Simon Owens and Kunal Mukherjee
Operating System CS470
Process Scheduling

High-level Design

This program is a process scheduler class which demonstrates how a preemptive round robin queue works. The purpose of this assignment is to gain appreciation of the work involved in managing processes. There are two files, the process-management driver, and the other is the input file. Output is written to a file. The usage for the program is as follows:

"python3 process-scheduler.py input-file q output-file"

This program is combined into three large pieces: the "process" class, the "process object block" class, and the main program. The "process_obj" class acts as a simple process for us: it holds its PID, initial burst time, state, and the time left till it is complete. This class also has functions to easily change and get values residing inside this class. The process control block "PCB" class manages all of the processes by keep track of the ready queue, wait queue, current running process, the parent process, q, and how much time q has left to run. This class was created so that users could easily assert control or manipulate with processes however they want: it is very flexible. You just give it commands, and it manages everything for you. That is what the main program does, it defines all of the commands, opens an input file, and executes those commands in the file.

The processes control block is where all of the design is for this project. It can create processes, destroy them, interrupt them, place them in a wait queue, take them out of the wait queue, print out the status of all processes, and exit. The user does not need to be concerned with how processes change state or values: everything is taken care of in the "update" function. This function is called after every operation to ensure that the correct amount of time has been taken, and that everything is in the right state.

A major change in our design is that when a 'E' is called and the process waiting for that event if put in the ready queue from waiting queue, the process is placed in front of the queue rather than in the back. Therefore, our result is slight different that what is provided in the example sheet. This change in design was made keeping in mind that if a E or event has been triggered then the priority of the process is high and it should have to be completed first. Our design wanted to keep in the fact that if the user should have the most flexibility and have the ultimate say on how the process should be scheduled, in terms of OS, the process scheduler should be able to schedule process however it wishes and should have this freedom.

Python lists are used so that processes can easily be added or removed. The algorithm used for removal of a process from ready or wait queue is a simple linear search while iterating though the node and removal of the particular node containing the process. The addition of a process is also trivial as it is used to place it in front of the queue. The reason behind using a python list the flexibility of using the add and removal function, also the addition is in O(1) and removal is in O(n), so the program is optimally sorted. They are also excellent because they are ordered, I only need to take the 0 index when trying to find the next process in the ready queue. A dictionary was used for passing commands to the class because this reduces the amount of conditional statements. If operations were to be added, they could either be stuck into the existing conditional statements, or one more could be easily made.

Additional Questions

1) The RR scheduling algorithm seems to be moderately efficient at completing processes. However, the RR scheduling has huge draw back. If the burst time is significantly large of a process than the time quantum, then the process will consume huge amount of CPU time come to conclusion or termination. Conversely, if the quantum time is much larger than any processes burst time, then some processes wait a tremendous amount of time or in other words CPU time will be wasted by being idle. The RR scheduling algorithm worked best with quantum size near the average burst time of the processes. The program design in general makes it very difficult to compare the performances since no data related to the running of the program is calculated or in other words no statistics is generated.
2) It was very difficult making sure everything was in the right state. I had to make several conditionals and functions just to keep track of everything. There were several edge cases to consider.
3) It was very easy adding, removing, and searching for processes from the ready queue and waiting queue. This was mostly because of python and the data structure of python list, which has linked-list implemented behind the interface. The experience of using linked-list to do editor in cs215, helped with the understanding of the implementation of the program.
4) I would refactor a TON of stuff. I think my design is decent and very flexible but many things could be cleaned up. I did not refactor because lack of time. I would like to add a method that will dynamically calculate the quantum time, so that the RR algorithm is optimal. The method will look into the ready queue and generate the time quantum based on the average time left and once a process is added from the waiting queue, the method will automatically change the time quantum to the average value of the updated ready queue.
5) I was unsure if the quantum time decrement on an event is a wait or signal. This project was awesome, it improved my process scheduling knowledge as well as my design skills. It was hard keeping everything clean, simple, and reusable.