

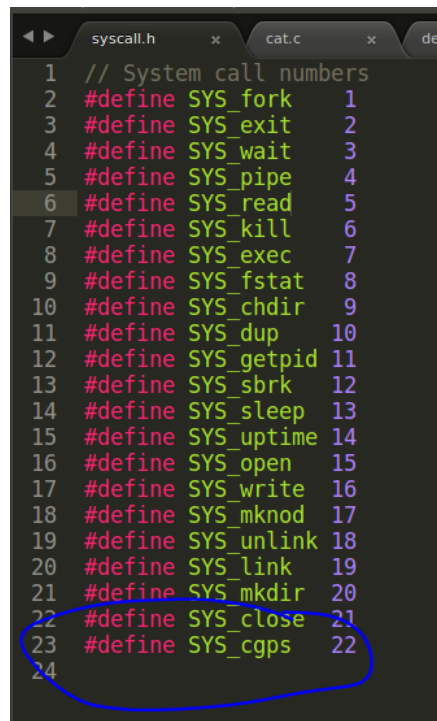
## Design Document

### Overview of the files related to system calls in xv6 OS:

The files **sysproc.c**, **syscall.c**, **syscall.h**, **usys.S**, **user.h**, **defs.h** and **proc.c** are related to system calls in xv6. Out of these files **sysproc.c** is the file in which the system calls are implemented. The file **syscall.c** is the file in which the system call is processed by getting the respective system calls number from **trapeframe.c** file. The file **syscall.h** is used to define integer macros for each system calls. The files **usys.S** and **user.h** are used to tell the microprocessor that a certain function is a system call and not a regular function so that a trap interrupt is generated accordingly, **defs.h** is the file where we have defined our function and **proc.c** where we have defined the functionality for the function.

### Implementation of **cgps** system call:

1:- firstly we need to define system call under **syscall.h** and assign number(22) to it i.e:-



```
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_cgps 22
24
```

2:- now need to declare system call under **defs.h** under **proc.c** and **user.h**:-

```

100 int    piperead(struct pipe*, char*, int);
101 int    pipewrite(struct pipe*, char*, int);
102
103 //PAGEBREAK: 16
104 // proc.c
105 struct proc* copyproc(struct proc*);
106 void    exit(void);
107 int     fork(void);
108 int     growproc(int);
109 int     kill(int);
110 void    pinit(void);
111 void    procdump(void);
112 void    scheduler(void) __attribute__((noreturn));
113 void    sched(void);
114 void    sleep(void*, struct spinlock*);
115 void    userinit(void);
116 int     wait(void);
117 void    wakeup(void*);
118 void    yield(void);
119 int     cgps(void);
120
121 // switch.S

```

```

1 struct stat;
2
3 // system calls
4 int fork(void);
5 int exit(void) __attribute__((noreturn));
6 int wait(void);
7 int pipe(int*);
8 int write(int, void*, int);
9 int read(int, void*, int);
10 int close(int);
11 int kill(int);
12 int exec(char*, char**);
13 int open(char*, int);
14 int mknod(char*, short, short);
15 int unlink(char*);
16 int fstat(int fd, struct stat*);
17 int link(char*, char*);
18 int mkdir(char*);
19 int chdir(char*);
20 int dup(int);
21 int getpid(void);
22 char* sbrk(int);
23 int sleep(int);
24 int uptime(void);
25 int cgps(void);
26

```

3:- now its time to definition for sys\_cgps() under sysproc.c:-

```

85
86 acquire(&tickslock);
87 xticks = ticks;
88 release(&tickslock);
89 return xticks;
90 }
91 int
92 sys_cgps(void)
93 {
94     return cgps();
95 }
96

```

4:- Now we need to declare system function so that when it need it can move to kernel level processing

```

1 #include "syscall.h"
2 #include "trans.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(cgps)
33

```



```

75 if(argint(n, &addr) < 0)
76     return -1;
77 return fetchstr(addr, pp);
78
79
80 extern int sys_chdir(void);
81 extern int sys_close(void);
82 extern int sys_dup(void);
83 extern int sys_exec(void);
84 extern int sys_exit(void);
85 extern int sys_fork(void);
86 extern int sys_fstat(void);
87 extern int sys_getpid(void);
88 extern int sys_kill(void);
89 extern int sys_link(void);
90 extern int sys_mkdir(void);
91 extern int sys_mknod(void);
92 extern int sys_open(void);
93 extern int sys_pipe(void);
94 extern int sys_read(void);
95 extern int sys_sbrk(void);
96 extern int sys_sleep(void);
97 extern int sys_unlink(void);
98 extern int sys_wait(void);
99 extern int sys_write(void);
100 extern int sys_uptime(void);
101 extern int sys_cgps(void);
102
103 static int (*syscall[]) (void) = {
104     [SYS_fork] sys_fork,
105     [SYS_exit] sys_exit,
106     [SYS_wait] sys_wait,
107     [SYS_pipe] sys_pipe,
108     [SYS_read] sys_read,
109     [SYS_kill] sys_kill,
110     [SYS_exec] sys_exec,
111     [SYS_fstat] sys_fstat,
112     [SYS_chdir] sys_chdir,
113     [SYS_dup] sys_dup,
114     [SYS_getpid] sys_getpid,
115     [SYS_sbrk] sys_sbrk,
116     [SYS_sleep] sys_sleep,
117     [SYS_uptime] sys_uptime,
118     [SYS_open] sys_open,
119     [SYS_cgps] sys_cgps,
120     [SYS_write] sys_write,
121     [SYS_mknod] sys_mknod,
122     [SYS_unlink] sys_unlink,
123     [SYS_link] sys_link,
124     [SYS_mkdir] sys_mkdir,
125     [SYS_close] sys_close,
126 };
127

```

6:- need to declare its corresponding system calls at syscall.c

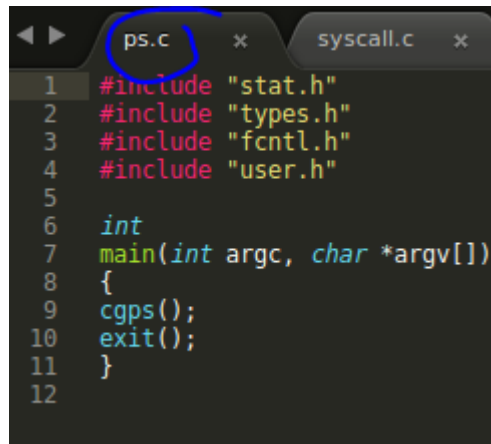
7:- Now its time to declare its functionality under proc.c:-

```

454 }
455     cprintf("\n");
456 }
457 }
458
459 int
460 cgps(){
461     struct proc *p;
462     sti();
463
464     acquire(&ptable.lock);
465     cprintf("name \t pid \t state \t \t memory \t \n");
466     for (p= ptable.proc; p<&ptable.proc[NPROC]; p++)
467     {
468         if (p->state==SLEEPING)
469             cprintf("%s \t %d \t SLEEPING \t %d KB \n ", p->name, p->pid, p->sz/1024);
470         else if (p->state==RUNNING)
471             cprintf("%s \t %d \t RUNNING \t %d KB \n ", p->name, p->pid, p->sz/1024);
472         else if (p->state==RUNNABLE)
473             cprintf("%s \t %d \t RUNNABLE \t %d KB \n ", p->name, p->pid, p->sz/1024);
474         else if (p->state==EMBRYO)
475             cprintf("%s \t %d \t EMBRYO \t %d KB \n ", p->name, p->pid, p->sz/1024);
476         else if (p->state==ZOMBIE)
477             cprintf("%s \t %d \t ZOMBIE \t %d KB \n ", p->name, p->pid, p->sz/1024);
478     }
479
480     release(&ptable.lock);
481     return 22;
482 }
483
484

```

Finally new c file creation with the name of ps.c which just call cgps function and return.



```
1 #include "stat.h"
2 #include "types.h"
3 #include "fcntl.h"
4 #include "user.h"
5
6 int
7 main(int argc, char *argv[])
8 {
9     cgps();
10    exit();
11 }
12
```

First time the ps program is called it will show the count of all the kernel calls and any calls that were made before the test program was executed as well as the calls made by the ps program. As the **sysCalls** array gets reset to zero after the **syscall\_cgps** finishes execution, when ps program is called second time it shows only the number of calls the test program has made.