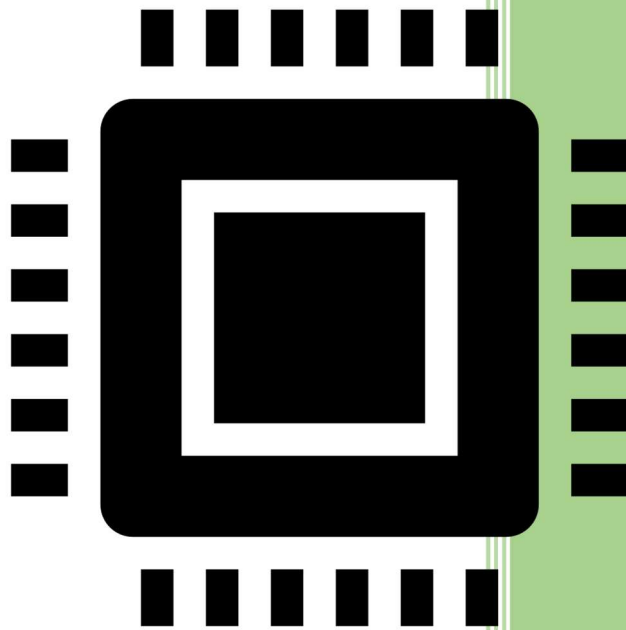


2019

XV6 Lottery Scheduler



Jaycie Raby and Srinivas Simhan
University of Michigan-Dearborn
CIS 450

Contents

Explanation of Implementation of Scheduling Policy	2
Files Modified During Implementing of New Scheduler.....	3
Detailed Report of Testing Results	5
Reflection of Project Completion.....	6

Explanation of Implementation of Scheduling Policy

In order to implement the scheduling algorithm in `proc.c`, few adjustments had to be made, but are as follows:

Note: All the below functionalities were added to the already created `proc.c` and weaved between already provided code from Professor Guo!

1. Create the seed for `srand()` to be implemented
 - a. `struct rtcdate r;`
 - b. `cmostime(&r);`
 - c. `srand(r.second);`
2. Initialize an integer value to hold number of tickets
 - a. `Int totalTickets = 0;`
 - b. Call the `rand()` function to create random number
 - c. `Unsigned int random_num = rand()%(totalTickets+1);`
3. Create a for loop to cycle through the struct `ptable` to add total number of tickets
 - a. `for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){`
 - b. `if(p->state != RUNNABLE)`
 - c. `continue;`
 - d. `totalTickets += p->tickets;`
 - e. `}`
4. Create a second for loop to cycle through struct `ptable` to choose the lottery winner
 - a. `for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){`
 - b. `if(p->state != RUNNABLE)`
 - c. `continue;`
 - d. `}`
5. Call `random_num` again to get a new random number and check If random number is greater than or equal to number of tickets? If yes, continue through rest of the scheduler process

Files Modified During Implementing of New Scheduler

Files edited during this project are as follows:

1. Makefile
 - a. Added the lotteryTest.c and ps.c into
2. proc.c
 - a. Added header files pstat.h, random.h, implemented settickets, getpinfo and added scheduler functionalities as described above in “Explanation of Implementation of Scheduling Policy”.
 - b. Settickets implementation
 - i. Make a pointer to struct proc and myproc()
 - ii. Set an acquire lock on ptable
 - iii. Return 1 if tickets are greater than 10000
 - iv. Else, loop through ptable to get current process ids and tickets
 - v. Release lock on ptable
 - vi. Return 0
 - c. Getpinfo implementation
 - i. Make a pointer to struct proc
 - ii. Set acquire lock on ptable
 - iii. Create pointer to number of processes in struct pstat and set to zero
 - iv. Loop through ptable, and if a process state is unused, get the id, ticks and tickets and increment the number of processes.
 - v. Release lock on ptable
 - vi. Return 0
3. proc.h
 - a. //set default tickets
 - b. #define DEFAULT_TICKETS 10
 - c. Add tickets and ticks integer to the struct proc
4. pstat.h
 - a. add extern int for struct pstat and tickets so that getpinfo and settickets functions can access this information in proc.c
5. syscall.c
 - a. extern int sys_settickets(void);
 - b. extern int sys_getpinfo(void);
 - c. extern int sys_yield(void);
 - d. [SYS_settickets] sys_settickets,
 - e. [SYS_getpinfo] sys_getpinfo,
 - f. [SYS_yield] sys_yield,
6. syscall.h

- a. #define SYS_settickets 22
- b. #define SYS_getpinfo 23
- c. #define SYS_yield 24

7. sysproc.c

- a. #include "pstat.h"
- b. extern int settickets(int);
- c. extern int getpinfo(struct pstat*);
- d.
- e. int
- f. sys_settickets(void){
- g. int tickets;
- h. if (argint(0, &tickets) < 0) return -1;
- i. return settickets(tickets);
- j. }
- k.
- l. int
- m. sys_getpinfo(void){
- n. struct pstat* p;
- o. if (argptr(0,(void*)&p,sizeof(struct pstat*)) < 0) return -1;
- p. return getpinfo(p);
- q.
- r. }
- s.
- t. int
- u. sys_yield(void){
- v. yield();
- w. return 0;
- x.
- y. }

8. user.h

- a. Struct pstat;
- b. //system calls
- c. int settickets(int);
- d. int getpinfo(struct pstat*);
- e. int yield(void);

9. usys.S

- a. SYSCALL(settickets)
- b. SYSCALL(getpinfo)

c. SYSCALL(yield)

Detailed Report of Testing Results

LotteryTest.c

For each test, as provided in documentation by Professor Guo, we input an amount of time to run for {test1 = 1000, test2 = 2000, test3 = 2000} and lists of numbers of tickets to assign for each of the subprocesses {test1 = [500,500,100,50,100], test2 = [200,200,200,200,100,100,2], test3 = [10,20,50,100,200,300]}. The output of each of the tests indicates that number of tickets assigned to each subprocess and the number of tickets it ran; for example, test 1 indicated 500 tickets were assigned, and 755 ticks were ran. Images of each test are included below.

1. Test 1

```
$ lotteryTest 1000 500 400 100 50 10
TICKETS TICKS
500 755
400 752
100 286
50 127
10 32
```

2. Test 2

```
$ lotteryTest 2000 200 200 200 200 100 100 200
TICKETS TICKS
200 640
200 658
200 673
200 664
100 369
100 346
200 611
```

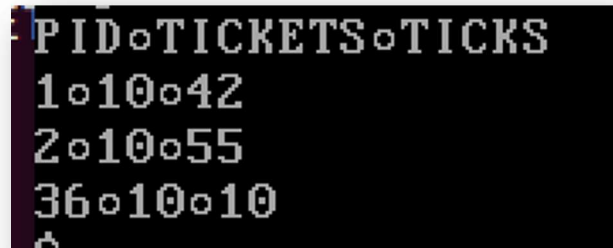
3. Test 3

```
$ lotteryTest 2000 10 20 50 100 200 300
TICKETS TICKS
10 75
20 157
50 398
100 704
200 1153
300 1423
```

PS.c

For the ps.c test, the call to ps.c runs getpinfo which returns all of the information from the struct including the process id, max number of tickets, and ticks. A screenshot of the output is included below.

1. Test 1



A screenshot of a terminal window showing the output of the ps.c program. The output is a table with three columns: PID, TICKETS, and TICKS. The first three rows of data are visible.

PID	TICKETS	TICKS
1	10	42
2	10	55
36	10	10

Reflection of Project Completion

1. Project 4 was extremely challenging mentally on my partner and I, as our knowledge of xv6 and Linux based systems was slim to none. We faced difficulties with getting our virtual machine to run after Oracle ran an update on their software and caused osprojects VM to stop working entirely. After around 8 hours of time placed around just getting the VM to open, we reverted to an old build (VirtualBox version 5), the VM began working again. Once we passed this hurdle, implementing the three system calls was easy as we just followed the same steps we utilized from Project 1. However, implementing the scheduler was extremely difficult for us; specifically, srand() and rand(). The compiler kept giving us problems about srand() and rand() being used but never called and complaining about pointers. After about another 10 hours of troubleshooting this issue, we realized we had included "random.h" in the sysproc.c file, which is why the compiler was saying "these functions are defined but never initialized"; meaning we had been looking at the wrong program to correct this problem. After about 25+ hours of invested time in project 4, we feel much more comfortable with the inner workings of the xv6 processor and have gained a lot of troubleshooting skills with VirtualBox, virtual machines, xv6 and CPU scheduling. Though this project was stressful and time consuming, but the troubleshooting and problem-solving skills will help us along the rest of our careers as computer scientists.