# Operating Systems- COL331
# Assignment 2

Ankesh Gupta
2015CS10435

## Part 1

**Pointers on printing currently running processes**

1. We modified printing system call which now also traces priority of each active process.
2. Firstly, another field name *priority* was added in ***process control block***. In our code, *proc struct* was updated.
3. The value of priority was initialised to 5 at time of process creation.
4. Printing was modified to take care of printing priority as well.

**Pointers on setpriority System call**

1. As a continuation, we created a system call which can *setpriority* of currently active processes.
2. A system call was implemented which takes *process_id* and *priority* as parameters.
3. If the priority is out of range [1-20], then an error message is flagged.
4. Otherwise, the process that matches the pid gets its priority updated.

## Part 2

**Pointers on implementing Priority Scheduler**

1. This part was interesting as we replaced the default ***round-robin*** scheduler of *xv6* with a ***priority based scheduler***.
2. For this, the *scheduler* method in *proc.c* was modified.
3. The tricky part was to ensure *round-robin* property amongst the same priority processes.
4. For this, while scheduling what was done is that we start finding the *first-highest* priority process, starting with next process of currently ***context-switched process***.
5. The above was done in a cyclic fashion to ensure round robin property.
6. If there was some process with unique highest priority, it would get picked.
7. Otherwise, it would pick up processes with same highest priority in cyclic fashion.
8. An important point is that although this new scheduler has same ***space efficiency***, the time complexity has increased by an ***order of magnitude***.

9. For each scheduling, we run through the entire process list and select the one with the highest priority.
10. The above step could be efficiently implemented using a ***heap***, in our case a *max-heap*. This would have reduced the ***linear factor*** to ***logarithmic***. But since, number of processes was low, and it was more for a proof-of-concept experiment, I went with the linear search.

# Part 3

**Pointers on preventing Starvation**

1. A system call was implemented which returns priority of a process, given its *process_id*.
2. The implementation is similar to the one in which we were setting priority of a process.
3. The main part was incrementing priority of a process, if it has remain inactive for a long time, i.e., if the process hasn't been scheduled for long because of lower priority.
4. For this, a variable counter was introduced, which records how many *context switches* has been noticed by the process since its *inception*.
5. If this limit exceeds a threshold(50, in our case), priority of the process is increased by 1.
6. Since processes with same priority respects ***round-robin***, in some way, we are ensuring *fairness* together in our priority scheduling, which was inherent in ***round-robin*** type scheduler.
7. Whenever a context switch takes place, counter of all process with runnable state is updated.

***Note: Most of the scheduling works are only for Running and Runnable processes. Also, setting priority subsumes that the process is in valid state. This hasn't been explicitly handled.***