XV6 Project Phase 2:

Make a branch named “phase_2” for this phase of the project and commit all the things you change on that branch. If you want to do the part 3, make a branch “phase_2_bonus” for it. **Make sure to check “Important Notes” at the end of this document.**

PART 1:

Add a new system call to xv6 that can be used when you try to implement your scheduler functions.

The first step is to extend the current proc structure and add new fields stime, etime, iotime and rtime for start time, end-time, I/O time and total time respectively of a process. When a new process gets created the kernel code should update the process creation time. The run-time should get updated after, every clock tick for the process. To extract this information from the kernel, add a new system call which extends wait. The new call will be:

```
int waitx(int *wtime, int *rtime)
```

The two arguments are pointers to integers to which waitx will assign the total number of clock ticks during which process was waiting and total number of clock ticks when the process was running. The return values for waitx should be same as that of wait system-call.

PART 2:

Replace the current round robin scheduler for XV6 and replace it with a priority based scheduler. A priority based scheduler selects the process with highest priority for execution. In case two or more processes have same priority, we choose them in a round robin fashion. The priority of a process can be in the range [0,100], smaller value will represent higher priority. Set the default priority of a process as 60. To change the default priority, add a new system-call set priority which can change the priority of a process.

```
int set_priority(int)
```

The system-call returns the old-priority value of the process. In case the the priority of the process increases (the value is lower than before), then rescheduling should be done. You can use yield() system call. Add to your report a small example and a comparison of your current (priority based) scheduling policy and round-robin approach.

PART 3 (BONUS):
Implement a Multi-Level Queue Scheduling which has 3 queues:

- First Queue: The first queue will hold the high priority processes. The processes in this queue will be scheduled according to the guaranteed scheduling policy (Implement any fair scheduling policy of your choice).
- Second Queue: The second queue will hold medium priority processes. The processes in this queue will be scheduled according to the FIFO round robin scheduling policy.
- Third Queue: The third queue will hold low priority processes. The processes in this queue will be scheduled according to the round robin scheduling policy. A process with a higher priority will be preferred and run before a process with a lower priority.

Add a sys-call `nice()` which can decrease the priority of a process (returns 0 on success, else -1) and can be used to demonstrate the results.

Hint: First, find the most suitable structure to store the priority of the processes!

Output: Output the results in the form of a report for a small example which demonstrates the correct working of the scheduler. Take the advantage of the output of `waitx` system-call, and report the average turn-around time for each of the process.

This part is not required doing it will have bonus points. Also, if you can’t implement it, you can think and write about it in the report.

Important Notes:

You should **commit and push** the changes and the progress you have on the source code's repository. (we expect more than 10 valid commits). Make sure you **add comments** to your code whenever you add new parts to the source code.

Besides committing on the repository, you should upload to 3 things in a zip file to Quera:

1) A pdf report on **how** to add this system call to xv6 and talk about **the files** that should be modified and **what** each file does.

2) The **changed files** of xv6 on the final commit in the repository before the deadline. (Most files of the xv6 won't be changed.)

3) A text file containing the link to your repository.

Your projects will be graded based on:
- Your report
- Your comments on the code
- Your test file
- The fact that your codes work or not
- Your commits on the repository

Your grades will be zero if:
- It's not done by you.