**Akshay Mall**

CS 5A

20184037

OS Lab Week 8

**Lottery Scheduling**

Lottery scheduling is a probabilistic scheduling algorithm for processes in an operating system.

Processes are each assigned some number of lottery tickets, and the scheduler draws a random

ticket to select the next process. The distribution of tickets need not be uniform; granting a process

more tickets provides it a relatively higher chance of selection. This technique can be used to approximate other scheduling algorithms, such as Shortest job next and Fairshare scheduling.

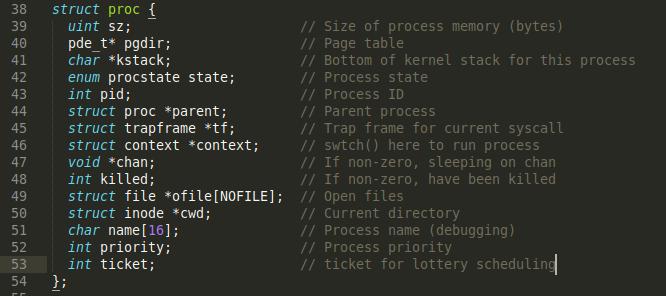
Lottery scheduling solves the problem of starvation. Giving each process at least one lottery ticket

guarantees that it has a nonzero probability of being selected at each scheduling operation.

Modifitcations needed to be done to xv6

Each struct proc has an additional field, tickets , that tracks how many tickets it has.

Akshay Mall 1



When the scheduler runs, it picks a random number between 0 and the total number of tickets. It then uses the algorithm described in class to loop over runnable processes and pick the one with the winning ticket.



int lottery\_Total(void){

struct proc \*p;

int ticket\_aggregate=0;

//loop over process table and increment total tickets if a runnable process is found for(p = ptable.proc; p < &ptable.proc[NPROC]; p++) {

ticket\_aggregate+=p->tickets;

}

return ticket\_aggregate; // returning total number of tickets for runnable processes

}

code for calculating total tickets



int scheduler(void)

{

struct proc \*p;

//struct proc \*p1;

struct cpu \*c = mycpu();

c->proc = 0;

//struct proc \*proc;

int count = 0;

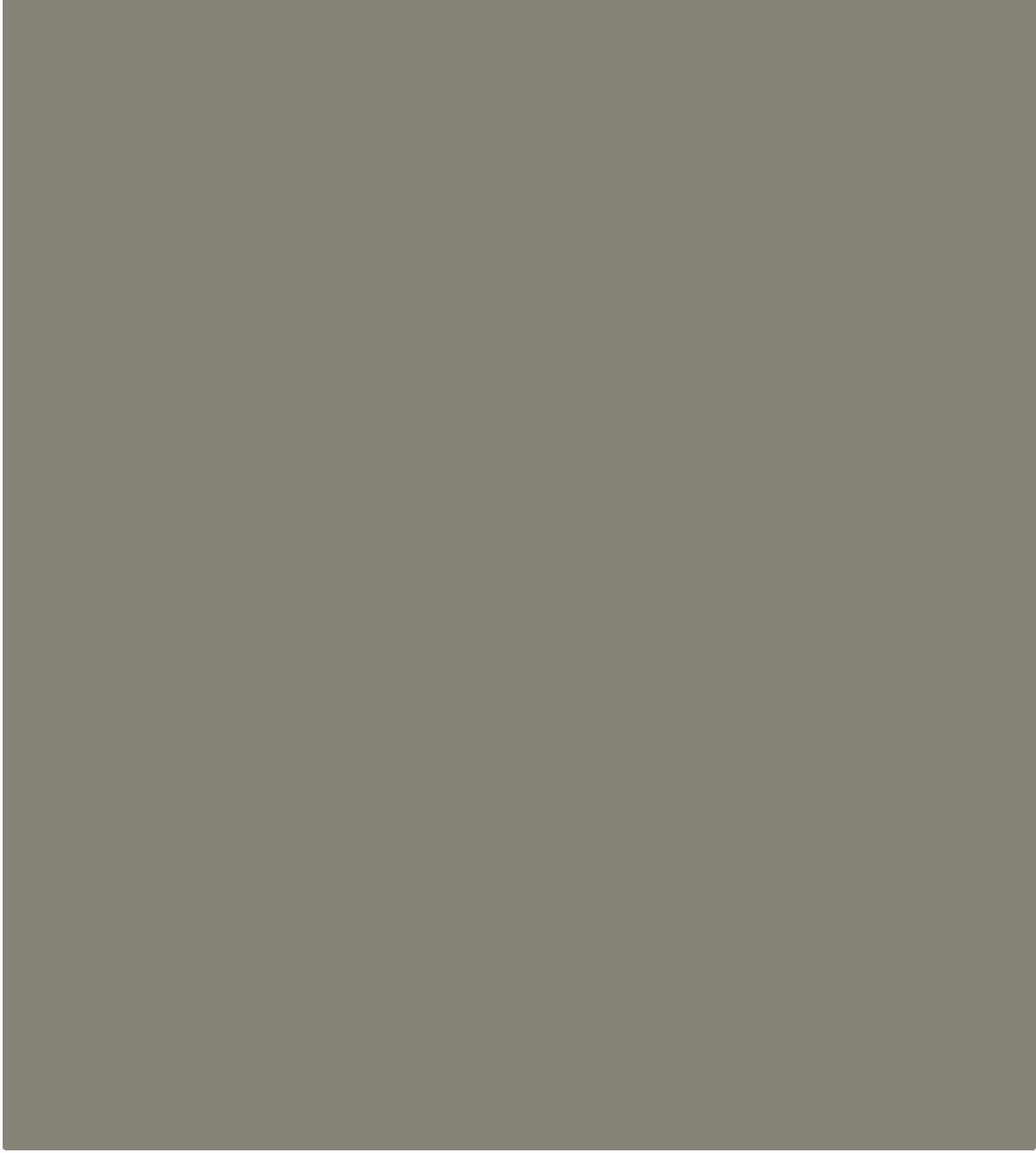
long golden\_ticket = 0;

int total\_no\_tickets = 0;

for(;;){

* Enable interrupts on this processor. sti();

Akshay Mall 2



* Loop over process table looking for process to run. acquire(&ptable.lock);

//resetting the variables to make scheduler start from the beginning of the process queue

golden\_ticket = 0;

count = 0;

total\_no\_tickets = 0;

//calculate Total number of tickets for runnable processes

total\_no\_tickets = lottery\_Total();

//pick a random ticket from total available tickets

golden\_ticket = random\_at\_most(total\_no\_tickets);

//cprintf("Golden ticket is:%d\n" , golden\_ticket);

for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){

if(p->state != RUNNABLE)

continue;

//find the process which holds the lottery winning ticket

if ((count + p->tickets) < golden\_ticket){

count += p->tickets;

continue;

}

* Switch to chosen process. It is the process's job
* to release ptable.lock and then reacquire it
* before jumping back to us.

//cprintf("Total ticket is:%d\n" , total\_no\_tickets);

c->proc = p;

switchuvm(p);

p->state = RUNNING;

swtch(&(c->scheduler), p->context);

switchkvm();

* Process is done running for now.
* It should have changed its p->state before coming back. c->proc = 0;

break;

}

release(&ptable.lock);

}

}

code for scheduler

3. User processes have a new system call, settickets , that allows a process to specify how

many lottery tickets it wants. Normally this would be a bad idea, since it would let a process hog the CPU by specifying an arbitrary number of tickets -- but xv6 has no security anyway, so this is not that big a deal.

|  |  |
| --- | --- |
| Akshay Mall | 3 |



int

sys\_settickets(void){

int pid, tic;

struct proc \*p;

acquire(&ptable.lock);

if(argint(0, &pid) < 0)

return -1;

for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){

if(p->pid == pid){

if(argint(1, &tic) < 0)

p->tickets = 10;

p->tickets = tic;

break;

}

}

release(&ptable.lock);

return pid;

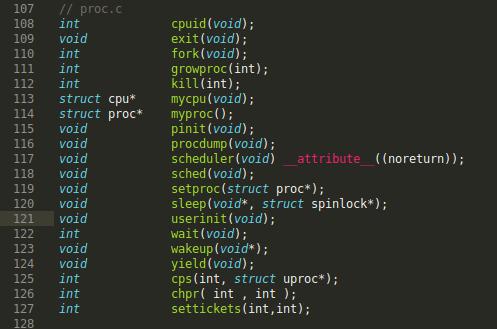
}

settickets system call in sysproc.c that sets the ticket for a specified process. If no ticket number has be specified then it will set default value of 10.

**Adding system call**

Modify the following:

def.h



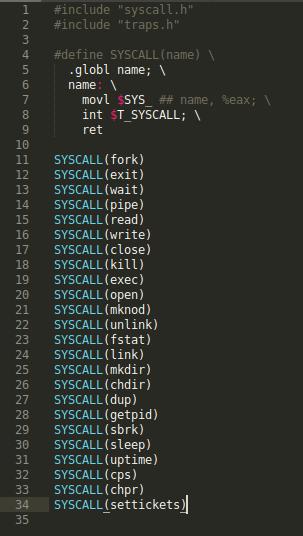
Akshay Mall 4

user.h



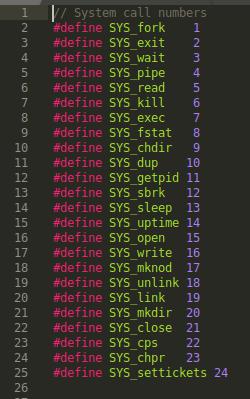
usys.S

Akshay Mall 5



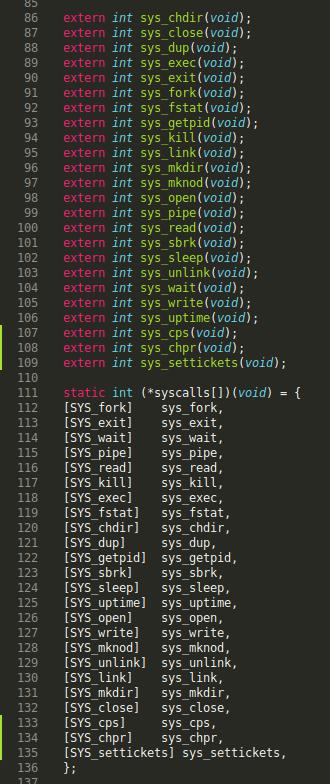
syscall.h

Akshay Mall 6



syscall.c

Akshay Mall 7



**Random number generator**

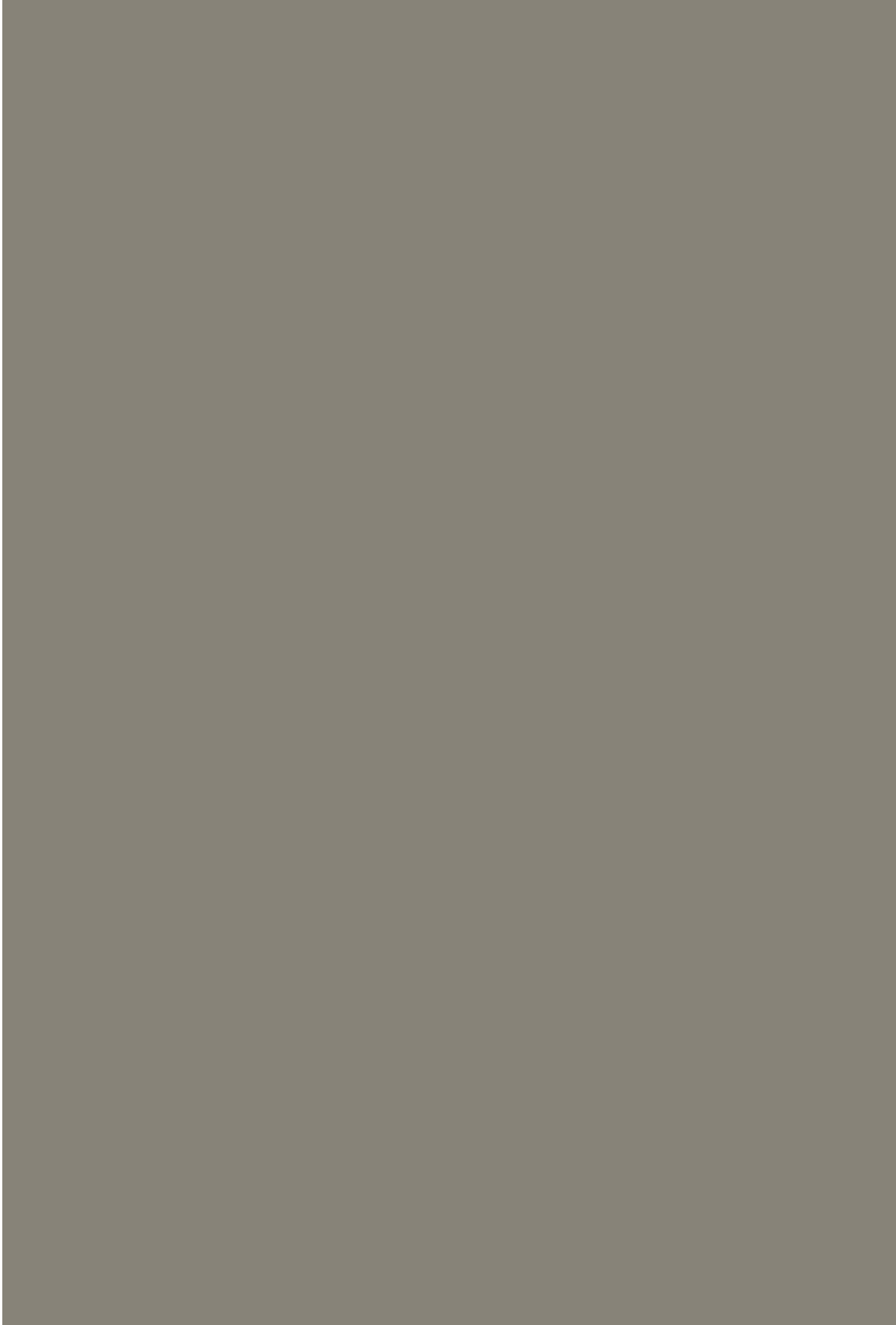
**Rand.c**



#define N 624

#define M 397

Akshay Mall 8



#define MATRIX\_A 0x9908b0df /\* constant vector a \*/

#define UPPER\_MASK 0x80000000 /\* most significant w-r bits \*/ #define LOWER\_MASK 0x7fffffff /\* least significant r bits \*/

/\* Tempering parameters \*/

#define TEMPERING\_MASK\_B 0x9d2c5680

#define TEMPERING\_MASK\_C 0xefc60000

#define TEMPERING\_SHIFT\_U(y) (y >> 11)

#define TEMPERING\_SHIFT\_S(y) (y << 7)

#define TEMPERING\_SHIFT\_T(y) (y << 15)

#define TEMPERING\_SHIFT\_L(y) (y >> 18)

#define RAND\_MAX 0x7fffffff

static unsigned long mt[N]; /\* the array for the state vector \*/ static int mti=N+1; /\* mti==N+1 means mt[N] is not initialized \*/

/\* initializing the array with a NONZERO seed \*/

void

sgenrand(unsigned long seed)

{

/\* setting initial seeds to mt[N] using \*/

/\* the generator Line 25 of Table 1 in \*/

/\* [KNUTH 1981, The Art of Computer Programming \*/

/\* Vol. 2 (2nd Ed.), pp102] \*/

mt[0]= seed & 0xffffffff;

for (mti=1; mti<N; mti++)

mt[mti] = (69069 \* mt[mti-1]) & 0xffffffff;

}

long /\* for integer generation \*/

genrand()

{

unsigned long y;

static unsigned long mag01[2]={0x0, MATRIX\_A};

/\* mag01[x] = x \* MATRIX\_A for x=0,1 \*/

if (mti >= N) { /\* generate N words at one time \*/

int kk;

if (mti == N+1) /\* if sgenrand() has not been called, \*/

sgenrand(4357); /\* a default initial seed is used \*/

for (kk=0;kk<N-M;kk++) {

* = (mt[kk]&UPPER\_MASK)|(mt[kk+1]&LOWER\_MASK); mt[kk] = mt[kk+M] ^ (y >> 1) ^ mag01[y & 0x1];

}

for (;kk<N-1;kk++) {

* = (mt[kk]&UPPER\_MASK)|(mt[kk+1]&LOWER\_MASK); mt[kk] = mt[kk+(M-N)] ^ (y >> 1) ^ mag01[y & 0x1];

}

* = (mt[N-1]&UPPER\_MASK)|(mt[0]&LOWER\_MASK);

mt[N-1] = mt[M-1] ^ (y >> 1) ^ mag01[y & 0x1];

mti = 0;

}

y = mt[mti++];

* ^= TEMPERING\_SHIFT\_U(y);
* ^= TEMPERING\_SHIFT\_S(y) & TEMPERING\_MASK\_B;
* ^= TEMPERING\_SHIFT\_T(y) & TEMPERING\_MASK\_C;
* ^= TEMPERING\_SHIFT\_L(y);

Akshay Mall 9



* Strip off uppermost bit because we want a long,
* not an unsigned long

return y & RAND\_MAX;

}

* Assumes 0 <= max <= RAND\_MAX
* Returns in the half-open interval [0, max] long random\_at\_most(long max) {

unsigned long

* max <= RAND\_MAX < ULONG\_MAX, so this is okay. num\_bins = (unsigned long) max + 1,

num\_rand = (unsigned long) RAND\_MAX + 1,

bin\_size = num\_rand / num\_bins,

defect = num\_rand % num\_bins;

long x;

do {

x = genrand();

}

* This is carefully written not to overflow while (num\_rand - defect <= (unsigned long)x);
* Truncated division is intentional

return x/bin\_size;

code for random number generator (rand.c)

**Rand.h**



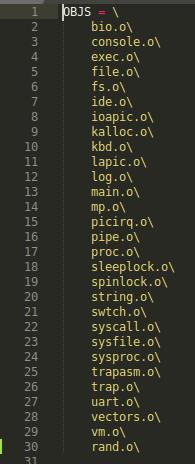
void sgenrand(unsigned long);

long genrand(void);

long random\_at\_most(long);

for scheduler to be able to access random\_at\_most() in rand.c we must add rand.o in the makefile

|  |  |
| --- | --- |
| Akshay Mall | 10 |



**Nice.c**

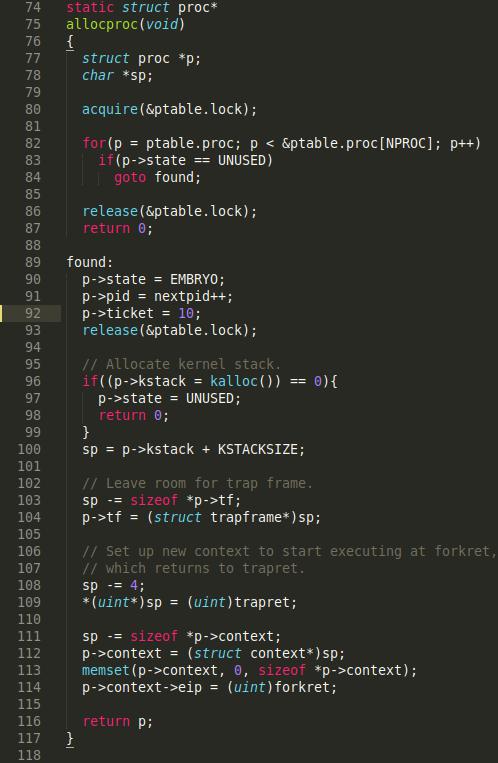
command that takes in pid and a number as ticket and calls settickets syscall to set the ticket number of the process with given pid.



Akshay Mall 11

**Allocproc.c**

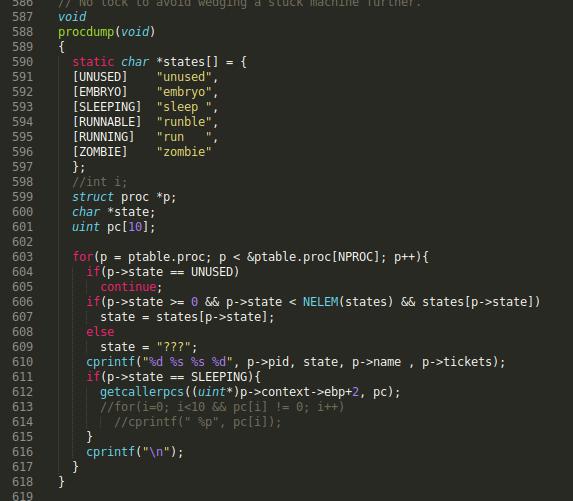
We have to make chnages in allocproc.c in proc.c to assign a ticket to a newly made process



Procdump.c

modifying procdump.c to show the ticket number of the process.

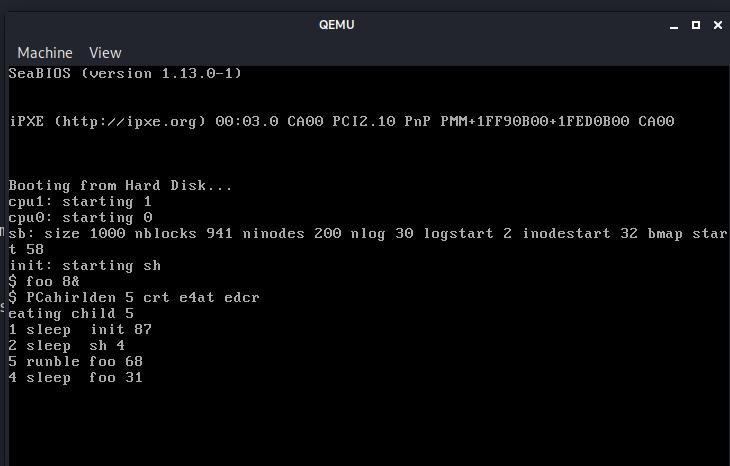
Akshay Mall 12



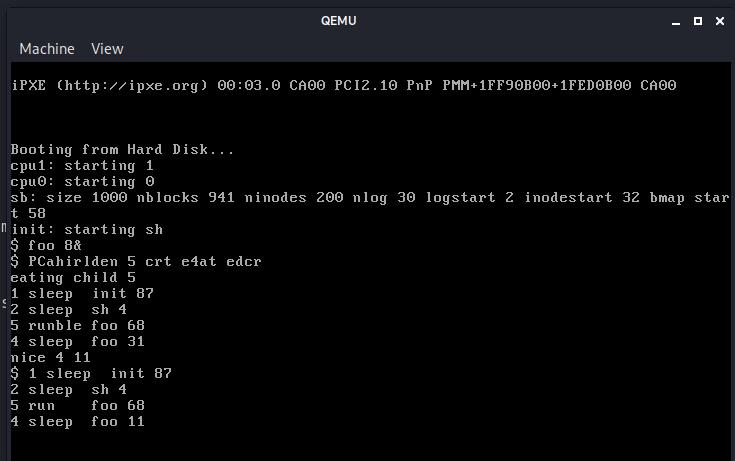
**Test Run lottery scheduling**

running foo 8 & and then doing ctrl+p will give the process dump of currently running processes which shows the pid, name, status and ticket number.

Akshay Mall 13



Now on running nice 4 11 will change the ticket number of process with pid 4



Akshay Mall 14

But we cannot predict when which process will run because a random number is selected for deciding the running process.

Akshay Mall 15