


Question 1. You need to copy of command “ls” as “myls” and modify the make file accordingly to run command “myls” in terminal.



```
1 //COPY PASTED CODE FROM ls.c FILE
2
3 #include "types.h"
4 #include "stat.h"
5 #include "user.h"
6 #include "fs.h"
7
8 char*
9 fmtname(char *path)
10 {
11     static char buf[DIRSIZ+1];
12     char *p;
13
14     // Find first character after last slash.
15     for(p=path+strlen(path); p >= path && *p != '/'; p--)
16         ;
17     p++;
18
19     // Return blank-padded name.
20     if(strlen(p) >= DIRSIZ)
21         return p;
22     memmove(buf, p, strlen(p));
23     memset(buf+strlen(p), ' ', DIRSIZ-strlen(p));
24     return buf;
25 }
26
27 void
28 ls(char *path)
29 {
30     char buf[512], *p;
31     int fd;
32     struct dirent de;
33     struct stat st;
34
35     if((fd = open(path, 0)) < 0){
36         printf("ls: cannot open %s\n", path);
```

-CREATING OUR NEW myls.c FILE , THE CONTENTS OF THIS ARE EXACTLY THE SAME AS IN ls.c FILE

```
164 # http://www.gnu.org/software/nake/manual/html\_node/Chained-Rules.html
165 .PRECIOUS: %.o
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     _myls\
184
185 fs.img: mkfs README $(UPROGS)
186     ./mkfs fs.img README $(UPROGS)
187
188 -include *.d
189
190 clean:
191     rm -f *.tex *.dvi *.idx *.aux *.log *.ind *.ilg \
192         *.o *.d *.asm *.sym vectors.S bootblock entryother \
193         initcode initcode.out kernel xv6.img fs.img kernelmemfs \
194         xv6memfs.img mkfs .gdbinit \
195         $(UPROGS)
196
197 # make a printout
198 FILES = $(shell grep -v '^\\#' runoff.list)
```

-MAKING THE CHANGES IN THE makefile TO INCLUDE myls

```
$ ls
.          Directory 1 512
..         Directory 1 512
README    File 2 2286
cat       File 3 16256
echo      File 4 15112
forktest  File 5 9416
grep      File 6 18476
init      File 7 15696
kill      File 8 15140
ln        File 9 14992
ls        File 10 17880
mkdir     File 11 15240
rm        File 12 15216
sh        File 13 27852
stressfs  File 14 16128
usertests File 15 67236
wc        File 16 16992
zombie    File 17 14808
myls      File 18 17888
console   Device 19 0
$
```

-OUTPUT BY LS COMMAND

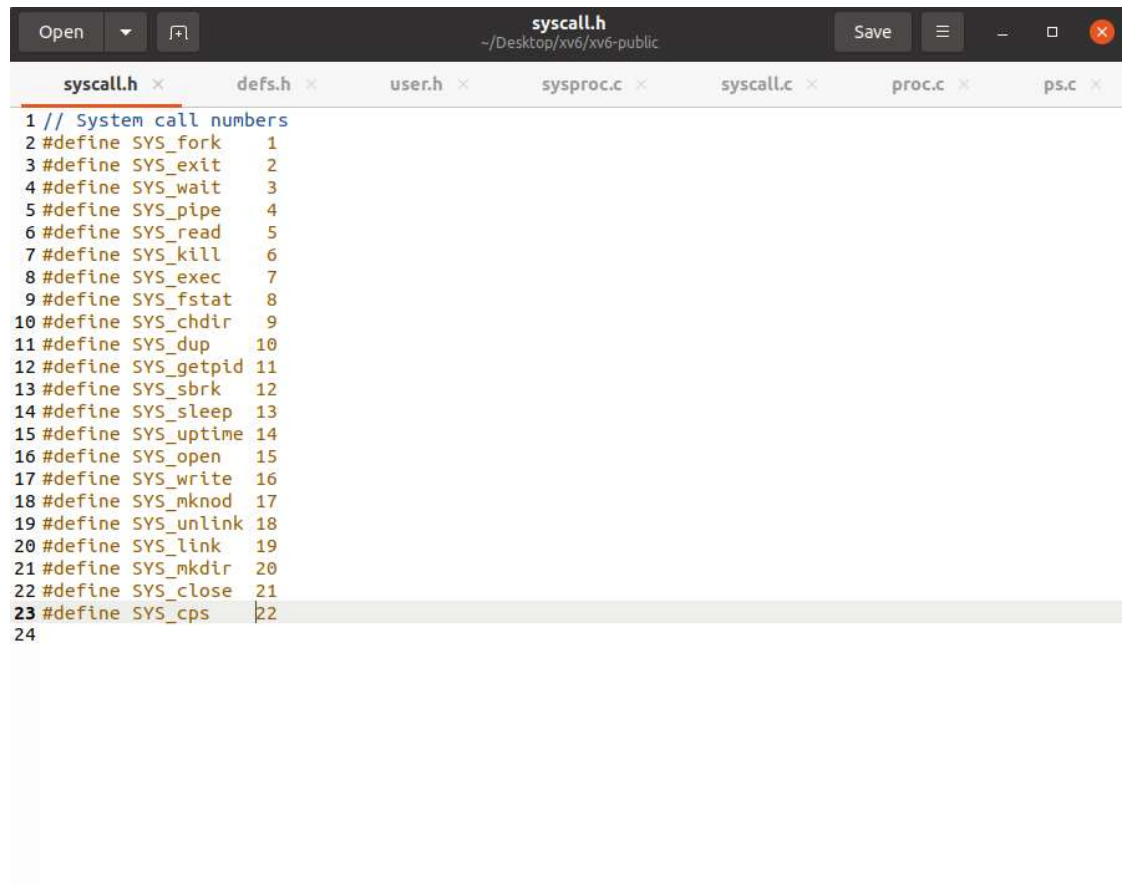
```
$ myls
.          Directory 1 512
..         Directory 1 512
README    File 2 2286
cat       File 3 16256
echo      File 4 15112
forktest  File 5 9416
grep      File 6 18476
init      File 7 15696
kill      File 8 15140
ln        File 9 14992
ls        File 10 17880
mkdir     File 11 15240
rm        File 12 15216
sh        File 13 27852
stressfs  File 14 16128
usertests File 15 67236
wc        File 16 16992
zombie    File 17 14808
mysls     File 18 17888
console   Device 19 0
$
```

-OUTPUT BY myls COMMAND

Question 2.

1. Modify cps() in proc.c so that it returns the total number of processes that are SLEEPING or RUNNING.
2. Modify ps.c so that it prints out a message telling the total number of SLEEPING and RUNNING processes. Copy your code and outputs to your report.

Add name to **syscall.h**:



The screenshot shows a code editor window with the title bar "syscall.h" and the path "~/Desktop/xv6/xv6-public". The editor has several tabs: "syscall.h", "defs.h", "user.h", "sysproc.c", "syscall.c", "proc.c", and "ps.c". The "syscall.h" tab is active, showing the following code:

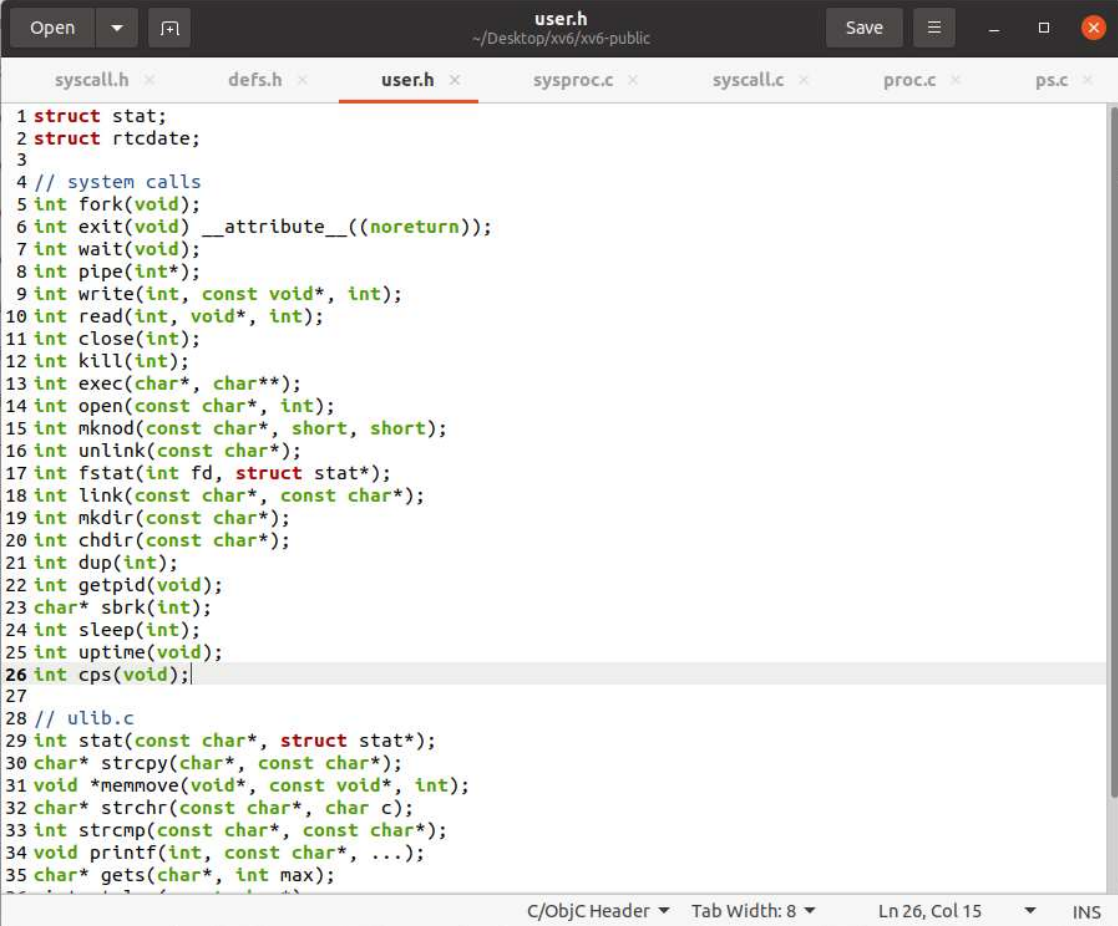
```
1 // System call numbers
2 #define SYS_fork 1
3 #define SYS_exit 2
4 #define SYS_wait 3
5 #define SYS_pipe 4
6 #define SYS_read 5
7 #define SYS_kill 6
8 #define SYS_exec 7
9 #define SYS_fstat 8
10 #define SYS_chdir 9
11 #define SYS_dup 10
12 #define SYS_getpid 11
13 #define SYS_sbrk 12
14 #define 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
```

Add function prototype to defs.h:

```
96 void      picinit(void);
97
98 // pipe.c
99 int        pipealloc(struct file**, struct file**);
100 void       pipeclose(struct pipe*, int);
101 int        piperead(struct pipe*, char*, int);
102 int        pipewrite(struct pipe*, char*, int);
103
104 //PAGEBREAK: 16
105 // proc.c
106 int        cpuid(void);
107 void       exit(void);
108 int        fork(void);
109 int        growproc(int);
110 int        kill(int);
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
125 // swtch.S
126 void       swtch(struct context**, struct context*);
127
128 // spinlock.c
129 void       acquire(struct spinlock*);
130 void       getcallerpcs(void*, uint*);
131 int        holding(struct spinlock*);
```

C/ObjC Header ▾ Tab Width: 8 ▾ Ln 123, Col 27 ▾ INS

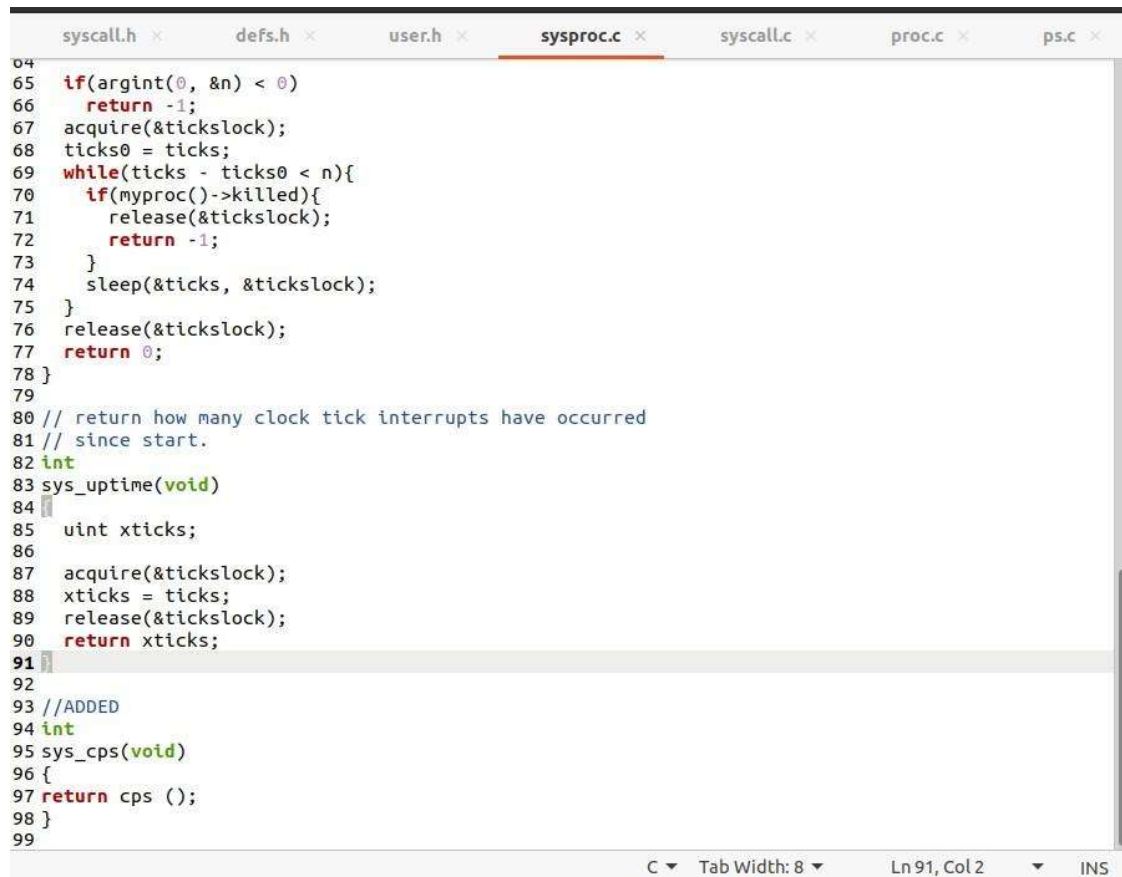
Add function prototype to user.h:



```
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 cps(void);
27
28 // ulib.c
29 int stat(const char*, struct stat*);
30 char* strcpy(char*, const char*);
31 void *memmove(void*, const void*, int);
32 char* strchr(const char*, char c);
33 int strcmp(const char*, const char*);
34 void printf(int, const char*, ...);
35 char* gets(char*, int max);
```

C/ObjC Header Tab Width: 8 Ln 26, Col 15 INS

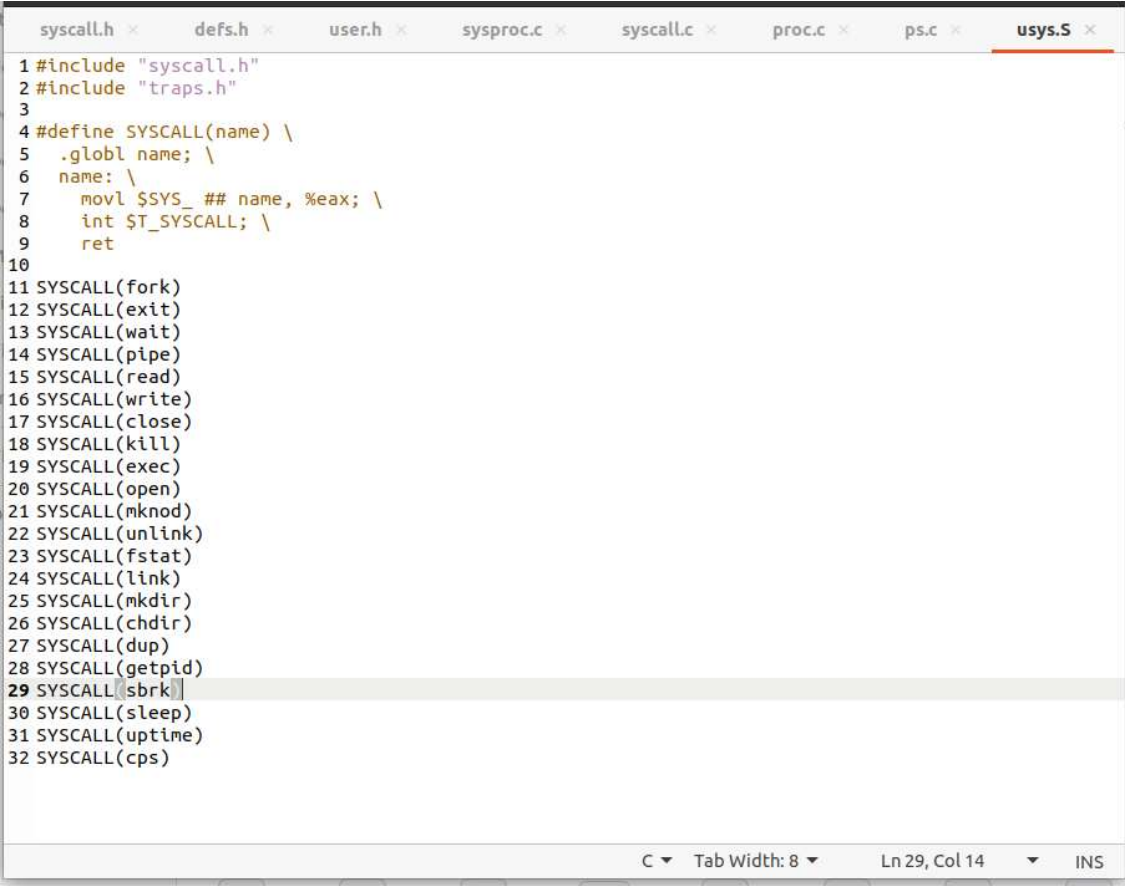
Add function call to **sysproc.c**:



```
syscall.h x  defs.h x  user.h x  sysproc.c x  syscall.c x  proc.c x  ps.c x
04
65  if(argint(0, &n) < 0)
66      return -1;
67  acquire(&tickslock);
68  ticks0 = ticks;
69  while(ticks - ticks0 < n){
70      if(myproc()->killed){
71          release(&tickslock);
72          return -1;
73      }
74      sleep(&ticks, &tickslock);
75  }
76  release(&tickslock);
77  return 0;
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
87     acquire(&tickslock);
88     xticks = ticks;
89     release(&tickslock);
90     return xticks;
91 }
92
93 //ADDED
94 int
95 sys_cps(void)
96 {
97     return cps ();
98 }
99

C  Tab Width: 8  Ln 91, Col 2  INS
```

Add call to usys.S:

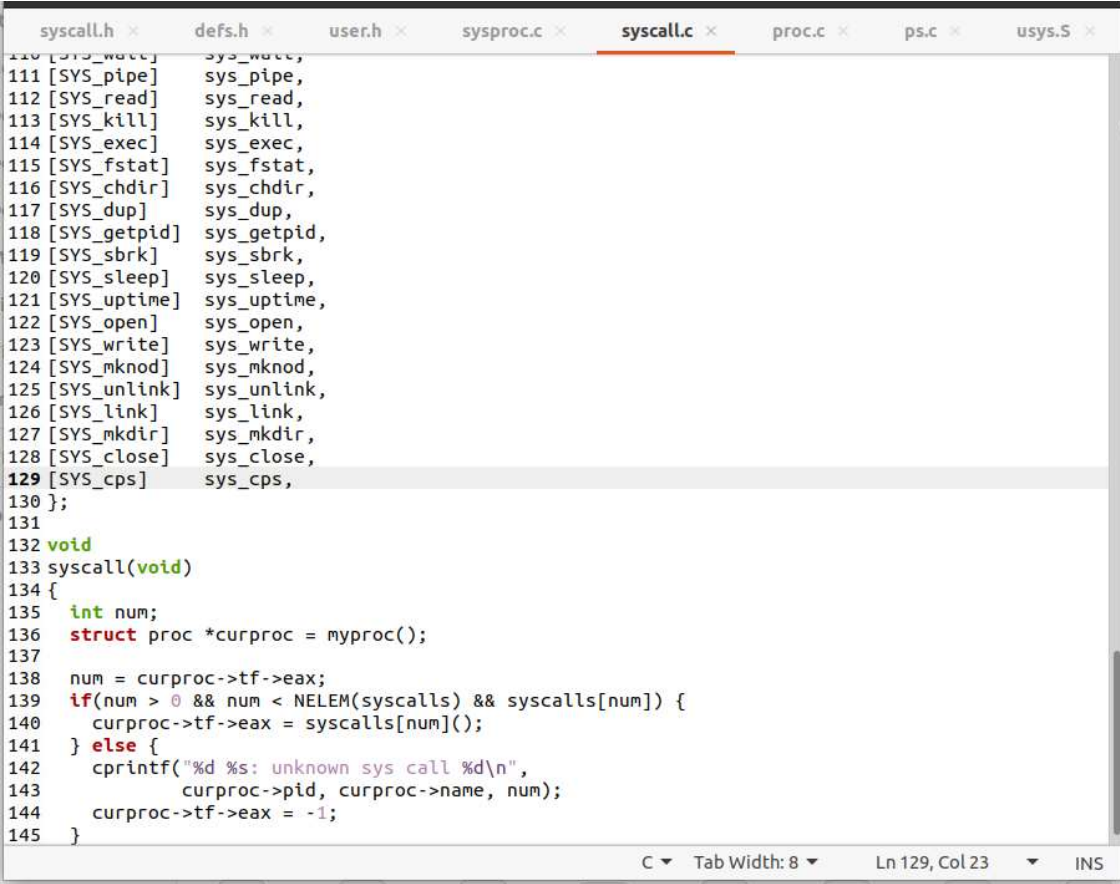


```
syscall.h x  defs.h x  user.h x  sysproc.c x  syscall.c x  proc.c x  ps.c x  usys.S x
1 #include "syscall.h"
2 #include "traps.h"
3
4 #define SYSCALL(name) \
5     .globl name; \
6     name: \
7         movl $SYS_ ## name, %eax; \
8         int $T_SYSCALL; \
9         ret
10
11 SYSCALL(fork)
12 SYSCALL(exit)
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(cps)

C Tab Width: 8 Ln 29, Col 14 INS
```



Add call to syscall.c:

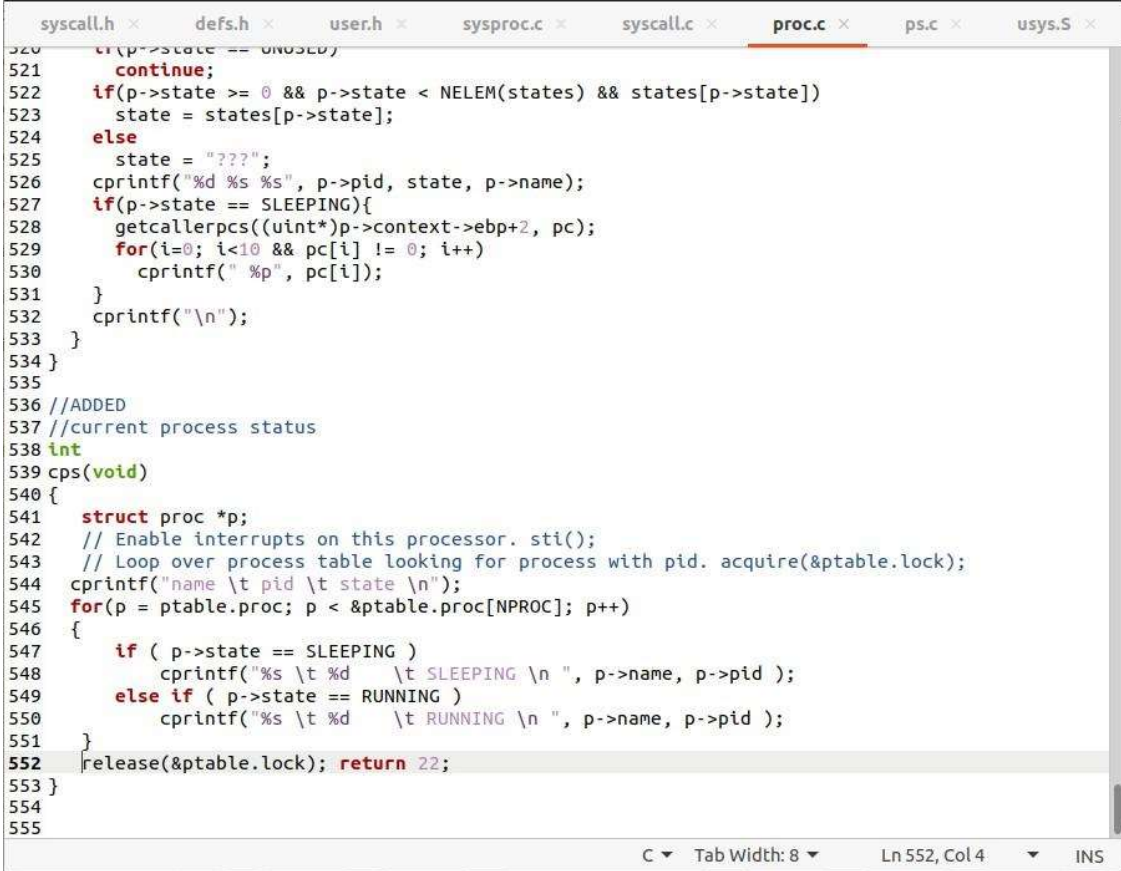


The screenshot shows a code editor with several tabs open: syscall.h, defs.h, user.h, sysproc.c, syscall.c (active), proc.c, ps.c, and usys.S. The active file, syscall.c, contains a list of system calls from line 110 to 128, each mapped to a sys\_\* function. Line 129 has been added, mapping [SYS\_cps] to sys\_cps. The code ends with a closing brace and semicolon on line 130. Below this, a void syscall(void) function is defined, starting on line 132. The function body begins on line 134 with a local int num variable and a struct proc \*curproc assignment. It then enters a loop on line 138 where it checks if the system call number is valid and calls the corresponding function. If the number is invalid, it prints an error message and returns -1. The status bar at the bottom indicates the current position is at line 129, column 23.

```
110 [SYS_wait]    sys_wait,
111 [SYS_pipe]    sys_pipe,
112 [SYS_read]    sys_read,
113 [SYS_kill]    sys_kill,
114 [SYS_exec]    sys_exec,
115 [SYS_fstat]   sys_fstat,
116 [SYS_chdir]   sys_chdir,
117 [SYS_dup]     sys_dup,
118 [SYS_getpid]  sys_getpid,
119 [SYS_sbrk]    sys_sbrk,
120 [SYS_sleep]   sys_sleep,
121 [SYS_uptime]  sys_uptime,
122 [SYS_open]    sys_open,
123 [SYS_write]   sys_write,
124 [SYS_mknod]   sys_mknod,
125 [SYS_unlink]  sys_unlink,
126 [SYS_link]    sys_link,
127 [SYS_mkdir]   sys_mkdir,
128 [SYS_close]   sys_close,
129 [SYS_cps]     sys_cps,
130 };
131
132 void
133 syscall(void)
134 {
135     int num;
136     struct proc *curproc = myproc();
137
138     num = curproc->tf->eax;
139     if(num > 0 && num < NELEM(syscalls) && syscalls[num]) {
140         curproc->tf->eax = syscalls[num]();
141     } else {
142         cprintf("%d %s: unknown sys call %d\n",
143             curproc->pid, curproc->name, num);
144         curproc->tf->eax = -1;
145     }
```

C Tab Width: 8 Ln 129, Col 23 INS

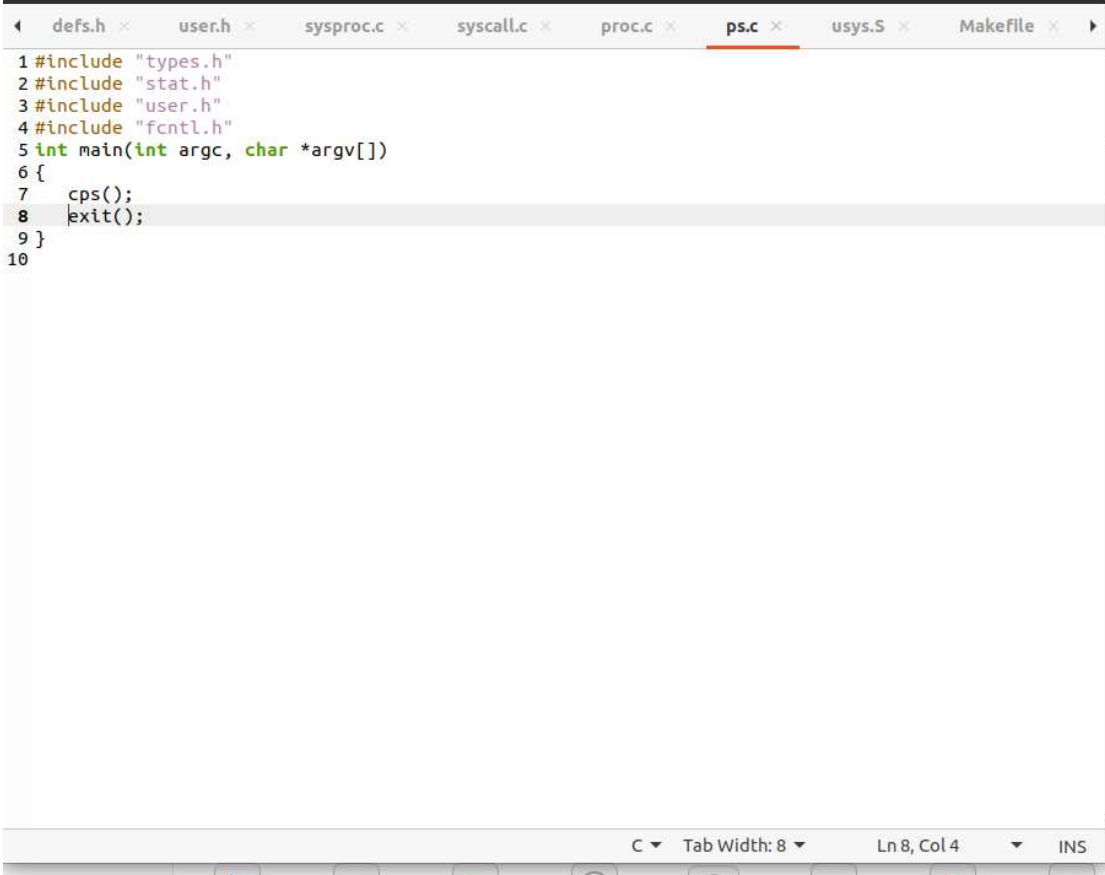
Add code to proc.c:



```
520     if(p->state == UNDEAD)
521         continue;
522     if(p->state >= 0 && p->state < NELEM(states) && states[p->state])
523         state = states[p->state];
524     else
525         state = "???";
526     cprintf("%d %s %s", p->pid, state, p->name);
527     if(p->state == SLEEPING){
528         getcallerpcs((uint*)p->context->ebp+2, pc);
529         for(i=0; i<10 && pc[i] != 0; i++)
530             cprintf(" %p", pc[i]);
531     }
532     cprintf("\n");
533 }
534 }
535
536 //ADDED
537 //current process status
538 int
539 cps(void)
540 {
541     struct proc *p;
542     // Enable interrupts on this processor. sti();
543     // Loop over process table looking for process with pid. acquire(&ptable.lock);
544     cprintf("name \t pid \t state \n");
545     for(p = ptable.proc; p < &ptable.proc[NPROC]; p++)
546     {
547         if ( p->state == SLEEPING )
548             cprintf("%s \t %d \t SLEEPING \n ", p->name, p->pid );
549         else if ( p->state == RUNNING )
550             cprintf("%s \t %d \t RUNNING \n ", p->name, p->pid );
551     }
552     release(&ptable.lock); return 22;
553 }
554
555
```

C Tab Width: 8 Ln 552, Col 4 INS

Create testing file ps.c:

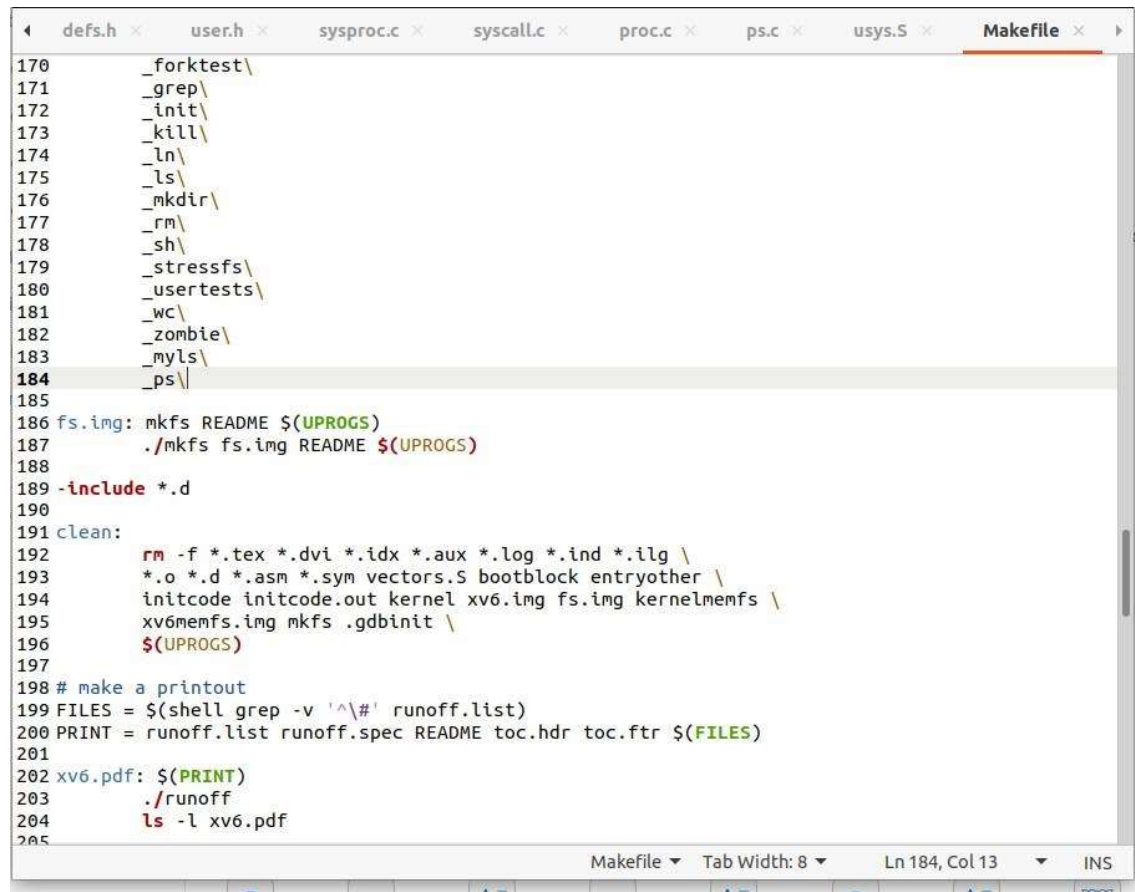


The screenshot shows a code editor with several tabs at the top: `defs.h`, `user.h`, `sysproc.c`, `syscall.c`, `proc.c`, `ps.c` (which is the active tab), `usys.S`, and `Makefile`. The `ps.c` file contains the following code:

```
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     cps();
8     exit();
9 }
10
```

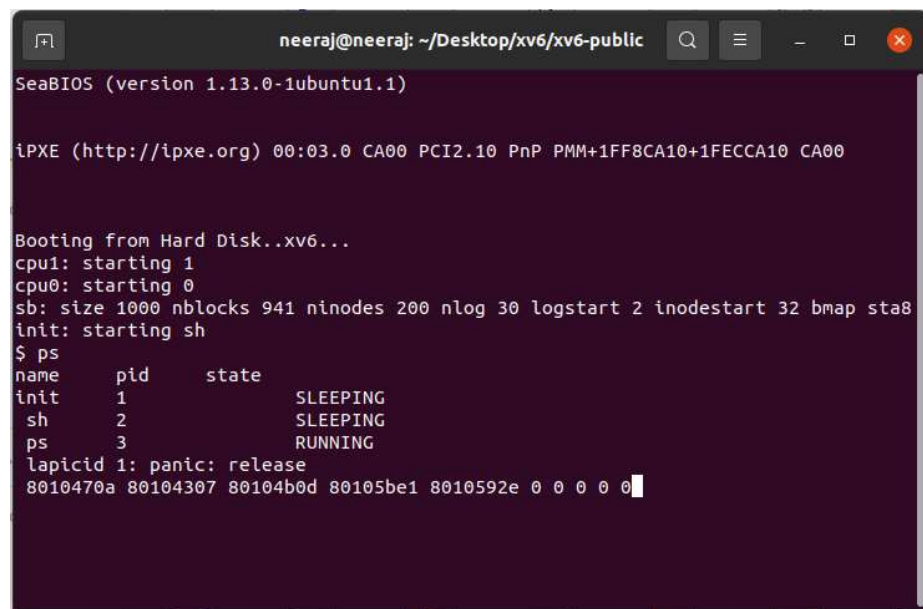
The status bar at the bottom of the editor indicates the current cursor position as "Ln 8, Col 4" and the input mode as "INS".

Modify Makefile :



```
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
186 fs.img: mkfs README $(UPROGS)
187     ./mkfs fs.img README $(UPROGS)
188
189 -include *.d
190
191 clean:
192     rm -f *.tex *.dvi *.idx *.aux *.log *.ind *.ilg \
193         *.o *.d *.asm *.sym vectors.S bootblock entryother \
194         initcode initcode.out kernel xv6.img fs.img kernelmemfs \
195         xv6memfs.img mkfs .gdbinit \
196         $(UPROGS)
197
198 # make a printout
199 FILES = $(shell grep -v '^#' runoff.list)
200 PRINT = runoff.list runoff.spec README toc.hdr toc.ftr $(FILES)
201
202 xv6.pdf: $(PRINT)
203     ./runoff
204     ls -l xv6.pdf
205
```

-final output by “ps” 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
$ ps
name    pid    state
init     1      SLEEPING
sh       2      SLEEPING
ps       3      RUNNING
laptopid 1: panic: release
8010470a 80104307 80104b0d 80105be1 8010592e 0 0 0 0 0
```