Lab 2: Modifying xv6 Scheduler

1. Lottery Scheduling

Files modified for implementing Lottery Scheduling:

- syscall.c
- sysproc.c
- syscall.h
- proc.c
- proc.h
- usys.S
- user.h

Libraries added to generate random number for selecting ticket:

- rand.c
- rand.h

Implementation:

- Assigning default tickets as 10 for every process

```
found:
  p->state = EMBRYO;
  p->pid = nextpid++;
  p->tickets = 10;
```

- Calculating the total tickets of all processes

```
int lottery_Total(void){
  struct proc *p;
  int ticket_aggregate=0;


for(p = ptable.proc; p < &ptable.proc[NPROC]; p++)
{
if(p->state==RUNNABLE){
ticket_aggregate+=p->tickets;
}
}
return ticket_aggregate;
}
```

- Implementing the Lottery Scheduler in place of default Round Robin Scheduler

```
//Lottery Scheduler
void
scheduler(void)
{
  struct proc *p;
  struct cpu *c = mycpu();
  c->proc = 0;
  int count = 0;
  long golden_ticket = 0;
  int total_no_tickets = 0;


  for(;;){

    sti();


to run.
    acquire(&ptable.lock);


                              //resetting the variables to make scheduler
    golden_ticket = 0;
    count = 0;
    total_no_tickets = 0;


                                                                        //calc
    total_no_tickets = lottery_Total();


    golden_ticket = random_at_most(total_no_tickets);

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

      if(p->state != RUNNABLE)
        continue;


                                                                        //fi
      if ((count + p->tickets) < golden_ticket){
        count += p->tickets;

        continue;
      }



      c->proc = p;
      switchuvm(p);
      p->state = RUNNING;

          p->usage=p->usage+1;
```

```
      }


      c->proc = p;
      switchuvm(p);
      p->state = RUNNING;

          p->usage=p->usage+1;
      swtch(&(c->scheduler), p->context);
      switchkvm();


      c->proc = 0;
break;
    }

    release(&ptable.lock);

  }
}
```

- Adding columns of tickets and usage in process structure struct proc

```
// Per-process state
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)
  int tickets;                 // Number of tickets for each process
  int usage;                   // Number of times each process is scheduled
```

- Adding a new system call to set the number of tickets

```
int
sys_settickets(void){
  int ticket_number;
  if (argint(0, &ticket_number) < 0)
  {
     myproc()->tickets = 10;
  }
  else{
     myproc()->tickets = ticket_number;
  }
  return 0;
}
```

- Modifying the files syscall.h, syscall.c and usys.S for adding system call of changing the number of tickets and MakeFile to add the user program.

2. Stride Scheduling

Files modified for implementing Stride Scheduling:

- syscall.c
- sysproc.c
- syscall.h
- proc.c
- proc.h
- usys.S
- user.h

Implementation:

- Initializing the number of strides to 10000, number of tickets to 1 and pass to 0

```
found:
  p->state = EMBRYO;
  p->pid = nextpid++;
  p->tickets = 1;
  p->pass = 0;
  p->stride = 10000 / p->tickets;

  release(&ptable.lock);
```

- Adding the system calls to set the number of tickets and get the usage of each process

```
int
sys_settickets(void) {
  int tix;
  if(argint(0, &tix) < 0)
        {return -1;}
        myproc()->tickets = tix;
        myproc()->stride = 10000 / tix;
        return 0;
}

int
sys_getusage(void) {
  return myproc()->usage;
}
```

- Adding the columns for storing the number of tickets, usage, stride and pass to process structure struct proc

```
// Per-process state
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)
  int tickets;                 // Number of tickets of each process
  int usage;                   // Number of times each process is called
  int stride;                  // Strides of each process
  int pass;                    // Pass for each process

};
```

- Implementing Stride scheduler in place of default Round Robin Scheduler

```
void
scheduler(void)
{
  struct proc *p;
  struct proc *current;
  struct cpu *c = mycpu();
  c->proc = 0;


  for(;;){

    sti();
    int minPass = -1;

    acquire(&ptable.lock);

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

      if(p->state != RUNNABLE)
        continue;
      if (minPass < 0 || p->pass < minPass){
        current = p;
                                minPass = p->pass;
                        }

      |
```

```


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

      if(p->state != RUNNABLE)
        continue;

      if (p->pass==minPass){
        current=p;
        c->proc=current;
        current->pass += current->stride;
        switchuvm(current);
        current->state = RUNNING;
        current->usage = current->usage+1;
        swtch(&(c->scheduler), p->context);
        switchkvm();


      c->proc = 0;
      break;
      }
    }
    release(&ptable.lock);

  }
}|
```

- Modifying the files syscall.h, syscall.c and usys.S for adding system call of changing the number of tickets and getting the usage for each process and MakeFile to add the user program.

Performance

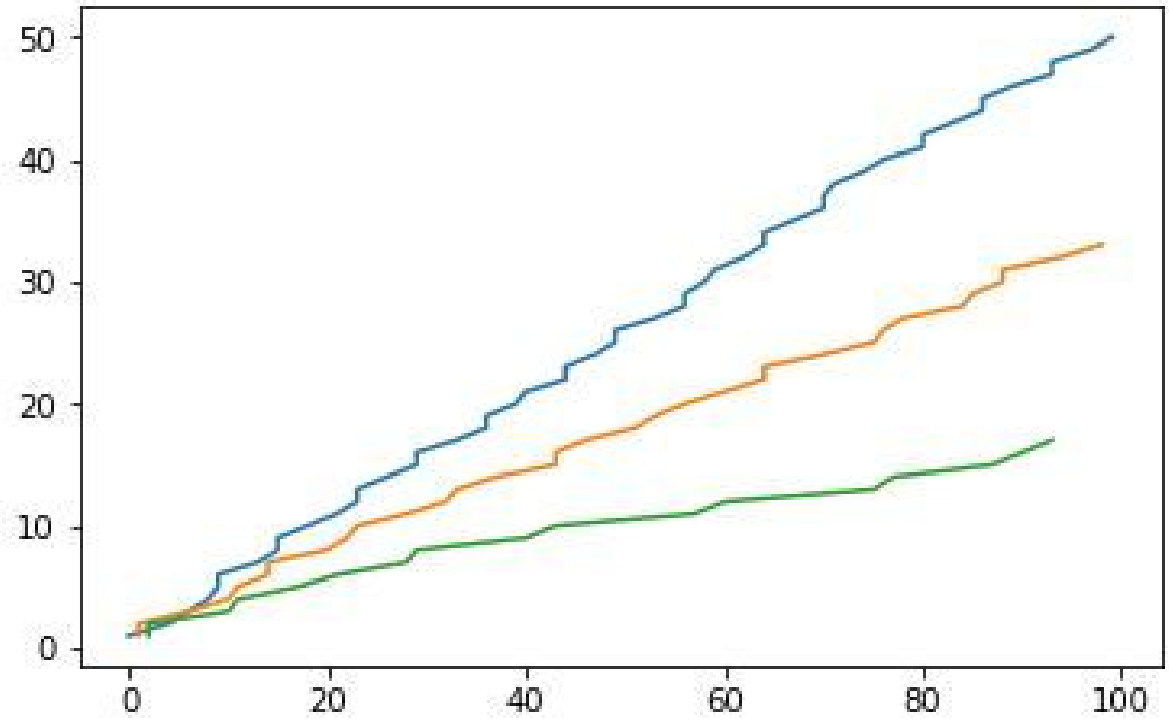
Lottery Scheduler:

```
$ lotteryTest1

  child# 6 with 30 tickets has finished!
From  lotteryTest1-6: 1 sleep  init sched_times=24 ticket=10
From  lotteryTest1-6: 2 sleep  sh sched_times=22 ticket=10
From  lotteryTest1-6: 5 sleep  lotteryTest1 sched_times=37 ticket=60
From  lotteryTest1-6: 6 run    lotteryTest1 sched_times=366 ticket=30
From  lotteryTest1-6: 7 runble lotteryTest1 sched_times=226 ticket=20
From  lotteryTest1-6: 8 runble lotteryTest1 sched_times=114 ticket=10

  child# 7 with 20 tickets has finished!
From  lotteryTest1-7: 1 sleep  init sched_times=24 ticket=10
From  lotteryTest1-7: 2 sleep  sh sched_times=22 ticket=10
From  lotteryTest1-7: 5 sleep  lotteryTest1 sched_times=38 ticket=60
From  lotteryTest1-7: 7 run    lotteryTest1 sched_times=366 ticket=20
From  lotteryTest1-7: 8 runble lotteryTest1 sched_times=169 ticket=10

  child# 8 with 10 tickets has finished!
From  lotteryTest1-8: 1 sleep  init sched_times=24 ticket=10
From  lotteryTest1-8: 2 sleep  sh sched_times=22 ticket=10
From  lotteryTest1-8: 5 sleep  lotteryTest1 sched_times=39 ticket=60
From  lotteryTest1-8: 8 run    lotteryTest1 sched_times=374 ticket=10
From  lotteryTest1-5: 1 sleep  init sched_times=24 ticket=10
From  lotteryTest1-5: 2 sleep  sh sched_times=22 ticket=10
From  lotteryTest1-5: 5 run    lotteryTest1 sched_times=41 ticket=60
$
```



Stride Scheduler:

```
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ strideTest1

  child# 4 with 30 tickets has finished!
From  strideTest1-4: 1 sleep  init sched_times=22 ticket=1 stride=10000
From  strideTest1-4: 2 sleep  sh sched_times=16 ticket=1 stride=10000
From  strideTest1-4: 3 sleep  strideTest1 sched_times=9 ticket=60 stride=166
From  strideTest1-4: 4 run    strideTest1 sched_times=449 ticket=30 stride=333
From  strideTest1-4: 5 run    strideTest1 sched_times=321 ticket=20 stride=500
From  strideTest1-4: 6 runble strideTest1 sched_times=161 ticket=10 stride=1000

  child# 5 with 20 tickets has finished!
From  strideTest1-5: 1 sleep  init sched_times=22 ticket=1 stride=10000
From  strideTest1-5: 2 sleep  sh sched_times=16 ticket=1 stride=10000
From  strideTest1-5: 3 sleep  strideTest1 sched_times=10 ticket=60 stride=166
From  strideTest1-5: 5 run    strideTest1 sched_times=459 ticket=20 stride=500
From  strideTest1-5: 6 run    strideTest1 sched_times=294 ticket=10 stride=1000

  child# 6 with 10 tickets has finished!
From  strideTest1-6: 1 sleep  init sched_times=22 ticket=1 stride=10000
From  strideTest1-6: 2 sleep  sh sched_times=16 ticket=1 stride=10000
From  strideTest1-6: 3 sleep  strideTest1 sched_times=11 ticket=60 stride=166
From  strideTest1-6: 6 run    strideTest1 sched_times=466 ticket=10 stride=1000
From  strideTest1-3: 1 sleep  init sched_times=22 ticket=1 stride=10000
From  strideTest1-3: 2 sleep  sh sched_times=16 ticket=1 stride=10000
From  strideTest1-3: 3 run    strideTest1 sched_times=13 ticket=60 stride=166
$
```