0 || quantumLeft == quantum) {
                current->responded = 1;
                current->st = lastRuntime;

                ganttChart = realloc(ganttChart, sizeof(struct GanttNode) *
(ganttChartSize + 1));
                ganttChart[ganttChartSize].label = current->label;
                ganttChart[ganttChartSize].st = lastRuntime;
                ganttChart[ganttChartSize].queueLevel = "RR";
                ganttChartSize++;
            }

            current->rbt -= 1;
            quantumLeft -= 1;

            updateProcessTime(runningRRQueue, *runtime);
            updateProcessTime(runningFCFSQueue, *runtime);

            if (current->rbt == 0) {
                current->et = *runtime;
                current->ct = *runtime;
                current->tat = current->ct - current->at;
                current->wt = current->tat - current->bt;
```

```
                enqueue(completedQueue, *current);
                dequeue(runningRRQueue);

                quantumLeft = quantum;

                ganttChart[ganttChartSize - 1].et = *runtime;
            } else if (quantumLeft == 0) {
                struct PCB incomplete = dequeue(runningRRQueue);
                enqueue (runningFCFSQueue, incomplete);

                quantumLeft = quantum;

                ganttChart[ganttChartSize - 1].et = *runtime;

                // Marks the demotion to FCFS.
                ganttChart = realloc(ganttChart, sizeof(struct GanttNode) *
(ganttChartSize + 1));
                ganttChart[ganttChartSize].label = incomplete.label;
                ganttChart[ganttChartSize].st = *runtime;
                ganttChart[ganttChartSize].et = *runtime;
                ganttChart[ganttChartSize].queueLevel = "FCFS";
                ganttChartSize++;
            }
        } else if (!isEmpty(runningFCFSQueue)) {
            struct PCB* current = &peek(runningFCFSQueue)->current;

            // Processing restarts its time slice from the allocated quantum.
            if (current->responded == 0) {
                current->responded = 1;
                current->st = lastRuntime;

                ganttChart[ganttChartSize - 1].et = *runtime;

                ganttChart = realloc(ganttChart, sizeof(struct GanttNode) *
(ganttChartSize + 1));
                ganttChart[ganttChartSize].label = current->label;
                ganttChart[ganttChartSize].st = lastRuntime;
                ganttChart[ganttChartSize].queueLevel = "FCFS";
                ganttChartSize++;
            }

            current->rbt -= 1;

            updateProcessTime(runningFCFSQueue, *runtime);

            if (current->rbt == 0) {
                current->et = *runtime;
                current->ct = *runtime;
                current->tat = current->ct - current->at;
                current->wt = current->tat - current->bt;

                enqueue(completedQueue, *current);
                dequeue (runningFCFSQueue);
```

```
                ganttChart[ganttChartSize - 1].et = *runtime;
            }
        }

        while (!isEmpty(waitingQueue) && peek(waitingQueue)->current.at == *runtime) {
            struct PCB arrived = dequeue(waitingQueue);

            // Set the start time for the process if it's the first time being
processed
            if (arrived.responded == 0) {
                arrived.st = *runtime;
            }
            enqueue(runningRRQueue, arrived);
            begin = 1;
        }

        (*runtime)++;
        lastRuntime++;

        if (isEmpty(waitingQueue) && isEmpty(runningRRQueue) &&
isEmpty(runningFCFSQueue)) {
            break;
        }
    }

    return (struct SolutionResult){completedQueue, ganttChart, ganttChartSize};
}

int intLength(int value) {
    if (value == 0) return 1;
    value = abs(value);
    return floor(log10(value)) + 1;
}

int getInput(char* input, const char* prompt, int (*validateFunc)(char*), const char*
errorMsg) {
    while (1) {
        printf("%s", prompt);
        scanf("%s", input);

        if (strcmp(input, "q") == 0) {
            printf("\nProgram terminated.\n");
            return 0; // Quit signal
        } else if (strcmp(input, "r") == 0) {
            return 1; // Restart signal
        }

        if (!validateFunc(input)) {
            printf("%s\n\n", errorMsg);
            continue;
        }

        return 2; // Valid input signal
    }
```

```
}

int isValidAlgorithm(char* input) {
    int algorithm = atoi(input);
    return (algorithm == 1 || algorithm == 2 || algorithm == 3);
}

int isValidProcessCount(char* input) {
    int count = atoi(input);
    return (count > 0);
}

int isValidTime(char* input) {
    float time = atof(input);
    return (time >= 0);
}

int isValidQuantum(char* input) {
    float quantum = atof(input);
    return (quantum > 0);
}

int main() {
    printf("\n\n");
    printf("                    ---------- [ CPU SCHEDULING ALGORITHMS ] ---------
-                   \n\n");
    printf("[Enter \"q\" to quit or \"r\" to restart the program at any time.]\n");

    char input[100];

    while (1) {
        struct Queue* waitingQueue = createQueue();

        int algorithm, processCounter = 0;
        float quantum = 0;
        float runtime = -1;

        printf("\n");
        printf("                    -------------------- [ Input ] -------------------
-                   \n\n");
        printf("Choose the scheduling algorithm:\n");
        printf("1. First Come First Served (FCFS)\n");
        printf("2. Round Robin (RR)\n");
        printf("3. Multi-Level Feedback Queue (MLFQ)\n");

        int inputSignal = getInput(input, "Enter your choice of algorithm: ",
isValidAlgorithm, "Invalid choice of algorithm.");
        if (inputSignal == 1) {
            continue;
        } else if (inputSignal == 0) {
            return inputSignal;
        }
        algorithm = atoi(input);
```

```
        printf("\n");

        inputSignal = getInput(input, "Enter the total number of processes: ",
isValidProcessCount, "Invalid number of processes.");
        if (inputSignal == 1) {
            continue;
        } else if (inputSignal == 0) {
            return inputSignal;
        }
        processCounter = atoi(input);

        printf ("\n");

        for (int i = 0; i < processCounter; i++) {
            float arrivalTime, burstTime;

            printf("Process No. %d: \n", i + 1);

            inputSignal = getInput(input, "Enter arrival time: ", isValidTime,
"Invalid arrival time.");
            if (inputSignal == 1) {
                break;
            } else if (inputSignal == 0) {
                return inputSignal;
            }
            arrivalTime = atof(input);

            inputSignal = getInput(input, "Enter burst time: ", isValidTime, "Invalid
burst time.");
            if (inputSignal == 1) {
                break;
            } else if (inputSignal == 0) {
                return inputSignal;
            }
            burstTime = atof(input);

            printf("\n");

            struct ProcessNode* node = createProcess(i + 1, arrivalTime, burstTime);
            enqueue(waitingQueue, node->current);
            free(node);
        }

        if (inputSignal == 1) {
            continue;
        }

        if (algorithm == 2 || algorithm == 3) {
            inputSignal = getInput(input, "Enter time quantum: ", isValidQuantum,
"Invalid time quantum.");
            if (inputSignal == 1) {
                continue;
            } else if (inputSignal == 0) {
                return inputSignal;
```

```
            }
            quantum = atof(input);
        }

        printf("\n");

        struct Queue* initialWaitingQueue = duplicateQueue(waitingQueue);
        sortQueueByArrivalTime(waitingQueue, processCounter);

        struct SolutionResult result;

        switch (algorithm) {
            case 1:
                result = solveFCFS(waitingQueue, &runtime);
                break;
            case 2:
                result = solveRR(waitingQueue, quantum, &runtime);
                break;
            case 3:
                result = solveMLFQ(waitingQueue, quantum, &runtime);
                break;
        }

        printf("                    -------------------- [ Output ] ------------------
--                    \n\n");
        switch (algorithm) {
            case 1:
                printf("                    ------------ [ First Come First Served ] -
-----------                         \n\n");
                break;
            case 2:
                printf("                    ----------------- [ Round Robin ] --------
---------                       \n\n");
                break;
            case 3:
                printf("                    ------------ [ Multi-Level Feedback
Queue ] ------------                    \n\n");
                break;
        }

        printf("Initial Queue");
        printf(algorithm == 1 ? ": \n" : " (with time quantum of %.0f): \n", quantum);
        printf("+------------+--------------+------------+\n");
        printf("| Process ID | Arrival Time | Burst Time |\n");
        printf("+------------+--------------+------------+\n");
        while (!isEmpty(initialWaitingQueue)) {
            struct PCB current = dequeue(initialWaitingQueue);
            printf("| %-10d | %-12.0f | %-10.0f |\n", current.label, current.at,
current.bt);
        }
        printf("+------------+--------------+------------+\n\n");
        freeQueue(initialWaitingQueue);

        printf("Process Start and End Times (in ascending order of completion):\n");
```

```
        printf("+------------+------------+----------+\n");
        printf("| Process ID | Start Time | End Time |\n");
        printf("+------------+------------+----------+\n");
        struct Queue* completedQueue = result.queue;
        struct ProcessNode* process = completedQueue->front;
        while (process != NULL) {
            printf("| %-10d | %-10.0f | %-8.0f |\n", process->current.label,
process->current.st, process->current.et);
            process = process->next;
        }
        printf("+------------+------------+----------+\n\n");

        struct GanttNode* ganttChart = result.ganttChart;
        int ganttChartSize = result.ganttChartSize;
        int ganttBlocksPerLine = 10;
        int ganttBlocksLeftOnLine = ganttBlocksPerLine;
        int totalIdleBlocks = 0;
        int totalExecutionBlocks = 0;

        printf("Gantt Chart (with time units");
        printf(algorithm == 3 ? " and queue levels):\n" : "):\n");
        int ganttBlocksInCurrentLine = 0;

        for (int i = 0; i < ganttChartSize; ++i) {
            if (ganttBlocksInCurrentLine == 0) {
                printf("+");
            }

            int label = ganttChart[i].label;
            if (label == -1) {
                totalIdleBlocks++;
            } else {
                totalExecutionBlocks++;
            }

            int timeUnitLen = intLength((int) ganttChart[i].et - (int)
ganttChart[i].st);

            int totalDashes = label == -1 ? 9 : 7;
            if (ganttChart[i].queueLevel != NULL) {
                totalDashes += strlen(ganttChart[i].queueLevel) + 1;
            }
            totalDashes += timeUnitLen;

            for (int k = 0; k < totalDashes; ++k) {
                printf("-");
            }
            printf("+");
            ganttBlocksInCurrentLine++;

            if (ganttBlocksInCurrentLine >= ganttBlocksPerLine || i == ganttChartSize
- 1) {
                printf("\n|");
```

```
                // Process labels for the current line
                for (int j = i - ganttBlocksInCurrentLine + 1; j <= i; ++j) {
                    int label = ganttChart[j].label;
                    int timeUnit = (int) ganttChart[j].et - (int) ganttChart[j].st;
                    char* queueLevel = ganttChart[j].queueLevel;

                    if (label == -1) {
                        printf(" Idle [%d] |", timeUnit);
                    } else {
                        if (queueLevel == NULL) {
                            printf(" P%-d [%d] |", label, timeUnit);
                        } else {
                            printf(" %s P%-d [%d] |", queueLevel, label, timeUnit);
                        }
                    }
                }

                printf("\n+");

                for (int j = i - ganttBlocksInCurrentLine + 1; j <= i; ++j) {
                    int label = ganttChart[j].label;
                    char* queueLevel = ganttChart[j].queueLevel;

                    if (label == -1) {
                        printf("----------");
                    } else {
                        printf("-------");
                        if (queueLevel != NULL) {
                            for (int k = 0; k < strlen(queueLevel) + 1; ++k) {
                                printf("-");
                            }
                        }
                    }

                    int timeUnitLen = intLength((int) ganttChart[j].et - (int)
ganttChart[j].st);
                    for (int k = 0; k < timeUnitLen; ++k) {
                        printf("-");
                    }
                    printf("+");
                }

                // Time units for the current line of blocks
                printf("\n");
                for (int j = i - ganttBlocksInCurrentLine + 1; j <= i; ++j) {
                    int label = ganttChart[j].label;
                    int timeUnitLen = intLength((int) ganttChart[j].et - (int)
ganttChart[j].st);
                    char* queueLevel = ganttChart[j].queueLevel;
                    int cellSize = (label == -1 ? 10 : 8) + timeUnitLen;
                    if (queueLevel != NULL) {
                        cellSize += strlen(queueLevel) + 1;
                    }
```

```
                    printf("%-*d", cellSize, (int) ganttChart[j].st);
                }

                if (ganttBlocksInCurrentLine == ganttBlocksPerLine || i ==
ganttChartSize - 1) {
                    printf("%-8d", (int) ganttChart[i].et);
                }

                printf("\n");

                ganttBlocksInCurrentLine = 0;
            }
        }

        printf("\n");

        printf("+------------------+-------------+\n");
        printf("| Execution Blocks | Idle Blocks |\n");
        printf("+------------------+-------------+\n");
        printf("| %-16d | %-11d |\n", totalExecutionBlocks, totalIdleBlocks);
        printf("+------------------+-------------+\n\n");

        float averageRT = 0, averageWT = 0, averageTAT = 0;

        printf("Scheduling Calculation Information (in ascending order of completion):
\n");
        printf("+------------+--------------+------------+-----------------+----------
-------+--------------+---------------+\n");
        printf("| Process ID | Arrival Time | Burst Time | Completion Time |
Turnaround Time | Waiting Time | Response Time |\n");
        printf("+------------+--------------+------------+-----------------+----------
-------+--------------+---------------+\n");

        struct Queue* tempQueue = duplicateQueue(completedQueue);
        while (!isEmpty(tempQueue)) {
            struct PCB current = dequeue(tempQueue);
            averageRT += current.rt;
            averageWT += current.wt;
            averageTAT += current.tat;

            printf("| %-10d | %-12.0f | %-10.0f | %-15.0f | %-15.0f | %-12.0f | %-
13.0f |\n",
                    current.label, current.at, current.bt,
                    current.ct, current.tat, current.wt, current.rt);
        }

        printf("+------------+--------------+------------+-----------------+----------
-------+--------------+---------------+\n\n");

        averageRT /= (float) processCounter;
        averageWT /= (float) processCounter;
        averageTAT /= (float) processCounter;
        float throughput = (float) processCounter / runtime;
```

```
        printf("Scheduling Calculation Performance (in milliseconds): \n");
        printf("+----------------------+--------------------+----------------------
---+------------+\n");
        printf("| Average Response Time | Average Waiting Time | Average Turnaround
Time | Throughput |\n");
        printf("+----------------------+--------------------+----------------------
---+------------+\n");
        printf("| %-21.3f | %-20.3f | %-23.3f | %-10.3f |\n",
               averageRT, averageWT, averageTAT, throughput);
        printf("+----------------------+--------------------+----------------------
---+------------+\n");

        freeQueue(waitingQueue);
        freeQueue(completedQueue);
    }
}
```

Appendix B: Console Input / Output Interface

B.1 First Come First Served Algorithm

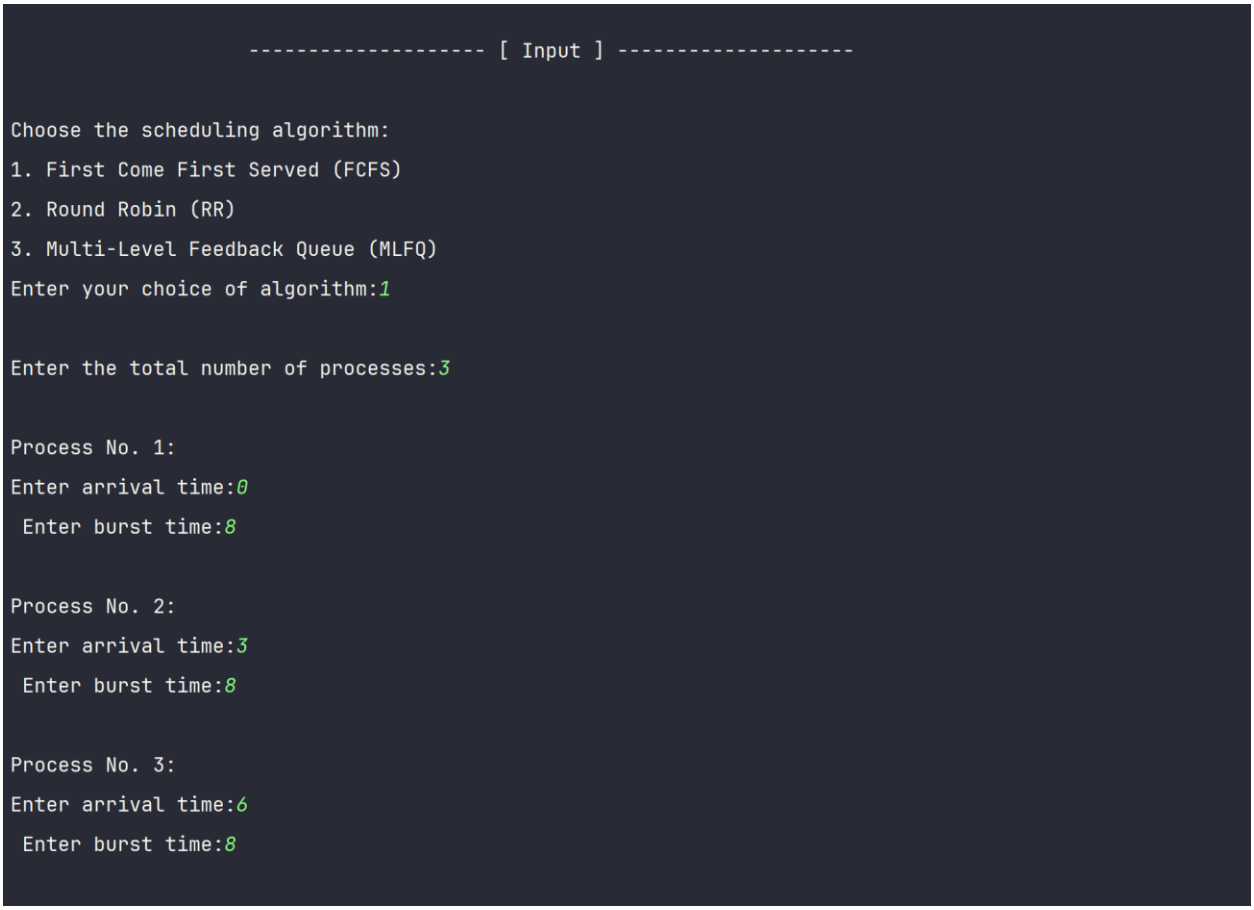


Figure B.1 Input Interface for FCFS for 3 tasks with Difference in Arrival Times of 3 and Fixed Burst Time of 8

```
-------------------- [ Output ] --------------------

------------ [ First Come First Served ] ------------

Initial Queue:
+------------+--------------+------------+
| Process ID | Arrival Time | Burst Time |
+------------+--------------+------------+
| 1          | 0            | 8          |
| 2          | 3            | 8          |
| 3          | 6            | 8          |
+------------+--------------+------------+

Process Start and End Times (in ascending order of completion):
+------------+------------+----------+
| Process ID | Start Time | End Time |
+------------+------------+----------+
| 1          | 0          | 8        |
| 2          | 8          | 16       |
| 3          | 16         | 24       |
+------------+------------+----------+
```

Figure B.2a Output Interface for FCFS for 3 tasks with Difference in Arrival Times of 3 and Fixed Burst Time of 8

```
Gantt Chart (with time units):
+--------+--------+--------+
| P1 [8] | P2 [8] | P3 [8] |
+--------+--------+--------+
0        8        16       24

+------------------+-------------+
| Execution Blocks | Idle Blocks |
+------------------+-------------+
| 3                | 0           |
+------------------+-------------+

Scheduling Calculation Information (in ascending order of completion):
+------------+--------------+------------+-----------------+-----------------+--------------+---------------+
| Process ID | Arrival Time | Burst Time | Completion Time | Turnaround Time | Waiting Time | Response Time |
+------------+--------------+------------+-----------------+-----------------+--------------+---------------+
| 1          | 0            | 8          | 8               | 8               | 0            | 0             |
| 2          | 3            | 8          | 16              | 13              | 5            | 5             |
| 3          | 6            | 8          | 24              | 18              | 10           | 10            |
+------------+--------------+------------+-----------------+-----------------+--------------+---------------+

Scheduling Calculation Performance (in milliseconds):
+----------------------+---------------------+-------------------------+------------+
| Average Response Time | Average Waiting Time | Average Turnaround Time | Throughput |
+----------------------+---------------------+-------------------------+------------+
| 5.000                | 5.000               | 13.000                  | 0.120      |
+----------------------+---------------------+-------------------------+------------+
```

Figure B.2b Output Interface for FCFS for 3 tasks with Difference in Arrival Times of 3 and Fixed Burst Time of 8

```
                    -------------------- [ Input ] --------------------

Choose the scheduling algorithm:
1. First Come First Served (FCFS)
2. Round Robin (RR)
3. Multi-Level Feedback Queue (MLFQ)
Enter your choice of algorithm:1

Enter the total number of processes:5

Process No. 1:
Enter arrival time:10
 Enter burst time:9

Process No. 2:
Enter arrival time:23
 Enter burst time:4

Process No. 3:
Enter arrival time:8
 Enter burst time:19

Process No. 4:
Enter arrival time:2
 Enter burst time:48

Process No. 5:
Enter arrival time:5
 Enter burst time:39
```

Figure B.3 Input Interface for FCFS for 5 tasks with Random Arrival and Burst Times

```
                    -------------------- [ Output ] --------------------

                    ------------ [ First Come First Served ] ------------

 Initial Queue:
 +------------+--------------+------------+
 | Process ID | Arrival Time | Burst Time |
 +------------+--------------+------------+
 | 1          | 10           | 9          |
 | 2          | 23           | 4          |
 | 3          | 8            | 19         |
 | 4          | 2            | 48         |
 | 5          | 5            | 39         |
 +------------+--------------+------------+

 Process Start and End Times (in ascending order of completion):
 +------------+------------+----------+
 | Process ID | Start Time | End Time |
 +------------+------------+----------+
 | 4          | 2          | 50       |
 | 5          | 50         | 89       |
 | 3          | 89         | 108      |
 | 1          | 108        | 117      |
 | 2          | 117        | 121      |
 +------------+------------+----------+
```

Figure B.4a Output Interface for FCFS for 5 tasks with Random Arrival and Burst Times

```
Gantt Chart (with time units):
+---------+---------+---------+--------+--------+
| P4 [48] | P5 [39] | P3 [19] | P1 [9] | P2 [4] |
+---------+---------+---------+--------+--------+
2         50        89        108      117      121

+------------------+-------------+
| Execution Blocks | Idle Blocks |
+------------------+-------------+
| 5                | 0           |
+------------------+-------------+

Scheduling Calculation Information (in ascending order of completion):
+------------+--------------+------------+-----------------+-----------------+--------------+---------------+
| Process ID | Arrival Time | Burst Time | Completion Time | Turnaround Time | Waiting Time | Response Time |
+------------+--------------+------------+-----------------+-----------------+--------------+---------------+
| 4          | 2            | 48         | 50              | 48              | 0            | 0             |
| 5          | 5            | 39         | 89              | 84              | 45           | 45            |
| 3          | 8            | 19         | 108             | 100             | 81           | 81            |
| 1          | 10           | 9          | 117             | 107             | 98           | 98            |
| 2          | 23           | 4          | 121             | 98              | 94           | 94            |
+------------+--------------+------------+-----------------+-----------------+--------------+---------------+

Scheduling Calculation Performance (in milliseconds):
+----------------------+---------------------+-------------------------+------------+
| Average Response Time | Average Waiting Time | Average Turnaround Time | Throughput |
+----------------------+---------------------+-------------------------+------------+
| 63.600               | 63.600              | 87.400                  | 0.041      |
+----------------------+---------------------+-------------------------+------------+
```

Figure B.4b Output Interface for FCFS for 5 tasks with Random Arrival and Burst Times

B.2 Round Robin

```
                    -------------------- [ Input ] --------------------

Choose the scheduling algorithm:
1. First Come First Served (FCFS)
2. Round Robin (RR)
3. Multi-Level Feedback Queue (MLFQ)
Enter your choice of algorithm:2

Enter the total number of processes:3

Process No. 1:
Enter arrival time:0
 Enter burst time:8

Process No. 2:
Enter arrival time:5
 Enter burst time:8

Process No. 3:
Enter arrival time:10
 Enter burst time:8

Enter time quantum:3
```

Figure B.5 Input Interface for RR for 3 tasks with Difference in Arrival Times of 5, Fixed Burst Time of 8 and Quantum of 8

```
                    -------------------- [ Output ] --------------------

                    ----------------- [ Round Robin ] -----------------

Initial Queue (with time quantum of 3):
+------------+--------------+------------+
| Process ID | Arrival Time | Burst Time |
+------------+--------------+------------+
| 1          | 0            | 8          |
| 2          | 5            | 8          |
| 3          | 10           | 8          |
+------------+--------------+------------+

Process Start and End Times (in ascending order of completion):
+------------+------------+----------+
| Process ID | Start Time | End Time |
+------------+------------+----------+
| 1          | 0          | 11       |
| 2          | 6          | 19       |
| 3          | 14         | 24       |
+------------+------------+----------+
```

Figure B.6a Output Interface for RR for 3 tasks with Difference in Arrival Times of 5, Fixed Burst Time of 8 and Quantum of 8

```
Gantt Chart (with time units):
+--------+--------+--------+--------+--------+--------+--------+--------+--------+
| P1 [3] | P1 [3] | P2 [3] | P1 [2] | P2 [3] | P3 [3] | P2 [2] | P3 [3] | P3 [2] |
+--------+--------+--------+--------+--------+--------+--------+--------+--------+
0        3        6        9        11       14       17       19       22       24

+------------------+-------------+
| Execution Blocks | Idle Blocks |
+------------------+-------------+
| 9                | 0           |
+------------------+-------------+

Scheduling Calculation Information (in ascending order of completion):
+------------+--------------+------------+-----------------+-----------------+--------------+--------------+
| Process ID | Arrival Time | Burst Time | Completion Time | Turnaround Time | Waiting Time | Response Time |
+------------+--------------+------------+-----------------+-----------------+--------------+--------------+
| 1          | 0            | 8          | 11              | 11              | 3            | 0            |
| 2          | 5            | 8          | 19              | 14              | 6            | 1            |
| 3          | 10           | 8          | 24              | 14              | 6            | 4            |
+------------+--------------+------------+-----------------+-----------------+--------------+--------------+

Scheduling Calculation Performance (in milliseconds):
+----------------------+---------------------+------------------------+------------+
| Average Response Time | Average Waiting Time | Average Turnaround Time | Throughput |
+----------------------+---------------------+------------------------+------------+
| 1.667                | 5.000               | 13.000                 | 0.120      |
+----------------------+---------------------+------------------------+------------+
```

Figure B.6b Output Interface for RR for 3 tasks with Difference in Arrival Times of 5, Fixed Burst Time of 8 and Quantum of 8

```
                    -------------------- [ Input ] --------------------

Choose the scheduling algorithm:
1. First Come First Served (FCFS)
2. Round Robin (RR)
3. Multi-Level Feedback Queue (MLFQ)
Enter your choice of algorithm:2

Enter the total number of processes:6
```

Figure B.7a Input Interface for RR for 3 tasks with Random Arrival and Burst Times with a Quantum of 2

```
Process No. 1:
Enter arrival time:7
 Enter burst time:4

Process No. 2:
Enter arrival time:5
 Enter burst time:9

Process No. 3:
Enter arrival time:3
 Enter burst time:71

Process No. 4:
Enter arrival time:5
 Enter burst time:3

Process No. 5:
Enter arrival time:82
 Enter burst time:5

Process No. 6:
Enter arrival time:4
 Enter burst time:2

Enter time quantum:2
```

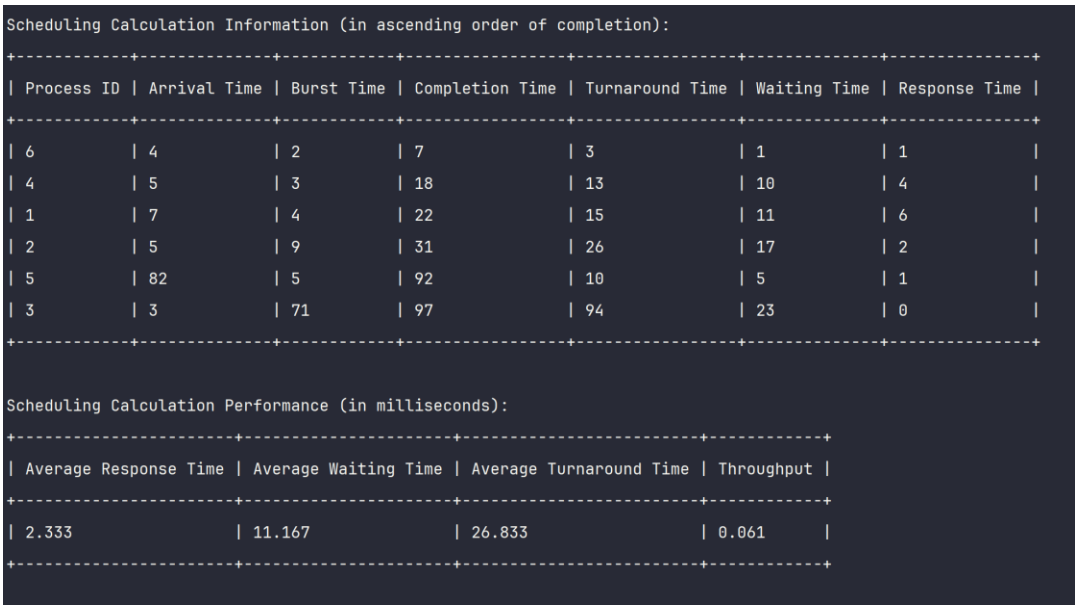
Figure B.7b Input Interface for RR for 3 tasks with Random Arrival and Burst Times with a Quantum of 2

```
                    ------------------- [ Output ] -------------------

                    ----------------- [ Round Robin ] -----------------

Initial Queue (with time quantum of 2):
+------------+--------------+------------+
| Process ID | Arrival Time | Burst Time |
+------------+--------------+------------+
| 1          | 7            | 4          |
| 2          | 5            | 9          |
| 3          | 3            | 71         |
| 4          | 5            | 3          |
| 5          | 82           | 5          |
| 6          | 4            | 2          |
+------------+--------------+------------+

Process Start and End Times (in ascending order of completion):
+------------+------------+----------+
| Process ID | Start Time | End Time |
+------------+------------+----------+
| 6          | 5          | 7        |
| 4          | 9          | 18       |
| 1          | 13         | 22       |
| 2          | 7          | 31       |
| 5          | 83         | 92       |
| 3          | 3          | 97       |
+------------+------------+----------+
```

Figure B.8a Output Interface for RR for 3 tasks with Random Arrival and Burst Times with a Quantum of 2

```
Gantt Chart (with time units):
+--------+--------+--------+--------+--------+--------+--------+--------+--------+--------+
| P3 [2] | P6 [2] | P2 [2] | P4 [2] | P3 [2] | P1 [2] | P2 [2] | P4 [1] | P3 [2] | P1 [2] |
+--------+--------+--------+--------+--------+--------+--------+--------+--------+--------+
3        5        7        9        11       13       15       17       18       20       22
+--------+--------+--------+--------+--------+--------+--------+--------+--------+--------+
| P2 [2] | P3 [2] | P2 [2] | P3 [2] | P2 [1] | P3 [2] | P3 [2] | P3 [2] | P3 [2] | P3 [2] |
+--------+--------+--------+--------+--------+--------+--------+--------+--------+--------+
22       24       26       28       30       31       33       35       37       39       41
+--------+--------+--------+--------+--------+--------+--------+--------+--------+--------+
| P3 [2] | P3 [2] | P3 [2] | P3 [2] | P3 [2] | P3 [2] | P3 [2] | P3 [2] | P3 [2] | P3 [2] |
+--------+--------+--------+--------+--------+--------+--------+--------+--------+--------+
41       43       45       47       49       51       53       55       57       59       61
+--------+--------+--------+--------+--------+--------+--------+--------+--------+--------+
| P3 [2] | P3 [2] | P3 [2] | P3 [2] | P3 [2] | P3 [2] | P3 [2] | P3 [2] | P3 [2] | P3 [2] |
+--------+--------+--------+--------+--------+--------+--------+--------+--------+--------+
61       63       65       67       69       71       73       75       77       79       81
+--------+--------+--------+--------+--------+--------+--------+--------+--------+
| P3 [2] | P5 [2] | P3 [2] | P5 [2] | P3 [2] | P5 [1] | P3 [2] | P3 [2] | P3 [1] |
+--------+--------+--------+--------+--------+--------+--------+--------+--------+
81       83       85       87       89       91       92       94       96       97

+------------------+-------------+
| Execution Blocks | Idle Blocks |
+------------------+-------------+
| 49               | 0           |
+------------------+-------------+
```

Figure B.8b Output Interface for RR for 3 tasks with Random Arrival and Burst Times with a Quantum of 2



Scheduling Calculation Information (in ascending order of completion):

Process ID	Arrival Time	Burst Time	Completion Time	Turnaround Time	Waiting Time	Response Time
6	4	2	7	3	1	1
4	5	3	18	13	10	4
1	7	4	22	15	11	6
2	5	9	31	26	17	2
5	82	5	92	10	5	1
3	3	71	97	94	23	0

Scheduling Calculation Performance (in milliseconds):

Average Response Time	Average Waiting Time	Average Turnaround Time	Throughput
2.333	11.167	26.833	0.061

Figure B.8c Output Interface for RR for 3 tasks with Random Arrival and Burst Times with a Quantum of 2

B.3 Multi-Level Feedback Queue

```
                    -------------------- [ Input ] --------------------

Choose the scheduling algorithm:
1. First Come First Served (FCFS)
2. Round Robin (RR)
3. Multi-Level Feedback Queue (MLFQ)
Enter your choice of algorithm:3

Enter the total number of processes:3

Process No. 1:
Enter arrival time:0
 Enter burst time:8

Process No. 2:
Enter arrival time:4
 Enter burst time:8

Process No. 3:
Enter arrival time:8
 Enter burst time:8

Enter time quantum:4
```

Figure B.9 Input Interface for MLFQ for 3 tasks with Difference in Arrival Times of 4, Fixed Burst Time of 8 and Quantum of 4

```
                    -------------------- [ Output ] --------------------

                    ------------ [ Multi-Level Feedback Queue ] ------------

Initial Queue (with time quantum of 4):
+------------+--------------+------------+
| Process ID | Arrival Time | Burst Time |
+------------+--------------+------------+
| 1          | 0            | 8          |
| 2          | 4            | 8          |
| 3          | 8            | 8          |
+------------+--------------+------------+

Process Start and End Times (in ascending order of completion):
+------------+------------+----------+
| Process ID | Start Time | End Time |
+------------+------------+----------+
| 1          | 0          | 16       |
| 2          | 4          | 20       |
| 3          | 8          | 24       |
+------------+------------+----------+
```

Figure B.10a Output Interface for MLFQ for 3 tasks with Difference in Arrival Times of 4, Fixed Burst Time of 8 and Quantum of 4

```
Gantt Chart (with time units and queue levels):
+-----------+-------------+-----------+-------------+-----------+-------------+
| RR P1 [4] | FCFS P1 [0] | RR P2 [4] | FCFS P2 [0] | RR P3 [4] | FCFS P3 [12] |
+-----------+-------------+-----------+-------------+-----------+-------------+
0           4             4           8             8           12            24

+------------------+-------------+
| Execution Blocks | Idle Blocks |
+------------------+-------------+
| 6                | 0           |
+------------------+-------------+

Scheduling Calculation Information (in ascending order of completion):
+------------+--------------+------------+-----------------+-----------------+--------------+---------------+
| Process ID | Arrival Time | Burst Time | Completion Time | Turnaround Time | Waiting Time | Response Time |
+------------+--------------+------------+-----------------+-----------------+--------------+---------------+
| 1          | 0            | 8          | 16              | 16              | 8            | 0             |
| 2          | 4            | 8          | 20              | 16              | 8            | 0             |
| 3          | 8            | 8          | 24              | 16              | 8            | 0             |
+------------+--------------+------------+-----------------+-----------------+--------------+---------------+

Scheduling Calculation Performance (in milliseconds):
+-----------------------+----------------------+-------------------------+------------+
| Average Response Time | Average Waiting Time | Average Turnaround Time | Throughput |
+-----------------------+----------------------+-------------------------+------------+
| 0.000                 | 8.000                | 16.000                  | 0.120      |
+-----------------------+----------------------+-------------------------+------------+
```

Figure B.10b Output Interface for MLFQ for 3 tasks with Difference in Arrival Times of 4, Fixed Burst Time of 8 and Quantum of 4

```
                    -------------------- [ Input ] --------------------

Choose the scheduling algorithm:
1. First Come First Served (FCFS)
2. Round Robin (RR)
3. Multi-Level Feedback Queue (MLFQ)
Enter your choice of algorithm:3

Enter the total number of processes:5

Process No. 1:
Enter arrival time:2
 Enter burst time:8

Process No. 2:
Enter arrival time:3
 Enter burst time:9

Process No. 3:
Enter arrival time:1
 Enter burst time:4

Process No. 4:
Enter arrival time:7
 Enter burst time:2

Process No. 5:
Enter arrival time:81
 Enter burst time:21

Enter time quantum:6
```

Figure B.11 Input Interface for MLFQ for 5 tasks with Random Arrival and Burst Times with a Quantum of 6

```
-------------------- [ Output ] --------------------

------------ [ Multi-Level Feedback Queue ] ------------

Initial Queue (with time quantum of 6):
+------------+--------------+------------+
| Process ID | Arrival Time | Burst Time |
+------------+--------------+------------+
| 1          | 2            | 8          |
| 2          | 3            | 9          |
| 3          | 1            | 4          |
| 4          | 7            | 2          |
| 5          | 81           | 21         |
+------------+--------------+------------+

Process Start and End Times (in ascending order of completion):
+------------+------------+----------+
| Process ID | Start Time | End Time |
+------------+------------+----------+
| 3          | 1          | 5        |
| 4          | 17         | 19       |
| 1          | 5          | 21       |
| 2          | 11         | 24       |
| 5          | 81         | 102      |
+------------+------------+----------+
```

Figure B.12a Output Interface for MLFQ for 5 tasks with Random Arrival and Burst Times with a Quantum of 6

```
Gantt Chart (with time units and queue levels):
+----------+----------+-------------+----------+-------------+----------+----------+----------+--------------+
| RR P3 [4] | RR P1 [6] | FCFS P1 [0] | RR P2 [6] | FCFS P2 [0] | RR P4 [7] | Idle [57] | RR P5 [6] | FCFS P5 [15] |
+----------+----------+-------------+----------+-------------+----------+----------+----------+--------------+
1          5          11           11         17           17         24         81         87           102


+------------------+-------------+
| Execution Blocks | Idle Blocks |
+------------------+-------------+
| 8                | 1           |
+------------------+-------------+

Scheduling Calculation Information (in ascending order of completion):
+------------+--------------+------------+-----------------+-----------------+--------------+---------------+
| Process ID | Arrival Time | Burst Time | Completion Time | Turnaround Time | Waiting Time | Response Time |
+------------+--------------+------------+-----------------+-----------------+--------------+---------------+
| 3          | 1            | 4          | 5               | 4               | 0            | 0             |
| 4          | 7            | 2          | 19              | 12              | 10           | 10            |
| 1          | 2            | 8          | 21              | 19              | 11           | 3             |
| 2          | 3            | 9          | 24              | 21              | 12           | 8             |
| 5          | 81           | 21         | 102             | 21              | 0            | 0             |
+------------+--------------+------------+-----------------+-----------------+--------------+---------------+

Scheduling Calculation Performance (in milliseconds):
+----------------------+---------------------+-----------------------+------------+
| Average Response Time | Average Waiting Time | Average Turnaround Time | Throughput |
+----------------------+---------------------+-----------------------+------------+
| 4.200                | 6.600               | 15.400                | 0.049      |
+----------------------+---------------------+-----------------------+------------+
```

Figure B.12b Output Interface for MLFQ for 5 tasks with Random Arrival and Burst Times with a Quantum of 6

8.0 Marking Scheme

Student Name & Stud-Id:	1) Lua Chong En (20417309) – CS 2) Joseph Emmanuel Chay (20580363) – CS 3) Jonathan Rawding (20510088) – CS 4) Jie Zhan Keh (20375609) - CSAI	
Group No:		
Module Title & Code	**Operating Systems and Concurrency (G52OSC)**	
Attributes	Marks Alloted	Marks Awarded
Part A: Software Part: (30%)		

2. Report Part: (20%)		
Total	50	