XV6 RISCV (Modified)

SCHEDULING ALGORITHMS:

- ### FCFS:
  - First, disable the preemptive scheduling for FCFS scheduling. For doing this, go to `trap.c`. Under `usertrap()` and `kerneltrap()` place the `yield()` section of code under `#ifdef` as follows:

    ```
    #ifdef FCFS
        //relevant code
        yield()
    #endif
    ```

  - Go to `proc.h` , and under `struct proc` , define a new variable `start time`.
  - Go to `proc.c`. Under `allocproc()`, initialise this newly created variable as `p->create_time = ticks`.
  - Next, traverse the process table, and find a process with status `RUNNABLE`, which has the shortest start time.
  - Schedule this process, while acquiring and releasing the locks appropriately.
- ### PBS:
  - Under `struct proc` in `proc.h`, define six new variable `time_stopped` and `time_stopped_temp`, `times_chosen` , `static_priority` , `dynamic_priority` and `niceness`. Also, disable preemption as discussed above.
  - Whenever the process goes to sleep, it enters the `sleep()` function in `proc.c`. Over here, set `time_stopped_temp = ticks`. Now go to `wakeup()`. From here, we can find the time for which the process was sleeping. So, just do `time_stopped += (ticks - time_stopped_temp)`.
  - The `waitx()` function already checks for the run-time under the variable `rtime`, so we need not implement is separately. Using `p->rtime` and `p->time_stopped` we can calculate niceness.
  - Now just scan the process table, and schedule the appropriate process according to the priority order <dynamic priority, number of times it was scheduled before, start time>. We can keep updating `dynamic_priority` , `niceness` and `times_chosen` here itself.
  - `setpriority()` can be easily implemented by making a stub in user space, and changing the appropriate files as we did in spec 1.
- ### MLFQ:
  - Make a `struct Queue` which supports the functions : `push()`, `pop()`, `front()`, `eraseq()`
  - Make an array of 5 elements (say , mlfq[5]) , where each element is of the form `struct Queue`.
  - Traverse the process table, and place each `RUNNABLE` process into the appropriate queues.

- Under `trap.c` , `yield()` the process, when there's a timer interrupt , and it's queue time is over.
- Implement `ageing()`. The `eraseq()` function that we implemented earlier will be quite useful here.
- If the process voluntarily relinquishes the control of CPU, it is removed from the queue. It is rescheduled to the same queue level later. This helps in avoiding useless wait time for the next process in same/lower level.

Analysis(schedulertest):

	rtime	wtime
Round Robin	18	114
FCFS	33	40
PBS	17	106
MLFQ	19	172

(MLFQ was checked on 1 CPU, while others were checked on multiple CPU's)

FCFS has has the highest rtime, and lowest wtime as expected - as it spends a lot less time in context switching.

MLFQ has considerably high wtime due to two reasons - single CPU and also it takes a lot of time to decide which process to run next.

Round Robin also has a considerably high wtime because of regular context switching.

Round Robin and PBS have almost same wtime's and rtime's. This is because - although PBS is non-preemptive, updating niceness, and selecting process on basis of dynamic priority takes some time.