Specification 3:

NOTE: Run *make qemu SCHEDULER=…* to choose a scheduler other than the DEFAULT (Round Robin) scheduler! "…" can be either FCFS or MLFQ. Remove the "SCHEDULER" variable altogether to choose the default scheduler.

Scheduler Algorithms Implementation:

First Come, First Serve (Non-Preemptive) (FCFS):

1. Used the ctime variable in the proc struct (in proc.h) which keeps track of the time (in ticks) at which the process arrived.
2. This variable has already been initialised in the allocproc() function under the found label (in proc.c), and is set to the ticks at which the process arrived.
3. Wrote the algorithm for the process under scheduler() function (in proc.c) with a pre-processor directive #ifdef FCFS, which is set by the Makefile, if FCFS is chosen at compile time. The algorithm is essentially as follows:
    1. Made a struct proc* currProcess variable to keep track of the currently chosen process.
    2. Initially, it is set to 0. I trivially set it to the first process in the list that is RUNNABLE, and for every subsequent process, I compare it's starting time with this process. If it starts earlier, then I choose it instead. I keep doing this until I determine the earliest started process. Note that I do not release the lock of the process that I've chosen to be my running process, since I don't want a race condition to arise when this process's struct is accessed on multiple threads.
    3. With the earliest started process, I do the exact same thing that Round Robin (default) scheduler does to run the process; I already have this process's lock (from earlier), change its state to RUNNING, and then hand its control to the CPU. When it is done, I release this process's lock.
4. I ensured that this scheduler is Non-Preemptive, by disabling the timer interrupt (in trap.c), when FCFS macro is defined (for DEFAULT and MLFQ, they work as expected.) I did this, by making sure that yield() in usertrap() and kerneltrap() is called only when scheduler chosen is NOT FCFS, when a timer interrupt is hit (which_dev == 2 for a timer interrupt, as it takes the return value from devintr()).
5. **ASSUMPTION:** CPU-bound processes run before I/O or interactive processes.

Multilevel Feedback Queues (Preemptive) (MLFQ):

1. Added 4 variables to the proc struct (in proc.h), wherein, I keep track of the current priority queue that the process is in, the number of ticks that have elapsed since the process was created, the position in the priority queue where the process is at, and the number of ticks it has been waiting for because of the process that is currently running.
2. Also defined macros (in proc.h), to quickly change out the MAX_WAIT_TIME of a process (before its priority gets bumped up, i.e., aging time), and also to quickly add more queues if necessary (just change the LEAST_PQ_PRIORITY macro (in proc.h)). Remember to change the maxTickTimes array, which is defined as a global array (in proc.c, at the top).
3. Wrote the algorithm for process under scheduler() function (in proc.c) with a pre-processor directive #elifdef MLFQ, which is set by Makefile, if MLFQ is chosen at compile time. The algorithm is essentially as follows:
    1. Made a struct proc* currProcess variable to keep track of the currently chosen process.
    2. First, I go through the entire process list and figure out which process to run (the process in the highest priority queue ($0 > 1 > 2 > 3 > \dots$ in terms of priority), with numerically the least index possible in that queue).
    3. I run this process just like how the Round Robin (default) scheduler runs the process; I first acquire the process, then I change its state to RUNNING, and hand its control to the CPU.
    4. When it is done, I then increment the tick count of this process (as a timer interrupt is called every 1 tick, so, I get control back after 1 tick). If the process is NOT RUNNABLE anymore, then it has voluntarily relinquished control of the CPU. So, I remove it from its current position in the queue, and push it to the end of the queue, PROVIDED that it hasn't exceeded its max tick time for the priority queue that it is currently in.
    5. If the process has exhausted the max tick time for the priority queue that it is currently in, then I demote the process by one priority queue (provided that it isn't in the last priority queue) and push it to the end of that queue. Then, I reset it's ticksElapsed count, and also its waitTicks. After this is done, I release the process's lock.
    6. Note that this is Round Robin on each priority queue, since, if this earlier index process voluntarily relinquishes control of the CPU (for example, turns into an I/O bound process), then it gets put at the end of the priority queue. Else, when it's time slice is up, it gets demoted (unless it is in the last priority queue, and if it is, then I just move it to the end of the last queue.)
    7. For all other processes except the chosen process (ONLY IF THEY ARE RUNNABLE), I increment their wait ticks time by 1, every time control is given back to the hardware after running the process. (i.e., when a timer interrupt occurs)

    8. If any process has been waiting for more than MAX_WAIT_TIME (aging ticks) as defined (in proc.h), then I promote it up by a priority queue (unless it is in the highest priority queue), and put at the end of that queue.

4. I ensured that this scheduler is Preemptive, since the timer interrupt (once every tick) (in trap.c) will occur for this scheduler. (it is disabled ONLY for FCFS.)
5. **ASSUMPTION:** According to Q9 of the doubts document, I will not be resetting the ticksElapsed of a process if it is no longer RUNNABLE (say, it has become I/O bound). I will be resetting the ticksElapsed only for those processes which are still RUNNABLE, but have exceeded their allocated time slice and thus have been demoted. I will also be resetting the ticksElapsed for those processes that are still RUNNABLE, but are currently waiting to be run and have reached their aging time, and thus have been promoted.

Average runtime and wait times noticed for the given schedulertest program on 1 CPU:

DEFAULT (Round Robin):

13 ticks, 153 ticks respectively. (the first time it is run, after starting xv6 up)
Nearly 11 ticks, 146 ticks respectively. (every other time it is run)

First Come, First Serve (Non-Preemptive) (FCFS):

12 ticks, 124 ticks respectively. (the first time it is run, after starting xv6 up)
Nearly 11 ticks, 123 ticks respectively. (every other time it is run)

Multilevel Feedback Queues (Preemptive) (MLFQ):

12 ticks, 149 ticks respectively. (the first time it is run, after starting xv6 up)
Nearly 11 ticks, 145 ticks respectively. (every other time it is run)

Timeline plot for CPU-bound processes in MLFQ on 1 CPU:

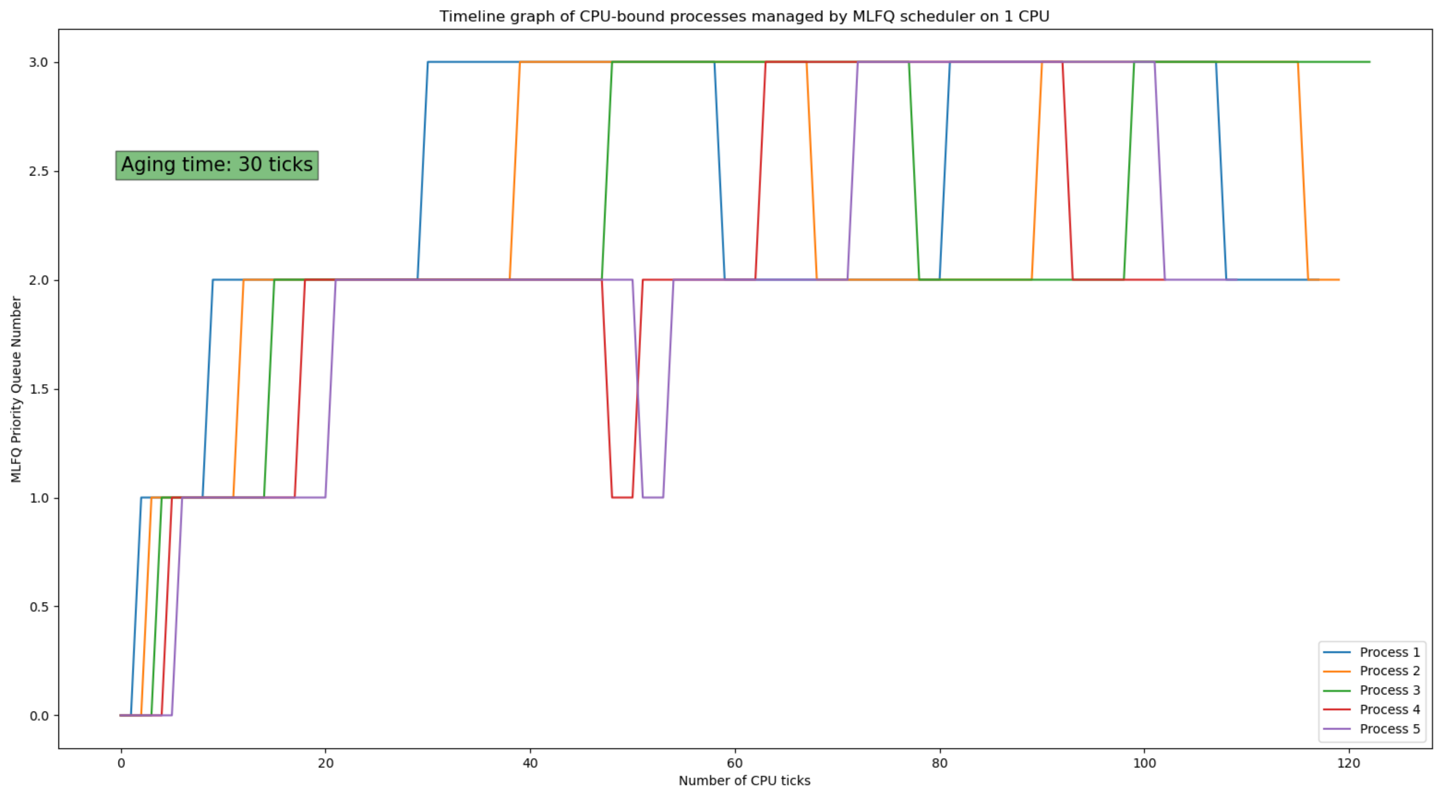

Figure 1: Spec3Part2MLFQplot.png
