To Schedule or Not To Schedule?

Name: Hanaa Zaqout
Student Id: 2020400381

Development environment: Wsl on windows 10

Implementation Logic:

First step in my scheduler is reading files and storing needed information in data structures and structs. (Note: Process processes[10] stored all process structs in the order they came in input, and along the whole program, processes will be accessed from here using their index which is fixed all the way long.)

Now the scheduling journey starts in schedule() method:

- time_now variable starting at zero and incremented at the end of each cpu burst execution. Whenever the time_now is updated, check for new processes that may have arrived at the last cpu burst execution. For this purpose, load_at_station() and load_till_endt() were implemented such that the first load process at a given station (arrival time) e.g. all processes which had arrived at time 0. While the second, load all processes till specified end time t starting from the next station (this may include 0, 1, or many stations (arrival times)).
- At the beginning, processes with arrival time=0 are loaded to ready queues (for Platinum, Paltinum_readyQ and for Gold and Silver, round_robin queues)
- Now let's schedule processes till all of them are done!
    - If both Paltinum_readyQ and round_robin queues are empty then the cpu will be idle for an amount of time, so time_now jumps to the next arrival time of a process.
    - If Paltinum_readyQ is not empty schedule the highest priority process at index zero by calling process_one_PLATINUM(). (Note: indexes of processes in Platinum_readyQ are sorted with compare_structs() considering priorities, arrival times, then processes' names)
    - otherwise, schedule a Gold or Silver process by calling process_one_GS()

What is process_one_PLATINUM() doing?

- Execute all instructions in the process
- Delete the process from PLATINUM_readyQ
- Update time_now to its completion time
- Check for new arrived processes

What is process_one_GS() doing?

- Execute atomic instructions for Gold and Silver processes. After execution of each atomic instruction, it checks for newly arrived processes.
- Quite execution of current process if:
    - Platinum newly arrived
    - Higher priority Process newly arrived
    - Time quantum has finished for the current process and there are other processes with equal priorities in the round robin queue
- Upgrade the Gold process to Platinum if 5 quantum times have been executed for this process. Now, as it is Platinum:
    - If it is the only Platinum in the Paltinum_readyQ or if it has the highest Platinum priority, then execute all instructions till the end.

    - Otherwise, add the process to the Platinum ready queue using insertionsort() which considers the priorities while inserting the process to the ready queue.
- Upgrade Silver to Gold if 3 quantum times have been executed.
- If the instruction is already done, then update necessary variables e.g. global variable p_done that has the number of processes that have terminated. Also, there is an edge case here that the done process may be the last process in its priority queue, so we need to find the new highest priority queue in our round robin structure.
- If the process is not done, enqueue() again to the same queue with new arrival time. (note I made a new_arrival_time variable distinguished from arrival_time since calculating the turnaround time in calculate() depends on the arrival_time variable and I do not want to miss its value). Worth to mention, there is an edge case that when the current process reenters the queue, it will be equeued at the end always. But, it might be the case that re-entering process enters the queue at the same time of a newly arrived time with lower priority (e.g. in terms of process name). To locate the re-entering process at the correct location in the queue, edit() function does the necessary swaps.
- Note: I use mapped_priority rather than priority to choose the priority queue in the round robin structure. The idea is the following: we have at most 10 distinct priorities in the input. I order them in ascending order and assign every priority to a corresponding index. E.g. priorities 5 5 -1 2 17 are mapped to 2 2 0 1 3 indices of queues such that 0 is the lowest priority and 3 is the highest priority. That is the task of the check_priorites() method.

About context switches in the program, I add 10 to completion time in case of Platinum. But for Gold and Silver, I needed to check that it is not the same previous executed process (as we do not context switch between Px and Px).

Challenges:

- I tried to implement the queue for round robin without using the queue data structure by just traversing an index pointing to the current running process in the queue. But, when new processes enter the queue while the queue is running, stuff gets so complicated. So, I switched my implementation to a normal queue (actually a queue for each priority).
- Calculating context switches: at the beginning I was counting them as an independent variable and adding them in the calculating final step. But, this is wrong because context switches must be included in the time otherwise we may miss loading higher priority arrived (not adding 10 to time makes the scheduler think that they have not arrived yet!!!) processes when the current cpu burst finishes.

To Improve:

- As I worked on the project, I kept discovering new situations I hadn't thought about initially. This led to making many changes, and the code became a bit complicated. Now, I believe starting over will help create a simpler and clearer solution by learning from those experiences.
- Some searching or sorting parts of my scheduler can be achieved with a lower time complexity, but it is hopefully okay for this project as we have 10 processes at most.
