Operating System

Project -
XV6 Operating System

NAME: DEBASHREE CHAKRABORTY
BRANCH: CSE-3
SEMESTER:4

Ques 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*).

Answer 1:

Step1: Open **syscall.h** file to assign number to the system call getppid() in this Xv6 system.Add the command : **#define SYS_getppid 22**

```
// System call numbers
#define SYS_fork    1
#define SYS_exit    2
#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_getppid 22
```

Step2: Next, open the file **syscall.c file** and add 2 statements as below:

extern int sys_sps(void);

[SYS_getppid] sys_getppid,

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

```
[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_getppid] sys_getppid,
```

Step 3: Now open the file **sysproc.c file** to implement system call by writing the below code snippet.

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

Step 4: Open file called **usys.S** and add line **SYSCALL(getppid)** at the end.

```
kiit@kiit-VirtualBox:~/Downloads/xv6-public$ gedit usys.S
```

```
SYSCALL(mknod)
SYSCALL(unlink)
SYSCALL(fstat)
SYSCALL(link)
SYSCALL(mkdir)
SYSCALL(chdir)
SYSCALL(dup)
SYSCALL(getpid)
SYSCALL(sbrk)
SYSCALL(sleep)
SYSCALL(uptime)
SYSCALL(getppid)
```

Step 5: Next,open file called **user.h** and add **int getppid(void)** line. This is function that user program will be calling.

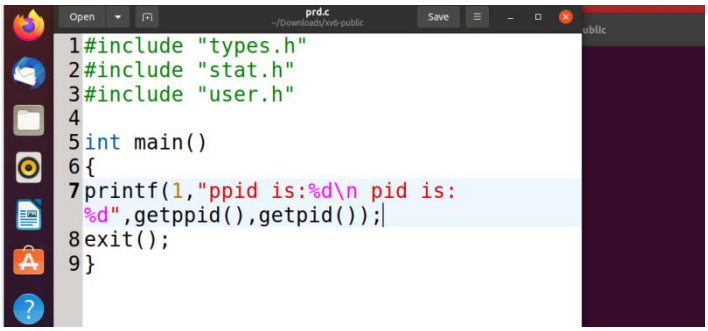
```
int getpid(void);
char* sbrk(int);
int sleep(int);
int uptime(void);
int getppid(void);
```

Step 6: After completing above procedure, we have **successfully added new system call named getppid() to xv6**.

```
kiit@kiit-VirtualBox:~/Downloads/xv6-public$ gedit syscall.h
kiit@kiit-VirtualBox:~/Downloads/xv6-public$ gedit syscall.c
kiit@kiit-VirtualBox:~/Downloads/xv6-public$ gedit sysproc.c
kiit@kiit-VirtualBox:~/Downloads/xv6-public$ gedit usys.S
kiit@kiit-VirtualBox:~/Downloads/xv6-public$ gedit user.h
kiit@kiit-VirtualBox:~/Downloads/xv6-public$
```

Step 7: However, in order to test functionality of this, we would need to add **user program named prd.c** which calls this system call.Inside the **prd.c** file write the below C code snippet:-

```
#include "types.h"
#include "stat.h"
#include "user.h"
int main()
{
  printf(1,"Ppid is:%d\nPid is:%d",getppid(),getpid());
  exit();
}
```



Step 8:Now open Makefile. Then, add prd.c under EXTRA and _prd\ under UPROGS.

```
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\
```

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

Step 9: Now write the commands - make followed by make qemu-nox.

```
kiit@kiit-VirtualBox: ~/Downloads/xv6-public
kiit@kiit-VirtualBox:~/Downloads/xv6-public$ gedit syscall.h
kiit@kiit-VirtualBox:~/Downloads/xv6-public$ gedit syscall.c
kiit@kiit-VirtualBox:~/Downloads/xv6-public$ gedit sysproc.c
kiit@kiit-VirtualBox:~/Downloads/xv6-public$ gedit usys.S
kiit@kiit-VirtualBox:~/Downloads/xv6-public$ gedit user.h
kiit@kiit-VirtualBox:~/Downloads/xv6-public$ gedit prd.c
kiit@kiit-VirtualBox:~/Downloads/xv6-public$ gedit Makefile
kiit@kiit-VirtualBox:~/Downloads/xv6-public$ make clean
rm -f *.tex *.dvi *.idx *.aux *.log *.ind *.ilg \
*.o *.d *.asm *.sym vectors.S bootblock entryother \
initcode initcode.out kernel xv6.img fs.img kernelmemfs \
xv6memfs.img mkfs .gdbinit \
_cat _echo _forktest _grep _init _kill _ln _ls _mkdir _rm _sh _stressfs _usertests _wc _zombie _prd _ps
kiit@kiit-VirtualBox:~/Downloads/xv6-public$ make qemu-nox
```

Step 10: Finally, on qemu terminal type the **command prd** to see the final output.

Final Output in Qemu for 1st question--

```
kiit@kiit-VirtualBox: ~/Downloads/xv6-public
SeaBIOS (version 1.13.0-1ubuntu1.1)


iPXE (http://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1FF8CA10+1FECCA10 CA00



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
$ prd
Ppid is:2
Pid is:3
$
```

Ques 2). Create a *ps* command that will display the following. You need to prepare a system call called ps(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, size of process memory.

Step1: Open **syscall.h** file to assign number to the system call getppid() in this Xv6 system.Add the command : **#define SYS_sps 23**

```
#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_getppid 22
#define SYS_sps 23
```

Step2: Now open the file **syscall.c file** and add 2 statements as below:

extern int sys_sps(void);
[SYS_sps] sys_sps,

```
extern int sys_uptime(void);
extern int sys_getppid(void);
extern int sys_sps(void);
```

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

Step 3: Now open the file **sysproc.c file** to implement system call by writing the below code snippet.

```
}

int sys_sps(void)

{
 return sps();
 }
```

Step 4: Open file called **usys.S** and add line **SYSCALL(sps)** at the end.

```
SYSCALL(dup)
SYSCALL(getpid)
SYSCALL(sbrk)
SYSCALL(sleep)
SYSCALL(uptime)
SYSCALL(getppid)
SYSCALL(sps)
```

Step 5: Next,open file called **user.h** and add int **sps(void)** line. This is function that user program will be calling.

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

Step 6: We need to now update the **proc.h** file by adding int time;

```
// Per-process state
struct proc {
  uint sz;                     // Size of process memory (bytes)
  pde_t* pgdir;                // Page table
  char *kstack;                // Bottom of kernel stack for this process
  enum procstate state;        // Process state
  int pid;                     // Process ID
  struct proc *parent;         // Parent process
  struct trapframe *tf;        // Trap frame for current syscall
  struct context *context;     // swtch() here to run process
  void *chan;                  // If non-zero, sleeping on chan
  int killed;                  // If non-zero, have been killed
  struct file *ofile[NOFILE];  // Open files
  struct inode *cwd;           // Current directory
  char name[16];               // Process name (debugging)
  int time;
};
```

Step 7: Now open the file **defs.h** and add the statement int sps(void);

```
pde_t*          copyuvm(pde_t*, uint);
void            switchuvm(struct proc*);
void            switchkvm(void);
int             copyout(pde_t*, uint, void*, uint);
void            clearpteu(pde_t *pgdir, char *uva);
int             sps(void);

// number of elements in fixed-size array
#define NELEM(x) (sizeof(x)/sizeof((x)[0]))
```

Step 8: Next open **proc.c** and **add p->time=ticks inside allocproc() and userinit() function**. Also, add int sys_ps() function to that prints desired details about process PID, PPID, Process name, process state, process creation time, size of process memory.

```
int sps(void)
{
  struct proc *p;
  sti();
  acquire(&ptable.lock);
  cprintf("\nPID- PPID- Name- State- Time- Size\n");
  for(p=ptable.proc;p<&ptable.proc[NPROC];p++)
  {
   if(p->state ==SLEEPING)
   cprintf("%d: %d: %s:SLEEPING: %d  %u\n",p->pid,p->parent->pid,
       p->name,p->ctime,p->sz);
   else if(p->state==RUNNABLE)
   cprintf("%d: %d: %s:RUNNABLE: %d  %u\n",p->pid,p->parent->pid,
       p->name,p->ctime,p->sz);
   else if(p->state==RUNNING)
   cprintf("%d: %d: %s:RUNNING: %d  %u\n",p->pid,p->parent->pid,
       p->name,p->ctime,p->sz);
  }
  release(&ptable.lock);
  return 0;
  }
```

```
//PAGEBREAK: 32
// Set up first user process.
void
userinit(void)
{
  struct proc *p;
  extern char _binary_initcode_start[], _binary_initcode_size[];

  p = allocproc();
  p->time=ticks;
  initproc = p;
  if((p->pgdir = setupkvm()) == 0)
    panic("userinit: out of memory?");
  inituvm(p->pgdir, _binary_initcode_start,
(int)_binary_initcode_size);
  p->sz = PGSIZE;
```

```c
static struct proc*
allocproc(void)
{
  struct proc *p;
  char *sp;

  acquire(&ptable.lock);

  for(p = ptable.proc; p < &ptable.proc[NPROC]; p++)
    if(p->state == UNUSED)
      goto found;

  release(&ptable.lock);
  return 0;

found:
  p->state = EMBRYO;
  p->pid = nextpid++;
  p->time=ticks;

  release(&ptable.lock);

  // Allocate kernel stack.
```

```c

int sps(void)
{
   struct proc *p;
   sti();
   acquire(&ptable.lock);
   cprintf("\nPID- PPID-   Name-       State-       Time-       Size\n");
   for(p=ptable.proc;p<&ptable.proc[NPROC];p++)
   {
    if(p->state ==SLEEPING)
    cprintf("%d    %d     %s       SLEEPING    %d          %d\n",p->pid,p->parent->pid,p->name,p->time,p->sz);
    else if(p->state==RUNNABLE)
    cprintf("%d     %d      %s        RUNNABLE    %d          %d\n",p->pid,p->parent->pid,p->name,p->time,p->sz);
    else if(p->state==RUNNING)
    cprintf("%d    %d     %s        RUNNING:       %d         %d\n",p->pid,p->parent->pid,p->name,p->time,p->sz);
   }
   release(&ptable.lock);
   return 0;
 }
```

Step 9: To test functionality of this, we would create a **command named ps.c** which calls this system call.Inside the **ps.c** file write the below C code snippet:-

```c
#include "types.h"
#include "stat.h"
#include "user.h"
int main(int argc,char **argv)
{
    sps();
  printf(1,"\n\n**My Roll no.- 20051139**\n\n");
  exit();
 }
```

```c
#include "types.h"
#include "stat.h"
#include "user.h"
int main(int argc,char **argv)
{

   sps();
   printf(1,"\n\nMy Roll no.- 20051139\n\n");
   exit();
  }
```

Step 10: Finally **open Makefile**. Then, add ps.c under EXTRA and _ps\ under UPROGS.

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

```
        ln.c ls.c mkdir.c rm.c stressfs.c usertests.c wc.c zombie.c ps.c\
        printf.c umalloc.c\
        README dot-bochsrc *.pl toc.* runoff runoff1 runoff.list\
        .gdbinit.tmpl gdbutil\
```

Step 11: Now type **make clean** followed by make qemu-nox to run qemu terminal.

```
kiit@kiit-VirtualBox: ~/Downloads/xv6-public
kiit@kiit-VirtualBox:~/Downloads/xv6-public$ gedit syscall.h
kiit@kiit-VirtualBox:~/Downloads/xv6-public$ gedit syscall.c
kiit@kiit-VirtualBox:~/Downloads/xv6-public$ gedit sysproc.c
kiit@kiit-VirtualBox:~/Downloads/xv6-public$ gedit usys.S
kiit@kiit-VirtualBox:~/Downloads/xv6-public$ gedit user.h
kiit@kiit-VirtualBox:~/Downloads/xv6-public$ gedit proc.h
kiit@kiit-VirtualBox:~/Downloads/xv6-public$ gedit defs.h
kiit@kiit-VirtualBox:~/Downloads/xv6-public$ gedit proc.c
kiit@kiit-VirtualBox:~/Downloads/xv6-public$ gedit ps.c
kiit@kiit-VirtualBox:~/Downloads/xv6-public$ gedit Makefile
kiit@kiit-VirtualBox:~/Downloads/xv6-public$ make clean
rm -f *.tex *.dvi *.idx *.aux *.log *.ind *.ilg \
*.o *.d *.asm *.sym vectors.S bootblock entryother \
initcode initcode.out kernel xv6.img fs.img kernelmemfs \
xv6memfs.img mkfs .gdbinit \
_cat _echo _forktest _grep _init _kill _ln _ls _mkdir _rm _sh _stressfs _usertests _wc _zombie _ps
kiit@kiit-VirtualBox:~/Downloads/xv6-public$ make qemu-nox
```

Step 12: Finally, on qemu terminal type the **command ps** to see the final output.

Final Output in Qemu for 2nd question--

```
kiit@kiit-VirtualBox: ~/Downloads/xv6-public
SeaBIOS (version 1.13.0-1ubuntu1.1)


iPXE (http://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1FF8CA10+1FECCA10 CA00



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
$ ps

PID- PPID-   Name-      State-       Time-       Size
1    -326938139     init       SLEEPING    0       12288
2    1      sh       SLEEPING    7          16384
3    2      ps        RUNNING:       539        12288


My Roll no.is: 20051139

$
```

Ques 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();
}
```

Step 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
```

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

Step 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

$
```

Ques 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 your roll no.

Step 1: Open **syscall.h** file to assign number to the system call waitpid() in this Xv6 system.Add the command : **#define SYS_waitpid 24**

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

Step 3: Now open the file **sysproc.c file** to implement system call by writing the below code snippet.

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

```
return sps();
}

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

  return waitpid(pid);
}
```

Step 4: Open file called **usys.S** and add line **SYSCALL(waitpid)** at the end.

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

Step 5: Next,open file called **user.h** and add **int waitpid(int)** line. This is function that user program will be calling.

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

Step 6: Now open the file **defs.h** and add the statement **int waitpid(int);**

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

Step 7: Next open **proc.c** and add **int sys_waitpid()** function to that prints desired details.

The code snippet is-

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);
  }
}
```

```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);
    }
}
```

Step 8: To test functionality of this, we would create a **command a C-program process4.c** which calls this system call.Inside the **process4.c** file write the below C code snippet:-

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

int main(int argc, char **argv)
{
        printf(1,"\nMy roll No. is: ####1139.\n");
        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());
                    wait();
            }
        }

        int c=waitpid(a[1]);
        printf(1,"WAIT:%d %d\n",c,getpid());

    exit();

}
```

process4.c
~/Downloads/xv6-public

```
#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();
}
```

Step 9: Finally **open Makefile**. Then, add process4.c under EXTRA and _process4\ under UPROGS.

```
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\
```

Step 10: Now type **make clean** followed by make qemu-nox inorder to run the qemu prompt.

```
kiit@kiit-VirtualBox:~/Downloads/xv6-public$ gedit syscall.h
kiit@kiit-VirtualBox:~/Downloads/xv6-public$ gedit syscall.c
kiit@kiit-VirtualBox:~/Downloads/xv6-public$ gedit sysproc.c
kiit@kiit-VirtualBox:~/Downloads/xv6-public$ gedit usys.S
kiit@kiit-VirtualBox:~/Downloads/xv6-public$ gedit user.h
kiit@kiit-VirtualBox:~/Downloads/xv6-public$ gedit defs.h
kiit@kiit-VirtualBox:~/Downloads/xv6-public$ gedit proc.c
kiit@kiit-VirtualBox:~/Downloads/xv6-public$ gedit process4.c
kiit@kiit-VirtualBox:~/Downloads/xv6-public$ gedit Makefile
kiit@kiit-VirtualBox:~/Downloads/xv6-public$ make clean
rm -f *.tex *.dvi *.idx *.aux *.log *.ind *.ilg \
*.o *.d *.asm *.sym vectors.S bootblock entryother \
initcode initcode.out kernel xv6.img fs.img kernelmemfs \
xv6memfs.img mkfs .gdbinit \
_cat _echo _forktest _grep _init _kill _ln _ls _mkdir _rm _sh _stressfs _usertests _wc _zombie _prd _ps _cal
kiit@kiit-VirtualBox:~/Downloads/xv6-public$ make qemu-nox
```

Step 11: Finally, on qemu terminal type the **command process4** to see the final output.

Final Output in Qemu for 4th question--

```
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
$
```