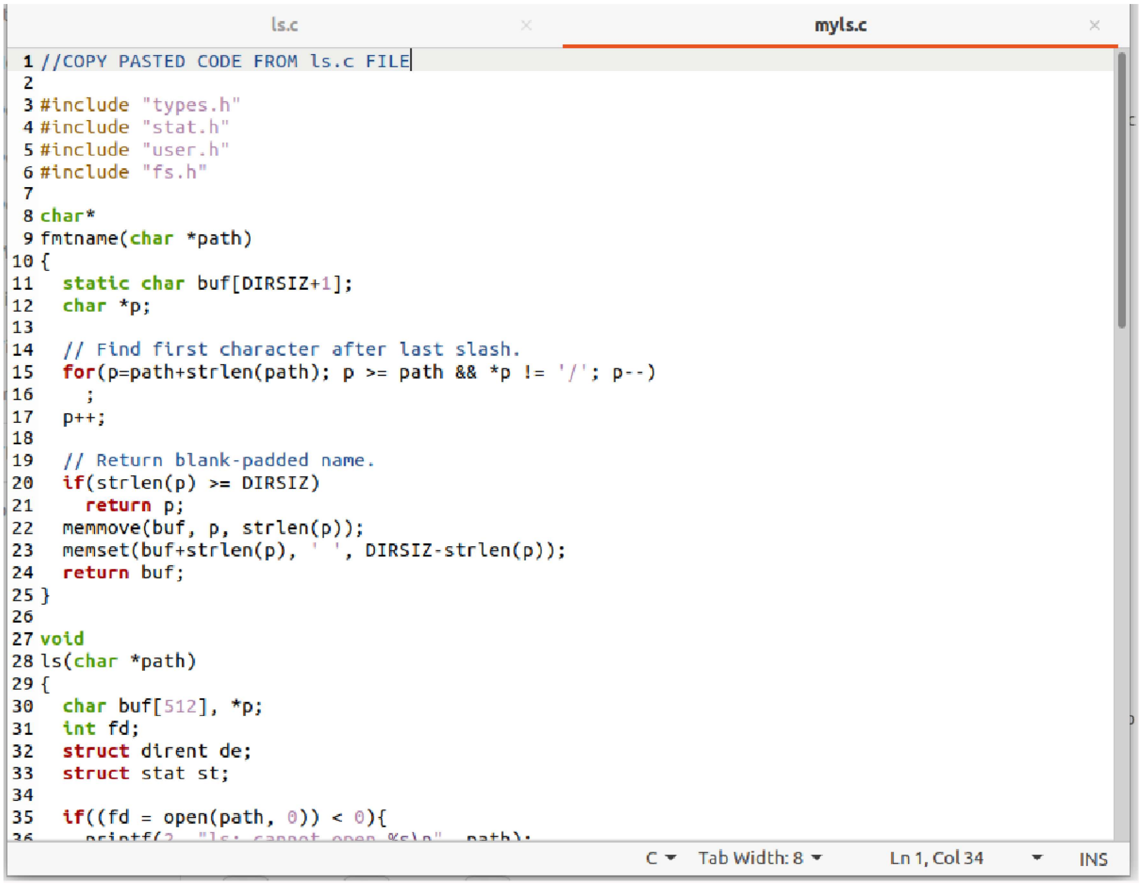
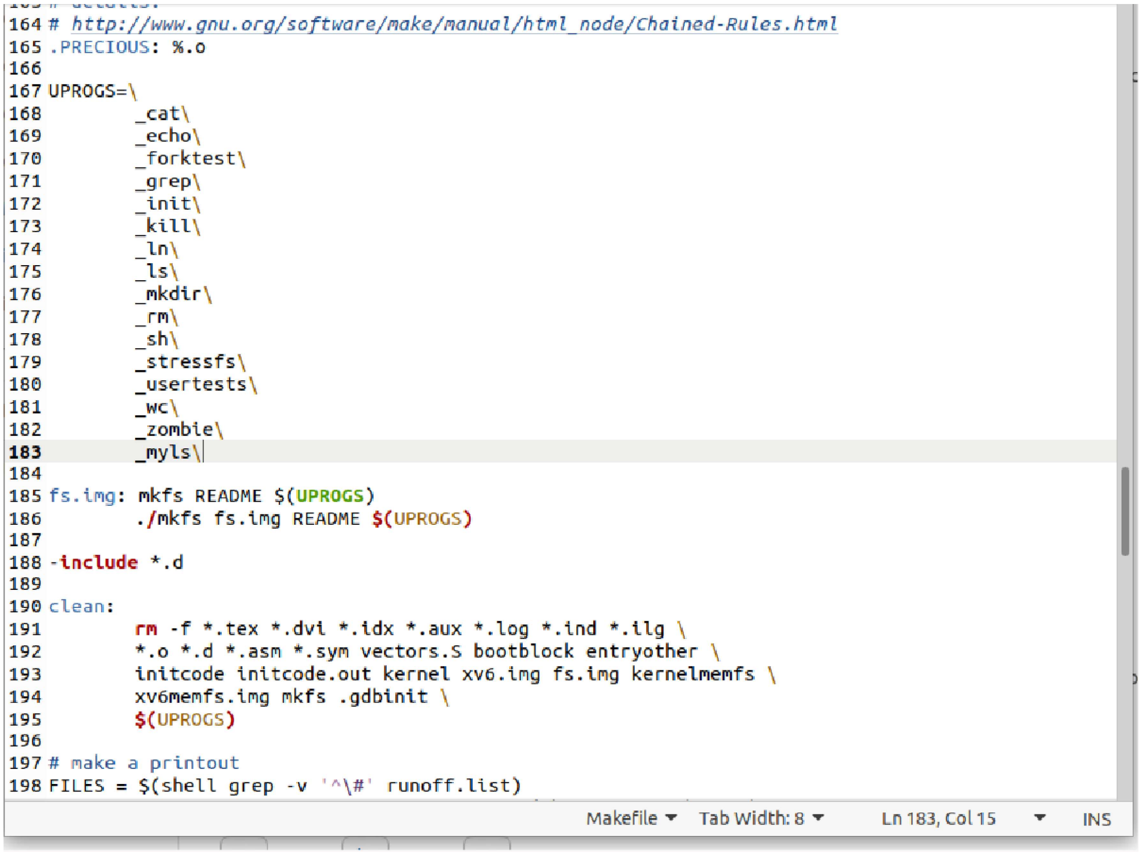
Question 1. You need to copy of command “ls” as “myls” and modify the make file accordingly to run command “myls” in terminal.



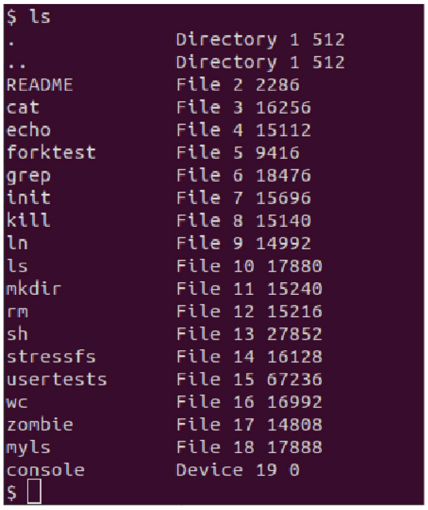
```c
//COPY PASTED CODE FROM ls.c FILE

#include "types.h"
#include "stat.h"
#include "user.h"
#include "fs.h"

char*
fmtname(char *path)
{
  static char buf[DIRSIZ+1];
  char *p;

  // Find first character after last slash.
  for(p=path+strlen(path); p >= path && *p != '/'; p--)
    ;
  p++;

  // Return blank-padded name.
  if(strlen(p) >= DIRSIZ)
    return p;
  memmove(buf, p, strlen(p));
  memset(buf+strlen(p), ' ', DIRSIZ-strlen(p));
  return buf;
}

void
ls(char *path)
{
  char buf[512], *p;
  int fd;
  struct dirent de;
  struct stat st;

  if((fd = open(path, 0)) < 0){
    printf(2, "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



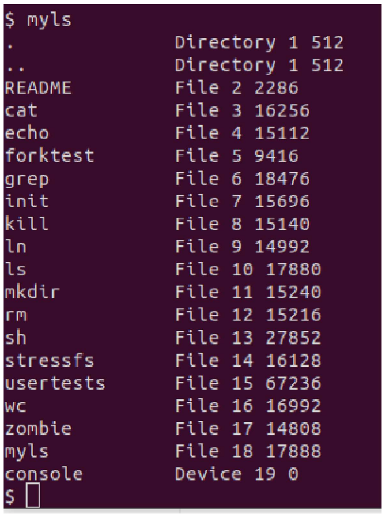
```
# http://www.gnu.org/software/make/manual/html_node/Chained-Rules.html
.PRECIOUS: %.o

UPROGS=\
	_cat\
	_echo\
	_forktest\
	_grep\
	_init\
	_kill\
	_ln\
	_ls\
	_mkdir\
	_rm\
	_sh\
	_stressfs\
	_usertests\
	_wc\
	_zombie\
	_myls\

fs.img: mkfs README $(UPROGS)
	./mkfs fs.img README $(UPROGS)

-include *.d

clean:
	rm -f *.tex *.dvi *.idx *.aux *.log *.ind *.ilg \
	*.o *.d *.asm *.sym vectors.S bootblock entryother \
	initcode initcode.out kernel xv6.img fs.img kernelmemfs \
	xv6memfs.img mkfs .gdbinit \
	$(UPROGS)

# make a printout
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
myls            File 18 17888
console         Device 19 0
$ 
```

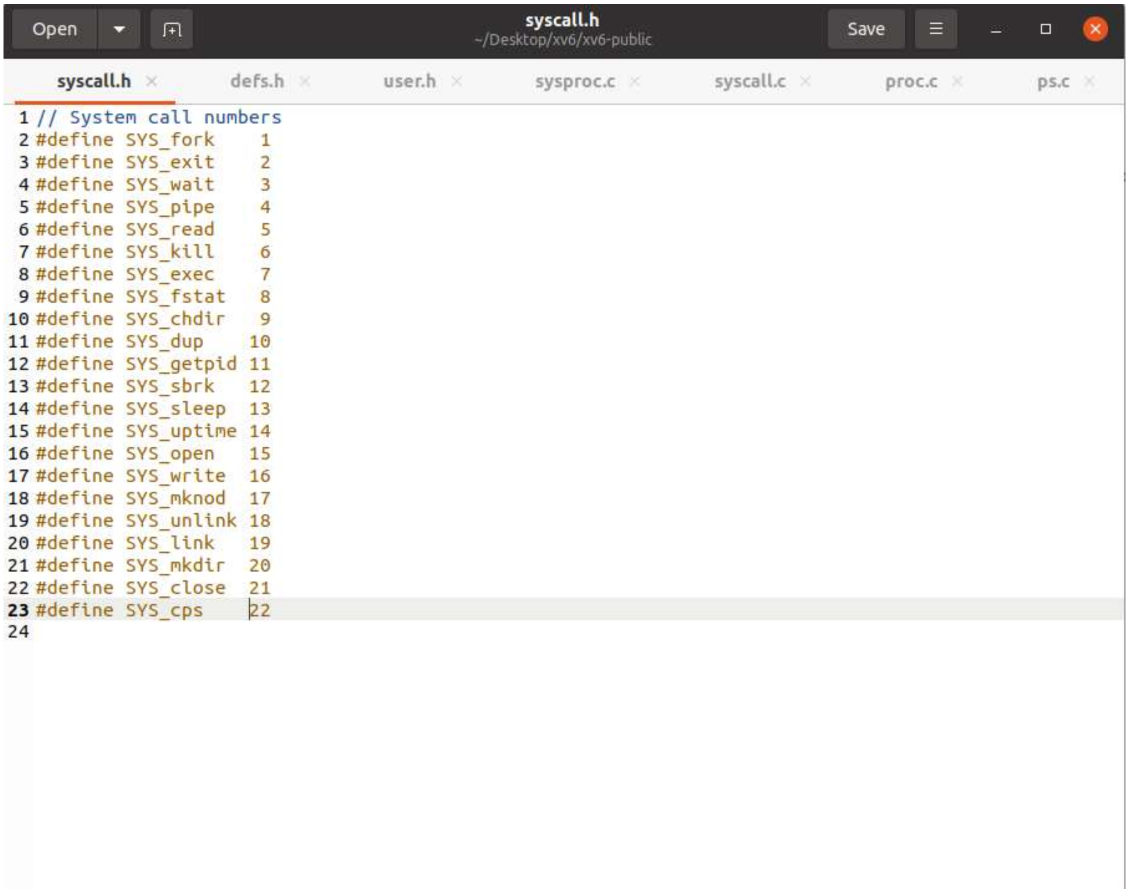
-OUTPUT BY myls COMMAND

Question 2.

1. Modify cps() in proc.c so that it returns the total number of processes that are SLEEPING orRUNNING.

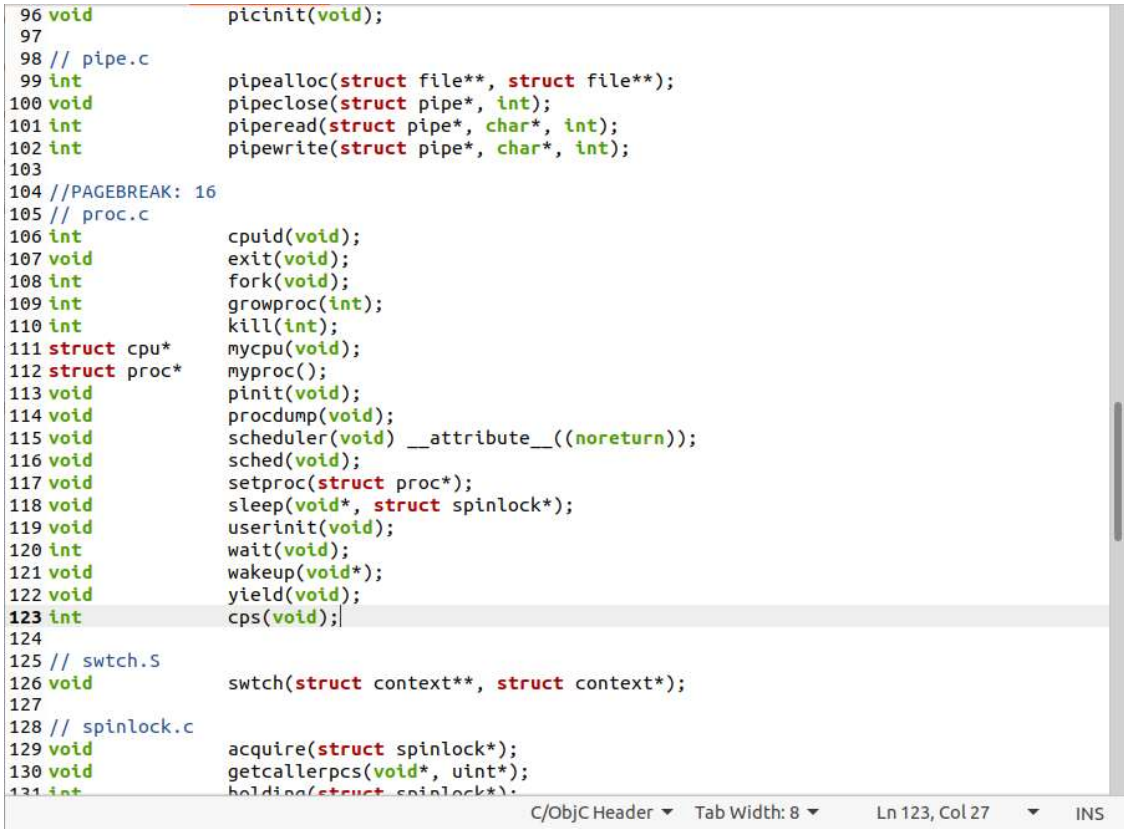
2. Modify ps.c so that it prints outa message telling the total number of SLEEPING and RUNNING processes. Copy your code and outputs to your report.

Add name to **syscall.h:**



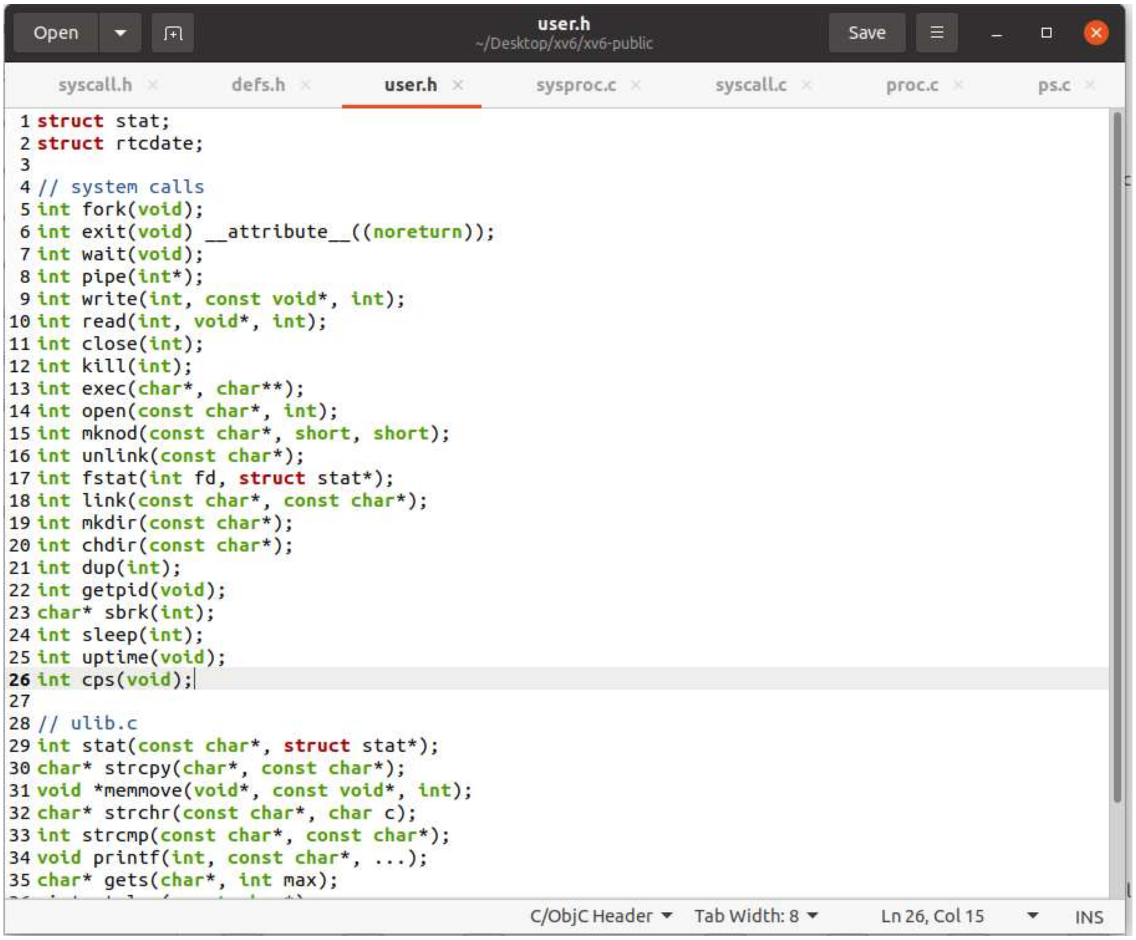
```
// System call numbers
#define SYS_fork    1
#define SYS_exit    2
#define SYS_wait    3
#define SYS_pipe    4
#define SYS_read    5
#define SYS_kill    6
#define SYS_exec    7
#define SYS_fstat   8
#define SYS_chdir   9
#define SYS_dup    10
#define SYS_getpid 11
#define SYS_sbrk   12
#define SYS_sleep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

Add function prototype to defs.h:



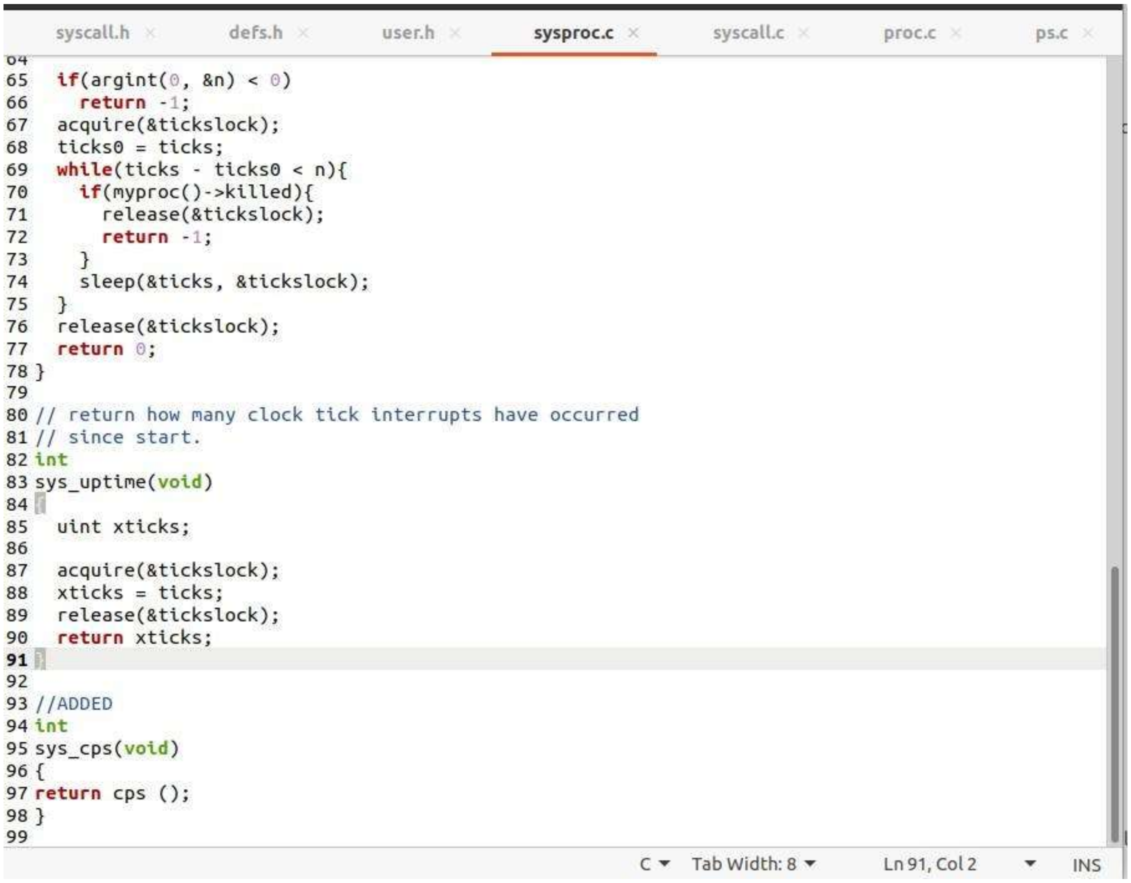
```
void            picinit(void);

// pipe.c
int             pipealloc(struct file**, struct file**);
void            pipeclose(struct pipe*, int);
int             piperead(struct pipe*, char*, int);
int             pipewrite(struct pipe*, char*, int);

//PAGEBREAK: 16
// proc.c
int             cpuid(void);
void            exit(void);
int             fork(void);
int             growproc(int);
int             kill(int);
struct cpu*     mycpu(void);
struct proc*    myproc();
void            pinit(void);
void            procdump(void);
void            scheduler(void) __attribute__((noreturn));
void            sched(void);
void            setproc(struct proc*);
void            sleep(void*, struct spinlock*);
void            userinit(void);
int             wait(void);
void            wakeup(void*);
void            yield(void);
int             cps(void);

// swtch.S
void            swtch(struct context**, struct context*);

// spinlock.c
void            acquire(struct spinlock*);
void            getcallerpcs(void*, uint*);
int             holding(struct spinlock*);
```

Add function prototype to user.h:



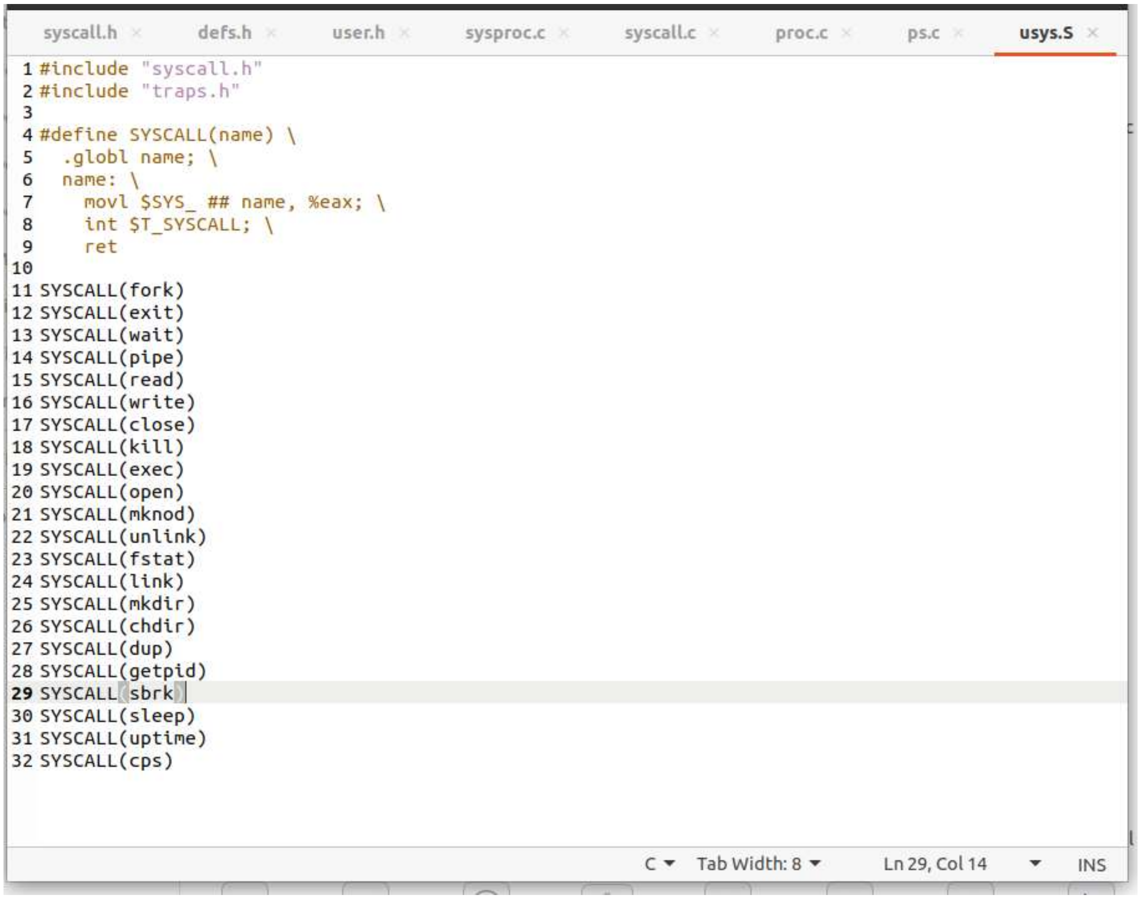
```
struct stat;
struct rtcdate;

// system calls
int fork(void);
int exit(void) __attribute__((noreturn));
int wait(void);
int pipe(int*);
int write(int, const void*, int);
int read(int, void*, int);
int close(int);
int kill(int);
int exec(char*, char**);
int open(const char*, int);
int mknod(const char*, short, short);
int unlink(const char*);
int fstat(int fd, struct stat*);
int link(const char*, const char*);
int mkdir(const char*);
int chdir(const char*);
int dup(int);
int getpid(void);
char* sbrk(int);
int sleep(int);
int uptime(void);
int cps(void);

// ulib.c
int stat(const char*, struct stat*);
char* strcpy(char*, const char*);
void *memmove(void*, const void*, int);
char* strchr(const char*, char c);
int strcmp(const char*, const char*);
void printf(int, const char*, ...);
char* gets(char*, int max);
```

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



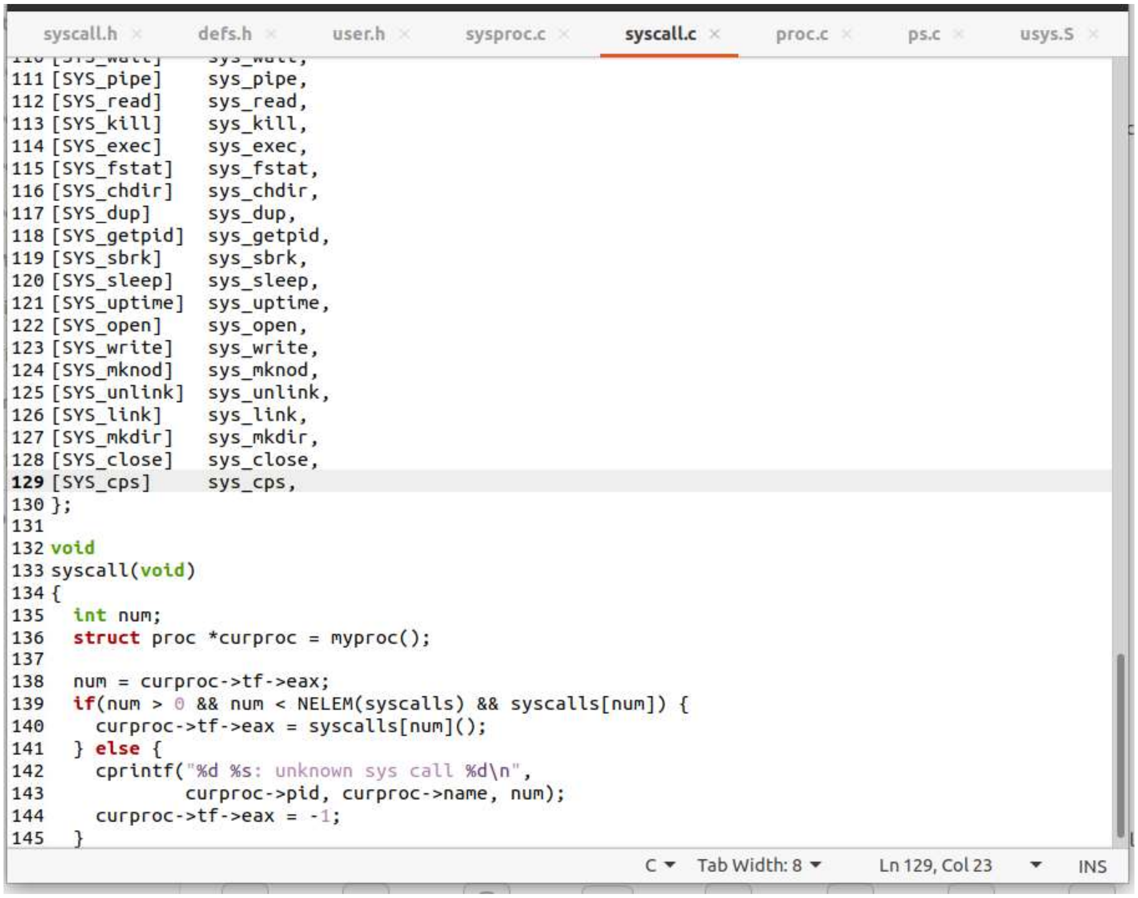
```
  if(argint(0, &n) < 0)
    return -1;
  acquire(&tickslock);
  ticks0 = ticks;
  while(ticks - ticks0 < n){
    if(myproc()->killed){
      release(&tickslock);
      return -1;
    }
    sleep(&ticks, &tickslock);
  }
  release(&tickslock);
  return 0;
}

// return how many clock tick interrupts have occurred
// since start.
int
sys_uptime(void)
{
  uint xticks;

  acquire(&tickslock);
  xticks = ticks;
  release(&tickslock);
  return xticks;
}

//ADDED
int
sys_cps(void)
{
return cps ();
}

```

Add call to usys.S:



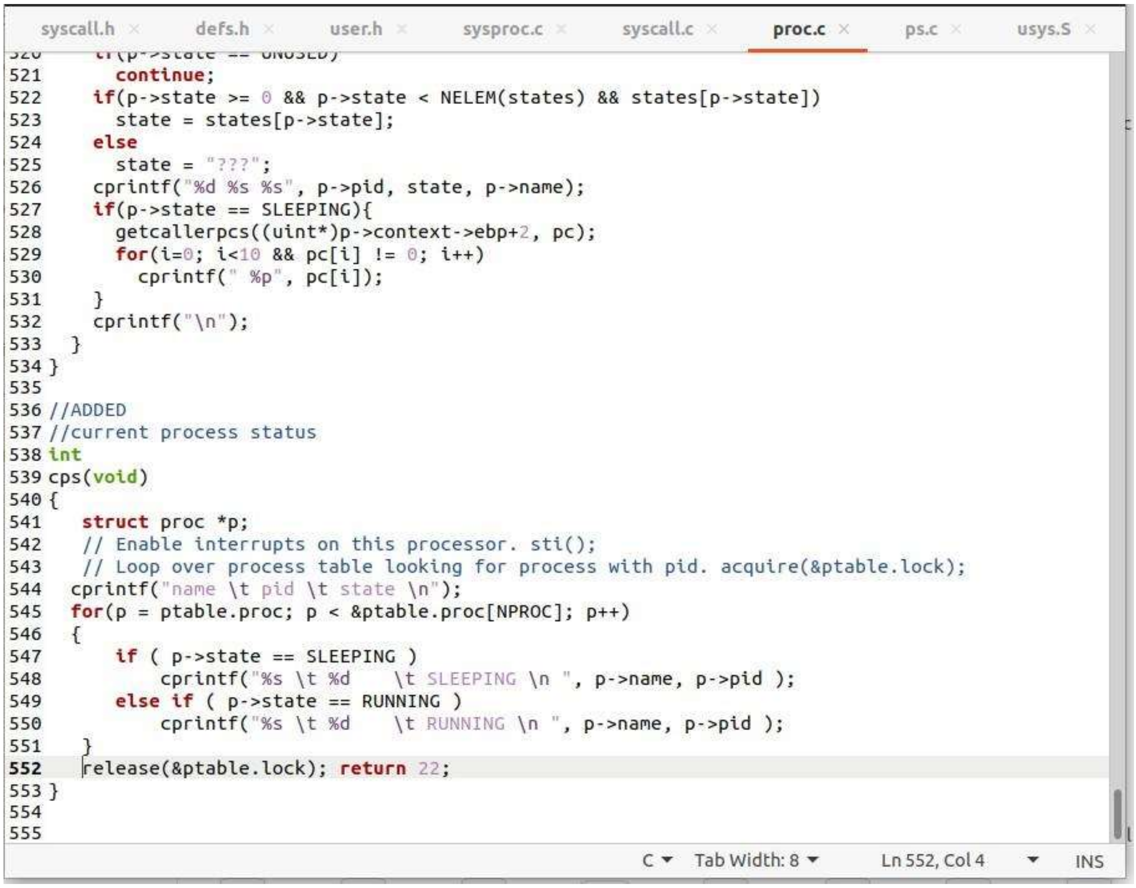
```
#include "syscall.h"
#include "traps.h"

#define SYSCALL(name) \
  .globl name; \
  name: \
    movl $SYS_ ## name, %eax; \
    int $T_SYSCALL; \
    ret

SYSCALL(fork)
SYSCALL(exit)
SYSCALL(wait)
SYSCALL(pipe)
SYSCALL(read)
SYSCALL(write)
SYSCALL(close)
SYSCALL(kill)
SYSCALL(exec)
SYSCALL(open)
SYSCALL(mknod)
SYSCALL(unlink)
SYSCALL(fstat)
SYSCALL(link)
SYSCALL(mkdir)
SYSCALL(chdir)
SYSCALL(dup)
SYSCALL(getpid)
SYSCALL(sbrk)
SYSCALL(sleep)
SYSCALL(uptime)
SYSCALL(cps)
```

Add call to syscall.c:



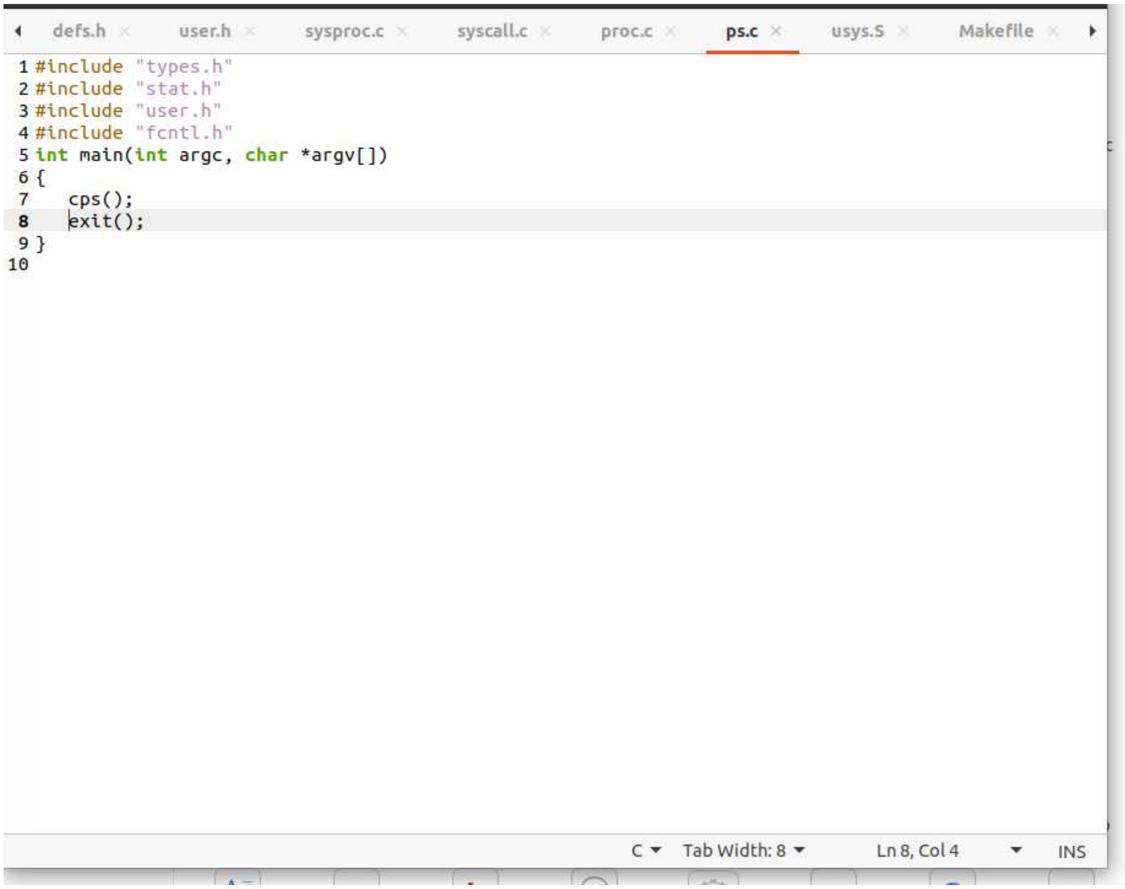
```
[SYS_wait]    sys_wait,
[SYS_pipe]    sys_pipe,
[SYS_read]    sys_read,
[SYS_kill]    sys_kill,
[SYS_exec]    sys_exec,
[SYS_fstat]   sys_fstat,
[SYS_chdir]   sys_chdir,
[SYS_dup]     sys_dup,
[SYS_getpid]  sys_getpid,
[SYS_sbrk]    sys_sbrk,
[SYS_sleep]   sys_sleep,
[SYS_uptime]  sys_uptime,
[SYS_open]    sys_open,
[SYS_write]   sys_write,
[SYS_mknod]   sys_mknod,
[SYS_unlink]  sys_unlink,
[SYS_link]    sys_link,
[SYS_mkdir]   sys_mkdir,
[SYS_close]   sys_close,
[SYS_cps]     sys_cps,
};

void
syscall(void)
{
  int num;
  struct proc *curproc = myproc();

  num = curproc->tf->eax;
  if(num > 0 && num < NELEM(syscalls) && syscalls[num]) {
    curproc->tf->eax = syscalls[num]();
  } else {
    cprintf("%d %s: unknown sys call %d\n",
            curproc->pid, curproc->name, num);
    curproc->tf->eax = -1;
  }
```

Add code to proc.c:



```
    if(p->state == UNUSED)
      continue;
    if(p->state >= 0 && p->state < NELEM(states) && states[p->state])
      state = states[p->state];
    else
      state = "???";
    cprintf("%d %s %s", p->pid, state, p->name);
    if(p->state == SLEEPING){
      getcallerpcs((uint*)p->context->ebp+2, pc);
      for(i=0; i<10 && pc[i] != 0; i++)
        cprintf(" %p", pc[i]);
    }
    cprintf("\n");
  }
}

//ADDED
//current process status
int
cps(void)
{
   struct proc *p;
   // Enable interrupts on this processor. sti();
   // Loop over process table looking for process with pid. acquire(&ptable.lock);
  cprintf("name \t pid \t state \n");
  for(p = ptable.proc; p < &ptable.proc[NPROC]; p++)
  {
      if ( p->state == SLEEPING )
          cprintf("%s \t %d    \t SLEEPING \n ", p->name, p->pid );
      else if ( p->state == RUNNING )
          cprintf("%s \t %d    \t RUNNING \n ", p->name, p->pid );
   }
   release(&ptable.lock); return 22;
}
```

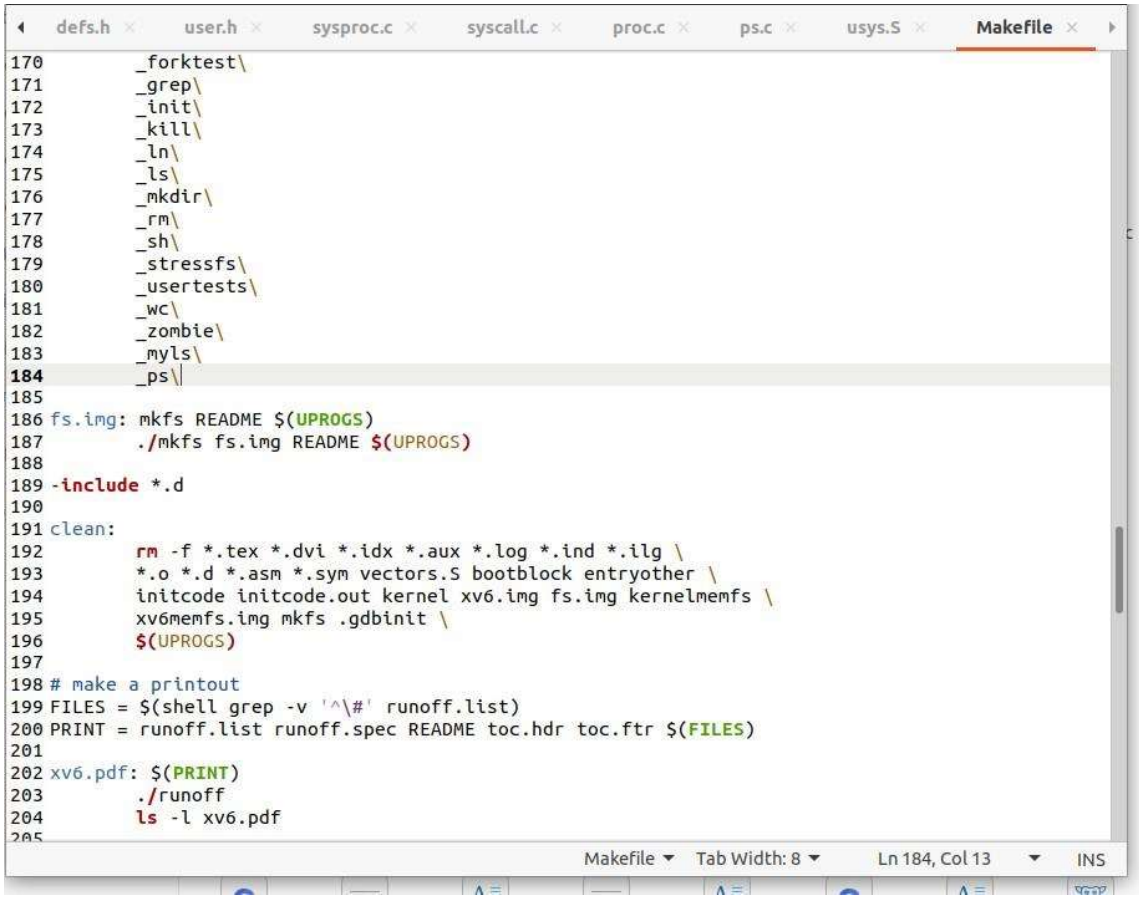
Create testing file ps.c:



```
#include "types.h"
#include "stat.h"
#include "user.h"
#include "fcntl.h"
int main(int argc, char *argv[])
{
    cps();
    exit();
}

```

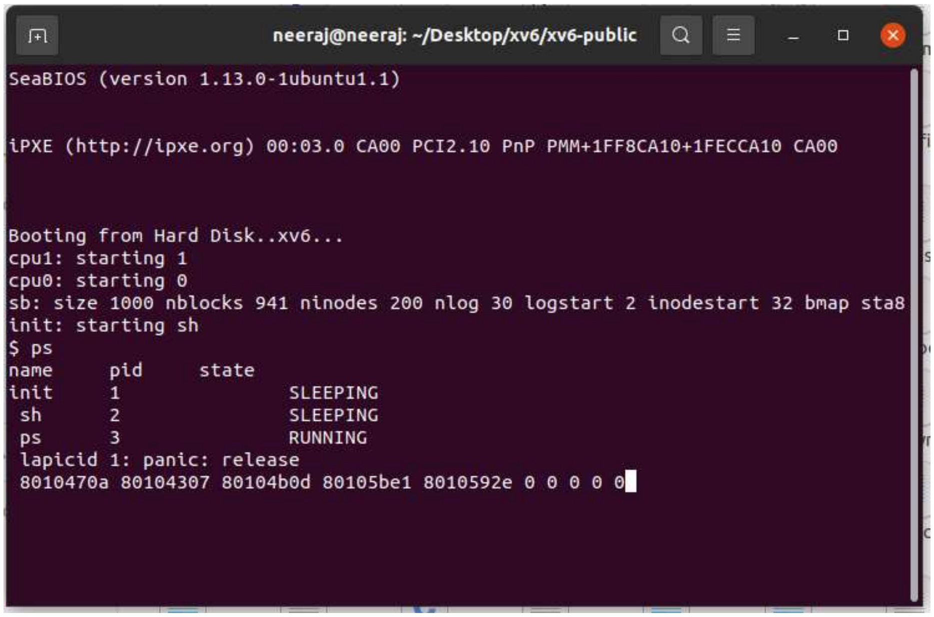
C Tab Width: 8 Ln 8, Col 4 INS

Modify Makefile :



```
	_forktest\
	_grep\
	_init\
	_kill\
	_ln\
	_ls\
	_mkdir\
	_rm\
	_sh\
	_stressfs\
	_usertests\
	_wc\
	_zombie\
	_myls\
	_ps\

fs.img: mkfs README $(UPROGS)
	./mkfs fs.img README $(UPROGS)

-include *.d

clean:
	rm -f *.tex *.dvi *.idx *.aux *.log *.ind *.ilg \
	*.o *.d *.asm *.sym vectors.S bootblock entryother \
	initcode initcode.out kernel xv6.img fs.img kernelmemfs \
	xv6memfs.img mkfs .gdbinit \
	$(UPROGS)

# make a printout
FILES = $(shell grep -v '^\#' runoff.list)
PRINT = runoff.list runoff.spec README toc.hdr toc.ftr $(FILES)

xv6.pdf: $(PRINT)
	./runoff
	ls -l xv6.pdf
```

-final output by “ps” command:



```
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
 lapicid 1: panic: release
 8010470a 80104307 80104b0d 80105be1 8010592e 0 0 0 0 0
```