# First Come First Served

First come first served was the simplest of the algorithms to implement, and had the smallest overhead. However, FCFS was liable to suffer from the convoy effect trapping smaller processes behind larger ones and in my implementation of the algorithm, suffered from generally having the second highest average turnaround time, and second highest waiting time. In general, FCFS can also be a strong advantage in a CPU heavy environment, as a long CPU job can monopolise the processor, and waste potential I/O Time

# Shortest Process Next

SPN was also very simple to implement and my implementation of it had very little overhead and presented a high throughput with a good response time for smaller processes, however it was very apparent that it can penalize long processes and potentially lead to starvation should the system be continually accepting a mixture of short and long processes

# Pre-emptive Priority

My implementation of Pre-emptive priority scheduling was, for me, my most effective algorithm, presenting the lowest turnaround AND waiting time. Its pre-emption increased the throughput of shorter processes as they arrived, and ensured an extremely high response rate. However, Priority Scheduling can easily suffer from starvation unless Priority Boosting is implemented, and in my implementation of the algorithm, should there be a continual input of processes, lower priority processes would become starved.

# Priority Round Robin

In general, I found that my implementation of the PRR algorithm suffered the worst waiting time, and worst turnaround time, a large part of which was caused by the constant calling of the dispatcher, and lack of pre-emption as new processes arrived. PRR would heavily favour CPU bound processes, with I/O processes using the CPU for less than the time quantum, and then being blocked while waiting for I/O until it reaches the head of the queue once again.