REPORT

Part 1:

Problems with such a syscall:

- A malicious process can update it’s priority using the syscall, and by doing so, many processes which are actually of higher importance, will be schedule to run after the process completes execution. If such a malicious process have a complex code, then the really needed processes get delayed.
- Sometimes the user processes can get higher priority than the much need kernel processes.

Solutions:

- One solution might be to given max priority to the really important processes. So even if the wrong process is given high priority, then also the required process would be schedule as it of max priority.
- Another solution might be having different max priority levels for user and kernel processes, so that kernel, if wanted, could be given higher priority always.

Part 2:

Problems with priority based scheduling policy:

- The main problem with priority based scheduling is Starvation. Some processes can keep on entering into the system which are of higher priority than some low priority processes, so that the lower priority processes cannot get a chance of executing for a long time.

Solution:

- The aging technique could be used to mitigate the problem. A process’s priority can be increased on waiting for long time so that it will eventually get higher priority and get a chance of execution in the priority based scheduling algorithm.

Implementation:

Part 1:

- It is just the implementation of a syscall as done in previous assignments.
- But while updating the priority in the sysproc.c we need a lock, as we don’t want the process to change it’s priority once the scheduler started scheduling. So we can write another function proc.c, and update the priority there using the pid of the process, by acquiring the process table lock.
- And also error checking is done, for the cases when no argument is passed in the syscall, also to check if the provided priority is in between the values.
- We also have to set default priority for the process while it is being created, so in the allocproc function, i have set the default value to 5.

Part 2:

- We have to modify the scheduler function in proc.c which intially is implementing round robin scheduling.
- Once the scheduler starts running, we have to find the process with maximum priority by checking all the entries in the ptable, and we have to schedule that process.

- This can be done by running a for loop through all entries and keeping in check the process with highest priority.

Observations:

- A process with lower priority is given last preference in the priority based scheduling algorithm. So, if we test the scheduling, we can see that a process which is runnable and has the lesser priority is executed at last, compared to others with higher priority.

Learning from the assignment:

- I got a good clarity on how a scheduler works.
- Also, i have learnt how to add different scheduling strategies
- I have got more clarity on how priority based scheduling causes starvation.

Note:

- If the number of CPUs is 1 in param.h, then the output comes in understandable way
- If the number of CPUs is 8 i.e., defalut, then the output contains lines merged with each other.