IMPLEMENTING USER SPECIFIED ALGORITHM IN XV6 OPERATING SYSTEM

TEAM MEMBERS -

(1) AMRIT KANOI 18BCI0108

(2) ADIT SACHIN BHOSLE 18BCI0124

GUIDED BY OUR FACULTY :

Dr . PADMA PRIYA R.

ABSTRACT

To understand the Operating System and its practical aspects hands on experience with some project is must . A good project would be to implement features to some experimental small Operating System. The Xv6 is a simple Unix like Operating System developed for MIT's OS courses . The great thing about the Xv6 is that we can implement features like scheduling, process creation and lot more. For the reference we have implemented few features (system call cps() , system call clear() , system call shutdown() , sytem call foo(), system call nice()) for our own practical knowledge . Our team has designed a new scheduling algortihm for XV6 operating system . Instead of default ROUND-ROBIN algorithm , our algorithm will allow the user to change the priority of the process according to his requirement . The state of the process which is currently in runnable state can be changed to running state as per the user requirement . The user can assign a default priority to any process . The user can make any process go to sleep too .

MOTIVATION

The motivation behind our project was to enable user to create system call and play with the operating system . xv6 is unix lie operating system , which gives us power to manipulate the assembly level codes along with changing the priority of the process and create and implement our own process scheduling algorithm . we can shut down the xv6 operating system , by just calling a system call . we can list all the process state and process priority by calling a command ps . we can clear the screen using our own implemented clear command . The command foo enables us to create child process . by passing the process id and new priority of the process to the command nice , we can change the state of the process .

INTRODUCTION

xv6 is a modern reimplementation of Sixth Edition Unix in ANSI C for multiprocessor x86 and RISC-V systems. It is used for pedagogical purposes in MIT's Operating Systems Engineering course as well as Georgia Tech's Design of Operating Systems Course as well as many other institutions.

Default user interface : Command-line interface

Kernel type : Monolithic kernel

OS family : Unix-like

Source model : Open-source software

Developed by : Massachusetts Institute of Technology

Written in : C, Assembly language

SYSTEM CALLS IMPLEMENTED BY US -

(A) ls – list all states and priority of the process

(B) clear – clear the command line interface

(C) foo - create child process

(D) nice – change the priority of the process

(E) shutdown – halt the system

MODULES IMPLEMENTED

(1) SYSCALL.H - giving a fixed number to our system call

(2) DEFS.H - including the definition of our system call

(3) USER.H - including the definition for the user

(4) PROC.C - definition of function of our system call

(5) SYSPROC.C - defining a function sys_cps which will invoke the cps system call

(6) USYS.S - adding the assembly level code

(7) CPS.C - c program which will invoke the cps system call

(8) EXEC.C - changing the priority of process

INSTALLING XV6 :

(1) OPEN THE TERMINAL (ctrl+alt+T)

(2) Run sudo apt update

(3) Run the following commands -

- (a) sudo apt-get install gcc
- (b) sudo apt-get install make
- (c) sudo apt-get install qemu-system
- (d) sudo apt-get install git
- (e) git clone git://github.com/mit-pdos/xv6-public.git

amrit@amrit-Inspiron-15-3567: ~

File Edit View Search Terminal Help

```
amrit@amrit-Inspiron-15-3567:~$ sudo apt update
[sudo] password for amrit:
Ign:1 http://dl.google.com/linux/chrome/deb stable InRelease
Hit:2 http://archive.canonical.com/ubuntu cosmic InRelease
Hit:3 http://archive.ubuntu.com/ubuntu cosmic InRelease
Get:4 http://dl.google.com/linux/chrome/deb stable Release [943 B]
Get:5 http://ppa.launchpad.net/linuxuprising/java/ubuntu cosmic InRelease [15.9
kB]
Hit:6 http://archive.ubuntu.com/ubuntu cosmic-updates InRelease
Hit:7 http://security.ubuntu.com/ubuntu cosmic-security InRelease
Get:8 http://dl.google.com/linux/chrome/deb stable Release.gpg [819 B]
Hit:9 http://archive.ubuntu.com/ubuntu cosmic-backports InRelease
Get:10 http://dl.google.com/linux/chrome/deb stable/main amd64 Packages [1,105 B
]
Hit:11 http://ppa.launchpad.net/webupd8team/java/ubuntu cosmic InRelease
Fetched 18.8 kB in 2s (11.6 kB/s)
Reading package lists... Done
Building dependency tree
Reading state information... Done
1 package can be upgraded. Run 'apt list --upgradable' to see it.
amrit@amrit-Inspiron-15-3567:~$ sudo apt intstall qemu
```

Ubuntu18.04 [Running] - Oracle VM VirtualBox

File Machine View Input Devices Help

Activities Terminal Tue 11:00

adit@18BCI0124: ~

File Edit View Search Terminal Help

```
adit@18BCI0124:~$ sudo apt-get install gcc
[sudo] password for adit:
Sorry, try again.
[sudo] password for adit:
Reading package lists... Done
Building dependency tree
Reading state information... Done
gcc is already the newest version (4:7.4.0-1ubuntu2.3).
0 upgraded, 0 newly installed, 0 to remove and 96 not upgraded.
adit@18BCI0124:~$ sudo apt-get install make
Reading package lists... Done
Building dependency tree
Reading state information... Done
make is already the newest version (4.1-9.1ubuntu1).
0 upgraded, 0 newly installed, 0 to remove and 96 not upgraded.
adit@18BCI0124:~$ sudo apt-get install qemu-system
Reading package lists... Done
Building dependency tree
Reading state information... Done
qemu-system is already the newest version (1:2.11+dfsg-1ubuntu7.19).
0 upgraded, 0 newly installed, 0 to remove and 96 not upgraded.
adit@18BCI0124:~$ sudo apt-get install git
Reading package lists... Done
Building dependency tree
Reading state information... Done
git is already the newest version (1:2.17.1-1ubuntu0.4).
0 upgraded, 0 newly installed, 0 to remove and 96 not upgraded.
adit@18BCI0124:~$ git clone git://github.com/mit-pdos
```

Right Ctrl

LAUNCHING XV6 :

(1) change the directory : cd xv6-public

(2) running the command make

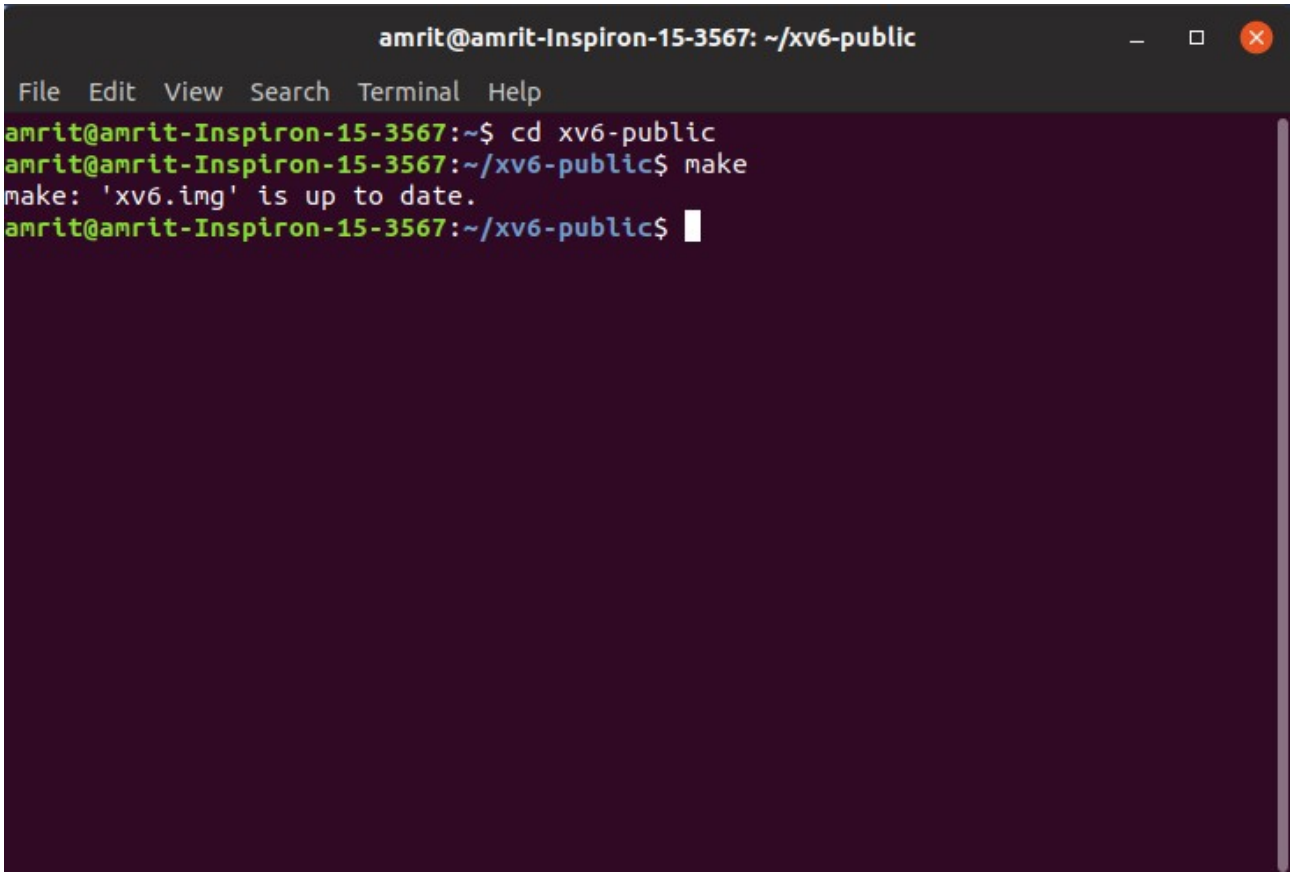
(3) opening XV6 by running command make qemu-nox

(4) XV6 gets launched

(5) To view all the commands – use command ls

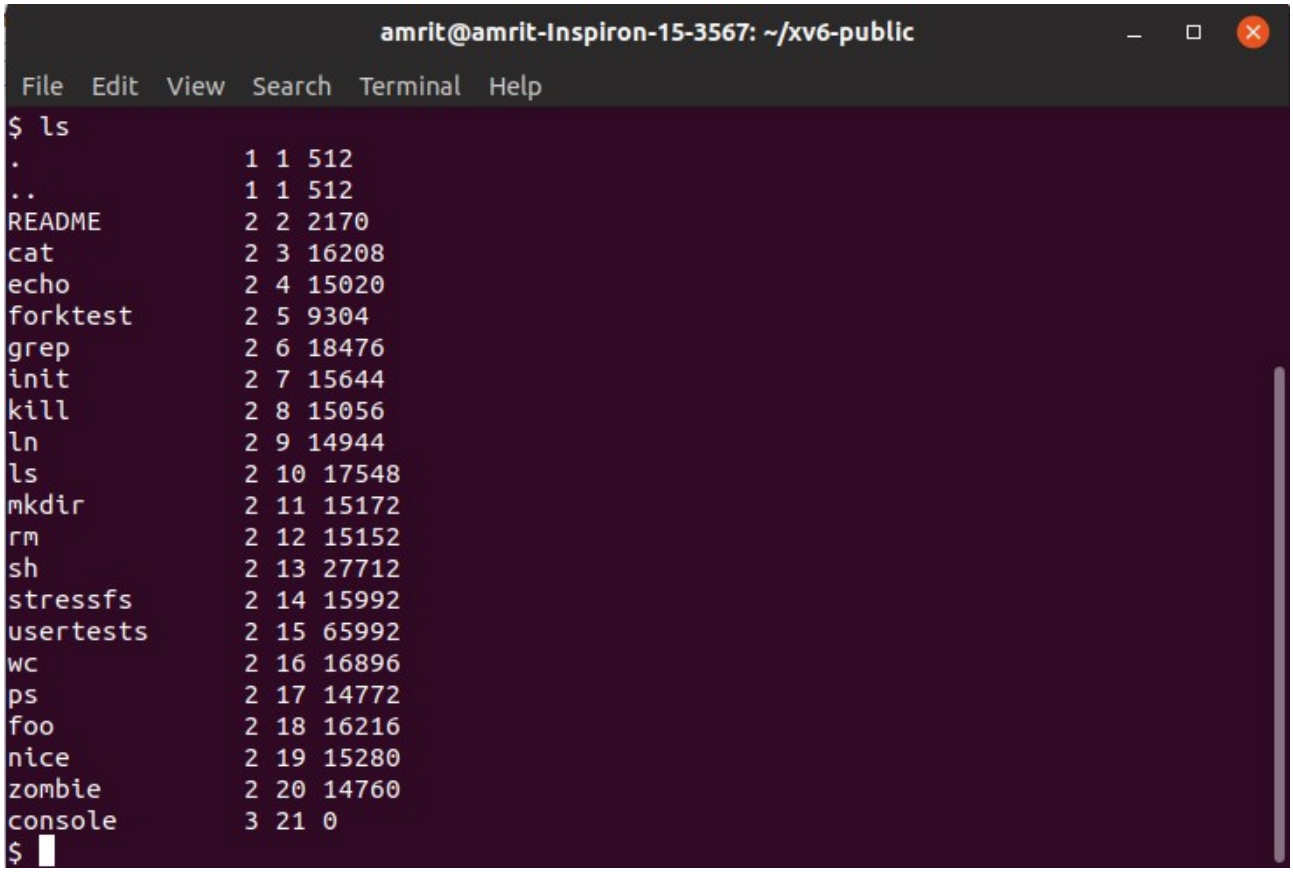
(6) All the predefined and user defined commands gets listed

(7) To terminate XV6 use – ctrl+A+X;



```
amrit@amrit-Inspiron-15-3567: ~/xv6-public
File  Edit  View  Search  Terminal  Help
SeaBIOS (version 1.11.1-1ubuntu1)


iPXE (http://ipxe.org) 00:03.0 C980 PCI2.10 PnP PMM+1FF8D340+1FECD340 C980



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 sta8
init: starting sh
$
$ QEMU: Terminated
amrit@amrit-Inspiron-15-3567:~/xv6-public$
amrit@amrit-Inspiron-15-3567:~/xv6-public$ make qemu-nox
```

```
amrit@amrit-Inspiron-15-3567: ~/xv6-public
File  Edit  View  Search  Terminal  Help
SeaBIOS (version 1.11.1-1ubuntu1)


iPXE (http://ipxe.org) 00:03.0 C980 PCI2.10 PnP PMM+1FF8D340+1FECD340 C980



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 sta8
init: starting sh
$
```



IMPLEMENTATION OF COMMAND - ps

>> ps command defined in xv6 will list all the process along with pid and state . Ps is GET PROCESS INFO command .

(1) Giving our system call ps a number . Run gedit 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_cps    22
#define SYS_halt   23
#define SYS_clear  24
#define SYS_chpr   25
```

(2) Next , we have to include the declaration of our system call ps .

Run gedit defs.h

```
void            pipeclose(struct pipe*, int);
int             piperead(struct pipe*, char*, int);
int             pipewrite(struct pipe*, char*, int);

//PAGEBREAK: 16
// proc.c
int             cpuid(void);
void            exit(void);
int             fork(void);
int             growproc(int);
int             kill(int);
struct cpu*     mycpu(void);
struct proc*    myproc();
void            pinit(void);
void            procdump(void);
void            scheduler(void) __attribute__((noreturn));
void            sched(void);
void            setproc(struct proc*);
void            sleep(void*, struct spinlock*);
void            userinit(void);
int             wait(void);
void            wakeup(void*);
void            yield(void);
int             cps(void);
int             halt(void);
int             chpr(int pid, int priority);

// swtch.S
```

(3) Similarly , adding our system call in user.h

```
int mknod(const char*, short, short);
int unlink(const char*);
int fstat(int fd, struct stat*);
int link(const char*, const char*);
int mkdir(const char*);
int chdir(const char*);
int dup(int);
int getpid(void);
char* sbrk(int);
int sleep(int);
int uptime(void);
int cps(void);
int halt(void);
int chpr(int pid,int priority);

// 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*);
```

(4) Running the proc.c and implementing our algorithm

```
    cprintf("\n");
  }
}

//current process status
int
cps()
{
        struct proc *p;
        //enable interuppt on this processor
        sti();
        //loop over process table looking for process wit pid
        acquire(&ptable.lock);
        cprintf("name \t pid \t state \t \t priority\n");
        for(p = ptable.proc; p<&ptable.proc[NPROC];p++)
        {
                if(p->state == SLEEPING)
                cprintf("%s \t %d \t SLEEPING \t %d\n",p->name,p->pid,p->priority);
                else if(p->state == RUNNING)
                cprintf("%s \t %d \t RUNNING \t %d\n",p->name,p->pid,p->priority);
                else if(p->state == RUNNABLE)
                cprintf("%s \t %d \t RUNNABLE \t %d\n",p->name,p->pid,p->priority);
        }
        release(&ptable.lock);
        return 22;
}

//change priority
```

(5) Now , we need to define a function which will invokw our system call .

Run gedit sysproc.c

```
// since start.
int
sys_uptime(void)
{
  uint xticks;

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

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

(6) add line SYSCALL(cps) . This is Assembly level code and it interacts with the hardware of the system . %eax – register which hold the system call number .

```
    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(dup)
SYSCALL(getpid)
SYSCALL(sbrk)
SYSCALL(sleep)
SYSCALL(uptime)
SYSCALL(cps)
SYSCALL(halt)
SYSCALL(chpr)
```

(7) in syscall.c we have to declare the function cps , which we have defined in sysproc.c

syscall.c
~/xv6-public

```
  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_cps(void);
extern int sys_halt(void);
extern int sys_chpr(void);
```

(8) Creating a new file ps.c to call our sys_cps function

ps.c
~/xv6-public

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

int main(int argc, char *argv[])
{
        cps();
        exit();
}
```

(9) Now , we have to make corresponding changes in Makefile .

Open ▾ Makefile ~/xv6-public Save

```
# http://www.gnu.org/software/make/manual/html_node/Chained-Rules.html
.PRECIOUS: %.o

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

fs.img: mkfs README $(UPROGS)
	./mkfs fs.img README $(UPROGS)

-include *.d
```

Open ▾ Makefile ~/xv6-public Save

```
# 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 ps.c halt.c foo.c
nice.c zombie.c\
	printf.c umalloc.c\
	README dot-bochsrc *.pl toc.* runoff runoff1 runoff.list\
	.gdbinit.tmpl gdbutil\

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

dist-test:
	rm -rf dist
	make dist
	rm -rf dist-test
	mkdir dist-test
```

(10) TESTING OUR OWN DEFINED SYSTEM CALL

```
amrit@amrit-Inspiron-15-3567: ~/xv6-public
File  Edit  View  Search  Terminal  Help
amrit@amrit-Inspiron-15-3567:~/xv6-public$ gedit syscall.h
amrit@amrit-Inspiron-15-3567:~/xv6-public$ gedit defs.h
amrit@amrit-Inspiron-15-3567:~/xv6-public$ gedit users.h
amrit@amrit-Inspiron-15-3567:~/xv6-public$ gedit user.h
amrit@amrit-Inspiron-15-3567:~/xv6-public$ gedit proc.c

(gedit:6589): GtkSourceView-WARNING **: 10:27:56.557: gtk_source_search_context.
amrit@amrit-Inspiron-15-3567:~/xv6-public$ gedit proc.c
amrit@amrit-Inspiron-15-3567:~/xv6-public$ gedit sysproc.c
amrit@amrit-Inspiron-15-3567:~/xv6-public$ gedit usys.S
amrit@amrit-Inspiron-15-3567:~/xv6-public$  gedit sysproc.c
amrit@amrit-Inspiron-15-3567:~/xv6-public$ gedit syscall.c
amrit@amrit-Inspiron-15-3567:~/xv6-public$ gedit ps.c
amrit@amrit-Inspiron-15-3567:~/xv6-public$ gedit Makefile
amrit@amrit-Inspiron-15-3567:~/xv6-public$
```

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

IMPLEMENTING NICE SYSTEM CALL AND CHANGING THE PRIORITY OF THE PROCESS

(1) Creating a dummy c program foo.c which takes the number of child process to be created through command line arguments as specified by the users .

Open foo.c ~/xv6-public Save

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

int
main(int argc, char *argv[])
{
        long long int k,n,id;
        double x=0,z,d;
        if(argc < 2)
                n=1; //default vlaue
        else
            n=atoi(argv[1]); //from command line
        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) // parent process
                {
                        printf(1,"Parent %d creating child %d\n",getpid(),id);
                        wait();
                }
                else
                {
                        printf(1,"Child %d created\n",getpid());
                        for(z=0;z<800000000000.0;z += d)
                        {
                                x = x + 3.14*89.64;
                                for(int d=1;d<10000000000;d++)
                                {
                                        for(int b=1;b<100000000000;b++){}
                                }
                        }
                        break;
                }
        }
        exit();
}
```

(2) Now , in the PCB we have tot include a new member PRIORITY .

Run Gedit proc.h .

```
  uint eip;
};

enum procstate { UNUSED, EMBRYO, SLEEPING, RUNNABLE, RUNNING, ZOMBIE };

// 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 priority;          // Process Priority (0-20);lower value,higher priority
};
```

(3) Changing the ALLOCPROC function , and setting default priority of a process as 10 .

```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.
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->priority = 10; //default priority
  release(&ptable.lock);
```

(4) Child process should have the greater priority than parent procss . So , gedit exec.c , curr_proc->priority = 3 .

```c
  if(copyout(pgdir, sp, ustack, (3+argc+1)*4) < 0)
    goto bad;

  // Save program name for debugging.
  for(last=s=path; *s; s++)
    if(*s == '/')
      last = s+1;
  safestrcpy(curproc->name, last, sizeof(curproc->name));

  // Commit to the user image.
  oldpgdir = curproc->pgdir;
  curproc->pgdir = pgdir;
  curproc->sz = sz;
  curproc->tf->eip = elf.entry;  // main
  curproc->tf->esp = sp;
  curproc->priority = 3;
  switchuvm(curproc);
  freevm(oldpgdir);
  return 0;

 bad:
  if(pgdir)
    freevm(pgdir);
  if(ip){
    iunlockput(ip);
    end_op();
  }
  return -1;
```

(5) Now , we have to follow same steps that we used earlier . So , to implement our nice system call , gedit syscall.h

syscall.h
~/xv6-public

```
// 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_cps    22
#define SYS_halt   23
#define SYS_clear  24
#define SYS_chpr   25
```

(6) similarly , declare the function in user.h and defs.h

user.h
~/xv6-public

```
int mknod(const char*, short, short);
int unlink(const char*);
int fstat(int fd, struct stat*);
int link(const char*, const char*);
int mkdir(const char*);
int chdir(const char*);
int dup(int);
int getpid(void);
char* sbrk(int);
int sleep(int);
int uptime(void);
int cps(void);
int halt(void);
int chpr(int pid,int priority);

// 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*);
```

```
Open ▾                defs.h                Save
                      ~/xv6-public
void            scheduler(void) __attribute__((noreturn));
void            sched(void);
void            setproc(struct proc*);
void            sleep(void*, struct spinlock*);
void            userinit(void);
int             wait(void);
void            wakeup(void*);
void            yield(void);
int             cps(void);
int             halt(void);
int             chpr(int pid, int priority);

// swtch.S
void            swtch(struct context**, struct context*);

// spinlock.c
void            acquire(struct spinlock*);
void            getcallerpcs(void*, uint*);
int             holding(struct spinlock*);
void            initlock(struct spinlock*, char*);
void            release(struct spinlock*);
void            pushcli(void);
void            popcli(void);

// sleeplock.c
void            acquiresleep(struct sleeplock*);
void            releasesleep(struct sleeplock*);
int             holdingsleep(struct sleeplock*);
```

(7) Now , lets include the definition of our system call . Gedit proc.c

```
//change priority
int
chpr(int pid,int priority)
{
    struct proc *p;
    acquire(&ptable.lock);
    for(p=ptable.proc ; p<&ptable.proc[NPROC];p++)
    {
                if(p->pid == pid)
                {
                        p->priority = priority;
                        break;
                }
   }
        release(&ptable.lock);
        return pid;
}
```

(8) gedit sysproc.c – calling the system call chpr in function sys_chpr

```c
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);
}
```

(9) gedit USYS.S – assembly level code

```
    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(dup)
SYSCALL(getpid)
SYSCALL(sbrk)
SYSCALL(sleep)
SYSCALL(uptime)
SYSCALL(cps)
SYSCALL(halt)
SYSCALL(chpr)
```

(10) Now , providing the declaration of sys_chpr which contains our system call . Gedit syscall.c

```
extern int sys_sbrk(void);
extern int sys_sleep(void);
extern int sys_unlink(void);
extern int sys_wait(void);
extern int sys_write(void);
extern int sys_uptime(void);
extern int sys_cps(void);
extern int sys_halt(void);
extern int sys_chpr(void);
```

(11) gedit nice.c – which will call chpr

nice.c
~/xv6-public

```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();
}
```

(12) MAKING appropriatechange in uprogs .

Open Makefile ~/xv6-public Save

```
# Prevent deletion of intermediate files, e.g. cat.o, after first build, so
# that disk image changes after first build are persistent until clean.  More
# details:
# http://www.gnu.org/software/make/manual/html_node/Chained-Rules.html
.PRECIOUS: %.o

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

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 ps.c halt.c foo.c
nice.c zombie.c\
	printf.c umalloc.c\
	README dot-bochsrc *.pl toc.* runoff runoff1 runoff.list\
	.gdbinit.tmpl gdbutil\
```

(13) Now , changing the scheduling algorithm from round robin to priority based .

Open proc.c ~/xv6-public Save

```
//PAGEBREAK: 42
// Per-CPU process scheduler.
// Each CPU calls scheduler() after setting itself up.
// Scheduler never returns.  It loops, doing:
//  - choose a process to run
//  - swtch to start running that process
//  - eventually that process transfers control
//      via swtch back to the scheduler.
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;
```

Open ▾ proc.c ~/xv6-public Save

```
      highP = p;
          //choose one with highest priority
                for(p1 = ptable.proc ;p1 < &ptable.proc[NPROC] ; p1++)
                {
                        if(p1->state != RUNNABLE)
                                continue;
                        if(highP ->priority > p1->priority)//larger value , lower
priority
                                highP = p1;
                }
                p = highP;
                //no need of this proc = p;
      // 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;
      switchuvm(p);
      p->state = RUNNING;

      swtch(&(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);
```

(14) ALL commands ran on terminal

```
amrit@amrit-Inspiron-15-3567: ~/xv6-public
File  Edit  View  Search  Terminal  Help
amrit@amrit-Inspiron-15-3567:~/xv6-public$ gedit foo.c
amrit@amrit-Inspiron-15-3567:~/xv6-public$ gedit foo.c
amrit@amrit-Inspiron-15-3567:~/xv6-public$ gedit proc.h
amrit@amrit-Inspiron-15-3567:~/xv6-public$ gedit proc.c
amrit@amrit-Inspiron-15-3567:~/xv6-public$ gedit exec.c
amrit@amrit-Inspiron-15-3567:~/xv6-public$ gedit syscall.h
amrit@amrit-Inspiron-15-3567:~/xv6-public$ gedit user.h
amrit@amrit-Inspiron-15-3567:~/xv6-public$ gedit defs.h
amrit@amrit-Inspiron-15-3567:~/xv6-public$ gedit proc.c
amrit@amrit-Inspiron-15-3567:~/xv6-public$ gedit sysproc.c
amrit@amrit-Inspiron-15-3567:~/xv6-public$ gedit usys.S
amrit@amrit-Inspiron-15-3567:~/xv6-public$ gedit syscall.c
amrit@amrit-Inspiron-15-3567:~/xv6-public$ gedit nice.c
amrit@amrit-Inspiron-15-3567:~/xv6-public$ gedit Makefile

(gedit:9378): GtkSourceView-WARNING **: 11:42:23.877: gtk_source_search_context.

(gedit:9378): GtkSourceView-WARNING **: 11:42:28.710: gtk_source_search_context.
amrit@amrit-Inspiron-15-3567:~/xv6-public$ gedit proc.c
amrit@amrit-Inspiron-15-3567:~/xv6-public$ gedit proc.c
amrit@amrit-Inspiron-15-3567:~/xv6-public$ gedit proc.c
amrit@amrit-Inspiron-15-3567:~/xv6-public$
```

TESTING OF OUR SYSTEM CALL :

(1) creating child process foo 2 0.01 &; foo 2 0.01 &;

```
$ foo 2 0.01 &; foo 2 0.01 &;
$ Parent 8 creating child 9
Child 9 created
Parent 6 creating child 10
Child 10 created
ps
name     pid     state          priority
init     1       SLEEPING       3
sh       2       SLEEPING       3
foo      9       RUNNABLE       10
foo      10      RUNNING        10
foo      6       SLEEPING       3
foo      8       SLEEPING       3
ps       11      RUNNING        3
```

(2) Nice 9 5 , will change the priority of process with PID 9 to 5 . it changes from runnable to running .

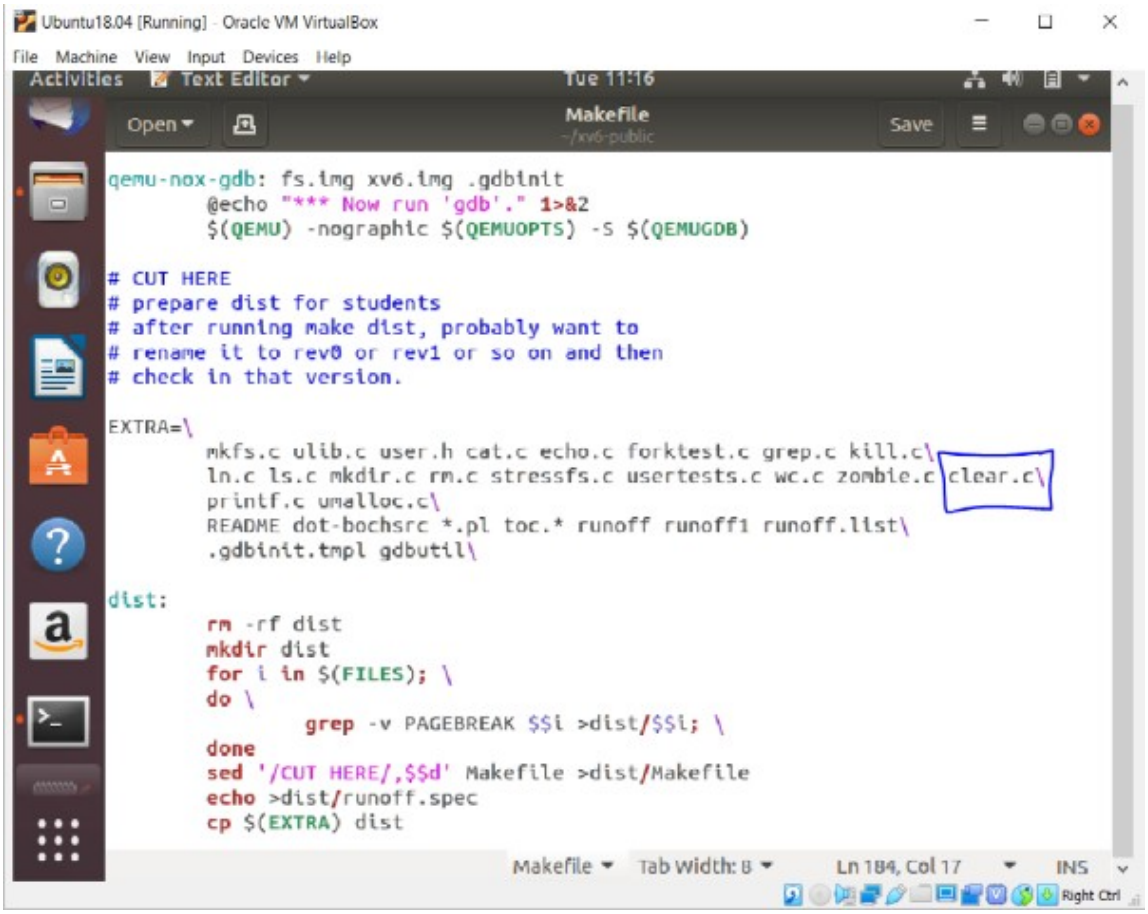
```
amrit@amrit-Inspiron-15-3567: ~/xv6-public
File  Edit  View  Search  Terminal  Help
Child 9 created
Parent 6 creating child 10
Child 10 created
ps
name     pid     state           priority
init     1       SLEEPING        3
sh       2       SLEEPING        3
foo      9       RUNNABLE        10
foo      10      RUNNING         10
foo      6       SLEEPING        3
foo      8       SLEEPING        3
ps       11      RUNNING         3
$ nice 9 5
 pid=9, pr=5
$ ps
name     pid     state           priority
init     1       SLEEPING        3
sh       2       SLEEPING        3
foo      9       RUNNING         5
foo      10      RUNNABLE        10
foo      6       SLEEPING        3
foo      8       SLEEPING        3
ps       13      RUNNING         3
$
```

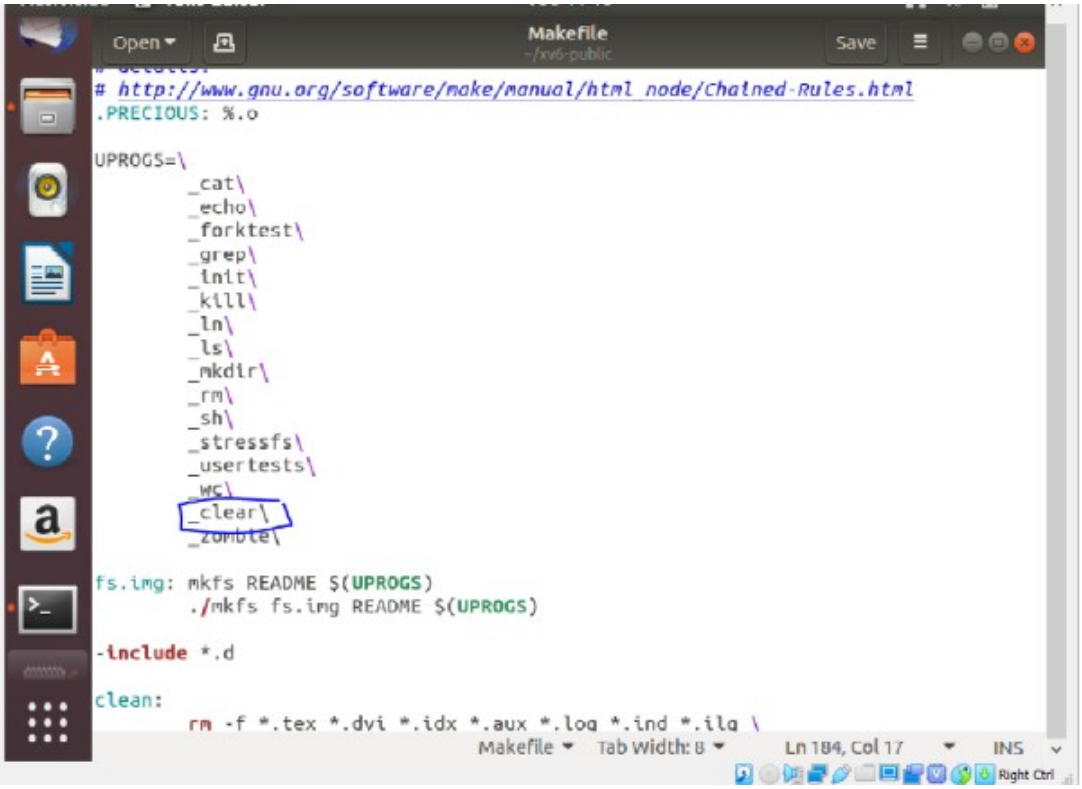
IMPLEMENTING CLEAR SYSTEMCALL :

(1) gedit clear.c

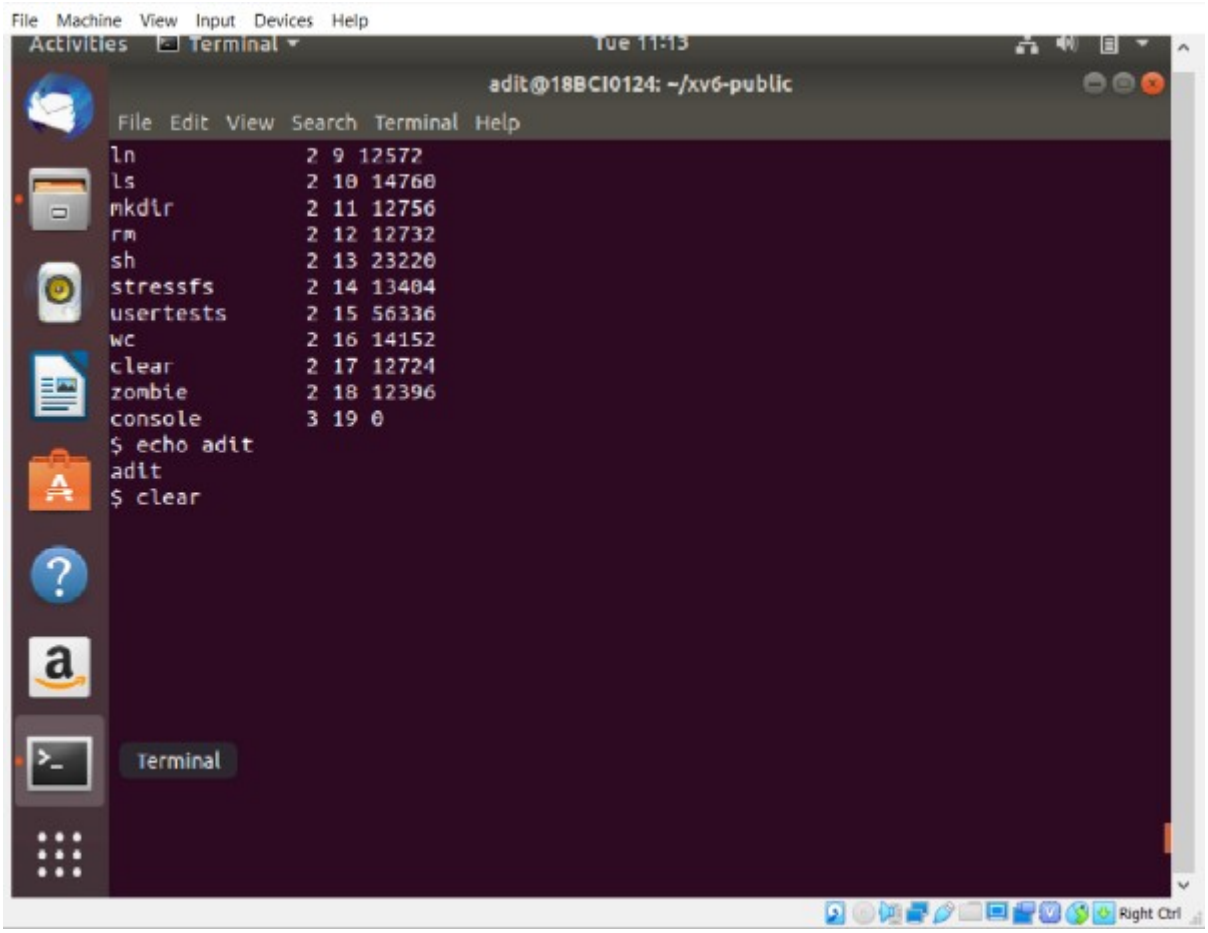
```
Ubuntu18.04 [Running] - Oracle VM VirtualBox
File  Machine  View  Input  Devices  Help
Activities  Text Editor  Tue 11:19
Open  clear.c  ~/xv6-public  Save

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

void clear(int x)
{
        if(x=='x')
                return;
        printf(1,"\xa");
        clear(x+('1'-48));
}
int main(void)
{
        clear('A');
        exit();
}

Amazon

C  Tab Width: 8  Ln 15, Col 2  INS
```

(2) Changes in Makefile



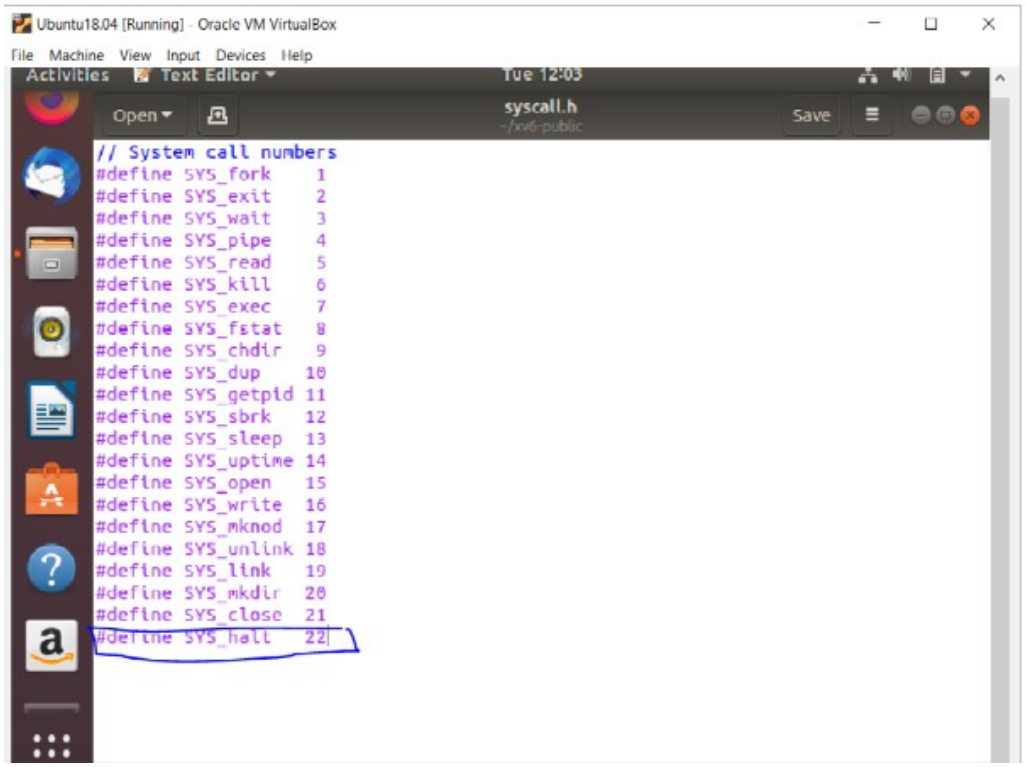


(3) OUTPUT :

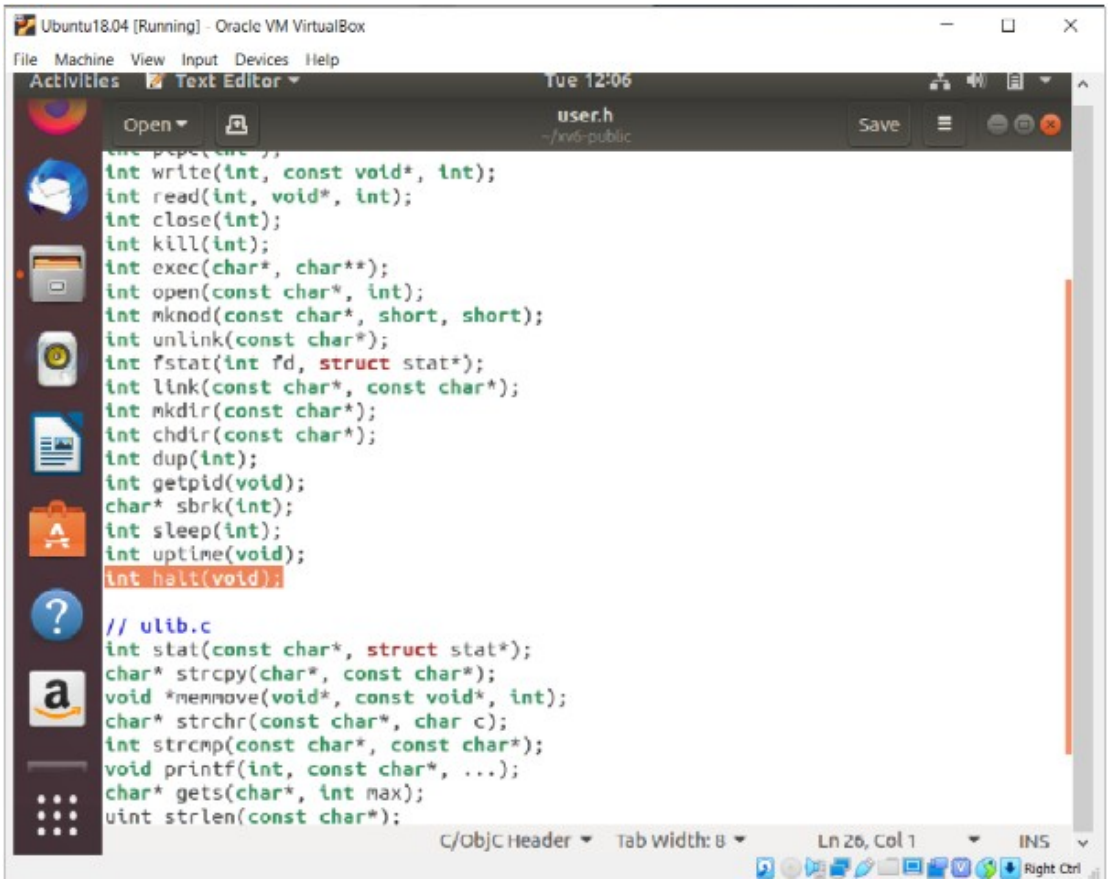


IMPLEMENTING HALT/SHUT DOWN FUNCTION CALL :

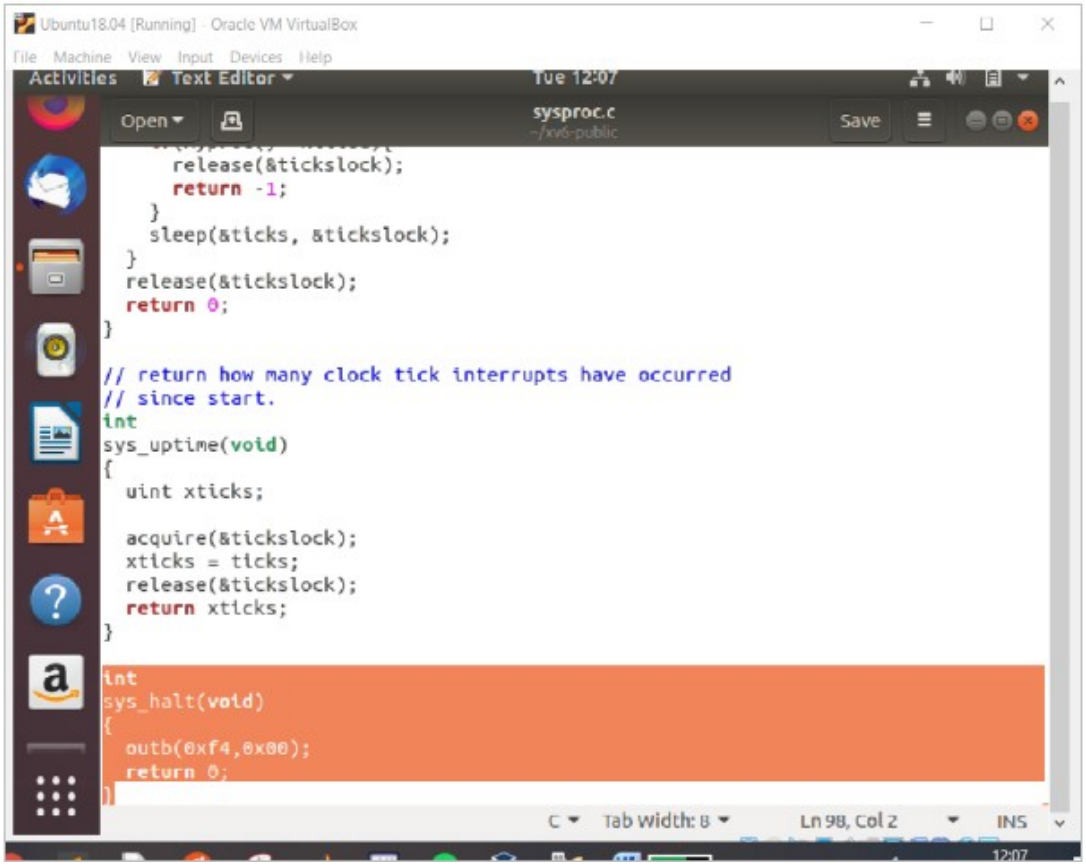
(1) gedit syscall.h



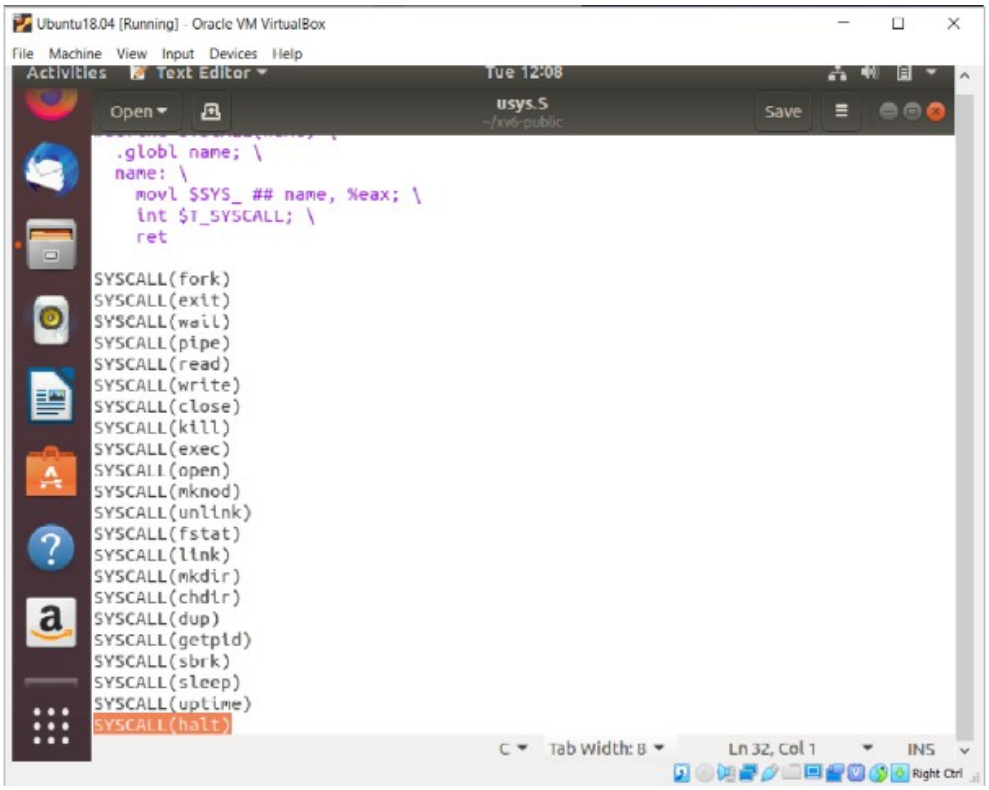
(2) declaring the function in user.h



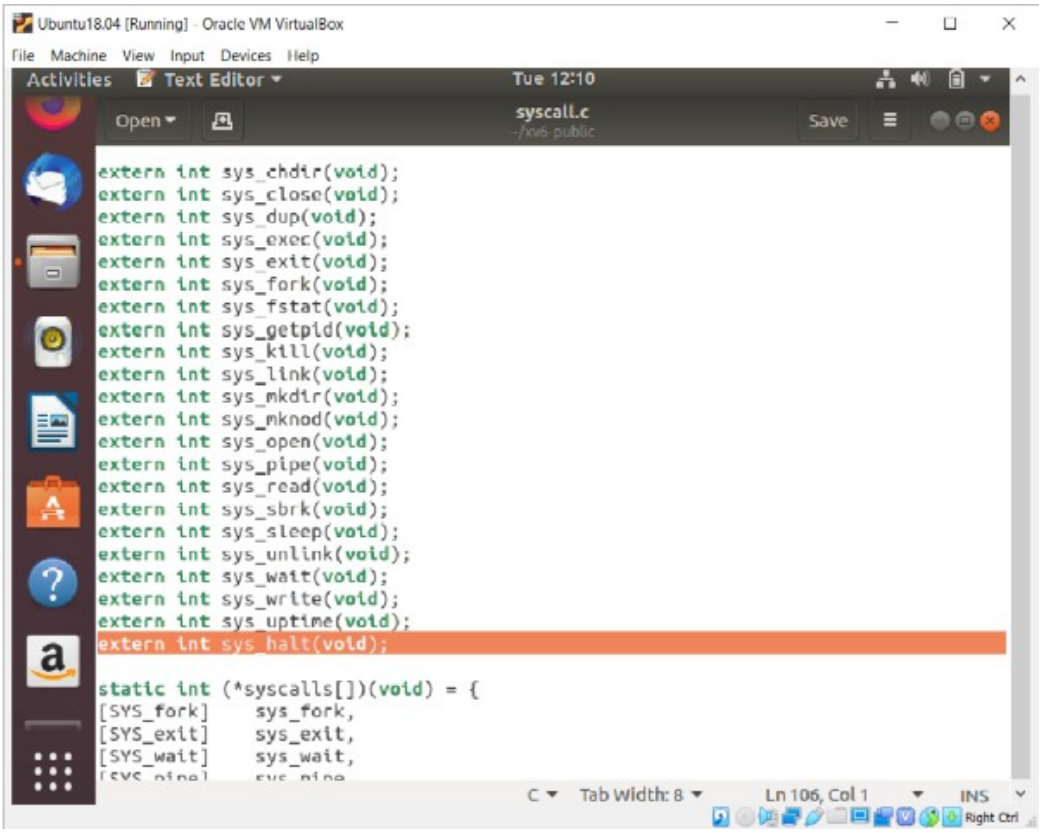
(3) defining the function to invoke our halt system call in sysproc.c



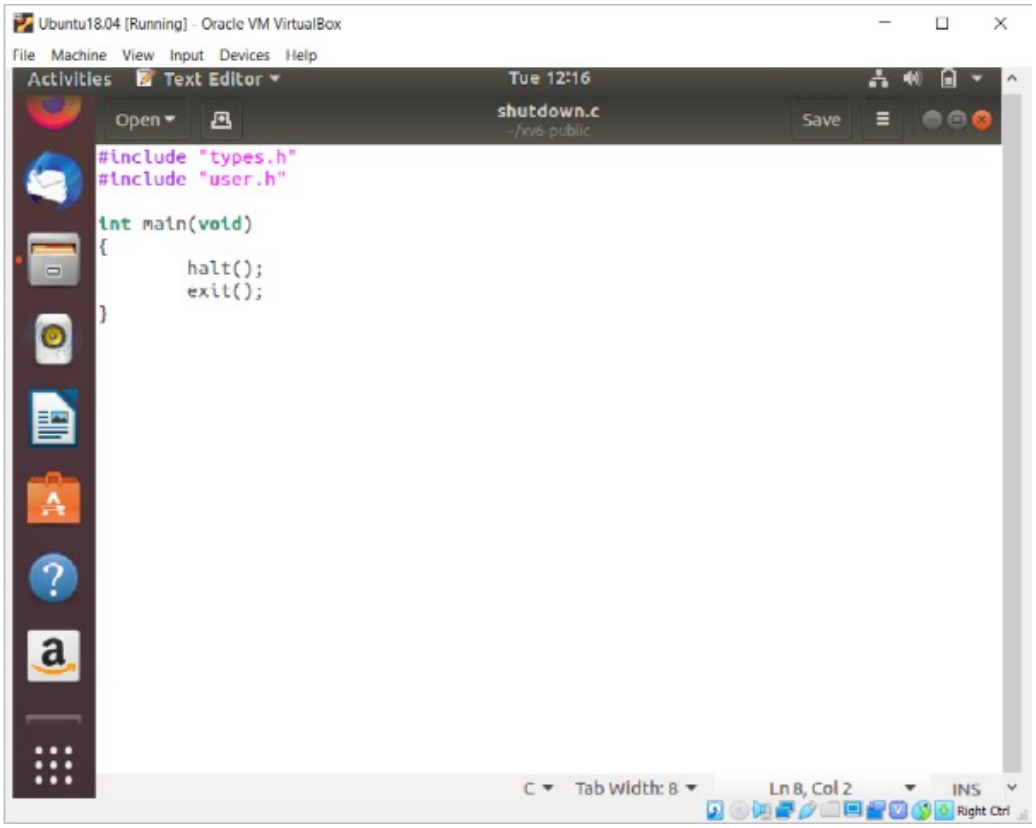
(4) Assembly level declaration in gedit usys.S



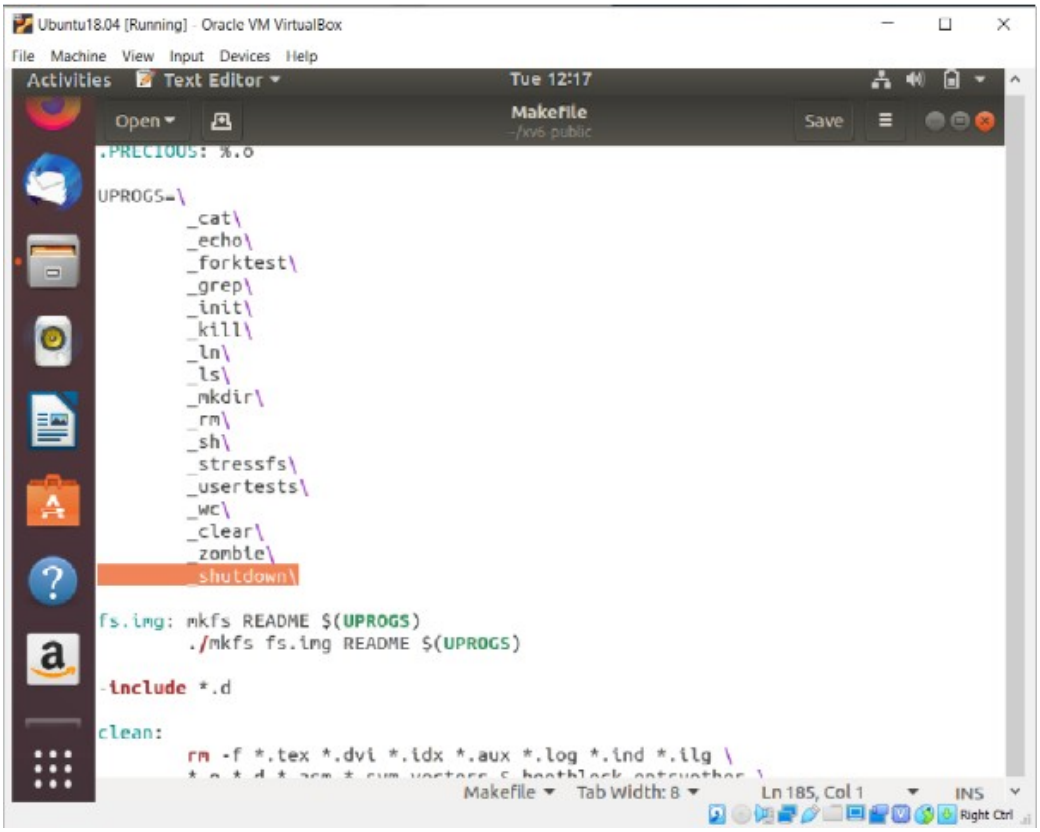
(5) syscall.c DECLARATION OF SYS_HALT

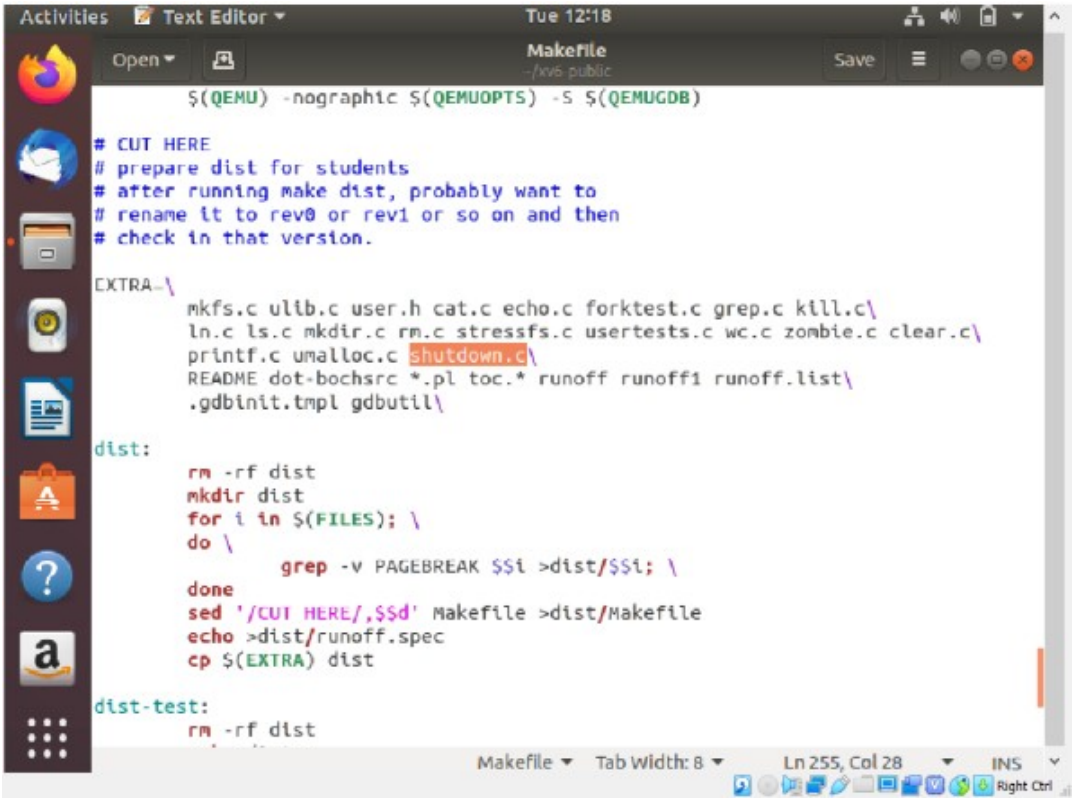


(6) implementing gedit shutdown.c



(7) makefile changes in UPROGS -





WORKING -

```
File  Edit  View  Search  Terminal  Help
ri 58
init: starting sh
$ Thunderbird Mail
.              1 1 512
..             1 1 512
README         2 2 2170
cat            2 3 13636
echo           2 4 12644
forktest       2 5 8080
grep           2 6 15512
init           2 7 13232
kill           2 8 12696
ln             2 9 12596
ls             2 10 14780
mkdir          2 11 12776
rm             2 12 12756
sh             2 13 23240
stressfs       2 14 13424
usertests      2 15 56356
wc             2 16 14176
clear          2 17 12752
zombie         2 18 12420
shutdown       2 19 12376
console        3 20 0
$ shutdown
adit@18BCI0124:~/xv6-public$
```

INFERENCE

We can infer from the project that its very easy to play and manipulate the with the system calls and implement our own scheduling algorithms in XV6 . Implementing our own system calls and changing the assemlby level codes gives us a insight on how the operating system is made to behave and how it interacts with the hardware . Declaring the functions in defs.h and user.h , defining the function in proc.h , changing the attributes of the PCB in proc.c , adding register codes in usys.S and doing all the necessary changes in Makefile really taught us how the operating system functions . But , me and my team will continue to learn and enhance our skills and build more such system calls and improve the xv-6 system . We are really glad and would like to thank our FACULTY MRS. PADMA PRIYA MAM who provided us this opportuinty to explore the technology and encouraged us all throughout the project .

REFERENCES

(1) https://pdos.csail.mit.edu/6.828/2012/xv6.html

(2) https://gateoverflow.in/blog/5506/xv6-operating-system