## **Introduction**

We decided to implement a Lottery Scheduler in place of the current Round-Robin xv6 scheduler. We chose to do this because the implementation seemed straight forward and we believe that a lottery scheduler works better in picking processes to run then regular round-robin. In the end we were able to fully implement lottery scheduling into our xv6 scheduler and were able to test that it picks processes to run at random.

We believe that we should receive an A on this project because we were able to implement the lottery scheduler fully and our test cases worked consistently. It followed our rules where higher priority process were picked more often, and tickets were adjusted when needed.

**Design**

The design of our project is that processes are created and assigned a random priority (if not being set in the processes function). The tickets are assigned based on priority \* 100. When the scheduler runs, a threshold is created that will choose a random number from 1 to 1000. We chose to do it this way because it won’t require us to make an array that holds each ticket for each process and have to randomly pick that specific ticket. We believe that this implementation is better. For all current runnable processes, their ticket value is added to a running sum and once a process comes along that brings that running sum equal to or over the threshold, we run that process. If no process is ran, we keep track of the process that has the most number of tickets and call it the highest priority process. We decided to add this so that we ensure that if there is a runnable process we are able to run it when our processes don’t meet the threshold. We then change the threshold so that it will be able to run processes with our current tickets in circulation. When a process yields it is deducted tickets up until its ticket total reaches the minimum amount of tickets (10). If a process calls userYield() they will be awarded 100 tickets because we want them to run more often because it is a fast process.

## **Assumptions**

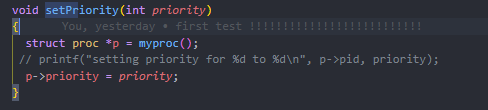
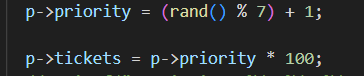
We did not make any assumptions we are aware of.

## **Implementation**

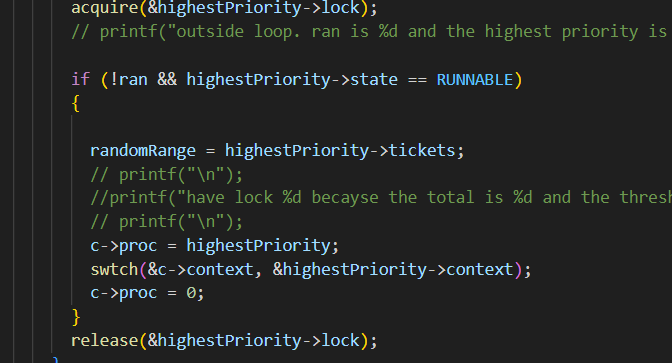
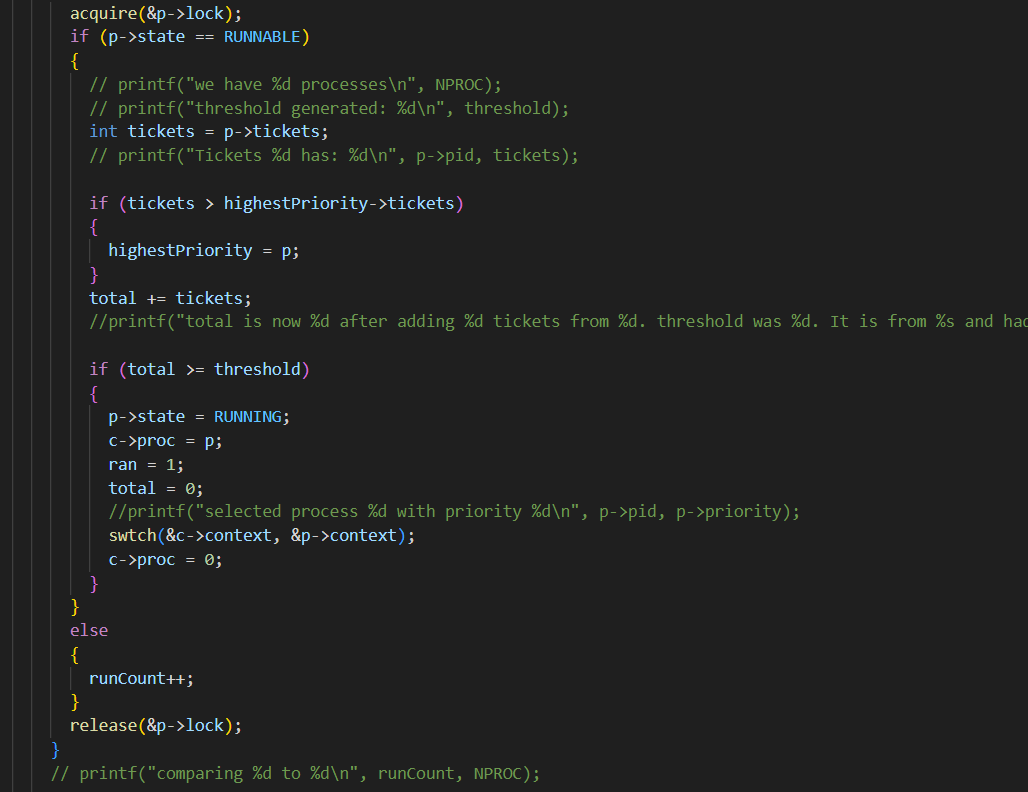
To implement a Lottery Scheduler into xv6, we had to implement some new methods and make some changes to currently existing methods and headers beforehand.



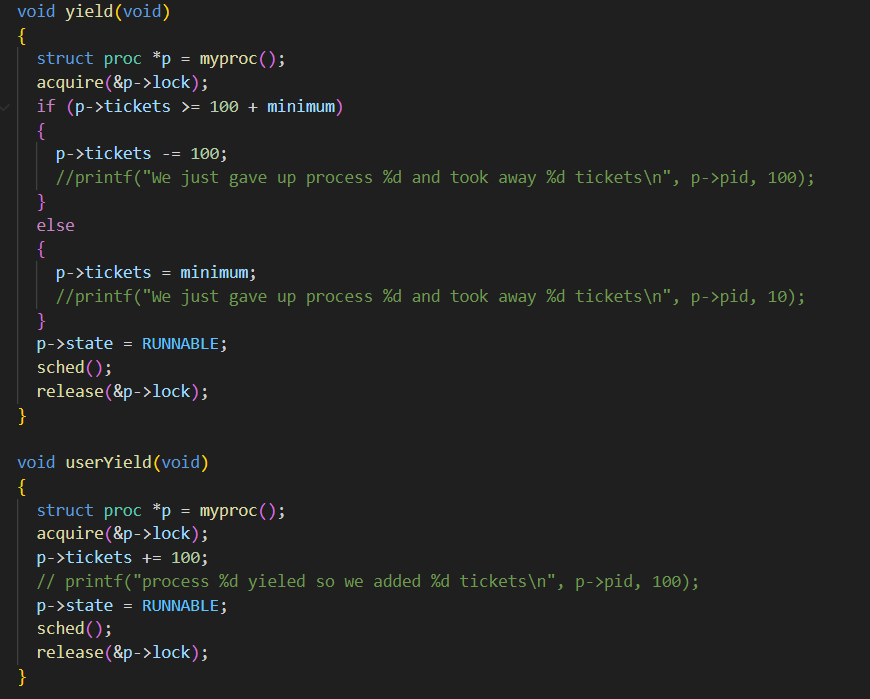
* First we added fields to the proc struct that were the priority of the process and the number of tickets the process had. We had to implement a random number generator that would randomly choose our threshold which would pick the process that ran.



* In the allocproc function, we would assign a random priority to our process and then assign it priority \* 100 tickets.



* In our scheduler we keep track of the process that has the most tickets so that if no process ran because it did not meet the threshold, that process will run and then we would readjust the max that the threshold can be so that we are still able to run processes. As we go into the for loop, we first check if the current process that we are looking at is runnable. We then keep a running sum of tickets so that when a process comes along that has enough tickets to add to the running sum that brings it equal to or greater than the threshold, that process is run.



* In the yield function, functions are deducted 100 tickets up until it reaches the minimum of 10 so that those processes are still able to run. We implemented the function userYield so that processes that finish early are able to earn more tickets for being a faster process and they will run more often.



* Finally when a process is freed we set its priority to 0 and its tickets to zero so that it will not be run anymore.

## **Test Cases**

We implemented several test cases to test that our scheduler works as advertised.

In xv6test we do a basic test. The first process has a long loop so we give it a low priority while the second process has a short loop so we give it a high priority. We also throw in a sleep call.

In xv6test2 we test 50 forks. So we have 50 processes all given a different priority from 1-50.

In xv6test3 we are testing that we can set the priority for our different processes and expect the higher priority processes to run more frequently then the lower priority processes. The two processes are runnin the exact same code, but we give one a priority of 7 and the other 2.

In xv6test4 we test giving a lon process only a few tickets, so they will be at our minimum fr a long time. We also give a very short process a lot of tickets in hope that when it frees, all its tickets get released

In xv6test5 we test our urserYeild function to make sure it works as expected. It should add tickets whenever called on. We give each process the same priority but the userYield will cause the first process to have more tickets.

## **References**

<https://rhit-csse.github.io/csse332/notes/session38/>

<https://www.usenix.org/legacy/publications/library/proceedings/osdi/full_papers/waldspurger.pdf>

https://pages.cs.wisc.edu/~remzi/OSTEP/cpu-sched-lottery.pdf