REPORT – 1

Aditya Raghuvanshi
2021114009

The report must contain explanation about the implementation of the various implemented scheduling algorithms.

FCFS

This is the code I made for FCFS which I included in the scheduler function in proc.c file.

```
#ifdef FCFS

void scheduler(void)
{
  struct proc *p;
  struct cpu *c = mycpu();
  c->proc = 0;
  printf("FCFS");

  for (;;)
  {
    // Avoid deadlock by ensuring that devices can interrupt.
    intr_on();
    struct proc *next_process = 0;

    for (p = proc; p < &proc[NPROC]; p++)
    {
      acquire(&p->lock);
      if (p->state == RUNNABLE)
      {
        next_process = p;
        break;
      }
    }
    for (p++; p < &proc[NPROC]; p++)
    {
      acquire(&p->lock);
      if (p->state == RUNNABLE && next_process->ctime > p->ctime)
      {
        next_process = p;
        continue;
      }
    }
    for (p = proc; p < &proc[NPROC]; p++)
    {
      if (p != next_process)
```

```
      {
        release(&p->lock);
      }
    }
    p = next_process;
    if (next_process != 0)
    {
      // printf("%d ", p->pid);
      // printf("%d\n", p->pid);
      // Switch to chosen process.  It is the process's job
      // to release its lock and then reacquire it
      // before jumping back to us.
      p->state = RUNNING;
      c->proc = p;
      swtch(&c->context, &p->context);
      // Process is done running for now.
      // It should have changed its p->state before coming back.
      c->proc = 0;
      release(&p->lock);
    }
  }
}

#endif
```

B) MLFQ

Made these additions to struct proc

```
int queue;
int slice;
uint64 wtime;
int timesscheduled;
uint64 queueenter;
```

Made changes to usertrap in trap.c to handle changing queues

```
#ifdef MLFQ
  if (which_dev == 2)
  {
    struct proc *p = myproc();
    int queue = p->queue;
    switch (queue)
    {
    case 0:
      p->slice = 0;
      p->queue++;
```

```
      p->enter_queue = ticks;
      yield();
      break;
    case 1:
      if (p->slice != 2)
      {
        p->slice++;
        break;
      }
      else
      {
        p->slice = 0;
        p->queue++;
        p->enter_queue = ticks;
        yield();
        break;
      }
    case 2:
      if (p->slice != 8)
      {
        p->slice++;
        break;
      }
      else
      {
        p->slice = 0;
        p->queue++;
        p->enter_queue = ticks;
        yield();
        break;
      }
    case 3:
      if (p->slice != 14)
      {
        p->slice++;
        break;
      }
      else
      {
        p->slice = 0;
        p->queue++;
        p->enter_queue = ticks;
        yield();
        break;
      }
    }
  }
#endif
```

Also made the same changes to kernaltrap :

```
#ifdef MLFQ
 if (which_dev == 2 && myproc() != 0 && myproc()->state == RUNNING)
 {
   struct proc *p = myproc();
   int queue = p->queue;
   switch (queue)
   {
   case 0:
     p->slice = 0;
     p->queue++;
     p->enter_queue = ticks;
     yield();
     break;
   case 1:
     if (p->slice != 2)
     {
       p->slice++;
       break;
     }
     else
     {
       p->slice = 0;
       p->queue++;
       p->enter_queue = ticks;
       yield();
       break;
     }
   case 2:
     if (p->slice != 8)
     {
       p->slice++;
       break;
     }
     else
     {
       p->slice = 0;
       p->queue++;
       p->enter_queue = ticks;
       yield();
       break;
     }
   case 3:
     if (p->slice != 14)
     {
       p->slice++;
       break;
     }
     else
     {
       p->slice = 0;
       p->queue++;
```

```
        p->enter_queue = ticks;
        yield();
        break;
      }
    }
  }
#endif
```

Made changes to scheduler

```
#ifdef MLFQ
void scheduler(void)
{
  printf("MLFQ is running\n");
  struct proc *p;
  struct cpu *c = mycpu();
  c->proc = 0;
  for (;;)
  {
    intr_on();
    for (p = proc; p < &proc[NPROC]; p++)
    {
      if(p->state == RUNNABLE)
      {
        acquire(&p->lock);
        if(p->queue!=0&&p->wtime>=32)
        {
          p->wtime=0;
          p->queue--;
          p->enter_queue=ticks;
        }
        release(&p->lock);
      }
    }
    // Avoid deadlock by ensuring that devices can interrupt.
    struct proc *selected = 0;
    for (p = proc; p < &proc[NPROC]; p++)
    {
      acquire(&p->lock);
      if (p->state == RUNNABLE && p->queue == 0)
      {
        if (selected == 0)
        {
          selected = p;
          continue;
        }
        if (p->enter_queue < selected->enter_queue)
        {
          release(&selected->lock);
          selected = p;
```

```
        continue;
      }
    }
    release(&p->lock);
  }
  if(selected!=0)
  {
    //printf("MLFQ is running\n");
    selected->state = RUNNING;
    selected->wtime=0;
    c->proc = selected;
    swtch(&c->context, &selected->context);
    c->proc = 0;
    release(&selected->lock);
    continue;
  }
  // selected = 0;
  for (p = proc; p < &proc[NPROC]; p++)
  {
    acquire(&p->lock);
    if (p->state == RUNNABLE && p->queue == 1)
    {
      // p->wtime++;
      // if(p->wtime==32)
      // {
      //   p->queue--;
      //   p->wtime=0;
      //   selected = p;
      //   break;
      // }
      if (selected == 0)
      {
        selected = p;
        continue;
      }
      if (p->enter_queue < selected->enter_queue)
      {
        release(&selected->lock);
        selected= p;
        continue;
      }
    }
    release(&p->lock);
  }
  if(selected!=0)
  {
    selected->state = RUNNING;
    selected->wtime=0;
    c->proc = selected;
    swtch(&c->context, &selected->context);
    c->proc = 0;
```

```
      release(&selected->lock);
      continue;
    }
    // selected = 0;
    for (p = proc; p < &proc[NPROC]; p++)
    {
      acquire(&p->lock);
      if (p->state == RUNNABLE && p->queue == 2)
      {
        // p->wtime++;
        // if(p->wtime==32)
        // {
        //   p->queue--;
        //   p->wtime=0;
        //   selected = p;
        //   break;
        // }
        if (selected == 0)
        {
          selected = p;
          continue;
        }
        if (p->enter_queue < selected->enter_queue)
        {
          release(&selected->lock);
          selected= p;
          continue;
        }
      }
      release(&p->lock);
    }
    if(selected!=0)
    {
      selected->state = RUNNING;
      selected->wtime=0;
      c->proc = selected;
      swtch(&c->context, &selected->context);
      c->proc = 0;
      release(&selected->lock);
      continue;
    }
    // selected = 0;
    for (p = proc; p < &proc[NPROC]; p++)
    {
      acquire(&p->lock);
      if (p->state == RUNNABLE && p->queue == 3)
      {
        // p->wtime++;
        // if(p->wtime==32)
        // {
        //   p->queue--;
```

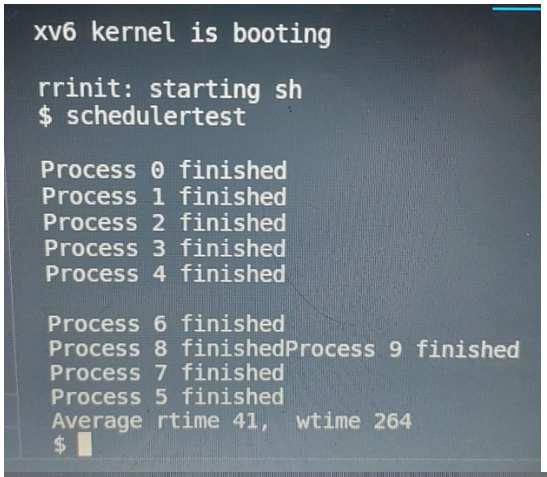
```
        //   p->wtime=0;
        //   selected = p;
        //   break;
        // }
        if (selected == 0)
        {
          selected = p;
          continue;
        }
        if (p->enter_queue < selected->enter_queue)
        {
          release(&selected->lock);
         selected= p;
          continue;
        }
      }
      release(&p->lock);
    }
    if(selected!=0)
    {
      selected->state = RUNNING;
      selected->wtime=0;
      c->proc = selected;
      swtch(&c->context, &selected->context);
      c->proc = 0;
      release(&selected->lock);
    }
    // selected = 0;
  }
}
#endif
```

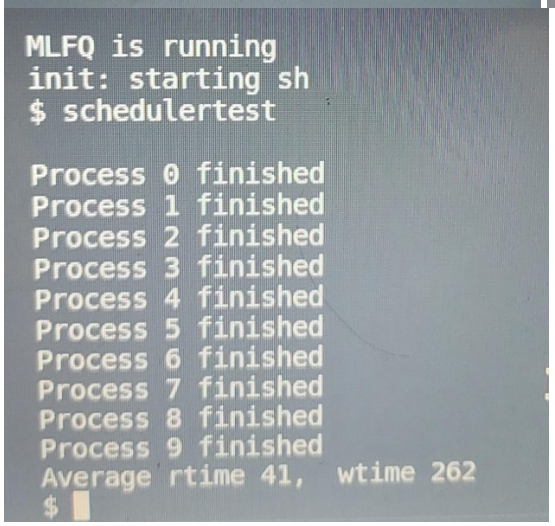
UPDATE TIME FUNCTION :

```
void update_time()
{
  struct proc *p;
  for (p = proc; p < &proc[NPROC]; p++)
  {
    acquire(&p->lock);
    if (p->state == RUNNING)
    {
      p->rtime++;
    }
    else if (p->state == SLEEPING)
    {
      p->stime++;
    }
    else if(p->state ==RUNNABLE)
    {
      p->wtime++;
    }
```

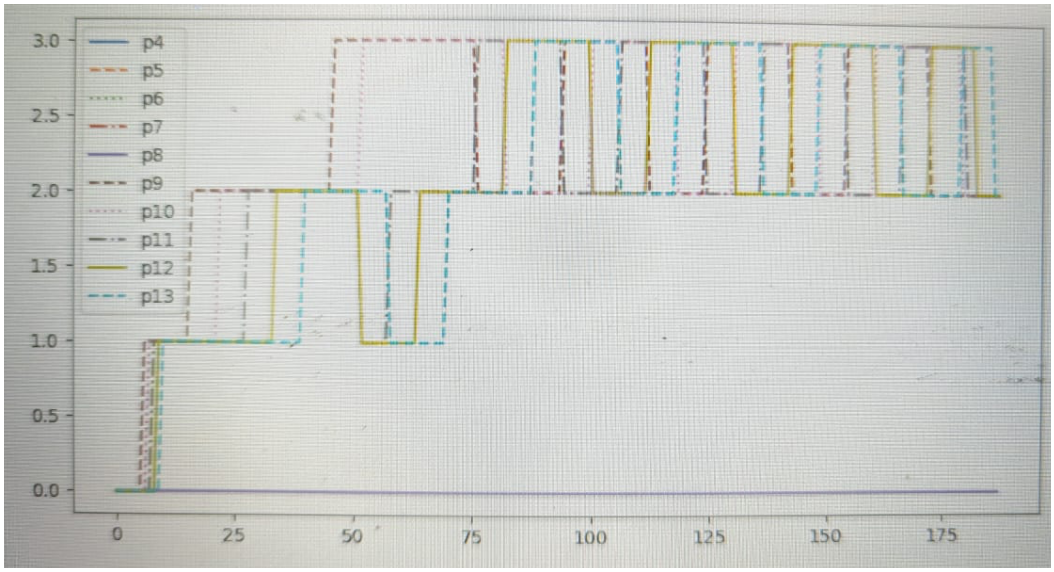
```
    release(&p->lock);
  }
}
```

COMPARISION BETWEEN ALGORITHMS:





GRAPH:


p4
p5
p6
p7
p8
p9
p10
p11
p12
p13
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
25
50
75
100
125
150
175