Design Doc

• Describe your changes to the xv6 code, Name the files your modified and briefly describe the changes

The files modified were proc.c, proc.h, pstat.h, trap.c, MAKEfile(files for system calls and test files), and the test files.

Pstat.h: created file to hold the sched_stat_t struct which held the priority, duration, and start tick for the processes.

Proc.h: Added the times and ticks array, the wait time variable along with ticks, total ticks, num_stats_used, and the sched stats array.

Proc.c:

Created counter variable for each queue along with the q0,1,2 arrays for the processes. Made flag variables as well.

Allocproc:

When a process is found, we now instead check each of the queues and if there is an unused process, remove it from the queue. Once removed from the queue need to shift the queue left as well. Initialize it to be able to run in the kernel and add it to the first priority queue.

Scheduler:

Instead of looping through the process table, change it to loop through each of the queues. Once the first runnable is found you would do the normal scheduling process. But once it returns to the kernel must set duration, times, ticks, numstats used, and other variables to enable the MFQ to work properly. If not found in the first queue it do the same process in the following queues. At the end of searching each queue boost is also called. Although in the last q boost is called before the shifting left that is done after a process is scheduled. Will elaborate more on the specifics of the scheduler in the MFQ policies section. But the gist of changes in the scheduler is that it changed from round robin to and MFQ.

Boost:

Boost is a function at the end of the proc.c file that will boost process that have been waiting for more than 50 ticks. Will also elaborate more in the boost mechanism policy.

Getpinfo:

Getpinfo takes the process id inside. It first acquires the lock, has a for loop to go through all processes in the ptable. If the pid is found, you break. If it is not found, you release the lock and return -1. Also, through the function, you print out helpful information such as the pid, wait time, and details such as start, duration, and priority.

Trap.c:

trap c was changed to check that before yielding that the process meet the MFQ policies at each level of the queue.

Makefile: added ability to run tests

Files for system calls:
We had to add system calls such as getpinfo to important files such as usys.S, syscall.c, syscall.h, and sysproc.c after implementing the function in proc.c

▪ Clearly explain the changes you made as a part of Task1 to implement:
1. MFQ policies

POLICIES: The scheduler has 3 levels, q0,1,2. On every timer tick the highest priority runnable process is scheduled to run. The scheduler will schedule processes in the same priority level ina round robin fashion. The time slices are 1,2 and 8. When a timer tick occurs, whichever process was currently using the CPU should be considered to have used up an entire timer tick's worth of CPU. When a new process arrives it should be placed at the end the priority 0 queue. At priorities 0, and 1, after a process consumes its time-slice it should be downgraded one priority level. Whenever a process is moved to a lower priority level, it should be placed at the end of the queue. If a process wakes up after voluntarily giving up the CPU it stays at the same priority level. It should not preempt a process with the same priority. The scheduler should never preempt a lower priority process if a higher priority process is available to run. When a process is moved to a lower priority it is added to the end of the lower queue and the current queue shifts left to replace the moved process, adjusting vars as needed.

2. Priority boost mechanism

BOOST: The boost mechanism is to avoid starvation. After a RUNNABLE process has been waiting in the lowest priority queue for 50 ticks or more, move the process to the end of the highest priority queue. When boost is called it loops through q2 to find runnable processes(that was not the last process scheduled) and increased their wait time by the duration of the last process. If the wait time is above 50 ticks it promotes it to the highest priority queue and shifts q2 left adjusting variables as needed.

3. Workload description for your test programs:

There were three tests that we ran in total. Test1, we implemented a mixed workload that includes an IO intensive process and a CPU intensive process. Initially we fork, and the children, we print out a bunch of lines using a newline to create a IO intensive process. Then, to create a CPU intensive process by creating a volatile variable and doing a lot of arithmetic operations. For test 2, we implement a workload that demonstrates how priority boost can improve performance of a long running CPU bound process. We call a fork in the beginning, and we only run on CPU intensive operations on the parents. And lastly for test3, we implement a workload that demonstrates how MFQ scheduler can be gamed. This test file is pretty simple where we fork once, and for the parent, we do nothing, and if it is a child, we run a for loop where we sleep in each for loop iteration. Then, we call getpinfo and exit.