Programming Assignment #3

Lottery Scheduler

2024. 11. 22

Overview

- Due data: 12/6
    - No late submission is allowed

- Objective

    - To gain further knowledge of a real kernel
    - To familiarize yourself with a scheduler
    - To change that scheduler to a new algorithm

Phase #1: Intro

- **Objective**
  - Your system call, `getreadcount()`, simply returns how many times that the `read()` system call has been called by user processes since the time that the kernel was booted.

Programming Enviroment

- **Install packages**

```
prompt> sudo apt update
prompt> sudo apt install gcc make git qemu-system-x86 wget build-essential gdb
```

- **Download script for testing the code**

```
prompt> git clone https://github.com/Song-HyeonJin/HW3_xv6-test.git
prompt> cd HW3_xv6-test/initial-xv6
prompt> git clone https://github.com/Song-HyeonJin/xv6-public.git
```

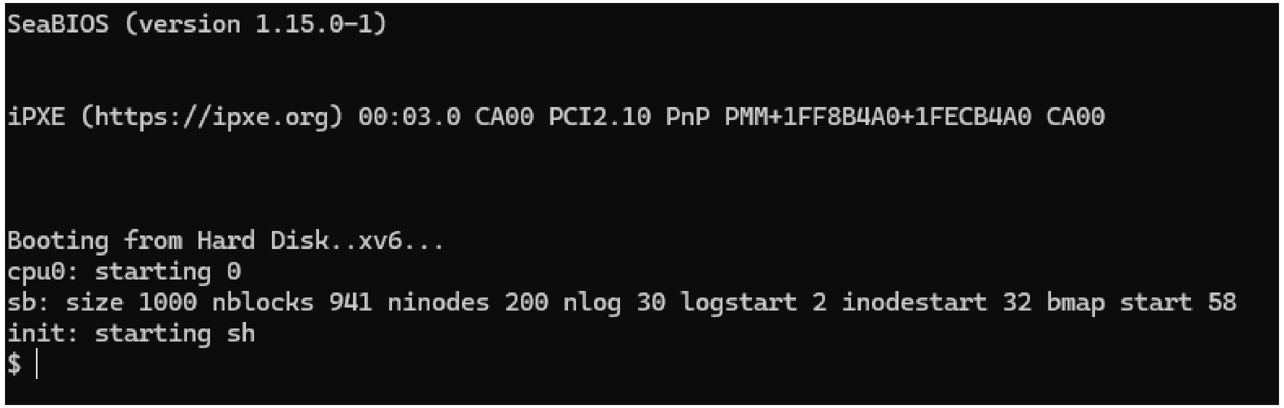
- **You will do programming at “xv6-pubic” directory**

Programming Enviroment

- **Test your code and submit the results**

prompt> make qemu-nox

If all has worked well, you’ll see this screen



```
SeaBIOS (version 1.15.0-1)


iPXE (https://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1FF8B4A0+1FECB4A0 CA00



Booting from Hard Disk..xv6...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ 
```

Ctrl+a and press x to quit

prompt> cd …/HW3_xv6-test/initial-xv6
prompt> ./test-getreadcount.sh

Phase #2: Scheduling-xv6-lottery

- **Objective**
  - Xv6 kernel does scheduling processes with round-robin
  - You should reimplement lottery scheduler in xv6 kernel

- **Implementations**
  - New system calls
    - Implementation int settickets(int number)
    - Implementation int getpinfo(struct pstat*)
  - Handle child process state
    - Modify fork() system call
  - Lottery scheduler
    - Reimplement scheduler() to a lottery scheduler algorithm

- **You will do programming at “xv6-pubic” directory**

Programming Enviroment

- Background
  - Ostep-projects
    https://github.com/remzi-arpacidusseau/ostep-projects/tree/master/scheduling-xv6-lottery
  - xv6 book
    https://www.cs.virginia.edu/~cr4bd/4414/F2018/files/xv6book.pdf
  - Lottery Scheduling
    http://pages.cs.wisc.edu/~remzi/OSTEP/cpu-sched-lottery.pdf

- Follow the example of similar system calls (kill, fork, …)
  - sysproc.c
  - syscall.c
  - syscall.h
  - usys.S
  - user.h
  - proc.c
  - defs.h

- You should pass arguments into kernel with argptr(), argint()
  - Good examples of how to pass arguments into the kernel are found in existing system calls.
    Ex) sys_read() in sysfile.c

Program scheduler

- **`int settickets(int number)`**
  - Set the number of tickets of the calling process
  - This routine should return 0 if successful and -1 otherwise

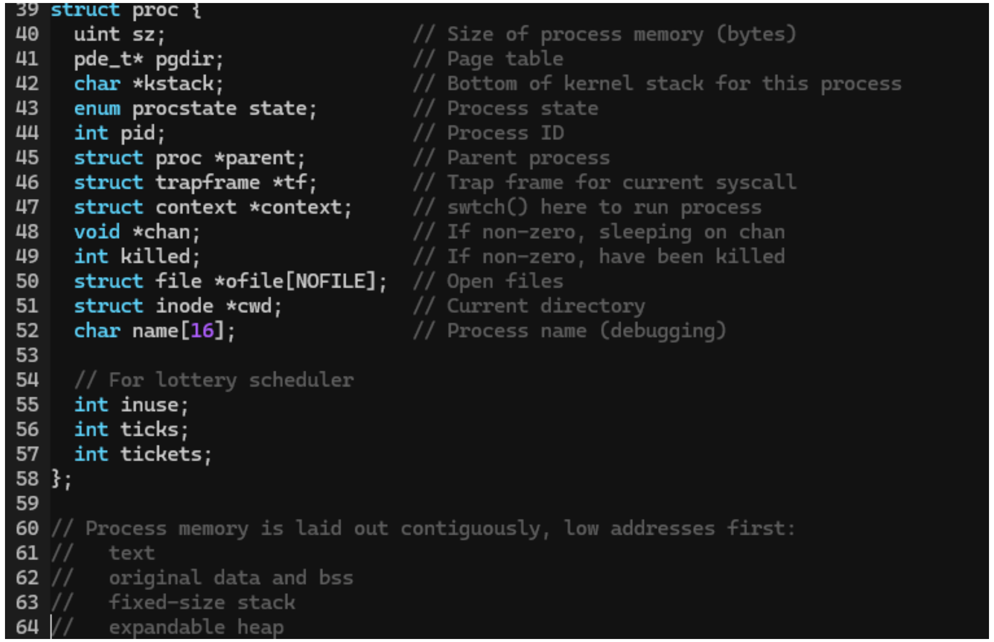
    Ex) if the caller passes in a number less than one, return -1.

- **`int getpinfo(struct pstat *)`**
  - Return some information about all running processes (pstat)
  - This routine should return 0 if successful and -1 otherwise

    Ex) if a bad or NULL pointer is passed into the kernel, return -1

Program scheduler

- `Proc.h`
  - Xv6 kernel stores “process state” in struct proc
  - Each process has proc struct
  - You’ll need to introduce some new variables in proc to implement lottery scheduler



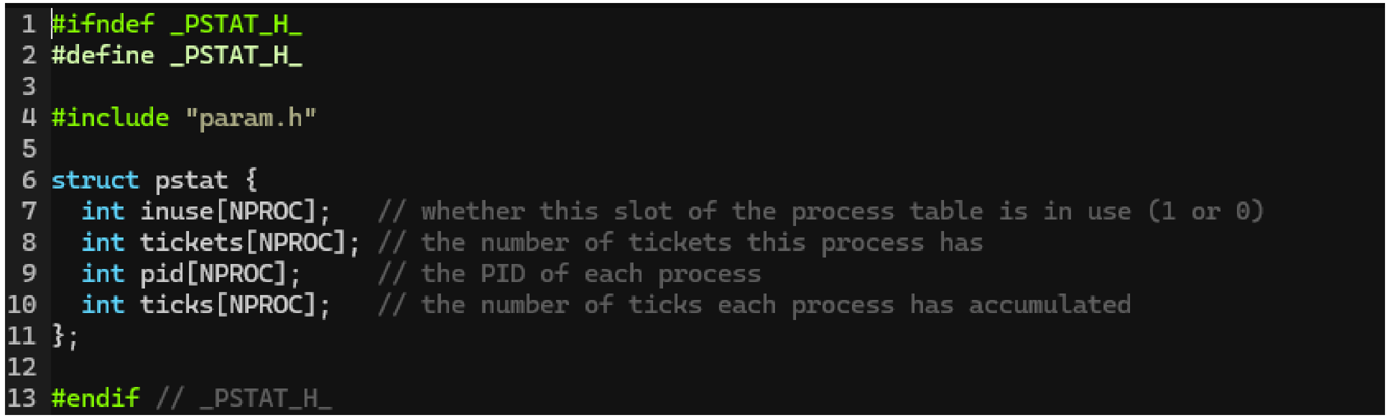
```
struct proc {
  uint sz;                     // Size of process memory (bytes)
  pde_t* pgdir;                // Page table
  char *kstack;                // Bottom of kernel stack for this process
  enum procstate state;        // Process state
  int pid;                     // Process ID
  struct proc *parent;         // Parent process
  struct trapframe *tf;        // Trap frame for current syscall
  struct context *context;     // swtch() here to run process
  void *chan;                  // If non-zero, sleeping on chan
  int killed;                  // If non-zero, have been killed
  struct file *ofile[NOFILE];  // Open files
  struct inode *cwd;           // Current directory
  char name[16];               // Process name (debugging)

  // For lottery scheduler
  int inuse;
  int ticks;
  int tickets;
};

// Process memory is laid out contiguously, low addresses first:
//   text
//   original data and bss
//   fixed-size stack
//   expandable heap
```

Program scheduler

- **`pstat.h`**
  - You may need struct `pstat` for `int getpinfo(struct pstat*)`
  - If getpinfo(struct pstat *p) is invoked, you should store state of processes to pstat pointed by p



```
#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_
```

Tickets of Child Process

- **You’ll need to make sure a child process inherits the same number of tickets as its parents**
- Example:
  - If the parent has 10 tickets and calls fork() to create a child process
  - The child should also get 10 tickets

- **You’ll need to make sure a child tick is initialized**
- Example
  - If the parent has 100 ticks, and calls fork() to create a child process
  - The child's ticks should be initialized to 0

- **Modify fork() in `proc.c`**

Lottery Scheduler

- **Most of the code for the scheduler can be found in `proc.c`**

- **Modify the `scheduler()` in `proc.c`**
  - **Winner**: a random number between [0, total # of tickets]
  - Choose a process to run based on tickets

- **Use `rand()`, `srand()` in `proc.c`**

Test Lottery Scheduler

- Use `xv6-public/test_lottery.c`
- test_lottery.c:
  - Create three processes (using fork(), settickets())
  - Process A: 30 tickets
  - Process B: 20 tickets
  - Process C: 10 tickets

- You must implement "tickets of child process" for the test
  - `int setticket(int number)`
  - `int getpinfo`

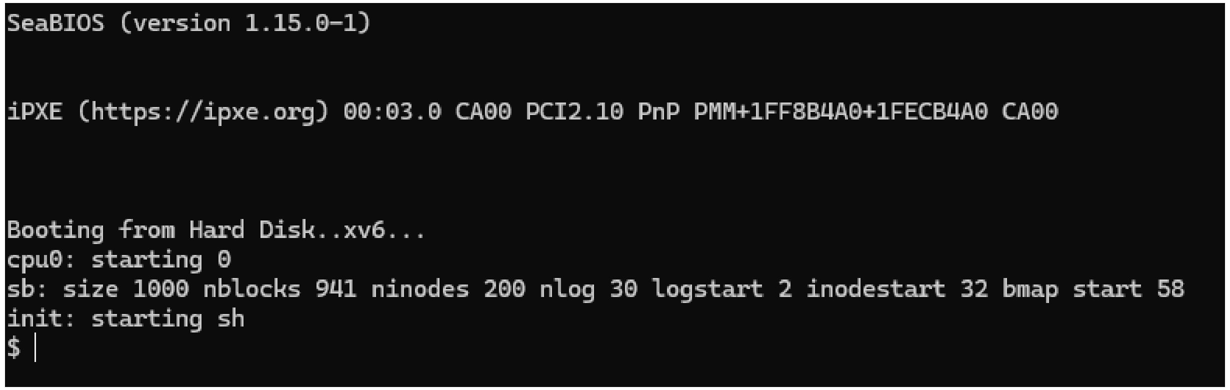
- Show the number of ticks a set of three processes
  - Get `pstat.tick[NPROC]` using `getpinfo()`

Test Program

- You can implement your own test program
- Implement test program in xv6-public/ directory
  - ex) xv6-public/test.c

- Add make target for test program to Makefile

- Build and run xv6 kernel with ‘make qemu-nox’



```
SeaBIOS (version 1.15.0-1)


iPXE (https://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1FF8B4A0+1FECB4A0 CA00



Booting from Hard Disk..xv6...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ 
```

- ls : You can see your test program
- test: You can run your test program

Conclusion

- **Within Due: 12/6**
  - No late submission is allowed 😢
- **Take it step by step** 🙂