Dov Cattan

Computer Operating Systems/ COP 4610

CPU Scheduler

Table of Contents

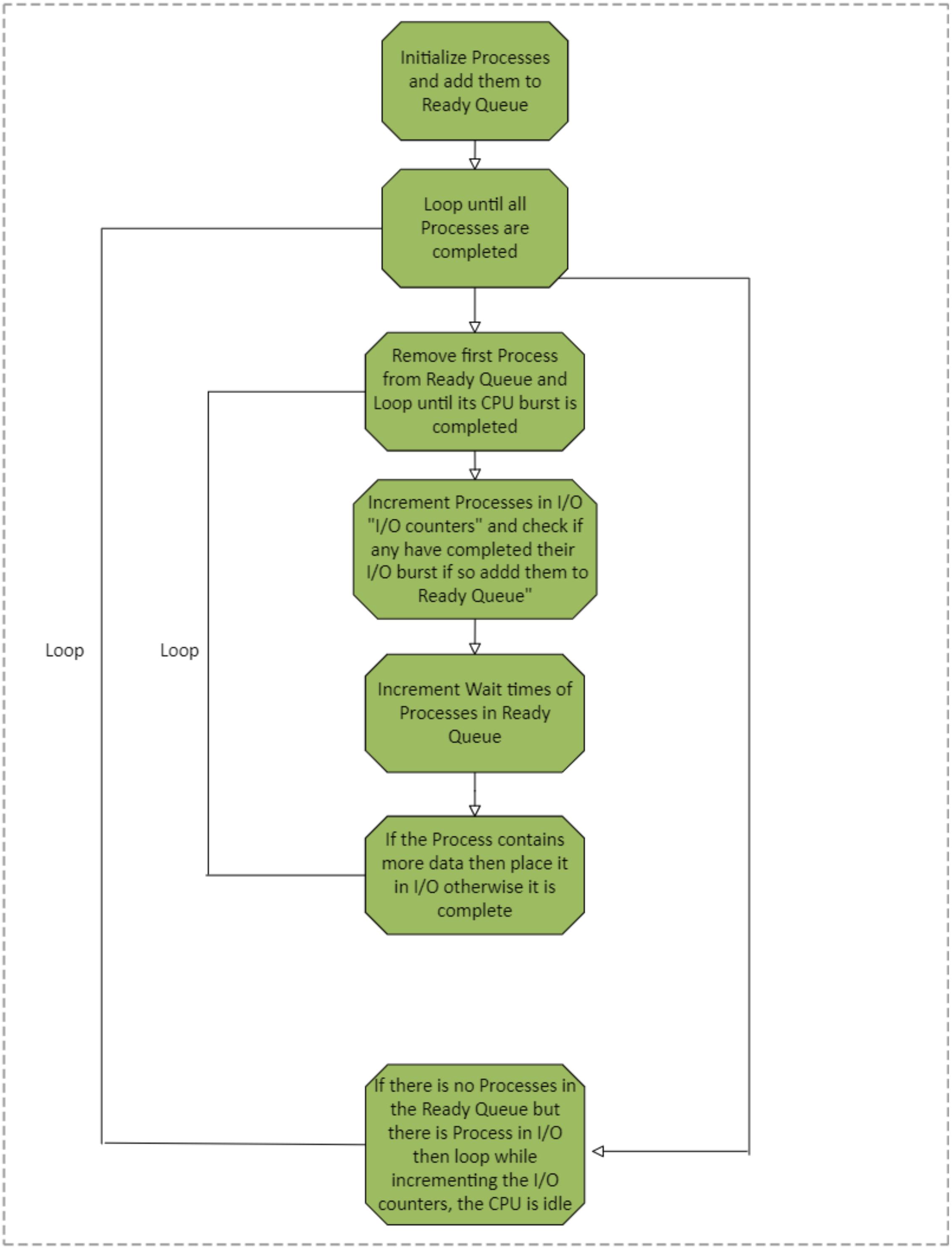
Introduction

For this project I implemented three different CPU Scheduler algorithms. The algorithms consisted of FCFS (First Come First Serve), SJF (Shortest Job First), and MLFQ (Multi Level Feedback Queue). Through the simulation programs I built I was able to calculate the average wait time, turnaround time, CPU utilization, and response time. After each Process completes a CPU burst, it then does an I/O burst and this will also factor into the results. The final results of the scheduling algorithms can be used to conclude which of the three algorithms is the most efficient.

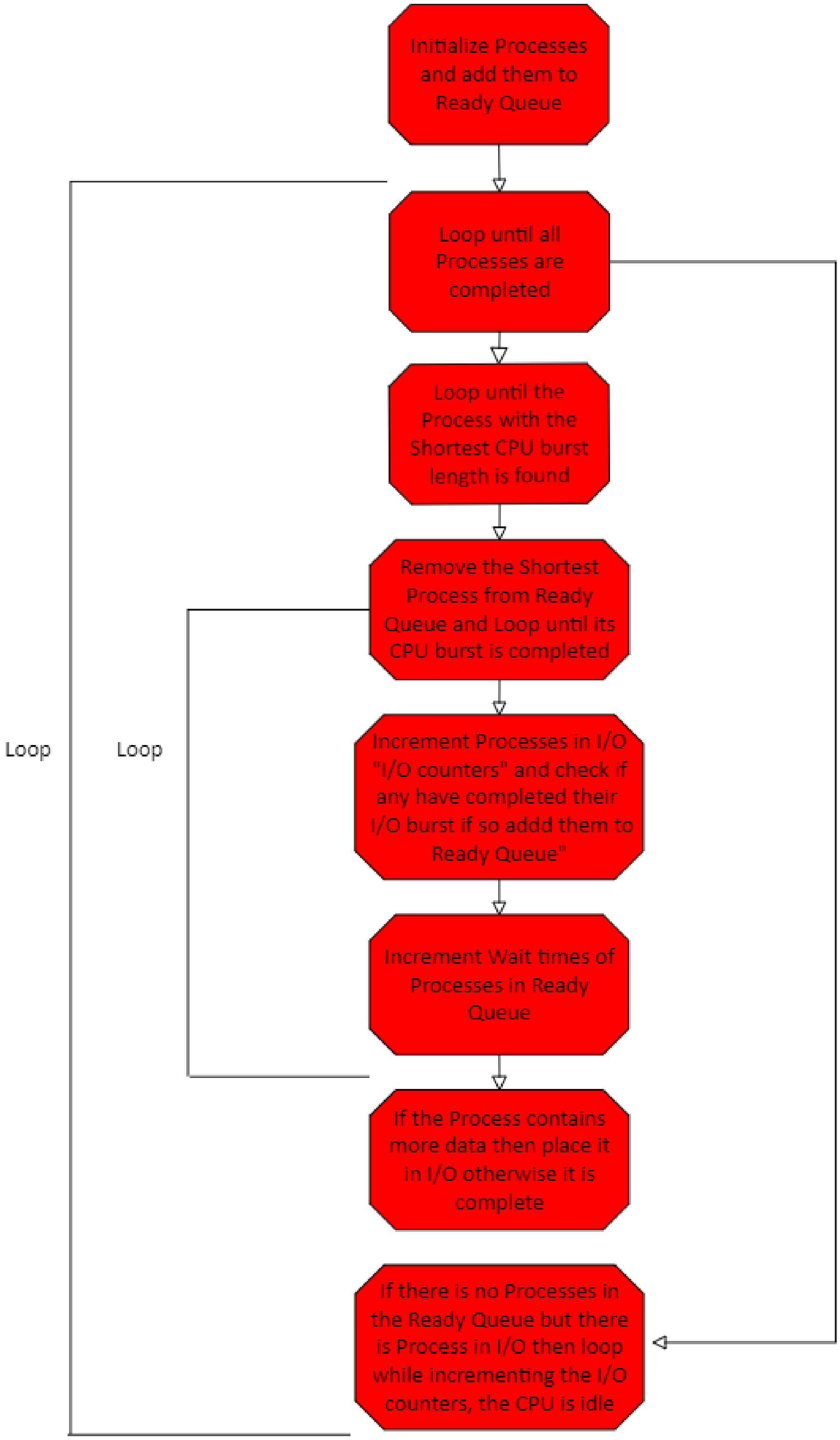
Each algorithm has different criteria for selecting the Process to execute and for how long to allow it to execute. FCFS selects the first Process in the Ready Queue and executes the process for the full length of its CPU burst. SJF selects the Process in the Ready Queue with the smallest CPU burst and also executes the process for the full length of its CPU burst. MLFQ consists of three different queue's each with different priority levels and scheduling algorithms. Queue 1 uses round robin with a time quantum of 6 as its algorithm and has the highest priority level. Queue2 also uses round robin with a time quantum of 11 as its algorithm and has the second highest priority level. Queue3 uses FCFS as its algorithm and has the lowest priority level. MLFQ also uses preemption in order to stop a execute a Process in a higher priority queue halting the execution of Processes in lower queues.

I used C++ as the programming language for the simulation programs. I used sort algorithms with a C++ library for all of the scheduling Processes. I called all of the processes. This Scheduler ensures that all processes are deleted rather than terminated if all bursts are complete. All the options of which Scheduler to use in the Simulator are forced to the right acronym by having the user type the right string.

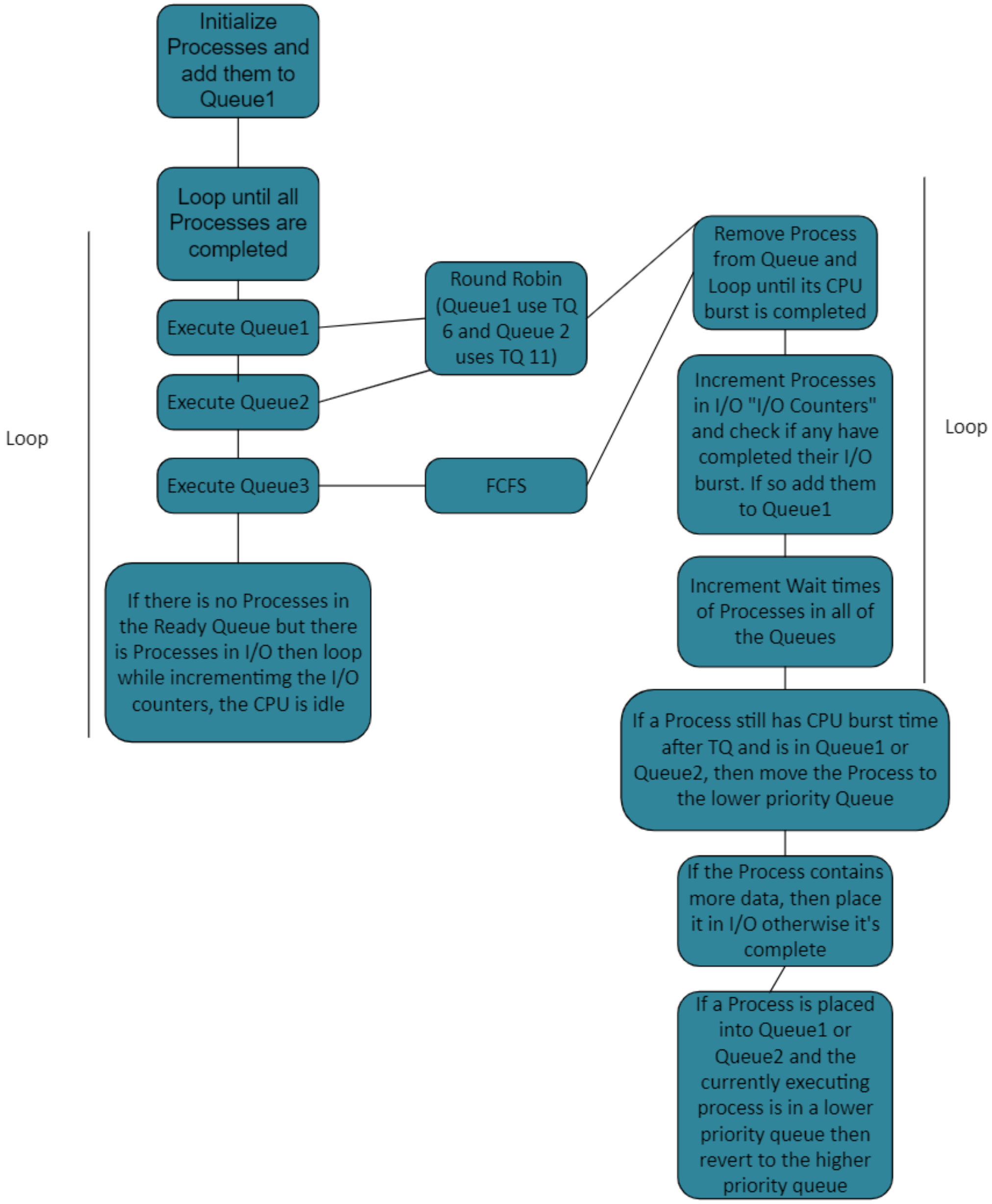
FCFS Logic Flow Chart


Initialize Processes and add them to Ready Queue
Loop until all Processes are completed
Remove first Process from Ready Queue and Loop until its CPU burst is completed
Increment Processes in I/O "I/O counters" and check if any have completed their I/O burst if so addd them to Ready Queue"
Increment Wait times of Processes in Ready Queue
If the Process contains more data then place it in I/O otherwise it is complete
If there is no Processes in the Ready Queue but there is Process in I/O then loop while incrementing the I/O counters, the CPU is idle
Loop
Loop

SJF Logic Flow Chart


Initialize Processes and add them to Ready Queue
Loop until all Processes are completed
Loop until the Process with the Shortest CPU burst length is found
Remove the Shortest Process from Ready Queue and Loop until its CPU burst is completed
Increment Processes in I/O "I/O counters" and check if any have completed their I/O burst if so addd them to Ready Queue"
Increment Wait times of Processes in Ready Queue
If the Process contains more data then place it in I/O otherwise it is complete
If there is no Processes in the Ready Queue but there is Process in I/O then loop while incrementing the I/O counters, the CPU is idle
Loop
Loop

MLFQ Logic Flow Chart


Initialize Processes and add them to Queue1
Loop until all Processes are completed
Execute Queue1
Execute Queue2
Execute Queue3
If there is no Processes in the Ready Queue but there is Processes in I/O then loop while incrementimg the I/O counters, the CPU is idle
Loop
Round Robin (Queue1 use TQ 6 and Queue 2 uses TQ 11)
FCFS
Remove Process from Queue and Loop until its CPU burst is completed
Increment Processes in I/O "I/O Counters" and check if any have completed their I/O burst. If so add them to Queue1
Increment Wait times of Processes in all of the Queues
If a Process still has CPU burst time after TQ and is in Queue1 or Queue2, then move the Process to the lower priority Queue
If the Process contains more data, then place it in I/O otherwise it's complete
If a Process is placed into Queue1 or Queue2 and the currently executing process is in a lower priority queue then revert to the higher priority queue
Loop

Tables and Discussion

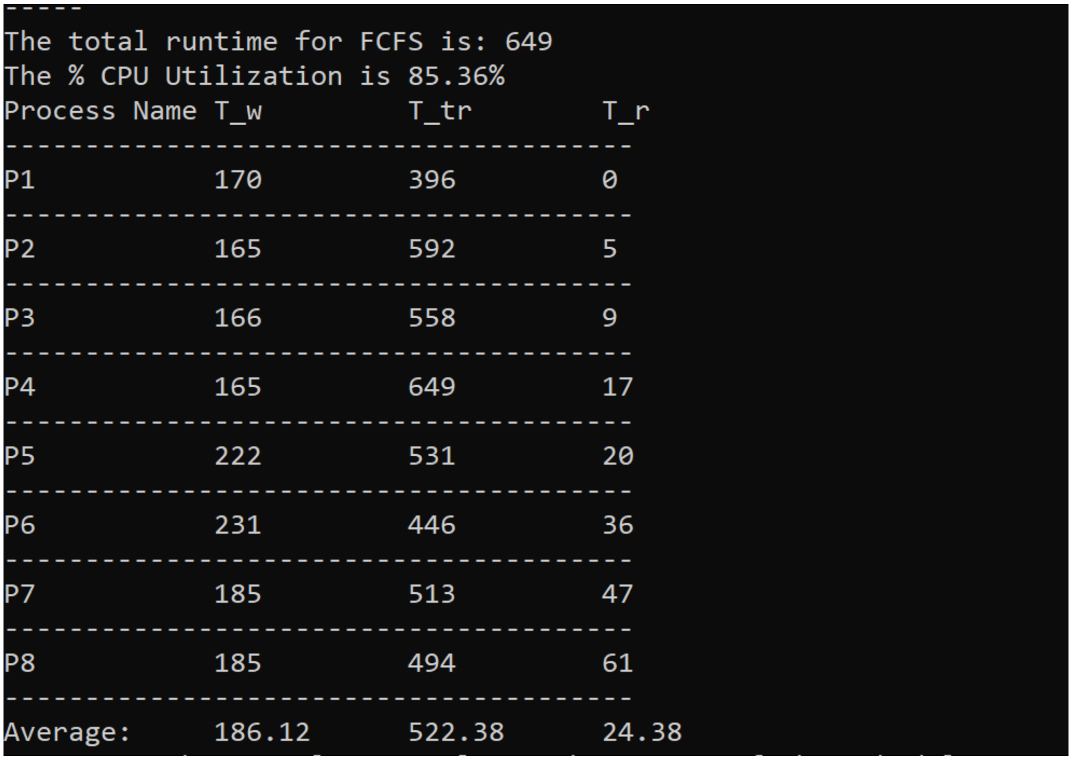
	SJF	**FCFS**	**MLFQ**
CPU utilization	82.81%	85.36%	89.35%
Avg Waiting time (Tw)	134.38	186.12	166.38
Avg Turnaround time (Tr)	470.62	522.38	502.62
Avg Response time (Tr)	27.12	24.38	15.75

	SJF CPU utilization: 82.81%				**FCFS CPU utilization:** 85.36%				**MLFQ CPU utilization:** 89.35%			
	Tw	***Ttr***	***Tr***		***Tw***	***Ttr***	***Tr***		***Tw***	***Ttr***	***Tr***	
P1	43	269	11		170	396	0		40	266	0	
P2	74	501	3		165	592	5		148	575	5	
P3	277	669	16		166	558	9		228	620	9	
P4	51	535	0		165	649	17		43	527	14	
P5	238	547	109		222	531	20		284	593	17	
P6	122	337	24		231	446	36		182	397	22	
P7	150	478	47		195	513	47		209	537	27	
P8	120	429	7		185	494	61		197	506	32	
Avg	***134.38***	***470.62***	***27.12***		***186.12***	***522.38***	***24.38***		***166.38***	***502.62***	***15.75***	

From analyzing the data resulting from the simulation programs, varying observations can be made. SJF has the smallest average waiting time and the smallest average turnaround time compared to the other algorithms. FCFS has a very high average wait time of 186.12 and average turnaround time of 522.38 which are relatively high compared to the other algorithms. FCFS is almost 52 milliseconds higher than SJF when it comes to average turnaround time. MLFQ does happen to have close results to SJF in terms of average wait time. When it comes to average response time, MLFQ has the best results followed by FCFS. SJF has the worst results having an average of 27.12. An observation can be made that MLFQ is the most efficient by comparing most of the results with the other two algorithms. Since the MLFQ algorithm does have the highest degree of degree of complexity of the three algorithms, its efficiency is not a surprise.

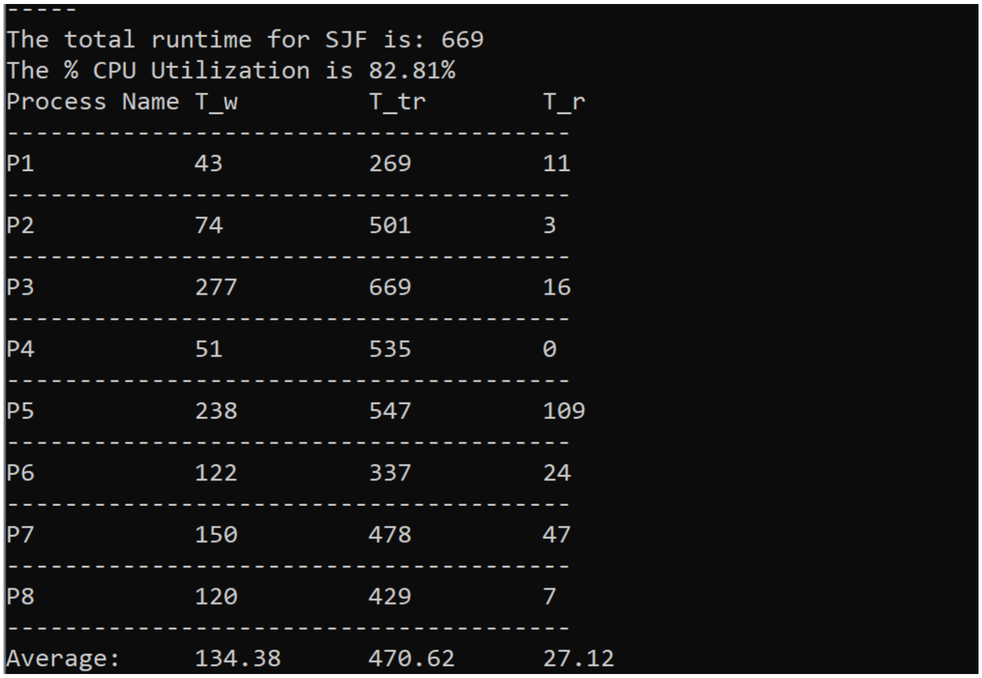
End of Simulation Results Output

FCFS



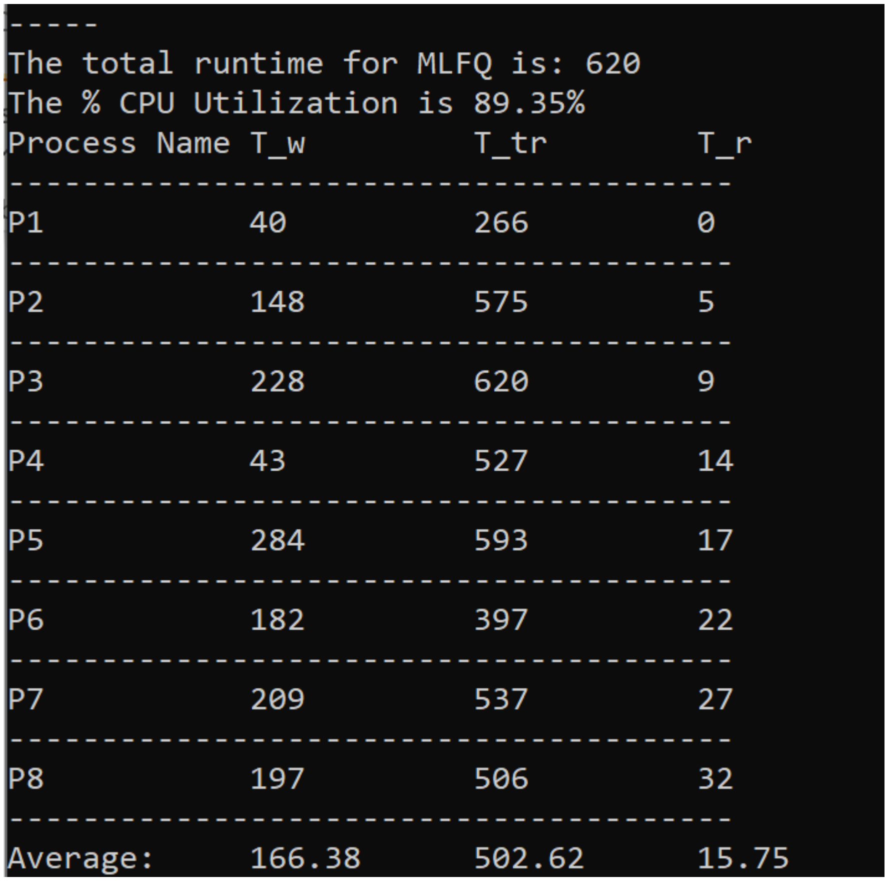
```
-----
The total runtime for FCFS is: 649
The % CPU Utilization is 85.36%
Process Name T_w          T_tr         T_r
----------------------------------------
P1           170          396          0
----------------------------------------
P2           165          592          5
----------------------------------------
P3           166          558          9
----------------------------------------
P4           165          649          17
----------------------------------------
P5           222          531          20
----------------------------------------
P6           231          446          36
----------------------------------------
P7           185          513          47
----------------------------------------
P8           185          494          61
----------------------------------------
Average:     186.12       522.38       24.38
```

SJF



```
-----
The total runtime for SJF is: 669
The % CPU Utilization is 82.81%
Process Name T_w          T_tr         T_r
----------------------------------------
P1           43           269          11
----------------------------------------
P2           74           501          3
----------------------------------------
P3           277          669          16
----------------------------------------
P4           51           535          0
----------------------------------------
P5           238          547          109
----------------------------------------
P6           122          337          24
----------------------------------------
P7           150          478          47
----------------------------------------
P8           120          429          7
----------------------------------------
Average:     134.38       470.62       27.12
```

MLFQ



```
-----
The total runtime for MLFQ is: 620
The % CPU Utilization is 89.35%
Process Name T_w         T_tr        T_r
---------------------------------------
P1           40          266         0
---------------------------------------
P2           148         575         5
---------------------------------------
P3           228         620         9
---------------------------------------
P4           43          527         14
---------------------------------------
P5           284         593         17
---------------------------------------
P6           182         397         22
---------------------------------------
P7           209         537         27
---------------------------------------
P8           197         506         32
---------------------------------------
Average:     166.38      502.62      15.75
```

Source Code

All the Source Code is attached in the Zip File. This includes the header, implementation code, and driver code.

Dynamic Output

All of the Dynamic Output is attached to the Zip File in a Text File For FCFS, SJF, and MLFQ