



KALINGA INSTITUTE OF INDUSTRIAL TECHNOLOGY (KIIT)

Deemed to be University U/S 3 of UGC Act, 1956

OS LAB ASSIGNMENT-2

Question:-

Implement Round Robin and Priority Scheduling technique both in the single scheduler of xv6. The default scheduler of xv6 is round-robin. The implementation of priority scheduling link is as given below(You may follow other links). You go through both the scheduler and try to implement by combining both in one.

<https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=video&cd=&cad=rja&uact=8&ved=2ahUKEwiMs5al4eX2AhVGr1YBHxf7ALsQtWj6BAGGEAI&url=https%3A%2F%2Fwww.youtube.com%2Fwatch%3Fv%3DDZ0-GMtOtEc&usg=AOvVaw0muVqafo-VtIsq8D0YBW-T>

<https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=video&cd=&cad=rja&uact=8&ved=2ahUKEwiMs5al4eX2AhVGr1YBHxf7ALsQtWj6BAGLEAI&url=https%3A%2F%2Fwww.youtube.com%2Fwatch%3Fv%3De-7tDx26zTk&usg=AOvVaw0uRsBCertRAG6egRn82-iT>

>> Create two system call ps and chpr for displaying processes context at that time and for changing priority respectively.

STEPS FOR SETTING UP SYSTEM CALL:-

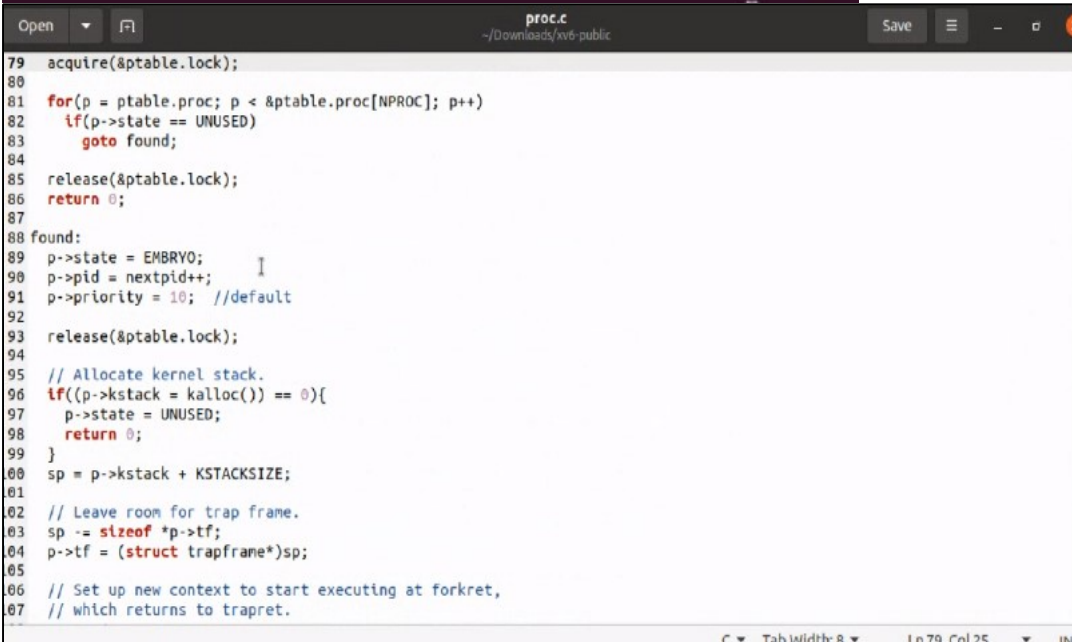
Step-1:

```
aditi@aditi:~/Downloads$ cd xv6-public
aditi@aditi:~/Downloads/xv6-public$ gedit proc.h
aditi@aditi:~/Downloads/xv6-public$
```

```
31 uint ebp;
32 uint elp;
33 };
34
35 enum procstate { UNUSED, EMBRYO, SLEEPING, RUNNABLE, RUNNING, ZOMBIE };
36
37 // Per-process state
38 struct proc {
39     uint sz; // Size of process memory (bytes)
40     pde_t* pgdir; // Page table
41     char *kstack; // Bottom of kernel stack for this process
42     enum procstate state; // Process state
43     int pid; // Process ID
44     struct proc *parent; // Parent process
45     struct trapframe *tf; // Trap frame for current syscall
46     struct context *context; // switch() here to run process
47     void *chan; // If non-zero, sleeping on chan
48     int killed; // If non-zero, have been killed
49     struct file *ofile[NOFILE]; // Open files
50     struct inode *cwd; // Current directory
51     char name[16]; // Process name (debugging)
52     int priority; // priority process
53 };
54
55 // Process memory is laid out contiguously, low addresses first:
56 // text
57 // original data and bss
58 // fixed-size stack
59 // expandable heap
```

Step-2:

```
aditi@aditi:~/Downloads/xv6-public$ gedit proc.h
aditi@aditi:~/Downloads/xv6-public$ gedit proc.c
aditi@aditi:~/Downloads/xv6-public$
```



```
79 acquire(&table.lock);
80
81 for(p = ptable.proc; p < &table.proc[NPROC]; p++)
82     if(p->state == UNUSED)
83         goto found;
84
85 release(&table.lock);
86 return 0;
87
88 found:
89 p->state = EMBRYO;
90 p->pid = nextpid++;
91 p->priority = 10; //default
92
93 release(&table.lock);
94
95 // Allocate kernel stack.
96 if((p->kstack = kalloc()) == 0){
97     p->state = UNUSED;
98     return 0;
99 }
100 sp = p->kstack + KSTACKSIZE;
101
102 // Leave room for trap frame.
103 sp -= sizeof *p->tf;
104 p->tf = (struct trapframe*)sp;
105
106 // Set up new context to start executing at forkret,
107 // which returns to trapret.
```

Step-3:

```
aditi@aditi:~/Downloads/xv6-public$ gedit exec.c
```



```
76 if(copyout(pgdir, sp, argv[argc], strlen(argv[argc]) + 1) < 0)
77     goto bad;
78 ustack[3+argc] = sp;
79 }
80 ustack[3+argc] = 0;
81
82 ustack[0] = 0xffffffff; // fake return PC
83 ustack[1] = argc;
84 ustack[2] = sp - (argc+1)*4; // argv pointer
85
86 sp -= (3+argc+1) * 4;
87 if(copyout(pgdir, sp, ustack, (3+argc+1)*4) < 0)
88     goto bad;
89
90 // Save program name for debugging.
91 for(last=s=path; *s; s++)
92     if(*s == '/')
93         last = s+1;
94 safestrcpy(curproc->name, last, sizeof(curproc->name));
95
96 // Commit to the user image.
97 oldpgdir = curproc->pgdir;
98 curproc->pgdir = pgdir;
99 curproc->sz = sz;
100 curproc->tf->eip = elf.entry; // main
101 curproc->tf->esp = sp;
102 curproc->priority = 3;
103 switchvm(curproc);
104 freevm(oldpgdir);
105 return 0;
106
```

Step-4:

```
aditi@aditi:~/Downloads/xv6-public$ gedit syscall.h
```

```
Open  ▾  📄  syscall.h  ~/Downloads/xv6-public

1 // System call numbers
2 #define SYS_fork    1
3 #define SYS_exit    2
4 #define SYS_wait    3
5 #define SYS_pipe    4
6 #define SYS_read    5
7 #define SYS_kill    6
8 #define SYS_exec    7
9 #define SYS_fstat   8
10 #define SYS_chdir   9
11 #define SYS_dup     10
12 #define SYS_getpid  11
13 #define SYS_sbrk    12
14 #define SYS_sleep   13
15 #define SYS_uptime   14
16 #define SYS_open    15
17 #define SYS_write   16
18 #define SYS_mknod   17
19 #define SYS_unlink  18
20 #define SYS_link    19
21 #define SYS_mkdir   20
22 #define SYS_close   21
23 #define SYS_ps      22
24 #define SYS_chpr    23
```

Step-5:

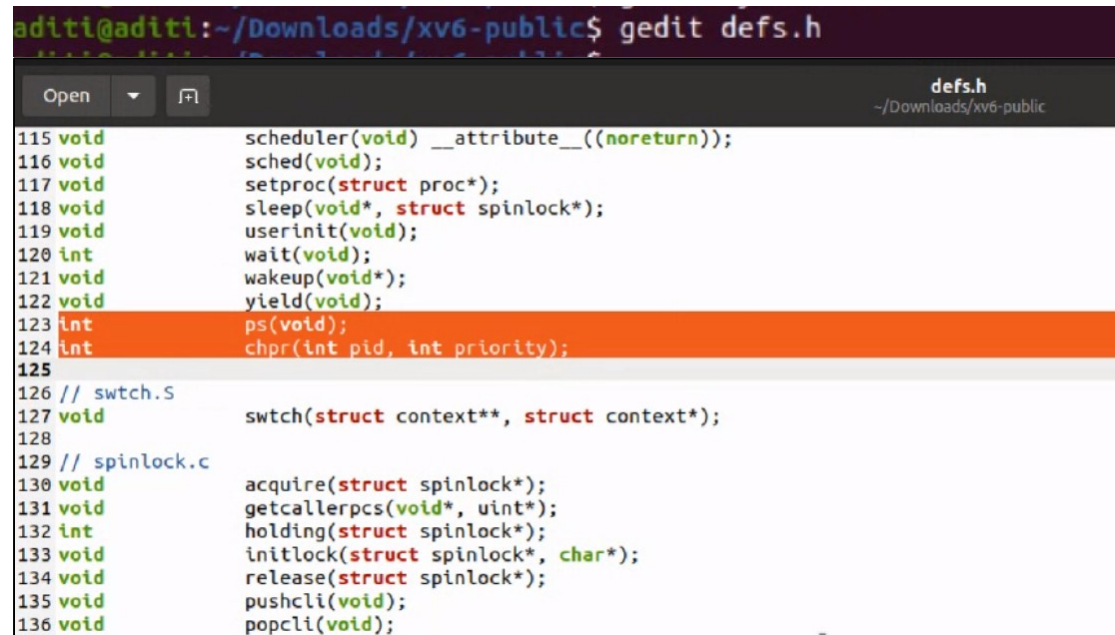
```
aditi@aditi:~/Downloads/xv6-public$ gedit syscall.c

Open  ▾  📄  syscall.c  ~/Downloads/xv6-public

86 extern int sys_close(void);
87 extern int sys_dup(void);
88 extern int sys_exec(void);
89 extern int sys_exit(void);
90 extern int sys_fork(void);
91 extern int sys_fstat(void);
92 extern int sys_getpid(void);
93 extern int sys_kill(void);
94 extern int sys_link(void);
95 extern int sys_mkdir(void);
96 extern int sys_mknod(void);
97 extern int sys_open(void);
98 extern int sys_pipe(void);
99 extern int sys_read(void);
100 extern int sys_sbrk(void);
101 extern int sys_sleep(void);
102 extern int sys_unlink(void);
103 extern int sys_wait(void);
104 extern int sys_write(void);
105 extern int sys_uptime(void);
106 extern int sys_ps(void);
107 extern int sys_chpr(void);
108
109 static int (*syscalls[])(void) = {
110 [SYS_fork]    sys_fork,
111 [SYS_exit]    sys_exit,
112 [SYS_wait]    sys_wait,
113 [SYS_pipe]    sys_pipe,
114 [SYS_read]    sys_read,
115 [SYS_kill]    sys_kill,
116 [SYS_exec]    sys_exec,
117 [SYS_fstat]   sys_fstat,
118 [SYS_chdir]   sys_chdir,
119 [SYS_dup]     sys_dup,
120 [SYS_getpid]  sys_getpid,
121 [SYS_sbrk]    sys_sbrk,
122 [SYS_sleep]   sys_sleep,
123 [SYS_uptime]  sys_uptime,
124 [SYS_open]    sys_open,
125 [SYS_write]   sys_write,
126 [SYS_mknod]   sys_mknod,
127 [SYS_unlink]  sys_unlink,
128 [SYS_link]    sys_link,
129 [SYS_mkdir]   sys_mkdir,
130 [SYS_close]   sys_close,
131 [SYS_ps]      sys_ps,
132 [SYS_chpr]    sys_chpr,
133 };
134
```

Step-6:

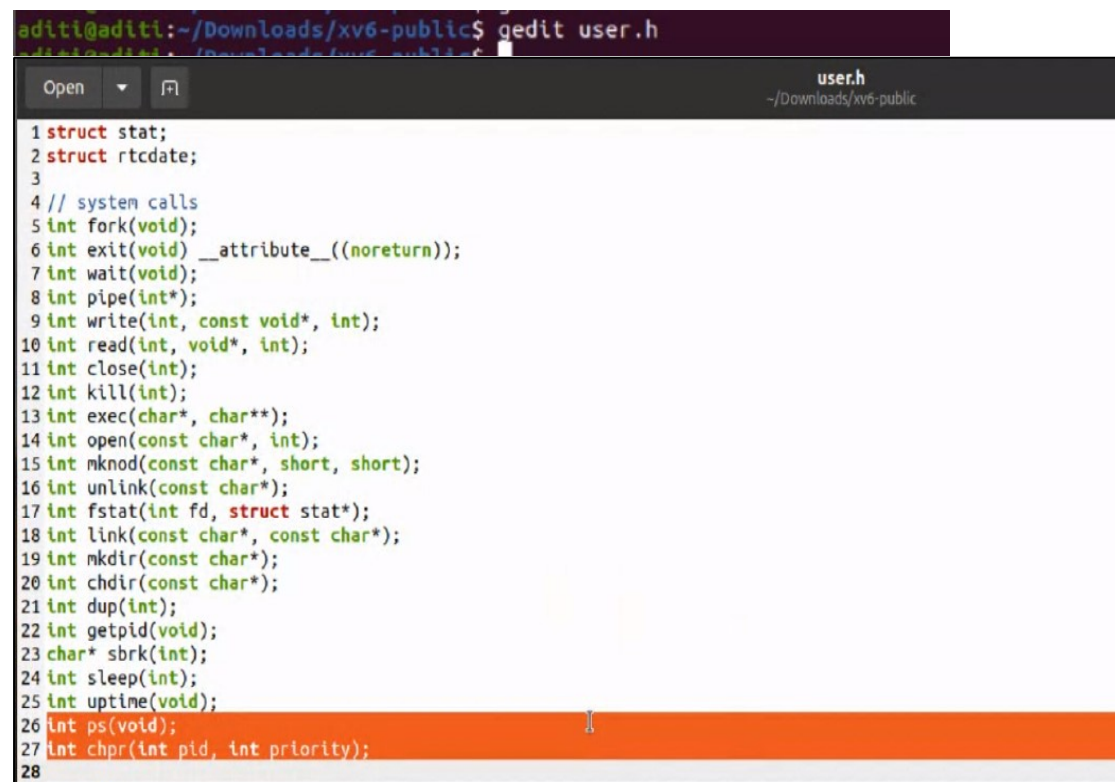
```
aditi@aditi:~/Downloads/xv6-public$ gedit defs.h
```



```
115 void scheduler(void) __attribute__((noreturn));
116 void sched(void);
117 void setproc(struct proc*);
118 void sleep(void*, struct spinlock*);
119 void userinit(void);
120 int wait(void);
121 void wakeup(void*);
122 void yield(void);
123 int ps(void);
124 int chpr(int pid, int priority);
125
126 // swtch.S
127 void swtch(struct context**, struct context*);
128
129 // spinlock.c
130 void acquire(struct spinlock*);
131 void getcallerpcs(void*, uint*);
132 int holding(struct spinlock*);
133 void initlock(struct spinlock*, char*);
134 void release(struct spinlock*);
135 void pushcli(void);
136 void popcli(void);
```

Step-7:

```
aditi@aditi:~/Downloads/xv6-public$ gedit user.h
```



```
1 struct stat;
2 struct rtcdate;
3
4 // system calls
5 int fork(void);
6 int exit(void) __attribute__((noreturn));
7 int wait(void);
8 int pipe(int*);
9 int write(int, const void*, int);
10 int read(int, void*, int);
11 int close(int);
12 int kill(int);
13 int exec(char*, char**);
14 int open(const char*, int);
15 int mknod(const char*, short, short);
16 int unlink(const char*);
17 int fstat(int fd, struct stat*);
18 int link(const char*, const char*);
19 int mkdir(const char*);
20 int chdir(const char*);
21 int dup(int);
22 int getpid(void);
23 char* sbrk(int);
24 int sleep(int);
25 int uptime(void);
26 int ps(void);
27 int chpr(int pid, int priority);
28
```


Step-8:

```
aditi@aditi:~/Downloads/xv6-public$ gedit proc.c

proc.c
~/Downloads/xv6-public

533     cprintf("\n");
534 }
535 }
536
537 int
538 ps()
539 {
540     struct proc *p;
541     //Enables interrupts on this processor.
542     sti();
543
544     //Loop over process table looking for process with pid.
545     acquire(&ptable.lock);
546     cprintf("name \t pid \t state \t priority \n");
547     for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
548         if(p->state == SLEEPING)
549             cprintf("%s \t %d \t SLEEPING \t %d \n ", p->name, p->pid, p->priority);
550         else if(p->state == RUNNING)
551             cprintf("%s \t %d \t RUNNING \t %d \n ", p->name, p->pid, p->priority);
552         else if(p->state == RUNNABLE)
553             cprintf("%s \t %d \t RUNNABLE \t %d \n ", p->name, p->pid, p->priority);
554     }
555     release(&ptable.lock);
556     return 22;
557 }
558
559 int
560 chpr(int pid, int priority)
561 {
562     struct proc *p;
563     acquire(&ptable.lock);
564     for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
565         if(p->pid == pid){
566             p->priority = priority;
567             break;
568         }
569     }
570 }
571
572 int
573 ps()
574 {
575     struct proc *p;
576     //Enables interrupts on this processor.
577     sti();
578
579     //Loop over process table looking for process with pid.
580     acquire(&ptable.lock);
581     cprintf("name \t pid \t state \t priority \n");
582     for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
583         if(p->state == SLEEPING)
584             cprintf("%s \t %d \t SLEEPING \t %d \n ", p->name, p->pid, p->priority);
585         else if(p->state == RUNNING)
586             cprintf("%s \t %d \t RUNNING \t %d \n ", p->name, p->pid, p->priority);
587         else if(p->state == RUNNABLE)
588             cprintf("%s \t %d \t RUNNABLE \t %d \n ", p->name, p->pid, p->priority);
589     }
590     release(&ptable.lock);
591     return 22;
592 }
593
594 int
595 chpr(int pid, int priority)
596 {
597     struct proc *p;
598     acquire(&ptable.lock);
599     for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
600         if(p->pid == pid){
601             p->priority = priority;
602             break;
603         }
604     }
605     release(&ptable.lock);
606     return pid;
607 }
```

Step-9:

```
aditi@aditi:~/Downloads/xv6-public$ gedit sysproc.c

*sysproc.c
~/Downloads/xv6-public

78 }
79
80 // return how many clock tick interrupts have occurred
81 // since start.
82 int
83 sys_uptime(void)
84 {
85     uint xticks;
86     acquire(&tickslock);
87     xticks = ticks;
88     release(&tickslock);
89     return xticks;
90 }
91
92
93
94 int
95 sys_ps()
96 {
97     return ps();
98 }
99
100 int
101 sys_chpr()
102 {
103     int pid, pr;
104     if(argint(0, &pid) < 0)
105         return -1;
106     if(argint(1, &pr) < 0)
107         return -1;
108     return chpr(pid, pr);
109 }
110
111
```

Step-10:

```
aditi@aditi:~/Downloads/xv6-public$ gedit usys.S

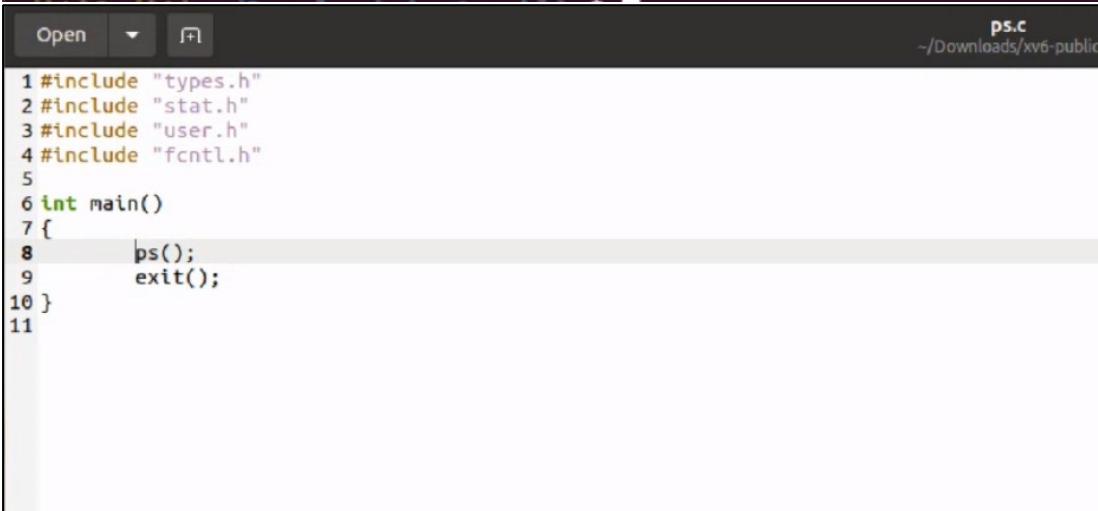
usys.S
~/Downloads/xv6-public

1 #include "syscall.h"
2 #include "traps.h"
3
4 #define SYSCALL(name) \
5     .globl name; \
6     name: \
7         movl $SYS_ ## name, %eax; \
8         int $T_SYSCALL; \
9         ret
10
11 SYSCALL(fork)
12 SYSCALL(exit)
13 SYSCALL(wait)
14 SYSCALL(pipe)
15 SYSCALL(read)
16 SYSCALL(write)
17 SYSCALL(close)
18 SYSCALL(kill)
19 SYSCALL(exec)
20 SYSCALL(open)
21 SYSCALL(mknod)
22 SYSCALL(unlink)
23 SYSCALL(fstat)
24 SYSCALL(link)
25 SYSCALL(mkdir)
26 SYSCALL(chdir)
27 SYSCALL(dup)
28 SYSCALL(getpid)
29 SYSCALL(sbrk)
30 SYSCALL(sleep)
31 SYSCALL(uptime)
32 SYSCALL(ps)
33 SYSCALL(chpr)
34
```

Step-11:

1.) ps.c

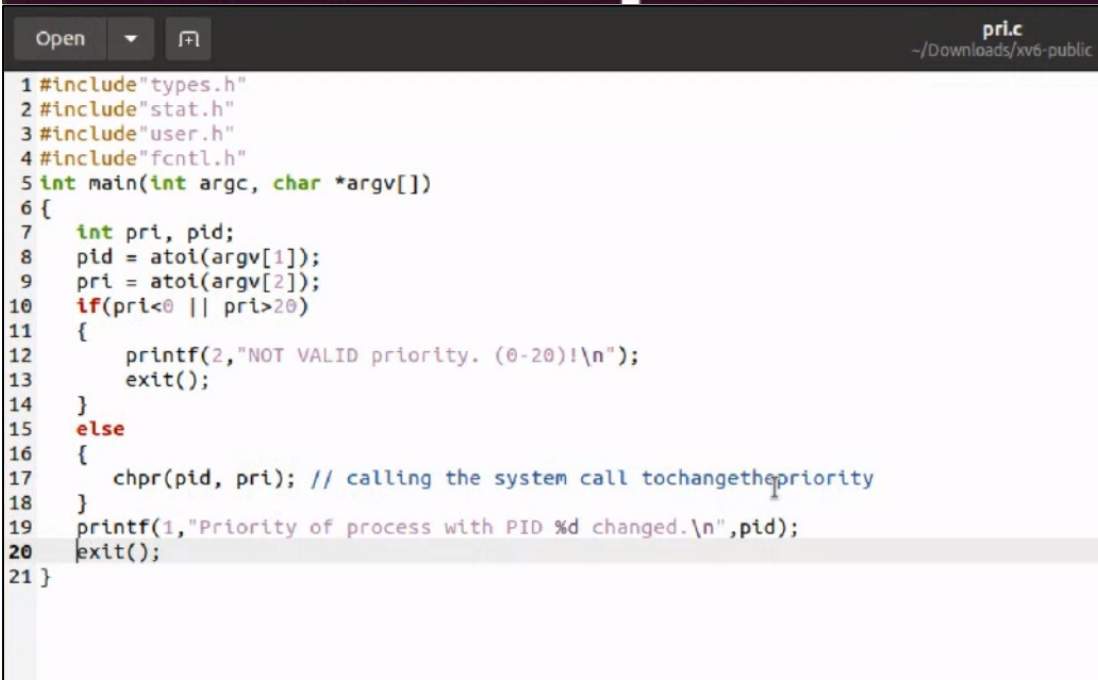
```
aditi@aditi:~/Downloads/xv6-public$ gedit ps.c
```



```
1 #include "types.h"
2 #include "stat.h"
3 #include "user.h"
4 #include "fcntl.h"
5
6 int main()
7 {
8     ps();
9     exit();
10 }
11
```

2.) pri.c

```
aditi@aditi:~/Downloads/xv6-public$ gedit pri.c
```



```
1 #include "types.h"
2 #include "stat.h"
3 #include "user.h"
4 #include "fcntl.h"
5 int main(int argc, char *argv[])
6 {
7     int pri, pid;
8     pid = atoi(argv[1]);
9     pri = atoi(argv[2]);
10    if(pri<0 || pri>20)
11    {
12        printf(2,"NOT VALID priority. (0-20)!\n");
13        exit();
14    }
15    else
16    {
17        chpr(pid, pri); // calling the system call to change the priority
18    }
19    printf(1,"Priority of process with PID %d changed.\n",pid);
20    exit();
21 }
```

Step-12:

The default scheduler of xv6 is based on Round Robin algorithm.

Now, we have to create a c program which creates a number of child process as mentioned by the user and consumes CPU time for testing our system calls and scheduling. So we create a new file *dpro.c*(dummy program)and write the following code:

```
aditi@aditi:~/Downloads/xv6-public$ gedit dpro.c

dpro.c
~/Downloads/xv6-public

1#include "types.h"
2#include "stat.h"
3#include "user.h"
4#include "fcntl.h"
5
6int main(int argc, char *argv[]) {
7    int pid;
8    int k, n;
9    int x, z;
10
11    if(argc < 2)
12        n = 1; //Default
13    else
14        n = atoi(argv[1]);
15    if (n < 0 || n > 20)
16        n = 2;
17    x = 0;
18    pid = 0;
19
20    for (k = 0; k < n; k++) {
21        pid = fork();
22        if (pid < 0) {
23            printf(1, "%d failed in fork!\n", getpid());
24        } else if (pid > 0) {
25            // parent
26            printf(1, "Parent %d creating child %d\n", getpid(), pid);
27            wait();
28        } else {
29            printf(1, "Child %d created\n", getpid());
30            for(z = 0; z < 4000000000; z+=1)
31                x = x + 3.14*89.64; //Useless calculation to consume CPU Time
32            break;
33        }
34    }
35    exit();
36}
```

Step-13:

```
aditi@aditi:~/Downloads/xv6-public$ gedit Makefile

Makefile
~/Downloads/xv6-public

166 .PRECIOUS: %.o
167
168 UPROGS=\
169     _cat\
170     _echo\
171     _forktest\
172     _grep\
173     _init\
174     _kill\
175     _ln\
176     _ls\
177     _mkdir\
178     _rm\
179     _sh\
180     _stressfs\
181     _usertests\
182     _wc\
183     _zombie\
184     _ps\
185     _pri\
186     _dpro\
187
188 fs.img: mkfs README $(UPROGS)
189     ./mkfs fs.img README $(UPROGS)
190
191 -include *.d
192
193 clean:
```



```

240 @echo "**** Now run 'gdb'." 1>&2
241 $(QEMU) -serial mon:stdio $(QEMUOPTS) -S $(QEMUGDB)
242
243 qemu-nox-gdb: fs.img xv6.img .gdbinit
244 @echo "**** Now run 'gdb'." 1>&2
245 $(QEMU) -nographic $(QEMUOPTS) -S $(QEMUGDB)
246
247 # CUT HERE
248 # prepare dist for students
249 # after running make dist, probably want to
250 # rename it to rev0 or rev1 or so on and then
251 # check in that version.
252
253 EXTRA=\
254     mkfs.c ulib.c user.h cat.c echo.c forktest.c grep.c kill.c\
255     ln.c ls.c mkdir.c rm.c stressfs.c usertests.c wc.c zombie.c ps.c pri.c d.c\
256     printf.c umalloc.c\
257     README dot-bochsrc *.pl toc.* runoff runoff1 runoff.list\
258     .gdbinit.tmpl gdbutil\
259
260 dist:
261     rm -rf dist
262     mkdir dist
263     for i in $(FILES); \
264     do \
265         grep -v PAGEBREAK $$i >dist/$$i; \
266     done
267     sed '/CUT HERE/,$$d' Makefile >dist/Makefile
268     echo >dist/runoff.spec
269     cp $(EXTRA) dist

```

---Implemented the system calls successfully.

>> Running the whole configuration:-

We make the required changes in scheduler function in proc.c file.

```

aditi@aditi:~/Downloads/xv6-public$ gedit proc.c
proc.c
~/Downloads/xv6-public
323 void
324 scheduler(void)
325 {
326     struct proc *p, *p1;
327     struct cpu *c = mycpu();
328     c->proc = 0;
329
330     for(;;){
331         // Enable interrupts on this processor.
332         sti();
333         struct proc *highP;
334         // Loop over process table looking for process to run.
335         acquire(&ptable.lock);
336         for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
337             if(p->state != RUNNABLE)
338                 continue;
339
340             // Switch to chosen process. It is the process's job
341             // to release ptable.lock and then reacquire it
342             // before jumping back to us.
343             highP = p;
344             //choose one with highest priority
345             for(p1 = ptable.proc; p1 < &ptable.proc[NPROC]; p1++){
346                 if(p1->state != RUNNABLE)
347                     continue;
348                 if(highP->priority > p1->priority) //larger value, lower priority
349                     highP = p1;
350             }
351             p = highP;
352             c->proc = p;
353             switchvm(p);
354             p->state = RUNNING;
355
356             swtch(&(c->scheduler), p->context);
357             switchkvm();
358         }

```

```
Open  [icon]  proc.c  ~/Downloads/xv6-publ
339
340 // Switch to chosen process. It is the process's job
341 // to release ptable.lock and then reacquire it
342 // before jumping back to us.
343 highP = p;
344 //choose one with highest priority
345 for(p1 = ptable.proc; p1 < &ptable.proc[NPROC]; p1++){
346     if(p1->state != RUNNABLE)
347         continue;
348     if(highP->priority > p1->priority) //larger value, lower priority
349         highP = p1;
350 }
351 p = highP;
352 c->proc = p;
353 switchvm(p);
354 p->state = RUNNING;
355
356 swtch(&(c->scheduler), p->context);
357 switchkvm();
358
359 // Process is done running for now.
360 // It should have changed its p->state before coming back.
361 c->proc = 0;
362 }
363 release(&ptable.lock);
364
365 }
366
```

We have implemented the system calls and changed the scheduling policy in xv6.

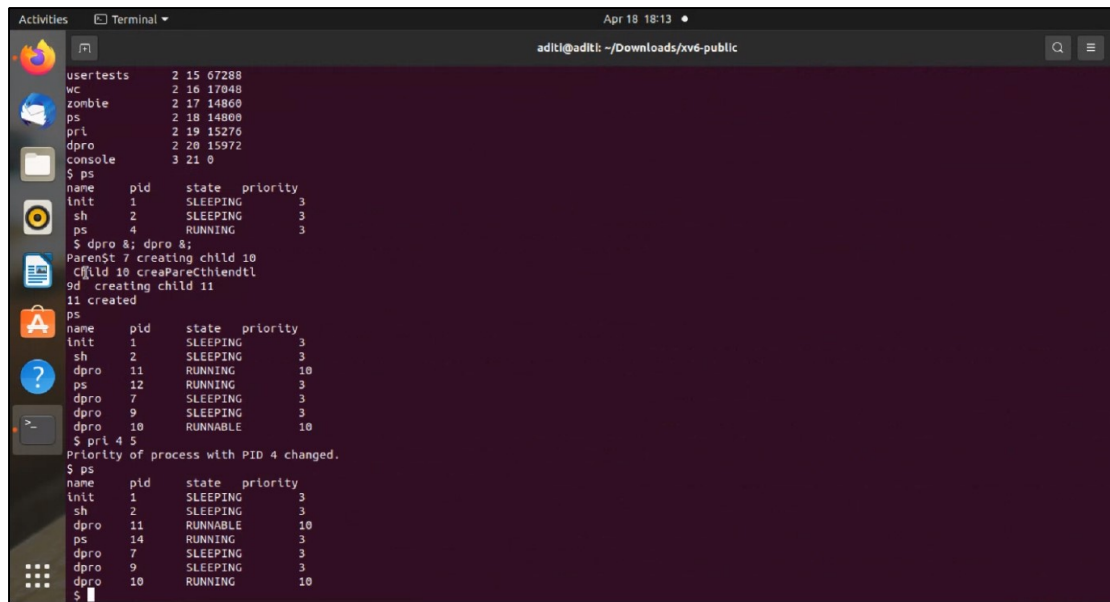
We have called dpro functions twice simultaneously.

OUTPUT:--

Make qemu-nox

```
Activities  [icon]  Terminal  Apr 18 18:11  aditi@aditi: ~/Downloads/xv6-public
IPXE (http://ipxe.org) 08:03:0 CA00 PCI2.10 PnP PMM+1FF8CA10+1FECCA10 CA00

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
$ ls
.          1 1 512
..         1 1 512
README    2 2 2286
cat        2 3 16312
echo       2 4 15168
forktest   2 5 9472
grep       2 6 18528
init       2 7 15748
kill       2 8 15196
ln         2 9 15048
ls         2 10 17680
mkdir      2 11 15292
rm         2 12 15272
sh         2 13 27904
stressfs   2 14 16184
usertests  2 15 67288
wc         2 16 17048
zombie     2 17 14860
ps         2 18 14800
pri        2 19 15276
dpro       2 20 15972
console    3 21 0
$ ps
name      pid  state  priority
init      1    SLEEPING  3
sh        2    SLEEPING  3
ps        4    RUNNING   3
$
```



The image shows a terminal window with the following content:

```
Activities  Terminal  Apr 18 18:13  aditi@aditi: ~/Downloads/xv6-public

userstats  2 15 67288
wc         2 16 17048
zombie    2 17 14868
ps         2 18 14808
pri        2 19 15276
dpro       2 20 15972
console    3 21 0
$ ps
name  pid  state  priority
init   1   SLEEPING  3
sh     2   SLEEPING  3
ps     4   RUNNING  3
$ dpro 8; dpro 8;
Parent$ 7 creating child 10
Child 10 creaParent$ child 11
9d creating child 11
11 created
$ ps
name  pid  state  priority
init   1   SLEEPING  3
sh     2   SLEEPING  3
dpro   11  RUNNING  10
ps     12  RUNNING  3
dpro   7   SLEEPING  3
dpro   9   SLEEPING  3
dpro  10  RUNNING  10
$ prt 4 5
Priority of process with PID 4 changed.
$ ps
name  pid  state  priority
init   1   SLEEPING  3
sh     2   SLEEPING  3
dpro  11  RUNNING  10
ps     14  RUNNING  3
dpro   7   SLEEPING  3
dpro   9   SLEEPING  3
dpro  10  RUNNING  10
$
```

Submitted by:-

Aditi Singh
20051629
CSE-03