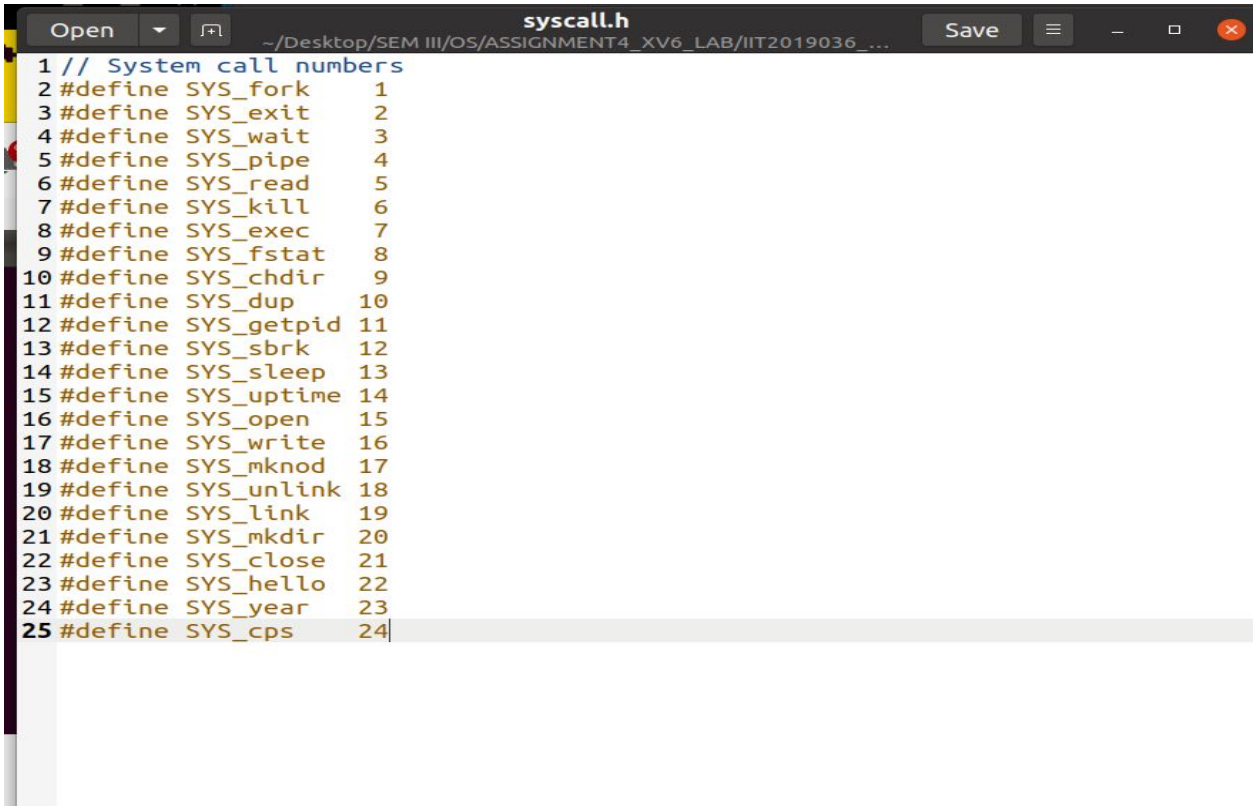
ADDING OF ps SYSTEM CALL on XV6

The ps (i.e., process status) command is used to provide information about the currently running processes, including their process identification numbers (PIDs).

This is a summary of steps, of all I did to add the system call ps and run it on xv6

STEP 1:
Adding the system call in syscall.h, hence added the system call cps with the corresponding number. Since, System call interface maintains the table of all the system call and associates each with a number

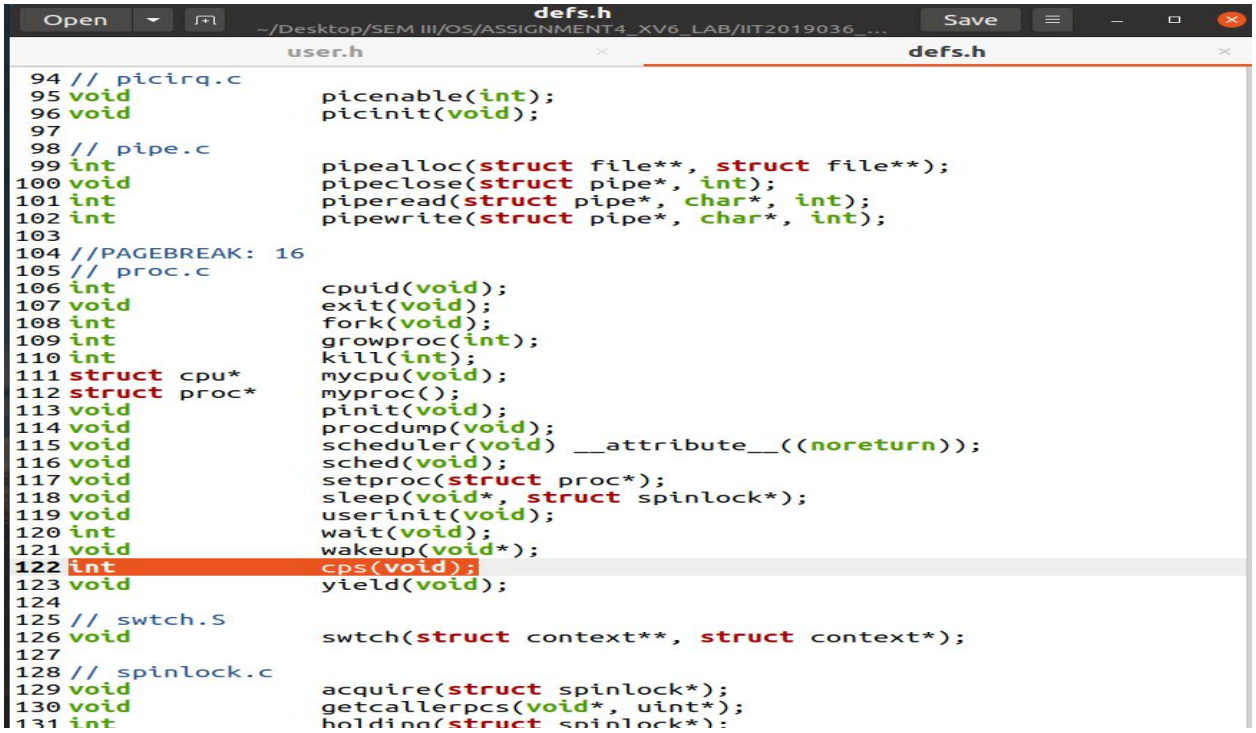


STEP2:

In the struct proc in the proc.h file, add a new attribute ‘priority’ of int data type.

STEP3 :

Next, we have to include the declaration of these functions in ***defs.h*** and ***users.h*** files.




```
 94 // picirq.c
 95 void                picenable(int);
 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 int                 cps(void);
123 void                yield(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*);
```

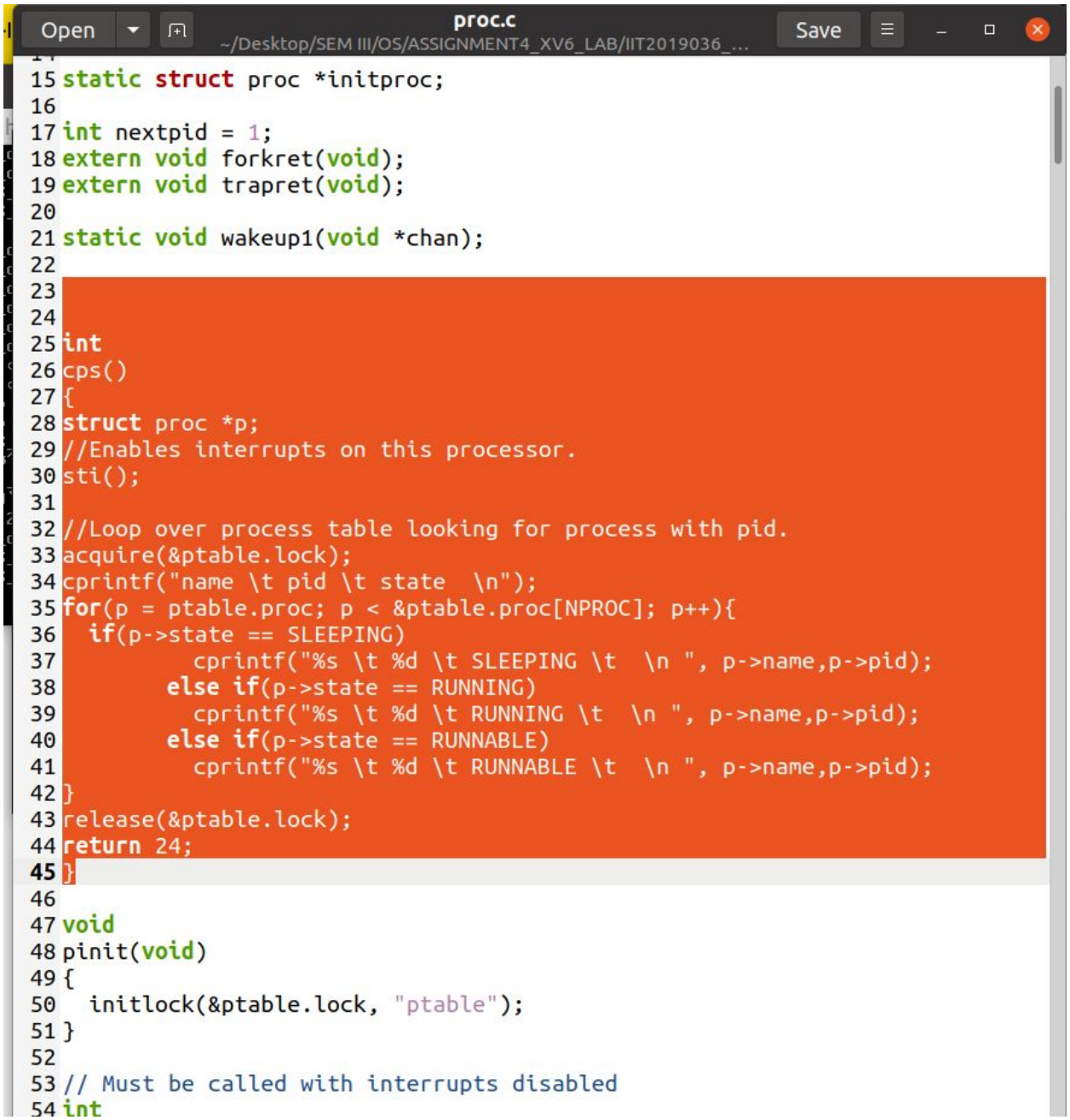
Open ▾ | user.h | ~/Desktop/SEM III/OS/ASSIGNMENT4_XV6_LAB/IIT2019036_... | Save

```
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 hello(void);
26 int year(void);
27 int cps(void);
28 int uptime(void);
29
30 // ulib.c
31 int stat(const char*, struct stat*);
32 char* strcpy(char*, const char*);
33 void *memmove(void*, const void*, int);
34 char* strchr(const char*, char c);
35 int strcmp(const char*, const char*);
36 void printf(int, const char*, ...);
37 char* gets(char*, int max);
38 uint strlen(const char*);
39 void* memset(void*, int, uint);
40 void* malloc(uint);
41 void free(void*);
```

STEP4:

we have to include the definition of the cps function in ***proc.c***

The cps function will include the acquiring of the lock and after that, the piece of code that will print the process states, then finally the lock will get released.




```
15 static struct proc *initproc;
16
17 int nextpid = 1;
18 extern void forkret(void);
19 extern void trapret(void);
20
21 static void wakeup1(void *chan);
22
23
24
25 int
26 cps()
27 {
28 struct proc *p;
29 //Enables interrupts on this processor.
30 sti();
31
32 //Loop over process table looking for process with pid.
33 acquire(&ptable.lock);
34 cprintf("name \t pid \t state  \n");
35 for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
36   if(p->state == SLEEPING)
37           cprintf("%s \t %d \t SLEEPING \t  \n ", p->name,p->pid);
38         else if(p->state == RUNNING)
39           cprintf("%s \t %d \t RUNNING \t  \n ", p->name,p->pid);
40         else if(p->state == RUNNABLE)
41           cprintf("%s \t %d \t RUNNABLE \t  \n ", p->name,p->pid);
42 }
43 release(&ptable.lock);
44 return 24;
45 }
46
47 void
48 pinit(void)
49 {
50   initlock(&ptable.lock, "ptable");
51 }
52
53 // Must be called with interrupts disabled
54 int
```

STEP5:

In sysproc.c, we have to define a function in which our cps functions will be called.

STEP6:

We have to make some minor changes in the usys.S file. The '.S' extension indicates that this file has assembly-level code and this file interacts with the hardware of the system.

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(hello)
SYSCALL(year)
SYSCALL(cps)
SYSCALL(uptime)

```

STEP7:

we open the sysproc.c file and add the system call.

STEP8:

we have to create a ps.c file in which our cps function will be called.

ps.c

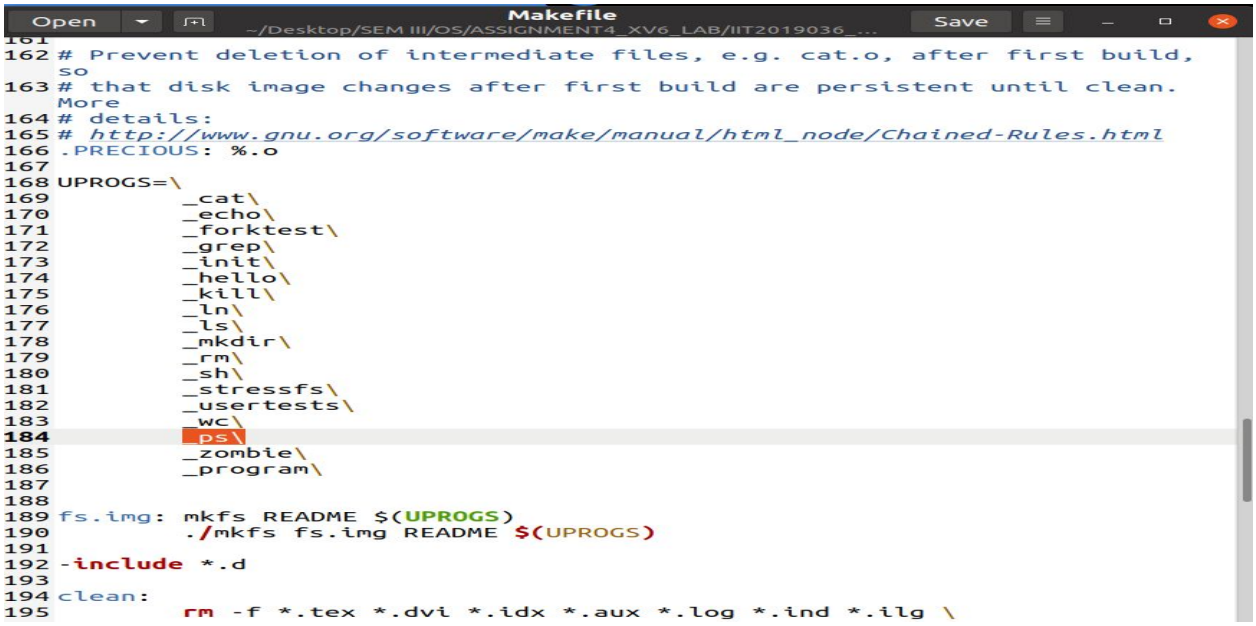
```
#include "types.h"
#include "stat.h"
#include "user.h"
#include "fcntl.h"

int
main()
{
cps();

exit();
}
```

STEP9:

Now we have to make the appropriate changes in the Makefile. In Makefile, under 'UPROGS' and 'EXTRAS'



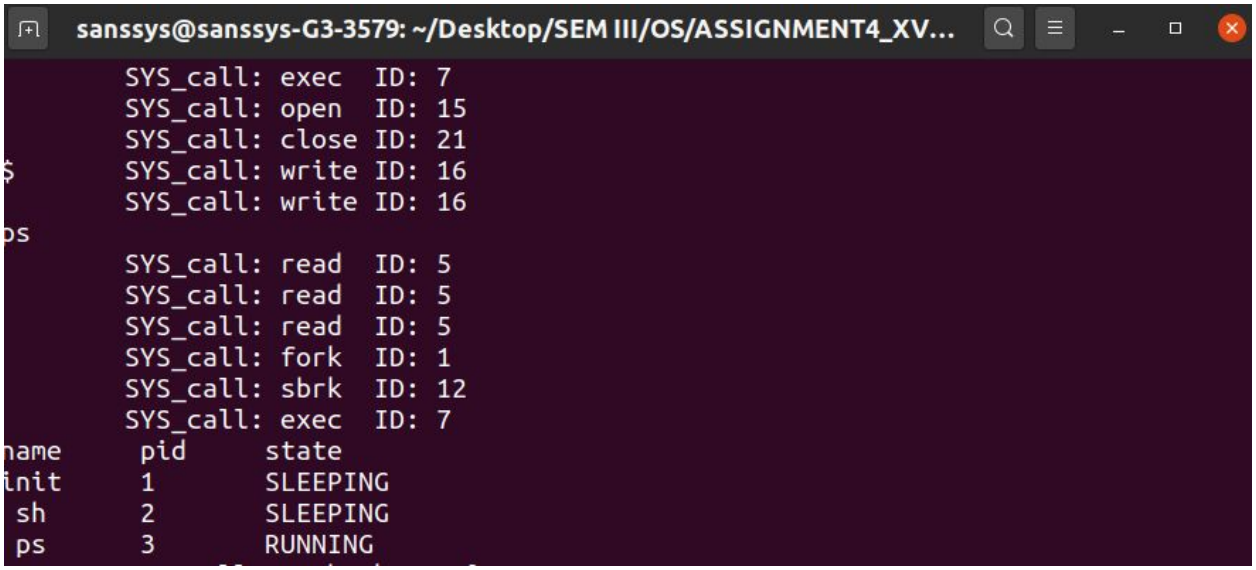
STEP 10:

Now on the terminal, we have to write :

make

make qemu

Then, finally type the command ps, which is the system call that was just added that will detect the running, sleeping states and finally print them on the screen.


sanssys@sanssys-G3-3579: ~/Desktop/SEM III/OS/ASSIGNMENT4_XV...
SYS_call: exec ID: 7
SYS_call: open ID: 15
SYS_call: close ID: 21
SYS_call: write ID: 16
SYS_call: write ID: 16
SYS_call: read ID: 5
SYS_call: read ID: 5
SYS_call: read ID: 5
SYS_call: fork ID: 1
SYS_call: sbrk ID: 12
SYS_call: exec ID: 7
pid state
1 SLEEPING
2 SLEEPING
3 RUNNING