Anthony Mejia - SID: **861152139**
Christian Campos - SID: **862080812**

Lab 2: Scheduler

Implement Priority Scheduling:

The purpose of lab 2 is to implement a scheduler from a simple round robin to a priority scheduler. We added a new system call that changes the priority of a process in which process with the highest priority (the one with the lowest value) gets scheduled first.
The system call we added: int setPriority(int p) and void getPriority(void).

1) Int setPriority (int priority): The current process is obtained by using myproc(). The process priority is updated with the passing argument.
2) Int getPriority(): To print updated priorities , we created a getter system call

We also have to modified the following files: syscall.h, syscall.c, defs.h, user.h,sysproc.c,usys.S,proc.h and proc.c

Adjust priority (avoid starvation):

In order to avoid starvation , we implement our aging of priority by decreasing priority of processes due to high priority and increased priority of processes which to run.
We do this by modifying the proc.c scheduler function; scheduler(void). The change in the scheduler function from simple round robin to a priority scheduler.

Tracking Scheduling Performance:

We created a turnaround time for each process to print when the process exists.To track the scheduling performance of each process, we modified the proc.c exit function; exit(void). The change is needed to print the turnaround time when it exists.

The changes in our code are listed below.

syscall.h:

```
#define SYS_setPriority 24
#define SYS_getPriority 25
```

syscall.c:

```
// Added syscall to set and get priority of a process - assignment 2
extern int sys_setPriority(void);
extern int sys_getPriority(void);
```

```
[SYS_setPriority]   sys_setPriority,
[SYS_getPriority]   sys_getPriority,
};
```

usys.S:

```
SYSCALL(setPriority)
SYSCALL(getPriority)
```

defs.h:

```
int             setPriority(int); // Add set Priority function - assignment 2
int             getPriority(void);
```

sysproc.c:

```c
int sys_setPriority(void)
{
    int priority;
    if (argint(0,&priority)<0){
        return -1;
    }
    return setPriority(priority);
}

int sys_getPriority(void)
{
    return getPriority();
}
```

user.h:

```c
// Add user syscall for setPriority - assignment 2
int setPriority(int);
int getPriority(void);
```

proc.h:

```
// Per-process state
struct proc {
  uint sz;                     // Size of process memory (bytes)
  pde_t* pgdir;                // Page table
  char *kstack;                // Bottom of kernel stack for this proces
  enum procstate state;        // Process state
  int pid;                     // Process ID
  struct proc *parent;         // Parent process
  struct trapframe *tf;        // Trap frame for current syscall
  struct context *context;     // swtch() here to run process
  void *chan;                  // If non-zero, sleeping on chan
  int killed;                  // If non-zero, have been killed
  struct file *ofile[NOFILE];  // Open files
  struct inode *cwd;           // Current directory
  char name[16];               // Process name (debugging)
  int exitstatus;              // Exit Status - assignment 1
  int priority;                // Priority value - assignment 2
  int start_time;
};
```

proc.c

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

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

    // Loop over process table looking for process to run.
    low_priority = 1000;
    acquire(&ptable.lock);
      for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
          if (p->state == RUNNABLE && p->priority < low_priority) {
              low_priority = p->priority;
          }
      }

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

      if(p->priority != low_priority){
          if (p->priority > 0) {
              p->priority--;
          }
          continue;
      }

      // 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++;
```

```
      p->state = RUNNING;
      p->priority++;

      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);
  }
}
```

```c
//get and set priority of process - assignment
int setPriority(int priority)
{
    struct proc *p = myproc();
    p->priority = priority;
    return 0;
}

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

```c
  curproc->cwd = 0;

  end_time = ticks;
  cprintf("\n turnaround time is %d\n", end_time - curproc->start_time);
```

Lab 2 Test Code:

```c
#include "types.h"
#include "user.h"

int main(int argc, char *argv[])
{
    int PScheduler(void);
    printf(1, "\n This program tests the correctness of your lab#2\n");
    PScheduler();
    return 0;
}


int PScheduler(void){
    // use this part to test the priority scheduler. Assuming that the priorities range between range between 0 to 31
    // 0 is the highest priority and 31 is the lowest priority.
    int pid;
    int i,j,k;
    int priorityArr[] = {30, 15, 25, 0, 9};

    printf(1, "\n  Testing the priority scheduler and setPriority(int priority) system call:\n");
    printf(1, "\n  Assuming that the priorities range between range between 0 to 31\n");
    printf(1, "\n  0 is the highest priority. All processes have a default priority of 10\n");
    printf(1, "\n  The parent processes will switch to priority 0\n");
    setPriority(0);

    for(i = 0; i < 5; i++) {
        pid = fork();
        if (pid > 0 ) {
            continue;
        } else if ( pid == 0) {
            setPriority(priorityArr[i]);
            printf(1, "\n child# %d has priority %d before starting its work", getpid(), getPriority());
            for (j=0;j<50000;j++) {
                for(k=0;k<1000;k++) {
                    asm("nop");
                }
            }
            printf(1, "\n child# %d has priority %d after finishing its work", getpid(), getPriority());
            printf(1, "\n child# %d with original priority %d has finished! \n", getpid(), priorityArr[i]);
```

```c
            exit();
        } else {
            printf(2," \n Error \n");
        }
    }

    if(pid > 0) {
        for(i = 0; i < 5; i++) {
            wait(0);
        }
        printf(1,"\n if processes with highest priority finished first then its correct \n");
    }
    exit();
}
```

To run our test file; lab2.c

Use make clean qemu-nox inside the xv6 directory.

Enter Lab2 to run our test code program.

However, a weird format issue occurred for the test output as shown below.

```
vagrant@ubuntu-xenial:~/CS153/Lab2$ make clean qemu-nox
```

```
173524 bytes (174 kB, 169 KiB) copied, 0.00142049 s, 122 MB/s
qemu-system-i386 -nographic -drive file=fs.img,index=1,media=disk,format=raw -drive file=xv6.img,index=0,
xv6...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ lab2
```

Test output:

```
$ lab2

 This program tests the correctness of your lab#2

  Testing the priority scheduler and setPriority(int priority) system call:

  Assuming that the priorities range between range between 0 to 31

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

  The parent processes will switch to priority 0

 child# 4 has priority 3
 child# 5 has priority 15 before starting its work
 child# 6 has priority 25 before starting its work
 child# 7 has priority 0 before starting its work
 child# 8 has priority 9 befo
 childre starting its work# 7 has priority 7 after finishing its work
 child# 7 with original priority 0 has finished!

 turnaround time is 19
0 before starting its work
 child# 8 has priority 6 after finishing its work
 child#
 child# 5 has priority 6 after finishing its8 with original priority 9 has finished!

 turnaround time is 43
 work
 child# 5 with original priority 15 has finished!
```

```
turnaround time is 43
work
child# 5 with original priority 15 has finished!

turnaround time is 52

child# 6 has priority 5 after finishing its work
child# 6 with original priority 25 has finished!

turnaround time is 58

child# 4 has priority 9 after finishing its work
child# 4 with original priority 30 has finished!

turnaround time is 62

if processes with highest priority finished first then its correct

turnaround time is 72
```