

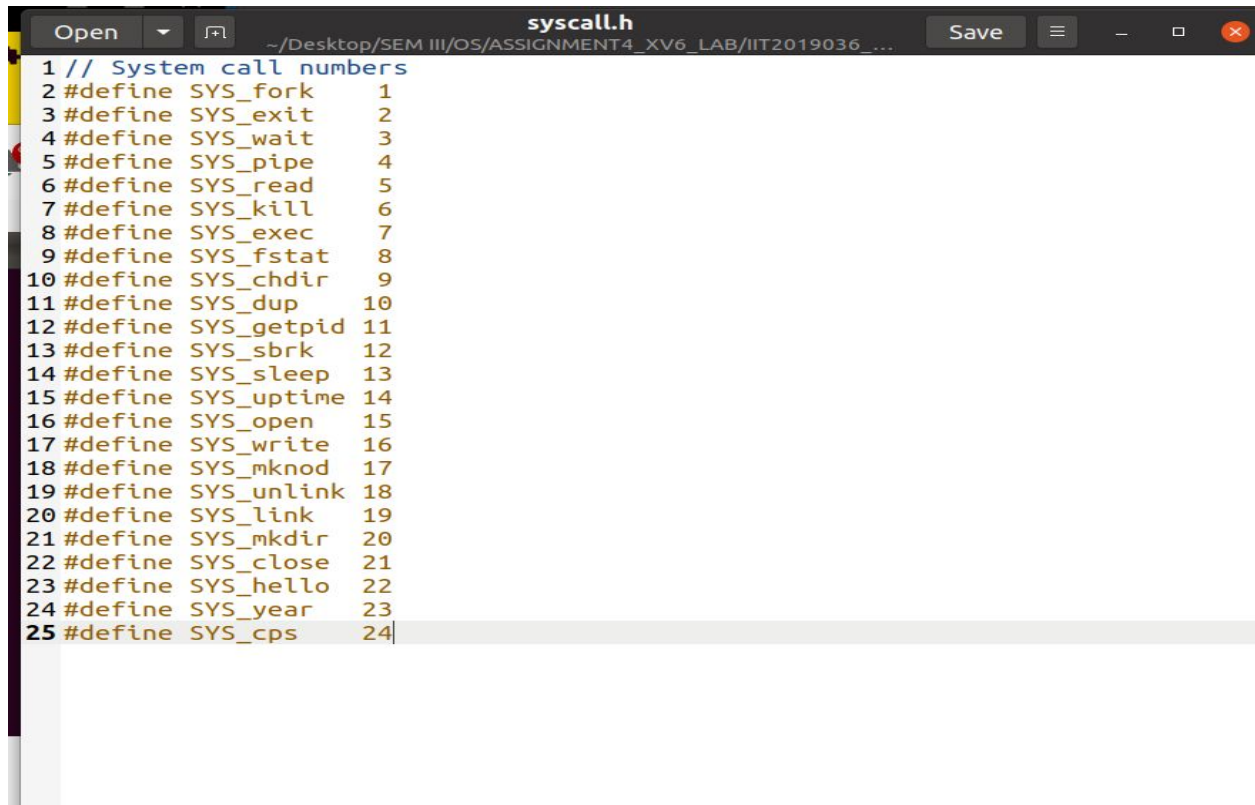
ADDING OF ps SYSTEM CALL on XV6

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



```
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_hello 22
24 #define SYS_year 23
25 #define SYS_cps 24
```

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.

```
Open  [icon] defs.h  ~/Desktop/SEM III/OS/ASSIGNMENT4_XV6_LAB/IIT2019036_...  Save  [icon] [icon] [icon] [icon]
user.h  defs.h
94 // picirq.c
95 void      picenable(int);
96 void      picinit(void);
97
98 // pipe.c
99 int
100 void      pipealloc(struct file**, struct file**);
101 void      pipeclose(struct pipe*, int);
102 int      piperead(struct pipe*, char*, int);
103 void      pipewrite(struct pipe*, char*, int);
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 Save
~/Desktop/SEM III/OS/ASSIGNMENT4_XV6_LAB/IIT2019036_...

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*);
42 int atoi(const char*);
```

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.

```
proc.c
~/Desktop/SEM III/OS/ASSIGNMENT4_XV6_LAB/IIT2019036_...
Save

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.



```
1 #include "types.h"
2 #include "x86.h"
3 #include "defs.h"
4 #include "date.h"
5 #include "param.h"
6 #include "memlayout.h"
7 #include "mmu.h"
8 #include "proc.h"
9
10 int sys_hello(void){
11     cprintf("hello from the kernal!\n");
12     return 0;
13 }
14 int sys_year(void){
15     return 1975;
16 }
17
18 int
19 sys_cps(void)
20 {
21     return cps();
22 }
23
24 int
25 sys_fork(void)
26 {
27     return fork();
28 }
29
30 int
31 sys_exit(void)
32 {
33     exit();
34     return 0; // not reached
35 }
36
37 int
38 sys_wait(void)
39 {
40     return wait();
41 }
42
43 int
44 main(void)
45 {
46     // ...
47 }
```

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
~/Desktop/SEM III/OS/ASSIGNMENT4_XV6_LAB/IIT2019036_...
Save

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(hello)
32 SYSCALL(year)
33 SYSCALL(cps)
34 SYSCALL(uptime)
35
```

STEP7:

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

STEP8:

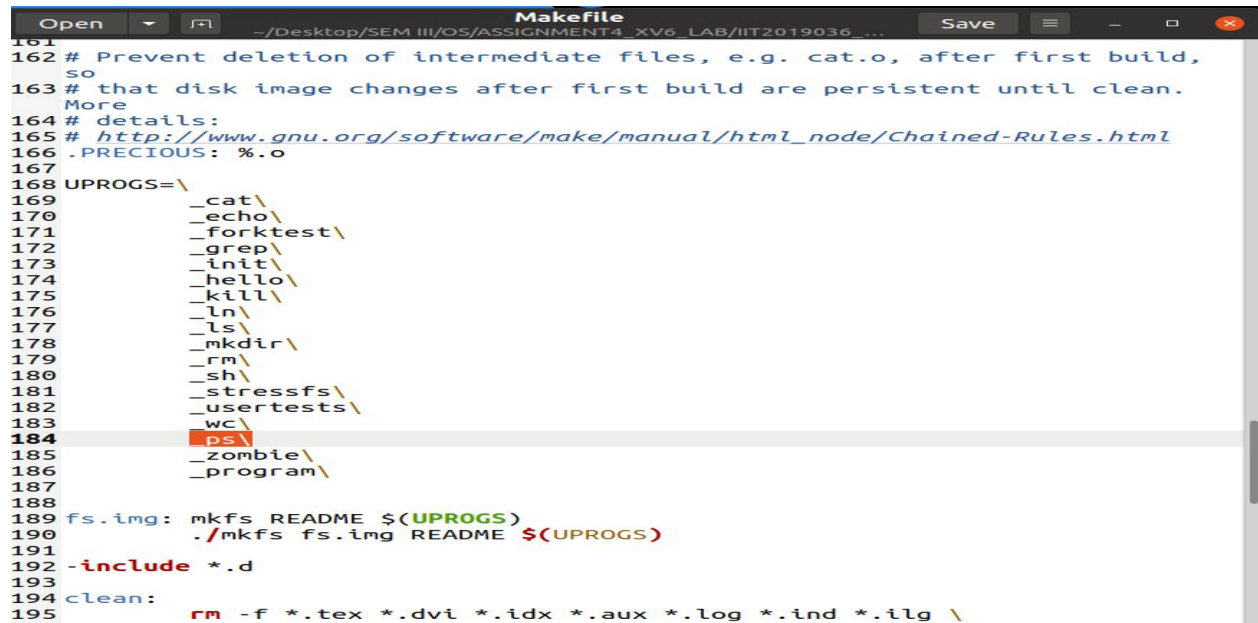
we have to create a ps.c file in which our cps function will be called.

```
ps.c
~/Desktop/SEM III/OS/ASSIGNMENT4_XV6_LAB/IIT2019036_...
Save

1 #include "types.h"
2 #include "stat.h"
3 #include "user.h"
4 #include "fcntl.h"
5
6 int
7 main()
8 {
9     cps();
10
11     exit();
12 }
```

STEP9:

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



```
161
162 # Prevent deletion of intermediate files, e.g. cat.o, after first build,
163 # so
164 # that disk image changes after first build are persistent until clean.
165 # details:
166 # http://www.gnu.org/software/make/manual/html_node/Chained-Rules.html
167 .PRECIOUS: %.o
168 UPROGS=\
169     _cat\
170     _echo\
171     _forktest\
172     _grep\
173     _init\
174     _hello\
175     _kill\
176     _ln\
177     _ls\
178     _mkdir\
179     _rm\
180     _sh\
181     _stressfs\
182     _usertests\
183     _wc\
184     _ps\
185     _zombie\
186     _program\
187
188 fs.img: mkfs README $(UPROGS)
189 ./mkfs fs.img README $(UPROGS)
190
191 -include *.d
192
193 clean:
194 rm -f *.tex *.dvi *.idx *.aux *.log *.ind *.ilg \
```



```

231 qemu-memfs: xv6memfs.img
232 $(QEMU) -drive file=xv6memfs.img,index=0,media=disk,format=raw -
    smp $(CPUS) -m 256
233
234 qemu-nox: fs.img xv6.img
235 $(QEMU) -nographic $(QEMUOPTS)
236
237 .gdbinit: .gdbinit.tmpl
238 sed "s/localhost:1234/localhost:$(GDBPORT)/" < $^ > $@
239
240 qemu-gdb: fs.img xv6.img .gdbinit
241 @echo "*** Now run 'gdb'." 1>&2
242 $(QEMU) -serial mon:stdio $(QEMUOPTS) -S $(QEMUGDB)
243
244 qemu-nox-gdb: fs.img xv6.img .gdbinit
245 @echo "*** Now run 'gdb'." 1>&2
246 $(QEMU) -nographic $(QEMUOPTS) -S $(QEMUGDB)
247
248 # CUT HERE
249 # prepare dist for students
250 # after running make dist, probably want to
251 # rename it to rev0 or rev1 or so on and then
252 # check in that version.
253
254 EXTRA=\
255     mkfs.c ulib.c user.h cat.c echo.c forktest.c grep.c kill.c\
256     ln.c ls.c mkdir.c rm.c stressfs.c usertests.c wc.c
257     zombie.c\program.c\ps.c\
258     printf.c umalloc.c\
259     README dot-bochsrc *.pl toc.* runoff runoff1 runoff.list\
260     .gdbinit.tmpl gdbutil\
261
262 dist:
263     rm -rf dist
264     mkdir dist
265     for i in $(FILES); \
266     do \
267         ares -v PAGEBREAK $$i >dist/$$i: \

```

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
ps 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
name pid state
init 1 SLEEPING
sh 2 SLEEPING
ps 3 RUNNING
```