# 

# 

# 

# CPU Scheduler Project

# Team Stone

CSB340

**Table of Contents:**

1. Introduction
2. Implementation
   1. First Come, First Serve (Chris)
   2. Shortest Job First (Riko)
   3. Priority Scheduling (Yen)
   4. Round Robin (Riko)
   5. Multilevel Queue (Yen)
   6. Multilevel Feedback Queue (Chris)
3. UML diagram
4. Algorithms’ performance measurement and analysis
   1. Data tables for all algorithms
   2. CPU utilization
   3. Waiting times
   4. Turnaround times
   5. Response times
5. Conclusion

### **Introduction**

In this project, we are going to explore six scheduling algorithms: First-Come, First-Served - non-preemptive, Shortest Job First - non-preemptive, Priority Scheduling, Round Robin, Multilevel Queue, and Multilevel Feedback Queue. We will compare their performances based on four criteria: CPU utilization (%), the average waiting time for all processes (seconds), the average turnaround time (seconds), and the average response time (seconds). We will also briefly discuss the implementation of each algorithm and examine their pros and cons.

1. **Implementation**
   1. **First Come, First Serve**

The First-Come, First-Served - non-preemptive (FCFS) algorithm is the simplest CPU scheduling algorithm. As this algorithm is non-preemptive, once the next running process is scheduled it is dispatched and runs until it finishes its CPU burst (before heading to I/O if the process is not complete). As the name suggests, the next running process is scheduled in a first in first out fashion, and so the ready queue is implemented using a simple FIFO Queue. This means that as processes switch context from I/O back to the ready queue, they are added to the tail end of the queue and are scheduled in order of arrival where the next running process gets removed from the ready queue and is (again) dispatched, running until it finishes its CPU burst. This cycle is repeated until all processes are completed. The FCFS algorithm is notorious for long wait times (for ready processes). This disadvantage is known as the convoy effect, where ready processes (that could have a very short CPU burst) must wait for processes with long CPU bursts, which may result in lower CPU and device utilization than if the shorter processes were allowed to go first. Therefore, this algorithm has a simplicity versus performance tradeoff, favoring simplicity over performance.

* 1. **Shortest Job First**

Once a process is dispatched, the Shortest Job First - non-preemptive (SJF) scheduling algorithm will continue running until it has finished its burst cycle. This scheduling algorithm always selects the process with the smallest execution time as the next process to be dispatched. Consequently, this algorithm is considered greedy. However, this approach can lead to process starvation if other processes with shorter burst times keep arriving. To address this issue, the concept of aging can be employed.

Compared to the First Come First Serve algorithm, the Shortest Job First algorithm is advantageous because it reduces the average waiting time. It is particularly well-suited for batch jobs where run times are unknown. However, there are some disadvantages associated with the Shortest Job First algorithm. It can result in very long turnaround times and may also cause starvation.

##### **Priority Scheduling**

##### Priority scheduling algorithm is a CPU scheduling algorithm that assigns priorities to each process based on certain criteria, such as the priority value associated with the process or the type of task it represents. The process with the highest priority is given the highest preference for CPU execution. Though it can be non-preemptive or preemptive, based on the requirements of this project (arrival time is zero for all processes), both will execute the same. While the scheduling algorithm can be beneficial for time-sensitive or critical tasks, it can potentially cause starvation and unfair distribution of resources.

In our implementation, processes with higher priority are executed first until idle time, and processes in the next priority queue are executed until preempted by the previous processes being ready again. The algorithm continues until all processes are executed.

##### **Round Robin (with FCFS)**

##### The Round Robin algorithm is primarily used for time-sharing techniques. In the preemptive method, the allotted time for each process to run is referred to as the time quantum. In this algorithm, all processes in the queue take turns running and each process runs for a maximum of the time quantum before yielding the CPU to the next process. The process may then either go back to the ready queue, enter the I/O state, or complete all cycles.

##### The Round Robin algorithm is simple, easy to implement, and free from starvation since every process in the queue receives a fair share of the CPU. When a new process is added to the queue, it is placed at the end. However, there are a couple of disadvantages associated with the Round Robin algorithm. Waiting times and response times tend to be greater, throughput is lower, and there are more context switches. When scheduling is performed with smaller time quantums, the algorithm can be time-consuming. It is crucial to consider that the dispatcher incurs latency, and the higher number of context switches required, the more affected the overall time will be.

##### **Multilevel Queue**

Multilevel Queue (MLQ) is a scheduling algorithm that uses multiple queues to manage the execution of processes. It involves dividing processes into multiple queues based on different priority levels or characteristics and scheduling them accordingly. Each queue may have its own scheduling algorithm, allowing for different scheduling policies to be applied to different types of processes. In this project, priority queue one is assigned a RR queue with a time quantum of 4 while the second priority queue is assigned FCFS. MLQ aims to provide better resource allocation and responsiveness by prioritizing processes based on their characteristics or priorities.

The disadvantage of MLQ scheduling algorithms is the complexity of managing multiple queues and their associated scheduling policies. It requires careful design and configuration to ensure optimal performance. If the priorities or characteristics assigned to processes are not well-defined or appropriately chosen, it may lead to imbalances in resource allocation and potential starvation of lower-priority processes. Another drawback is that the MLQ algorithm may not adapt well to dynamic changes in workload or resource demands, as it operates on predefined queue structures and scheduling policies.

In this implementation, we build MLQ based on the fixed priority preemptive scheduling method. The processes are first assigned to their predefined priority queues. In the first priority queue, processes are fairly executed based on RR until either they are all processed completely or hit an idle time within this queue. Then, the processes in the second priority queue will be executed based on FCFS. While doing that, if the processes in the first priority queue are ready again (end of the idle time), these processes get to go and the current processes are preempted. MLQ will keep going until all the processes of all queues have been processed completely.

##### **Multilevel Feedback Queue**

##### The Multilevel Feedback Queue (MLFQ) algorithm is the most complex CPU scheduling algorithm. This algorithm is preemptive, so the algorithm facilitates more context switching than non-preemptive algorithms. As the name suggests, the algorithm uses multiple ready queues to schedule processes, where each of the multiple queues are organized by level, signifying scheduling priorities. Also, these queues are dynamic (compared to MLQ) which signifies the feedback property in the name, and means that the processes themselves can move between the levels (changing the priority, which affects scheduling order). This implementation uses two round robin queues and one first-come first-served queue, organized into three levels (all queues use a FIFO queue for implementing ready lists). The highest priority queue uses RR scheduling with a time quantum of five, preceded by another RR queue with a time quantum of ten, and the lowest priority queue uses FCFS scheduling. In this implementation all processes start in the first RR ready queue. If the time quantum expires before the CPU burst is complete in either of the first two queues (by RR preemption) the process is downgraded to the next lower priority queue, and once a process is downgraded, it will not be upgraded. Processes are not, however, downgraded when preempted by a higher queue level process (when coming off an I/O burst). The idea driving the MLFQ algorithm is to separate the processes being scheduled according to the characteristics of their CPU bursts. So, when a process uses too much CPU time, it will be moved to a lower priority queue. Therefore, this algorithm (also) has a simplicity versus performance tradeoff, favoring performance over simplicity.

### **UML diagram**

### 

### **Algorithms’ performance measurement and analysis**

##### **Data tables for all algorithms**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | FCFS - NonPre | SJF - NonPre | Priority | RR | MLQ | MLFQ |
| CPU utilization | 85.34% | 82.78% | 77.98% | 92.94% | 95.84% | 93.73% |
| Avg Waiting time (Tw) | 185.25 | 133.5 | 150.5 | 175.5 | 148 | 154 |
| Avg Turnaround time (Ttr) | 521.37 | 469.62 | 486.63 | 511.63 | 484.13 | 490.13 |
| Avg Response time (Tr) | 24.37 | 27.13 | 53.25 | 15.75 | 24.63 | 15.75 |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | FCFS  CPU utilization: | | | SJF NonPre  CPU utilization: | | | Priority  CPU utilization: | | |
|  | 85.34% | | | 82.78% | | | 77.98% | | |
|  | Tw | Ttr | Tr | Tw | Ttr | Tr | Tw | Ttr | Tr |
| P1 | 170 | 395 | 0 | 43 | 268 | 11 | 73 | 298 | 27 |
| P2 | 164 | 591 | 5 | 73 | 500 | 3 | 197 | 624 | 67 |
| P3 | 165 | 557 | 9 | 276 | 668 | 16 | 157 | 549 | 35 |
| P4 | 164 | 648 | 17 | 50 | 534 | 0 | 97 | 581 | 32 |
| P5 | 221 | 530 | 20 | 237 | 546 | 109 | 0 | 309 | 0 |
| P6 | 230 | 445 | 36 | 121 | 336 | 24 | 50 | 265 | 16 |
| P7 | 184 | 512 | 47 | 149 | 477 | 47 | 381 | 709 | 158 |
| P8 | 184 | 493 | 61 | 119 | 428 | 7 | 249 | 558 | 91 |
| **Avg** | **185.25** | **521.38** | **24.38** | **133.50** | **469.63** | **27.13** | **150.50** | **486.63** | **53.25** |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | RR CPU utilization: | | | MLQ CPU utilization: | | | MLFQ CPU utilization: | | |
|  | 92.94% | | | 95.84 | | | 93.73% | | |
|  | Tw | Ttr | Tr | Tw | Ttr | Tr | Tw | Ttr | Tr |
| P1 | 107 | 332 | 0 | 15 | 240 | 0 | 50 | 275 | 0 |
| P2 | 125 | 552 | 5 | 35 | 462 | 4 | 134 | 561 | 5 |
| P3 | 203 | 595 | 9 | 57 | 449 | 8 | 198 | 590 | 9 |
| P4 | 88 | 572 | 14 | 17 | 501 | 12 | 17 | 501 | 14 |
| P5 | 255 | 564 | 17 | 268 | 577 | 20 | 256 | 565 | 17 |
| P6 | 206 | 421 | 22 | 325 | 540 | 36 | 179 | 394 | 22 |
| P7 | 249 | 577 | 27 | 244 | 572 | 46 | 247 | 575 | 27 |
| P8 | 171 | 480 | 32 | 223 | 532 | 71 | 151 | 460 | 32 |
| **Avg** | **175.50** | **511.63** | **15.75** | **148.00** | **484.13** | **24.63** | **154.00** | **490.13** | **15.75** |

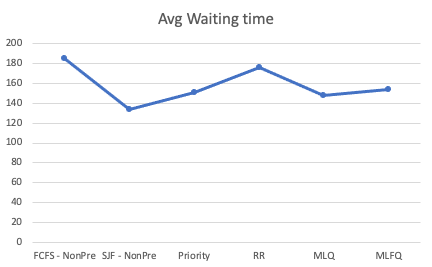
##### **CPU utilization**

##### 

##### CPU utilization is the measurement of the percentage of time that the CPU is actively executing processes. The higher CPU utilization indicates the more effective the algorithm is. When comparing the CPU utilization of different scheduling algorithms, MLQ and MLFQ scheduling algorithms exhibit the highest CPU utilization among the listed algorithms, with a utilization of 95.84% and 93.73%, respectively. This is because MLQ and MLFQ adapt to the workload using multiple queues and different scheduling policies. It dynamically adjusts priorities and allocates CPU time, resulting in efficient utilization. Round-Robin (RR) scheduling follows with a CPU utilization of 92.94%, showcasing its fairness in allocating CPU time among processes.

The utilization rates for the remaining algorithms are as follows: FCFS - 85.34%, SJF - 82.78%, and Priority - 77.98%. FCFS scheduling operates based on the arrival order of processes. While it is simple, it may not achieve high CPU utilization in scenarios where shorter processes are waiting behind long-running ones. SJF scheduling focuses on minimizing waiting time by prioritizing processes with the shortest burst time. While it optimizes waiting time, it may not result in the highest CPU utilization if long processes are scheduled first, causing idle CPU time. Priority scheduling assigns priorities to processes, and the CPU utilization depends on the distribution and dynamics of priorities. If high-priority processes frequently arrive, the CPU utilization can be high, but if there are only a few high-priority processes, the utilization may be lower.

* 1. **Waiting times**

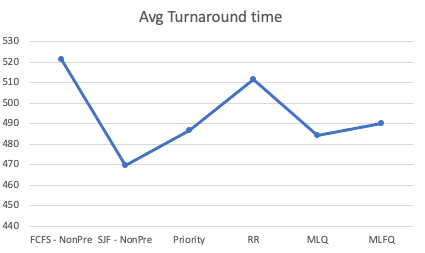


The waiting time is the process’s duration in the ready queue before it begins execution. When considering the average waiting time performance of the scheduling algorithms, SJF scheduling algorithm achieves the lowest average waiting time of 133.50, which suggests that it prioritizes shorter job durations and minimizes waiting times. This is because SJF allows smaller processes to be executed first. This results in a lower average waiting time, as shorter jobs are completed sooner, reducing the time other processes must wait.

The MLQ, Priority, and MLFQ scheduling algorithm follows with an average waiting time of 148, 150.50 and 154, respectively. Priority scheduling assigns priorities to processes based on predefined criteria. Processes with higher priority are given preference, leading to a relatively lower average waiting time for those processes. However, if there is a significant difference in priority levels, lower priority processes may experience longer waiting times. MLFQ and MLQ assign processes to multiple queues based on their characteristics. While MLFQ allows processes to move between queues based on predefined criteria, MLQ prioritizes their processes in their preassigned order. Both MLQ and MLFQ allows processes to be preempted to reduce waiting time to the minimum. MLFQ may have a higher average waiting time compared to others due to the complexity of managing multiple queues and potential process migrations between them.

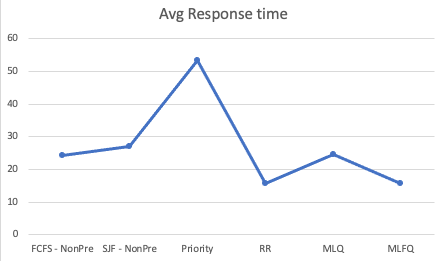
RR and FCFS scheduling algorithms exhibit an average waiting time of 175.50 and 185.25, respectively, indicating that processes experience longer waiting times before execution. RR scheduling allocates a fixed time slice to each process before moving to the next. This can lead to a higher average waiting time compared to SJF and Priority, as processes may need to wait for their turn to receive CPU time, especially if the time slice is relatively large. FCFS scheduling processes in the order of their arrival can result in a higher average waiting time. If a long process arrives first, subsequent processes must wait until it completes, leading to increased waiting times for other processes in the queue.

* 1. **Turnaround Times**



The turnaround time is the time elapsed between the arrival of a process and its completion. Among the algorithms compared, SJF exhibits the lowest average turnaround time of 469.62. This implies that SJF can prioritize shorter job bursts, resulting in quicker process completion and turnaround. Following closely is MLQ, Priority, and MLFQ algorithms with an average turnaround time of 484.13, 486.63, and 490.13, respectively, indicating slightly longer process completion times compared to SJF - NonPre. In Priority scheduling, processes are assigned priorities, and those with higher priorities are executed first. This can result in slightly longer turnaround times compared to SJF if processes with higher priorities have longer burst times. MLQ and MLFQ though can manage multiple queues efficiently, they can have longer turnaround time due to ineffective queue management. For example, with MLFQ, processes with long burst time are pushed to the second priority queue, resulting in longer waiting and execution time. The same issue could happen to MLQ when processes are initially assigned to -not-optimized priority queues.

RR and FCFS exhibit a longer average turnaround time of 511.63 and 521.37, indicating a comparatively slower process completion rate. In RR, processes are allocated a fixed time quantum for execution before being preempted. This can increase delays due to frequent processes switching, resulting in longer turnaround times. First-Come, First-Served scheduling executes processes in the order they arrive, without considering their burst times. This lack of prioritization can lead to longer waiting times for processes, resulting in a higher average turnaround time.

* 1. **Response Times**  
       
       
     

The response time is the amount of time it takes for the CPU to respond to a process. When examining the average response times of the scheduling algorithms, we can observe different patterns. RR and MLFQ algorithms have the lowest average response times at 15.75. This can be attributed to their preemptive nature and the allocation of equal time slices or multiple priority queues, respectively. Following closely is FCFS and MLQ algorithms with an average response time of 24.37 and 24.63, respectively. These numbers are expected as FCFS operates based on the order of process arrival and MLQ operates based on multiple queues including a priority queue. MLQ has a longer waiting time than MLFQ is probably because processes with longer burst time are assigned in the higher priority queue so that it delays the response time for the processes in the lower priority queue. SJF and priority algorithms have a higher average response time of 27.13, and 53.25, respectively. This is expected because SJF focuses on the shortest job first which will delay response time for the longer process. Priority algorithm focuses on a predefined set of priorities which cause delay response time for the process on a lower priority queue. These results emphasize the influence of scheduling policies and algorithm characteristics on the average response times.

1. **Conclusion**

Based on the results, we can see each algorithm has its own strengths and weaknesses based on the four criteria. The MLQ and MLFQ algorithms stand out with the most efficient CPU utilization, and they are also among the ones that have low Tw, Tt, and Tr. Between these two, MLFQ performs slightly better on the response time. It is also worth pointing out that SJF has the best performance in Ttr and Tw but not so much in CPU utilization and Tr. Overall, it’s important to consider the workload characteristics and system requirements when selecting the most suitable scheduling algorithm, whether it is maximizing CPU utilization, minimizing waiting time, or achieving optimal turnaround or response times.