

MNNIT ALLAHABAD

OPERATING SYSTEM

SUBMITTED BY :

NAME : SHISHU

ROLL/REG ID : 2020CA089

SUBMIT DATE : 19/12/2021

SUBMITTED TO :

TEACHER _____

DEPPT : COMPUTER SCIENCE

DEADLINE: 19/12/2021

Question 1. Adding Process Priority: In this lab, we will walk you through the steps of adding a priority attribute to a process in xv6 and changing its value. We assign a process with a value between 0 and 20, the smaller the value, the higher the priority. The default value is 10.

- i. Add priority to struct proc in proc.h
- ii. Assign default priority in allocproc() in proc.c
- iii. Modify cps() in proc.c discussed in the last lab to include the print the priority
- iv. Write a dummy program named foo.c that creates some child processes and consumes some computing time
- v. Add the function chpr() (meaning change priority) in proc.c
- vi. Add sys_chpr() in sysproc.c
- vii. Add chpr() as a system call to xv6
- viii. Create the user file nice.c with which calls chpr(). Test nice

```
proc.h      x      proc.c      x      k
24 // The layout of the context matches the layout of the stack in switch.
25 // at the "Switch stacks" comment. Switch doesn't save eip explicitly,
26 // but it is on the stack and allocproc() manipulates it.
27 struct context {
28     uint edi;
29     uint esi;
30     uint ebx;
31     uint ebp;
32     uint eip;
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;          // Process Priority
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
C/ObjC Header  Tab Width: 8  Ln 33, Col 3  INS
```

Adding priority in proc.h

```
proc.h      x      proc.c      x
71 // state required to run in the kernel.
72 // Otherwise return 0.
73 static struct proc*
74 allocproc(void)
75 {
76     struct proc *p;
77     char *sp;
78
79     acquire(&ptable.lock);
80
81     for(p = ptable.proc; p < &ptable.proc[NPROC]; p++)
82         if(p->state == UNUSED)
83             goto found;
84
85     release(&ptable.lock);
86     return 0;
87
88 found:
89     p->state = EMBRYO;
90     p->pid = nextpid++;
91     p->priority = 10; //default priority |
92
93     release(&ptable.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.
```

C ▾ Tab Width: 8 ▾ Ln 91, Col 40 ▾ INS

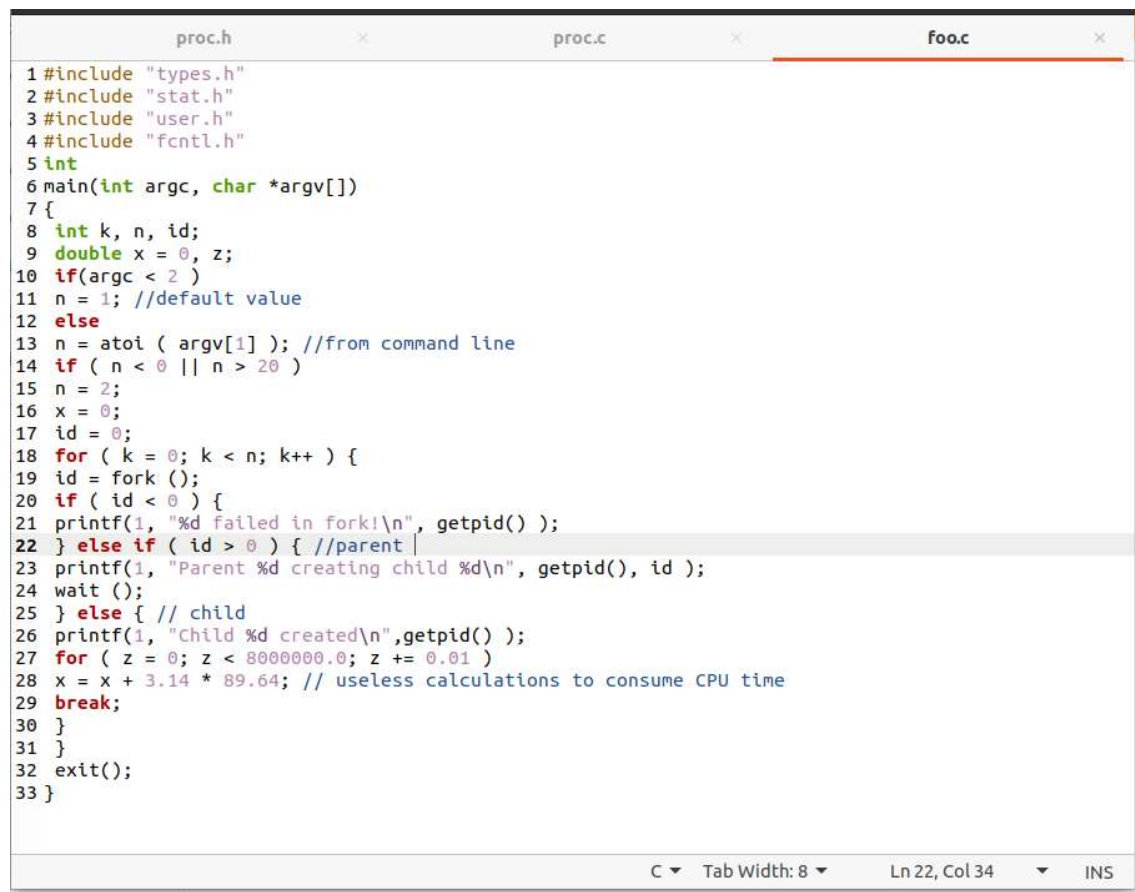
Assigning default priority in allocproc()

```
Open  [icon]  proc.c  ~/Desktop/xv6/xv6-public  Save  [icon]  [icon]  [icon]
549     cprintf(" %p", pc[i]);
550 }
551     cprintf("\n");
552 }
553 }
554
555 //ADDED
556 //current process status
557 int
558 cps(void)
559 {
560     struct proc *p;
561     // Enable interrupts on this processor.
562     sti();
563     // Loop over process table looking for process with pid.
564     acquire(&table.lock);
565     cprintf(" name \t pid \t state \t priority \n");
566     for(p = table.proc; p < &table.proc[NPROC]; p++)
567     {
568         if ( p->state == SLEEPING )
569             cprintf("%s \t %d \t SLEEPING \t %d \n ", p->name, p->pid, p->priority );
570         else if ( p->state == RUNNING )
571             cprintf("%s \t %d \t RUNNING \t %d \n ", p->name, p->pid, p->priority );
572         else if ( p->state == RUNNABLE )
573             cprintf("%s \t %d \t RUNNABLE \t %d \n ", p->name, p->pid, p->priority );
574     }
575     release(&table.lock);
576     return 22;
577 }
578
579 //current process name
580 int
581 cpsn()
582 {
583     struct proc * p;
584     // Enable interrupts on this processor.
585     sti();
586     // Loop over process table looking for process with pid.
587     acquire(&table.lock);
588     cprintf("NAME\n");
589     for (p = table.proc; p < &table.proc[NPROC]; p++)
590     {
591         if (p->state == SLEEPING || p->state == RUNNING)
```

Modifying the cps in proc.c that now also prints the priority

```
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 sta8
init: starting sh
$ ps
name    pid    state  priority
init    1      SLEEPING    10
sh      2      SLEEPING    10
ps      3      RUNNING     10
lapicid 1: panic: release
80104b5a 80104317 80104f5d 801060b1 80105df5 0 0 0 0 0
```

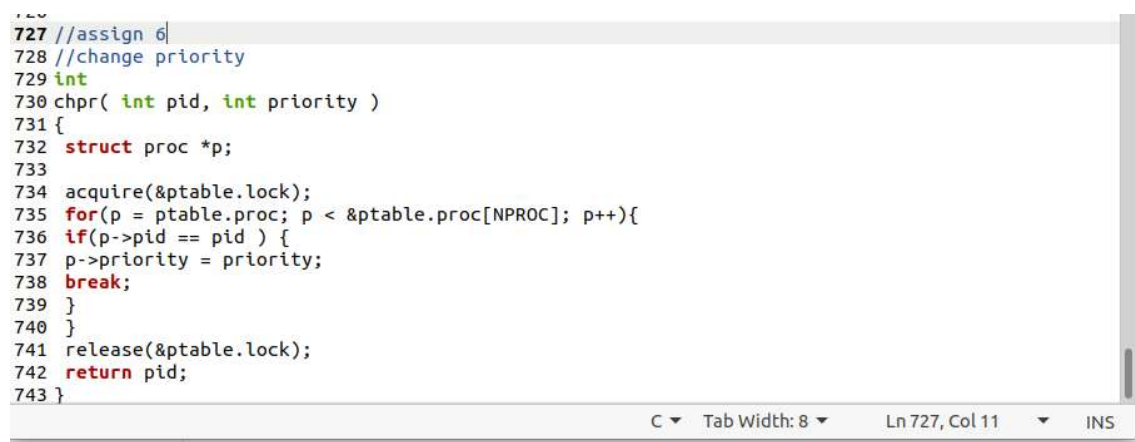
Dummy foo.c



```
1#include "types.h"
2#include "stat.h"
3#include "user.h"
4#include "fcntl.h"
5int
6main(int argc, char *argv[])
7{
8    int k, n, id;
9    double x = 0, z;
10    if(argc < 2)
11        n = 1; //default value
12    else
13        n = atoi ( argv[1] ); //from command line
14    if ( n < 0 || n > 20 )
15        n = 2;
16    x = 0;
17    id = 0;
18    for ( k = 0; k < n; k++ ) {
19        id = fork ();
20        if ( id < 0 ) {
21            printf(1, "%d failed in fork!\n", getpid() );
22        } else if ( id > 0 ) { //parent
23            printf(1, "Parent %d creating child %d\n", getpid(), id );
24            wait ();
25        } else { // child
26            printf(1, "Child %d created\n",getpid() );
27            for ( z = 0; z < 8000000.0; z += 0.01 )
28                x = x + 3.14 * 89.64; // useless calculations to consume CPU time
29            break;
30        }
31    }
32    exit();
33}
```

C Tab Width: 8 Ln 22, Col 34 INS

Added chpr() in proc.c



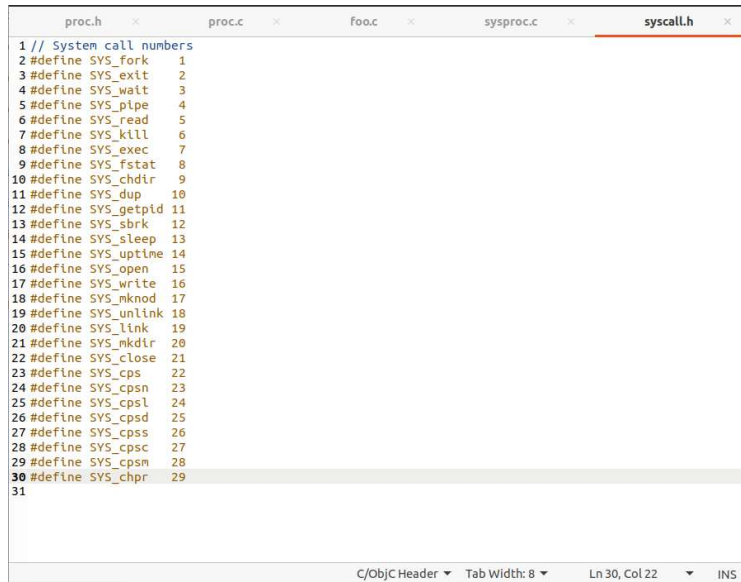
```
727 //assign 6
728 //change priority
729 int
730 chpr( int pid, int priority )
731 {
732     struct proc *p;
733
734     acquire(&ptable.lock);
735     for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
736         if(p->pid == pid ) {
737             p->priority = priority;
738             break;
739         }
740     }
741     release(&ptable.lock);
742     return pid;
743 }
```

C Tab Width: 8 Ln 727, Col 11 INS

Added sys_chpr() in sysproc.c

```
139 //assign 6
140 int
141 sys_chpr (void)
142 {
143     int pid, pr;
144     if(argint(0, &pid) < 0)
145         return -1;
146     if(argint(1, &pr) < 0)
147         return -1;
148     return chpr ( pid, pr );
149 }
150
151
```

Now in the below screenshots we're just adding `chpr()` as a system call just like we did for `cps()`



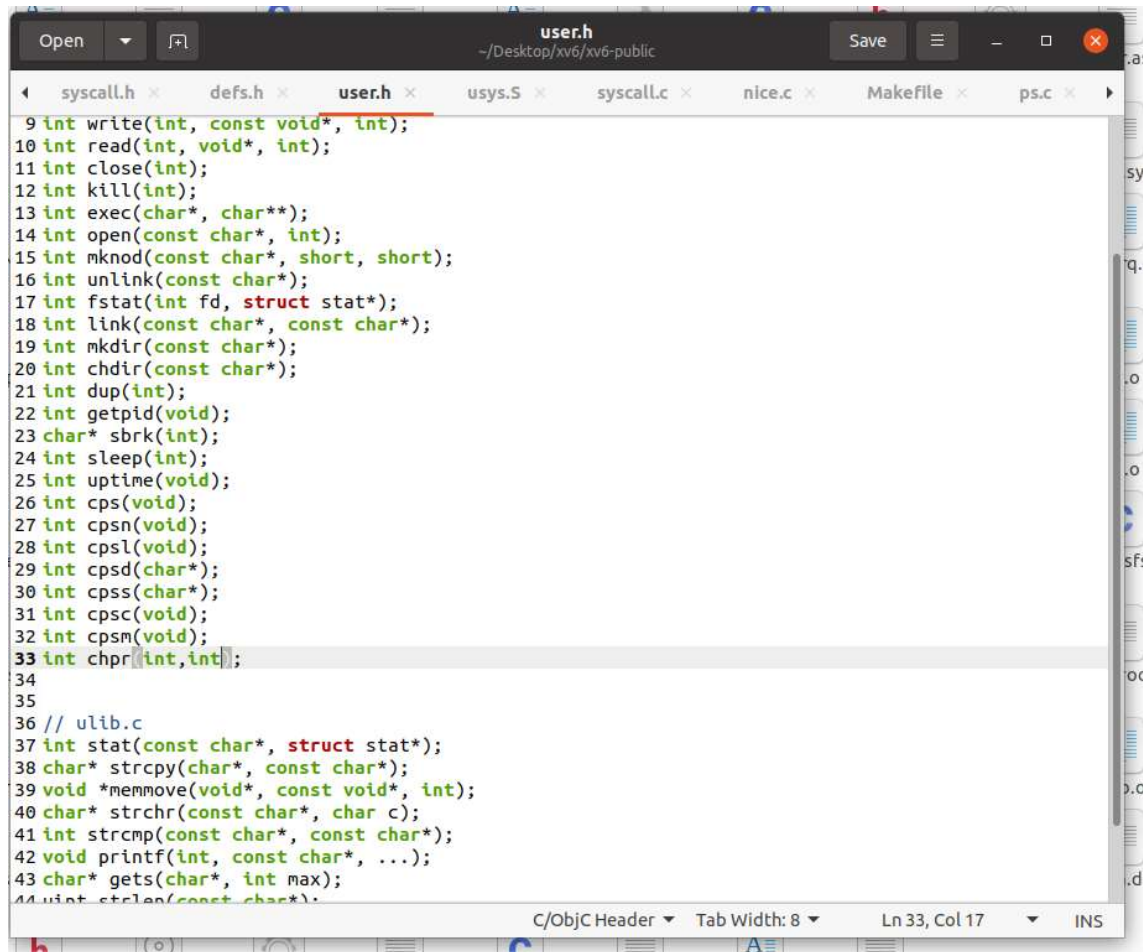
The screenshot shows a code editor with five tabs: `proc.h`, `proc.c`, `foo.c`, `sysproc.c`, and `syscall.h`. The `syscall.h` tab is active, displaying a list of system call numbers. The list starts with `1 // System call numbers` and continues with `#define SYS_fork 1` through `#define SYS_chpr 29`. Line 30 is highlighted, showing `#define SYS_chpr 29`. Line 31 is empty. The status bar at the bottom indicates `C/ObjC Header`, `Tab Width: 8`, `Ln 30, Col 22`, and `INS`.

```
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_cps 22
24 #define SYS_cpsn 23
25 #define SYS_cpsl 24
26 #define SYS_cpss 25
27 #define SYS_cpss 26
28 #define SYS_cpss 27
29 #define SYS_cpss 28
30 #define SYS_chpr 29
31
```



```
Open  defs.h  Save  ~/Desktop/xv6/xv6-public  sys.h  defs.h  user.h  usys.S  syscall.c  nice.c  Makefile  ps.c  ...
111 struct cpu* mycpu(void);
112 struct proc* myproc();
113 void pinit(void);
114 void procdump(void);
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 cps(void);
124 int cpsn(void);
125 int cpsl(void);
126 int cpsd(char*);
127 int cpss(char*);
128 int cpssc(void);
129 int chpr(int, int);
130
131 // swtch.S
132 void swtch(struct context**, struct context*);
133
134 // spinlock.c
135 void acquire(struct spinlock*);
136 void getcallerpcs(void*, uint*);
137 int holding(struct spinlock*);
138 void initlock(struct spinlock*, char*);
139 void release(struct spinlock*);
140 void pushcli(void);
141 void popcli(void);
142
143 // sleeplock.c
144 void acquiresleep(struct sleeplock*);
145 void releasesleep(struct sleeplock*);
```

C/ObjC Header Tab Width: 8 Ln 129, Col 29 INS



The image shows a code editor window with the title bar "user.h" and a path "~/Desktop/xv6/xv6-public". The editor contains a list of system call prototypes in C. The tabs at the top are: syscall.h, defs.h, user.h (selected), usys.S, syscall.c, nice.c, Makefile, and ps.c. The code is as follows:

```
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 cps(void);
27 int cpsn(void);
28 int cpsl(void);
29 int cpsd(char*);
30 int cpss(char*);
31 int cpvc(void);
32 int cpsm(void);
33 int chpr(int, int);
34
35
36 // ulib.c
37 int stat(const char*, struct stat*);
38 char* strcpy(char*, const char*);
39 void *memmove(void*, const void*, int);
40 char* strchr(const char*, char c);
41 int strcmp(const char*, const char*);
42 void printf(int, const char*, ...);
43 char* gets(char*, int max);
44 int stolen(const char*);
```

The status bar at the bottom shows "C/ObjC Header", "Tab Width: 8", "Ln 33, Col 17", and "INS".

```
proc.h x  proc.c x  foo.c x  sysproc.c x  syscall.h x  defs.h x  user.h x  usys.S x
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(cps)
33 SYSCALL(cpsn)
34 SYSCALL(cpsl)
35 SYSCALL(cpsd)
36 SYSCALL(cpss)
37 SYSCALL(cpsc)
38 SYSCALL(cpsm)
39 SYSCALL(chpr)
```

C ▾ Tab Width: 8 ▾ Ln 39, Col 13 ▾ INS

Open

✕

✕

*syscall.c

Save

☰

—

□

✕

~/Desktop/xv6/xv6-public

proc.c ✕foo.c ✕sysproc.c ✕syscall.h ✕defs.h ✕user.h ✕usys.S ✕*syscall.c ✕

```
106 extern int sys_cps(void);
107 extern int sys_cpsn(void);
108 extern int sys_cpsl(void);
109 extern int sys_cpsd(void);
110 extern int sys_cpss(void);
111 extern int sys_cpsc(void);
112 extern int sys_cpsm(void);
113 extern int sys_chpr(void);
114
115
116
117 static int (*syscalls[])(void) = {
118 [SYS_fork]      sys_fork,
119 [SYS_exit]      sys_exit,
120 [SYS_wait]      sys_wait,
121 [SYS_pipe]      sys_pipe,
122 [SYS_read]      sys_read,
123 [SYS_kill]      sys_kill,
124 [SYS_exec]      sys_exec,
125 [SYS_fstat]     sys_fstat,
126 [SYS_chdir]     sys_chdir,
127 [SYS_dup]       sys_dup,
128 [SYS_getpid]    sys_getpid,
129 [SYS_sbrk]      sys_sbrk,
130 [SYS_sleep]     sys_sleep,
131 [SYS_uptime]    sys_uptime,
132 [SYS_open]      sys_open,
133 [SYS_write]     sys_write,
134 [SYS_mknod]     sys_mknod,
135 [SYS_unlink]    sys_unlink,
136 [SYS_link]      sys_link,
137 [SYS_mkdir]     sys_mkdir,
138 [SYS_close]     sys_close,
139 [SYS_cps]       sys_cps,
140 [SYS_cpsn]      sys_cpsn,
```

C ▾Tab Width: 8 ▾Ln 113, Col 27 ▾INS

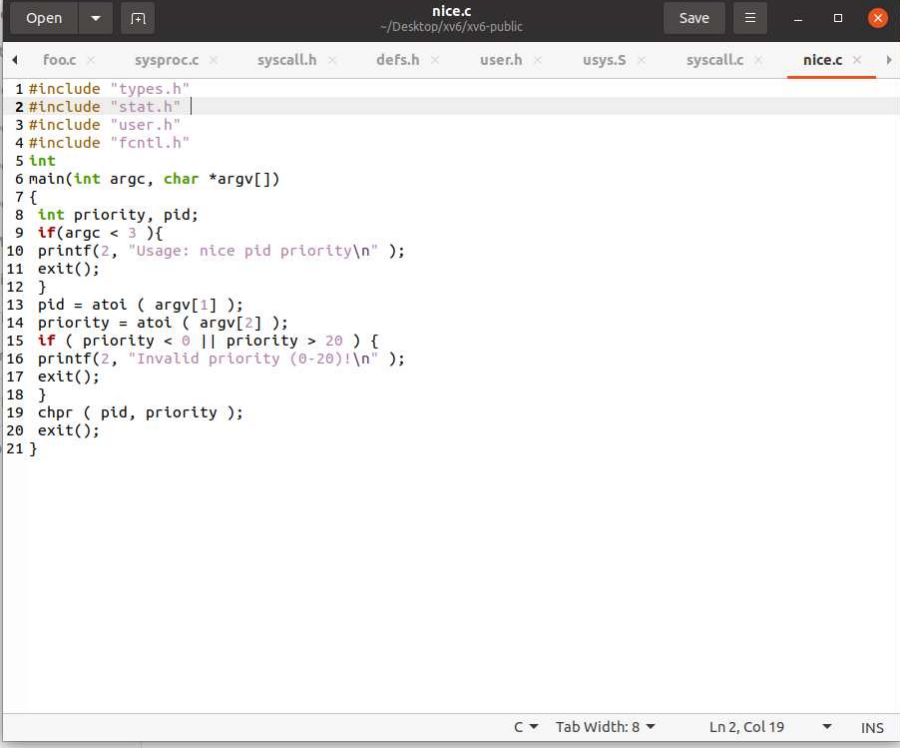
Open syscall.c ~/Desktop/xv6/xv6-public Save

proc.c x foo.c x sysproc.c x syscall.h x defs.h x user.h x usys.S x syscall.c x

```
118 [SYS_fork] sys_fork,
119 [SYS_exit] sys_exit,
120 [SYS_wait] sys_wait,
121 [SYS_pipe] sys_pipe,
122 [SYS_read] sys_read,
123 [SYS_kill] sys_kill,
124 [SYS_exec] sys_exec,
125 [SYS_fstat] sys_fstat,
126 [SYS_chdir] sys_chdir,
127 [SYS_dup] sys_dup,
128 [SYS_getpid] sys_getpid,
129 [SYS_sbrk] sys_sbrk,
130 [SYS_sleep] sys_sleep,
131 [SYS_uptime] sys_uptime,
132 [SYS_open] sys_open,
133 [SYS_write] sys_write,
134 [SYS_mknod] sys_mknod,
135 [SYS_unlink] sys_unlink,
136 [SYS_link] sys_link,
137 [SYS_mkdir] sys_mkdir,
138 [SYS_close] sys_close,
139 [SYS_cps] sys_cps,
140 [SYS_cpsn] sys_cpsn,
141 [SYS_cpsl] sys_cpsl,
142 [SYS_cpzd] sys_cpzd,
143 [SYS_cpss] sys_cpss,
144 [SYS_cpzc] sys_cpzc,
145 [SYS_chpr] sys_chpr,
146
147 };
148
149 void
150 syscall(void)
151 {
152     int num;
153     struct proc *curproc = myproc();
```

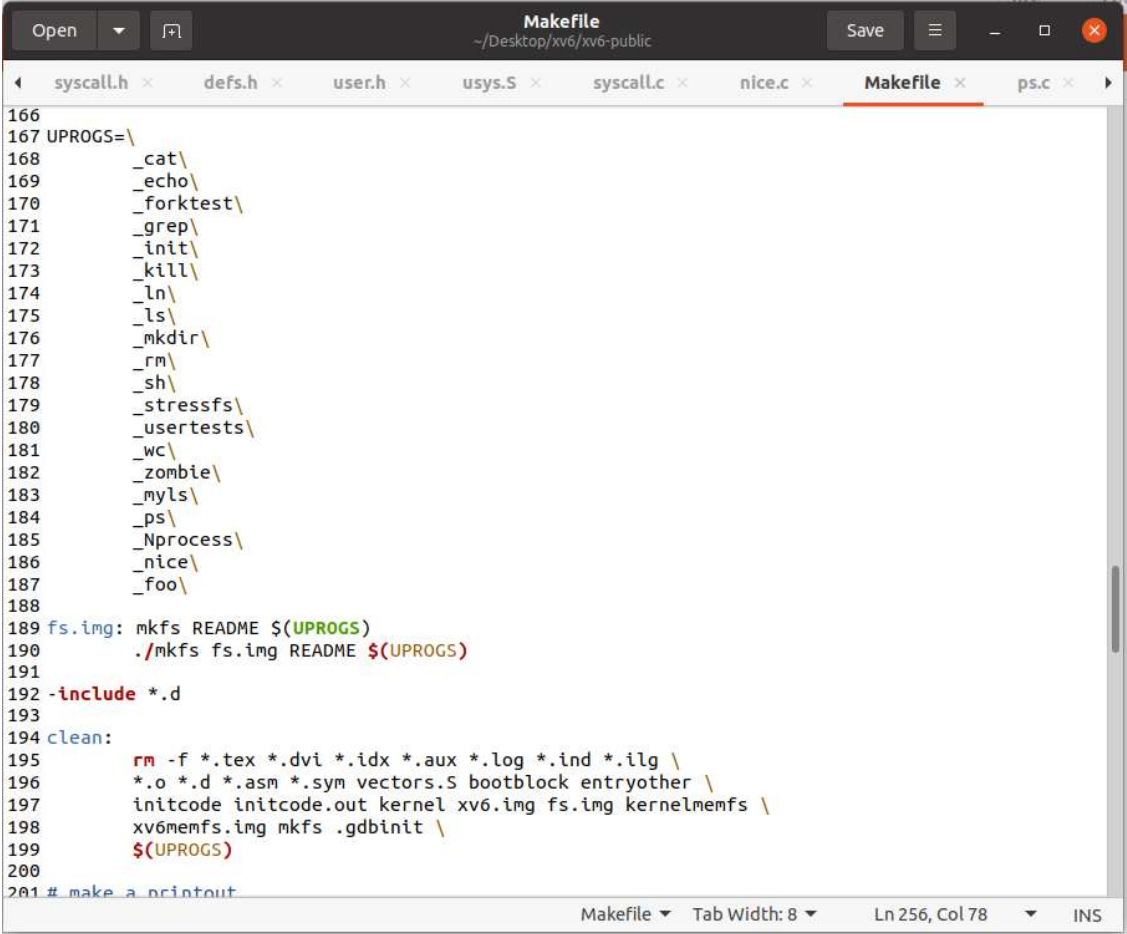
C Tab Width: 8 Ln 145, Col 24 INS

Now creating a new user file nice.c which will call chpr



```
1#include "types.h"
2#include "stat.h"
3#include "user.h"
4#include "fcntl.h"
5int
6main(int argc, char *argv[])
7{
8    int priority, pid;
9    if(argc < 3){
10        printf(2, "Usage: nice pid priority\n" );
11        exit();
12    }
13    pid = atoi ( argv[1] );
14    priority = atoi ( argv[2] );
15    if ( priority < 0 || priority > 20 ) {
16        printf(2, "Invalid priority (0-20)!\n" );
17        exit();
18    }
19    chpr ( pid, priority );
20    exit();
21}
```

Modifying the makefile



The screenshot shows a text editor window titled "Makefile" with the path "~/Desktop/xv6/xv6-public". The editor has several tabs open: syscall.h, defs.h, user.h, usys.S, syscall.c, nice.c, Makefile (active), and ps.c. The Makefile content is as follows:

```
166
167 UPROGS=\
168     _cat\
169     _echo\
170     _forktest\
171     _grep\
172     _init\
173     _kill\
174     _ln\
175     _ls\
176     _mkdir\
177     _rm\
178     _sh\
179     _stressfs\
180     _usertests\
181     _wc\
182     _zombie\
183     _mysls\
184     _ps\
185     _Nprocess\
186     _nice\
187     _foo\
188
189 fs.img: mkfs README $(UPROGS)
190     ./mkfs fs.img README $(UPROGS)
191
192 -include *.d
193
194 clean:
195     rm -f *.tex *.dvi *.idx *.aux *.log *.ind *.ilg \
196         *.o *.d *.asm *.sym vectors.S bootblock entryother \
197         initcode initcode.out kernel xv6.img fs.img kernelmemfs \
198         xv6memfs.img mkfs .gdbinit \
199         $(UPROGS)
200
201 # make a printout
```

The status bar at the bottom indicates "Makefile", "Tab Width: 8", "Ln 256, Col 78", and "INS".

Now making some dummy processes using foo command and then changing the priority of process with pid 4 to 18 using nice command

```
neeraj@neeraj: ~/Desktop/xv6/xv6-public
SeaBIOS (version 1.13.0-1ubuntu1.1)

iPXE (http://ipxe.org) 00:03.0 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 sta8
init: starting sh
$ foo 4 &
$ Parent 4 creating child 5
Child 5 created
ps
name    pid    state  priority
init     1     SLEEPING    10
sh       2     SLEEPING    10
foo      5     RUNNING    10
foo      4     SLEEPING    10
ps       6     RUNNING    10
lapicid 1: panic: release
80104bba 80104317 80104fbd 80106161 80105ea4 0 0 0 0 0
```

```
neeraj@neeraj: ~/Desktop/xv6/xv6-public

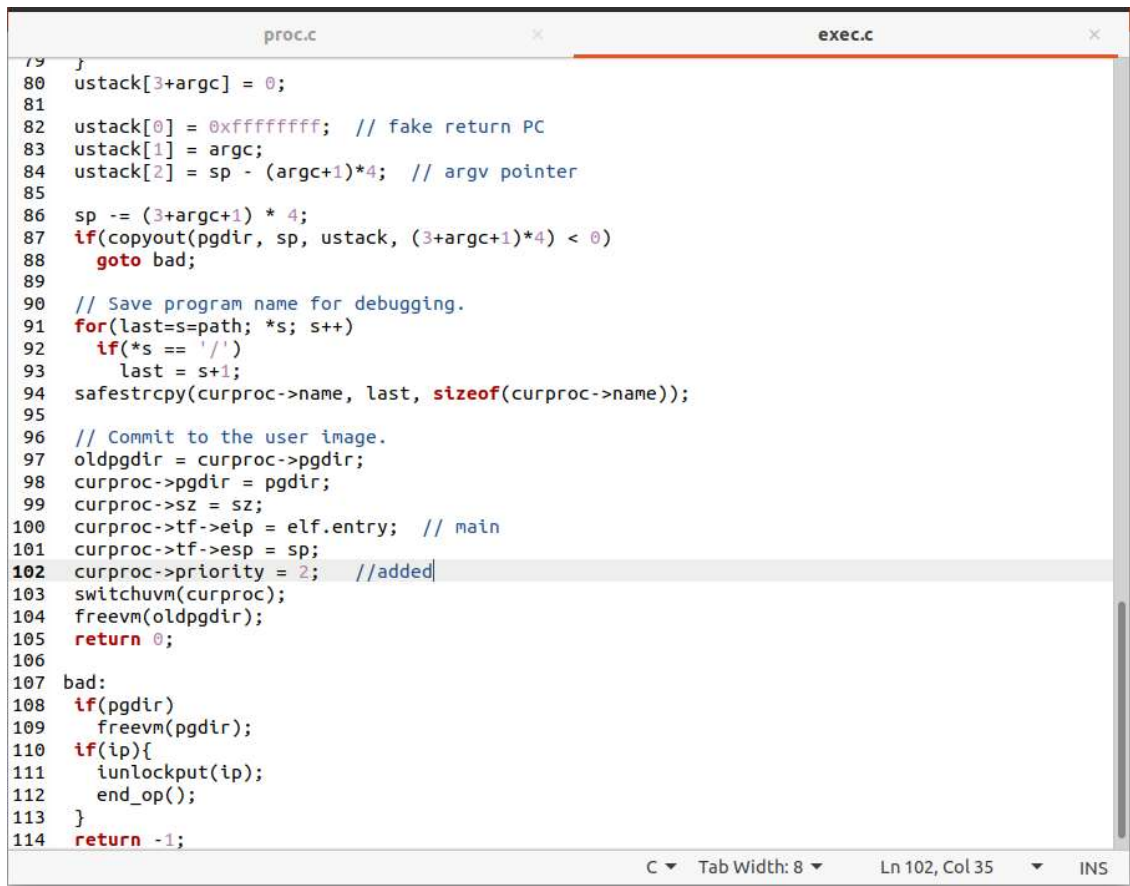
iPXE (http://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1FF8CA10+1FECCA10 CA00

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 sta8
init: starting sh
$ foo 4 &
$ Parent 4 creating child 5
Child 5 created
nice 4 18
$ ps
name    pid    state  priority
init     1     SLEEPING    10
sh       2     SLEEPING    10
foo      5     RUNNING    10
foo      4     SLEEPING    18
ps       7     RUNNING    10
lapicid 1: panic: release
80104bba 80104317 80104fbd 80106161 80105ea4 0 0 0 0 0
```


Question 2. XV6 Process Priority Scheduling: In the previous question, we have learned how to change the priority of a process. In this question, we will implement a very simple priority scheduling policy. We simply choose a runnable process with the highest priority to run. (In practice, multilevel queues are often used to put processes into groups with similar priorities.) As we have done in the previous question, we assume that a process has a value between 0 and 20, the smaller the value, the higher the priority. The default value is 10. The program `nice` that we implemented in the previous question is used to change the priority of a process.

- i. Give high priority to a newly loaded process by adding a priority statement in `exec.c`
- ii. ii. Modify `foo.c` so that the parent waits for the children and adjust the loop for your convenience
- iii. iii. Observe the default round-robin (RR) scheduling
- iv. iv. Implement Priority Scheduling
- v. v. Observe the priority scheduling

Adding high priority by adding a priority statement in exec.c



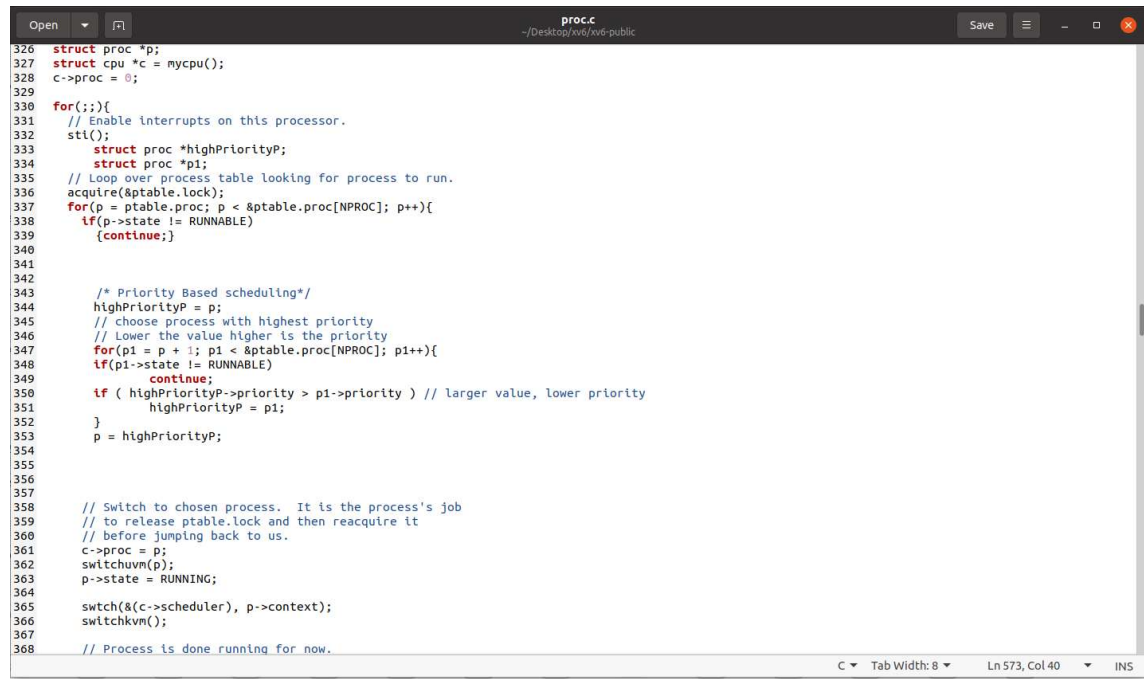
```
79 }
80 ustack[3+argc] = 0;
81
82 ustack[0] = 0xffffffff; // fake return PC
83 ustack[1] = argc;
84 ustack[2] = sp - (argc+1)*4; // argv pointer
85
86 sp -= (3+argc+1) * 4;
87 if(copyout(pgdir, sp, ustack, (3+argc+1)*4) < 0)
88     goto bad;
89
90 // Save program name for debugging.
91 for(last=s=path; *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 = 2; //added
103 switchvm(curproc);
104 freevm(oldpgdir);
105 return 0;
106
107 bad:
108 if(pgdir)
109     freevm(pgdir);
110 if(ip){
111     iunlockput(ip);
112     end_op();
113 }
114 return -1;
```

C Tab Width: 8 Ln 102, Col 35 INS

Observing the default round-robin scheduling

```
$ foo &;
$ Parent 8 creating child 9
Child 9 created
foo &;
$ Parent 12child 13 created
creating child 13
foo &;
$ Parent 16 creating child 17
Child 17 created
ps
name      pid      state  priority
init       1      SLEEPING      2
sh         2      SLEEPING      2
foo        13     RUNNING     10
foo         8     SLEEPING      2
foo        12     SLEEPING      2
ps         18     RUNNING      2
foo        16     SLEEPING      2
$ ps
name      pid      state  priority
init       1      SLEEPING      2
sh         2      SLEEPING      2
foo         9     RUNNING     10
foo         8     SLEEPING      2
foo        12     SLEEPING      2
ps         19     RUNNING      2
foo        16     SLEEPING      2
```

Modifying the schedule function in proc.c to select the highest priority runnable process



```
326 struct proc *p;
327 struct cpu *c = mycpu();
328 c->proc = 0;
329
330 for(;;){
331     // Enable interrupts on this processor.
332     sti();
333     struct proc *highPriorityP;
334     struct proc *p1;
335     // Loop over process table looking for process to run.
336     acquire(&ptable.lock);
337     for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
338         if(p->state != RUNNABLE)
339             continue;
340
341         /* Priority Based scheduling*/
342         highPriorityP = p;
343         // choose process with highest priority
344         // Lower the value higher is the priority
345         for(p1 = p + 1; p1 < &ptable.proc[NPROC]; p1++){
346             if(p1->state != RUNNABLE)
347                 continue;
348             if ( highPriorityP->priority > p1->priority ) // larger value, lower priority
349                 highPriorityP = p1;
350         }
351         p = highPriorityP;
352
353         // Switch to chosen process. It is the process's job
354         // to release ptable.lock and then reacquire it
355         // before jumping back to us.
356         c->proc = p;
357         switchuvm(p);
358         p->state = RUNNING;
359
360         switch(&(c->scheduler), p->context);
361         switchkvm();
362     }
363     // Process is done running for now.
364 }
```

Now first we're creating some dummy processes using our foo command and then observe the priorities of the different processes.

Now we're changing the priority of the runnable process with pid =15 from 10 to 8 so that it now become ready to run and then in the next ps command its status changes from runnable to running

```
$ foo &
$ Parent 6 creating cChild 7 created
child 7
foo &
$ Child 11 created
Parent 10 creating child 11
foo &
Parent 14 creating child Child 15 created
15
$ foo &
$ Parent 18 creating child 19
Child 19 created
ps
name      pid      state  priority
init       1      SLEEPING      2
sh         2      SLEEPING      2
foo        7      RUNNABLE     10
ps        20      RUNNING       2
foo        6      SLEEPING       2
foo        19      RUNNABLE     10
foo        10      SLEEPING       2
foo        11      RUNNING      10
foo        14      SLEEPING       2
foo        15      RUNNABLE     10
foo        18      SLEEPING       2
$ nice 15 8
$ ps
name      pid      state  priority
init       1      SLEEPING      2
sh         2      SLEEPING      2
foo        7      RUNNABLE     10
ps        22      RUNNING       2
foo        6      SLEEPING       2
foo        19      RUNNABLE     10
foo        10      SLEEPING       2
foo        11      RUNNABLE     10
foo        14      SLEEPING       2
foo        15      RUNNING       8
foo        18      SLEEPING       2
```