COL331, COL633, ELL405: Operating System Assignment 1: Process management and system calls [10 Marks]

Due Date: 18 Feb 2017, 23:55

Operating System

We recommend using Ubuntu for doing all the assignments. The setup part is little tricky on Windows and Mac based system. Install Instructions for Ubuntu can be found here.

Qemu

QEMU is a free and open-source hosted hypervisor that performs hardware virtualization. We will use Qemu to run the xv6. A small write up on **qemu** and **make** command can be found here.

xv6 Installation and Setup

Update:

Updated tar ball with modified check.sh to fix \r and \r\n issues.

Download xv6 setup from the following link.

This tar ball contains the original xv6 code and

assig1_1, assig1_3 and assig1_5 user programs to check the implementstion. (2 4 6 are hidden cases which will be run after the submission). Some modifications are needed to the Makefile for this. More on this can be found here

out_assig1_1,out_assig1_3, out_assig1_5 expected output for the respective user programs.

check.sh: To check the implementation.

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

test_assig1.sh: To run the user program. eg: to run the user program assig1_1, run

./test_assig1.sh assig1_1

Installtion Instructions

Commands to build and run the xv6.

- Install Qemu
  - On 32 bit systems:
  - **sudo apt-get install qemu-system-i386**
  - On 64 bit systems:
  - **sudo apt-get install qemu-system-x86_64**
- Decompress the downloaded tar ball.
  - cd xv6-os-assig1
  - To build xv6, run
  - **make**
  - To run in qemu, run
  - **make qemu-nox**

Part 1: System Call tracing [2 Marks]

Your first task is to modify the xv6 kernel to print out a line for each system call invocation. Please note no new system call has to be added for this part. Print the name of the system call and the number of times it has been called in the following format:

```
<System Call Name>  <count>
eg:
sys_fork 10
```

The system calls issued on booting the xv6 and the number of times they are called, is same all the time. Hence, your implementation should print the same sequence every time. Expected output can be seen in the file *out_assig1_1* for reference purposes. We will use some other hidden test cases, whose output will ***different*** to make sure that the implmentation is correct and not just a bunch of *printf* statements.

Part 1 (b) Add sys_toggle() system call

After you are done with the part a of the assignment, every time you boot or issue some command, you will see system trace printed on the screen, which makes it difficult to see the output of some other command. Hence, as part b, you will have to implement, a **sys_toggle()** system call, which will toggle the system trace. If it is ON, it will switch it OFF, and vice versa. By default, when xv6 boots, it should be ON.

After the system call implementation is done, you need a *user program* to actually make the system call. It can be named anything. Assuming you want

to add *user_toggle* user program to xv6, you need to create a file name *user_toggle.c,* whose contents will be like:

```
#include "types.h"
#include "user.h"
#include "date.h"

int
main(int argc, char *argv[])
{
  // If you follow the naming convetion, system call name will be sys_toggle and you
  // call it by calling
  // toggle();

  exit();
}
```

Adding a user program to xv6

- Create user_toggle.c in the xv6 directory.
- Add a line `"_user_toggle\"` to Makefile. Your Makefile should look like this

```
UPROGS=\
    _cat\
    _echo\
    _forktest\
    _grep\
    _init\
    _kill\
    _ln\
    _ls\
    _mkdir\
    _rm\
    _sh\
    _stressfs\
    _usertests\
    _wc\
    _zombie\
    _user_toggle\
```

- Also make changes to the Makefile as following:

```
EXTRA=\
    user_toggle.c\
    mkfs.c ulib.c user.h cat.c echo.c forktest.c grep.c kill.c\
    ln.c ls.c mkdir.c rm.c stressfs.c usertests.c wc.c zombie.c\
    printf.c umalloc.c\
    README dot-bochsrc *.pl toc.* runoff runoff1 runoff.list\
    .gdbinit.tmpl gdbutil\
```

- Now, enter the following commands:

```
make clean
make
```

- If you want to test the user program, launch xv6 using command
  - make qemu-nox
  - After xv6 has booted in the new terminal, you can type
  - **ls** (this will be with system traces, implemented in part a)
  - This will show the user_toggle in the list of available programs, call it using

    - **user_toggle**
    - To make sure it worked, try ls command again, which should be free of the system traces.
- You can also run user program using the test_assig1.sh script
    - eg: **./test_assig1.sh user_toggle**

Part 2: sys_add() System Call [2 Marks]

In this part you have to add a new system call to xv6. This system call should take two integer arguments and return their sum.

You should add a system call named, **sys_add()** to xv6 and then create a user-level program to test it. You can use user, program provided by us which is called **assig1_3.c** that calls your new system call.

Part 3: sys_ps() System Call [6 Marks]

In this part you have to add a new system call **sys_ps()**, to print a list of all the current running processes in the following format:

```
pid:<Process-Id> name:<Process Name>
...
...
eg:
pid:1 name:init
```

Similarly for this part, create a new user program, which should in turn call your `sys_ps()` system call, or you can use assig1_5.c provided by us.

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.

Checking the Code

There is a **check.sh** present in the code, which will compile the code and run some predefined test cases on the code. Make sure it passes the test cases before submitting. This also verifies that you have followed the naming conventions. Uncomment the following lines in **Makefile** at 177, when you complete the part 1,2 and 3 respectively.
_assig1_1\
_assig1_3\
_assig1_5\

To run the "**check"** script you need to install *expect*. Use the following command on Ubuntu:

sudo apt install expect

Note:

- 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 1. Your code will be evaluated using check script (check.sh) 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.

References

- Vim Tutorial: Click Here