COL331, COL633, ELL405: Operating System Assignment 2: Priority Scheduler [12 Marks]

Due Date: 29 March 2018

Update

Make the following changes in the source code

- Change the value of **NCPU from 8 to 2** in file *param.h*
- Change the value of **system_trace_on from 1 to 0** in file *syscall.c*

Download new input and checker files from the following link.

This tar ball contains

assig2_1, assig2_2, assig2_3 and assig2_4 user programs to check the implementation (some hidden cases which will be run after the submission). Do required modifications in the Makefile for this.

out_assig2_1 and out_assig2_2 are the expected output for the respective user programs. NOTE: There is no output file for test case 3 and 4.

check1.py, check2.py, check3.py and check4.py : To check the implementation of the respective user program.

submit.sh: This will create a tar ball which you have to submit on Moodle.

test_assig2.sh: To run the user program. eg: to run the user program assig2_1, run

```
./test_assig2.sh assig2_1
```

Part 1: Add and print priority of the process [3 Marks]

Part 1 (a)Modify sys_ps to print priority [1 Marks]

Your first task is to modify the sys_ps from assignment 1 to print priority for each process. The following format has to be followed:

```
pid: < Process-Id > name:< Process-Name > state: < Process-State > priority:< Priority >
```

For this part, create a new user program, which should in turn call your sys_ps() system call, or you can use assig2_1.c provided by us. The expected output can be seen in the file *out_assig2_1* for reference purposes. We will use some other hidden test cases also.

Part 1 (b) Add sys_setpriority system call [2 Marks]

After you are done with the part a of the assignment, the priority of each processes will be set to a default value (5).

In this part, you will add a new system call (setpriority) for the process to change its priority. It will take 2 arguments, pid and priority of the process, and then set the priority of the process to given value.

After the system call implementation is done, you need a *user program* to actually make the system call. You can also use user program provided by us called as assig2_2.c.

It can be called as:

```
assign2_2 < Process-Id > < Priority >
```

Part 2: Implement Priority Scheduler [5 Marks]

Replace the round-robin scheduler for xv6 with a priority-based scheduler. The valid priority for a process is in the range of 1 to 20. The smaller value represents the smaller priority. For example, a process with a priority of 20 has the highest priority, while a process with a priority of 1 has the lowest priority. A priority-based scheduler always selects the process with the highest priority for execution. If there are multiple processes with the same highest priority, the scheduler uses round-robin to execute.

Create a user-level program to test it. You can use user program provided by us which is called **assig2_3.c** that calls your new system call.

The output generated can be directly checked by running *check3.py*.

Part 3: Starvation[4 Marks]

Part 3 (a) Add sys_getpriority system call [1 Marks]

The system call will return the priority of the given process. It will take pid as an argument of the system call.

Part 3 (b) Handle starvation [3 Marks]

For each process, maintain a counter in the proc structure that is initialized to zero. It will count the total number of context switches. To implement starvation, modify the code of the scheduler such that after each timer interrupt (context switch), it updates the counter. When the counter becomes 50 increase the value of the priority of each process by one and reset the counter to 0. This will ensure that after some period of time, a starved process will be scheduled once.

Create a user-level program to test it. You can use user program provided by us which is called **assig2_4.c** that calls your new system call and handles starvation.

The output generated can be directly checked by running *check4.py*.

Part 4: Two page report.

Create a two page report, briefly explaining the code. This should list any new variables or data structures added by you along with their usage.

Note:

- **Please save your proc.c for Part 3 as proc_3.c. Submit both proc for part 2 (proc.c) and part 3 (proc_3.c).**
- Please make minimal changes to xv6; you do not want to make it hard for us to grade!
- There will be some more hidden testcases on which your code will be evaluated.
- Please make sure that you follow the naming convention mentioned above for system calls, otherwise the test cases will fail and you will receive no marks for that.
- We will run Moss on the submissions. We will also include submissions from other sources (past year or Internet). Any cheating will result in a zero in the assignment, a penalty as per the course policy and possibly much stricter penalties (including a fail grade and/or a DISCO).

- There will be NO demo for assignment 2. Your code will be evaluated using check script on hidden test cases and marks will be awarded based on that.
- No marks will be awarded if you do not follow the required format (naming conventions).

Submission Instructions

- Run
  - **submit.sh** This takes two arguments, Entry Number and path to the report file.
  - eg: **./submit.sh 2017ANZ8353 report.pdf**
  - This will create a tar ball 2017ANZ8353.tar.gz
- Submit the generated tar ball on Moodle.