Scheduling

Satoru Yamamoto

04/23/2022

Contents

1 late submission

I talked to Professor via MS Teams chat, and I received this message1

2 system calls

As we did in previous assignment, I first create system call for settickets and getpinfo after I created pstat.h for pstat struct.

2.1 settickets

This system call sets the number of tickets of calling process. To pass a value into a system call function, I used this stackoverflow page.[1]

sysproc.c

```
int
sys_settickets(void)
{
  int num_tikets;
  // if it can receive
  if (argint(0, &num_tikets) < 0)
    return -1;
  //cprintf("argint: %d\n" ,num_tikets);
  return settickets(num_tikets);
}
```

proc.c

```
int settickets(int tickets)
{
  if(tickets <0){
    return -1;
  }

  struct proc* current_process = myproc();
  current_process->tickets = tickets;

  return 0;
}
```

2.2 getpinfo

This system call returns some basic information about each running process, including how many times it has been chosen to run, its process ID, and which queue it is on (high or low).[2]
For settickets test, I used this code.

sysproc.c

```
int
sys_getpinfo(void)
{
  struct pstat* pinfo;
  //if(argfd(0, 0, &f) < 0 || argptr(1, (void*)&st, sizeof(*st)) < 0)
  if (argptr(0,(void*)&pinfo,sizeof(*pinfo)) < 0)
    return -1;
  return getpinfo(pinfo);
}
```

proc.c

```
int getpinfo(struct pstat *pinfo)
{
  struct proc * myproc;
  //cprintf("test\n");

  // Loop over process table looking for process to run.
  acquire(&ptable.lock);
  //cprintf("test\n");
  int index = 0;
  for (myproc = ptable.proc; myproc < &ptable.proc[NPROC]; myproc++){
    //cprintf("index: %d\n", index);
    if(myproc->pid == 0 && myproc->state == UNUSED){
      pinfo->inuse[index] = 0;
    }else{
      pinfo->inuse[index] = 1;
    }
    pinfo->pid[index] = myproc->pid;
    pinfo->hticks[index] = myproc->high_ticks;
    pinfo->lticks[index] = myproc->low_ticks;
    // change it to 1 if you want to display
    if(pinfo->inuse[index] == 2){
      cprintf("---\n");
      cprintf("parent pid:%d\n",myproc->parent->pid);
      cprintf("name: %s\n",myproc->name);
      cprintf("pid: %d, queue: %d, tickets: %d\n", myproc->pid, myproc->pri_queue, myproc->tickets);
      cprintf("pid: %d, hticks: %d, lticks: %d\n", pinfo->pid[index],
          ↪  pinfo->hticks[index],pinfo->lticks[index]);
      cprintf("---\n");
    }
    index++;
  }
  release(&ptable.lock);

  return 0;
}
```

3 MLFQ and lottery scheduler

The brief program flowchart for scheduling is:

- main.c call mpmain function after it finish processor's setup
- scheduler function is called by mpmain
- scheduler has infinite loop
- inside of infinite loop, it has for loop that loops through the process table to search RUNNABLE process.
- if it finds RUNNABLE process, it switch the process to it (context switch)

As we see how the program pick a process to run, we can compare the priority of processes to pick the highest priority processes. We are able able to count the total tickets for higher priority processes to create winner value for lottery scheduling. Then my scheduler pick one winner process*2.
For the priority boost, I define is_boost variable to get if the scheduler is ready to increase all process priorities. According to the material, "Every tick, the hardware clock forces an interrupt, which is handled in trap() by case T_IRQ0 + IRQ_TIMER"[3]. Therefore, I add this code to trap.c:

trap.c

```
case T_IRQ0 + IRQ_TIMER:
if(cpuid() == 0){
  acquire(&tickslock);
  ticks++;

  // set 1 each 100 ticks
  if(ticks%100 == 0){
    is_boost=1;
  }
  wakeup(&ticks);
  release(&tickslock);
  //cprintf("ticks: %d\n", ticks);
}
```

In proc.h, I add these to proc struct.

proc.h

```
int tickets;                // assigned tickets
int pri_queue;              // queue num (0 - 1)
int high_ticks;             // ticks for high queue (1)
int low_ticks;              // ticks for low queue (0)
```

These variables are initialized when unused process is found in allocproc function.

proc.c

```
p->tickets = 1;
//  first places jobs into the high-priority queue
p->pri_queue = 1;
p->high_ticks = 0;
p->low_ticks = 0;
```

In scheduler function, I first check what priority is the highest, and total tickets of processes with that priority.

proc.c

```
int sum_high_tickets = 0, sum_low_tickets = 0;
int running_priority = 0;
// get total tickets

for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
  if(p->state != RUNNABLE)
    continue;

  if(p->pri_queue == 1){
    sum_high_tickets += p->tickets;
    running_priority = 1;
  }else if(p->pri_queue == 0){
    sum_low_tickets += p->tickets;
  }
}
int winner = random_at_most(running_priority*(sum_high_tickets) +
    ↪  (1-running_priority)*sum_low_tickets);
```

Then I pick/run the winner process.

proc.c

```
int counter = 0;
for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
  if(p->state != RUNNABLE || p->pri_queue != running_priority)
    continue;

  counter+=p->tickets;
  //if(counter>winner)
  //  break;
  if(counter < winner)
    continue;

  // Switch to chosen process.  It is the process's job
  // to release ptable.lock and then reacquire it
  // before jumping back to us.
  c->proc = p;
  switchuvm(p);
  p->state = RUNNING;
  //cprintf("RUNNING: %s [pid %d]\n", proc->name, proc->pid);
  swtch(&(c->scheduler), p->context);
  switchkvm();
  ... other codes
  }
```

After running process, I check the priority of it to grade down it's priority and to count ticks.

proc.c

```
// if priority is 1, if priority is 0
  if(running_priority == 1){
    p->high_ticks++;
    p->pri_queue = 0;
  }else if(running_priority == 0){
    p->low_ticks++;
  }
```

After these steps, if the scheduler is ready to boost, I increment priority of processes.

proc.c

```
if(is_boost){
    cprintf("priority boost: [%d]\n", ticks);
    // boost priority to 1
    struct proc *temp_proc;
    for(temp_proc = ptable.proc; temp_proc < &ptable.proc[NPROC]; temp_proc++){
      temp_proc->pri_queue = 1;
    }
    //reset
    is_boost = 0;
  }
```

For the random number generator, I used these materials: [4], [5]. LOWER_MASK is the highest random number created by genrand() function. Therefore, I replaced RAND_MAX to LOWER_MASK.

4 testing

To test my program, I get some idea from the lecture video uploaded in YouTube[6].
I create infinite loop spin that creates tasks from each process I create. After 500ms, I store pinfo for each process to display test result. Kill all process before exit() because it has infinite loop.
The test result is here: *3

Each process has high ticks = 7 because I boost priority 6 times. Each process has high priority at the beginning, so 6 + 7. Then I can see that more tickets makes more process in lower priority. For example, the process with 10 tickets has 0 low ticks, and the process with 1250000 tickets has 519 low ticks. Therefore, I implemented MLFQ and lottery scheduler in my program.
Each test results different values. *4

References

[1] https://stackoverflow.com/questions/27068394/how-to-pass-a-value-into-a-system-call-function-in-xv6

[2] https://stackoverflow.com/questions/53383938/pass-struct-to-xv6-system-call

[3] https://pdos.csail.mit.edu/6.828/2014/homework/xv6-alarm.html

[4] http://www.math.sci.hiroshima-u.ac.jp/m-mat/MT/VERSIONS/C-LANG/980409/mt19937-1.c

[5] https://code-examples.net/en/q/264b6f

[6] https://www.youtube.com/watch?v=eYfeOT1QYmg

Figure 1: message from Professor

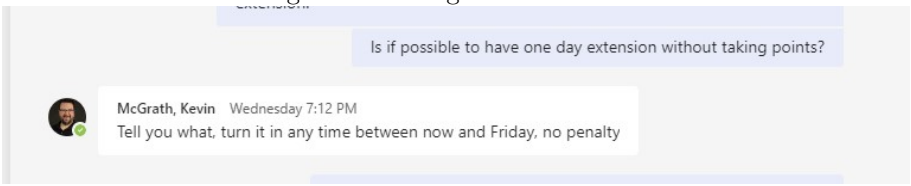

Figure 2: lottery sample code

```
// counter: used to track if we've found the winner yet
int counter = 0;

// winner: use some call to a random number generator to
//         get a value, between 0 and the total # of tickets
int winner = getrandom(0, totaltickets);

// current: use this to walk through the list of jobs
node_t *current = head;
while (current) {
    counter = counter + current->tickets;
    if (counter > winner)
        break; // found the winner
    current = current->next;
}
// 'current' is the winner: schedule it...
```

Figure 9.1: **Lottery Scheduling Decision Code**

Figure 3: test result

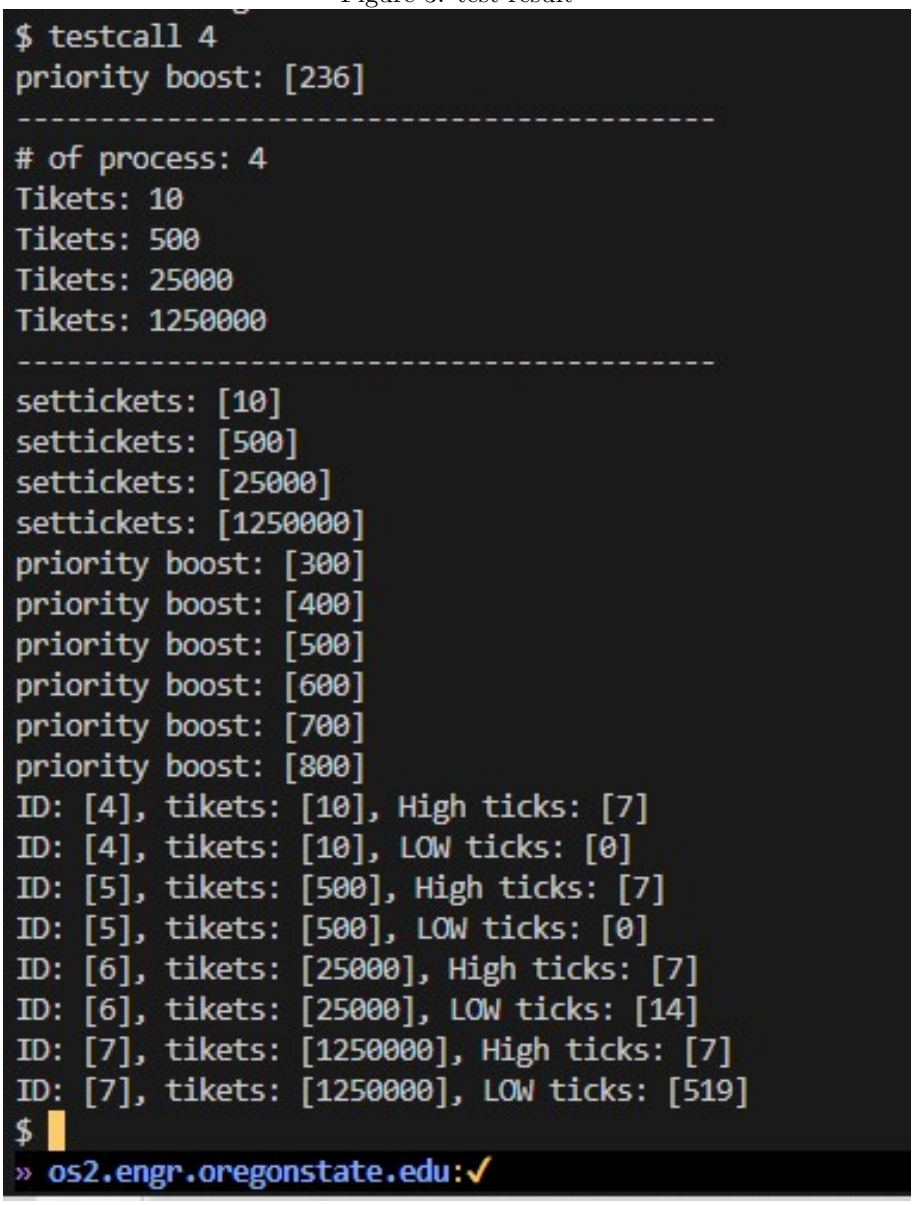

Figure 4: test result2