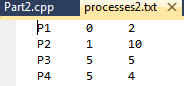
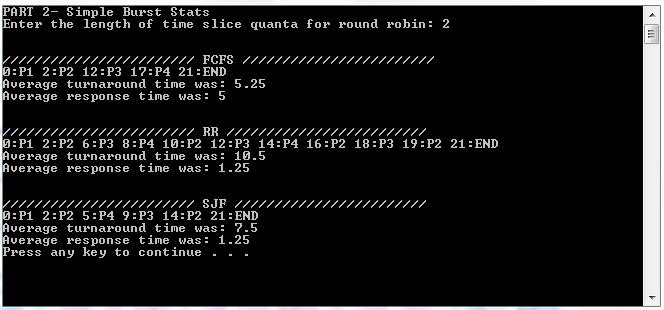
**Part 2**



Using this file containing process info:

I obtain these results.

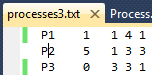


a) Looking at the image, FCFS turnaround time 5.25 is much better than RR 10.5. The conditions for FCFS to have better turnaround time are that the time slice for quanta is shorter than the run time of the processes. If processes run longer than the length of quanta, then they will start in RR and not be able to finish right away. FCFS processes always finish after they start though, so FCFS will have better turnaround when quanta lengths are shorter than burst lengths.

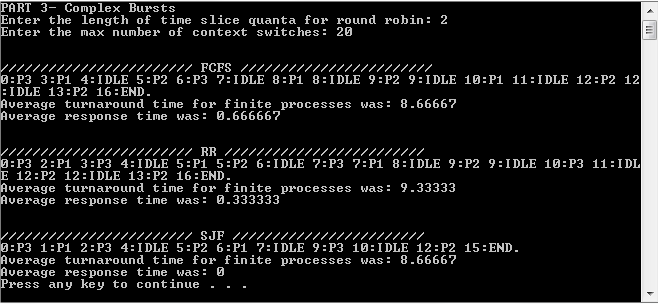
b) I don’t think this is actually possible. For the situation where only one CPU burst occurs at a time, FCFS will always have an advantage for turnaround time because a process will always start, run its burst duration, and finish in the shortest amount of time. For RR, best case scenario is that a process can start at the same time as a new quanta, giving turnaround time equal to FCFS. But outside of the best case scenario, a process will start, run for a bit, pause while other processes run, and then finish, making its turnaround time worse.

c) Looking at the image again, SJF has a response time much better than FCFS because when a new, shorter process arrives, it starts immediately. My example also shows SJF giving an even response time to RR. RR is also better than FCFS in terms of response time because the intermittent switching of processes encourages processes to start sooner rather than later.

**Part 3**

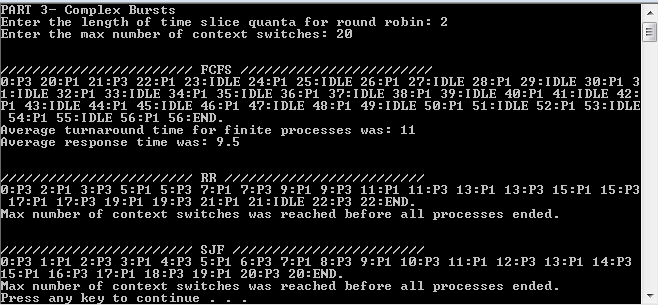


1. Using this file,

I obtained these results.

If you compare FCFS with SJF, you can see that at time 1 for SJF, there is an extra context switch not made for FCFS. This shows that SJF has been implemented to be pre-emptive; whenever a shorter CPU burst becomes available than the currently running one, the shorter burst will become active and will run.



1. Using this input file,

I obtained these results.

As you can see, The SJF scheduler said that “Max number of context switches was reached before all processes ended.” Because there are only two processes in this input, and one of the processes is infinite, that means that the other process (P3) was unable to finish according to the criteria for outputting turnaround and response time. That means that P3 starved during this schedule because it was unable to finish in the given 20 context switches. P3 ended up starving because it had one main CPU burst that was long, and the competing P1 had very short CPU and IO bursts in infinite sequence. While P1 was busy running and restarting, P3 was hardly able to get through.

1. If you will look back to my images from part a, you will see that FCFS and RR were able to finish without starvation. FCFS is quite likely to be able to avoid starvation because of the way it lets processes run for as long as they need to at once. For RR, it is a little less likely. You can see this by comparing my images from part a (RR finishes without starvation) and part b (RR finished with starvation). For RR, avoiding starvation is less likely based on the value given not only for number of context switches but also for quanta. If quanta is really low with respect to the CPU bursts of processes, then it is much more likely that starvation will occur in a dataset where a proportionally small number of context switches is allowed.

**Part 4**