**Homework 3 – Lottery Scheduler**

**Task 1: Implement Nice System Call**

Updates made in the following system files

A foo.c file is created to create child process and test the nice code. The foo.c program takes the number of child processes to be created from the user through the command line. A simple check is done to see if the user has correctly given the input, if the number given is a negative number, then terminate the process and such.

Here, we use fork system call to create the required number of processes. To make the child process a CPU bound process, a calculation part is included in the code so that CPU utilization is forced to occur.

Now, we have to include the priority of the process in the process control block which is stored in **proc.h** under struct proc

Int priority;

In **proc.c** we have to make appropriate changes under allocproc function. CPU is allocated the process using this function. Here we set the default priority of any process that comes into memory as 20.

Next, we change the priority of the child process when it is created such that the priority of the child process is greater than the parent process. Thus we make these changes in **exec.c** where we change

proc->priority = 15

Now, to implement nice system call, we first need to update the table that is stored in system call interface in **syscall.h**. Here I have defined

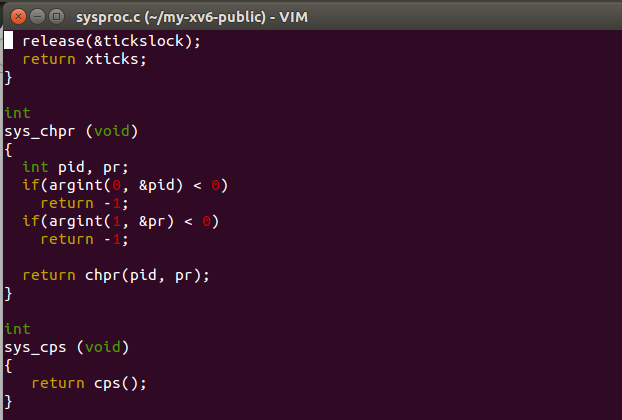
SYS\_chpr = 22

Now we need to include the declaration of chpr in **defs.h and user.h** as

Int chpr(int pid, int priority)

Include the definition of the system call in **proc.c** . We write function definition of chpr in this file.

After this, we need to include the definition of syschpr in **sysproc.c** which will then call the chpr function.



Then we need to call the system call in **usys.S**, I added

SYSCALL (chpr)

Now, we open **syscall.c** and declare the function we defined in sysproc.c i.e

extern int sys\_chpr(void)

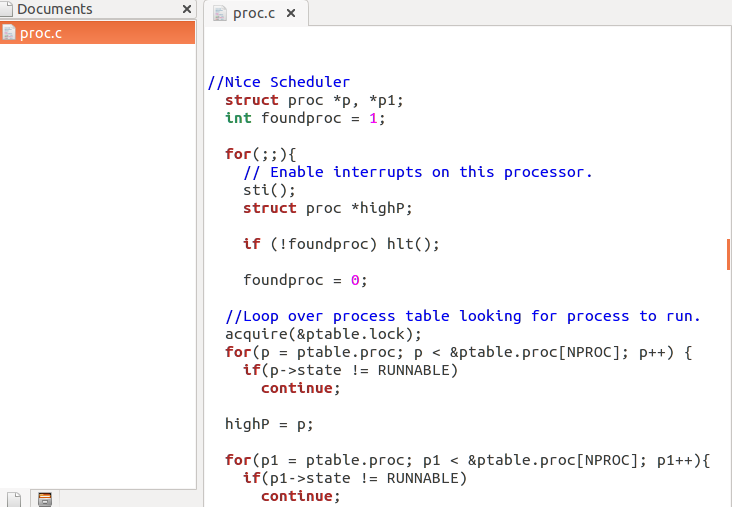
[SYS\_chpr] sys\_chpr,

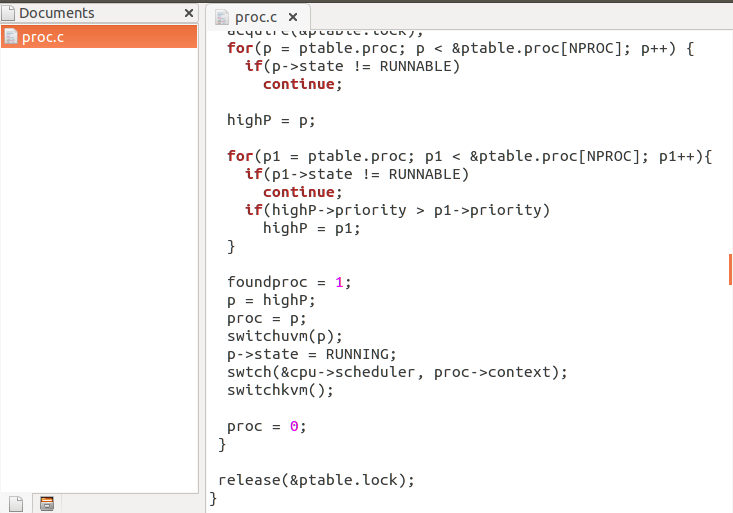
Now, we create a new file nice.c which calls the chpr function. Here we take pid and priority from the command line and check if the user has given correct values for the same.

The scheduler function has been included in **proc.c**

Thus, this changes the scheduling algorithm of xv6 from round robin to priority scheduling.

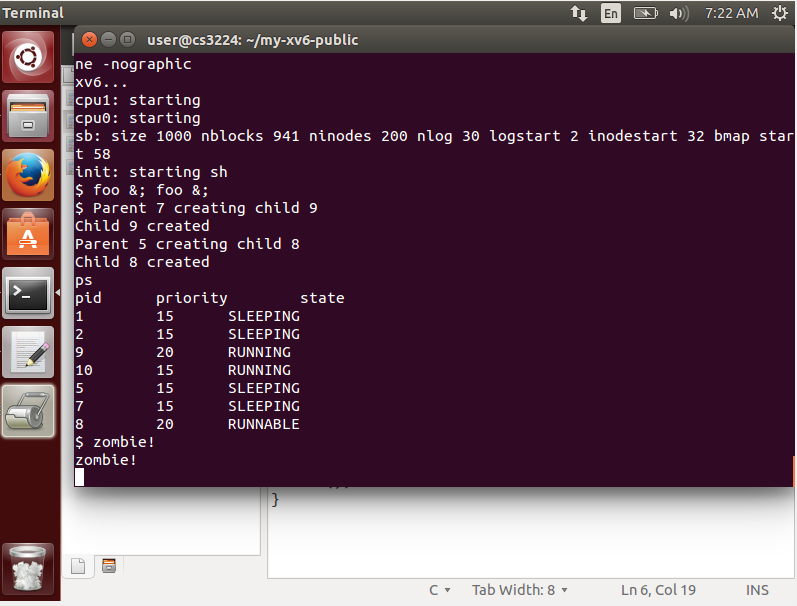
Scheduler function for Nice:





Testing Nice:

We run foo a couple of times where we fork child processes and we simultaneously run ps function to see the priority values and the states of the different processes.

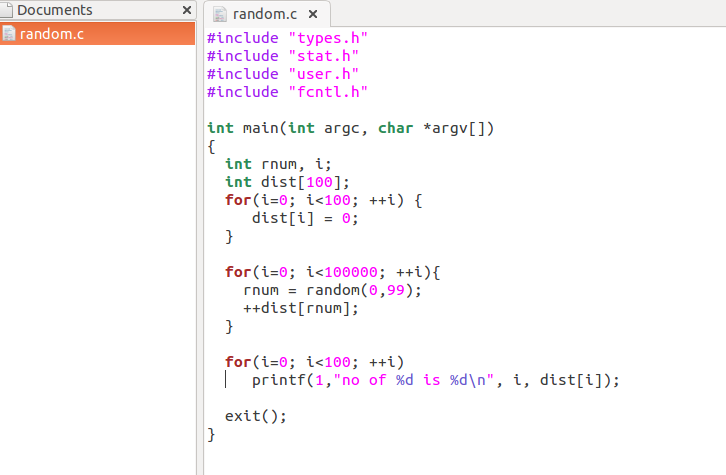


**Task 2: Random Number Generator**

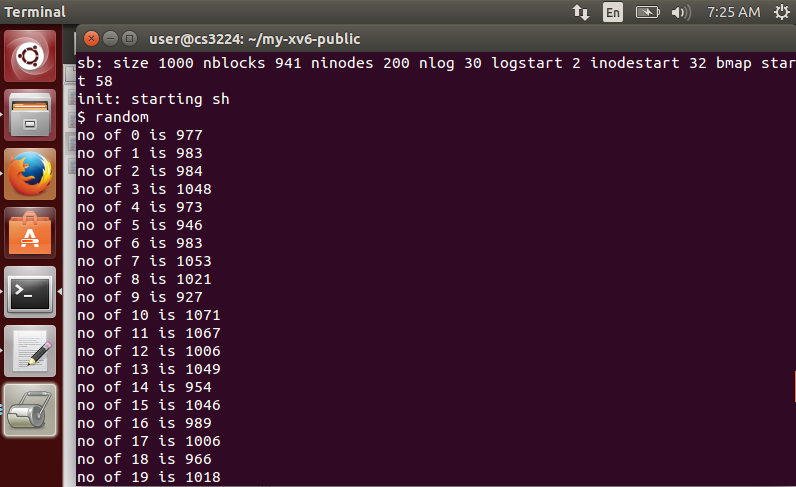
A ‘random’ system call is written in proc.c

A 32 bit LSFR generator is used to randomly generate numbers. A range is defined and then using this range a seed is generated.

A simple test case is written to randomly generate the numbers between 0-99 and observe the distribution.



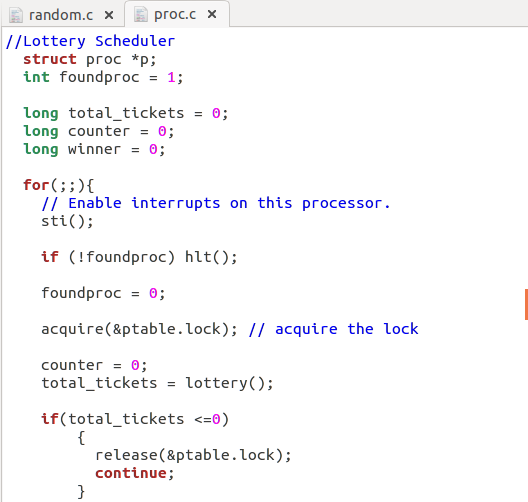
Output:

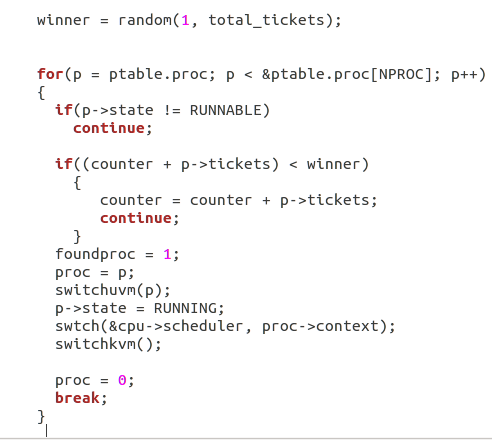


This can be called by ‘random’ in command prompt.

**Task 3: Implementing lottery scheduler**

The following scheduler function has been added to implement the lottery scheduler in proc.c



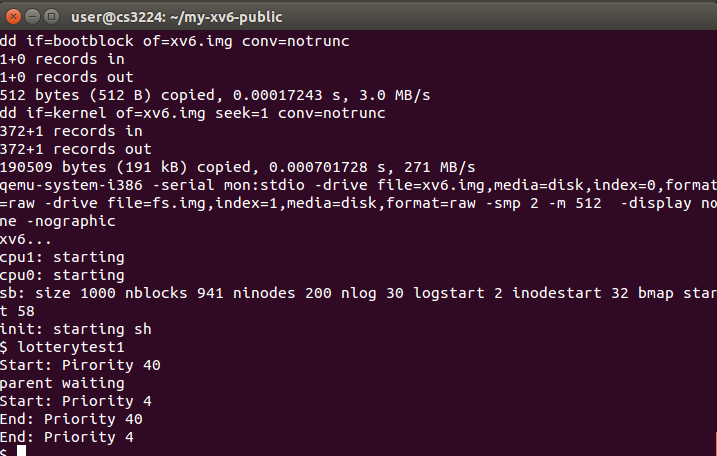




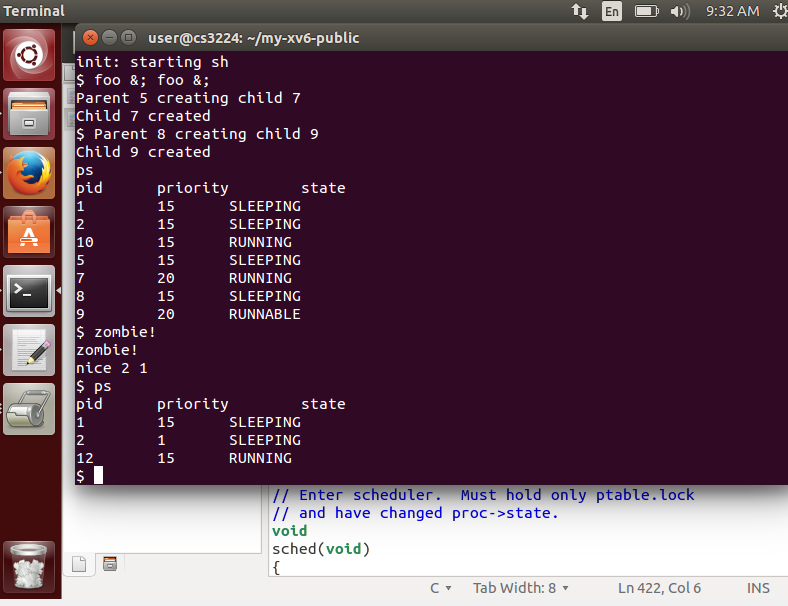
* A function is defined to run through the process table to check the runnable processes and thus add the number of tickets accordingly
* A list is maintained to map the priority values to the number of tickets. More tickets meaning higher the priority of the process
* After counting the total number of tickets, if it is 0, lock is released, and loop runs again to find a runnable process
* Post this step, a random winner is generated, and corresponding process is identified
* The process table is the data structure used to iterate through runnable process. The leaving and entering of the processes is also maintained by the process table along with the state information.
* A sum of all the runnable process is maintained to match the winning number back to the runnable process

Testing the scheduler:

A simple test case is written to check the functioning. Even though lottery scheduler is random, it only logically makes sense that process with higher priority will get more tickets and thus in turn more CPU time to complete processing. Lotterytest1 is run for testing.

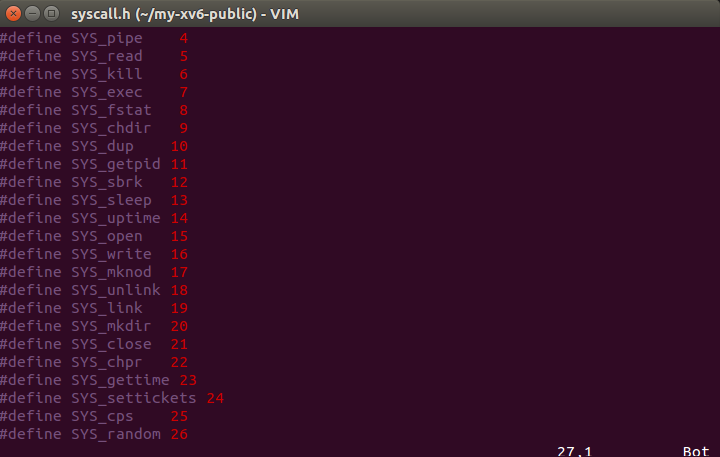


Another way to test would be to use foo test file to fork child processes and then try to change the priority of the processes using the nice function



Changes made for system calls in the following files:

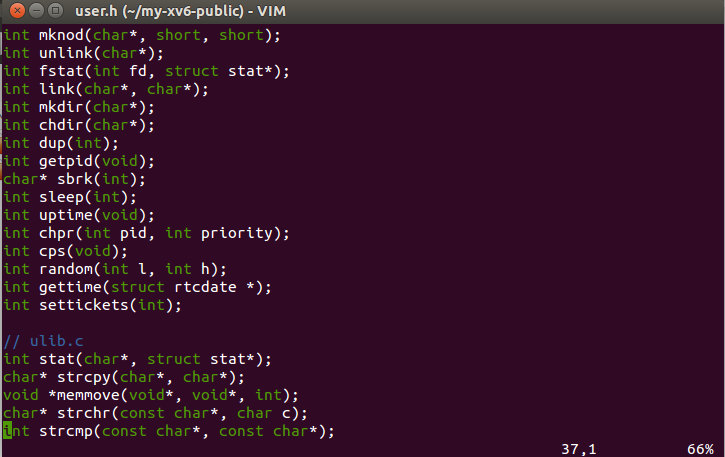
**syscall.h**



defs.h



user.h



usys.S

