Charley Conroy

Professor Ubal

Operating Systems

5/9/22

Report

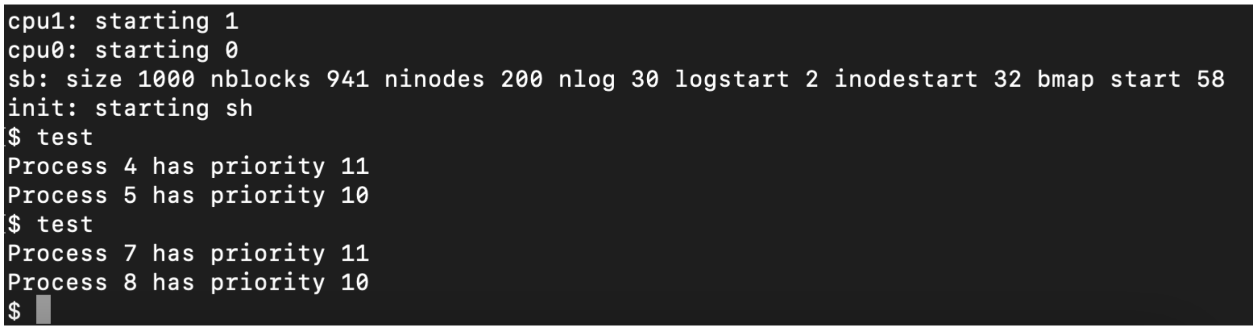
I decided that I wanted to implement the nice system call, which ended up being really interesting. It involves changing the scheduler and creating a new system call, which ended up being different from how we did things in homework 6. I think it was really worthwhile and interesting to research and I feel much more comfortable with xv6 now.

The first step in adding the nice system call is to change the scheduler. Currently xv6 uses a round robin scheduler, so I needed to adjust it to a priority based one. First I went into the scheduler() function in proc.c and I changed the line struct proc *p in the scheduler function to have two processes *p and *p2 as well as another proc struct called *firstproc to hold the process with the lowest priority value to be the current process. For there to be a priority value I go into proc.h and add an int priority element to the proc struct. Then in allocproc() in proc.c, which is where the state and pid for p are initialized, I initialize the priority to 10. Going back to scheduler() I then used similar code to the loop that looks for processes to run and has it assign *p to the *firstproc I made, or *p2 if a process has first priority. Then I assign *p to be the value that *first holds and the scheduler continues as normal. This makes the scheduler act as a modified round robin that takes priority into account, which should work well with adding the nice system call.

The next step is to decide how I want the nice syscall to work. The unistd library apparently changed how the function works somewhat recently so I went with the updated version. My nice() takes 2 int values: the pid and inc. The value inc is how much the priority should be shifted by, which notably should take negative values. The system call then makes the process priority value equal to itself plus inc. This means that when the scheduler checks the priority of this process it should be different from the default value of 10. If the value is above 10, that process has priority over any processes with a lower priority value. This differs from the actual nice function which has higher priority values equal lower priority, which is also why scheduler() uses *p2 if its priority value is greater than that of *p. I think it makes more sense for a higher priority value to mean a process has higher priority.

Now I was confused about how to implement this because nice differs from what we did with date on homework 6 in that xv6 doesn't come packaged with a nice implementation. However, I started with the same steps as in homework 6, editing syscall.c, syscall.h, sysproc.c, user.h, and usys.S to include nice as a new syscall. After researching how best to continue from there I found that adding nice() into proc.c along with defining the function in the proc.c section of defs.h would allow me to implement the function. This means I now have a system call that I can call in xv6 to change the priority of a function.

Now for implementation I made a file called test.c that can be called in xv6 by typing test. Test spawns a child thread with a call to fork which in turn also calls fork. The new child runs nice() with its pid from getpid() and an inc value of 0, while its parent runs nice with an inc value of 1. I then run a for loop that cycles for 10 million iterations in both processes and then have them print their pid and their priority. On the first run it should print process 4 has priority 11 and then process 5 has priority 10. Running the test again gives the same output for processes 7 and 8 and so on. The process with priority 11 always starts the print statement before the process with priority 10.



```
cpu1: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ test
Process 4 has priority 11
Process 5 has priority 10
$ test
Process 7 has priority 11
Process 8 has priority 10
$ 
```

This project required me to look deep into xv6 and go beyond what we did for homework 6. I struggled with how I wanted this system call to work, but I'm happy with how my project came out, and I enjoyed the opportunity to work with xv6 again.