REPORT

PART – 1:

DESIGN:

Initially, we need to add a new attribute “priority” to “struct proc” to store the priority number. Now let’s take the default priority to 5. We do this in “allocproc(void)” function by adding “p->priority = 5” which is present in “proc.c”.

Now, we create the syscall to change the priority of the current process. In “defs.h” declare function setPriority by adding “int setPriority(int *)”. Now in “sysproc.c” add function “int sys_setPriority(void){...}”. Next in “proc.c” define function “int setPriority (int *prio_num) {...}”. Here the “myproc()->priority” is changed to function argument “prio_num”. The valid priority value is between 0-and 9, but if the argument passed to this syscall is not 0-9, then the syscall invokes a panic message.

Disclaimer: the setPriority () syscall takes the integer pointer as input. So please store the integer in the variable and pass its address to the syscall.

What can be a problem with supporting such a system call in any OS?

- This will lead to starvation as a process can itself decide its priority. So, it can make itself run at the first step. Then essential processes can be dominated by some arbitrary process which increases their priority.
- If we take a case where all processes set their priority to 0(0 is the highest priority), it will lead to regular round-robin scheduling.

How could we mitigate this?

We can create a priority distributor, which will be in kernel control, and the setPriority syscall can only be called by this. So, this priority distributor prioritises the processes considering all the parameters rather than just setting a priority at our will.

PART – 2:

DESIGN:

Process scheduling is implemented in "proc.c". So, we are modifying this to implement priority-based scheduling. We know that the xv6 follows the round-robin scheduling algorithm. It uses a for loop to schedule processes in the same order they exist. Now we add another for loop inside, which searches for the first highest priority process and schedules it for running. And on the next iteration, it selects the following highest priority process. If all have the same priority, it just follows the outer for loop, and this for loop will have no effect, so it continues with round-robin scheduling.

What is the problem?

If the highest priority process is, at last, it comes out of the loop, and the ptable lock gets released, and new processes get added; if the new process has the higher priorities, then every process in the ptable. They go into starvation sometimes.

How can it be mitigated?

- We can make a dummy process always to be the ptable so that it will not be run, but it will occupy the final position so that the primary process won't be in the last part of the ptable. This will ensure no new processes are added until the existing processes are completed.
- And the second method is to increase the priority by 1 factor for every iteration. It makes the process not go to starvation.

Disclaimer: by keeping NCPU is 8; I can see the merging of statements, but not when NCPU is 1.

LEARNINGS:

- How the processes are scheduled.
- How to add new attributes to the process details.

- How to add a syscall to change the characteristics of the process arbitrarily.