To be honest my Virtual machine has ran into several problems while bring in system call so I borrowed others Virtual Machine to run the code.

1. Design

My scheduler is in userspace which is provided by the Linux scheduler. Since Linux scheduler provides FIFO policy it gives priority over normal processes. In my code the scheduler and the children will run in this policy which means that this process has priority over other processes. Under FIFO policy lower priority processes can't come first than the higher priority processes. I expected my code to make a context switch. Assuming that the scheduler runs the process with higher priority first and make sure there is always only one child being the 2nd priority and others being the lowest priority. If the scheduler is being blocked the child with the highest priority gets chance to run.

So in the main program, the code sorts out the schedules that has to be executed with the schedule. Then, it starts calculation. However, every time you meet a child process it has to be done but keeping the priority of parent – child relationship. Until all the subroutines aren't over the program is unable to be finished. The main program will sort the schedule this time that has to be performed with the schedule. Starting calculation, same as calculating execution but this time performed data is being saved into queue which is the time. As same as before until all the process are not finished, this program can not be finished.

2. Version of my core : 4.14.25 with 16.04 iso ubuntu.

3. Comparing the actual results with theoretical results and the differences that made the cause. (I had to borrow friends VM machine to see the results since my machine did not work)

I detected differences in RR_1 and RR_3. It's obvious that in RR_1 the time slice ends before P1 has a chance to finish, so it has to wait for P2 to P5. P2 has to wait P1, P3, P4, P5, but as P1 will end very shortly, its error isn't as big as P1. The same goes for P3, P4, and P5. In RR_3 it was more difficult to explain the difference between theoretical result and actual result. However I noticed that the actual run time is usually shorter than expected, so I recalculated the theoretical result by assuming the process always ends earlier than expected, so the next process doesn't get any CPU time. My conclusion is maybe there were several context switches that made the difference in the time.