

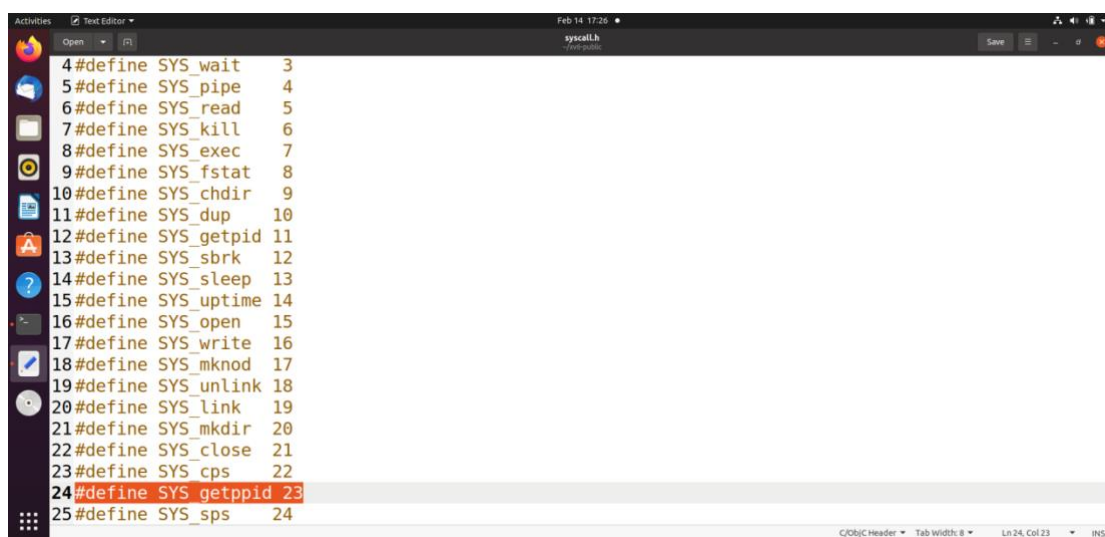
### GROUP1 MEMBERS :-

1	DEBANJAN DAS	2005448
2	Priyanka Dubey	2005253
3	SHUBHAM TIWARI	20051488
4	DEBASHREE CHAKRABORTY	20051139
5	Kaustav Maity	2005519
6	Saumya	2005827
7	Aditi Singh	20051629
8	Anirban Ball	2005642
9	Shivam Kumar	20051252
10	Devanshi	20051493

## OS ASSIGNMENT

1.) Create a system call called `getppid()` and create a command called "prd" where you need to display the process-id along with parent process-id. (use the help of `getpid`).

### Changes in syscall.h :-



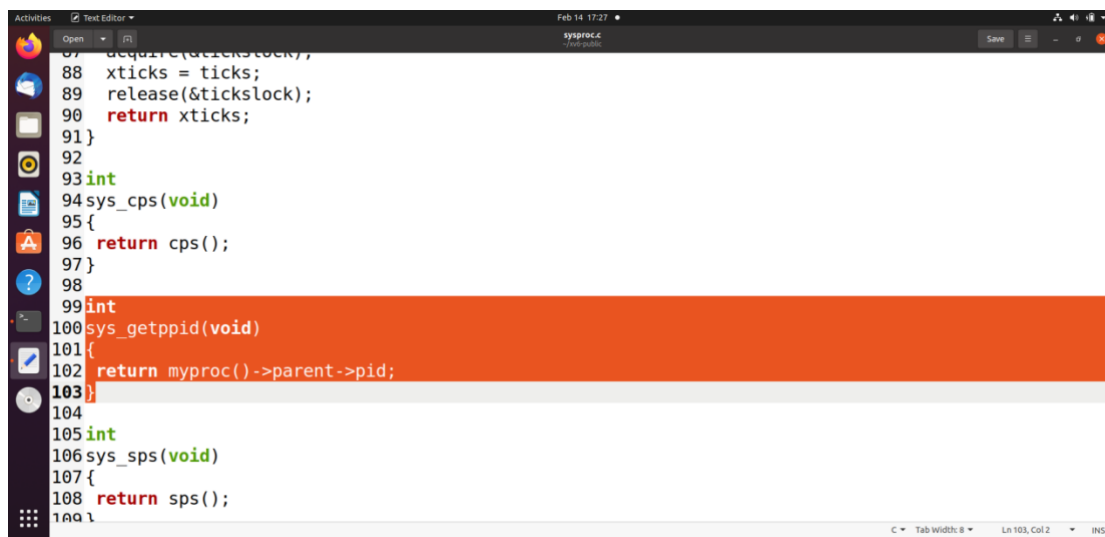
```
4#define SYS_wait 3
5#define SYS_pipe 4
6#define SYS_read 5
7#define SYS_kill 6
8#define SYS_exec 7
9#define SYS_fstat 8
10#define SYS_chdir 9
11#define SYS_dup 10
12#define SYS_getpid 11
13#define SYS_sbrk 12
14#define SYS_sleep 13
15#define SYS_uptime 14
16#define SYS_open 15
17#define SYS_write 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#define SYS_getppid 23
25#define SYS_sps 24
```

### Changes in user.h :-



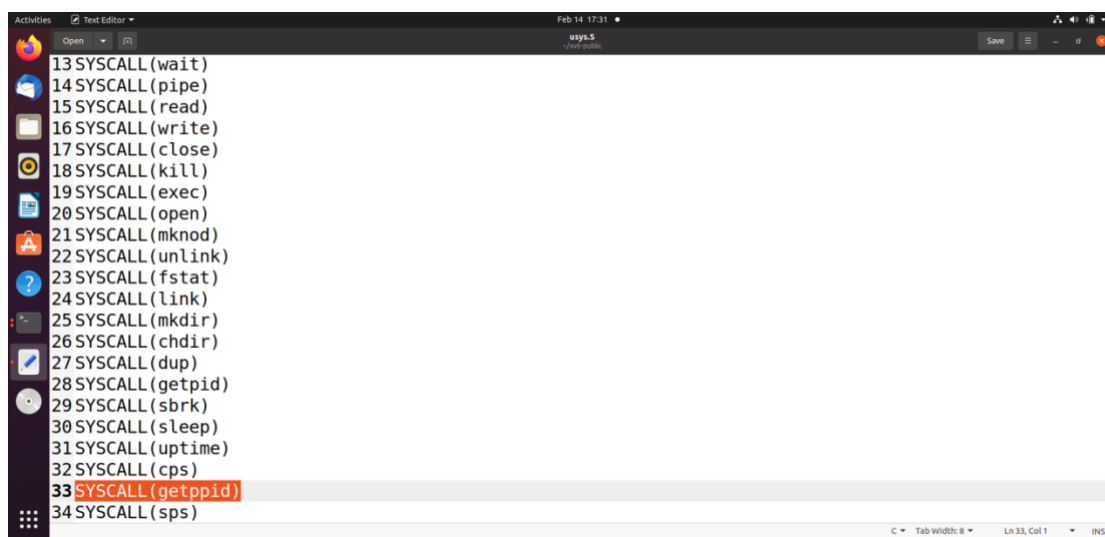
```
19 int mkdir(const char *);
20 int chdir(const char *);
21 int dup(int);
22 int getpid(void);
23 char* sbrk(int);
24 int sleep(int);
25 int uptime(void);
26 int cps(void);
27 int getppid(void);
28 int sps(void);
29
30 // ulib.c
31 int stat(const char*, struct stat*);
32 char* strcpy(char*, const 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*);
```

## Changes in sysproc.c :-



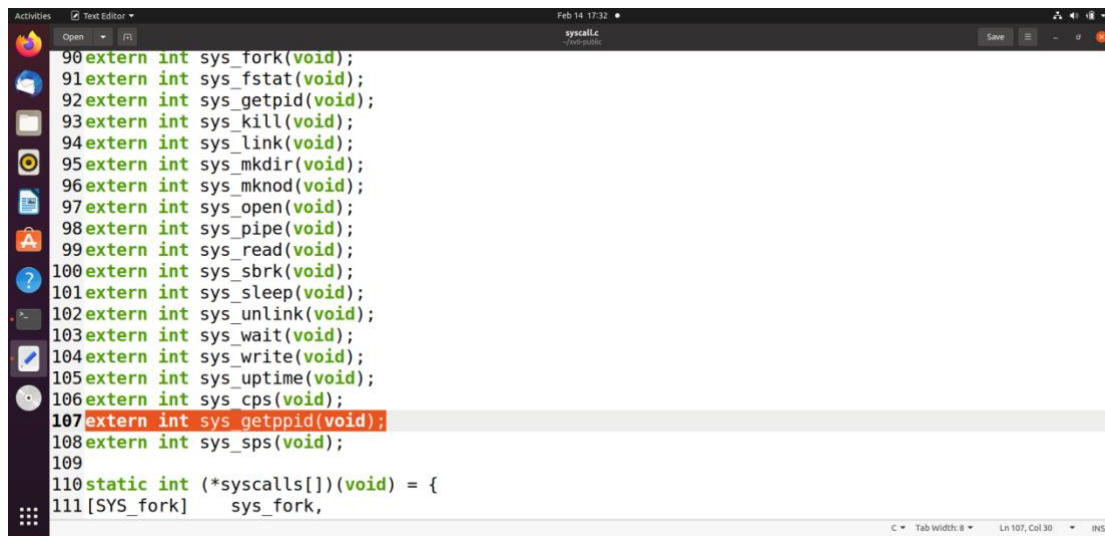
```
87 acquire(&tickslock);
88 xticks = ticks;
89 release(&tickslock);
90 return xticks;
91}
92
93 int
94 sys_cps(void)
95 {
96     return cps();
97 }
98
99 int
100 sys_getppid(void)
101 {
102     return myproc()->parent->pid;
103 }
104
105 int
106 sys_sps(void)
107 {
108     return sps();
109 }
```

## Changes in usys.S :-

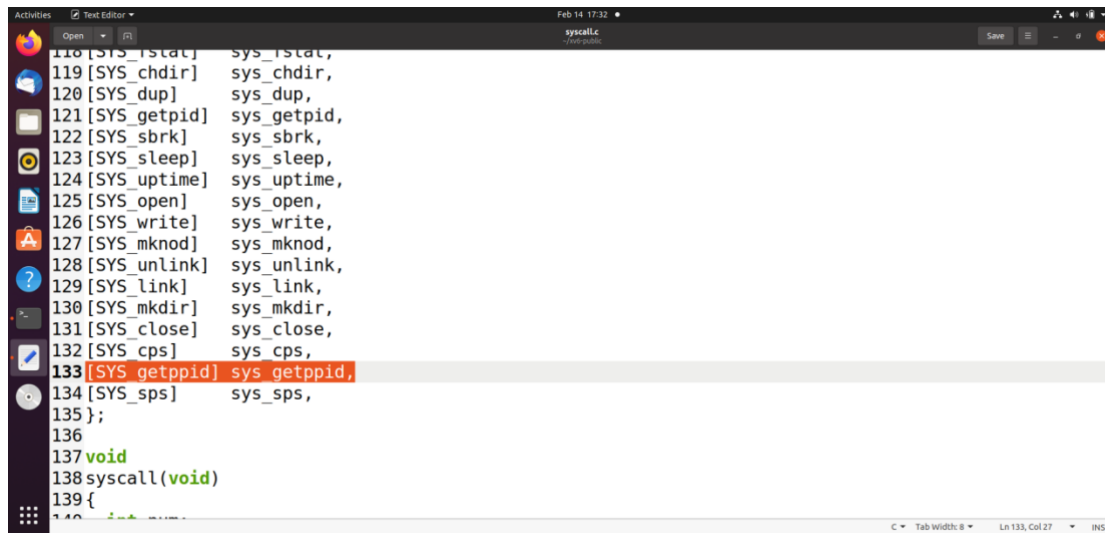


```
13 SYSCALL(wait)
14 SYSCALL(pipe)
15 SYSCALL(read)
16 SYSCALL(write)
17 SYSCALL(close)
18 SYSCALL(kill)
19 SYSCALL(exec)
20 SYSCALL(open)
21 SYSCALL(mknod)
22 SYSCALL(unlink)
23 SYSCALL(fstat)
24 SYSCALL(link)
25 SYSCALL(mkdir)
26 SYSCALL(chdir)
27 SYSCALL(dup)
28 SYSCALL(getpid)
29 SYSCALL(sbrk)
30 SYSCALL(sleep)
31 SYSCALL(uptime)
32 SYSCALL(cps)
33 SYSCALL(getppid)
34 SYSCALL(sps)
```

## Changes in syscall.c :-

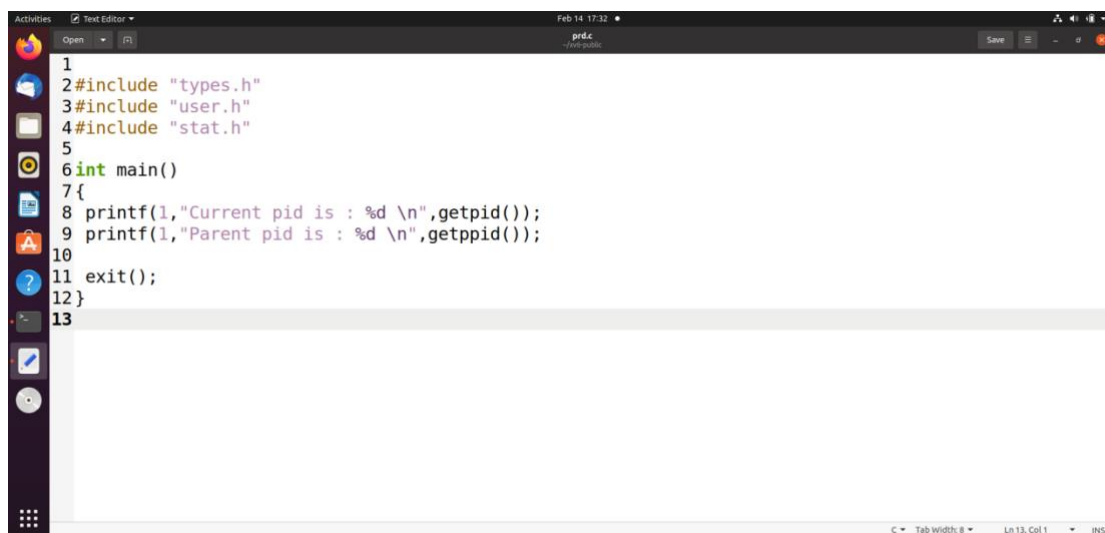


```
90extern int sys_fork(void);
91extern int sys_fstat(void);
92extern int sys_getpid(void);
93extern int sys_kill(void);
94extern int sys_link(void);
95extern int sys_mkdir(void);
96extern int sys_mknod(void);
97extern int sys_open(void);
98extern int sys_pipe(void);
99extern int sys_read(void);
100extern int sys_sbrk(void);
101extern int sys_sleep(void);
102extern int sys_unlink(void);
103extern int sys_wait(void);
104extern int sys_write(void);
105extern int sys_uptime(void);
106extern int sys_cps(void);
107extern int sys_getppid(void);
108extern int sys_sps(void);
109
110static int (*syscalls[])(void) = {
111[SYS_fork] sys_fork,
```



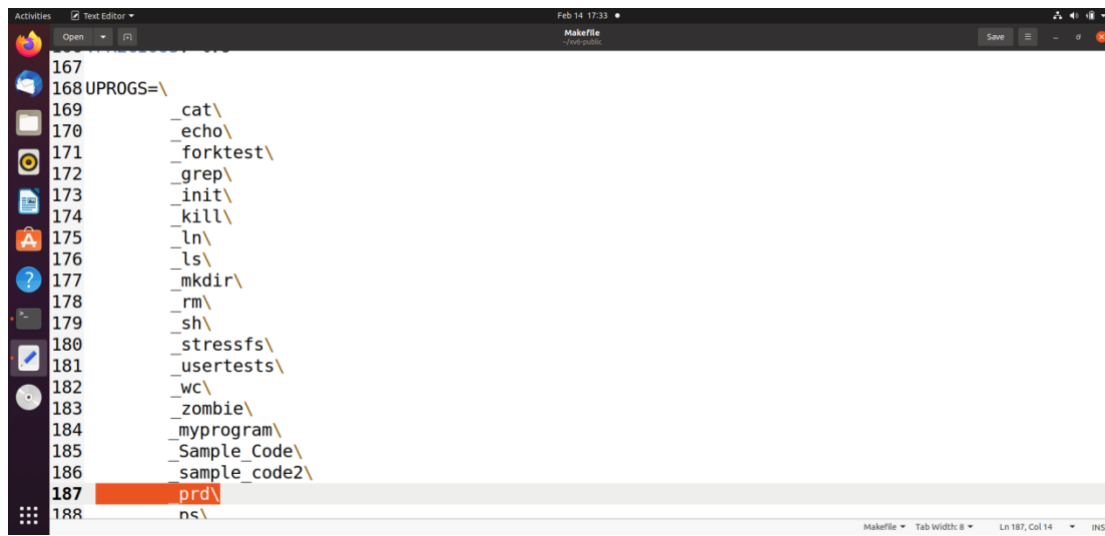
```
118[SYS_fstat] sys_fstat,
119[SYS_chdir] sys_chdir,
120[SYS_dup] sys_dup,
121[SYS_getpid] sys_getpid,
122[SYS_sbrk] sys_sbrk,
123[SYS_sleep] sys_sleep,
124[SYS_uptime] sys_uptime,
125[SYS_open] sys_open,
126[SYS_write] sys_write,
127[SYS_mknod] sys_mknod,
128[SYS_unlink] sys_unlink,
129[SYS_link] sys_link,
130[SYS_mkdir] sys_mkdir,
131[SYS_close] sys_close,
132[SYS_cps] sys_cps,
133[SYS_getppid] sys_getppid,
134[SYS_sps] sys_sps,
135};
136
137void
138syscall(void)
139{
140    int sysno;
```

## Creating file prd.c :-



```
1
2#include "types.h"
3#include "user.h"
4#include "stat.h"
5
6int main()
7{
8    printf(1, "Current pid is : %d \\n", getpid());
9    printf(1, "Parent pid is : %d \\n", getppid());
10
11    exit();
12}
13
```

## Changes in Makefile :-

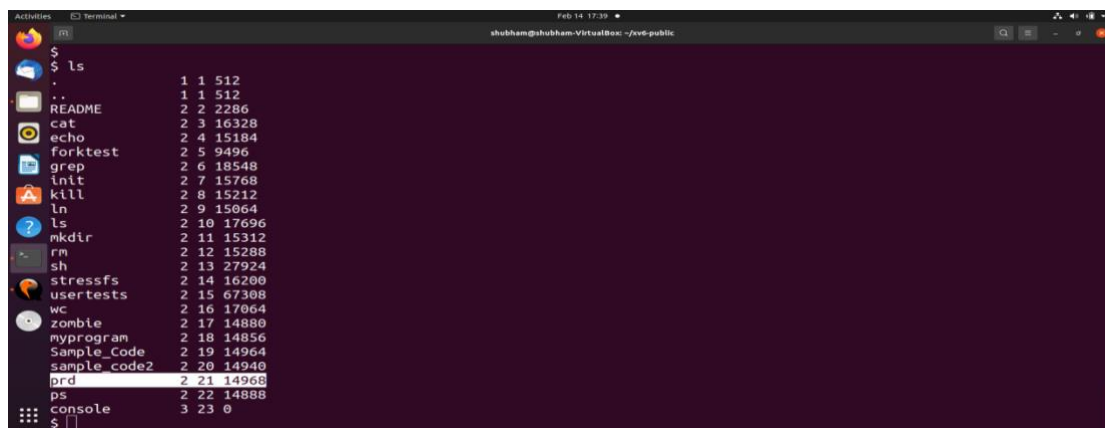


```
167
168 UPROGS=\
169     _cat\
170     _echo\
171     _forktest\
172     _grep\
173     _init\
174     _kill\
175     _ln\
176     _ls\
177     _mkdir\
178     _rm\
179     _sh\
180     _stressfs\
181     _usertests\
182     _wc\
183     _zombie\
184     _myprogram\
185     _sample_code\
186     _sample_code2\
187     _prd\
188     _ns\
```



```
253 # check in that version.
254
255 EXTRA=\
256     mkfs.c ulib.c user.h cat.c echo.c forktest.c grep.c kill.c\
257     ln.c ls.c mkdir.c rm.c stressfs.c usertests.c wc.c zombie.c\
258     printf.c umalloc.c/myprogram.c/sample_code.c/sample_code2.c/prd.c/ps.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         grep -v PAGEBREAK $$i >dist/$$i; \
268     done
269     sed '/CUT HERE/,$$d' Makefile >dist/Makefile
270     echo >dist/runoff.spec
271     cp $(EXTRA) dist
272
273 dist-test:
274     rm -rf dist
```

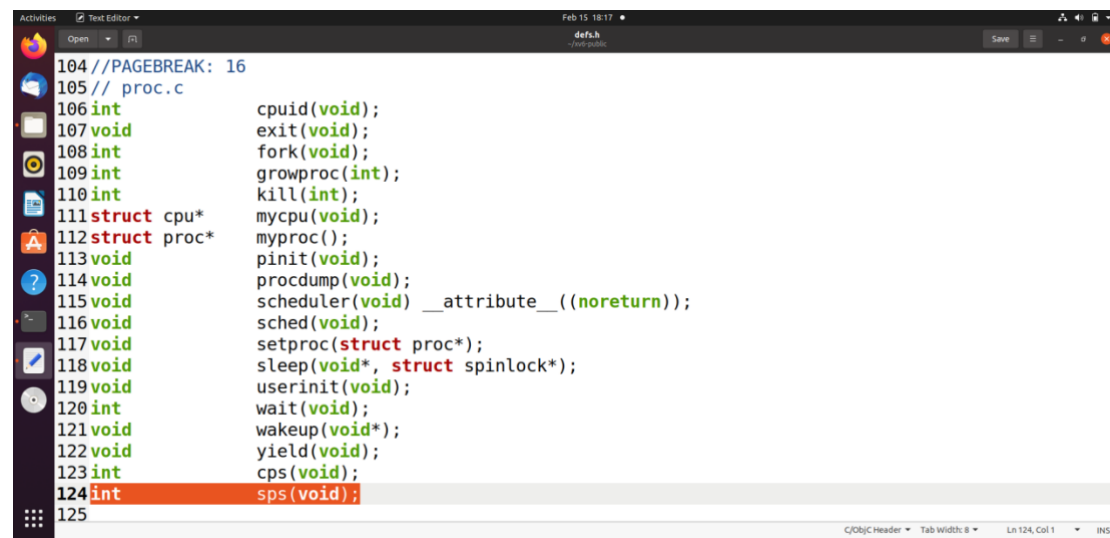
## OUTPUTS (SCREENSHOTS) :-



```
$ ls
. 1 1 512
.. 1 1 512
README 2 2 2286
cat 2 3 16328
echo 2 4 15184
forktest 2 5 9496
grep 2 6 18548
init 2 7 15768
kill 2 8 15212
ln 2 9 15064
ls 2 10 17696
mkdir 2 11 15312
rm 2 12 15288
sh 2 13 27924
stressfs 2 14 16200
usertests 2 15 67308
wc 2 16 17064
zombie 2 17 14880
myprogram 2 18 14856
sample_code 2 19 14964
sample_code2 2 20 14940
prd 2 21 14968
ps 2 22 14888
console 3 23 0
```



## Changes in defs.h :-



A screenshot of a text editor window titled 'defs.h' showing a list of function declarations. The file is divided into sections by comments: '// PAGEBREAK: 16', '// proc.c', and '// ulibc.c'. The declarations include various system calls and library functions. Line 124, 'sps(void);', is highlighted in orange.

```
104//PAGEBREAK: 16
105// proc.c
106int      cpuid(void);
107void     exit(void);
108int      fork(void);
109int      growproc(int);
110int      kill(int);
111struct cpu* mycpu(void);
112struct proc* myproc();
113void     pinit(void);
114void     procdump(void);
115void     scheduler(void) __attribute__((noreturn));
116void     sched(void);
117void     setproc(struct proc*);
118void     sleep(void*, struct spinlock*);
119void     userinit(void);
120int      wait(void);
121void     wakeup(void*);
122void     yield(void);
123int      cps(void);
124int      sps(void);
125
// ulibc.c
126int      stat(const char*, struct stat*);
127char*    strcpy(char*, const char*);
128void*    memmove(void*, const void*, int);
```

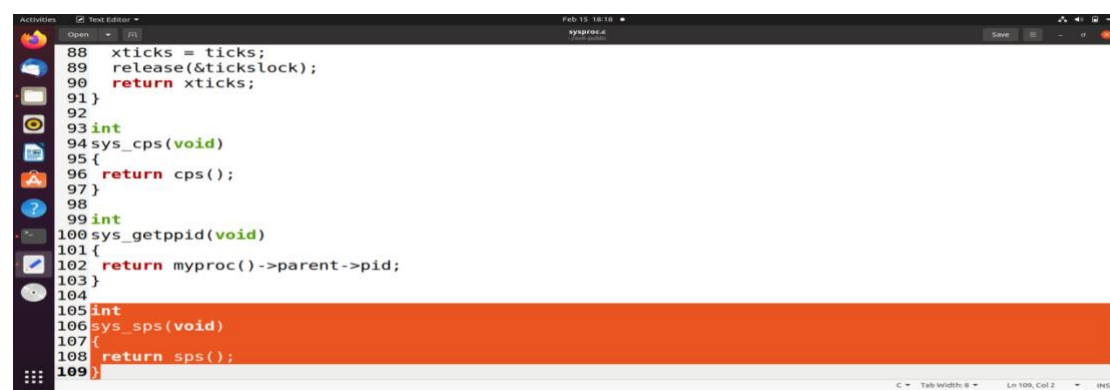
## Changes in user.h :-



A screenshot of a text editor window titled 'user.h' showing a list of function declarations. The declarations include various system calls and library functions. Line 28, 'sps(void);', is highlighted in orange.

```
12int      kill(int);
13int      exec(char*, char**);
14int      open(const char*, int);
15int      mknod(const char*, short, short);
16int      unlink(const char*);
17int      fstat(int fd, struct stat*);
18int      link(const char*, const char*);
19int      mkdir(const char*);
20int      chdir(const char*);
21int      dup(int);
22int      getpid(void);
23char*    sbrk(int);
24int      sleep(int);
25int      uptime(void);
26int      cps(void);
27int      getppid(void);
28int      sps(void);
29
30// ulibc.c
31int      stat(const char*, struct stat*);
32char*    strcpy(char*, const char*);
33void*    memmove(void*, const void*, int);
```

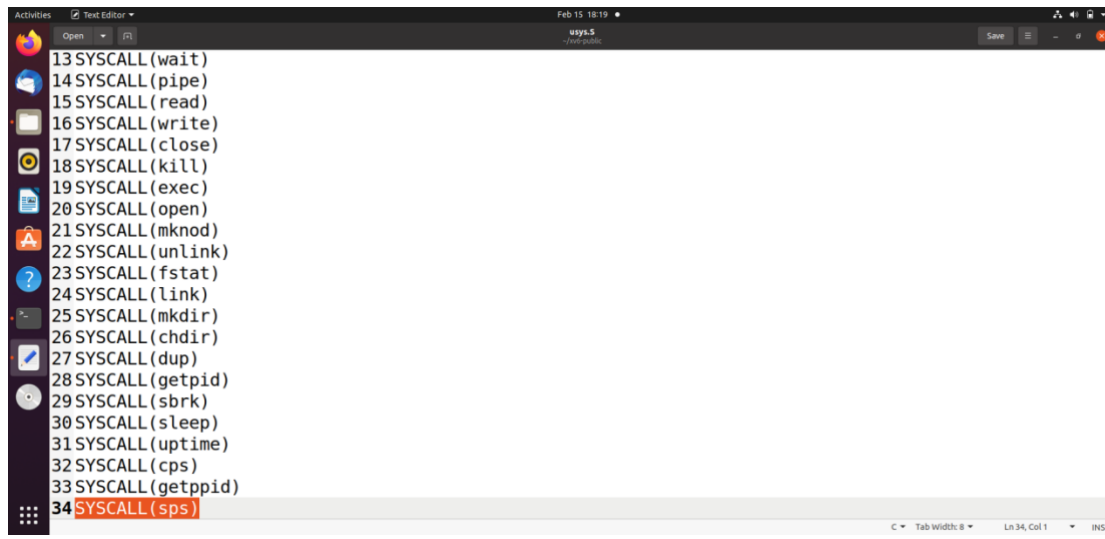
## Changes in sysproc.c :-



A screenshot of a text editor window titled 'sysproc.c' showing a list of function definitions. The definitions include various system calls and library functions. Line 109, 'return sps();', is highlighted in orange.

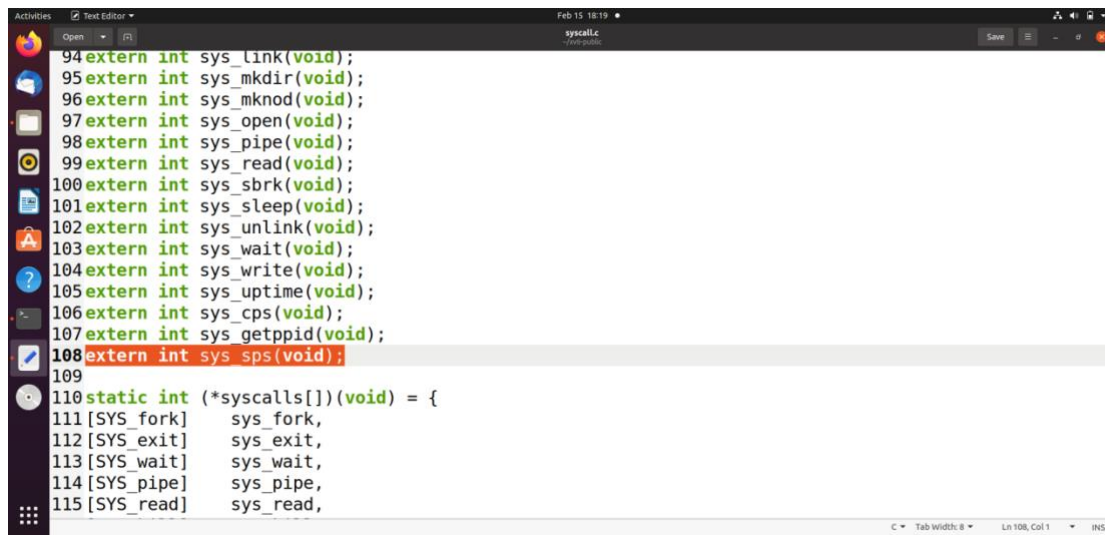
```
88  xticks = ticks;
89  release(&tickslock);
90  return xticks;
91}
92
93int
94sys_cps(void)
95{
96  return cps();
97}
98
99int
100sys_getppid(void)
101{
102  return myproc()->parent->pid;
103}
104
105int
106sys_sps(void)
107{
108  return sps();
109}
```

## Changes in usys.S :-

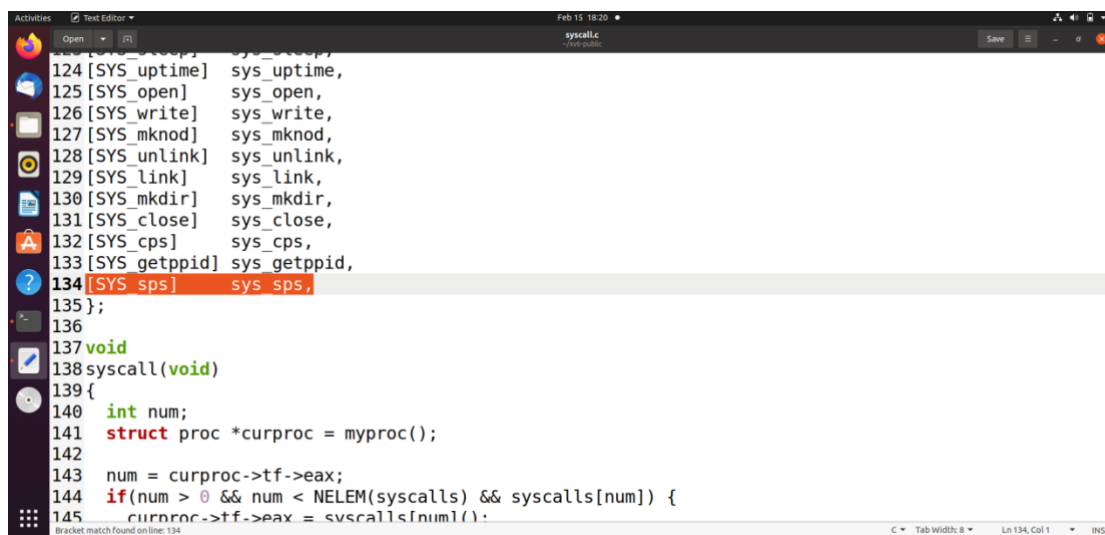


```
13 SYSCALL(wait)
14 SYSCALL(pipe)
15 SYSCALL(read)
16 SYSCALL(write)
17 SYSCALL(close)
18 SYSCALL(kill)
19 SYSCALL(exec)
20 SYSCALL(open)
21 SYSCALL(mknod)
22 SYSCALL(unlink)
23 SYSCALL(fstat)
24 SYSCALL(link)
25 SYSCALL(mkdir)
26 SYSCALL(chdir)
27 SYSCALL(dup)
28 SYSCALL(getpid)
29 SYSCALL(sbrk)
30 SYSCALL(sleep)
31 SYSCALL(uptime)
32 SYSCALL(cps)
33 SYSCALL(getppid)
34 SYSCALL(sps)
```

## Changes in syscall.c :-



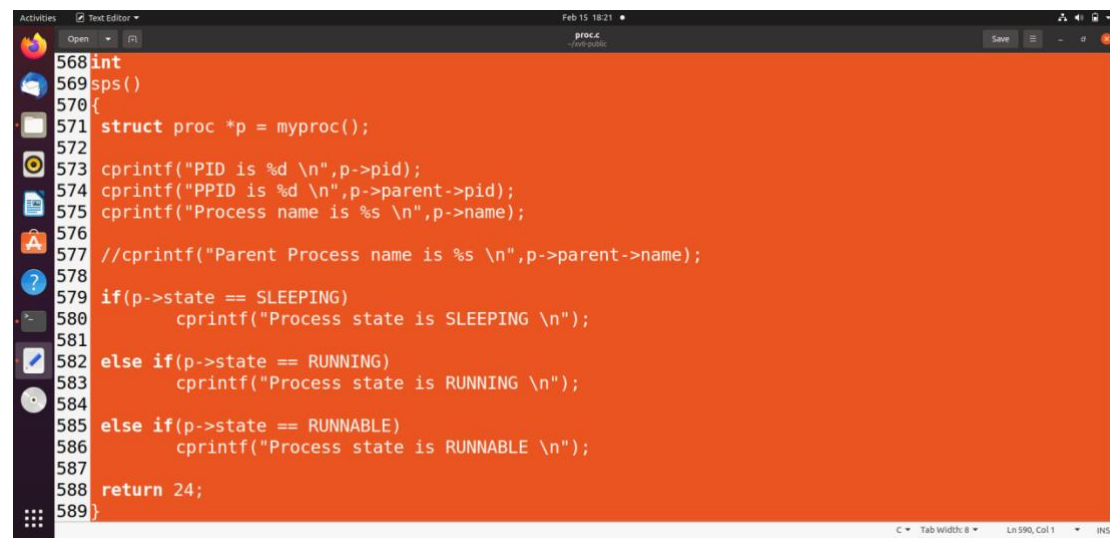
```
94 extern int sys_link(void);
95 extern int sys_mkdir(void);
96 extern int sys_mknod(void);
97 extern int sys_open(void);
98 extern int sys_pipe(void);
99 extern int sys_read(void);
100 extern int sys_sbrk(void);
101 extern int sys_sleep(void);
102 extern int sys_unlink(void);
103 extern int sys_wait(void);
104 extern int sys_write(void);
105 extern int sys_uptime(void);
106 extern int sys_cps(void);
107 extern int sys_getppid(void);
108 extern int sys_sps(void);
109
110 static int (*syscalls[])(void) = {
111 [SYS_fork] sys_fork,
112 [SYS_exit] sys_exit,
113 [SYS_wait] sys_wait,
114 [SYS_pipe] sys_pipe,
115 [SYS_read]
```



```
124 [SYS_uptime] sys_uptime,
125 [SYS_open] sys_open,
126 [SYS_write] sys_write,
127 [SYS_mknod] sys_mknod,
128 [SYS_unlink] sys_unlink,
129 [SYS_link] sys_link,
130 [SYS_mkdir] sys_mkdir,
131 [SYS_close] sys_close,
132 [SYS_cps] sys_cps,
133 [SYS_getppid] sys_getppid,
134 [SYS_sps] sys_sps,
135 };
136
137 void
138 syscall(void)
139 {
140     int num;
141     struct proc *curproc = myproc();
142
143     num = curproc->tf->eax;
144     if(num > 0 && num < NELEM(syscalls) && syscalls[num]) {
145         curproc->tf->eax = syscalls[num]();
146     }
```

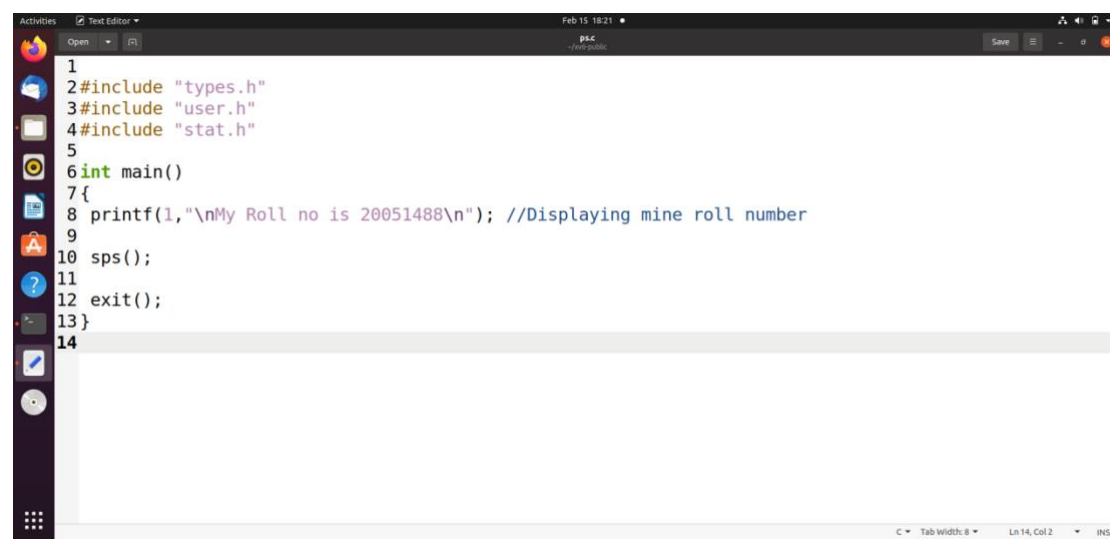


## Changes in proc.c :-

A screenshot of a text editor window titled 'proc.c'. The editor shows a C program with line numbers 568 to 589. The code defines a 'struct proc' and a function 'sps()' that prints various process information like PID, PPID, name, and state. The state is checked for SLEEPING, RUNNING, and RUNNABLE. The function returns 24.

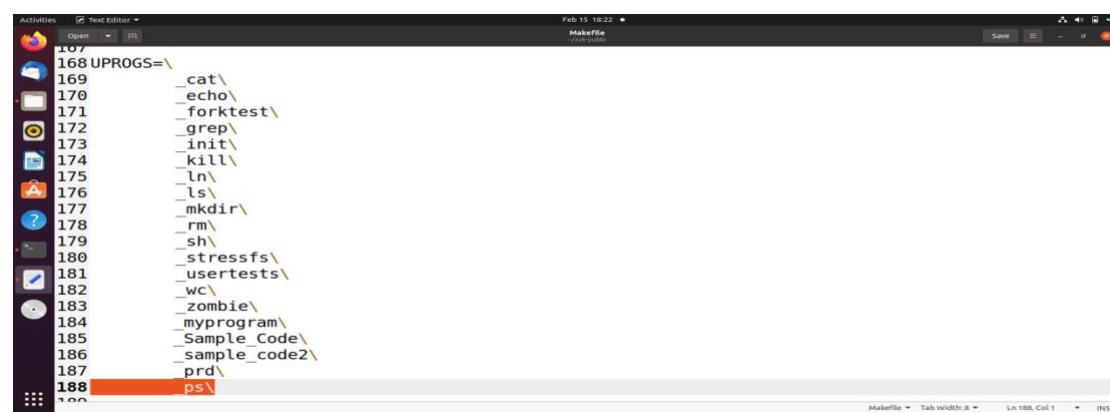
```
568 int
569 sps()
570 {
571     struct proc *p = myproc();
572
573     cprintf("PID is %d \n", p->pid);
574     cprintf("PPID is %d \n", p->parent->pid);
575     cprintf("Process name is %s \n", p->name);
576
577     //cprintf("Parent Process name is %s \n", p->parent->name);
578
579     if(p->state == SLEEPING)
580         cprintf("Process state is SLEEPING \n");
581
582     else if(p->state == RUNNING)
583         cprintf("Process state is RUNNING \n");
584
585     else if(p->state == RUNNABLE)
586         cprintf("Process state is RUNNABLE \n");
587
588     return 24;
589 }
```

## Creating ps.c file :-

A screenshot of a text editor window titled 'ps.c'. The editor shows a C program with line numbers 1 to 14. The code includes 'types.h', 'user.h', and 'stat.h'. The 'main()' function prints a roll number and calls 'sps()' before exiting.

```
1
2 #include "types.h"
3 #include "user.h"
4 #include "stat.h"
5
6 int main()
7 {
8     printf(1, "\nMy Roll no is 20051488\n"); //Displaying mine roll number
9
10    sps();
11
12    exit();
13 }
14
```

## Changes in Makefile :-

A screenshot of a text editor window titled 'Makefile'. The editor shows a list of directories under the 'UPROGS=' variable, with line numbers 167 to 188. The directory 'ps\' is highlighted at line 188.

```
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     _myprogram\
184     _sample_code\
185     _sample_code2\
186     _prd\
187     _ps\
188
```



```
245 qemu-nox-gdb: ls.img xv6.img .gdbinit
246 @echo "*** Now run 'gdb'. " 1>&2
247 $(QEMU) -nographic $(QEMUOPTS) -S $(QEMUGDB)
248
249# CUT HERE
250# prepare dist for students
251# after running make dist, probably want to
252# rename it to rev0 or rev1 or so on and then
253# check in that version.
254
255 EXTRA=\
256 mkfs.c ulib.c user.h cat.c echo.c forktest.c grep.c kill.c\
257 ln.c ls.c mkdir.c rm.c stressfs.c usertests.c wc.c zombie.c\
258 printf.c umalloc.c\myprogram.c\Sample_Code.c\sample_code2.c\prd.c\ps.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 \
```

## OUTPUTS(Screenshots) :-

```
$ ls
.          1 1 512
..         1 1 512
README    2 2 2286
cat        2 3 16328
echo       2 4 15184
forktest   2 5 9496
grep       2 6 18548
init       2 7 15768
kill       2 8 15212
ln         2 9 15064
ls         2 10 17696
mkdir      2 11 15312
rm         2 12 15288
sh         2 13 27924
stressfs   2 14 16200
usertests  2 15 67308
wc         2 16 17064
zombie     2 17 14880
myprogram  2 18 14856
Sample_Code 2 19 14964
sample_code2 2 20 14940
prd        2 21 14968
ps         2 22 14888
console    3 23 0
$
$
```

```
$ ps
My Roll no is 20051488
PID is 7
PPID is 2
Process name is ps
Process state is RUNNING
$
```

```
ls          10 17696
mkdir       11 15312
rm          12 15288
sh          13 27924
stressfs    14 16200
usertests   15 67308
wc          16 17064
zombie      17 14880
myprogram   18 14856
Sample_Code 19 14964
sample_code2 20 14940
prd         21 14968
ps          22 14888
console     23 0
$ ps
My Roll no is 20051488
PID is 7
PPID is 2
Process name is ps
Process state is RUNNING
$
```

### 3.) Create a cal command with different options as specified in Unix manual.

**Step:--**

**1.) Create cal.c file containing the command to display the calendar with different options as specified in Unix manual. The C code snippet is shown below:-**

```
#include "types.h"
#include "stat.h"
#include "user.h"
#define TRUE  1
#define FALSE 0

int d_month[]={0,31,28,31,30,31,30,31,31,30,31,30,31};
char *months[]=
{
    " ",
    "\nJanuary",
    "\nFebruary",
    "\nMarch",
    "\nApril",
    "\nMay",
    "\nJune",
    "\nJuly",
    "\nAugust",
    "\nSeptember",
    "\nOctober",
    "\nNovember",
    "\nDecember"
};

int find_day(int y)
{
    int day;
    int n1, n2, n3;

    n1 = (y - 1.)/ 4.0;
    n2 = (y - 1.)/ 100.;
    n3 = (y - 1.)/ 400.;
    day = (y + n1 - n2 + n3) % 7;
    return day;
}

int leapyear(int y)
{
    if((y% 4 == FALSE && y%100 != FALSE) || y%400 == FALSE)
    {
        d_month[2] = 29;
        return TRUE;
    }
}
```

```

    }
    else
    {
        d_month[2] = 28;
        return FALSE;
    }
}

void cal1(int y, int day)
{
    int month, d;
    for ( month = 1; month <= 12; month++ )
    {
        printf(1,"%s", months[month]);
        printf(1,"\n\nSun Mon Tue Wed Thu Fri Sat\n" );

        for ( d = 1; d <= 1 + day * 5; d++ )
        {
            printf(1," ");
        }

        for ( d = 1; d <= d_month[month]; d++ )
        {
            printf(1,"%d", d);

            if ( ( d + day ) % 7 > 0 )
                printf(1," ");
            else
                printf(1,"\n " );
        }

        day= ( day + d_month[month] ) % 7;
    }
}

void cal2(int year, int dcode, int m)
{
    int month, day;

    for ( month = 1; month < m; month++ )
    {
        dcode = ( dcode + d_month[month] ) % 7;
    }

    month = m;

```

```

printf(1,"%s", months[month]);
printf(1,"\n\nSun Mon Tue Wed Thu Fri Sat\n" );

for ( day = 1; day <= 1 + dcode * 5; day++ )
{
    printf(1," ");
}

for ( day = 1; day <= d_month[month]; day++ )
{
    printf(1,"%d", day );

    if ( ( day + dcode ) % 7 > 0 )
        printf(1," ");
    else
        printf(1,"\n ");
}
}

int main(int argc, char * argv[])
{
    int year, daycode;
    int month;

    if(argc == 1)
    {
        year = 2022;
        month = 1;
        printf(1,"\nCALENDAR %d\n", year);
        daycode = find_day(year);
        leapyear(year);
        cal2(year, daycode,month);
    }
    else if(argc == 2)
    {
        year = atoi(argv[1]);
        printf(1,"\nCALENDAR %d\n", year);
        daycode = find_day(year);
        leapyear(year);
        cal1(year, daycode);
    }
    else if(argc == 3)
    {
        month = atoi(argv[1]);
        year = atoi(argv[2]);
        printf(1,"\nCALENDAR %d\n", year);
        daycode = find_day(year);
        leapyear(year);
    }
}

```

```

        cal2(year, daycode, month);
    }
    else
    {
        printf(1,"Invalid Format\n");
        return 1;
    }
    printf(1,"\n");
    exit();
}

```

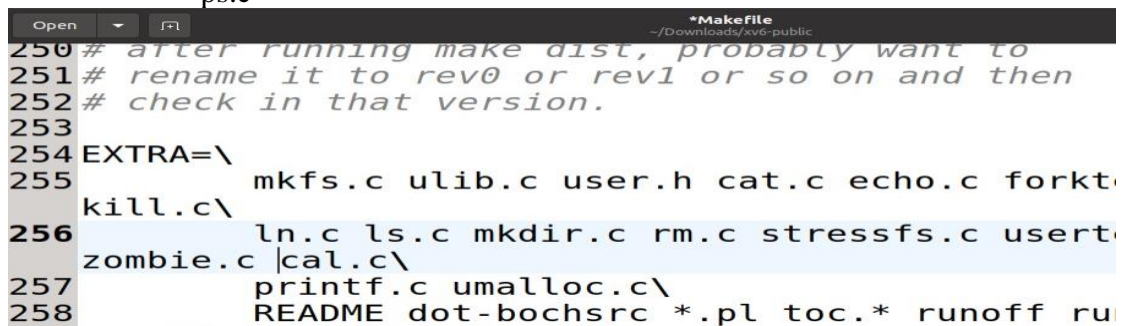
## 2.) Now open the Makefile command in the terminal:--

Add Under UPROGS

\_ps/

Add Under EXTRA

ps.c



A screenshot of a text editor window titled '\*Makefile' with the path '~/Downloads/xv6-public'. The editor shows a Makefile with line numbers 250 to 258. Line 254 contains 'EXTRA=\'. Line 255 contains 'mkfs.c ulib.c user.h cat.c echo.c forktest.c kill.c\' and line 256 contains 'ln.c ls.c mkdir.c rm.c stressfs.c usertests.c zombie.c\'.

```

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 kill.c\
256     ln.c ls.c mkdir.c rm.c stressfs.c usertests.c zombie.c\
257     printf.c umalloc.c\
258     README dot-bochsrc *.pl toc.* runoff run

```



A screenshot of a text editor window titled '\*Makefile' with the path '~/Downloads/xv6-public'. The editor shows a Makefile with line numbers 167 to 185. Line 168 contains 'UPROGS=\'. Lines 169 to 184 list various programs: \_cat\, \_echo\, \_forktest\, \_grep\, \_init\, \_kill\, \_ln\, \_ls\, \_mkdir\, \_rm\, \_sh\, \_stressfs\, \_usertests\, \_wc\, \_zombie\, and \_cal\.

```

167
168 UPROGS=\
169     _cat\
170     _echo\
171     _forktest\
172     _grep\
173     _init\
174     _kill\
175     _ln\
176     _ls\
177     _mkdir\
178     _rm\
179     _sh\
180     _stressfs\
181     _usertests\
182     _wc\
183     _zombie\
184     _cal\
185

```

## 3.) Now write the following:--

- Make clean
- Make qemu-nox

## OUTPUT:--

```
$ cal
CALENDAR 2022

January

Sun  Mon  Tue  Wed  Thu  Fri  Sat
      1
 2   3   4   5   6   7   8
 9  10  11  12  13  14  15
16  17  18  19  20  21  22
23  24  25  26  27  28  29
30  31
```

```
$ cal 2022
CALENDAR 2022

January

Sun  Mon  Tue  Wed  Thu  Fri  Sat
 2   3   4   5   6   7   8   1
 9  10  11  12  13  14  15
16  17  18  19  20  21  22
23  24  25  26  27  28  29
30  31
February

Sun  Mon  Tue  Wed  Thu  Fri  Sat
 6   7   8   9  10  11  12
13  14  15  16  17  18  19
20  21  22  23  24  25  26
27  28
March

Sun  Mon  Tue  Wed  Thu  Fri  Sat
 6   7   8   9  10  11  12
13  14  15  16  17  18  19
20  21  22  23  24  25  26
27  28  29  30  31
```

```
27  28  29  30  31
April

Sun  Mon  Tue  Wed  Thu  Fri  Sat
 3   4   5   6   7   8   9   1
10  11  12  13  14  15  16   2
17  18  19  20  21  22  23
24  25  26  27  28  29  30

May

Sun  Mon  Tue  Wed  Thu  Fri  Sat
 1   2   3   4   5   6   7
 8   9  10  11  12  13  14
15  16  17  18  19  20  21
22  23  24  25  26  27  28
29  30  31

June

Sun  Mon  Tue  Wed  Thu  Fri  Sat
 5   6   7   8   9  10  11
12  13  14  15  16  17  18
19  20  21  22  23  24  25
26  27  28  29  30
```





4. Create a system call called "waitpid(int pid)" which will wait for specific child as passed as parameter to this system call. Write a program to test this system call. If one pass the pid as 0 then it will wait for all its child. This will return how many child processes a parent could wait plus

#### Changes in syscall.h :-

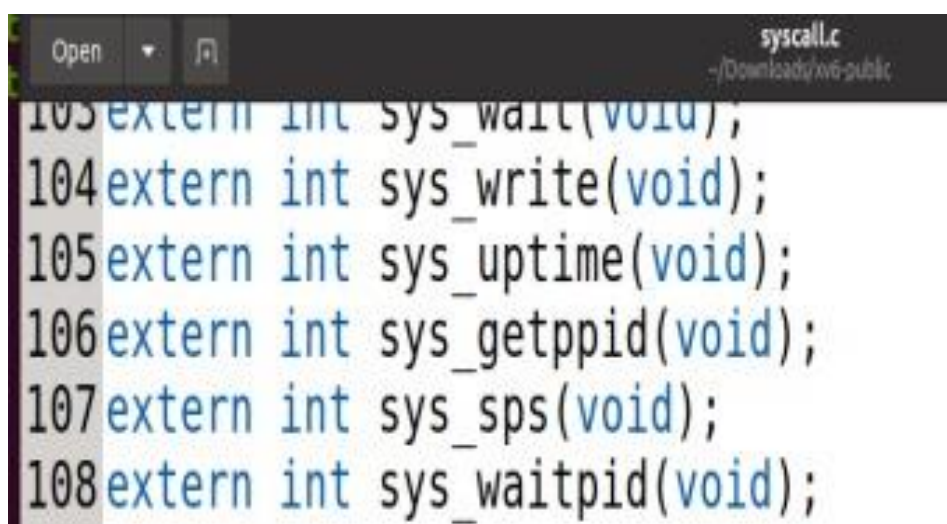


```
18 #define SYS_mknod 17
19 #define SYS_unlink 18
20 #define SYS_link 19
21 #define SYS_mkdir 20
22 #define SYS_close 21
23 #define SYS_getppid 22
24 #define SYS_sps 23
25 #define SYS_waitpid 24
```

The screenshot shows a text editor window titled 'syscall.h' with a file path of '~/.Downloads/n6-public'. The code defines several system call numbers. Line 25 shows the addition of 'SYS\_waitpid 24'.

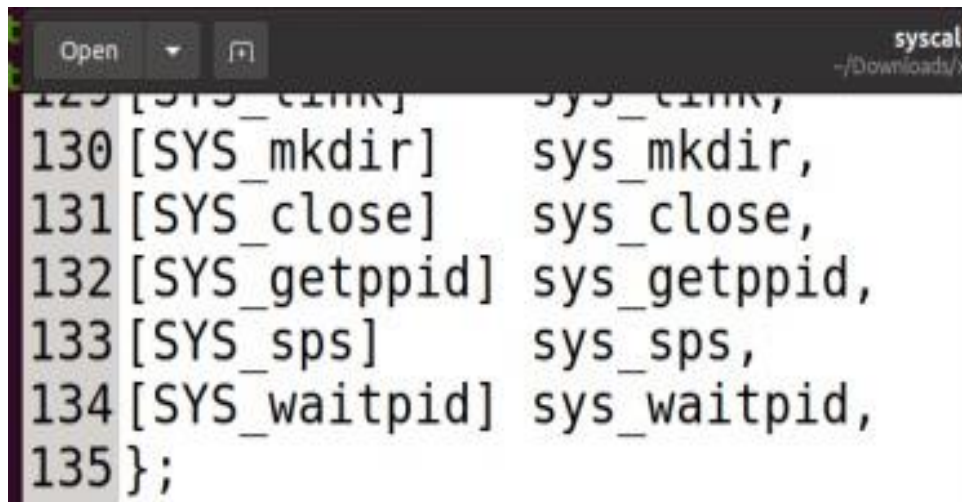
Step 2: Now open the file syscall.c file and add 2 statements as below:

extern int sys\_waitpid(void);  
[SYS\_waitpid] sys\_waitpid,



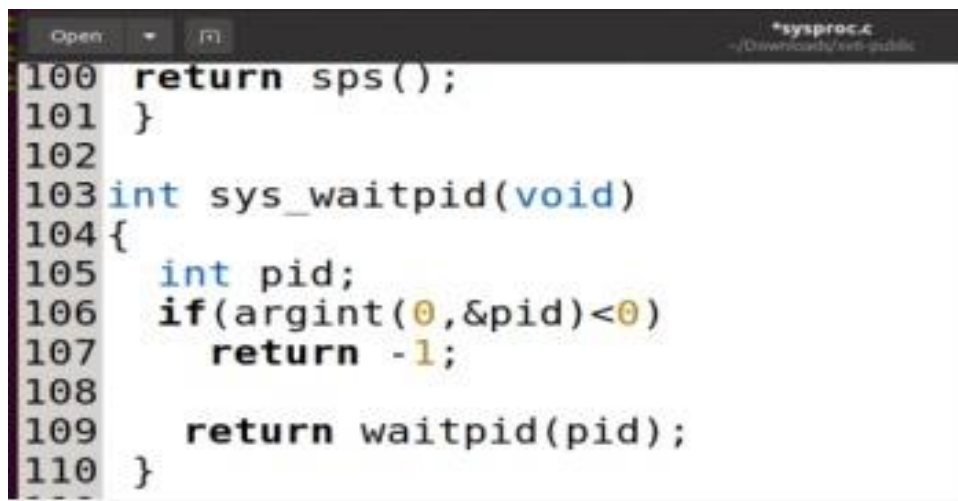
```
103 extern int sys_wait(void);
104 extern int sys_write(void);
105 extern int sys_uptime(void);
106 extern int sys_getppid(void);
107 extern int sys_sps(void);
108 extern int sys_waitpid(void);
```

The screenshot shows a text editor window titled 'syscall.c' with a file path of '~/.Downloads/n6-public'. The code lists several system call prototypes. Line 108 shows the addition of 'extern int sys\_waitpid(void);'.



```
129 [SYS_link]      sys_link,  
130 [SYS_mkdir]     sys_mkdir,  
131 [SYS_close]     sys_close,  
132 [SYS_getppid]   sys_getppid,  
133 [SYS_sps]       sys_sps,  
134 [SYS_waitpid]   sys_waitpid,  
135 };
```

#### Changes in sysproc.c :-



```
100 return sps();  
101 }  
102  
103 int sys_waitpid(void)  
104 {  
105     int pid;  
106     if(argint(0, &pid) < 0)  
107         return -1;  
108  
109     return waitpid(pid);  
110 }
```

#### Changes in usys.S :-



```
30 SYSCALL(sleep)  
31 SYSCALL(uptime)  
32 SYSCALL(getppid)  
33 SYSCALL(sps)  
34 SYSCALL(waitpid)
```

### Changes in user.h :-

A screenshot of a code editor window titled 'user.h' with a path of '~/.Downloads/xv6-public'. The editor shows a list of function declarations. Line 21 is 'int dup(int);', line 22 is 'int getpid(void);', line 23 is 'char\* sbrk(int);', line 24 is 'int sleep(int);', line 25 is 'int uptime(void);', line 26 is 'int getppid(void);', line 27 is 'int sps(void);', and line 28 is 'int waitpid(int);'. Line 29 is currently empty and highlighted in blue.

```
21 int dup(int);
22 int getpid(void);
23 char* sbrk(int);
24 int sleep(int);
25 int uptime(void);
26 int getppid(void);
27 int sps(void);
28 int waitpid(int);
29
```

### Changes in defs.h :-

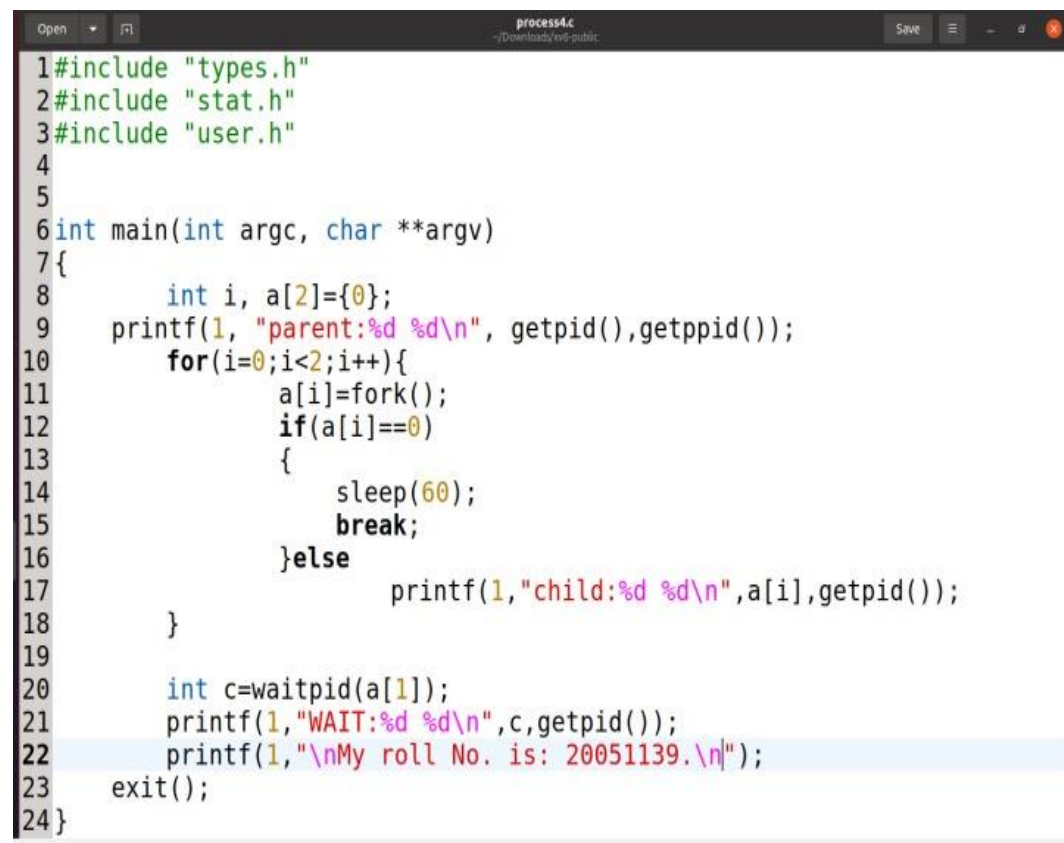
A screenshot of a code editor window titled 'defs.h' with a path of '~/.Downloads/xv6-public'. The editor shows several function declarations. Line 187 is 'void clearpteu(pde\_t \*pgdir, char \*uva);'. Line 188 is 'int sps(void);'. Line 189 is 'int waitpid(int);'. Line 190 is partially visible and highlighted in blue.

```
uint);
187 void
    clearpteu(pde_t *pgdir, char
        *uva);
188 int          sps(void);
189 int          waitpid(int);|
190
```

## Creating file proc.c :-

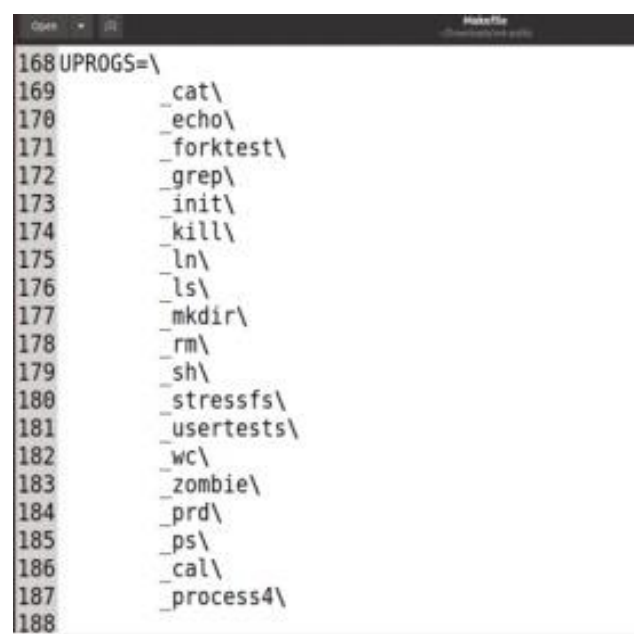
```
560
561 int waitpid(int cpid)
562 {
563     struct proc *p;
564     int child, pid;
565     struct proc *curproc=myproc();
566
567     acquire(&ptable.lock);
568     for(;;){
569         child=0;
570         for(p=ptable.proc;p<&ptable.proc[NPROC];p++){
571             if(p->pid!=cpid || p->parent!=curproc)
572                 continue;
573             child=1;
574             if(p->state==ZOMBIE){
575                 pid=p->pid;
576                 kfree(p->kstack);
577                 p->kstack=0;
578                 freevm(p->pgdir);
579                 p->pid=0;
580                 p->parent=0;
581                 p->name[0]=0;
582                 p->killed=0;
583                 p->state=UNUSED;
584                 release(&ptable.lock);
585
586                 return pid;
587             }
588
589             if(!child || curproc->killed){
590                 release(&ptable.lock);
591                 return -1;
592             }
593
594             sleep(curproc,&ptable.lock);
595         }
596     }
597
598
599
600
```

### Creating file process4.c :-



```
1#include "types.h"
2#include "stat.h"
3#include "user.h"
4
5
6int main(int argc, char **argv)
7{
8    int i, a[2]={0};
9    printf(1, "parent:%d %d\n", getpid(),getppid());
10    for(i=0;i<2;i++){
11        a[i]=fork();
12        if(a[i]==0)
13        {
14            sleep(60);
15            break;
16        }else
17            printf(1,"child:%d %d\n",a[i],getpid());
18    }
19
20    int c=waitpid(a[1]);
21    printf(1,"WAIT:%d %d\n",c,getpid());
22    printf(1,"\nMy roll No. is: 20051139.\n");
23    exit();
24}
```

### Finally open Makefile and apply the changes :-



```
168 UPROGS=\
169     _cat\
170     _echo\
171     _forktest\
172     _grep\
173     _init\
174     _kill\
175     _ln\
176     _ls\
177     _mkdir\
178     _rm\
179     _sh\
180     _stressfs\
181     _usertests\
182     _wc\
183     _zombie\
184     _prd\
185     _ps\
186     _cal\
187     _process4\
188
```

```
Firefox Web Browser
Makefile
~/Downloads/xv6-public
Save

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 zombie.c\
    prd.c ps.c cal.c process4.c\
```

## OUTPUT:--

```
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 start 58
init: starting sh
$ process4

My roll No. is: 20051139.
parent:3 2
child:4 3
WAIT:-1 4
child:5 3
WAIT:-1 5
WAIT:-1 3
$ █
```