To solve the issues being encountered by the current system I looked at what possible processing algorithms could be used, and which one would be best suited for the task. The current system is having the problem of needing to be able to handle a combination of long and short processes and requires that they are all completed successfully in the given time frame. This is not being accomplished due to poor algorithms choice and CPU starvation / long wait times. Therefore, I decided to use the round robin approach in accordance with the multi-priority level queues.

The current system was using the “short job first” algorithm which compared to the proposed “round robin” has a lower average turn around. Round robin has a better response time, meaning that the time it would take from a request being received to its first response being processed is shorter and as a result will go through the processes faster and effectively starting to solve the problem of CPU starvation and long wait times. Moreover, if we already know how many processes the queue will run we can know what the worst-case response time will be for the process.

The main problem with the current system was that the tasks with low to mild priority were not getting completed at all nor were they getting the attention they needed and as a result other tasks would also suffer from CPU starvation and long wait times. To solve this, I chose round robin over other scheduling algorithms as round robin allocates a small amount (unit) of the CPU time to each process; meaning the system will no longer suffer from CPU starvation and long wait times due to round robin’s cyclic nature. Since no process will wait more than (n-1)q time units, this usually being 10 to 100 milliseconds . This is because for *n* processes in the queue and time quantum (*q*), each process will get only “1/n” of the CPU time, having at most *q* time units of time at any given time for a task.

However, a major setback of round robin is that though it is great at giving each job a fair process in terms of the time slot / quantum given, if a task is not completed by then, the job coming next will interrupt it and take over. Effectively resulting in similar behaviour that you would expect to see from a first come first served algorithm. In addition, if the quantum of a task is shorter than needed the amount of times the CPU will have to switch from one process to the next process will increase, resulting in a significant decrease in CPU efficiency; i.e. if a task is completed faster than the allocated time the algorithm will move on to the next one. This is the case as some processes as suffering from small burst times. Priority cannot be given in the process and therefore we lose the priority function of the system / algorithm.