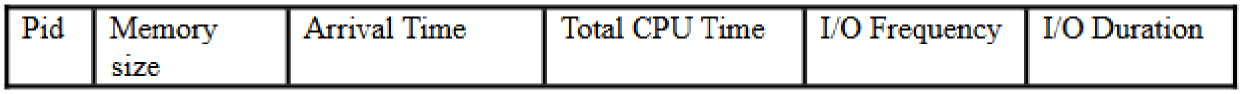
<u>**SYSC 4001 A3:** Part 1 Report</u>

Github Link: https://github.com/Raghav3141/SYSC4001_A3_P1
Student 1: Hariharan Thennarasu, 101304027
Student 2: Raghav Ramaswamy, 101310114

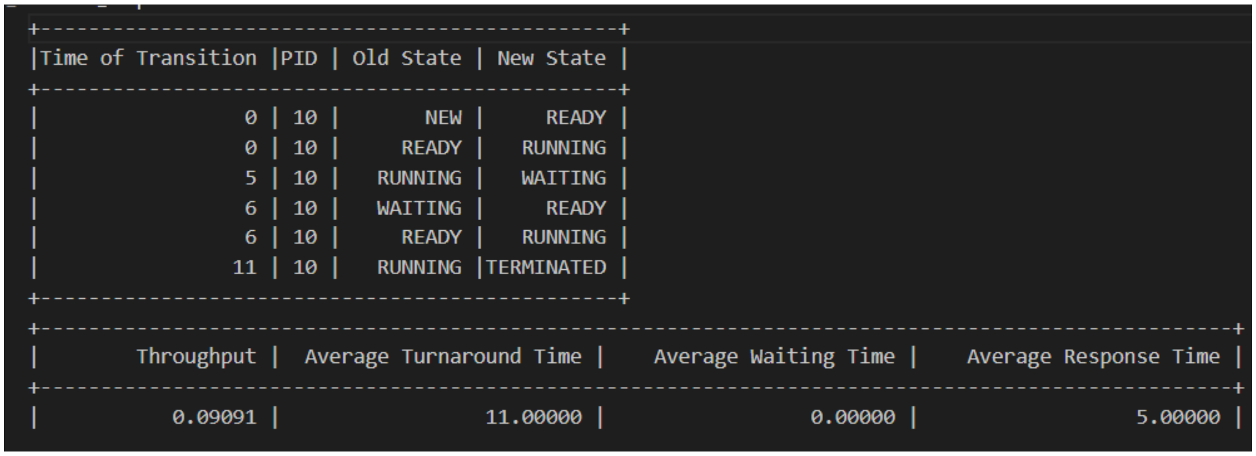
Introduction:
The purpose of this report is to evaluate the performance of different scheduling algorithms by creating a simulator and using the simulator to measure metrics such as average turnaround time, waiting time, response time, and throughput. Then the external priority (EP), round robin (RR), and a combination of the two are executed under different conditions to see how well each scheduler performs compared to each other.

Implementation:
To implement this experiment, 3 programs were created, each representing the EP, RR, and combination scheduling algorithms. Those programs were then given an input in the following form:



Pid	Memory size	Arrival Time	Total CPU Time	I/O Frequency	I/O Duration

Once given the input, the programs generated an execution output file in the following form:



```
+---------------------------------------------+
|Time of Transition |PID | Old State | New State |
+---------------------------------------------+
|                 0 | 10 |       NEW |     READY |
|                 0 | 10 |     READY |   RUNNING |
|                 5 | 10 |   RUNNING |   WAITING |
|                 6 | 10 |   WAITING |     READY |
|                 6 | 10 |     READY |   RUNNING |
|                11 | 10 |   RUNNING |TERMINATED |
+---------------------------------------------+
+------------------------------------------------------------------------------------------+
|       Throughput |  Average Turnaround Time |    Average Waiting Time |    Average Response Time |
+------------------------------------------------------------------------------------------+
|          0.09091 |                 11.00000 |                 0.00000 |                  5.00000 |
```

From above it is seen that the scheduler saves and outputs whenever a process shifts from NEW, READY, RUNNING, WAITING, and TERMINATED, and prints that data to the execution file. In order to properly assess the performance of the different scheduling algorithms in different conditions, 3 main types of test cases were used: CPU-heavy inputs, I/O-heavy inputs, and a balance of CPU and I/O inputs.

For this experiment 20 test cases were performed but for the sake of keeping this report brief a random test case will be chosen from the 3 types of test cases (CPU-heavy, I/O-heavy, and the combination of the two).

Test Cases:
Below are tables displaying the data for each scheduler for each type of test case:

CPU-heavy test case

	Throughput	**Average Turnaround Time**	**Average Waiting Time**	**Average Response Time**
EP	0.00120	1383.33337	550.00000	0
RR	0.00120	1850.00000	1016.66669	0
EP & RR	0.00120	1383.33337	550.00000	0

I/O-heavy test case

	Throughput	**Average Turnaround Time**	**Average Waiting Time**	**Average Response Time**
EP	0.00227	1115.00000	177.33333	7.33333
RR	0.00226	1163.33337	225.66667	7.33333
EP & RR	0.00208	1120.66663	183.00000	7.33333

Combination test case

	Throughput	**Average Turnaround Time**	**Average Waiting Time**	**Average Response Time**
EP	0.01429	178.00000	76.00000	14.00000
RR	0.01429	229.75000	127.75000	14.00000
EP & RR	0.01270	176.75000	74.75000	14.00000

Analysis:
When viewing the data, there are some points of note. For the CPU-bound test case, the EP and the EP & RR implementations have the exact same values. This makes sense as the only difference between the two is that the EP & RR implementation preempts the system if a process arrives later but has a lower PID than the current running process. The RR implementation is clearly the worst out of all the scheduling algorithms when viewing the metrics shown, it does the worst in the CPU-bound test case, the I/O-bound test case, and the combination test case. This is because RR focuses on the "fairness" of the system, so while each process gets a chance to run, they will take a while to complete because they keep getting interrupted (this is especially true for large CPU-bound processes). For the I/O test case all the algorithms had a relatively similar performance, with EP being the best due to the fact that for EP when an I/O occurs the next process runs until it terminates or until it receives an I/O (because it's non-preemptive). EP & RR on the other hand performs an I/O on the current running process, and then when it leaves the wait queue it can interrupt and kick out the next running process (because it is preemptive). This most likely makes it slightly more difficult for processes to reach completion in EP & RR compared to EP. The combination test case also had close performances, with EP & RR having the best overall metrics. This is because EP & RR is good for general test cases since its main goal is to complete the highest priority task as soon as possible. This allows it to for the most part focus on processes one by one as they complete which makes it more efficient in that case.

There are also a few things of note: the way the EP & RR is designed basically ensures that the RR does not contribute to the scheduling at all. This is because the moment a process is kicked out the scheduler runs and since that process has the lowest PID it just gets placed at the front of the queue and continues running. Another thing to note is that for the CPU-heavy and I/O heavy test cases the processes were quite long, which made the RR very inefficient. More test cases could be used with different process durations to view how that changes the metrics.