Chris Hung Huynh, SID: 862024281
Edwin Leon, SID: 862132870

Lab2 Report

In lab 2, we are asked to change the policy of a scheduler from a simple round-robin one to a priority scheduler. We chose to implement aging of priority, where we increase the priority of a process every time the process waits, and decrease its priority every time it runs. Here are the steps we took in order to do so:

1. Added a new field called “int priority” to the proc struct in the proc.h file.
2. Created a new getPrio and setPrio function signature to defs.h and user.h files.
3. Added the system calls “SYSCALL(getPrio)” and “SYSCALL(setPrio)” to the usys.S file.
4. Define the system calls “SYS_getPrio” and “SYS_setPrio” in the syscall.h file.
5. Added “extern int sys_getPrio(void)” and “extern int sys_setPrio(void)” to the syscall.c file.
6. Modified sysproc.c file so that the sys_getPrio and sys_setPrio functions would fetch an argument and return the priority.
7. Added the function “int getPrio(void)” and “int setPrio(int)” in the proc.c file, so that getPrio would return the priority of the process, and setPrio would modify the priority of the process.

```
int
sys_getPrio(void)
{
   return getPrio();
}

int
sys_setPrio(void)
{
   int prio;

   if(argint(0, &prio) < 0)
     return -1;
   return setPrio(prio);
}
```

```
int getPrio(void)
{
  struct proc *curproc = myproc();
  return curproc->priority;
}

int setPrio(int prio)
{
  struct proc *p = myproc();

  if((prio < 0) || (prio > 50)){
        return -1;
  }
  else{
        p->priority = prio;
        return 0;
  }
}
```

After adding our system calls to modify the priority of a process we started modifying the scheduler so that it would implement aging of priority. Basically, when a process would run its priority would decrease and when a process waited its priority would increase.

1. First we added two struct proc variables named “p2 (temporary process to determine the highest priority process)” and “p3 (highest priority process)”.
2. Then we looped through the processes and if it was RUNNABLE we set that process as the highest priority process (so far). If it was not RUNNABLE we would increase its priority.
3. After, we looped through the processes again and compared the priority of “p2” and “p3”. If the priority of “p2” was higher, then “p3” would be assigned “p2”.
4. After looping through all the processes “p3” would be the highest priority process and the process will run. Since the process will run, its priority will be decreased.

```
//New scheduler
void
scheduler(void)
{
  struct proc *p;
  struct cpu *c = mycpu();
  struct proc *p2; //Temp process to determine which process had the highest priority
  struct proc *p3; //Highest priority process
  c->proc = 0;

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

    // Loop over process table looking for process to run.
    acquire(&ptable.lock);
    for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
      if(p->state != RUNNABLE){
        if(p->priority > 0){ //If process does not run then increase its priority
                p->priority--;
        }
        continue;
      }

      p3 = p; //Assign p3 the first runnable process

      // Loop over process table looking for highest priority process to run
      for(p2 = ptable.proc; p2 < &ptable.proc[NPROC]; p2++){
        if(p2->state != RUNNABLE){
          continue;
        }
        //If the priority of p2 is higher than the priority of p3 then assign it to p3
        if(p2->priority < p3->priority){
          p3 = p2;
        }
      }
```

```
      // 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 = p3;
      switchuvm(p3);
      p3->state = RUNNING;

      //If the priority of the is less than 50 we decrease its priority
      if(p->priority < 50){
        p->priority++;
      }

      swtch(&(c->scheduler), p3->context);
      switchkvm();
      // Process is done running for now.
      // It should have changed its p->state before coming back.
      c->proc = 0;
    }
    release(&ptable.lock);
  }
}
```

Finally, we created a test program to test our new syscalls and scheduler. We initiated 5 processes and set their priorities different from one another (50, 40, 30, 20, and 10). We displayed the process ID's of each process to show that they are unique processes as well as their priority levels. Their priority levels will either increase or decrease depending on whether they wait or execute until all processes are finished executing.

```
iPXE (http://ipxe.org) 00:03.0 C980 PCI2.10 PnP PMM+1FF94780+1FED4780 C980



Booting from Hard Disk..xv6...
cpu1: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ ./test

 This program tests the correctness of aging of priority scheduler

Testing the scheduler

 This is child with PID: 4 and priority 50

 This child with PID: 4 exited with priority 5

 This is the parent: child with PID# 4 has exited

 This is child with PID: 5 and priority 40

 This child with PID: 5 exited with priority 1

 This is the parent: child with PID# 5 has exited

 This is child with PID: 6 and priority 30

 This child with PID: 6 exited with priority 22

 This is the parent: child with PID# 6 has exited

 This is child with PID: 7 and priority 20

 This child with PID: 7 exited with priority 1

 This is the parent: child with PID# 7 has exited

 This is child with PID: 8 and priority 10

 This child with PID: 8 exited with priority 1

 This is the parent: child with PID# 8 has exited
$
```