**Operating Systems and Concurrency (COMP2035)**

**Coursework name**: Operating Systems and Concurrency

|  |  |
| --- | --- |
| **Student name** | **Student ID** |
| LOH JIN XIAN | 016763 |
| TAN KHENG HAU | 014719 |

**Table of Content**

1. Introduction
2. Feedbacks from experiment

2.1 Same workload

2.1.1 Average turnaround time

2.1.2 Average waiting time

2.1.3 CPU usage

2.1.4 Comparison

2.2 Different workload

2.2.1 Average turnaround time, Average waiting time, CPU usage

* First come first served
* Shortest job first
* Round robin
* Priority

2.2.2 Comparison

1. Scheduling algorithm suitability
2. Questions to answer

* What happens when a process returns from I/O
* What happens when a new process is created
* When / how often the scheduling takes place

1. Conclusion
2. References
3. **Introduction**

An operating system contains many different types of scheduling algorithms. Scheduling algorithms are used by process scheduler to determine which process in the ready state should be moved to the running state. The main goal of a scheduling algorithm is to keep the CPU busy all the time and to deliver minimum response time for all programs whenever possible [1].

Scheduling algorithms can be divided into 2 categories which are non-preemptive and preemptive:

**1. Non-preemptive**

designed so that a process that is currently in the running state cannot be pre-empted until its allotted time is completed.

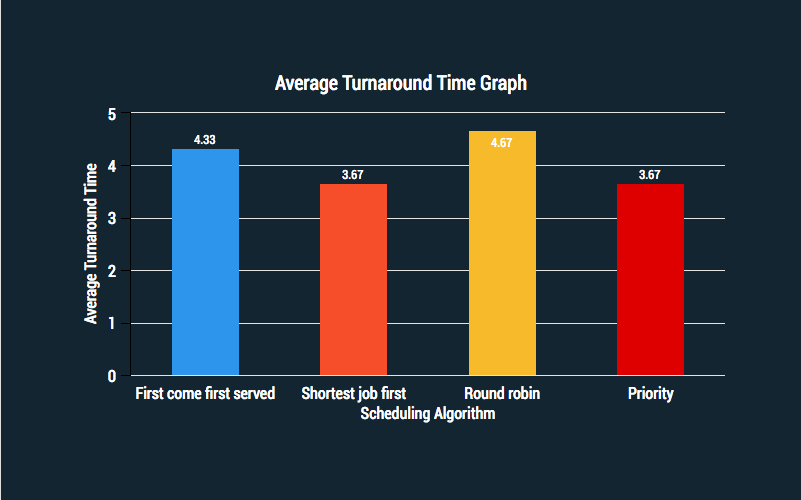
**2. Preemptive**

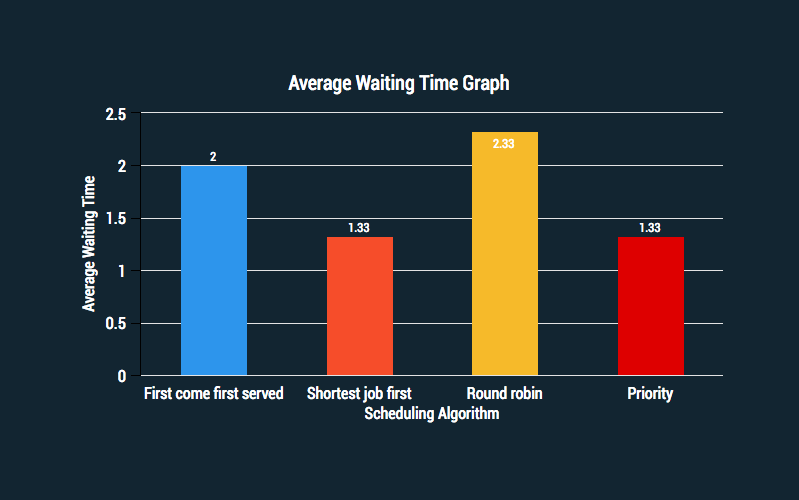
designed so that a low priority running process may be pre-empted if there's a high priority process in the ready state

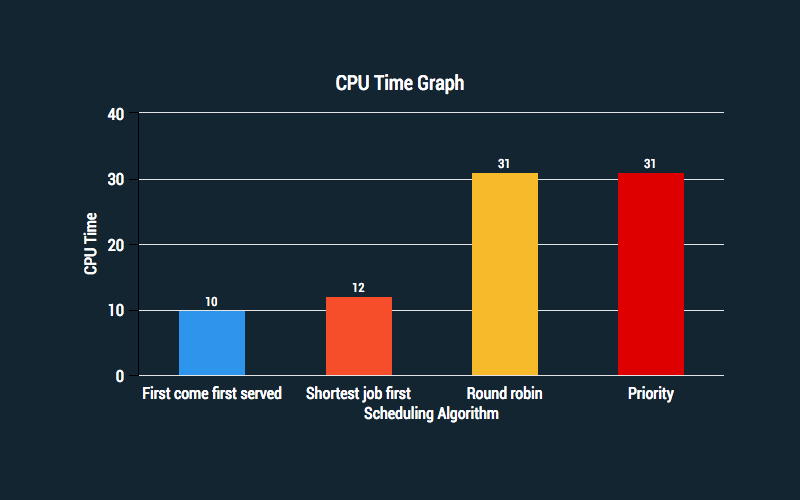
1. **Feedbacks from experiment**
   1. **Same workload**

The following results are obtained by performing the same workload on all scheduling algorithms. All results will be represented in a bar chart graph for better clarity. Comparison between all scheduling algorithms with their results is provided.

|  |
| --- |
| **Input entered** |
| Number of processes: 3 |
| Burst time for process 1: 1 |
| Burst time for process 2: 4 |
| Burst time for process 3: 2 |
| Time quantum for RR: 3 |

* + 1. **Average Turnaround Time**
    2. **Average Waiting Time**

****

* + 1. **CPU usage**
    2. **Comparison**

After running all scheduling algorithms with the same workload, we can conclude that Shortest Job First (preemptive version) and priority scheduling algorithm have the shortest average turnaround time. As for average waiting time, once again both SFJ (preemptive version) and priority scheduling algorithm have the shortest time. Lastly, first come first served (FCFS) scheduling algorithm has the shortest CPU usage compared to the rest with the given workload.

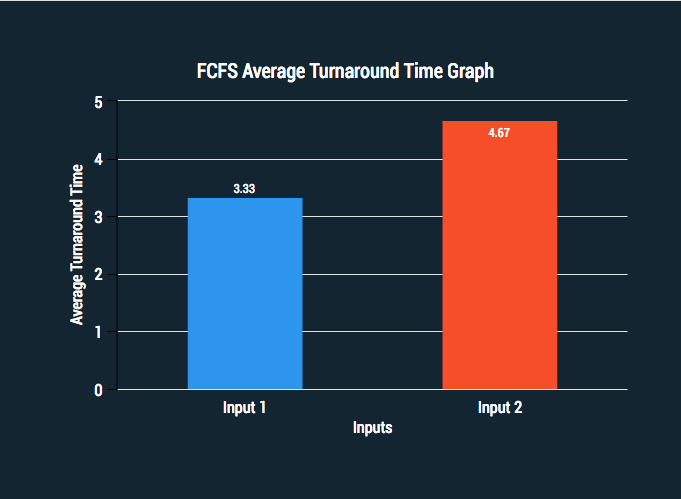
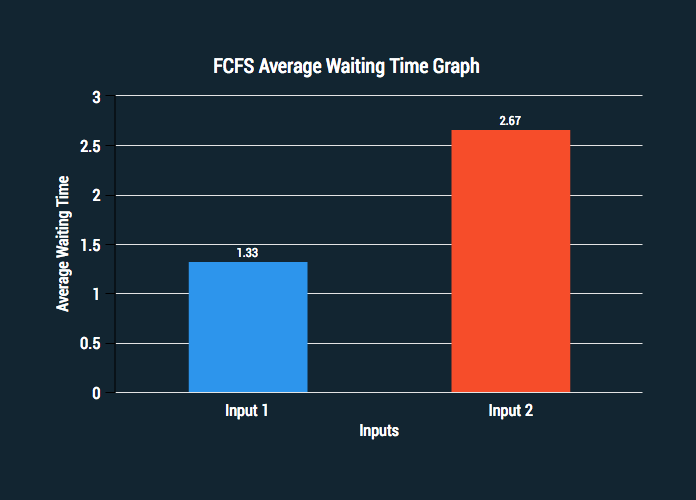
* 1. **Different workload**

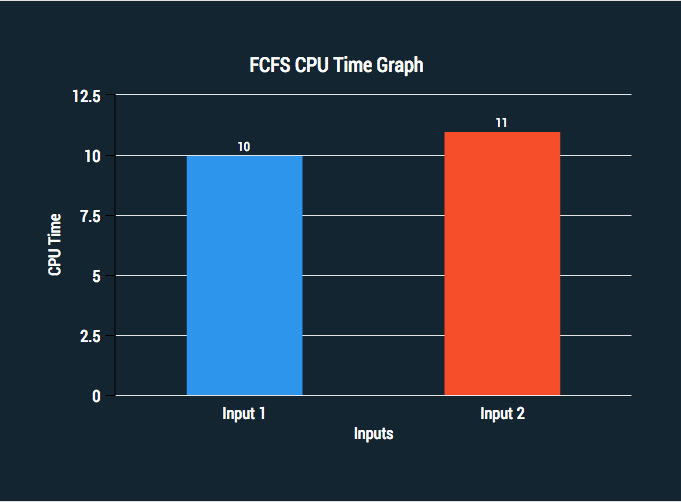
The following results are obtained by performing different workload on all scheduling algorithms. All of them are tested with 2 different inputs to show what is the advantages and disadvantages of that scheduling algorithm. All results will also be represented in a bar chart graph. Then a comparison between each individual algorithm is provided.

* + 1. **Average Turnaround Time, Waiting time, CPU usage**

**First come first served**

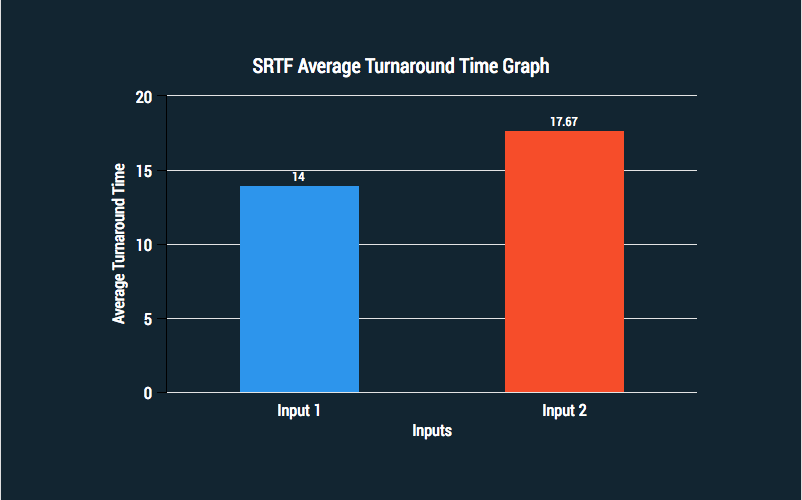
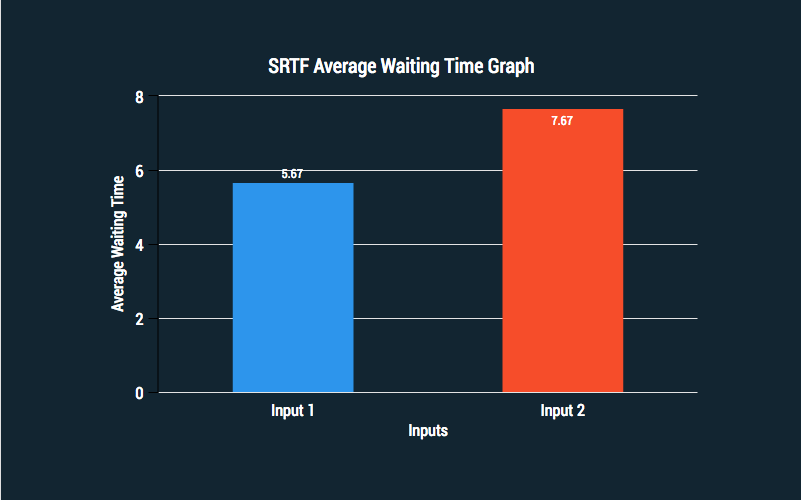
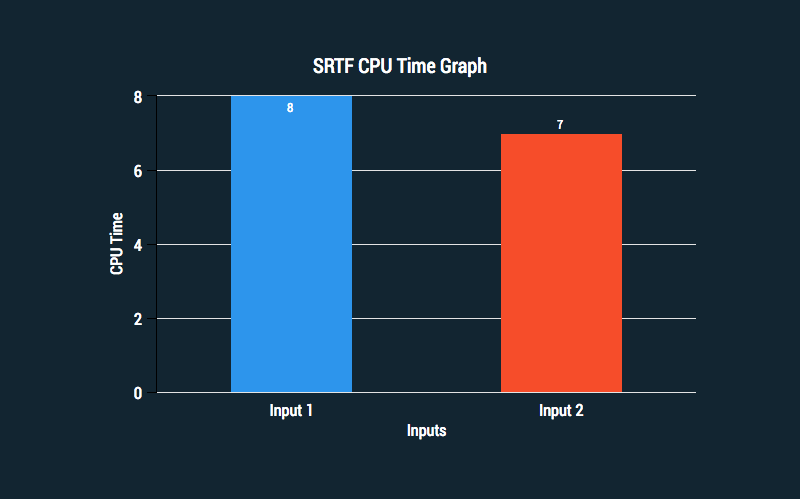
|  |  |
| --- | --- |
| **Input 1** | **Input 2** |
| Number of processes: 3 | Number of processes: 3 |
| Burst time for process 1: 1 | Burst time for process 1: 3 |
| Burst time for process 2: 2 | Burst time for process 2: 2 |
| Burst time for process 3: 3 | Burst time for process 3: 1 |

****



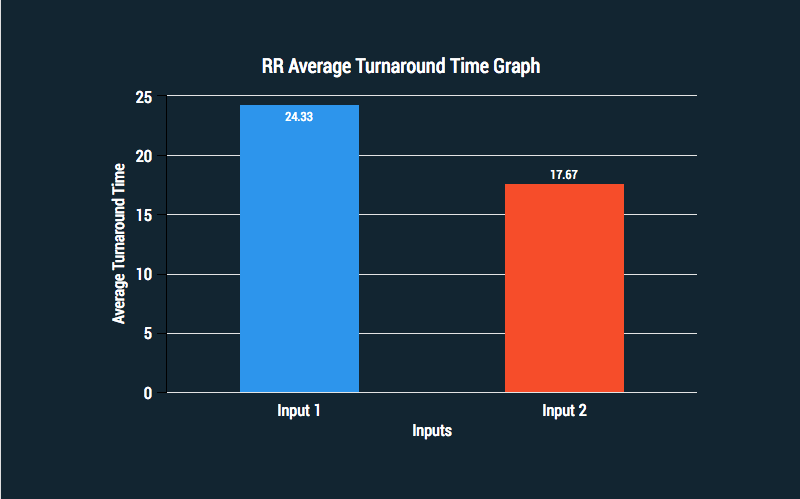
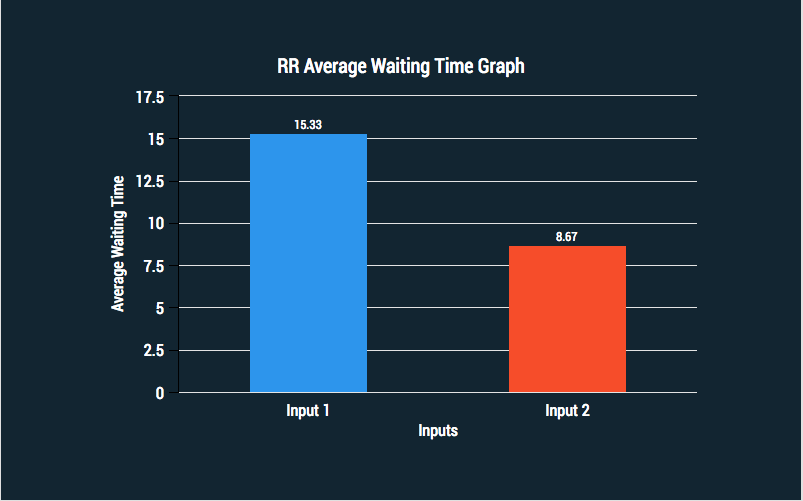
**Shortest Job First (preemptive version)**

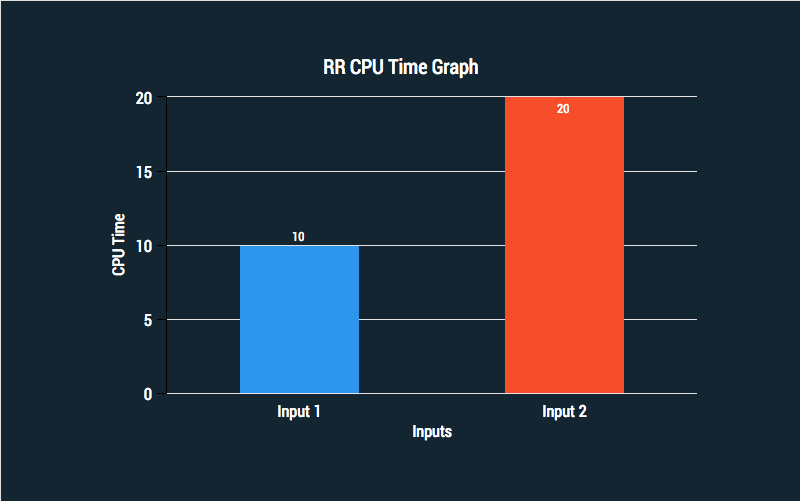
|  |  |
| --- | --- |
| **Input 1** | **Input 2** |
| Number of processes: 3 | Number of processes: 3 |
| Burst time for process 1: 8 | Burst time for process 1: 14 |
| Burst time for process 2: 12 | Burst time for process 2: 9 |
| Burst time for process 3: 5 | Burst time for process 3: 7 |
| Arrival time for process 1: 1 | Arrival time for process 1: 1 |
| Arrival time for process 1: 2 | Arrival time for process 1: 2 |
| Arrival time for process 1: 3 | Arrival time for process 1: 3 |

****

**Round Robin**

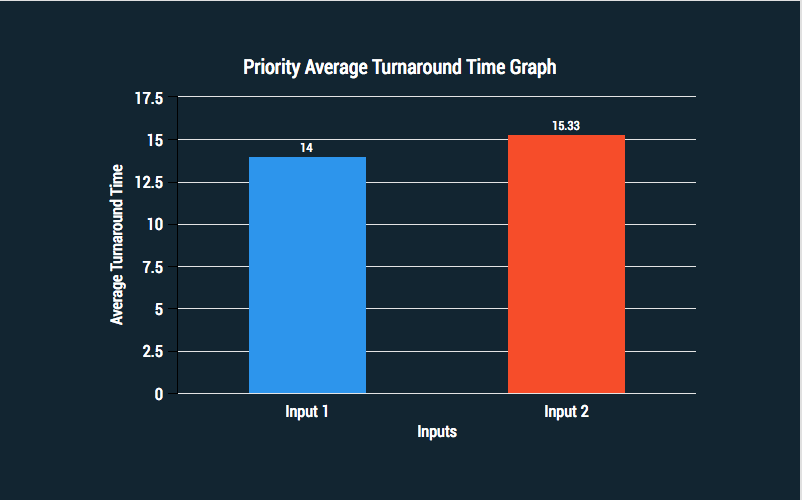
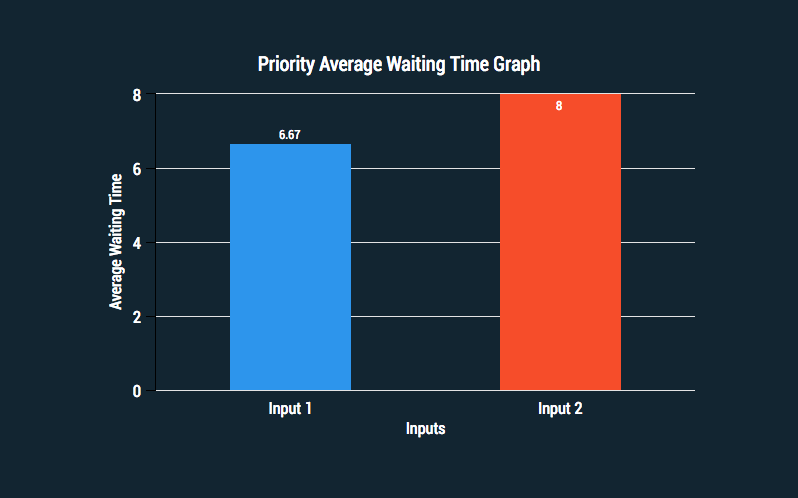
|  |  |
| --- | --- |
| **Input 1** | **Input 2** |
| Number of processes: 3 | Number of processes: 3 |
| Burst time for process 1: 8 | Burst time for process 1: 8 |
| Burst time for process 2: 10 | Burst time for process 2: 10 |
| Burst time for process 3: 9 | Burst time for process 3: 9 |
| Time quantum: 2 | Time quantum: 10 |

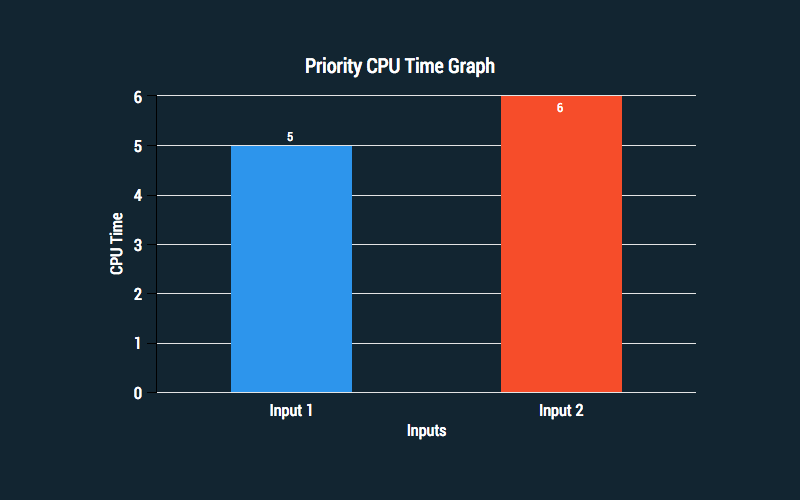
****

****

**Priority**

|  |  |
| --- | --- |
| **Input 1** | **Input 2** |
| Number of processes: 3 | Number of processes: 3 |
| Burst time for process 1: 8 | Burst time for process 1: 8 |
| Burst time for process 2: 4 | Burst time for process 2: 4 |
| Burst time for process 3: 10 | Burst time for process 3: 10 |
| Priority for process 1: 1 | Priority for process 1: 3 |
| Priority for process 1: 2 | Priority for process 1: 2 |
| Priority for process 1: 3 | Priority for process 1: 1 |





* + 1. **Comparison**

**First Come First Served (FCFS)**

FCFS has a very simple implementation using first in first out (FIFO) queue. Due to FCFS being a non-preemptive algorithm, FCFS scheduling is highly dependent on the workload. By performing different workload on FCFS, it is obvious that FCFS perform optimally if disk requests are arrived in a contiguous order. On the opposite, FCFS will perform poorly if request arrivals are scattered across disk, causing the response time of the processes too much to be permitted in an interactive environment.

**Shortest Remaining Time First (SRTF)**

SRTF is a preemptive form of **shortest job first** scheduling algorithm in which the process is chosen whose remaining run time is the shortest. For a new process to be executed, its run time is compared with remaining time of current process, if the new job needs lesser time to complete than the current process, then the current job is blocked, and the new process is executed. This system required very little overhead since it only decides when a process completes, or a new process is added. The algorithm only needs to compare the currently executing process with the new process, ignoring all other processes currently waiting to be executed.

**Round Robin (RR)**

For each process in RR scheduling algorithm, it is given a time quantum to ensure all processes have the same priority which prevent starvation to occur. But as seen in the result graphs, RR is largely depended on the size of the time quantum. If the time quantum is larger than necessary, it will degenerate to a FCFS policy. If time quantum is too small, it will cause unnecessarily frequent context switches leading to more overheads which result in low CPU efficiency.

**Priority**

In priority scheduling, priority can be defined either internally or externally. For the program created here, the priority is defined externally. If processes have equal priority, they are scheduled in a FCFS order. Since priority scheduling allocates CPU to the highest priority, problem may occur when a high priority process uses up a lot of CPU time, which will cause indefinite blocking or starvation for low priority processes. However, this problem can be solved by using a technique known as aging which will increase priority of a process that have waited in the system for a long time.

1. **Scheduling algorithm suitability**

**Situation 1:**

The incoming processes are short and there is no need for the processes to execute in a specific order.

In this case, FCFS works best when compared to SJF and RR because the burst time are short which means that no process will wait for a longer time. FCFS allocate the CPU in order in which the process arrive. When the CPU is free, it can process, which is occupying the front of the queue. When each process is executed one by one, every process will be executed eventually therefore every process will get a chance to run, so starvation doesn’t occur. Once this process goes into running state, its PCB (process control block) is removed from the queue [1].

#### **Situation 2:**

The processes are a mix of long and short processes and the task will only be completed of all the processes are executed successfully in each time.

Round Robin is a pre-emptive scheduling algorithm. This scheduling algorithm works efficiently here because it does not cause starvation because for each round robin cycle, every process is given a fixed time interval to run known as quantum is assigned to every process. A queue is maintained with quantum for each process and no process is left behind therefore all processes are given the same. After a process has finished its quantum, it is placed at the end of the queue and a new process is run from the queue [2].

#### **Situation 3:**

The processes are a mix of user based and kernel based processes.

Priority based scheduling works efficiently in this case because generally kernel based processes have higher priority when compared to user based processes. This is because priority of a process can be selected based on memory requirement, time requirement or user preference. A priority number is associated with each process, the CPU is allocated to the process with highest priority, meaning that smallest integer has the highest priority.

For example, the scheduler itself is a kernel based process, it should run first so that it can schedule other processes [3].

1. **Questions to answer**

* **What happens when a process returns from I/O**

When the process returns from I/O, it will either be in the state of waiting or ready. The process is in a waiting state because it cannot make any progress until an event is signalled. As for process that is in the ready state, it is ready to be executed but another process is still executing on the CPU [1].

* **What happens when a new process is created**

A process is created by another process. Parent is the creator and the children are created. After creating a child, the parent may wait for it to finish its task or continue in parallel or both. In some systems, the parent defines or donates resources and privileges to its children which provides extra flexibility. If required, the parent's process may contain the code for the child's process to run, so that execution is not always needed. The child may also set up inter-process communication with the parent, for example, with a pipe before running another program [1].

* **When / how often the scheduling takes place**

A scheduling system allows one process to use the CPU while another is waiting for I/O, thereby making full use of otherwise lost CPU cycles. Whenever the CPU becomes idle, it is the job of the CPU Scheduler to select another process from the ready queue to run next. The OS maintains a collection of queues that represent the state of all processes in the system [4].

1. **Conclusion**

To conclude, every scheduling algorithms have its own advantages and disadvantages when they are compared with the same or different workload. Therefore, mathematical analysis is performed to determine the performance of the scheduling algorithms for a given workload. This can be done by running the algorithms in a simulation to get a close-to-real-world performance result. In the end, the best way to evaluate a scheduling algorithm is to implement it into an actual system and monitor the performance in a real-world environment [4].

1. **References**

[1] “Process Scheduling.” [Online]. Available: http://www.studytonight.com/operating-system/process-scheduling. [Accessed: 12-Dec-2017].

[2] B. Alam, M. N. Doja, and R. Biswas, “Finding Time Quantum of Round Robin CPU Scheduling Algorithm Using Fuzzy Logic,” in *2008 International Conference on Computer and Electrical Engineering*, 2008, pp. 795–798.

[3] “Principles of Operating Systems.”

[4] A. Silberschatz, *Operating system concepts / Abraham Silberschatz, Peter Baer Galvin, Greg Gagne.*, 8th ed., I. New Delhi, India : New Delhi, India : Wiley-India, 2010.