CS 451 – Operating Systems

Lab 04 Assignment

Due: 11:59 PM on Tuesday, 18 August 2020

You are to submit your files, to the Lab 4 folder on Blackboard by the due date and time.

Objective

- To understand priority scheduling and implementing priority scheduling mechanism in xv6
- Add a priority attribute to a process in xv6
- Change a priority of a process

Note: This lab is an extension of Lab 4 (In order to finish this lab, you have to complete lab 4)

Implement Priority Scheduling

1. Default priority value of a process is 10
2. The smaller the priority value the higher the priority
3. Modify the **scheduler()** routine inside the proc.c to implement the priority scheduler

Below are the steps to follow:

1. Add priority to struct proc in proc.h

```
int priority; // Process priority
```

2. Assign default priority in allocproc() routine in proc.c (you have to place in the appropriate location)

```
p->priority = 10;    //default priority
```

3. If a process is loaded from the shell, make it high priority. This change should be made in exec.c. After line 101 add the following.

```
curproc->priority = 3;
```

4. Modify **procStat** in proc.c discussed in the last lab to include the printout of the priority like the following

```
Name       ProcID     State        Priority
init       1          SLEEPING      3
sh         2          SLEEPING      3
ps         27         RUNNING       3
```

5. Write a user program named **tester.c** that creates some child processes and consumes some computing time. The parent should be always waiting. The child should compute some useless calculations such as following:

```
for ( i = 0; i < 8000000.0; i += 0.01 )
     x =  x + 3.14 * 89.64;
```

   Remember this is a user file, hence you will be required to modify the Makefile. (**Refer to the previous lab**)

6. Add the function chpr() (meaning change priority) in proc.c

```
//change priority
int
chpr( int pid, int priority )
{
  struct proc *p;

  acquire(&ptable.lock);
  for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
    if(p->pid == pid ) {
        p->priority = priority;
        break;
    }
  }
  release(&ptable.lock);

  return pid;
}
```

7. Add sys_chpr() in sysproc.c

```
int
sys_chpr (void)
{
  int pid, pr;
  if(argint(0, &pid) < 0)// built-in functions for passing
                         // arguments into a kernel function
    return -1;
  if(argint(1, &pr) < 0)
    return -1;

  return chpr ( pid, pr );
}
```

8. Add chpr() as a system call to xv6 as discussed in the previous lab (Refer to the previous lab manual)
9. Add the user file chngp.c (provided) which calls chpr. Remember this is a user file, hence you will be required to modify the Makefile. (**Refer to the previous lab**)
10. Also make sure the number of CPU's used is 2. (Look into makefile)
11. Test chngp using tester by creating a few process in the background and use ps command to check the process states and priority. Your output should look something like below: (**run tester always as daemons** 😈)

```
$ tester 4 &
$ ps
         Name    ProcID       State       Priority
         init    1            SLEEPING    3
         sh      2            SLEEPING    3
         tester  7            RUNNING     10
         tester  6            SLEEPING    3
         ps      8            RUNNING     3
```

12. Change the priority of a process using chngp and check the status using ps again

```
$ chngp 7 5
$ ps

         Name    ProcID       State       Priority
         init    1            SLEEPING    3
         sh      2            SLEEPING    3
         tester  7            RUNNING     5
         tester  6            SLEEPING    3
         ps      11           RUNNING     3
```

13. Observe the default round-robin (RR) scheduling by creating a few tester processes in the background and running ps a few times at random time intervals in xv6

```
$ tester  &; tester  &; tester  &
```

You can observe that the three tester child processes are running alternately while the parents are sleeping.

Implement Priority Scheduling.

1. Modify the scheduler function in proc.c to select the highest priority runnable process. The code skelton is provided (**scheduler**) with few comments. Modify the code and implement the priority scheduling.
2. **Observe the priority scheduling.**
    a. Run xv6 with the scheduler and again use tester and ps to see how it works. Use chngp to change the priority of a process. If you have implemented your program correctly, you should observe that the lower priority process is always in running state.

What to turn in

- Compress the parent folder (xv6) along with your created files, to zip and upload the zipped folder.
- A README file specifying the changes you made including a brief description of the implemented scheduling mechanism, the user command and the system call.