Lab 5: Priority-based Scheduler

Instructor: Yaoqing Liu
TA: Garegin Grigoryan

What you should already have

- After Lab 4:
  - Each process has a **priority** field
  - By default, all processes have priority set to 0
  - After **fork()**, the parent has the same priority as the child
  - xv6 has new system calls:
    - **ps_yourname() -** prints current processes
    - **setpriority_yourname(pid, priority):** set priority

Why do we need priorities?

- Processes with higher priority should run first
- Now, the scheduler is running **round robin** algorithm and ignores priorities
- Scheduler is implemented in **proc.c** in function
  `void scheduler(void)`
- It iterates through the process table and runs every RUNNABLE process

Your task

The new scheduler should:

- Lock the global process table first
- Loop the global process table to find the maximum priority among RUNNABLE processes
  - The priorities range from 0 to 200.
- Perform round robin scheduling for processes with the same priority
- Tip 1: Finish Lab4 tasks completely without errors, traps, etc
- Tip 2: Your new scheduler needs to handle scenarios where there is no RUNNABLE process to handle
- Tip 3: Try Changing priority for your testing processes (see next few verification slides)
- Tip 4: Try not changing priority for process 1 and 2 because they are xv6's own processes
- Tip 5: Backup your current scheduler first

How to Verify (1)

- Test your scheduler by running a user program (see test_scheduler.c in next page):
  - Note: modify userspace/Makefile to run the testing program
- The user testing program runs two processes in parallel and they both run a very long loop for calculation
- Run it in background: $ **test_scheduler &**

test_scheduler.c

```c
#include "types.h"
#include "stat.h"
#include "user.h"

int
main(void)
{

  int j = 0;

  int rc = fork();
  if (rc < 0)
    printf (1, "Failed");
  else if (rc == 0){
    while (1){
        j = j + 3.14*6/18;
    if (j >= 300000000)
        break;
    }
  }
  else{
    while (1){
        j = j + 3.14*6/18;
    if (j >= 300000000)
        break;
    }
  }
   exit();
}
```

How to Verify (2)

- Process states: **0:** UNUSED, **1:** EMBRYO, **2:** SLEEPING, **3:** RUNNABLE; **4:** RUNNING; **5:** ZOMBIE
- Initially the ps program will have following output:
  - Parent test_scheduler normally has a lower pid
  - Both processes have the same priority 0
  - Two processes are running at once (two CPUs)
  - ps_yourname process is always running since this is the current running process

```
$ ps_grigorg
pid     name      state     priority
1       init      2         0
2       sh        2         0
5       test_scheduler  3         0
4       test_scheduler  4         0
8       ps_grigorg      4         0
```

```
$ ps_grigorg
pid     name      state     priority
1       init      2         0
2       sh        2         0
5       test_scheduler  4         0
4       test_scheduler  3         0
11      ps_grigorg      4         0
```

How to Verify (3)

- Test your scheduler by changing priorities of **test_scheduler** processes using **setpriority_yourname** program
- Verify that the process with the highest priority is RUNNING (state = 4) using **ps_yourname** program
  - Reminder: higher priorities have lower value
- Testing scenarios:
  - First, set parent's priority to 10 and verify it is always RUNNABLE (state = 3) before completion
  - Second, set child's priority to 15 and verify it is always RUNNABLE (state = 3) before completion

Example output (1)

1. Run the test_scheduler program in background
2. Print current processes all priorities are 1

3. Set the parent's priority to 10

4. Check twice that the child is always running (state == 4, pid == 5)

```
$ test_scheduler &
$ ps_grigorg
pid    name    state   priority
1       init    2       0
2       sh      2       0
5       test_scheduler  3       0
4       test_scheduler  4       0
6       ps_grigorg      4       0
$ setpriority_grigorg 4 10
Priority of the process with ID 4 is set to 10
$ ps_grigorg
pid    name    state   priority
1       init    2       0
2       sh      2       0
5       test_scheduler  4       0
4       test_scheduler  3       10
8       ps_grigorg      4       0
$ ps_grigorg
pid    name    state   priority
1       init    2       0
2       sh      2       0
5       test_scheduler  4       0
4       test_scheduler  3       10
9       ps_grigorg      4       0
```

Example output (2)

1. Lower child's priority to 15 (pid == 5)
2. Run ps_yourname program to verify the parent process is running

```
$ setpriority_grigorg 5 15
Priority of the process with ID 5 is set to 15
$ ps_grigorg
pid    name    state   priority
1       init    2       0
2       sh      2       0
5       test_scheduler  3       15
4       test_scheduler  4       10
9       ps_grigorg      4       0
$ ps_grigorg
pid    name    state   priority
1       init    2       0
2       sh      2       0
5       test_scheduler  3       15
4       test_scheduler  4       10
10      ps_grigorg      4       0
```

Submission

- Take screenshots about your scheduler source code, your compiling process, and the verification process as we demonstrated
- Submit a combined PDF file to moodle
- Leave your source code in the current work directory
- Delete all local files
- Due: Feb 26, 11:55pm (The Monday after Feb break)

Thank You!