KALINGA INSTITUTE OF INDUSTRIAL TECHNOLOGY

Deemed to be University U/S 3 of the UGC Act, 1956

Operating System (Theory)

Assignment -1

NAME: DEBASHREE CHAKRABORTY

BRANCH: CSE-3

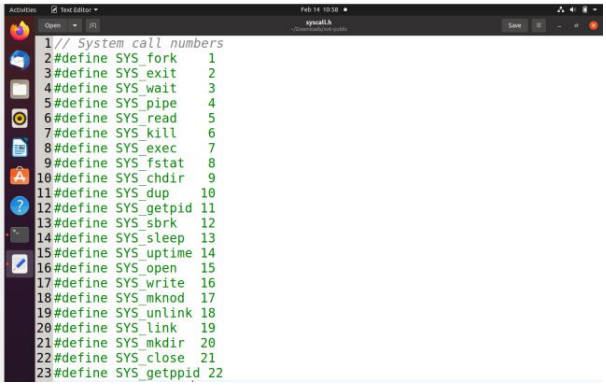
SEMESTER:4

ASSIGNMENT-1

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**

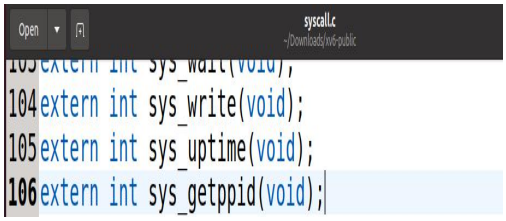


```
// System call numbers
#define SYS_fork    1
#define SYS_exit    2
#define SYS_wait    3
#define SYS_pipe    4
#define SYS_read    5
#define SYS_kill    6
#define SYS_exec    7
#define SYS_fstat   8
#define SYS_chdir   9
#define SYS_dup    10
#define SYS_getpid 11
#define SYS_sbrk   12
#define SYS_sleep  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
```

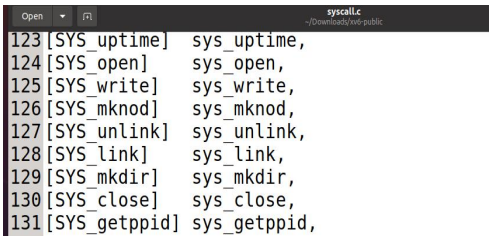
Step2: Next, open the file **syscall.c file** and add 2 statements as below:

extern int sys_sps(void);

[SYS_getppid] sys_getppid,

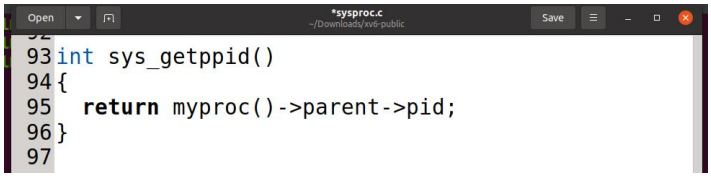


```
extern int sys_write(void);
extern int sys_uptime(void);
extern int sys_getppid(void);
```



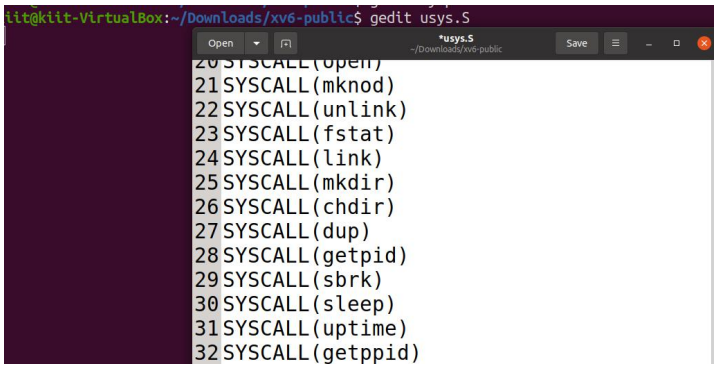
```
[SYS_uptime]  sys_uptime,
[SYS_open]    sys_open,
[SYS_write]   sys_write,
[SYS_mknod]   sys_mknod,
[SYS_unlink]  sys_unlink,
[SYS_link]    sys_link,
[SYS_mkdir]   sys_mkdir,
[SYS_close]   sys_close,
[SYS_getppid] sys_getppid,
```

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



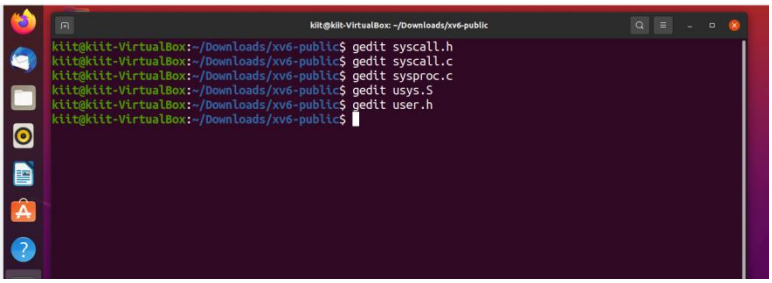
```
int sys_getppid()
{
  return myproc()->parent->pid;
}
```

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



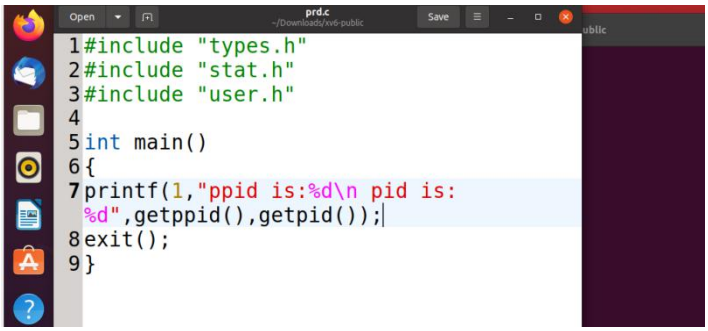
Step 5: Next,open file called **user.h** and add **int getppid(void)** line. This is function that user program will be calling.

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



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();
}
```

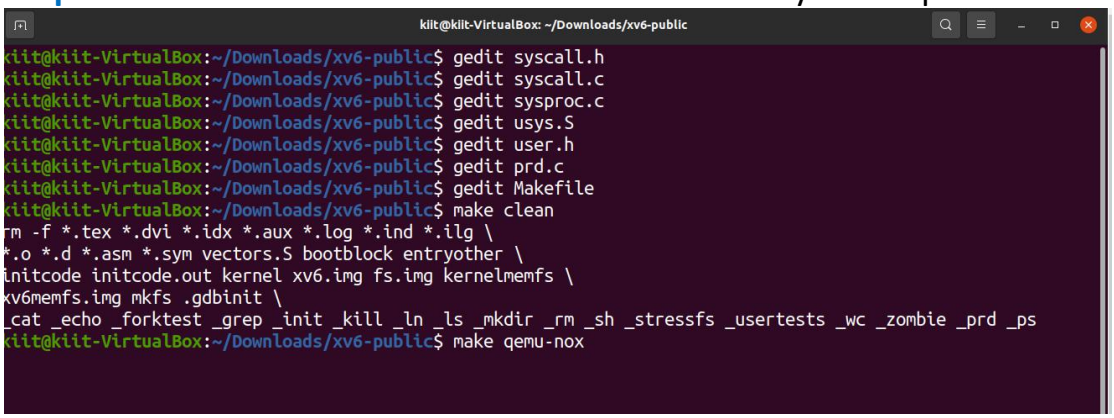


Step 8:Now open Makefile. Then, add prd.c under EXTRA and _prd\ under UPROGS.

```
251 EXTRA=\
252       mkfs.c ulib.c user.h cat.c echo.c forktest.c grep.c kill.c\
253       ln.c ls.c mkdir.c rm.c stressfs.c usertests.c wc.c zombie.c
  prd.c\
```

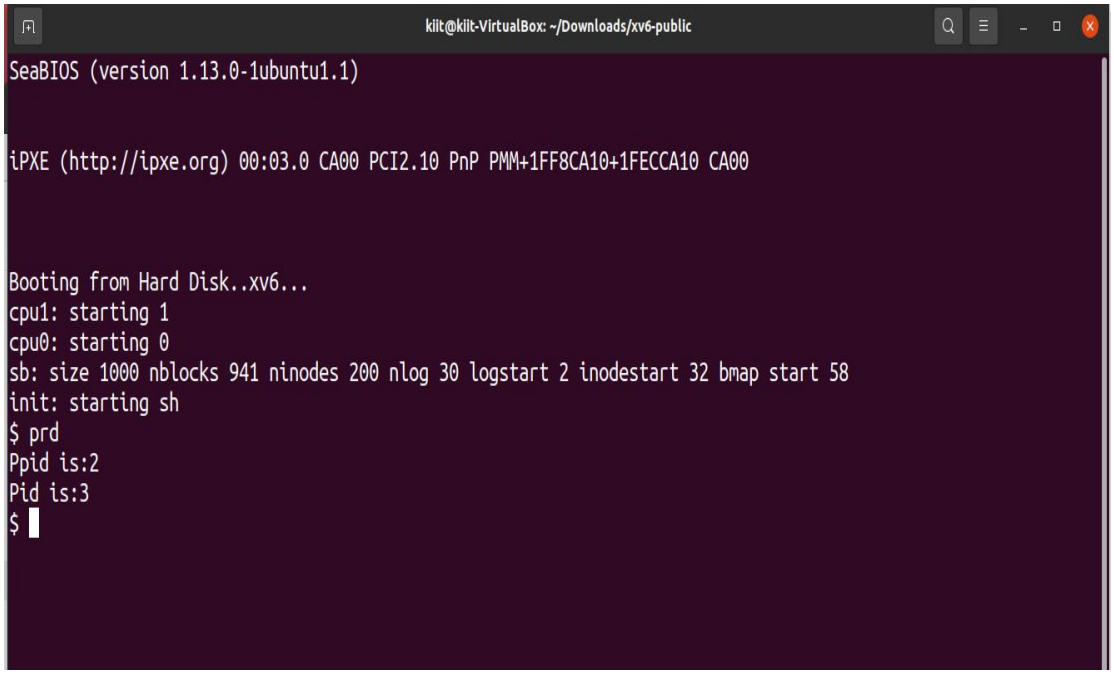
```
168 UPROGS=\
169         _cat\
170         _echo\
171         _forktest\
172         _grep\
173         _init\
174         _kill\
175         _ln\
176         _ls\
177         _mkdir\
178         _rm\
179         _sh\
180         _stressfs\
181         _usertests\
182         _wc\
183         _zombie\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

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



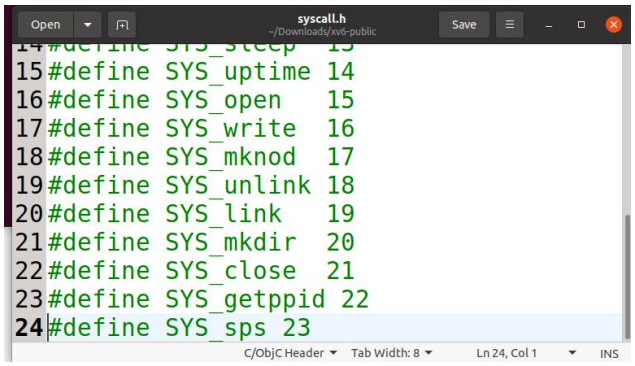
Step 10: Finally, on qemu terminal type the **command prd** to see the final output.

Final Output in Qemu for 1st question--



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.

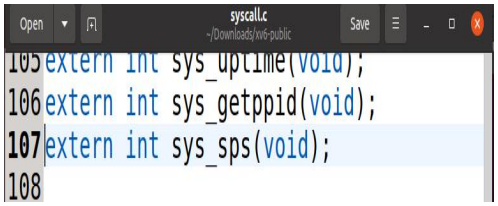
Step1: Open **syscall.h** file to assign number to the system call getppid() in this Xv6 system.Add the command : **#define SYS_sps 23**

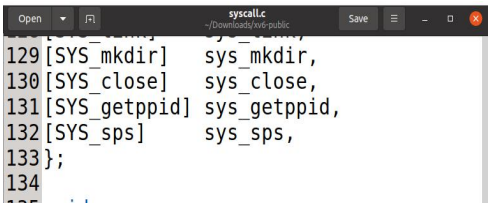


Step2: Now open the file **syscall.c file** and add 2 statements as below:

extern int sys_sps(void);

[SYS_sps] sys_sps,





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

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

```
*usys.S
~/Downloads/xv6-public
27 SYSCALL(dup)
28 SYSCALL(getpid)
29 SYSCALL(sbrk)
30 SYSCALL(sleep)
31 SYSCALL(uptime)
32 SYSCALL(getppid)
33 SYSCALL(sps)
34
```

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

```
user.h
~/Downloads/xv6-public
24 int sleep(int);
25 int uptime(void);
26 int getppid(void);
27 int sps(void);
```

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

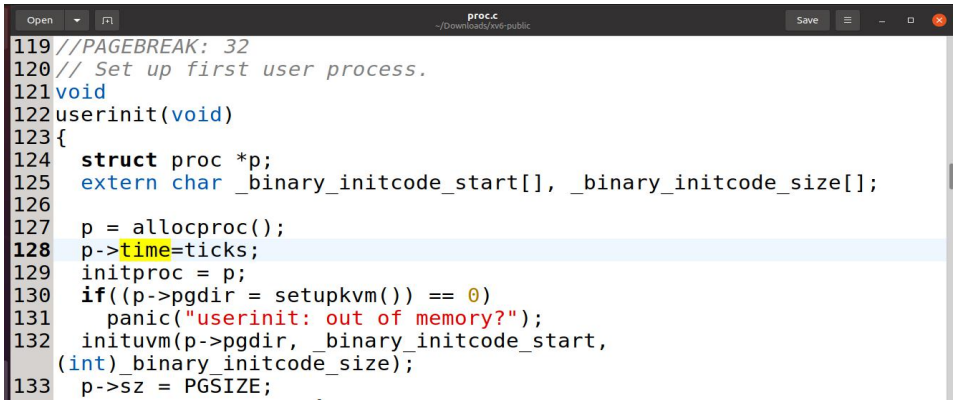
```
*proc.h
~/Downloads/xv6-public
36
37 // Per-process state
38 struct proc {
39   uint sz;                     // Size of process memory (bytes)
40   pde_t* pgdir;                // Page table
41   char *kstack;                // Bottom of kernel stack for this
   process
42   enum procstate state;        // Process state
43   int pid;                     // Process ID
44   struct proc *parent;         // Parent process
45   struct trapframe *tf;        // Trap frame for current syscall
46   struct context *context;     // swtch() here to run process
47   void *chan;                  // If non-zero, sleeping on chan
48   int killed;                  // If non-zero, have been killed
49   struct file *ofile[NOFILE];  // Open files
50   struct inode *cwd;           // Current directory
51   char name[16];               // Process name (debugging)
52   int time;
53 };
```

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

```
*defs.h
~/Downloads/xv6-public
183 pde_t*           copyuvm(pde_t*, uint);
184 void             switchuvm(struct proc*);
185 void             switchkvm(void);
186 int              copyout(pde_t*, uint, void*, uint);
187 void             clearpteu(pde_t *pgdir, char *uva);
188 int              sps(void);
189
190 // number of elements in fixed-size array
191 #define NELEM(x) (sizeof(x)/sizeof((x)[0]))
192
193
```

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;
   }
```



```
73 static struct proc*
74 allocproc(void)
75 {
76   struct proc *p;
77   char *sp;
78
79   acquire(&ptable.lock);
80
81   for(p = ptable.proc; p < &ptable.proc[NPROC]; p++)
82     if(p->state == UNUSED)
83       goto found;
84
85   release(&ptable.lock);
86   return 0;
87
88 found:
89   p->state = EMBRYO;
90   p->pid = nextpid++;
91   p->time=ticks;
92
93   release(&ptable.lock);
94
95   // Allocate kernel stack.
```

```
537
538 int sps(void)
539 {
540    struct proc *p;
541    sti();
542    acquire(&ptable.lock);
543    cprintf("\nPID- PPID-   Name-       State-       Time-
   Size\n");
544    for(p=ptable.proc;p<&ptable.proc[NPROC];p++)
545    {
546     if(p->state ==SLEEPING)
547     cprintf("%d     %d      %s       SLEEPING    %d          %d\n",p-
   >pid,p->parent->pid,p->name,p->time,p->sz);
548     else if(p->state==RUNNABLE)
549     cprintf("%d      %d       %s        RUNNABLE     %d
   %d\n",p->pid,p->parent->pid,p->name,p->time,p->sz);
550     else if(p->state==RUNNING)
551     cprintf("%d     %d      %s         RUNNING:        %d          %d\n",p-
   >pid,p->parent->pid,p->name,p->time,p->sz);
552    }
553    release(&ptable.lock);
554    return 0;
555 }
```

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:-

```
#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();
 }
```

Open ps.c ~/Downloads/xv6-public

```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\
```

Open Makefile ~/Downloads/xv6-public Save

```
        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--

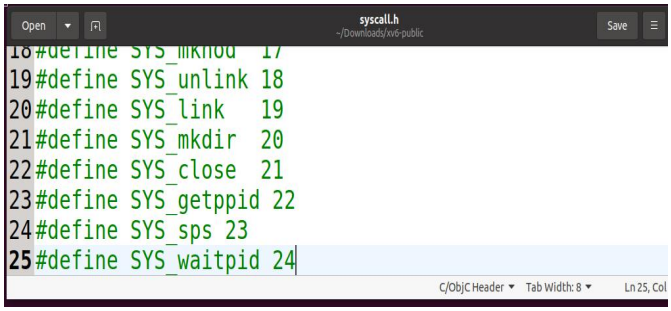
```
kiit@kiit-VirtualBox: ~/Downloads/xv6-public
SeaBIOS (version 1.13.0-1ubuntu1.1)


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:
Step 2:
Step 3:
Step 4:
Step 5:
Step 6:
Step 7:
Step 8:
Step 9:
Step 10:

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.

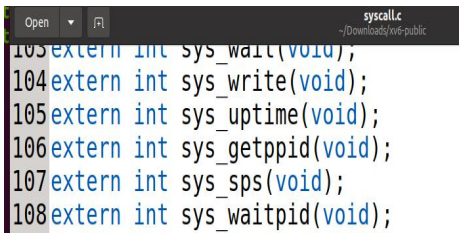
Step 1: Open **syscall.h** file to assign number to the system call waitpid() in this Xv6 system.Add the command : **#define SYS_waitpid 24**

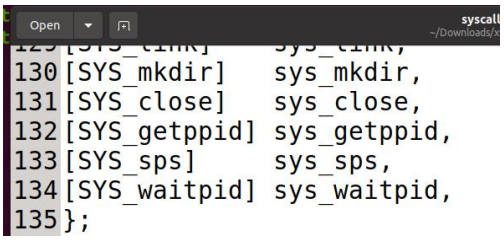


Step 2: Now open the file **syscall.c file** and add 2 statements as below:

extern int sys_waitpid(void);

[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);
 }
```

```
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 }
```

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

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

```
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
```

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

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

Open *proc.c ~/Downloads/xv6-public Save

```
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            return pid;
586          }
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
```

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:-

```
#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

```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();
}
```

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

```
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
```

```
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\
```

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
$
```