

Name: ANUJA PATIL

Student ID: 862123752

1) Add a system call to change priority of a process.

Approach: Added implementation for setPriority(int priority). To make the required changes, I changed some files. Following are the files changed and a brief summary of the changes done in each:

a) syscall.h

Defined a system call and associated it with a number:

#define SYS_setPriority 24

b) syscall.c

Added the function prototype for setPriority() – **extern int sys_setPriority(void);**

Added a pointer to the system call in the array of function pointers:

```
static int(*syscalls[]) (void) = {  
    .  
    .  
    .  
    [SYS_setPriority] sys_setPriority,  
}
```

c) usys.S

In order for a user program to call the system call, an interface was added:

SYS_CALL(setPriority)

d) sysproc.c

Added the handler for setPriority system call to update priority for process:

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

e) defs.h

Added forward declaration of system call setPriority() in “proc.c” section:

int setPriority(int);

f) user.h

Added user level definition of setPriority() under system calls:

int setPriority(int);

g) proc.h

Added a field ‘priority’ in the proc struct which stores the state of a process:

```

struct proc {
    .
    .
    int priority;
}

```

h) proc.c

This contains the actual implementation of the setPriority(). Current process is obtained using myproc(). Then, that process' priority is updated with the argument passed.

```

int setPriority(int p) {
    struct proc *curr = myproc();
    curr->priority = p;
    return 0;
}

```

2) Change scheduler from simple round robin to a priority scheduler. Implement aging of priority to avoid starvation.

Approach: The default priority of a process should be 10. The range of priority is between 0 to 31 both inclusive. When scheduling from the ready list, always the highest priority thread/process should be scheduled first. If a process waits, its priority increases. If a process runs, its priority decreases. For this, I changed two functions in **proc.c** file:

a) Set the default priority of process:

```

allocproc() {
    .
    .
    p->priority = 10;
    return p;
}

```

b) Changed the scheduler from Round Robin to priority scheduler with "aging":

```

scheduler() {
    .
    .
    for(;;) {
        sti();

        acquire(&ptable.lock);
        struct proc *nextProc = ptable.proc;
        int highestPriority = nextProc->priority;

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

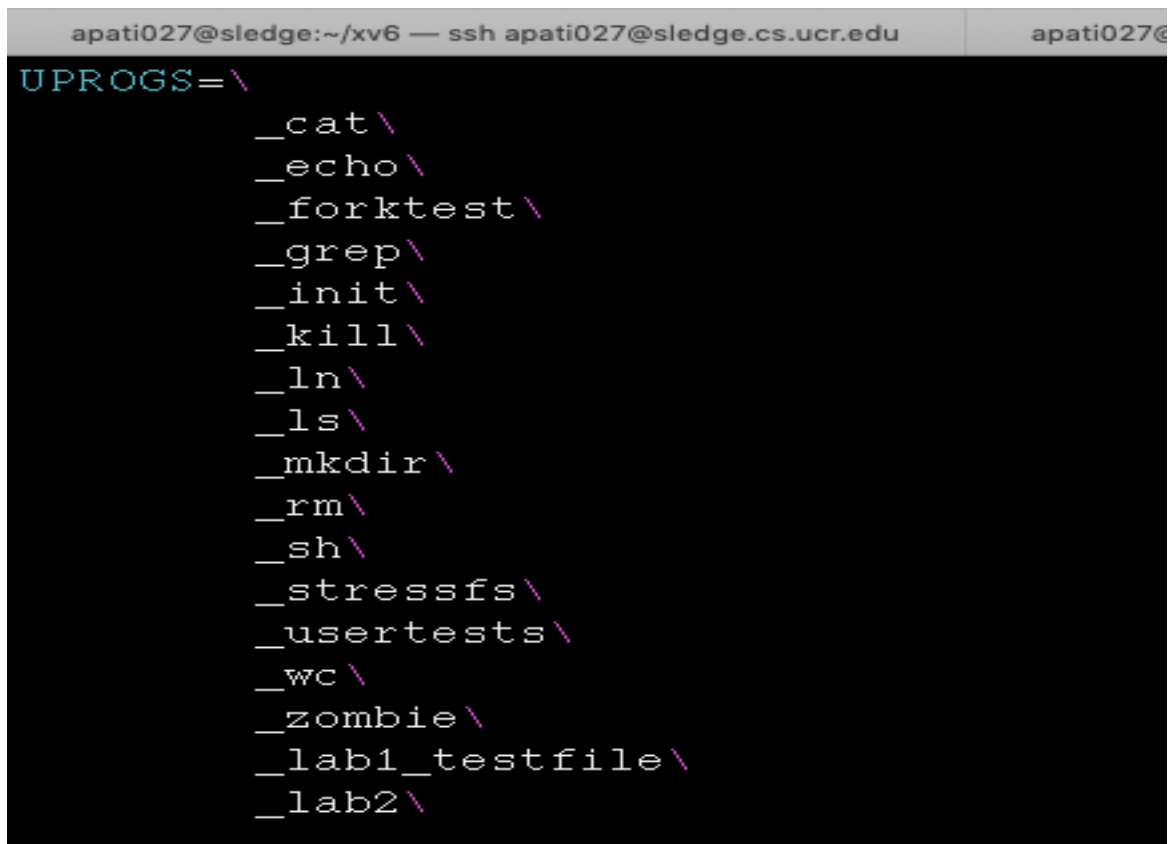
```

```

        if(p->priority < highestPriority) {
            highestPriority = p->priority;
            nextProc->priority -= 1;
            nextProc = p;
        } else {
            if(p->priority > 0) p->priority -= 1;
        }
    }
    if(nextProc->priority < 31) nextProc->priority += 1;
    c->proc = nextProc;
    switchvmm(nextProc);
    .
    .
    switchkvm();
    c->proc = 0;
    release(&ptable.lock);
}
}

```

3) Modified Makefile for including lab2 under UPROGS and changed the number of CPUS to 1.



The image shows a terminal window with a dark background. The title bar at the top indicates the user is 'apati027@sledge' and the session is an 'ssh' connection to 'apati027@sledge.cs.ucr.edu'. The terminal content shows a list of programs under the variable 'UPROGS'. The list includes: _cat, _echo, _forktest, _grep, _init, _kill, _ln, _ls, _mkdir, _rm, _sh, _stressfs, _usertests, _wc, _zombie, _lab1_testfile, and _lab2. Each item is preceded by a backslash character.

```

apati027@sledge:~/xv6 — ssh apati027@sledge.cs.ucr.edu  apati027@
UPROGS=\
    _cat \
    _echo \
    _forktest \
    _grep \
    _init \
    _kill \
    _ln \
    _ls \
    _mkdir \
    _rm \
    _sh \
    _stressfs \
    _usertests \
    _wc \
    _zombie \
    _lab1_testfile \
    _lab2 \

```

```
apati027@sledge:~/xv6 — ssh apati027@sledge.cs.ucr.edu apati
ifnndef CPUS
CPUS := 1
endif
```

4) Modified test file (lab2.c) for testing aging of priority with priority scheduler.

Approach: Created an array of priorities “priorityArr”. Put print statements before and after the nested for loops for child processes with initially set and updated priorities. To print updated priorities, I created another system call “**getPriority()**” and changed in the same files which are needed to create a system call.

5) To make and run the lab2.c: (We are inside xv6 folder)

> make clean

> make qemu-nox

This will open qemu console. Then run:

\$ lab2

Following are the outputs for 5 processes (created array “priorityArr” in **lab2.c** which contains priorities – 30, 15, 25, 0, and 9):

```
Booting from Hard Disk..xv6...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ lab2

This program tests the correctness of your lab#2

Step 2: testing the priority scheduler and setpriority(int priority)) system 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# 4 has priority 30 before starting its work
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 before starting its work
child# 7 has priority 3 after finishing its work
child# 7 with original priority 0 has finished!

child# 4 has priority 1 after finishing its work
child# 4 with original priority 30 has finished!

child# 5 has priority 1 after finishing its work
child# 5 with original priority 15 has finished!

child# 6 has priority 1 after finishing its work
child# 6 with original priority 25 has finished!

child# 8 has priority 1 after finishing its work
child# 8 with original priority 9 has finished!

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

Child processes 4, 5, 6, 7, and 8 are created with priorities 30, 15, 25, 0 and 9. We can see that child 7 has executed first and hence its priority increased to 3 (assuming the middle values have run quickly) and then it has finished execution. We can see that rest processes' priorities are increased to 1 and it stopped going lower than that as I have that condition put in the code that does not set priority lesser than 0. Since now their priorities are same, any order execution has happened.

I have another screenshot of outputs with same priorityArr as above. Just to slow down the process, I have also put print statements between the nested loops to indicate “in the middle of work”. I have reduced the number of iterations too so that program can run quickly.

```

/Users/anjapati@apat027@sledge:~ -- ssh apati027@sledge.cs.ucr.edu
apat027@sledge:~/xv6 -- ssh apati027@sledge.cs.ucr.edu
+
Booting from Hard Disk..xv6...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ lab2

This program tests the correctness of your lab#2

Step 2: testing the priority scheduler and setpriority(int priority)) system 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# 4 has priority 30 before starting its work
child 4 priority is 23 during work
child 4 priority is 23 during work
child 4 priority is 23 during work
child 4 priority is 20 during work
child 4 priority is 20 during work
child 4 priority is 20 during wo
child# 7 has priority 0 before starting its work
child 7 priority is 2 during work
child 7 priority is 2 during work
child 7 priority is 3 during work
child 7 priority is 3 during work
child 7 priority is 3 during work
child 7 priority is 3 during work
child 7 priority is 4 during work
child 7 priority is 4 during work
child 7 priority is 4 during work
child 7 priority is 5 during work
child 7 priority is 5 during work
child 7 priority is 5 during work
child 7 priority is 5 during work
child 7 priority is 6 during work
child 7 priority is 6 during work
```

We can see here that child 7's priority is increasing as it was having the highest priority and it was executing where as child 4' priority is decreasing.

```
/Users/anjapatil — apati027@sledge:~ — ssh apati027@sledge.cs.ucr.edu
apati027@sledge:~/xv6 — ssh apati027@sledge.cs.ucr.edu

child 7 priority is 5 during work
child 7 priority is 6 during work
child 7 priority is 6 during work
child 7 priority is
child# 8 has priority 9 before starting its work
child 8 priority is 5 during work
child 8 priority is 5 during wo 6 during work
child 7 priority is 6 during work
child 7 priority is 6 during work
child 7 priority is 6 during work
rk
child 8 priority is 5 during work
child 8 priority is 5 during work
child 8 priority is 5 during work
child child 7 priority is 6 during work
child 7 priority is 6 during work
child 7 priority is 6 during work
child 7 priority is 6 during work
child 7 priority is 5 during work
child 7 priority is 5 during work
child 7 priority is 5 during work
child# 5 has priority 15 before starting its work
child 5 priority is 6 during work
child 5 priority is 6 during work8 priority is 5 during work
child 8 priority is 4 during work
child 8 priority is 4 during work
child 8 priori
child 5 priority is 6 during work
child 5 priority is 4 during work
child 5 priority is 4ty is 4 during work
child 8 priority is 3 during work
child 8 priority is 3 during work
child 8 priork
child 4 priority is 3 during work
child 4 priority is 3 during work
child 4 priority is 3 during work
child
child# 6 has priority 25 before starting its work
child 6 priority is 2 during work
```

At some point, child 8 with priority 9 was not the highest and hence its priority increased to 5. Similarly child 5 having priority 15, was not run initially, so its priority increased to 6.

apati027@sledge:~/xv6 — ssh apati027@sledge.cs.ucr.edu

Child 4's priority is continuously at 2 because I have the lower limit condition on priority.

Skipping some screenshots and attaching the last few lines:

```
child 8 priority is 2 during work
child 8 priority is 2 during work
child 8 priority is 2 during work
child 8 priority is 2 during work
child 8 priority is 2 during work
child 8 priority is 2 during work
child 8 priority is 2 during work
child 8 priority is 2 during work
child 8 priority is 2 during work
child# 8 has priority 2 after finishing its work
child# 8 with original priority 9 has finished!

if processes with highest priority finished first then its correct
$ qemu: terminating on signal 15 from pid 44815
[apati027@sledge xv6]$
```

Following are some screenshots of modified code:

scheduler() in proc.c:

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

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

        // Loop over process table finding the highest priority ready process.
        // Implementing aging of priority: If process waits, its priority is
        // increased. If process runs, its priority is decreased.
        acquire(&ptable.lock);
        struct proc *nextProc = ptable.proc;
        int highestPriority = nextProc->priority;

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

            if(p->priority < highestPriority) {
                highestPriority = p->priority;
                // Increase priority of nextProc as it will now be waiting.
                nextProc->priority = nextProc->priority - 1;
                nextProc = p;
            } else {
                // this process is going to wait, so increase its priority if it
                // is greater than 0. Keep it unchanged for priority 0 (min priority).
                if(p->priority > 0) p->priority--;
            }
        }

        // Run the nextProc process which has the highest priority. Since this
        // process runs, decrease its priority if it is less than 31 (max
        // priority). First check if nextProc is not null, otherwise just continue.
        if(nextProc->priority < 31) {
```

```

// Run the nextProc process which has the highest priority. Since this
// process runs, decrease its priority if it is less than 31 (max
// priority). First check if nextProc is not null, otherwise just continue.
if(nextProc->priority < 31) {
    nextProc->priority = nextProc->priority + 1;
}
// Switch to the chosen process (nextProc). It is the process's job
// to release ptable.lock and then reacquire it before jumping
// back to us.
c->proc = nextProc;
switchvm(nextProc);
nextProc->state = RUNNING;

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

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

```

set and get priority methods in proc.c:

```

int
setPriority(int priority)
{
    struct proc *curproc = myproc();
    curproc->priority = priority;
    return 0;
}

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

```

Summary for lab2:

The lab2 was about learning to implement a priority scheduler. For this, we needed to create a system call for setting and getting priority of process, then modify the logic of scheduler. Also implemented aging of priority – when process waits, its priority should increase, and when it runs, its priority should decrease. This is done to avoid starvation.