REPORT :

In first part of the question , I added another field to the process state structure (Priority_Number) . While allocating memory to a process (new) , I made Priority_Number as 5 for it (I took 5 has default Priority Number).

For Changing Priority Number of a process , I created a new system call , which takes input and checks whether input is between 0 and 9 if it is , then it changes priority number of the current process to it's desired number ,given as input and it returns 0 or else it will return -1.

And while changing priority number of a process , which is shared variable . To maintain consistency , I imposed locks on it which is the same locked used in scheduler function .

What can be a problem with supporting such a system call in any OS? How could we mitigate this problem ?

The problem is ,whenever we change the priority of a system , we should again check the more priority process to run on cpu . If we do not check , then there is no use of priority

number . We can solve this by making current process state to runnable , by doing this scheduler will schedule process with higher priority after modifying.

In 2nd part of the question , I implemented priority scheduling in xv6 .

For that , there is to be separate field in process structure , which we added in first part of the question. In general xv6 follows , round robin scheduling . In scheduler function , I made changes in scheduling the process .

Now I check for the process , which is ready to run and with higher priority and assign cpu to it. Now if two process has same priority I schedule them using round robin (Which was done earlier by xv6 using that code , I added priority part inside the round robin part to solve this) .

To test the implementation of the features , I created a user file , in which I created child process using fork() . Now if multiple cpus are there ,then many process could run and we may not be able to see the impact of our modifications . For that I changed no of cpu to 1 in param.h file . Now only one process can run at a time and we can see the differences .

To get the exact impact , I created child process and printed statements in it. When they are of equal priority . i.e default priority , we can see child process to run first and print it statements first . Now I increased priority of parent process using the system call which we implemented in first part of the question , then we can see that parent was scheduled first as it ' s statements were printed first.

There is a problem with such a priority based policy. What is that problem? How can it be mitigated ?

The problem is starvation of the process , if a new process always comes with higher priority then older process must wait for it completion , if such process comes at a higher rate the older process might never get the chance and this leads to starvation of the old process . We can solve this by increasing the priority number of the old process after fixed period of time or whenever it tries to schedule a process.

From this assignment , I learned many things like how xv6 actually schedules process , and how we can modify it to run our preferable algorithm and many more ……..

==========