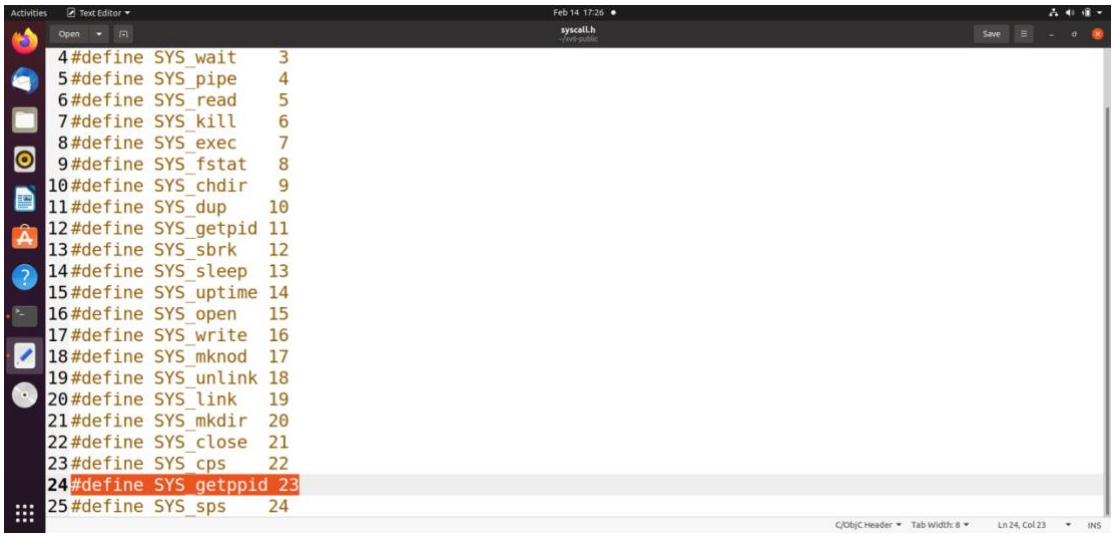
GROUP1 MEMBERS :-

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 :-



```
#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
#define SYS_getppid 23
#define SYS_sps    24
```

Changes in user.h :-

```c
int chdir(const char*);
int dup(int);
int getpid(void);
char* sbrk(int);
int sleep(int);
int uptime(void);
int cps(void);
int getppid(void);
int sps(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);
uint strlen(const char*);
void* memset(void*, int, uint);
void* malloc(uint);
void free(void*);
```

Changes in sysproc.c :-

```c
  xticks = ticks;
  release(&tickslock);
  return xticks;
}

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

int
sys_getppid(void)
{
 return myproc()->parent->pid;
}

int
sys_sps(void)
{
 return sps();
}
```

Changes in usys.S :-

```asm
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)
SYSCALL(getppid)
SYSCALL(sps)
```

Changes in syscall.c :-

```c
extern int sys_fork(void);
extern int sys_fstat(void);
extern int sys_getpid(void);
extern int sys_kill(void);
extern int sys_link(void);
extern int sys_mkdir(void);
extern int sys_mknod(void);
extern int sys_open(void);
extern int sys_pipe(void);
extern int sys_read(void);
extern int sys_sbrk(void);
extern int sys_sleep(void);
extern int sys_unlink(void);
extern int sys_wait(void);
extern int sys_write(void);
extern int sys_uptime(void);
extern int sys_cps(void);
extern int sys_getppid(void);
extern int sys_sps(void);

static int (*syscalls[])(void) = {
[SYS_fork]    sys_fork,
```

```c
[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,
[SYS_getppid] sys_getppid,
[SYS_sps]     sys_sps,
};

void
syscall(void)
{
```

Creating file prd.c :-

```c

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

int main()
{
 printf(1,"Current pid is : %d \n",getpid());
 printf(1,"Parent pid is : %d \n",getppid());

 exit();
}

```

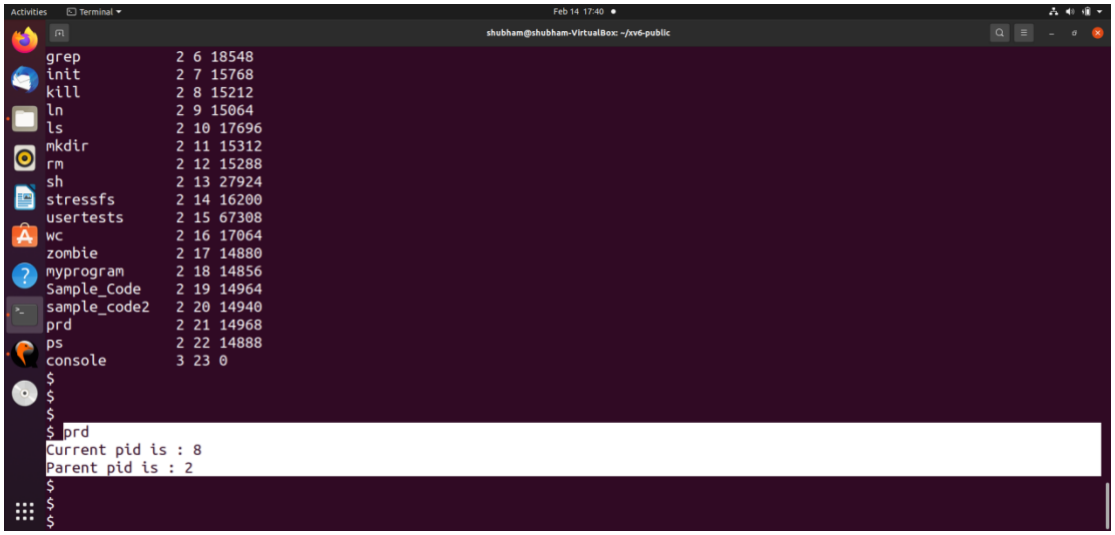
Changes in Makefile :-

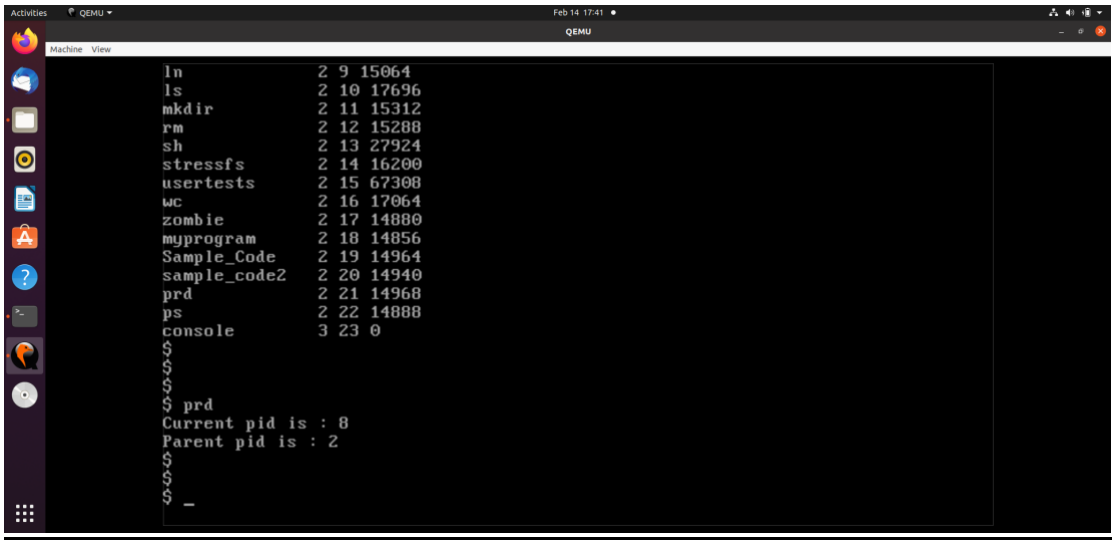
```
UPROGS=\
	_cat\
	_echo\
	_forktest\
	_grep\
	_init\
	_kill\
	_ln\
	_ls\
	_mkdir\
	_rm\
	_sh\
	_stressfs\
	_usertests\
	_wc\
	_zombie\
	_myprogram\
	_Sample_Code\
	_sample_code2\
	_prd\
	_ps\
```

```
# check in that version.

EXTRA=\
	mkfs.c ulib.c user.h cat.c echo.c forktest.c grep.c kill.c\
	ln.c ls.c mkdir.c rm.c stressfs.c usertests.c wc.c zombie.c\
	printf.c umalloc.c\myprogram.c\Sample_Code.c\sample_code2.c\prd.c\ps.c\
	README dot-bochsrc *.pl toc.* runoff runoff1 runoff.list\
	.gdbinit.tmpl gdbutil\

dist:
	rm -rf dist
	mkdir dist
	for i in $(FILES); \
	do \
		grep -v PAGEBREAK $$i >dist/$$i; \
	done
	sed '/CUT HERE/,$$d' Makefile >dist/Makefile
	echo >dist/runoff.spec
	cp $(EXTRA) dist

dist-test:
	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
$
```





2.) Create a ps command that will display the following. You need to prepare a systemcall called sps(system processes) that will provide the following information. PID, PPID, Process name, process state then you try to display the following Your roll no, PID, PPID, Process name, process state, process creation time, sizeofprocess memory.

Changes in syscall.h :-

Changes in defs.h :-

```
//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);
int             sps(void);
```

Changes in user.h :-

```
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);
int getppid(void);
int sps(void);

// ulib.c
int stat(const char*, struct stat*);
char* strcpy(char*, const char*);
void *memmove(void*, const void*, int);
```

Changes in sysproc.c :-

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

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

int
sys_getppid(void)
{
 return myproc()->parent->pid;
}

int
sys_sps(void)
{
 return sps();
}
```

Changes in usys.S :-

```
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)
SYSCALL(getppid)
SYSCALL(sps)
```

Changes in syscall.c :-

```
extern int sys_link(void);
extern int sys_mkdir(void);
extern int sys_mknod(void);
extern int sys_open(void);
extern int sys_pipe(void);
extern int sys_read(void);
extern int sys_sbrk(void);
extern int sys_sleep(void);
extern int sys_unlink(void);
extern int sys_wait(void);
extern int sys_write(void);
extern int sys_uptime(void);
extern int sys_cps(void);
extern int sys_getppid(void);
extern int sys_sps(void);

static int (*syscalls[])(void) = {
[SYS_fork]    sys_fork,
[SYS_exit]    sys_exit,
[SYS_wait]    sys_wait,
[SYS_pipe]    sys_pipe,
[SYS_read]    sys_read,
```

```
[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,
[SYS_getppid] sys_getppid,
[SYS_sps]     sys_sps,
};

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]();
```

Changes in proc.c :-

```c
int
sps()
{
 struct proc *p = myproc();

 cprintf("PID is %d \n",p->pid);
 cprintf("PPID is %d \n",p->parent->pid);
 cprintf("Process name is %s \n",p->name);

 //cprintf("Parent Process name is %s \n",p->parent->name);

 if(p->state == SLEEPING)
        cprintf("Process state is SLEEPING \n");

 else if(p->state == RUNNING)
        cprintf("Process state is RUNNING \n");

 else if(p->state == RUNNABLE)
        cprintf("Process state is RUNNABLE \n");

 return 24;
}
```

Creating ps.c file :-

```c

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

int main()
{
 printf(1,"\nMy Roll no is 20051488\n"); //Displaying mine roll number

 sps();

 exit();
}

```

Changes in Makefile :-

```
UPROGS=\
        _cat\
        _echo\
        _forktest\
        _grep\
        _init\
        _kill\
        _ln\
        _ls\
        _mkdir\
        _rm\
        _sh\
        _stressfs\
        _usertests\
        _wc\
        _zombie\
        _myprogram\
        _Sample_Code\
        _sample_code2\
        _prd\
        _ps\
```

```
qemu-nox-gdb: fs.img xv6.img .gdbinit
        @echo "*** Now run 'gdb'." 1>&2
        $(QEMU) -nographic $(QEMUOPTS) -S $(QEMUGDB)

# CUT HERE
# prepare dist for students
# after running make dist, probably want to
# rename it to rev0 or rev1 or so on and then
# check in that version.

EXTRA=\
        mkfs.c ulib.c user.h cat.c echo.c forktest.c grep.c kill.c\
        ln.c ls.c mkdir.c rm.c stressfs.c usertests.c wc.c zombie.c\
        printf.c umalloc.c\myprogram.c\Sample_Code.c\sample_code2.c\prd.c\ps.c\
        README dot-bochsrc *.pl toc.* runoff runoff1 runoff.list\
        .gdbinit.tmpl gdbutil\

dist:
        rm -rf dist
        mkdir dist
        for i in $(FILES); \
        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
$
$
```

```
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             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
$
```

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

```
Open    *Makefile  ~/Downloads/xv6-public
# after running make dist, probably want to
# rename it to rev0 or rev1 or so on and then
# check in that version.

EXTRA=\
        mkfs.c ulib.c user.h cat.c echo.c forkt
kill.c\
        ln.c ls.c mkdir.c rm.c stressfs.c usert
zombie.c cal.c\
        printf.c umalloc.c\
        README dot-bochsrc *.pl toc.* runoff ru
```

```
Open    *Makefile  ~/Downloads/xv6-public
UPROGS=\
        _cat\
        _echo\
        _forktest\
        _grep\
        _init\
        _kill\
        _ln\
        _ls\
        _mkdir\
        _rm\
        _sh\
        _stressfs\
        _usertests\
        _wc\
        _zombie\
        _cal\
```

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
                                 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
February
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
March
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
```

```
April
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

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
                 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
```

```
July

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
August

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
September

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
October

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
```

```
November

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
December

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

$
```

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 :-

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

```
extern int sys_wait(void);
extern int sys_write(void);
extern int sys_uptime(void);
extern int sys_getppid(void);
extern int sys_sps(void);
extern int sys_waitpid(void);
```

```
[SYS_mkdir]   sys_mkdir,
[SYS_close]   sys_close,
[SYS_getppid] sys_getppid,
[SYS_sps]     sys_sps,
[SYS_waitpid] sys_waitpid,
};
```

Changes in sysproc.c :-

```
  return sps();
}

int sys_waitpid(void)
{
  int pid;
  if(argint(0,&pid)<0)
    return -1;

   return waitpid(pid);
}
```

Changes in usys.S :-

```
SYSCALL(sleep)
SYSCALL(uptime)
SYSCALL(getppid)
SYSCALL(sps)
SYSCALL(waitpid)
```

Changes in user.h :-

```
int dup(int);
int getpid(void);
char* sbrk(int);
int sleep(int);
int uptime(void);
int getppid(void);
int sps(void);
int waitpid(int);
```

Changes in defs.h :-

```
uint);
void
clearpteu(pde_t *pgdir, char
*uva);
int             sps(void);
int             waitpid(int);
```

Creating file proc.c :-

```c

int waitpid(int cpid)
{
    struct proc *p;
    int child, pid;
    struct proc *curproc=myproc();

    acquire(&ptable.lock);
    for(;;){
      child=0;
      for(p=ptable.proc;p<&ptable.proc[NPROC];p++){
        if(p->pid!=cpid || p->parent!=curproc)
            continue;
        child=1;
        if(p->state==ZOMBIE){
           pid=p->pid;
           kfree(p->kstack);
           p->kstack=0;
           freevm(p->pgdir);
           p->pid=0;
           p->parent=0;
           p->name[0]=0;
           p->killed=0;
           p->state=UNUSED;
           release(&ptable.lock);
           return pid;
         }
      }

      if(!child ||curproc->killed){
        release(&ptable.lock);
        return -1;
      }

      sleep(curproc,&ptable.lock);
    }
}
```

Creating file process4.c :-

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


int main(int argc, char **argv)
{
        int i, a[2]={0};
    printf(1, "parent:%d %d\n", getpid(),getppid());
        for(i=0;i<2;i++){
                a[i]=fork();
                if(a[i]==0)
                {
                    sleep(60);
                    break;
                }else
                        printf(1,"child:%d %d\n",a[i],getpid());
        }

        int c=waitpid(a[1]);
        printf(1,"WAIT:%d %d\n",c,getpid());
        printf(1,"\nMy roll No. is: 20051139.\n");
    exit();
}
```

Finally open Makefile and apply the changes :-

```makefile
UPROGS=\
        _cat\
        _echo\
        _forktest\
        _grep\
        _init\
        _kill\
        _ln\
        _ls\
        _mkdir\
        _rm\
        _sh\
        _stressfs\
        _usertests\
        _wc\
        _zombie\
        _prd\
        _ps\
        _cal\
        _process4\
```

```
# after running make dist, probably want to
# rename it to rev0 or rev1 or so on and then
# check in that version.

EXTRA=\
        mkfs.c ulib.c user.h cat.c echo.c forktest.c grep.c kill.c\
        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
$
```