Kinjal Mugatwala
SID: 862014507
February 15, 2020

Lab 2 Report

Summary of Lab 2:

The purpose of this lab is to implement a priority scheduler. In the priority scheduler, I assign each process to a priority value and the process with the highest priority will be scheduled to run first. To do that, I added “int priority” in proc.h to the proc structure so that each process will have a priority. All processes were set to the default priority of 10 in the lab, however the newly built system call “setpriority(int priority)” allows the priority to change to the priority value mentioned in the system call’s function parameter. Then, to make sure the process with the highest priority got scheduled first, I made adjustments in the scheduler system and compare each process’ priority with other processes in the process table and make sure the process with the highest priority is found and chosen to run. However, with this method, we can face starvation as the process with the lowest priority will be waiting for a long amount of time. To fix that, I used the aging technique by increasing the priority when the process is waiting and decreasing the priority of the process that runs.

Lab 2 Output without Aging Testing:

```
$ lab2

 This program tests the correctness of your lab#2

  Step 2: testing the priority scheduler and setpriority(int priority)) systema call:

  Step 2: Assuming that the priorities range between range between 0 to 31

  Step 2: 0 is the highest priority. All processes have a default priority of 10

  Step 2: The parent processes will switch to priority 0

 child# 66 with priority 10 has finished!

 child# 65 with priority 20 has finished!

 child# 64 with priority 30 has finished!

 if processes with highest priority finished first then its correct
$
```

Lab 2 Output with Aging Testing:

```
$ lab2

 This program tests the correctness of your lab#2

  Step 2: testing the priority scheduler and setpriority(int priority)) systema call:

  Step 2: Assuming that the priorities range between range between 0 to 31

  Step 2: 0 is the highest priority. All processes have a default priority of 10

  Step 2: The parent processes will switch to priority 0

 child# 6 with priority 10 has finished!
Priority for child 6  running process: 20

 child# 5 with priority 20 has finished!
Priority for child 5  running process: 31

 child# 4 with priority 30 has finished!
Priority for child 4  running process: 42

 if processes with highest priority finished first then its correct
```

proc.h

```
  int priority;                // Priority value for each process (0 - 31) --> Lab 2
```

Part 1 - setpriority system call:

I created the setpriority system call by implementing in proc.c, user.h, defs.h, sysproc.c, syscall.h, syscall.c, and usys.S to define the system call. In proc.c, because the priority value is getting updated, we want to make sure that the process's priority is updated one at a time in order to avoid any interleaving problems and that is why I put a lock when the priority is updating. After the update, I release the lock so that other threads that are waiting can go and try to acquire the lock to update its priority.

proc.c:

```
int setpriority(int priority) {
   struct proc *curproc = myproc();
   acquire(&ptable.lock);
   curproc->priority = priority;
   release(&ptable.lock);

   return 0;
}
```

user.h

```
int setpriority(int priority);
```

defs.h

```
int             setpriority(int priority);
```

sysproc.c

```
int
sys_setpriority(void)
{
  int priority;
  if(argint(0, &priority) < 0)
    return 0;

  return setpriority(priority);
}
```

syscall.h

```
#define SYS_setpriority 23
```

syscall.c

```
extern int sys_setpriority(void);

[SYS_setpriority] sys_setpriority,
```

usys.S

```
SYSCALL(setpriority)
```

Part 2 - Implementing priority scheduling by setting highest priority to run first:
In allocproc in proc.c, I set the default priority to 10 as that is what asked from the lab 2 directions.
I made many adjustments in the proc.c file. In the outermost for loop, I created a proc structure, currHighestPriority and I set it to a default value of 0. The currHighestPriority process will be assigned to the process p that is currently runnable as now we want to compare with the rest of the runnable processes in the process table to make sure that process has the highest priority. If the currHighestPriority does not have the highest priority, then we want to assign currHighestPriority to the process with the higher priority we were comparing to. We continue to do this throughout the innermost loop and keep updating currHighestPriority until we get through the entire process table. That final process with the current highest priority will be assigned to run.

I decided to implement the first item out of the three items and that was to implement aging to avoid starvation. So, to do that, I incremented the priority of the process by decreasing the integer value of the priority variable that is waiting (the state is not runnable) in order to give it more priority so that it will increase the chance of that process to get run. I decremented the priority that is running (p->state = RUNNING) by increasing the integer value of the priority variable because the process just ran and got its turn and we now need to let other processes run.

proc.c:

```
void
scheduler(void)
{
  struct proc *p; //used for iteration through process table
  struct proc *p_iter; //used for iteration through process table
  struct cpu *c = mycpu();
  c->proc = 0;
  for(;;){
    // Enable interrupts on this processor.
    sti();
    struct proc *currHighestPriority = 0;
    // 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)
        continue;
      currHighestPriority = p;

      //need to iterate through entire ptable again because we have to compare the current process's priority with the rest of ptable
      for(p_iter = ptable.proc; p_iter < &ptable.proc[NPROC]; p_iter++) {
        if(p_iter->state != RUNNABLE) {
          p_iter->priority--; //increment priority by decreasing the integer value  here, because the p_iter process is not running, it is WAITING

          continue;
        }

        if(p_iter->priority < currHighestPriority->priority){
         //we iterated through the process table and we found a process with a higher priority and we want to schedule that first
         currHighestPriority = p_iter;
        }
       }

      p = currHighestPriority; //setting the process that runs to the process with the highest priority
      // 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;
      p->priority++; //decrement the priority here because the process set to RUN now and the process got its turn
      swtch(&(c->scheduler), p->context);
      switchkvm();
      // Process is done running for now.
      // It should have changed its p->state before coming back.
      c->proc = 0;
    }
    release(&ptable.lock);

  }
}
```

allocproc system call adjustment:

```
  p->priority = 10; //added for lab 2 for priority scheduling, need to set a default value
```

Testing for Aging:

I created a system call getpriority in order to be able to refer to it in lab2.c just like getpid. I created the getpriority system call by implementing in user.h, sysproc.c, syscall.h, syscall.c, and usys.S to define the system call. In lab2.c, I added a print statement to check the priority of the process after it runs. If the priority integer value increases and gives a higher value, we know that the priority of that process decreased and that helps processes that had lower priority before get the chance to run.

user.h

```
int getpriority(void);
```

sysproc.c

```
int
sys_getpriority(void)
{
  return myproc()->priority;
}
```

syscall.h

```
#define SYS_getpriority 24
```

syscall.c

```
extern int sys_getpriority(void);
```

```
[SYS_getpriority] sys_getpriority,
```

usys.S

```
SYSCALL(getpriority)
```

lab2.c

```
printf(1, "Priority for child %d  running process: %d \n", getpid(), getpriority());
```