Operating system Project 4

Xv6 Lottery CPU scheduler

Team Members: Marco Seman

Prof. Guo

In this project, we are to implement a simple version of lottery CPU scheduler and three additional files were given for testing.To implement this scheduler and make it executable, we need to make three syscall, the int settickets(int number) which sets the number of tickets of the calling process, by default, this process have only 10 tickets, the syscall will return -1 if the number of tickets exceed 100,000. The second syscall is the int getpindo(struct pstat\*), this routine will returns the information about all running processes, includes how many time each process have to chosen to run and the process ID of the process,, this process will return -1 if a bad or NULL pointer is passed into the kernel. The last syscall is the yield(), this is a wrapper system call, which it simply calls the real implementation yield() function and will return 0. Changes and modification had been made to the following files:

Syscall.c

Syscall.h

Defs.h

Proc.h

Proc.c

Usy.S

User.h

Makefile

Sysproc.c

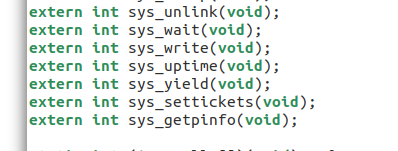
Sysproc.h

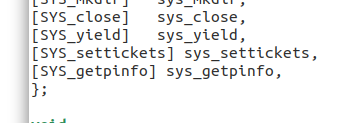
A header file pstat.h also added to the xv6 files, which contains the total number of non-UNUSED processes, num\_processes, the number of PID[NPROC], the number of tickets[NPROC] and the number of ticks[NPROC]

Source Added and Changed:

Random.h, pstat.h, lotteryTest.c, ps.c have been included in the xv6 file which were given by the professor. No changes to these files were made.

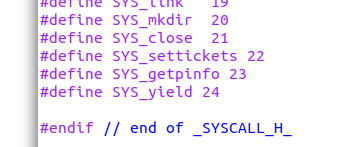
Syscall.c:





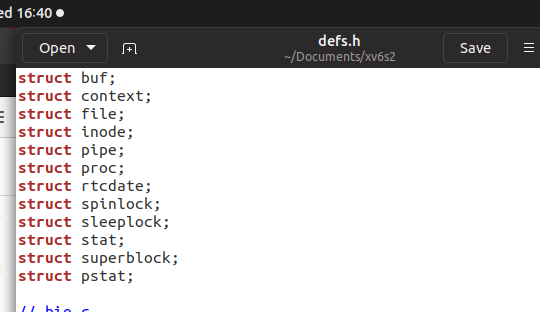
The syscalls for yield, settickets, getpinfo have been added. This file externally defines the function that make the connection between the shell and the kernel.

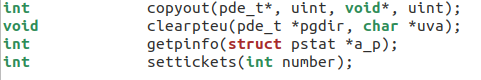
Syscall.h:



System calls for settickets, getpinfo, and yield have been added to call vector.

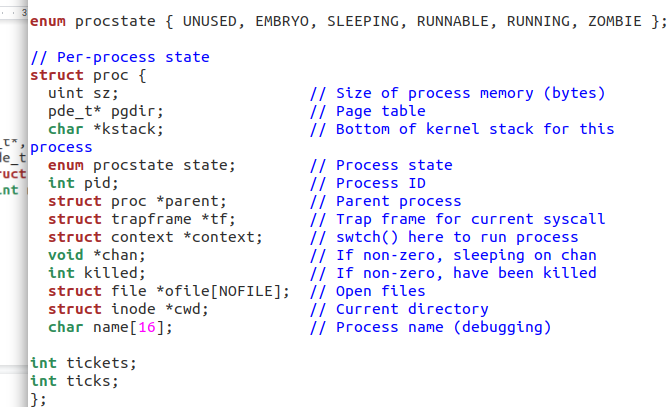
Defs.h





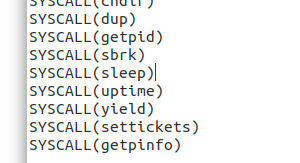
Struct pstat, getpinfo(struct pstat \*a\_p), and settickets(int number) have been added so the methods can be used in the real implementation in proc.c and sysproc.c

Proc.h:



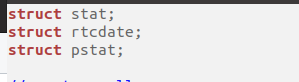
In proc.h, we only added tickets and ticks.

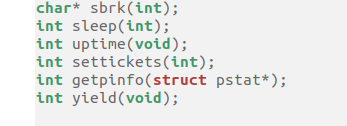
usys.S:



Syscalls for yield, settickets, and getpinfo have been defined to connect the call of user to the system call function.

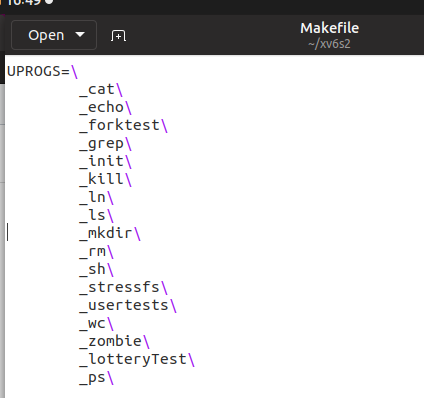
User.h:





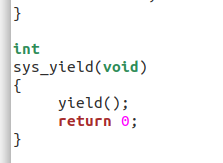
Struct pstat, int settickets, getpinfo, and yield have been defined so they can be called through the shell.

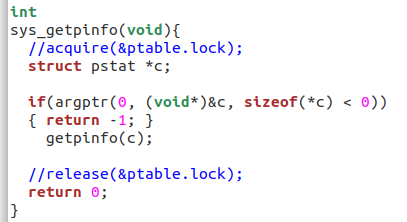
Makefile:

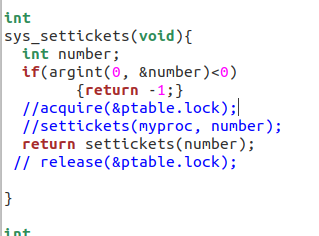


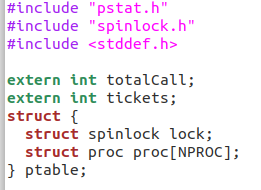
lotteryTest\ and ps\ have been added under UPROGS so that it can be run with qemu.

Sysproc.c



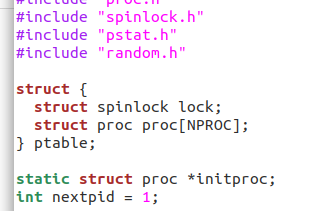


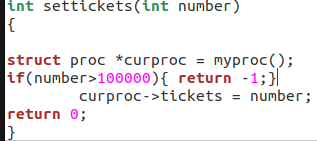


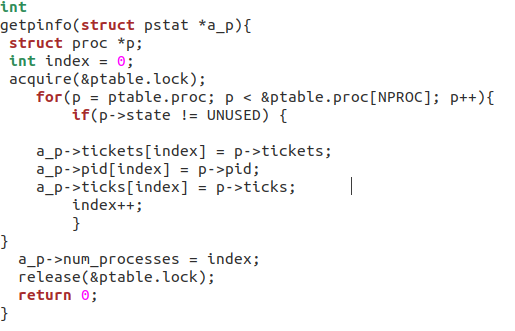


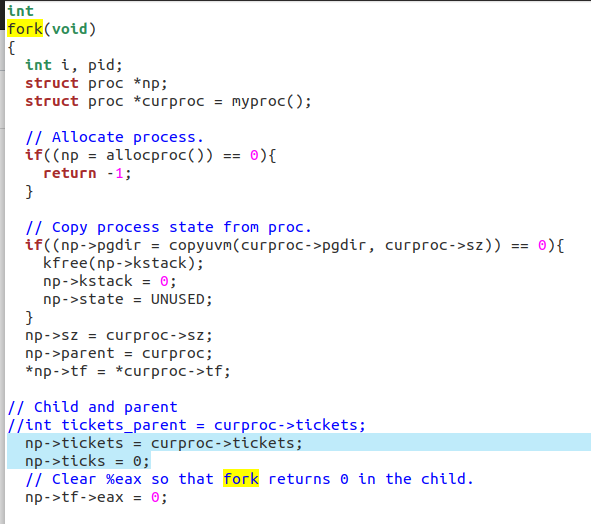
Sys\_proc is where the real implementation to our method is called. We added a function for gettickets and getpinfo. Here we change the number of tickets of the process. We return 0 if it is successful and -1 when there is an error.

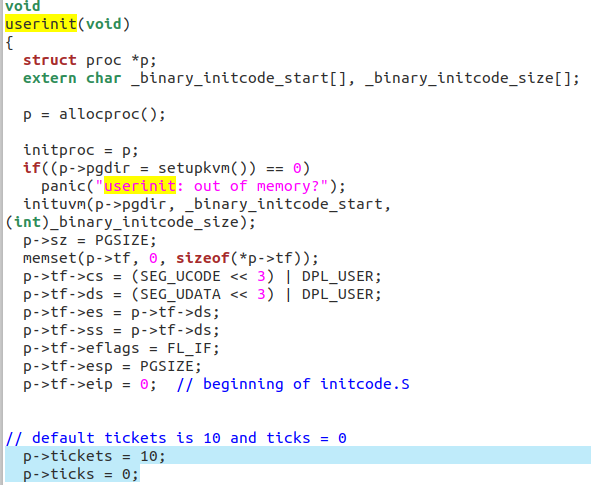
Proc.c:

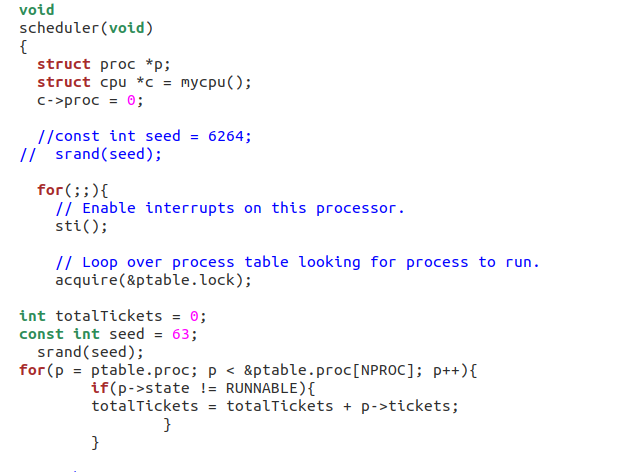


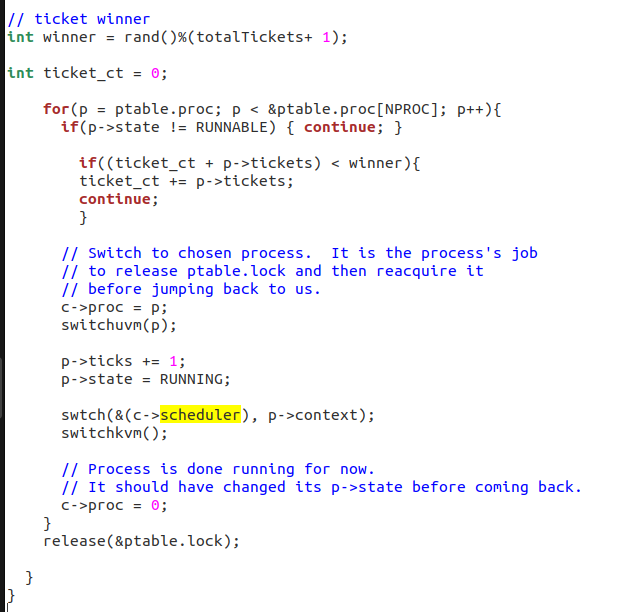






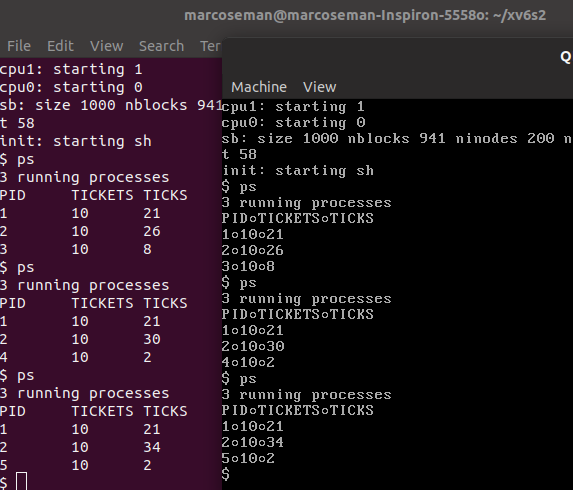


In the proc.c scheduler, we added another for loop that keep track of the total amount of ticket by the RUNNABLE state of the process. Then we use the rand()%(total + 1) to randamize the winner.

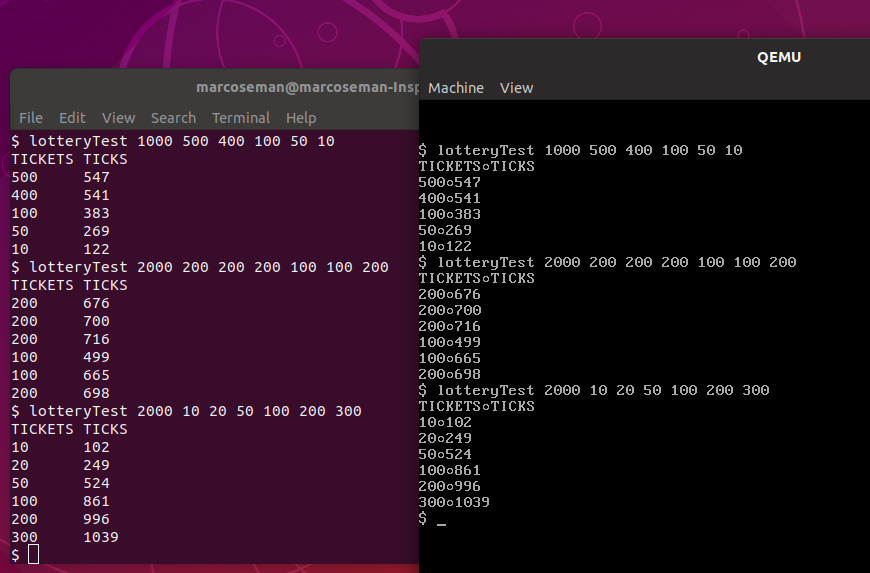
In the second for loop, we implement a if statement that check if the ticket is less than the winner ticket, if it's true, then it will add the ticket to the ticket count.

In proc.c, we added settickets, getpinfo, yield, and made modifications to the fork, userinit, and scheduler. For the yield, we just called the function then returned zero. For userinit, we set ticks to zero and tickets to 10. In fork, we added a function that accesses the member of the ticket class. In getpinfo, we obtain access to the ptable and gather information about pid, tickets, and ticks then pass the results to ps.c. In settickets, we take the number of ticks and make sure it is between 0 and 100000 and will return -1 if it fails.

Ps results:



The number of tickets are set to 10 and pid will always begin with 1 as a default. Ticks are determined by the number of time that each process has run. Pid represents the process number. Pid is created by calling the fork.



Most of the implementation for the lotteryTest is in the scheduler. The first number entered indicates the total number of tickets. We needed to implement a statement that checks if the tickets are less than the winning ticket, if it is true, then we add the ticket to the ticket count. The higher the number, the closer the ticket is to being a winner.

What we learned:

In this lab, we learned about how the different state work on the processes. We also learned about how the scheduler schedules the new process when it iterating through all processes in the kernel. Most of the code must be implemented inside the proc.c file whereas the sysproc.c will call the function that are in the proc.c file. However, we learned prior to the implementations of the functions, we must also include system calls and definitions to the functions in multiple other system call files. Xv6 must know that the function exists and we needed to make sure that the program the shell can communicate properly and display our results implementations inside of proc, otherwise we would receive multiple errors.