Part 3)

الناز رضایی– حوریه سبزواری
98412004 – 98411387

In this part of project we do these operations:

in proc.h file we add

```
int queue_number;            // Process Queue Number
```

to proc structure

then in proc.c file, first we change the ptable structure like this:

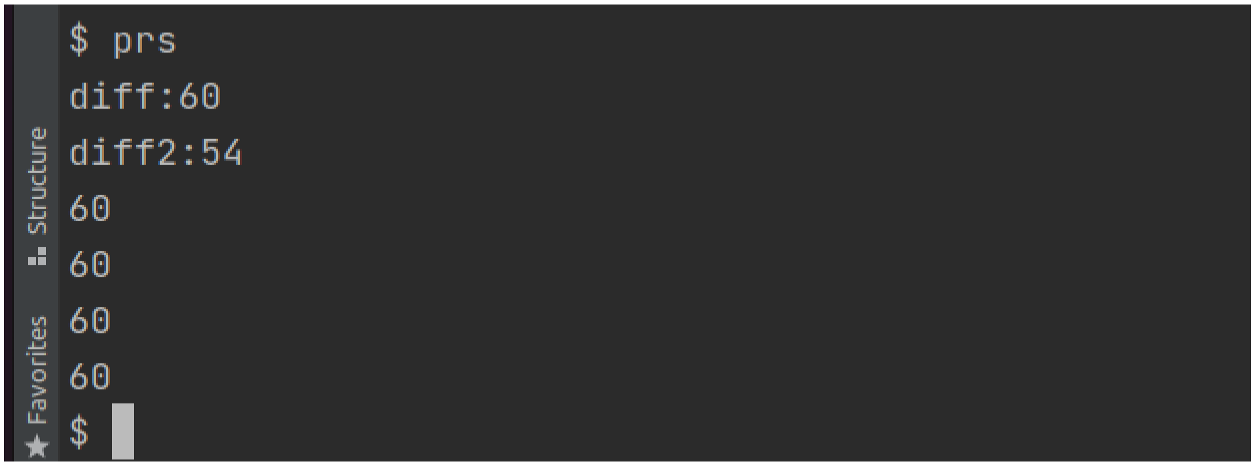
```
struct {
  struct spinlock lock;
  struct proc proc[NPROC];

  struct proc* procQueue[3][NPROC]; // array of 3 * 64 proc size
  int q1Count;
  int q2Count;
  int q3Count;
} ptable;
```

then we have to change the scheduler to use a multilevel queue as the new scheduling algorithm:

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

    for(;;)
    {
        // Enable interrupts on this processor.
        sti();

        // Loop over process table looking for process to run.
        acquire(&ptable.lock);
        struct proc *p = 0;

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

            ptable.q1Count=0;
            ptable.q2Count=0;
            ptable.q3Count=0;
        if(p->queue_number==1)
        {
            ptable.procQueue[0][ptable.q1Count]=p;
            ptable.q1Count++;
        }
        else if(p->queue_number==2)
        {
            ptable.procQueue[1][ptable.q2Count]=p;
            ptable.q2Count++;
        }
        else
        {
            ptable.procQueue[2][ptable.q3Count]=p;
            ptable.q3Count++;
        }
    }

    struct proc* processToRun=0;
    struct proc *p1 = 0;

    if(ptable.q1Count>0)
    {
        int i;
        int j;
        for(i = 0; i < ptable.q1Count; i++)
        {
            p=ptable.procQueue[0][i];
            if(p->state!=RUNNABLE)
                continue;
            processToRun=p;
            for(j = 0; j < ptable.q1Count; j++)
            {
                p1=ptable.procQueue[0][j];
                if(p1->state!=RUNNABLE)
                    continue;
                if(p1->rtime < processToRun->rtime)
                {
                    processToRun=p1;
                }
            }
            processToRun->queue_number=(processToRun->queue_number%20)+1;
            c->proc = processToRun;
            switchuvm(processToRun);
            processToRun->state = RUNNING;

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

            // Process is done running for now.
            // It should have changed its p->state before coming back.
            c->proc = 0;
        }
    }
    else
    {
        if(ptable.q2Count>0)
        {
            int i;
            int j;
            for(i = 0; i < ptable.q2Count; i++)
            {
                p=ptable.procQueue[1][i];
                if(p->state!=RUNNABLE)
                    continue;
                processToRun=p;
                for(j = 0; j < ptable.q2Count; j++)
                {
                    p1=ptable.procQueue[1][i];
                    if (p1->state != RUNNABLE)
                        continue;
                    if(p1->stime < processToRun->stime)
                    {
                        processToRun=p1;
                    }
                }
                processToRun->queue_number=(processToRun->queue_number%20)+1;
                c->proc = processToRun;
                switchuvm(processToRun);
                processToRun->state = RUNNING;

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

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

            }
        }
        else
        {
            int i;
            for(i = 0; i < ptable.q3Count; i++)
            {
                p=ptable.procQueue[2][i];
                if(p->state!=RUNNABLE)
                    continue;
                processToRun=p;

                processToRun->queue_number=(processToRun->queue_number%20)+1;
                c->proc = processToRun;
                switchuvm(processToRun);
```

```
                    processToRun->state = RUNNING;

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

                    // Process is done running for now.
                    // It should have changed its p->state before coming back.
                    c->proc = 0;
                }
            }
        }
        release(&ptable.lock);
    }
}
```

So now after compiling OS and typing prs as a user program:



```
$ prs
diff:60
diff2:54
60
60
60
60
$ 
```