Operating System Lab Project 3 Report

Reza Abdoli
Mohammad Reza Alavi
Farbod Azimmohseni
Commit Id: 088aae0943541cf9e8042f4a7ab71240e3053e28

December 10, 2023

Questions

1. **Sched()** is called when we want to reload context of scheduler. It is called in three situations. First when a process exits the **Sched()** is called the context of scheduler loads and continues running from where it called **swtch** to run the process. The other situation is when we call sleep and the last one is when **yeild** is called after one time slice to retrieve cpu from process.

2. In **CFS**, scheduler sets a target latency which is the ideal amount of time for all runnable tasks to finish their job. Then it breaks this time into the number of processes and sets each of them part of its time. A red-black tree is implemented and value of each node is the minimum time slice that process needs.

3. Linux uses a seperate schedule for each core and have differnt qeueus. On the other hand Xv6 uses a single queues. The most important adavantage for a single queue is a simple implementation because there is no need to worry about **load balancing**. The problem is that we need a lock to ensure synchronisation of cores. Multiple queues improves performance of operating systems running on multicore systems.

4. When ptable lock is activated all interrupts will be deactivated and some processes might be waiting for I/O and none of processes are Runnable, so none of the processes will run and because interrupts won't be activated after I/O, we can't change processes status to Runnable so in this loop before locking ptable we activate interrupts and we can change processes status.

5. There are two levels of interrupt handling in linux: First one is FLIH (First LevelInterrupts Handler) or upper half and the Second one is SLIH (Second Level Interrupts Handler) or lower half. FLIH is for handling essential interrupts as soon as possible it is answering interrupt or storing its essential data and It will be handled by SLIH. context switching happens in FLIH. SLIH is used for handling interrupts that take more time. It is handled by a thread pool or a kernel level thread. SLIHs are in a queue waiting for processer. Starvation happens when higher priority queus starve the low priority one. To handle this situation each time a runnable process doesnt get scheduled we add to the priorty of it until it get scheduled.

MLFQ Implementation

Code of **MLFQ** is shown here. It first checks for processes that are in **Round Robin** queue then for all other queues up to the last queue. On initialization are processes are assigned to **LCFS** queue except for **sh** and **init**.

```
for(;;){
  // Enable interrupts on this processor.
  // Loop over process table looking for process to run.
  sti();
  acquire(&ptable.lock);
  p = round_robin(last_p);
  if (p == 0){
    p = lcfc_schedule();      You, 21 hours ago • test file adde
    if (p ==0){
      p = bjf_schedule();
      if (p == 0){
        release(&ptable.lock);
        continue;
      }
    }
  }
  else{
    last_p = p;
  }
  // 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;

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

Figure 1: MLFQ code

This part of code increase cycle values of all runnable processes after each time slice. If number of cycles in a process reach 8000 it will move to **Round Robin** queue with higher priority.

```
for(;;){
  // Enable interrupts on this processor.
  // Loop over process table looking for process to run.
  sti();
  acquire(&ptable.lock);
  p = round_robin(last_p);
  if (p == 0){
    p = lcfc_schedule();      You, 21 hours ago • test file adde
    if (p ==0){
      p = bjf_schedule();
      if (p == 0){
        release(&ptable.lock);
        continue;
      }
    }
  }
  else{
    last_p = p;
  }
  // 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;

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

Figure 2: aging code

foo

A infinite loop is given as a child program for **foo** program to test aging. These are the results: Process with pid 8 has moved from queue 2 to 1.

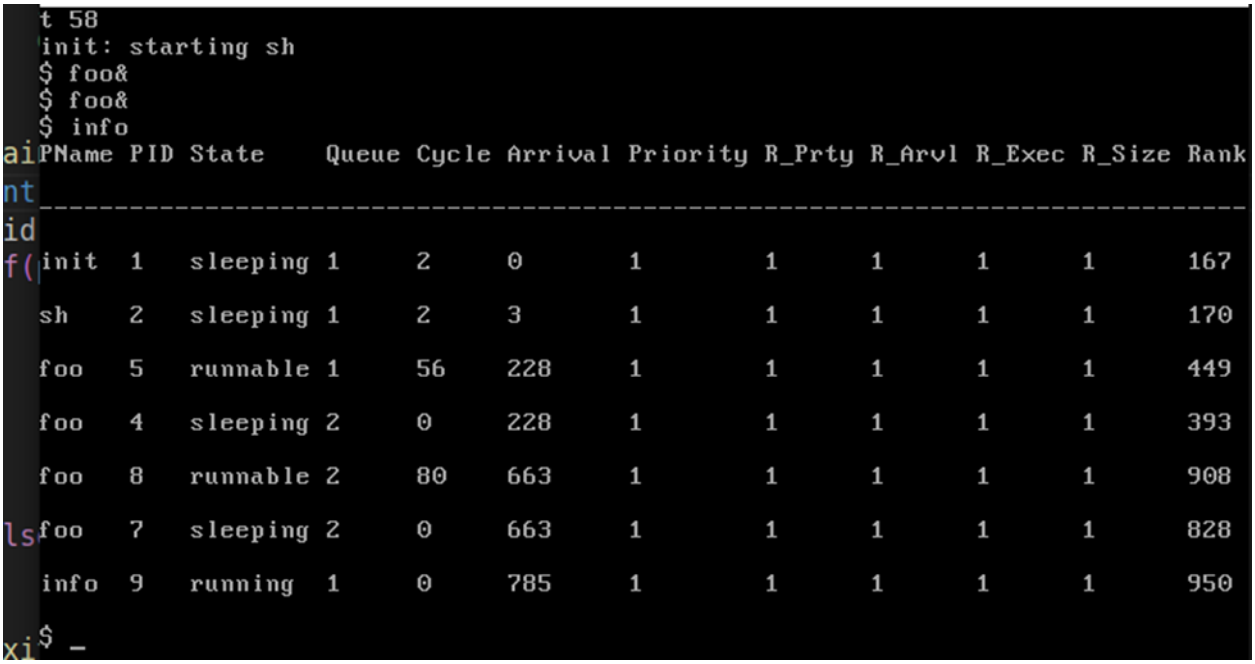

Figure 3: calling foo two times

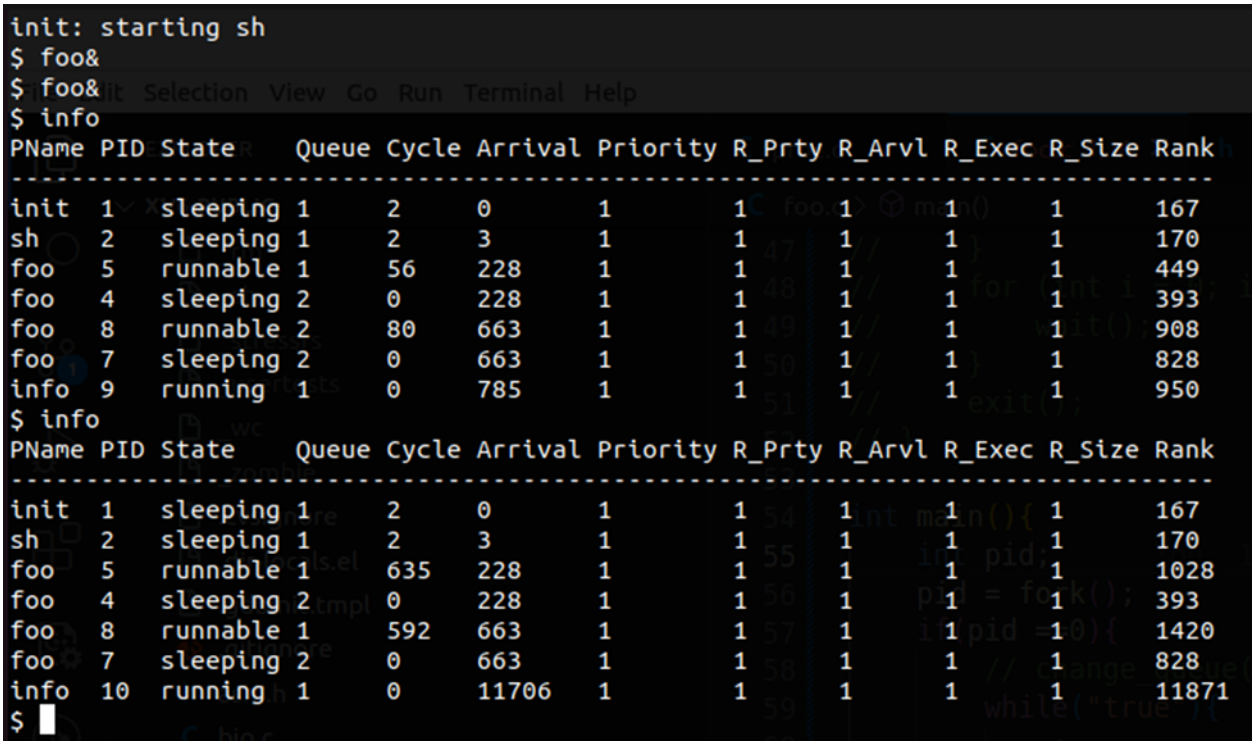

Figure 4: aging result