SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

B. TECH COMPUTER SCIENCE ENGINEERING

CSE 2005 OPERATING SYSTEMS

XV6 IMPLEMENTATION USING QEMU EMULATOR

Submitted by:-

Prakriti Singh 19BAI1139

Arjun Tandon 19BAI1146

Rishabh Singhal 19BCE1827

Under the guidance of

Prof. Renukadevi Saravanan

Table of Contents

Abstract

XV6 is an operating system developed by MIT for the exact same purpose. It is an implementation of the Unix version 6 using ANSI C language for x86 platforms. It is a great resource for learning operating systems and many universities all over the world have already used it in their courses.

XV6 is a modern re-implementation of UNIX. It is basically functioning like a parallel OS compiler in which we can new system calls can be identified.

This implementation of the project will contain running xv6 in Qemu emulator and adding some system calls to the XV6.

Introduction

XV6 is a lightweight operating system. It was designed in MIT for educational purposes. It has an easy-to-understand structure.

One can run XV6 in three different ways:

1. XV6 can be installed as a standalone OS

2. XV6 can run in an emulated environment

3. QEMU is the emulator we use to run XV6

Generally, the first process is not recommended. Here we are using the third approach.

QEMU stands for Quick Emulator. QEMU is a free and open-source emulator that performs hardware virtualization, and it is best approach to implement the XV6.

XV6 has user-space-memory and per-process state that can be private to one kernel. XV6 is a time-sharing process it can share the time to the waiting processes and can execute them. XV6 saves the CPU’s registers if the process is not executing.

System call- is the programmatic request that requests the OS to execute. It provides an interface between a process and OS.

Some system calls and their description

- fork() - Create process

- exit() - Terminate current process

- wait() - wait till the child completes the execution

- kill(pid) - Terminate process pid

- getpid() - Return current process’s id

Literature Survey

We have read some articles on xv6 implementation and majorly used are listed below:

- “XV6: a simple, Unix-like teaching operating system” an article written by Russ

- Cox, FransKaashoek, Robert Morris which is published on October 27,2019.

- “Process synchronization in xv6”- this is a lab task that have some related codes in
that how to do synchronization.

- Lecture Notes on Operating systems which published by IIT Bombay “Processes and
Scheduling in XV6”

Existing work

There is already a work done in xv6 implementation on Qemu. Some them are:

- creating the system calls in xv6.

- Creating the multiple processes

- Process synchronization

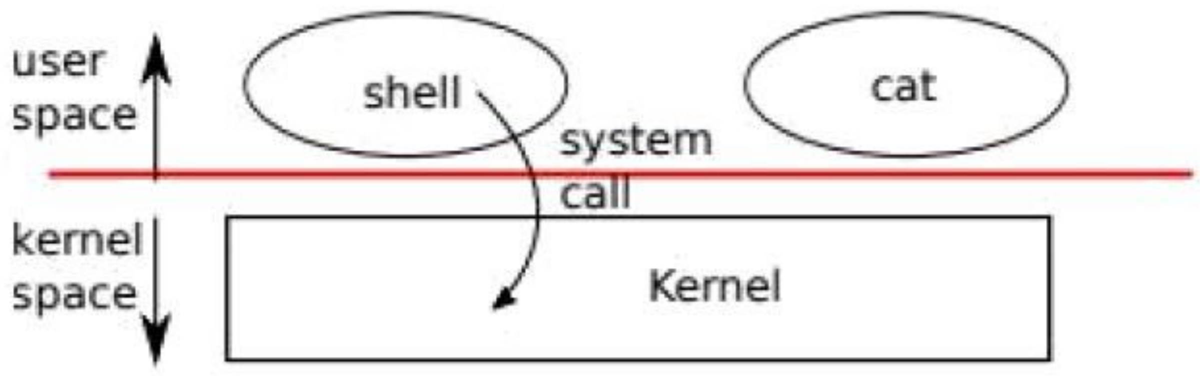
- Scheduling the processes

Proposed Work

- Implementing XV6 in Qemu
- Adding some system calls

Flow Chart

For system calls:



Working Module

- Implementation of XV6 in Qemu

- Adding system calls

Description of modules

- Implementation of XV6 in Qemu

We are running xv6 operating system on a QEMU virtual machine simply on top of a 64-bit Ubuntu 18.06 LTS machine. We have to install some tools and packages on our Ubuntu Linux system as follows:

- sudoapt-getupdate
- sudoapt-getinstallbuild-essential
- sudoapt-getinstallgcc-multilib
- sudoapt-getinstallqemu
- sudoapt-getinstallgit

Now, we will download the source code of the xv6 operating system. MIT provided an official git repository to download the source code.
git clone https://github.com/mit-pdos/xv6-public.git
Now, we compile and run xv6 system on QEMU emulator.

- Adding system calls
  In total we have 28 system calls in this operating system. The system calls are:

  - #define SYS_fork 1
  - #define SYS_exit
  - #define SYS_wait
  - #define SYS_pipe
  - #define SYS_read
  - #define SYS_kill
  - #define SYS_exec
  - #define SYS_fstat
  - #define SYS_chdir
  - #define SYS_dup
  - #define SYS_getpid 11
  - #define SYS_sbrk 12
  - #define SYS_sleep 13
  - #defineSYS_uptime14
  - #define SYS_open 15
  - #define SYS_write 16
  - #define SYS_mknod 17
  - #defineSYS_unlink18
  - #define SYS_link 19
  - #define SYS_mkdir 20
  - #define SYS_close 21
  - #defineSYS_getyear22
  - #define SYS_sum 23

- #define SYS_diff 24
- #define SYS_mul 25
- #define SYS_div 26
- #define SYS_rem 27
- #defineSYS_square28
- #defineSYS_cube29

Complete Code

• syscall.h

```
// 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_getyear 22
 #define SYS_sum 23
 #define SYS_diff 24
 #define SYS_mul 25
 #define SYS_div 26
 #define SYS_rem 27
 #define SYS_square 28
 #define SYS_cube 28
```

• syscall.c

```
#include "types.h"
#include "defs.h"
#include "param.h"
#include "memlayout.h"
#include "mmu.h"
#include "proc.h"
#include "x86.h"
#include "syscall.h"
//User code makes a system call with INT T_SYSCALL.
//System call number in %eax.
//Arguments on the stack, from the user call to the C
//library system call function. The saved user %esp points
//to a saved program counter, and then the first argument.
// Fetch the int at addr from the current process.

int
fetchint(uint addr, int *ip)
{
struct proc *curproc = myproc();
if(addr >= curproc->sz || addr+4 > curproc->sz)
return -1;
*ip = *(int*)(addr);
return 0;
}
// Fetch the nul-terminated string at addr from the current
process.
// Doesn't actually copy the string - just sets *pp to point at
it.
// Returns length of string, not including nul.
int
fetchstr(uint addr, char **pp)
{
char *s, *ep;
struct proc *curproc = myproc();
if(addr >= curproc->sz)
return -1;
*pp = (char*)addr;
ep = (char*)curproc->sz;
for(s = *pp; s < ep; s++){
if(*s == 0)
return s - *pp;
}
return -1;
}
// Fetch the nth 32-bit system call argument.
int
argint(int n, int *ip)
{
return fetchint((myproc()->tf->esp) + 4 + 4*n, ip);
}
// Fetch the nth word-sized system call argument as a pointer
// to a block of memory of size bytes. Check that the pointer
// lies within the process address space.
int
argptr(int n, char **pp, int size)
{
int i;
struct proc *curproc = myproc();
if(argint(n, &i) < 0)
return -1;
if(size < 0 || (uint)i >= curproc->sz || (uint)i+size > curproc-
>sz)
return -1;
*pp = (char*)i;
return 0;
}
// Fetch the nth word-sized system call argument as a string 
pointer.
// Check that the pointer is valid and the string is nul-
terminated.
// (There is no shared writable memory, so the string can't change
// between this check and being used by the kernel.)
int
```

```
argstr(int n, char **pp)
{
int addr;
if(argint(n, &addr) < 0)
return -1;
return fetchstr(addr, pp);
}
extern int sys_chdir(void);
extern int sys_close(void);
extern int sys_dup(void);
extern int sys_exec(void);
extern int sys_exit(void);
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_getyear(void);
extern int sys_sum(void);
extern int sys_diff(void);
extern int sys_mul(void);
extern int sys_div(void);
extern int sys_rem(void);
extern int sys_square(void);
extern int sys_cube(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_kill]
sys_kill,
[SYS_exec]
sys_exec,
[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_getyear] sys_getyear,
[SYS_sum] sys_sum,
[SYS_diff] sys_diff,
[SYS_mul] sys_mul,
[SYS_div] sys_div,
[SYS_rem] sys_rem,
[SYS_square] sys_square,
[SYS_cube] sys_cube,
};
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]();
} else {
cprintf("%d %s: unknown sys call %d\n",
curproc->pid, curproc->name, num);
curproc->tf->eax = -1;
}
}
```

• sysproc.c

```
#include "types.h"
#include "x86.h"
#include "defs.h"
#include "date.h"
#include "param.h"
#include "memlayout.h"
#include "mmu.h"
#include "proc.h"
int
sys_fork(void)
{
return fork();
}
int
sys_exit(void)
{
exit();
return 0; // not reached
```

```
}
int
sys_wait(void)
{
return wait();
}
int
sys_kill(void)
{
int pid;
if(argint(0, &pid) < 0)
return -1;
return kill(pid);
}
int
sys_getpid(void)
{
return myproc()->pid;
}
int
sys_sbrk(void)
{
int addr;
int n;
if(argint(0, &n) < 0)
return -1;
addr = myproc()->sz;
if(growproc(n) < 0)
return -1;
return addr;
}
int
sys_sleep(void)
{
int n;
uint ticks0;
if(argint(0, &n) < 0)
return -1;
acquire(&tickslock);
ticks0 = ticks;
```

```
while(ticks - ticks0 < n)
{
if(myproc()->killed)
{
release(&tickslock);
return -1;
}
sleep(&ticks, &tickslock);
}
release(&tickslock);
return 0;
}
// return how many clock tick interrupts have occurred
// since start.
int
sys_uptime(void)
{
uint xticks;
acquire(&tickslock);
xticks = ticks;
release(&tickslock);
return xticks;
}
// return the year of which Unix version 6 was released
int
sys_getyear(void)
{
return 1975;
}
// sum
int sys_sum(void) {
int num1, num2;
argptr(0, (void *)&num1, sizeof(num1));
argptr(1, (void *)&num2, sizeof(num2));
return num1+num2;
}
// difference
int sys_diff(void) {
int num1, num2;
argptr(0, (void *)&num1, sizeof(num1));
argptr(1, (void *)&num2, sizeof(num2));
return num1-num2;
}
// multiply
int sys_mul(void) {
int num1, num2;
argptr(0, (void *)&num1, sizeof(num1));
argptr(1, (void *)&num2, sizeof(num2));
return num1*num2;
}
// divide
int sys_div(void) {
int num1, num2;
argptr(0, (void *)&num1, sizeof(num1));
argptr(1, (void *)&num2, sizeof(num2));
return num1/num2;
}
// remainder
int sys_rem(void) {
int num1, num2;
argptr(0, (void *)&num1, sizeof(num1));
argptr(1, (void *)&num2, sizeof(num2));
return num1%num2;
}
// square
int sys_square(void) {
int num;
argptr(0, (void *)&num, sizeof(num));
return num*num;
}
// cube
int sys_cube(void) {
int num;
argptr(0, (void *)&num, sizeof(num));
return num*num*num;
}
```

• user.h

```
struct stat;
struct rtcdate;

// system calls
int fork(void);
int exit(void) __attribute__((noreturn));
int wait(void);
int pipe(int*);
int write(int, const void*, int);
int read(int, void*, int);
int close(int);
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 getyear(void);
int sum(int, int);
int diff(int, int);
int mul(int, int);
int div(int, int);
int rem(int, int);
int square(int);
int cube(int);

// 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*);
int atoi(const char*);
```

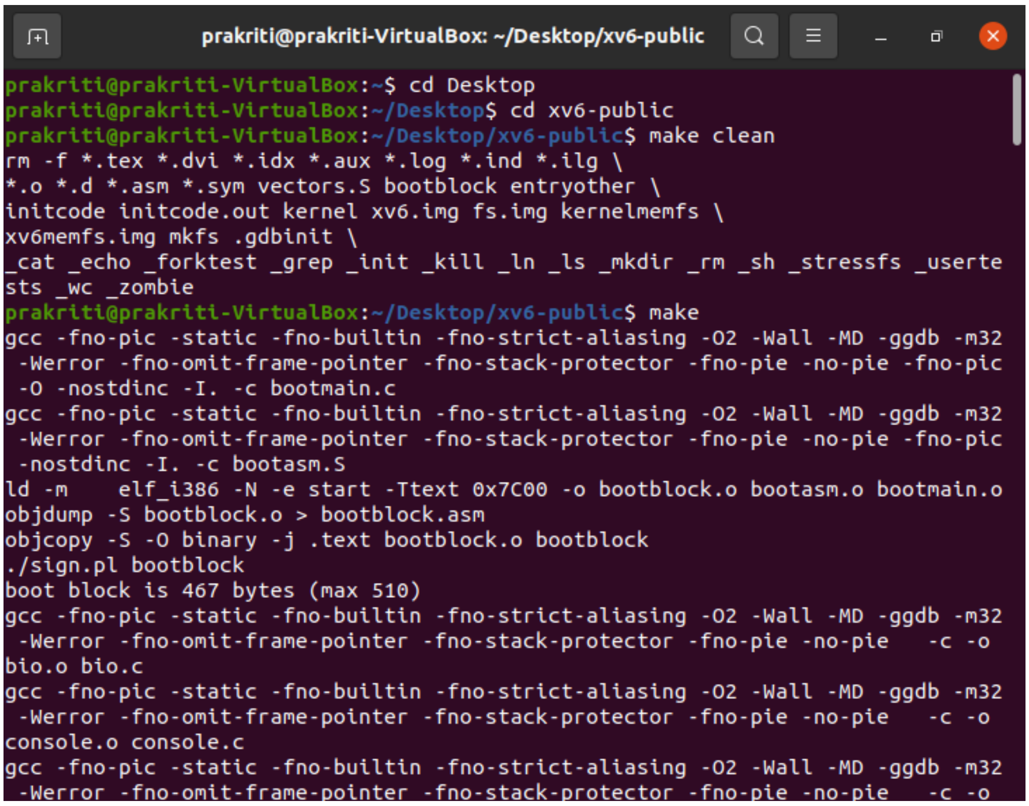
• usys.S

```
#include "syscall.h"
#include "traps.h"
// 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*);
int atoi(const char*);
#define SYSCALL(name) \
.globl name; \
name: \
movl $SYS_ ## name, %eax; \
int $T_SYSCALL; \
ret
SYSCALL(fork)
SYSCALL(exit)
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(getyear)
SYSCALL(dup)
SYSCALL(getpid)
SYSCALL(sbrk)
SYSCALL(sleep)
SYSCALL(uptime)
SYSCALL(sum)
SYSCALL(diff)
SYSCALL(mul)
SYSCALL(div)
SYSCALL(rem)
SYSCALL(square)
SYSCALL(cube)
```

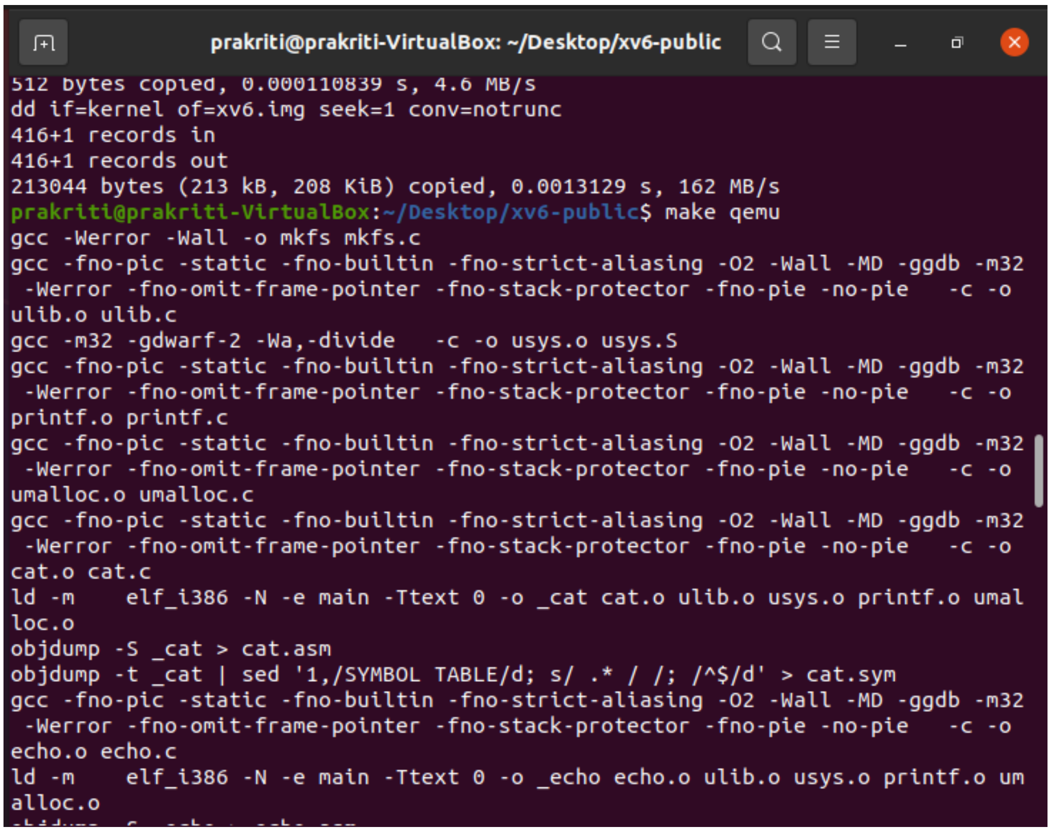
• myprogram.c

```
#include "types.h"
#include "stat.h"
#include "user.h"
int
main(void)
{
printf(0, "Get Year:\n");
printf(1, "Unix V6 was released in the year %d\n", getyear());

printf(0, "Sum:\n");
printf(1, "Sum of 8 and 5 is %d\n", sum(8,5));

printf(0, "Difference:\n");
printf(1, "Difference of 8 and 5 is %d\n", diff(8,5));

printf(0, "Multiply:\n");
printf(1, "Multiplication of 8 and 5 is %d\n", mul(8,5));
printf(0, "Divide:\n");
printf(1, "Divison of 8 and 5 is %d\n", div(8,5));

printf(0, "Remainder:\n");
printf(1, "Remainder of 8 and 5 is %d\n", rem(8,5));

printf(0, "Square:\n");
printf(1, "Square of 8 is %d\n", square(8));

printf(0, "Cube:\n");
printf(1, "Cube of 8 is %d\n", cube(8));
exit();
}
```

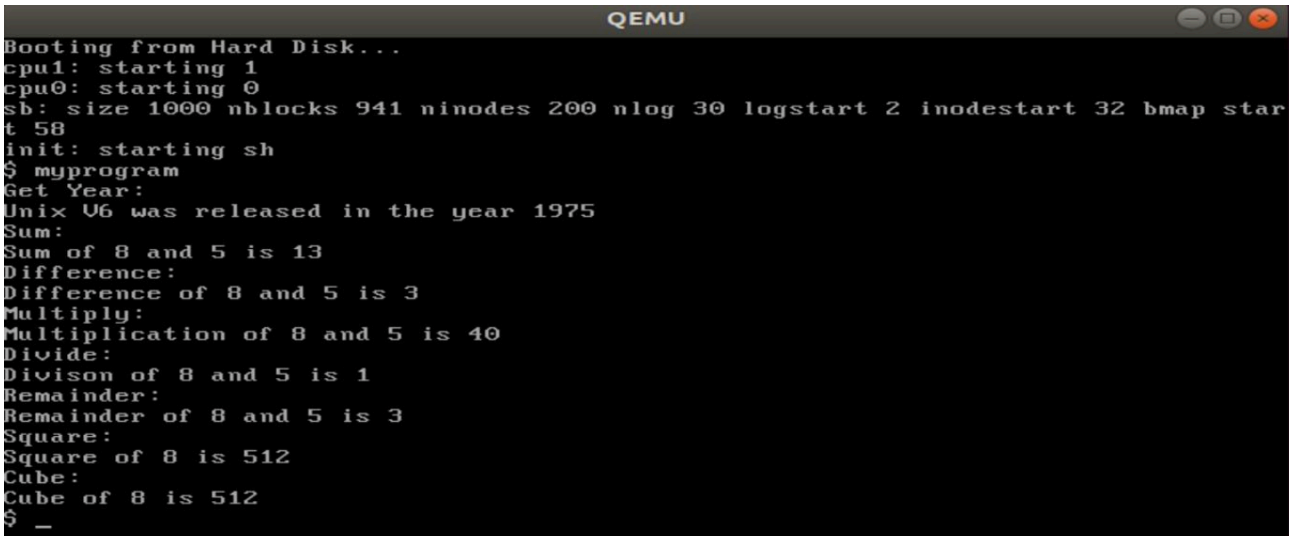
Screenshots



```
prakriti@prakriti-VirtualBox:~$ cd Desktop
prakriti@prakriti-VirtualBox:~/Desktop$ cd xv6-public
prakriti@prakriti-VirtualBox:~/Desktop/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 _userte
sts _wc _zombie
prakriti@prakriti-VirtualBox:~/Desktop/xv6-public$ 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 -fno-pic
 -O -nostdinc -I. -c bootmain.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 -fno-pic
 -nostdinc -I. -c bootasm.S
ld -m    elf_i386 -N -e start -Ttext 0x7C00 -o bootblock.o bootasm.o bootmain.o
objdump -S bootblock.o > bootblock.asm
objcopy -S -O binary -j .text bootblock.o bootblock
./sign.pl bootblock
boot block is 467 bytes (max 510)
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
bio.o bio.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
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
```



```
512 bytes copied, 0.000110839 s, 4.6 MB/s
dd if=kernel of=xv6.img seek=1 conv=notrunc
416+1 records in
416+1 records out
213044 bytes (213 kB, 208 KiB) copied, 0.0013129 s, 162 MB/s
prakriti@prakriti-VirtualBox:~/Desktop/xv6-public$ make qemu
gcc -Werror -Wall -o mkfs mkfs.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
ulib.o ulib.c
gcc -m32 -gdwarf-2 -Wa,-divide   -c -o usys.o usys.S
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
printf.o printf.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
umalloc.o umalloc.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
cat.o cat.c
ld -m    elf_i386 -N -e main -Ttext 0 -o _cat cat.o ulib.o usys.o printf.o umal
loc.o
objdump -S _cat > cat.asm
objdump -t _cat | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > cat.sym
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
echo.o echo.c
ld -m    elf_i386 -N -e main -Ttext 0 -o _echo echo.o ulib.o usys.o printf.o um
alloc.o
```



```
Booting from Hard Disk...
cpu1: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap star
t 58
init: starting sh
$ myprogram
Get Year:
Unix V6 was released in the year 1975
Sum:
Sum of 8 and 5 is 13
Difference:
Difference of 8 and 5 is 3
Multiply:
Multiplication of 8 and 5 is 40
Divide:
Divison of 8 and 5 is 1
Remainder:
Remainder of 8 and 5 is 3
Square:
Square of 8 is 512
Cube:
Cube of 8 is 512
$ _
```

Conclusion

In this way Xv6 works using QEMU emulator which functions as a parallel Linux os and helps to execute the processes in the similar way. Here we tried to implement certain modules and represented as shown.

Future Work

We can further add more system calls to the operating system and note the current process status and also change the features and go through the algorithms.

References

- http://ocw.mit.edu (Accessed 22 Feb, 2016). License: Creative CommonsBYNC-SA
- https://pdos.csail.mit.edu/6.828/2019/xv6/book-riscv-rev0.pdf
- http://people.cs.pitt.edu/~henriquepotter/resources/Lab2.pdf
- https://www.cse.iitb.ac.in/~mythili/os/labs/lab-xv6-proc/xv6-proc.pdf
- http://recolog.blogspot.com/2016/02/installing-and-running-xv6-operating.html
- https://github.com/mit-pdos/xv6-public