



VIT[®]

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

ADDING SYSTEM CALLS TO XV6OS

PROJECT REPORT

Submitted for the course: OPERATING SYSTEMS

By

SHAURYA CHOUDHARY	18BCE2113
ARYAN PATEL	18BCE0897
YASH RAJ SINGH	18BCE0206
ATHARVA JOSHI	18BCE0933

Slot: F1

Name of faculty: SRIMATHI C

October, 2019

TABLE OF CONTENTS

S.NO	TITLE	PAGE NO.
1.	ABSTRACT	3
2.	INTRODUCTION	4
2.1	INSTALLATION AND TOOLS USED	4
2.2	CONCEPT / METHODOLOGY TAKEN	4
3.	PREREQUISITES	5
4.	MODULES SHOWN IN PREVIOUS REVIEW i.e REVIEW 1	6
5.	MODULE IMPLEMENTED FOR FINAL REVIEW	9
6.	ISSUES FACED	20
7.	INFERENCES FROM THE PROJECT WORK	24
8.	REFERENCES	25

ABSTRACT

This project explain how to add a system call in Linux Kernel xv6 and covers prerequisites to add any system call in XV6 Kernel, Project also include Explanation of Process and commands required in kernel compilation, and briefly explain the process of system call addition, commands and packages used to do so. It also explains about the scheduling involved in the XV6 and how to change the scheduling policy of XV6 kernel. IT further explains about the data structure that could be used to store the data in the XV6 kernel.

INTRODUCTION

Xv6 is a teaching operating system developed in the summer of 2006 for MIT's operating systems course.

2.1 INSTALLATION AND TOOLS USED

Our team has Dell laptop which runs in Windows 10 Operating System. For the purpose of our OS project we dual booted our systems with different Linux Operating System which is Ubuntu. As our topic suggests we will now have to install xv6. We did not want to travel straight down the road by using a virtual setup in our Windows. By dual booting we were trying to explore possibilities and opening our path for more challenges.

Ubuntu Workstation from where Ubuntu OS .iso file is downloaded:

<https://www.ubuntu.com/download/desktop>

Software to burn .iso file into USB to make it bootable:

<https://www.pendrivelinux.com/universal-usb-installer-easy-as-1-2-3/>

Installation of QEMU and Xv6 will be discussed later in this report.

2.2 CONCEPT / METHODOLOGY TAKEN

TO ADD SYSTEM CALL IN LINUX KERNEL (XV6 OS):

To add a system call that can be called in xv6's shell, you should change the following files

- **sysproc.c** add the real implementation of your method here
- **syscall.h** define the position of the system call vector that connect to your implementation
- **user.h** define the function that can be called through the shell
- **syscall.c** external define the function that connect the shell and the kernel, use the position defined in **syscall.h** to add the function to the system call vector
- **usys.S** use the macro to define connect the call of user to the system call function
- **defs.h** add a forward declaration for your new system call
- **proc.c** add the function that needs the os to make or change the process

PREREQUISITES

There are certain requirements which must be fulfilled before adding a system call in Linux kernel XV6:

- a) Get XV6 OS from official MIT Github link.
- b) Set root password for Linux: After installation set root password by command: `sudo passwd root` and then execute `sudo passwd -u root` to unlock account.
- c) Access to root folder: There are many ways to do so, one among them is to go to terminal (by alt+tab+T) and type the following command: `sudo chmod -R 777/root`, after running this command one might see some error but when one will go to root folder, can have access to that folder but even after one gain access to root folder one can't create, delete or make changes in any existing file in root, usr or src folder.(to get this liberty refer next point)
- d) Permission to alter files in root, usr or src folder: There are various ways to get this but one may easily get this by typing command: `sudo nautilus`, in the terminal. After running this command, automatically a window will open and one will be able to do whatever one wants to do in root, src or usr folder.
- e) Installation of required packages: go to terminal and type command: `sudo apt-get install gcc`, `sudo apt-get update`, `sudo apt-get upgrade`.
- f) Extract the downloaded XV6 source code in home directory.

MODULES SHOWN IN 1ST REVIEW

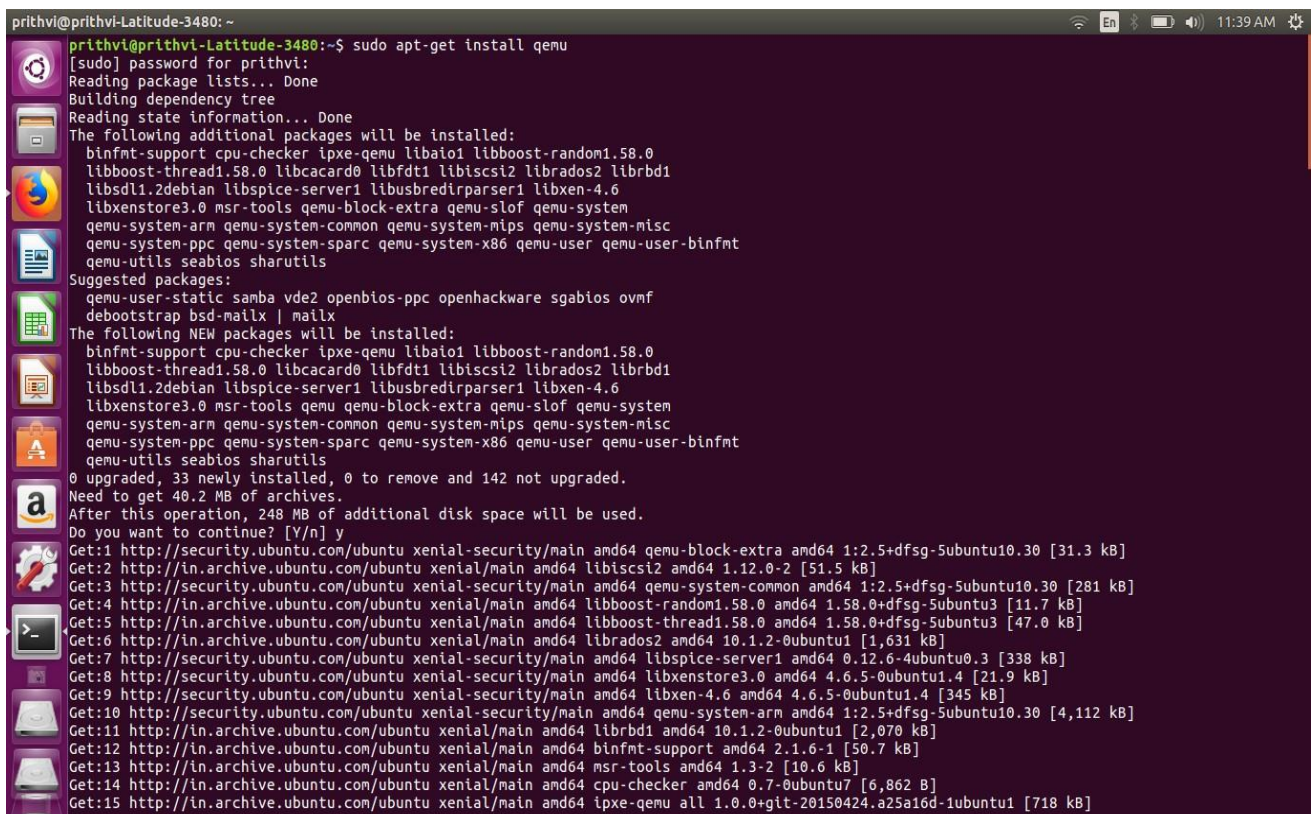
4.1 INSTALLATION OF QEMU

QEMU (short for Quick Emulator) is a free and open-source hosted hypervisor that performs hardware virtualization (not to be confused with hardware-assisted virtualization).

QEMU is a hosted virtual machine monitor: it emulates CPUs through dynamic binary translation and provides a set of device models, enabling it to run a variety of unmodified guest operating systems. It also can be used with KVM to run virtual machines at near-native speed (requiring hardware virtualization extensions on x86 machines). QEMU can also do CPU emulation for user-level processes, allowing applications compiled for one architecture to run on another.

STEPS:

```
sudo apt-get install qemu
```



```
prithvi@prithvi-Latitude-3480: ~  
prithvi@prithvi-Latitude-3480:~$ sudo apt-get install qemu  
[sudo] password for prithvi:  
Reading package lists... Done  
Building dependency tree  
Reading state information... Done  
The following additional packages will be installed:  
  binfmt-support cpu-checker ipxe-qemu libaio1 libboost-random1.58.0  
  libboost-thread1.58.0 libcacard0 libfdt1 libiscsi2 librados2 librbdl1  
  libsd11.2debian libspice-server1 libusbredirparser1 libxen-4.6  
  libxenstore3.0 msr-tools qemu-block-extra qemu-slof qemu-system  
  qemu-system-arm qemu-system-common qemu-system-mips qemu-system-misc  
  qemu-system-ppc qemu-system-sparc qemu-system-x86 qemu-user qemu-user-binfmt  
  qemu-utils seabios sharutils  
Suggested packages:  
  qemu-user-static samba vde2 openbios-ppc openhacking sgabios ovmf  
  debootstrap bsd-mailx | mailx  
The following NEW packages will be installed:  
  binfmt-support cpu-checker ipxe-qemu libaio1 libboost-random1.58.0  
  libboost-thread1.58.0 libcacard0 libfdt1 libiscsi2 librados2 librbdl1  
  libsd11.2debian libspice-server1 libusbredirparser1 libxen-4.6  
  libxenstore3.0 msr-tools qemu-block-extra qemu-slof qemu-system  
  qemu-system-arm qemu-system-common qemu-system-mips qemu-system-misc  
  qemu-system-ppc qemu-system-sparc qemu-system-x86 qemu-user qemu-user-binfmt  
  qemu-utils seabios sharutils  
0 upgraded, 33 newly installed, 0 to remove and 142 not upgraded.  
Need to get 40.2 MB of archives.  
After this operation, 248 MB of additional disk space will be used.  
Do you want to continue? [Y/n] y  
Get:1 http://security.ubuntu.com/ubuntu xenial-security/main amd64 qemu-block-extra amd64 1:2.5+dfsg-5ubuntu10.30 [31.3 kB]  
Get:2 http://in.archive.ubuntu.com/ubuntu xenial/main amd64 libiscsi2 amd64 1.12.0-2 [51.5 kB]  
Get:3 http://security.ubuntu.com/ubuntu xenial-security/main amd64 qemu-system-common amd64 1:2.5+dfsg-5ubuntu10.30 [281 kB]  
Get:4 http://in.archive.ubuntu.com/ubuntu xenial/main amd64 libboost-random1.58.0 amd64 1.58.0+dfsg-5ubuntu3 [11.7 kB]  
Get:5 http://in.archive.ubuntu.com/ubuntu xenial/main amd64 libboost-thread1.58.0 amd64 1.58.0+dfsg-5ubuntu3 [47.0 kB]  
Get:6 http://in.archive.ubuntu.com/ubuntu xenial/main amd64 librados2 amd64 10.1.2-0ubuntu1 [1,631 kB]  
Get:7 http://security.ubuntu.com/ubuntu xenial-security/main amd64 libspice-server1 amd64 0.12.6-4ubuntu0.3 [338 kB]  
Get:8 http://security.ubuntu.com/ubuntu xenial-security/main amd64 libxenstore3.0 amd64 4.6.5-0ubuntu1.4 [21.9 kB]  
Get:9 http://security.ubuntu.com/ubuntu xenial-security/main amd64 libxen-4.6 amd64 4.6.5-0ubuntu1.4 [345 kB]  
Get:10 http://security.ubuntu.com/ubuntu xenial-security/main amd64 qemu-system-arm amd64 1:2.5+dfsg-5ubuntu10.30 [4,112 kB]  
Get:11 http://in.archive.ubuntu.com/ubuntu xenial/main amd64 librbdl1 amd64 10.1.2-0ubuntu1 [2,070 kB]  
Get:12 http://in.archive.ubuntu.com/ubuntu xenial/main amd64 binfmt-support amd64 2.1.6-1 [50.7 kB]  
Get:13 http://in.archive.ubuntu.com/ubuntu xenial/main amd64 msr-tools amd64 1.3-2 [10.6 kB]  
Get:14 http://in.archive.ubuntu.com/ubuntu xenial/main amd64 cpu-checker amd64 0.7-0ubuntu7 [6,862 B]  
Get:15 http://in.archive.ubuntu.com/ubuntu xenial/main amd64 ipxe-qemu all 1.0.0+git-20150424.a25a16d-1ubuntu1 [718 kB]
```

4.2 INSTALLATION OF XV6

Xv6 is a teaching operating system developed in the summer of 2006 for MIT's operating systems course.

```
git clone https://github.com/mit-pdos/xv6-public.git xv6
```

```
prithvi@prithvi-Latitude-3480:~$ git clone https://github.com/mit-pdos/xv6-public.git xv6
Cloning into 'xv6'...
remote: Counting objects: 13906, done.
remote: Total 13906 (delta 0), reused 0 (delta 0), pack-reused 13906
Receiving objects: 100% (13906/13906), 17.08 MiB | 342.00 KiB/s, done.
Resolving deltas: 100% (9483/9483), done.
Checking connectivity... done.
```

Changing the mode to give extra permissions to xv6:

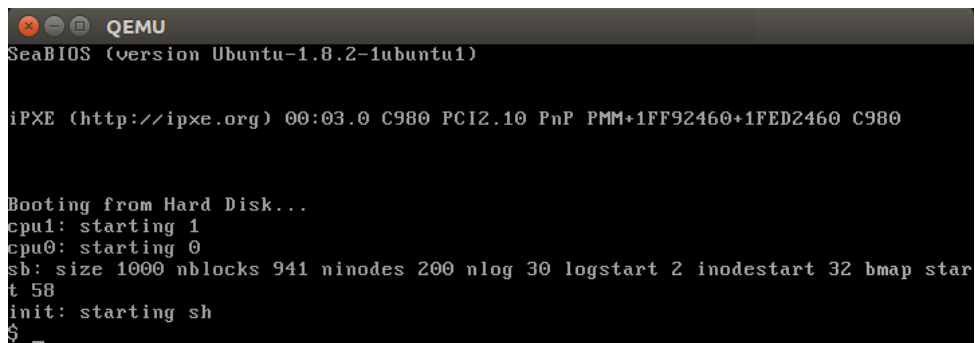
```
chmod 700 -R xv6
```

```
prithvi@prithvi-Latitude-3480:~$ chmod 700 -R xv6
```

4.3 RUNNING XV6

```
cd xv6
make
make qemu
```

```
prithvi@prithvi-Latitude-3480:~/xv6$ make qemu
qemu-system-i386 -serial mon:stdio -drive file=fs.img,index=1,media=disk,format=raw -drive file=xv6.img,index=0,media=disk,format=raw -smp 2 -m 512
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
```



```
QEMU
SeaBIOS (version Ubuntu-1.8.2-1ubuntu1)

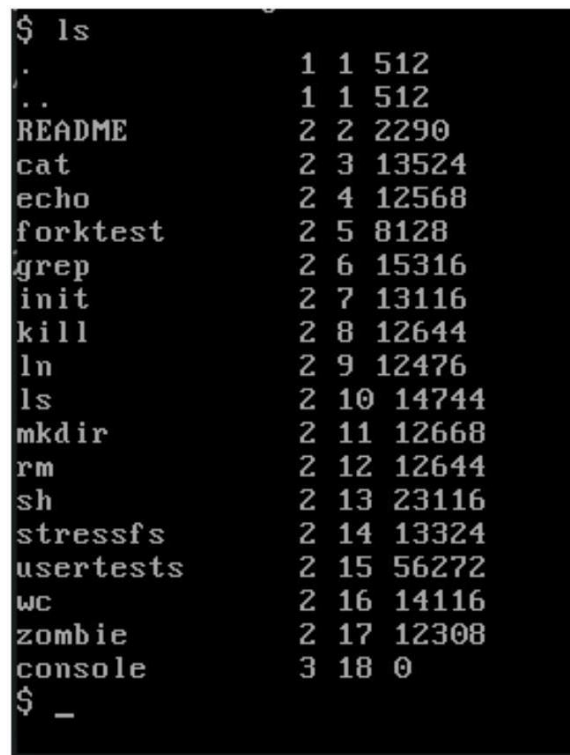
iPXE (http://ipxe.org) 00:03.0 C980 PCI2.10 PnP PMM+1FF92460+1FED2460 C980

Booting from Hard Disk...
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
$ _
```

Below is the list of the default system calls provided in the above installation of XV6:

System call	Description
fork()	Create process
exit()	Terminate current process
wait()	Wait for a child process to exit
kill(pid)	Terminate process pid
getpid()	Return current process's id
sleep(n)	Sleep for n seconds
exec(filename, *argv)	Load a file and execute it
sbrk(n)	Grow process's memory by n bytes
open(filename, flags)	Open a file; flags indicate read/write
read(fd, buf, n)	Read n bytes from an open file into buf
write(fd, buf, n)	Write n bytes to an open file
close(fd)	Release open file fd
dup(fd)	Duplicate fd
pipe(p)	Create a pipe and return fd's in p
chdir(dirname)	Change the current directory
mkdir(dirname)	Create a new directory
mknod(name, major, minor)	Create a device file
fstat(fd)	Return info about an open file
link(f1, f2)	Create another name (f2) for the file f1
unlink(filename)	Remove a file

Figure 0-2. Xv6 system calls



```

$ ls
.          1 1 512
..         1 1 512
README    2 2 2290
cat       2 3 13524
echo      2 4 12568
forktest  2 5 8128
grep      2 6 15316
init      2 7 13116
kill      2 8 12644
ln        2 9 12476
ls        2 10 14744
mkdir     2 11 12668
rm        2 12 12644
sh        2 13 23116
stressfs  2 14 13324
usertests 2 15 56272
wc        2 16 14116
zombie    2 17 12308
console   3 18 0
$ _

```

Directly from the console

MODULES SHOWN IN FINAL REVIEW

ADDING SYSTEM CALLS:

ps
listPid
parent
shutdown
foo
nice
insert

Changing Scheduling policy to priority scheduling

Step 1: Create file **ps.c**, **listPid.c**, **shutdown.c**, **parent.c**, **foo.c**, **nice.c**, **insert.c**

ps.c

```
#include "types.h"
#include "stat.h"
#include "user.h"
#include "fcntl.h"
int
main( int argc, char *argv[])
{
    cps();
    exit();}
```

listPid.c

```
#include "types.h"
#include "user.h"
#include "stat.h"
#include "fcntl.h"
#include "pstat.h"

int main()
{
    printf(1, "Process scheduling statistics:\n"); printf(1,
    "Slot\tPID\tHigh\tLow\n"); struct pstat st;
    getAllPids(&st);
    int i;
    for(i=0;i<NPROC;i++)
        if(st.inuse[i])
            printf(1, "%d\t%d\t%d\t%d\t%s\n",i,st.pid[i],st.hticks[i],s
            t.lticks[i],st.name[i]);
    exit();}
```

shutdown.c

```
#include "types.h"

#include "user.h"

#include "stat.h"
int main(int argc, char *argv[])

{
    shutdown();

    exit();

}
```

parent.c

```
#include "types.h"
#include "user.h"
#include "fcntl.h"
int main(void)
{
    int ChildPid=fork();
    if(ChildPid<0)
        printf(1,"Fork failed %d\n",ChildPid); else if (ChildPid >0) {
        printf(1,"I am the parent.My pid is %d, Child id is
        %d\n",getpid(),ChildPid);
        wait();
    }
    else
    {
        printf(1,"I am the child.My pid is %d, My parent id is
        d\n",getpid(),getppid());
    }
    exit();
}
```

foo.c

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

int main(int argc, char *argv[])
{
    int k,n,id;
    double x=0,z,d;
```

```

if(argc<2)
    n=1;
else
    n=atoi(argv[1]);
if (n<0 || n>20 )
    n=2;

if(argc<3)
    d=1.0;
else
    d=atoi(argv[2]);

x=0;
id=0;
for (k=0;k<n;k++)
{
    id=fork();
    if(id<0){
        printf(1,"%d failed in fork!\n", getpid() );
    }else if (id>0) {
        printf(1,"Parent %d creating child %d\n",getpid(),id);
        wait();
    }else{
        printf(1,"Child %d created \n",getpid() );
        for(z=0;z<8000000.0;z+= d)
            x=x+3.14*89.64;
        break;
    }
}
exit();
}

```

nice.c

```

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

int
main(int argc, char *argv[])
{
    int priority,pid;

    if (argc<3){
        printf(2,"Usage: nice pid priority\n");
        exit();
    }

    pid=atoi(argv[1]);
    priority=atoi(argv[2]);

```

```

if (priority<0 || priority>20){
printf(2,"Invalid priority(0-20)!\n");
exit();
}

printf(1,"pid=%d, pr=%d \n",pid,priority);
chpr(pid,priority);
exit();
}

```

Insert.c

```

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

#define MAX 50

int queue_array[MAX];

int rear = -1;

int front = -1;

int main(int num,char *argv[])

{
    int add_item;
    int i;
    add_item=atoi(argv[1]);
    if (rear == MAX - 1)
        printf(1,"Queue Overflow \n");
    else
    {
        if (front == -1)
            /*If queue is initially empty */
            front = 0;
        rear = rear + 1;
        queue_array[rear] = add_item;
    }
    if (front == -1)
        printf(1,"Queue is empty \n");
    else
    {
        printf(1,"Queue value inserted is : \n");
        for (i = front; i <= rear; i++)
            printf(1,"%d ", queue_array[i]);
        printf(1,"\n");
    }
}

```

```

        exit();
    }

```

Priority scheduling in proc.c:

```

void
scheduler(void)
{
    struct proc *p;
    struct proc *p1;
    struct cpu *c = mycpu();
    c->proc = 0;

    for(;;){
        // Enable interrupts on this processor.
        sti();
        struct proc *highP = NULL;
        // Loop over process table looking for process to run.
        acquire(&ptable.lock);
        for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
            if(p->state != RUNNABLE)
                continue;

            highP=p;
            for(p1=ptable.proc;p1<&ptable.proc[NPROC];p1++){
                if(p1->state != RUNNABLE)
                    continue;
                if(highP->priority>p1->priority)
                    highP=p1;
            }
            p=highP;

            // Switch to chosen process.  It is the process's job
            // to release ptable.lock and then reacquire it
            // before jumping back to us.
            c->proc = p;
            switchvm(p);
            p->state = RUNNING;

            switch(&(c->scheduler), p->context);
            switchkvm();

            // Process is done running for now.
            // It should have changed its p->state before coming
            back.
            c->proc = 0;
        }
        release(&ptable.lock);
    }
}

```

Step 2: Adding names and paths of above file in MakeFile

```
UPROGS=\...
...
...
_ps\
_shutdown\
_listPid\
_parent\
_foo\
_nice\
_insert\
_zombie\

EXTRA=\... wc.c ps.c shutdown.c listPid.c parent.c zombie.c\
foo.c nice.c insert.c\
```

Step 3: Add new system call to system call table in file **syscall.h**, **usys.S**, **syscall.c**

syscall.h

```
#define SYS_cps                22
#define SYS_getAllpids         23
#define SYS_shutdown           24
#define SYS_getppid            25
#define SYS_chpr               26
```

usys.S

```
SYSCALL(getppid)
SYSCALL(getAllPids)
SYSCALL(shutdown)
SYSCALL(cps)
SYSCALL(chpr)
```

syscall.c

```
extern int sys_cps(void);
extern int sys_shutdown(void);
extern int sys_getppid(void);
extern int sys_getAllPids(void);
extern int sys_chpr(void);

[SYS_cps]    sys_cps,
[SYS_getppid] sys_getppid,
[SYS_shutdown] sys_shutdown,
[SYS_getAllPids] sys_getAllPids,
[SYS_chpr]   sys_chpr,
```

Step 4: Add the relevant function call in **sysproc.c**

sysproc.c

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

struct pstat pstat;
int
sys_getAllpids(void)
{
    struct pstat *st;
    if(argptr(0, (void*)&st , sizeof(*st)) < 0)
        return -1;
    int i;
    for(i=0;i< NPROC; i++)
    {
        st -> inuse[i] = pstat.inuse[i];
        st -> pid[i] = pstat.pid[i];
        st -> name[i][0] = pstat.name[i][0];
        st -> name[i][1] = pstat.name[i][1];
        st -> name[i][2] = pstat.name[i][2];
        st -> hticks[i]= pstat.hticks[i];
        st -> lticks[i]= pstat.lticks[i];
    }
    return 0;
}

int
sys_shutdown(void)
{
    cprintf("Shutdown signal sent\n");
    char *s = "Shutdown";
    cprintf ("Powering off...\n");
    for (; *s != '\0'; s++)
    {
        outw (0xB004, 0x2000);
        outb (0x8900, *s);
    }
    return 0;
}

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

```

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

    return chpr( pid,pr );
}

```

Step 5: Changes to **proc.c**,**pstat.h** and **user.h**

proc.c

```

#include "pstat.h"
found:
p->state = EMBRYO;
p->pid = nextpid++;
//pstat.pid[pstat.n++]=nextpid;
pstat.inuse[p-htable.proc]=1;
pstat.pid[p-htable.proc]=p->pid;
pstat.name[p-htable.proc][0]=p->name[0];
pstat.name[p-htable.proc][1]=p->name[1]; pstat.name[p-
htable.proc][2]=p->name[2];
pstat.hticks[p-htable.proc]= 0;
pstat.lticks[p-htable.proc]= 0;
release(&htable.lock);
int
cps()
{
    struct proc *p;
    //Enable interrupts on this pros
    sti();
    //loop
    acquire(&htable.lock);
    cprintf("name \t pid \t state \t \n");
    for( p=htable.proc; p<&htable.proc[NPROC]; p++)
    {
        if(p->state == SLEEPING )
            cprintf("%s \t %d \t SLEEPING \t \n ", p->name,
                p->pid);
        else if(p->state == RUNNING )
            cprintf("%s \t %d \t RUNNING \t \n ", p->name,
                p->pid);
        else if(p->state == RUNNABLE )
            cprintf("%s \t %d \t RUNNABLE \t \n ", p->name,
                p->pid);
    }
    release(&htable.lock);
    return 22;
}

```


pstat.h

```
#ifndef _PSTAT_H_
#define _PSTAT_H_
#include "param.h"
struct pstat{
int inuse[NPROC];
int pid[NPROC];
char name[NPROC][16];
int hticks[NPROC];
int lticks[NPROC];
};
#endif
```

defs.h

//add the below lines under proc.c

```
int cps(void); //inserted
int chpr(int pid,int priority);
```

user.h

```
//add headers
struct pstat;

//systemcalls
int cps(void);
int getAllpids(struct pstat*);
int getppid(void);
int shutdown(void);
int chpr(int pid, int priority);
```

Now, we have modified all of our files and our next step is to make our xv6 folder again,
>>make
>>make qemu-nox

And we have modified XV6 with 4 new system calls, which are listed between **zombie & console**.

```
prithvi@prithvi-Latitude-3480:~/xv6$ make qemu-nox
qemu-system-i386 -nographic -drive file=fs.img,index=1,media=disk,format=raw -drive file=xv6.img,index=0,media=disk,format=raw -smp 2 -m 512
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
$ ls
.          1 1 512
..         1 1 512
README    2 2 2327
cat        2 3 13492
echo       2 4 12568
forktest   2 5 8284
grep       2 6 15320
init       2 7 13160
kill       2 8 12608
ln         2 9 12516
ls         2 10 14732
mkdir      2 11 12632
rm         2 12 12608
sh         2 13 23248
stressfs   2 14 13288
usertestfs 2 15 56168
wc         2 16 14144
zombie     2 17 12340
ps         2 18 12376
shutdown   2 19 12384
parent     2 20 12736
listPid    2 21 12960
foo        2 22 13432
nice       2 23 12796
insert     2 24 13236
console    3 25 0
```

Outputs

cps (gives the process states)

```
$ ps
name    pid    state    priority
init     1      SLEEPING    3
sh       2      SLEEPING    3
ps       4      RUNNING    3
$
```

parent (creates a child and a parent process with their respective pids)

```
$ parent
I am the parent.My pid is 5, Child id is 6
I am the child.My pid is 6, My parent id is 5
$
```

listPid (gives the process scheduling statistics of the os)

```
$ ps
name    pid    state    priority
init     1      SLEEPING    3
sh       2      SLEEPING    3
ps       4      RUNNING    3
$ parent
I am the parent.My pid is 5, Child id is 6
I am the child.My pid is 6, My parent id is 5
$ listPid
Process scheduling statistics:
Slot    PID    High    Low
0       1      0      0
1       2      0      0
2       7      0      0
3       6      0      0
$
```

Foo (creates processes in the os i.e each process has a child and parent)
In the below screenshot it created 2 processes.

```
$ foo 2 0.01 &;foo 2 0.01 &
$ Parent 15 creating child 17
Child 17 created
PChild 16 created
arent 14 creating child 16
Parent 14 creating child 18
Child 18 created
ps
name      pid      state      priority
init       1        SLEEPING      3
sh         2        SLEEPING      3
foo        18       RUNNABLE     10
foo        15       SLEEPING      3
foo        14       SLEEPING      3
foo        17       RUNNING     10
ps         19       RUNNING      3
```

nice and priority (nice is used to change the priority of the given process using its pid and in the below screenshot, we can see that by changing the priority of pid 10 to 5 it goes to the running state i.e priority scheduling takes place)

```
name      pid      state      priority
init       1        SLEEPING      3
sh         2        SLEEPING      3
foo        18       RUNNABLE     10
foo        15       SLEEPING      3
foo        14       SLEEPING      3
foo        17       RUNNING     10
ps         19       RUNNING      3
$ nice 18 5
pid=18, pr=5
$ ps
name      pid      state      priority
init       1        SLEEPING      3
sh         2        SLEEPING      3
foo        18       RUNNING      5
foo        15       SLEEPING      3
foo        14       SLEEPING      3
foo        17       RUNNABLE     10
ps         21       RUNNING      3
```

Shutdown

```
$ shutdown
Shutdown signal sent
Powering off...
$ █
```

Insert

```
prithvi@prithvi-Latitude-3480:~/xv6$ make qemu-nox
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-pointer -fno-stack-protector -fno-pie -
no-pie -c -o insert.o insert.c
ld -m elf_i386 -N -e main -Text 0 -o _insert insert.o ulib.o usys.o printf.o umalloc.o
objdump -S _insert > insert.asm
objdump -t _insert | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > insert.sym
./mkfs fs.img README _cat _echo _forktest _grep _init _kill _ln _ls _mkdir _rm _sh _stressfs _usertests _wc _zombie _ps _shutdown _parent _list
Pid _foo _nice _insert
nmeta 59 (boot, super, log blocks 30 inode blocks 26, bitmap blocks 1) blocks 941 total 1000
balloc: first 756 blocks have been allocated
balloc: write bitmap block at sector 58
qemu-system-i386 -nographic -drive file=fs.img,index=1,media=disk,format=raw -drive file=xv6.img,index=0,media=disk,format=raw -smp 2 -m 512
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
$ insert 7 !
Queue value inserted is :
7
$ insert 5 !
Queue value inserted is :
5
$ insert 100 !
Queue value inserted is :
100
```

ISSUES FACED

- 1) After we dual booted our dell laptop with Ubuntu, we had lost our windows operating system. We tried restarting the laptop but that did not help either. We searched various websites and we finally found a solution to the problem. We then updated our grub from the terminal by typing the command:

sudo update-grub

By doing this we solved our issue.

- 2) While downloading xv6 we faced an error as we did not have git preinstalled in Ubuntu and so we rectified our problem by downloading it from terminal:

```
prithvi@prithvi-Latitude-3480:~$ git clone https://github.com/mit-pdos/xv6-public.git xv6
The program 'git' is currently not installed. You can install it by typing:
sudo apt install git
```

```

prithvi@prithvi-Latitude-3480:~$ sudo apt install git
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  git-man liberror-perl
Suggested packages:
  git-daemon-run | git-daemon-sysvinit git-doc git-el git-email git-gui gitk gitweb git-arch git-cvs git-mediawiki git-svn
The following NEW packages will be installed:
  git git-man liberror-perl
0 upgraded, 3 newly installed, 0 to remove and 142 not upgraded.
Need to get 3,914 kB of archives.
After this operation, 25.6 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://security.ubuntu.com/ubuntu xenial-security/main amd64 git-man all 1:2.7.4-0ubuntu1.4 [736 kB]
Get:2 http://in.archive.ubuntu.com/ubuntu xenial/main amd64 liberror-perl all 0.17-1.2 [19.6 kB]
Get:3 http://security.ubuntu.com/ubuntu xenial-security/main amd64 git amd64 1:2.7.4-0ubuntu1.4 [3,158 kB]
Fetched 3,914 kB in 12s (317 kB/s)
Selecting previously unselected package liberror-perl.
(Reading database ... 176709 files and directories currently installed.)
Preparing to unpack .../liberror-perl_0.17-1.2_all.deb ...
Unpacking liberror-perl (0.17-1.2) ...
Selecting previously unselected package git-man.
Preparing to unpack .../git-man_1%3a2.7.4-0ubuntu1.4_all.deb ...
Unpacking git-man (1:2.7.4-0ubuntu1.4) ...
Selecting previously unselected package git.
Preparing to unpack .../git_1%3a2.7.4-0ubuntu1.4_amd64.deb ...
Unpacking git (1:2.7.4-0ubuntu1.4) ...
Processing triggers for man-db (2.7.5-1) ...
Setting up liberror-perl (0.17-1.2) ...
Setting up git-man (1:2.7.4-0ubuntu1.4) ...
Setting up git (1:2.7.4-0ubuntu1.4) ...

```

- 3) We also faced a problem while adding listPid. We had not declared a structure pstat correctly under user.h and proc.c and so we faced a problem,

```

prithvi@prithvi-Latitude-3480:~$ cd xv6
prithvi@prithvi-Latitude-3480:~/xv6$ make
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32
-Werror -fno-omit-frame-pointer -fno-stack-protector -fno-pie -no-pie -c -o pr
oc.o proc.c
proc.c: In function 'allocproc':
proc.c:95:3: error: 'pstat' undeclared (first use in this function)
  pstat.inuse[p-htable.proc]=1;
  ^
proc.c:95:3: note: each undeclared identifier is reported only once for each fun
ction it appears in
<builtin>: recipe for target 'proc.o' failed
make: *** [proc.o] Error 1

```

then we rectified the problem by adding the statement in user.h

struct pstat;

and made the following changes in proc.c


```

//PAGEBREAK: 32
// Look in the process table for an UNUSED proc.
// If found, change state to EMBRYO and initialize
// state required to run in the kernel.
// Otherwise return 0.
struct pstat pstat;
static struct proc*
allocproc(void)
{
    struct proc *p;
    char *sp;

```

- 4) The null pointer we usually use for the programs was not really declared in xv6 and so we faced a problem:

```

prithvi@prithvi-Latitude-3480:~$ cd xv6
prithvi@prithvi-Latitude-3480:~/xv6$ make qemu-nox
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-pointer -fno-stack-protector -fno-pie -no-pie -c -o console.o console.c
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-pointer -fno-stack-protector -fno-pie -no-pie -c -o exec.o exec.c
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-pointer -fno-stack-protector -fno-pie -no-pie -c -o fs.o fs.c
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-pointer -fno-stack-protector -fno-pie -no-pie -c -o ide.o ide.c
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-pointer -fno-stack-protector -fno-pie -no-pie -c -o kalloc.o kalloc.c
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-pointer -fno-stack-protector -fno-pie -no-pie -c -o lapic.o lapic.c
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-pointer -fno-stack-protector -fno-pie -no-pie -c -o main.o main.c
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-pointer -fno-stack-protector -fno-pie -no-pie -c -o mp.o mp.c
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-pointer -fno-stack-protector -fno-pie -no-pie -c -o pipe.o pipe.c
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-pointer -fno-stack-protector -fno-pie -no-pie -c -o proc.o proc.c
proc.c: In function 'scheduler':
proc.c:343:26: error: 'NULL' undeclared (first use in this function)
    struct proc *highP = NULL;
                        ^
proc.c:343:26: note: each undeclared identifier is reported only once for each function it appears in
<builtin>: recipe for target 'proc.o' failed
make: *** [proc.o] Error 1

```

And then we rectified this problem by defining the null pointer in the memory management unit of xv6 i.e mmu.h

```

// This file contains definitions for the
// x86 memory management unit (MMU).

```

```

// Eflags register
#define FL_IF          0x00000200    // Interrupt Enable
#define NULL          0000000000|

```

- 5) We also had the problem with the insert command which was storing only 3 even if we inserted any other value

```
$ insert 3 !
3
Queue is :
3
$ insert 100 !
3
Queue is :
3
```

The wrong code was

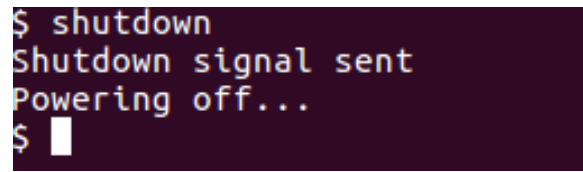
```
int main(int num,char *argv[])
{
    int add_item,i;
    if(num<100)
        add_item=num;
    else
        add_item=atoi(argv[1]);
    printf(1,"%d \n",add_item);
}
```

And then we rectified it by debugging the code (the code which is mentioned in the start under insert.c is the debugged code.

```
prithvi@prithvi-Latitude-3480:~/xv6$ make qemu-nox
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-pointer -fno-stack-protector -fno-pie -fno-pie -c -o insert.o insert.c
ld -m elf_i386 -N -e main -Ttext 0 -o _insert insert.o ulib.o usys.o printf.o umalloc.o
objdump -S _insert > insert.asm
objdump -t _insert | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > insert.sym
./mkfs fs.img README_cat_echo_forktest_grep_init_kill_ln_ls_mkdir_rm_sh_stressfs_usertests_wc_zombie_ps_shutdown_parent_list
Pid_foo_nice_insert
nmeta 59 (boot, super, log blocks 30 inode blocks 26, bitmap blocks 1) blocks 941 total 1000
ballocc: first 756 blocks have been allocated
ballocc: write bitmap block at sector 58
qemu-system-i386 -nographic -drive file=fs.img,index=1,media=disk,format=raw -drive file=xv6.img,index=0,media=disk,format=raw -smp 2 -m 512
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
$ insert 7 !
Queue value inserted is :
7
$ insert 5 !
Queue value inserted is :
5
$ insert 100 !
Queue value inserted is :
100
```

- 6) The project was running very smoothly until we arrived at the part to add the shutdown system call. The shutdown signal was sent but the qemu emulator did not power off. We are working towards a solution.

The screenshot for the same is shown below:

A terminal window with a dark purple background. The text displayed is: \$ shutdown, Shutdown signal sent, Powering off..., and \$ followed by a white cursor bar.

```
$ shutdown
Shutdown signal sent
Powering off...
$
```

INFERENCES FROM THE PROJECT WORK

While performing our project, we figured out that Kernel is the heart of OS since our project was related to modifying Kernel by the means of adding system calls. OS is itself a program which runs multiple programs having various number of files with .c,.h extensions. We can add features to any Open Source OS similarly. We also understood the importance of scheduling in the Operating System. The XV6 kernel uses Round Robin scheduling and we changed that to Priority scheduling and thus saw the importance of Round Robin scheduling in the OS. We learnt the importance of structures and also the basic layout of any OS. We also learnt that the OS requires important data structure techniques to store the data. We understood the importance of memory management in this OS by making some changes in 'mmu.h'. The project was challenging and fun and we got to learn a lot from the same.

REFERENCES

- [1]. www.google.co.in
- [2]. <https://www.ijsr.net/archive/v6i1/5011702.pdf>
- [3]. <http://moss.cs.iit.edu/cs450/assign01-xv6-syscall.html>
- [4]. <https://www.youtube.com/watch?v=21SVYiKhcwM>
- [5]. <https://www.youtube.com/watch?v=6zAHUcEt-QQ&t=438s>
- [6]. <https://namangt68.github.io/xv6/os/ubuntu/2016/05/08/quick-install-xv6.html>
- [7]. <https://stackoverflow.com/questions/8021774/how-do-i-add-a-system-call-utility-in-xv6>
- [8]. <https://pdos.csail.mit.edu/6.828/2012/xv6.html>
- [9]. <https://pdos.csail.mit.edu/6.828/2012/xv6/book-rev7.pdf>