EEE 3535 - Fall 2018

Project #2: An xv6 Lottery Scheduler

Objectives:

Implementing lottery scheduler into xv6 kernel

Due Dates:

Softcopy: December 12th, 10:00PM (on YSCEC)

Hardcopy: December 12th, 10:00PM (Engineering Hall III C518)

** Late submission within 24 hours automatically deducts 30%.*

Things to Submit:

- Soft copy: *Name(Korean)_StudentID.zip*

  Compress your source codes with your report

- Hard copy: *Report* only

Professor: W. W. Ro

T. A.: Jiwon Lee

Project 2 An xv6 Lottery Scheduler

1. Install Linux Operating System on your Window environment

1.1 Install QEMU on Linux

- Type ‘sudo apt-get install qemu’ to install QEMU.
- If you encounter no errors, you will see the screen printing “setting up” and “processing triggers” on the terminal.

```
Setting up qemu-system (1:2.5+dfsg-5ubuntu10.32) ...
Setting up qemu-user (1:2.5+dfsg-5ubuntu10.32) ...
Setting up qemu-utils (1:2.5+dfsg-5ubuntu10.32) ...
Setting up qemu (1:2.5+dfsg-5ubuntu10.32) ...
Setting up qemu-user-binfmt (1:2.5+dfsg-5ubuntu10.32) ...
Setting up sharutils (1:4.15.2-1ubuntu0.1) ...
Processing triggers for libc-bin (2.23-0ubuntu10) ...
Processing triggers for systemd (229-4ubuntu21.8) ...
Processing triggers for ureadahead (0.100.0-19) ...
jiwon@jiwon-VirtualBox:~/Desktop$
```

1.2 Download source codes of xv6

- Type ‘git clone git://github.com/mit-pdos/xv6-public.git’ on the terminal.
- If git has successfully downloaded the source codes, you will see xv6-public folder generated on your current directory.

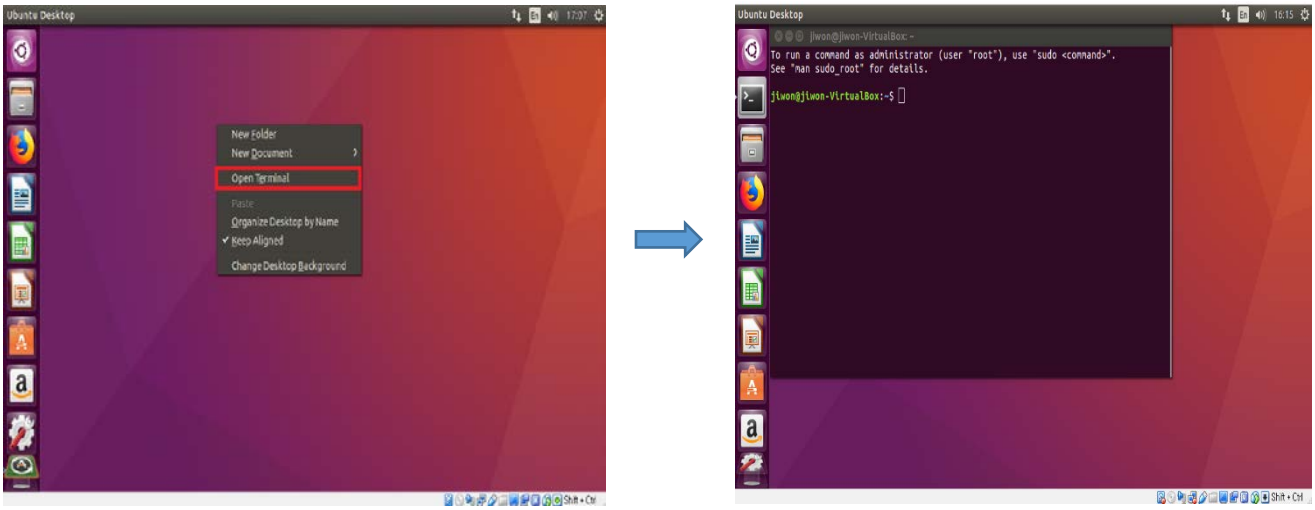
```
jiwon@jiwon-VirtualBox:~/Desktop$ git clone git://github.com/mit-pdos/xv6-p
ublic.git
Cloning into 'xv6-public'...
remote: Enumerating objects: 13974, done.
remote: Total 13974 (delta 0), reused 0 (delta 0), pack-reused 13974
Receiving objects: 100% (13974/13974), 17.14 MiB | 3.62 MiB/s, done.
Resolving deltas: 100% (9540/9540), done.
Checking connectivity... done.
jiwon@jiwon-VirtualBox:~/Desktop$ ls
hw1  proj1  xv6-public
```

- If git is not installed on your Linux, install it by using ‘sudo apt-get install git’ on the terminal type the command once again.

```
File  Edit  View  Search  Terminal  Help
jiwon@jiwon-VirtualBox:~$ git clone git://github.com/mit-pdos/xv6-public.git
The program 'git' is currently not installed. You can install it by typing:
sudo apt install git
```

2. How to Execute xv6 on QEMU with modified Source Codes

2.1 Run the terminal



2.2 Write or modify some of codes in the xv6

- Enter the following command in terminal to generate or modify a source code.

  $ gedit [Name of the Source Code].c (e.g., gedit usys.S)

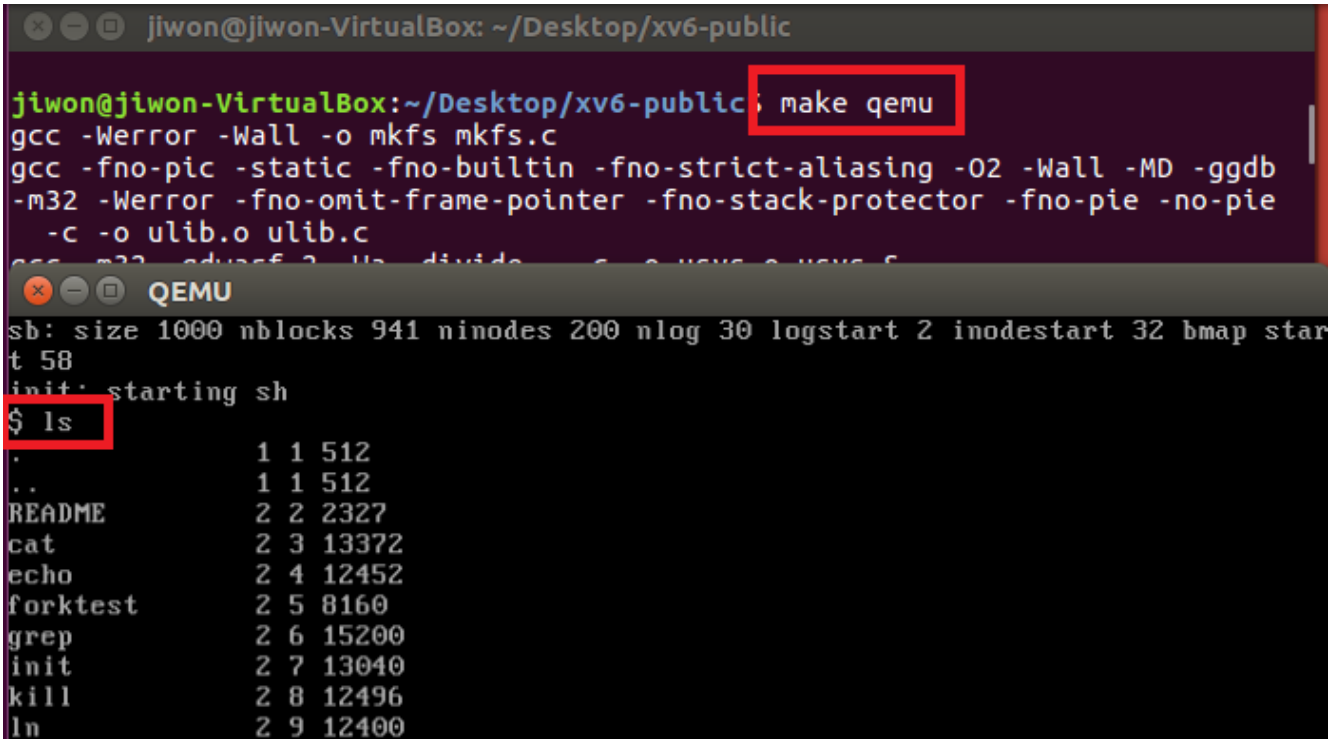
- You may use other editors such as vim, nano, emacs, etc.

2.3 Compile and execute xv6 on the QEMU

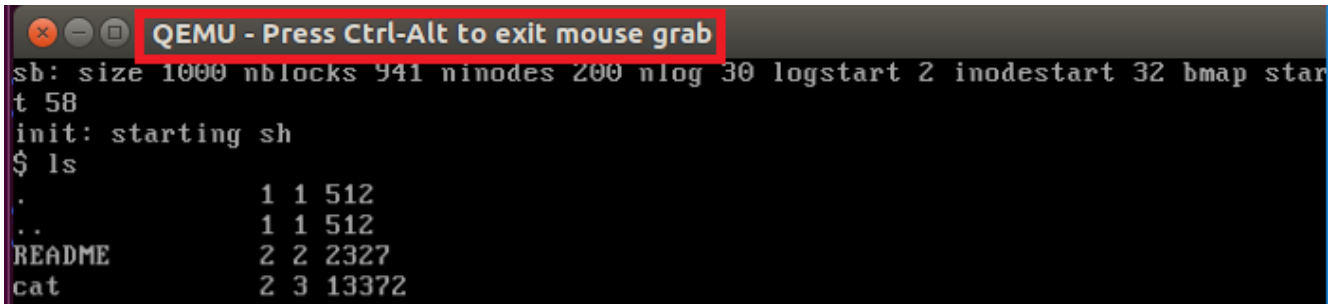
- If you have modified your source code, you will not use gcc in this time. Rather a command name ‘make’ will automatically compile your whole source code to create xv6 kernel image of your own.

- To compile and execute xv6 on the QEMU simply type ‘make qemu’. This command will automatically compile all your source codes and launch QEMU loading xv6 image.

- In QEMU, you can test existing simple programs such as cat, ls, etc. Since xv6 is a port of Unix, it works like Linux.



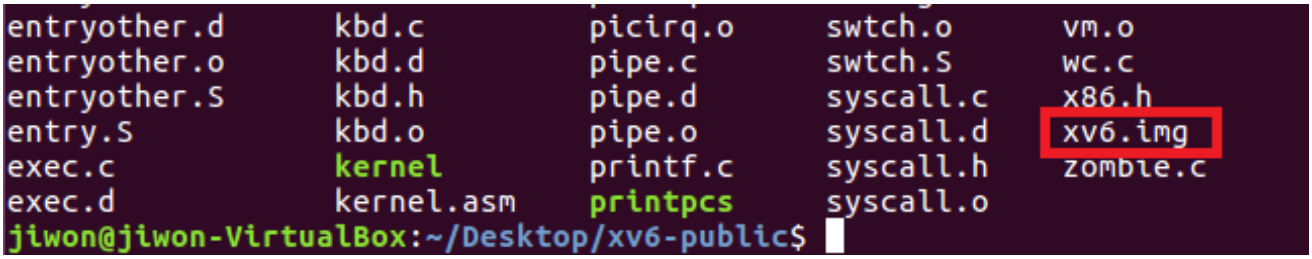
- To exit QEMU and return to your Linux, do as QEMU says on the top of the screen. Press both ctrl and alt keys to gain the control of your mouse grab.



- Press x button on the top-left of the QEMU widow to return to the Linux.

- To speed up compiling xv6 image, you may use '-j' flag to compile with multiple jobs. Hence the command will be 'make qemu -j'.

- If you enter make without other options, it will generate object files (.o files) of each source code and a xv6 kernel image file, without launching QEMU. 'make qemu' command do the same process; however, it will also automatically launch the kernel image with QEMU.



- If you modify a source code, the command ‘make’ will overwrite object files which are related to the source code, and the kernel image. If you want to delete all object files and the kernel image, use ‘make clean’ to remove those files.

3. Implementing a Lottery Scheduler

The basic idea is simple: assign each running process a slice of the processor based in proportion to the number of tickets it has; the more tickets a process has, the more it runs. Each time slice, a randomized lottery determines the winner of the lottery; that winning process is the one that runs for that time slice.

- You'll need two new system calls to implement this scheduler. The first is `int settickets(int number),` which sets the number of tickets of the calling process. By default, each process should get one ticket; calling this routine makes it such that a process can raise the number of tickets it receives, and thus receive a higher proportion of CPU cycles. This routine should return 0 if successful, and -1 otherwise (if, for example, the caller passes in a number less than one).
- The second is `int getpinfo(struct pstat *)`. This routine returns some information about all running processes, including how many times each has been chosen to run and the process ID of each. You can use this system call to build a variant of the command line program `ps`, which can then be called to see what is going on. The structure `pstat` is defined below; note, you cannot change this structure, and must use it exactly as is. This routine should return 0 if successful, and -1 otherwise (if, for example, a bad or NULL pointer is passed into the kernel). For a simpler verification, consider using `getpinfo()`.
- Most of the code for the scheduler is quite localized and can be found in proc.c; the associated header file, proc.h is also quite useful to examine. To change the scheduler, not much needs to be done; study its control flow and then try some small changes.
- You'll need to assign tickets to a process when it is created. Specfically, you'll need to make sure a child process inherits the same number of tickets as its parents. Thus, if the parent has 10 tickets, and calls fork() to create a child process, the child should also get 10 tickets.
- You'll also need to figure out how to generate random numbers in the kernel; some searching should lead you to a simple pseudo-random number generator, which you can then include in the kernel and use as appropriate.
- Finally, you'll need to understand how to fill in the structure pstat in the kernel and pass the results to user space. The structure should look like what you see here, in a file you'll have to include called pstat.h:.

```
#ifndef _PSTAT_H_
#define _PSTAT_H_

#include "param.h"

struct pstat {
  int inuse[NPROC];  // whether this slot of the process table is in use (1 or 0)
  int tickets[NPROC]; // the number of tickets this process has
  int pid[NPROC];     // the PID of each process
  int ticks[NPROC];   // the number of ticks each process has accumulated
};

#endif // _PSTAT_H_
```

- Good examples of how to pass arguments into the kernel are found in existing system calls. In particular, follow the path of read(), which will lead you to sys_read(), which will show you how to use argptr() (and related calls) to obtain a pointer that has been passed into the kernel. Note how careful the kernel is with pointers passed from user space -- they are a security threat(!), and thus must be checked very carefully before usage.
- Note that to add a new system call or a user program, you should change a file named Makefile to apply your modification to the kernel.
- After making the scheduler within the kernel, you must verify whether your scheduler is working correctly. For example, it would be easy to verify if you print tickets, random number, the number of total tickets and the result of lottery whenever the OS schedules processes. Other method can be also applied. However, notice that you must verify whether your scheduler is working without any doubt.

4. Things to Submit

A. Lottery Scheduler Source codes – soft copy only.

- A compressed zip file containing entire xv6 project directory (xv6-public in default) of your own work with your report file.

B. Report – both softcopy & hardcopy.

- Cover page with your name and Student ID.
- Explanations of your implementation strategy.
- Verification of your own scheduler.
- Screenshots of your successful implementation to the xv6 kernel.
- Explanations on your modified source codes in the whole procedure.
- Write discussions about your lottery scheduler.

5. Information

- Notice that printf() function is slightly different. In xv6, the first argument of the printf() will be an integer and the rest of the inputs are identical to typical printf(). In a nutshell, printf() in the xv6 source code will be used as printf(1, “Hello world #%d\n”, number). If the first argument is 2, it will notice an error has occurred to the kernel.

- There are some source codes for utility programs such as cat, grep, ls, etc. You don’t have to change these codes since you will modify the kernel. Still, you can write your own utility programs to execute on the xv6 kernel.

- Figure out which codes should be modified throughout the project. There are kernel codes which are related to implementing a lottery scheduler while some are not.

- The full score of the project is 100 points. You should work alone. We recommend you read the following pages of Operating Systems: Three Easy Pieces project as well. Some ambiguous or uncertain explanation on this manual might be supplemented.

  - ✓ https://github.com/remzi-arpacidusseau/ostep-projects/tree/master/initial-xv6
  - ✓ https://github.com/remzi-arpacidusseau/ostep-projects/blob/master/initial-xv6/background.md
  - ✓ https://github.com/remzi-arpacidusseau/ostep-projects/tree/master/scheduling-xv6-lottery