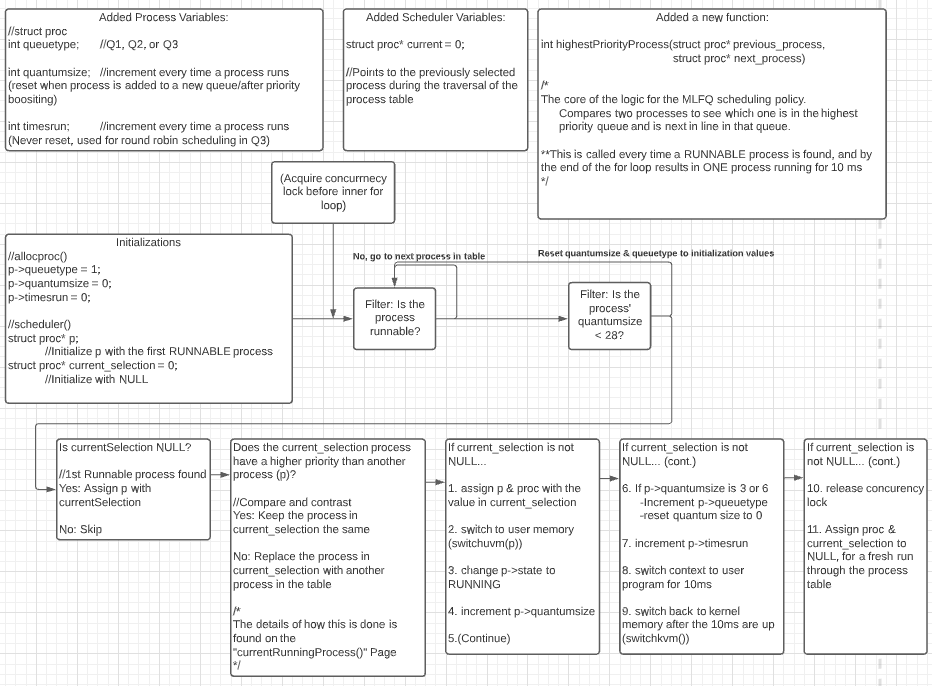
Programming 2 (P2): Revised MLFQ Scheduler.

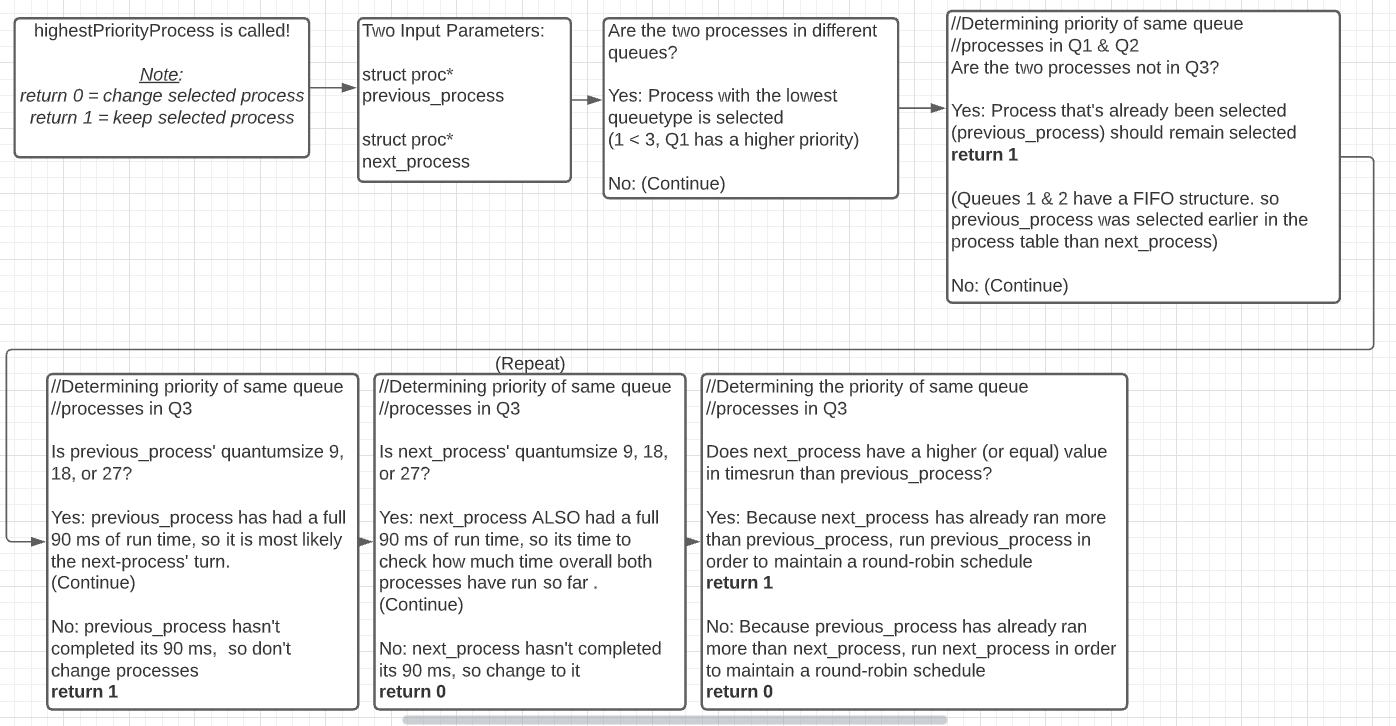
**Section 1: How the new scheduling Policy in xv6 was implemented (Diagram)**

The main design choice of this MLFQ policy is going through all 64 processes in the process table, so that a single process can be selected to run on the CPU for 10ms. This is repeated infinitely, so that a process can run for its given quantum size (30 ms, 60 ms, or 90 ms). Once a process has used up its quantum by running 3, 6, or 9 times; multiple filters and a logic is used to choose the next highest priority queue.

Below is a diagram of both the main flow of the program, as well as the logic held in the highestPriorityProcess function that is used to determine which RUNNABLE process in the process to select next.

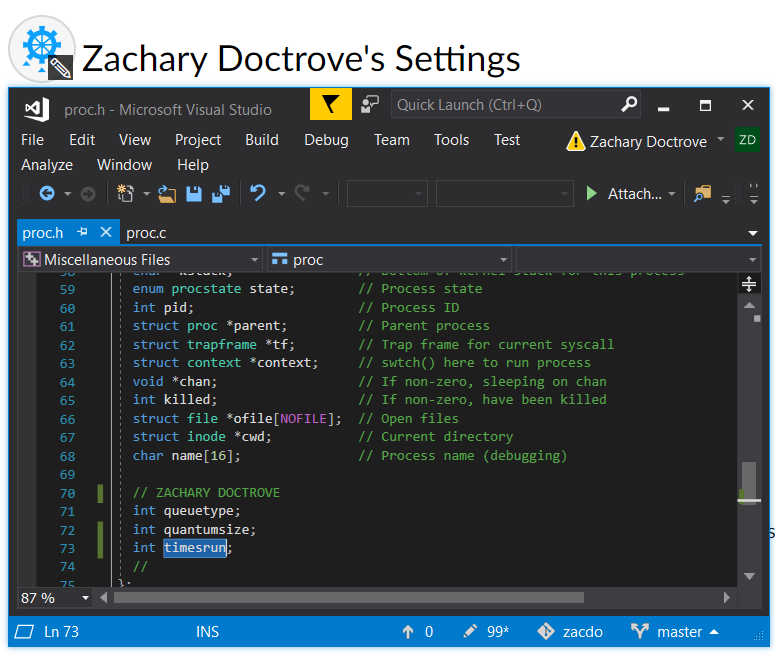
General Workflow in scheduler() (Plus variables and functions used)

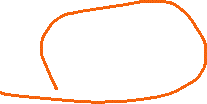


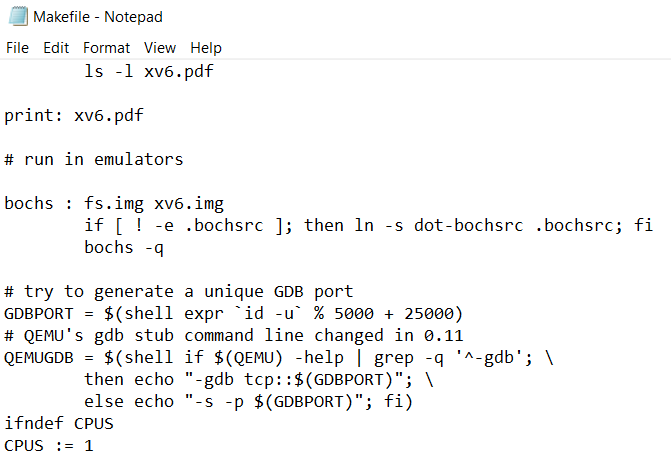
Workflow of highestPriorityProcess()

**Section 2: Added and Revised Code**

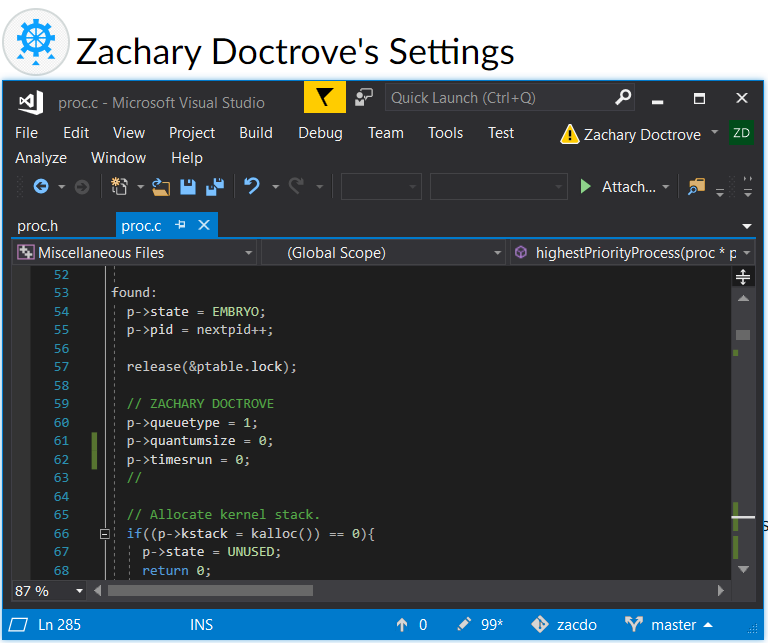
**Added** variable declarations in struct proc (proc.h)



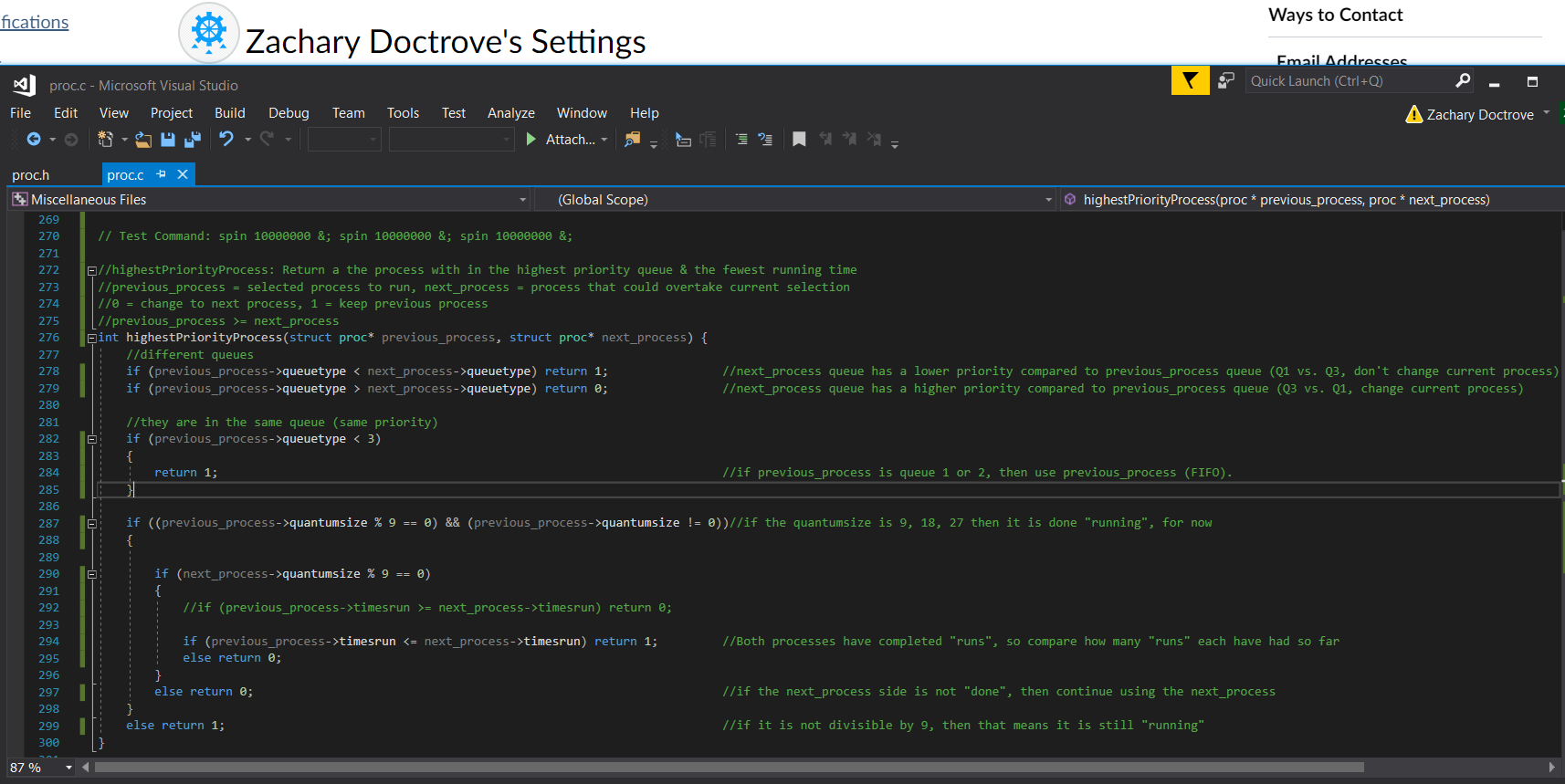
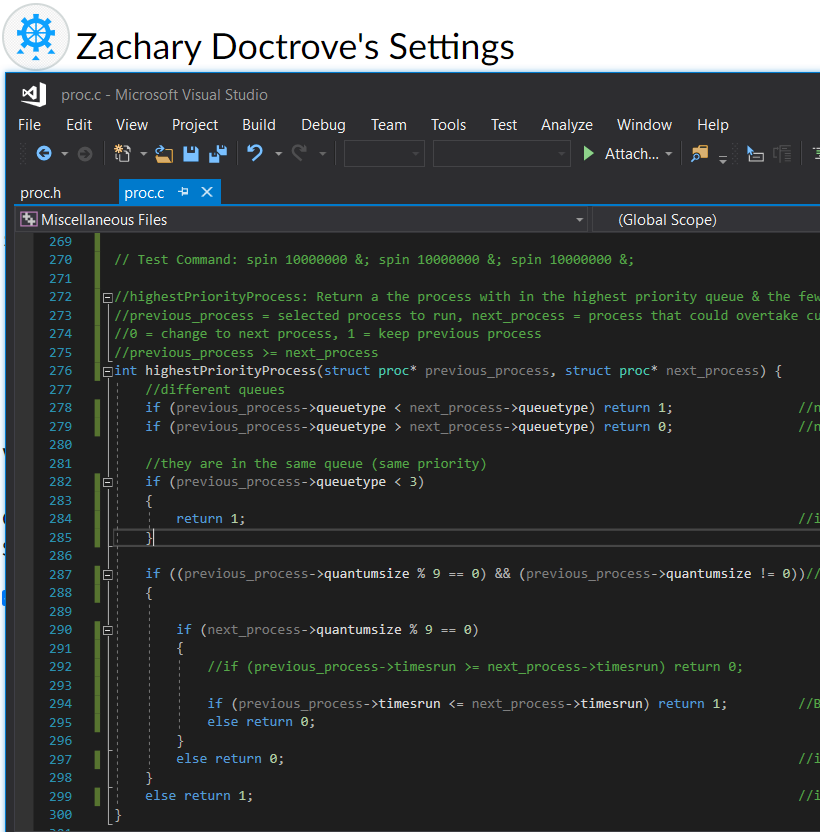


(Changed used CPUs to 1 in MakeFile)



**Added** variable initialization in allocproc() (proc.c)



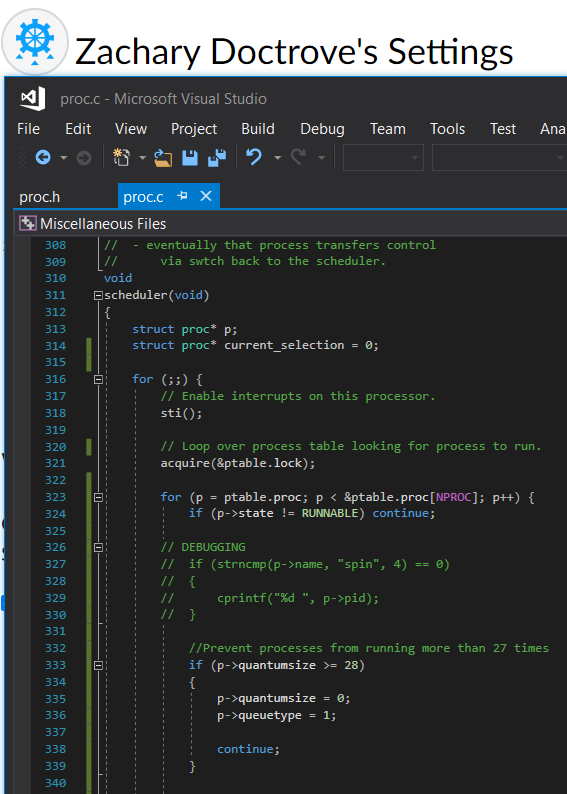
**Added** highestPriorityProcess function (proc.c)

(Comments Included)

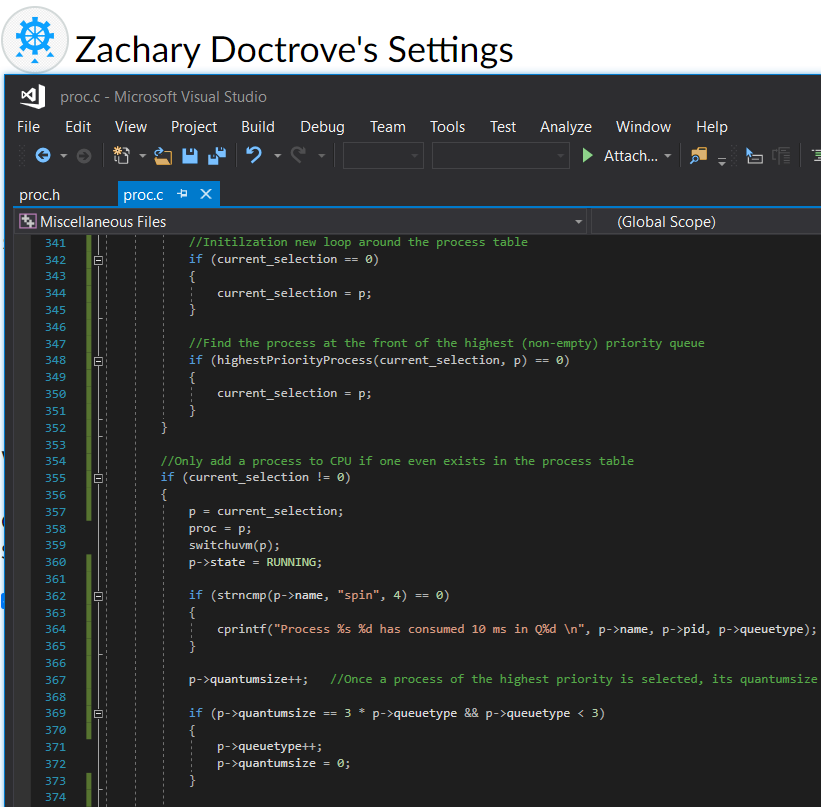
(Code Focused)

**Revised** scheduler() function

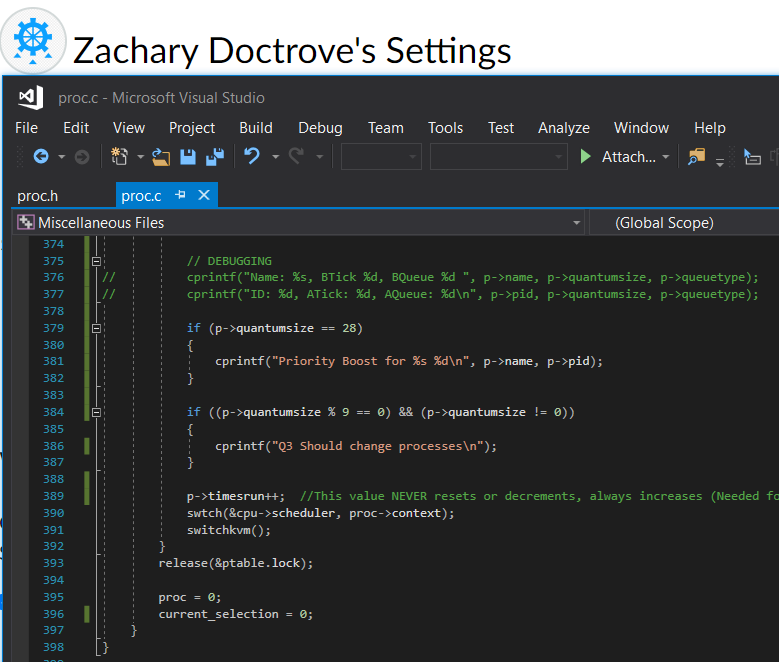
**(Page 1)**

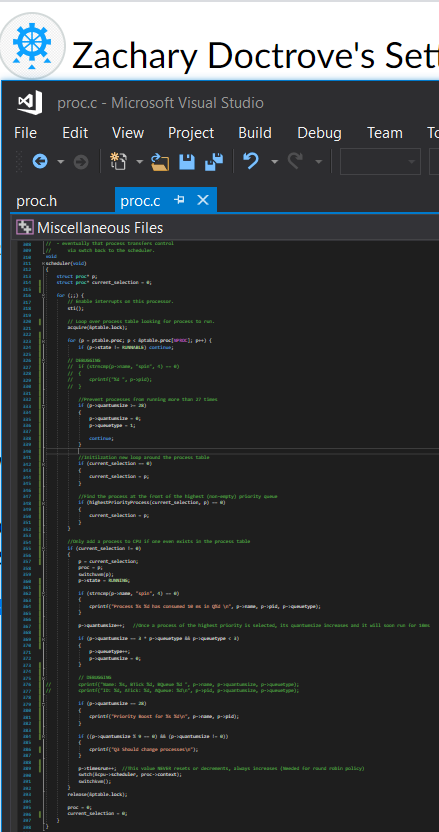


**(Page 2)**



**(Page 3)**



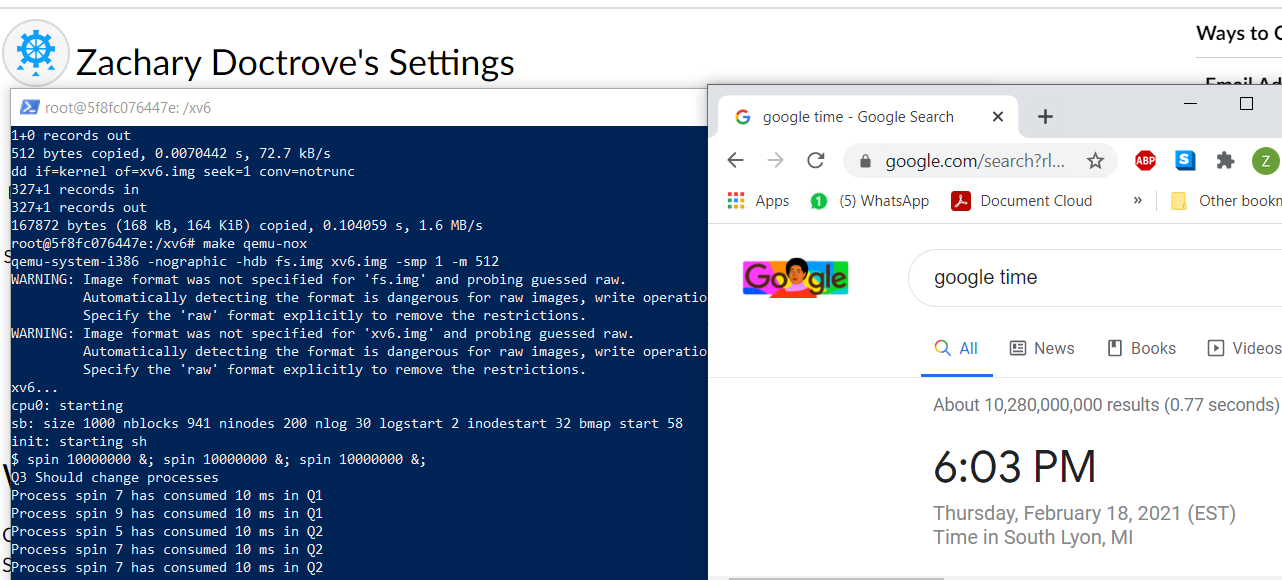
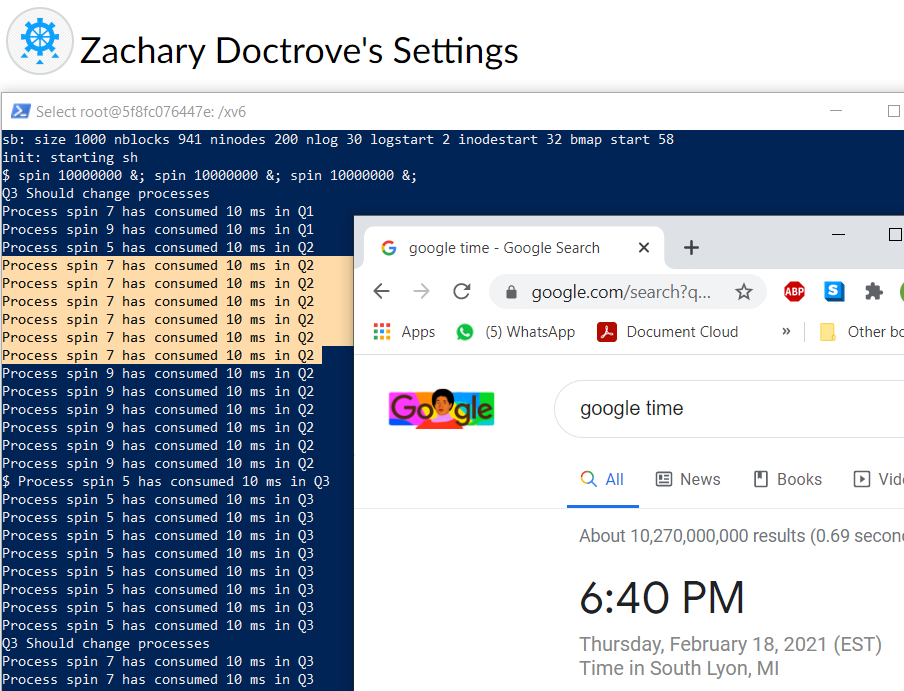
**(Entire scheduler() for reference)** 

**Section 3: Testing Results and Explanation**

First few seconds after entering the command “spin 10000000 &; spin 10000000 &; spin 10000000 &;” into xv6. This trio of processes runs three spin processes at the same time, which can be seen below with Process ID’s 5, 7, and 9.

“Q3 Should change processes” is called when a processes quantumsize equals 28, signifying a need to send the process back to queue 1. It is abruptly used before any of the spin processes even run, as the initialization processes run beforehand.

When the processes first begin running, their order and priority are quickly sent to Q2 and eventually Q3 for an extended period.

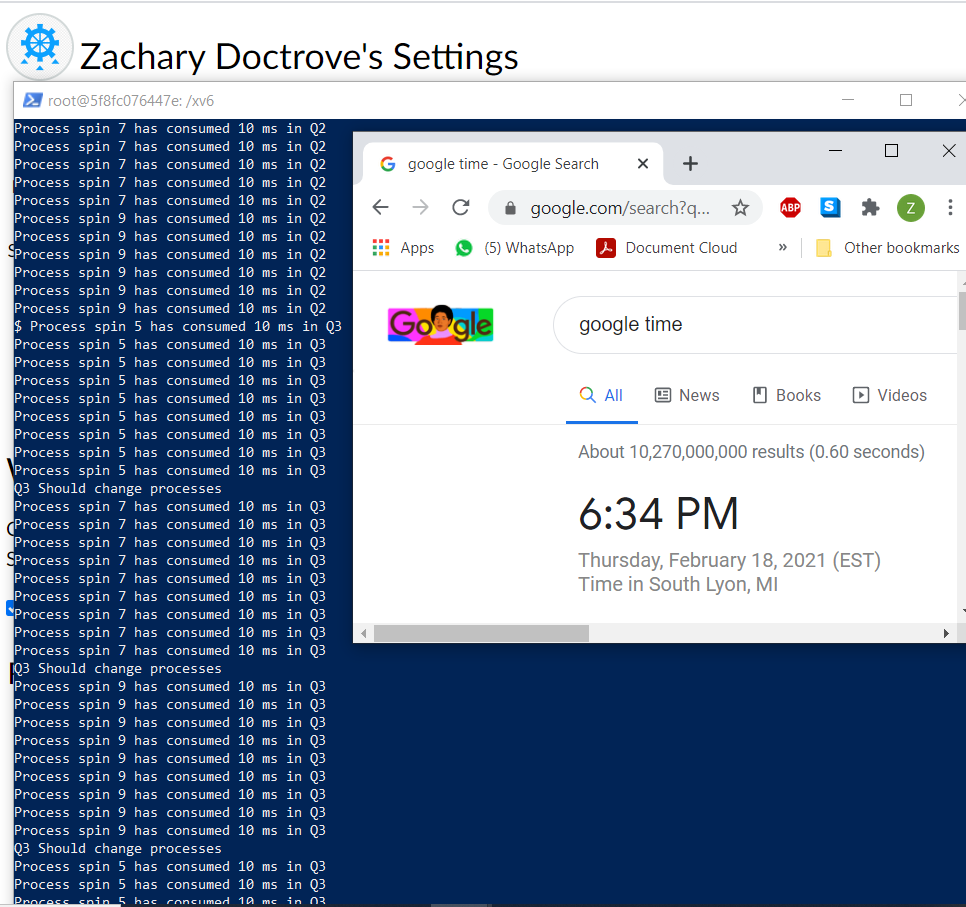


Each print statement is the result of one loop through the process table. For a given queue, the number of loops / printf() statements is based off of quantum size. Q1 = 3 Printf(), Q2 = 6 Prinf(), and Q3 = 9 Printf() statements before switching to the next process.

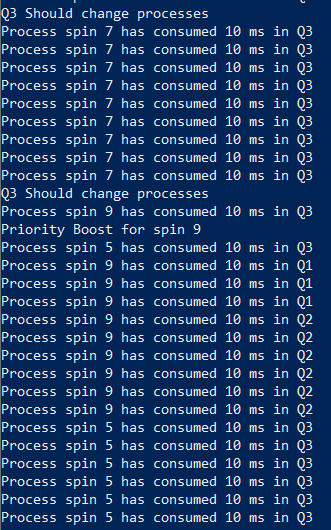
Due to the initialization steps, the processes in Q1 and Q2 do not show up well.

As such, the first fully visible quantum size is seen when “spin 7” reaches Q2 and is on the CPU six times in row (with interrupts between each time)

After a while, all the processes end up in Q3. They all get a chance to run on the CPU in a round robin-fashion (spin 5 🡪 spin 7 🡪 spin 9 🡪 spin 5, REPEAT). Since the quantumsize of Q3 is 90ms, each process runs nine times before they must give up the CPU.





Due to priority boosting, once a process in Q3 has run on the CPU for 27 times (three separate 90ms rounds on the CPU) it then has its queuetype set back to 1 (Go back to Q1) and its quantumsize reduced back to zero.

Both **spin 5** & **spin 7** have their priority boosts occur at nearly the same time. Because spin 9 is in Q3 while the other two processes have been promoted to Q1, spin 5 & spin 7 quickly takeover the CPU for a bit.

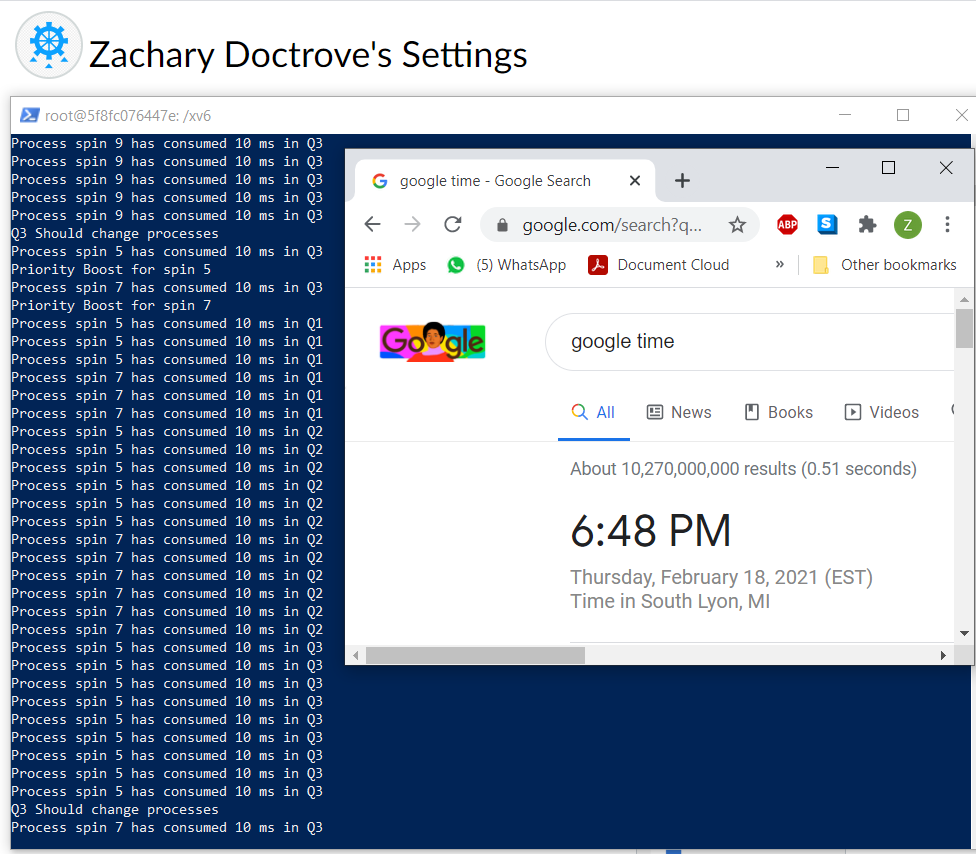


Note how spin 5 came first, so every queue change begins with it. For three times, six times, and nine times spin five was always ahead of spin 7 due to the FIFO policy of Q1 & Q2



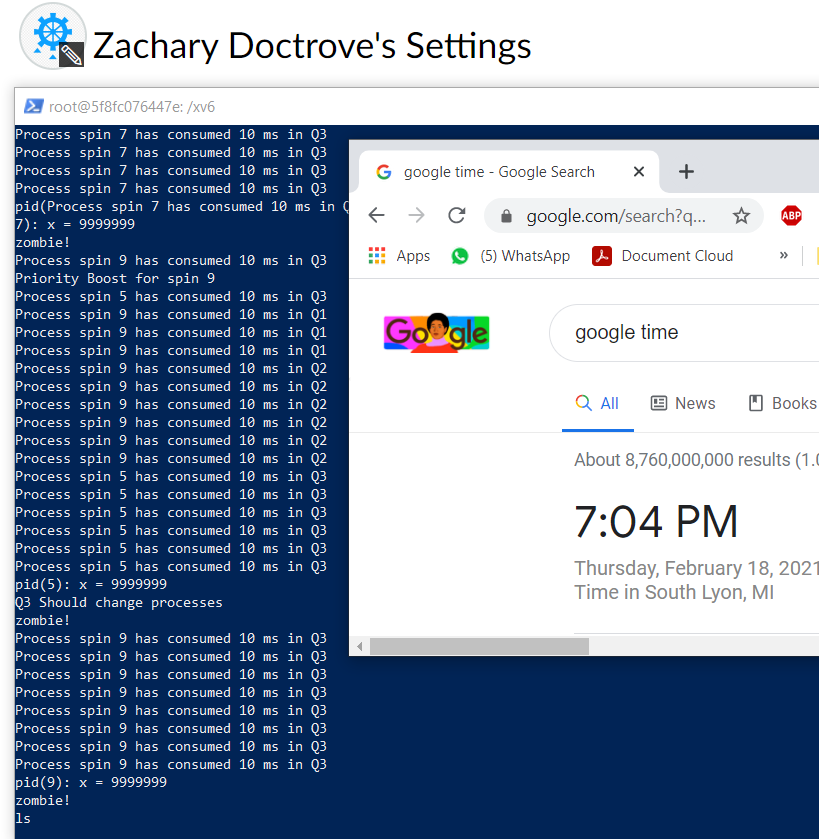
Eventually both spin 5 & spin 7 end up where they started: in Q3. Once this happens, spin 9 will once again be able to complete for turns. Funnily enough, once spin 9 gets a chance to run it too is ready for a priority boost to hog all the CPU to itself.







After an extensive period of simply demoting and boosting these three processes to the three queues, eventually all the processes finish up their tasks and leave on their own fruition. In the order of first to last; spin 7, spin 5, and spin 9 all exit from the scheduler and are no longer considered for running due to the fact that their process states have changed from RUNNING to RUNNABLE and eventually to ZOMBIE.





**Section 4: Contributions made by each member.**

Bryan Pierce: Implemented the main structure of the gtoet() based system in the scheduler.

Zachary Doctrove: Fixed bugs in the code and finalized the gtoet() function.