Assignment-7 Report
OS2 – CS3523

Name: *Taha Adeel Mohammed*
Roll Number: *CS20BTECH11052*

Part-1: System call to assign/change the priority of any process.

- Firstly, we declare a new system call **setPriority()** and add the user program *setPriority.c***,** similar to how we had done it in previous assignments, as shown below:
  - Give the system call an appropriate number in *syscall.h*
  - In *syscall.c*, externally link the system call function and add it to the static array containing other system calls.
  - Also declare it appropriately in *defs.h*, *user.h*, and *usys.S*
  - Edit the *MakeFile* to integrate the user program with the rest of the software.
- We add a new member to the **proc struct** in *proc.h* to store the priority of the process.
  - Then finally, in *sysproc.c*, in the **sys_setPriority()** function, we read the parameter given to the syscall and then call **setPriority()** defined in proc.c
  - In the **setPriority()** function in *proc.c*, we check whether the given priority is valid, and then set the value of the current process's priority to the given value.
- To set the default priority of a process, we edit the **allocproc()** function in *proc.c* which is called when the process is spawned. Hence the default priority is set as 5.

Problems:

- If such a system call is supported in any OS, i.e. if the user is given permission to edit the priorities of the processes freely, we could run into errors.
- The user could change the priority of very important OS processes, hence delaying their execution and possibly breaking some stuff.
- Also the user can set the priority of his user processes as much higher than required, hence delaying critical processes.
- We can mitigate this problem by not allowing the user to set the priorities of processes without having sudo privileges.

Part-2: Scheduling policy based on priority of processes

- To implement a scheduling process to schedule higher priority processes earlier, we edit the **scheduler()** function in *proc.c.*
- The **scheduler()** function loops over the process table and starts the next process.
- We edit it to go over all the processes in the table and schedule the process with the highest priority next.
- Also xv6 has round robin scheduling by default. In our edited code too, the process with the highest priority is interrupted after a while and the next process with the highest priority is scheduled in a round robin fashion.

Problems:

- The above method of setting priorities also has the drawback that all kernel processes are given the same priority(= 5). Hence the more important processes aren't given more priority while scheduling as they should be.
- Also, if the user creates a lot of process with higher priority than the kernel processes, the kernel processes could starve, hence leaving crucial tasks unprocessed.
- We can mitigate this by giving the kernel processes a lower priority by default as compared to user processes.