Assessed Exercise 2

Overall I must admit that this exercise what very challenging at times. I believe the reason for this was to do with the magnitude of different components in the code. Speaking to the demonstrators definitely helped me better understand the task at hand however as you shall see there were some parts which I just could not get. Overall I think my code is fairly average, no matter how hard I tried I just could not get the round robin working and I also could not figure out why my turning around time and waiting time were offset ever so slightly from the sample outputs in SRTF. Despite not being able to complete the whole exercise I definitely believe that it has developed my understanding of the course content. For this report I shall speak about my implementation and struggles for each algorithm, then I shall speak about the "interesting" seeds.

My struggle with this algorithm was mainly due my more significant struggle with the code as a whole. Once I understood the OOP approach better as well as the general idea of what was going on this algorithm was not to difficult to implement. One of the initial confusions was distinguishing the events and the processes; at first I kept trying to loop through the events queue rather than the processes. With the dispatcher function I was initially fairly confused as to what "running" the process entailed and at one point I actually thought it involved directly using the "run" function in the DES file. Once I realised my mistake it took a helping hand from the demonstrator to realise what the appropriate arguments were for the "run_for" function, I was initially just passing in 0 for the "cur time". To summarise once I got going this algorithm was definitely the easiest one.

The second algorithm was definitely a lot easier to implement due to its similarities with the first one. The main difference of course being that you have to first of all sort the list of processes. I considered various different methods of sorting however the one that I deemed best was the lambda method. Aside from this slight variation the rest of the code is essentially the same as the first algorithm so I will not waste any more time talking about it.

This algorithm was definitely the hardest for me to implement, I must have spent several hours on it. The part that really confused me was adding the processes to the end of the list if their execution was not complete. I first of all tried this by deleting and then appending processes from the list of processes however this brought me no luck. I was also informed by one of the demonstrators that it was best to leave the said list alone. After trying for ages I settled on a sort of "best effort" approach and picked the process which corresponded to the current event. I thought this would work as the events are popped and added back into the queue if the corresponding process is not finished executing however this clearly was not the solution. I think this was because the events were not necessarily added onto the end of the events list. In terms of the dispatcher function I believe that my implementation here is not too bad. The state keeping and execution of the process itself seem to work fine even though it is the wrong process being selected. Given that I get negative result for the turnaround time and waiting time I am aware that my solution with regards to the scheduler function is drastically incorrect. I believe if I had given myself more time to complete the assignment I could have gotten this working with a nudge in the right direction from a demonstrator however at least I got the dispatcher aspect working.

I believe I did a lot better for this algorithm than for the RR however my results were ever slow slightly divergent from the sample ones. The sorting algorithm which I used was the same as the one I used in the SJF scheduler and I believe it is also appropriate for this implementation. Given that my results were extremely similar to those of the sample solution I believe that my implementation of the dispatcher function must almost be correct. It is fairly similar to my implementation of the RR dispatcher with the main difference being the first parameter in the "run

for" function. My thinking for this was that the Scheduler selects a new process whenever a change occurs in the system which is basically the next event. By setting the max amount of time that the process can run for to the time of the next event I believe that correctly implements what is required of the SRTF algorithm. As for the rest of the dispatcher function it is essentially the same as the RR implementation, if the process needs more time a "PROC_CPU_REQ" type event is returned and if it is finished then a "PROC_CPU_DONE" type is returned. As to why my results varied every so slightly I really am not sure at all; I tried tweaking the arguments ever so slightly however this only drove me further away from the desired results.

For the first seed

As for all the seeds my third and fourth algorithms do not work properly so I will speaking with regards to if they did in fact work. FCFS is the slowest in this case which I guess is to be expected as it does not involve sorting of any kind. This is the reason I believe why SJF is faster than FCFS, it does not make faster processes wait on slower ones to complete before they can be executed themselves. This means that processes spend a lot less time in the ready queue on average as they will only be waiting on processes which have lower service times than themselves. Processes also wait less to get into the ready queue which is why the turnaround time is less. I believe the waiting time would be higher because a process could be made ready however not fully complete therefor it has to wait longer which adds to the time. On the other hand I believe that the turnaround time would be less than that of FCFS and SJF because processes spend less time waiting to get into the ready queue. I believe SRTF is the fastest in all regards because ready processes with short waiting times are selected immediately and processes don't have to wait long to get into the ready queue.

Second Seed

I believe SJF will always beat FCFS due the fact that its ordered and this case is no different. For this the average service time is a lot shorter which is why I believe RR would do very well with this seed as processes would not have to spend long in the READY queue. Also due to the rather small service times I believe SRTF would do particularly well.

Third Seed

For this seed there is quite the variation in service times. The fact that some of the larger service times are situated at the start is the reason why the times for FCFS are fairly high. This I believe is the same reason as to why SJF is much faster than FCFS, it does not have to wait for the processes with long service times to execute before the smaller ones can due to the list being order first. Due to the high volume of short service time processes I believe that SRTF would be pretty fast because the smaller processes would all be picked first same as with SJF. As for RR I believe that this would execute slower than SJF and SRTF however faster than FCFS.

I do appreciate that this is not the most concise evaluation of the provided seeds however I hope you can appreciate that it is harder when you do not have all of the algorithms working properly.