Documentation Q1

Implementation:

1.1) Thread Scheduling

For the creation of a thread, pthread_create is used. Global variables are maintained of pthread_t (id of thread) in order to schedule them later in functions.

In the main function, 3 threads are created and after creation, they all are joined will this process in order to wait for them to be finished. Before starting counting current time is stored and after the count is finished that moment time is separated from the start time to get the time taken by the function to do the calculation.

In function, scheduling is done using the pthread_setschedparam() function and global thread id is used for scheduling. Thread priority is set using struct sched_param (range of priority is 1-99 and hence priority is set between 1-99 only). The result of each function is written in the corresponding file and later on, using python's matplotlib library, the graph is made. For the sched_other scheduling class, we can't set sched priority and need to benchmark it using changing its nice value. The nice value(range -20 – 19) is changed using the setpriority() function.

Scheduling priorities are stored in an array and hence for each iteration a value from the array is used.

1.2) Process Scheduling

For the creation of 3 processes, 3 fork() calls are made if pid returned by fork ==0 then it's a child process, therefore we schedule the child process with the help of sched_setscheduler() function and we execute execl() function. With perror() system call, the error is handled. After the creation of all 3 processes, we wait for each process to over using waitpid (-1, NULL,0). Then the id returned by the waitpid is compared with all the three process ids and calculate the time of the process whose process id==id returned by waitpid(). In this way, I don't have to wait for a particular process, but rather calculate the time of the earliest completed process and hence process scheduling is executed perfectly. I used a while loop to wait for all the processes and break the loop in case of all the child processes have executed successfully.

Combined Result:

Only 1 core was allotted to the virtual machine for the accurate benchmark of scheduling classes

Time taken by the FIFO scheduling class as expected was the lowest among all other classes. Round Robin scheduling class stood second in the list whereas other stood last. FIFO is a first come first serve algorithm and should have the least time among all and therefore results matched with expectation. Round robin, on the other hand, has context switching therefore time was slightly more than FIFO and was as per theoretical expectations. Other took the most time as unlike other policy it has dynamic priority and with time its priority changes unlike other classes, no fixed priority is associated with it, and hence 'Other' have more time than the above-mentioned scheduling classes. Therefore, all results were at par with theoretical results.