LAB 2

In the following documentation we will explain the changes we have made in XV6 in chronological order.

1) In proc.h we created an “int priority” struct proc.
2) In the following files we created the syscall “setpriority”
    i) Syscall.c
    ii) Syscall.h
    iii) User.h
    iv) Proc.c
    v) Defs.h
    vi) sysproc.c

3) In sysproc.c under the function sys_setpriority we implemented the following:

```
118     argint(0, &pid);                                //passing firs
119     argptr(1,(void*)&status,sizeof(status));        //
120     argint(2, &options);
121
122     return waitpid(pid, status, options);
123 }
124
125 int
126 sys_setpriority(void)                                   //change p
127 {
128     int priority;
129     //cprintf("priority from sysproc: ");   //print priority
130
131     if(argint(0, &priority) < 0)
132         return -1;
133     return setpriority(priority);
134 }
135
```

4) in the proc.c we implemented in the function “scheduler” the following which loops through the pTable to find the highest priority and run the highest priority process.

```
489     sti();
490
491
492     int max = 31;//lab2
493     // Loop over process table looking for process to run.
494     acquire(&ptable.lock);
495
496     //lab2
497     //find the maximum priority
498
499     for ( p = ptable.proc; p < &ptable.proc[NPROC]; p++) //scan through table
500     {
501           //cprintf("find \n");
502           if ( p->priority < max && p->state == RUNNABLE) //make sure the maximum is always runnable
503           {
504                   max = p->priority; //set max to the the highest priortiy
505
506           }
507           //cprintf("find \n");
508         }
509
510         //cprintf("find %d \n", max);    ///why run forever
511
512         //lab2
513         //check highest priority
514     for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
515       if(/*p->priority != max ||*/ p->state != RUNNABLE ) //if it is not the max or not runnable    ://p-
516       {
517
518         continue;
519       }
520 //cprintf("the max is %d \n", max);
521       // Switch to chosen process.  It is the process's job
522       // to release ptable lock and then reacquire it
```

i) We initiate int max = 31 on line 492

ii) on line 502 we make a condition is p->priority is less than max we make it the new max

iii) on line 527 if p->priority is == max then we run the process

```
22      // to release ptable.lock and then reacquire it
23      // before jumping back to us.
24
25
26
27      if (p->priority == max)
28      {
29                  //p->priority++; //lower priority
30                  c->proc = p;
31                  //cprintf("we are now runnning process with priority : %d \n", p->priority);
32                  switchuvm(p);
33                  p->state = RUNNING;
34
35                  swtch(&(c->scheduler), p->context);
36                  switchkvm();
37                  c->proc = 0;
38                  // Process is done running for now.
39                  // It should have changed its p->state before coming back.
40
41      }
42      else
43      {
44             p->priority_counter--; //count down
45             if (p->priority_counter ==0)
46             {
47                    p->priority--; //increase priority
48                    p->priority_counter = 10;  //reset
49                }
50      }
51  }
52      release(&ptable.lock);
53
54
55    }
56  }
57
58  int
```

For the extra bonus we created a int priority_counter struct in proc.h.

This is to keep track of the counter for each time a process is waiting and each 10 time it waits the process's priority increases while the process that's running its priority decreases. The implementation can be seen in the above figure in line 544.