# ADDING SYSTEM CALLS IN XV6 OPERATING SYSTEM

Project Report

Subject: (**IT-204) OPERATING SYSTEM DESIGN**

BACHELOR OF TECHNOLOGY IN

INFORMATION TECHNOLOGY

4th Semester

Submitted by: VARUN KUMAR (2K19/IT/140)

YASHIT KUMAR (2K19/IT/149)

Under the Supervision of

Mr. Jasraj Meena

DEPARTMENT OF INFORMATION TECHNOLOGY DELHI TECHNOLOGICAL UNIVERSITY (Formerly Delhi College of Engineering) Bawana Road, Delhi-110042

# DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)

Bawana Road, Delhi-110042

## CANDIDATE'S DECLARATION

We Varun Kumar, Roll No – 2K19/IT/140 & Yashit Kumar, Roll No - 2K19/IT/149, students of B.Tech. (INFORMATION TECHNOLOGY), hereby declare that the project Dissertation titled "**Adding a system call in XV6 OS**" which is submitted by us to the Department of INFORMATION TECHNOLOGY, Delhi Technological University, Delhi in partial fulfillment of the requirement for the award of the 4$^{th}$ semester of the Bachelor of Technology, is made by us. This work has not previously formed the basis for the award of any Degree, Diploma Associateship, Fellowship or other similar title or recognition.

Place: Delhi

Date: 26-05-2021

Varun Kumar

Yashit Kumar

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)

Bawana Road, Delhi-110042

# CERTIFICATE

I hereby certify that the Project: " **Adding a system call in XV6 OS** " which is submitted by Varun Kumar, Roll No – 2K19/IT/140 & Yashit Kumar, Roll No – 2K19/IT/149, INFORMATION TECHNOLOGY, Delhi Technological University, Delhi in fulfillment of the requirement for the $4^{th}$ semester of Bachelor of Technology, is record of the project work carried out by the students under my supervision. To the best of my knowledge this work has not been submitted in part or full for any Degree or Diploma to this University or elsewhere.

Place: Delhi **Supervisor** : Mr. Jasraj Meena

Date: 26-May-2021 (Assistant Professor)

**DEPARTMENT OF INFORMATION TECHNOLOGY**

**DELHI TECHNOLOGICAL UNIVERSITY**

(Formerly Delhi College of Engineering)

Bawana Road, Delhi-110042

# ACKNOWLEDGEMENT

We would like to convey our heartfelt thanks to our supervisor Mr. Jasraj Meena for his ingenious ideas, tremendous help and cooperation.

We are extremely grateful to our friends who gave valuable suggestions and guidance for completion of our project. The cooperation and healthy criticism came handy and useful with them.

Finally we would like to thank all the above mentioned people once again.

We would also like to thank our college, Delhi Technological University for giving us an opportunity to study an important subject like "**Operating System**" which would help in our holistic development as an engineer.

**DEPARTMENT OF INFORMATION TECHNOLOGY**

**DELHI TECHNOLOGICAL UNIVERSITY**

(Formerly Delhi College of Engineering)

Bawana Road, Delhi-110042

# ABSTRACT

The idea of the project is to add system calls to the Linux Kernel xv6 developed by MIT UNIVERSITY.We have used the concepts of operating systems and have included the explanation of process and commands required in kernel compilation, and briefly explained the process of system call addition, commands and packages used to do so.We have tried our best to make the code as efficient as possible.

The objectives of this project were to extend the concepts of Operating System to real world activities.The objectives were fulfilled by learning the concepts of system call,priority scheduler,round robin scheduler and many others throughout the course of four months, simultaneously studying research papers about them and applying them by writing code which improved our skills of fixing bugs, problems and errors and helped in improving our concepts while gaining experience of working in a team

# TABLE OF CONTENTS

# OBJECTIVES

➢To create a system call that can be used to count the total number of method calls in the Linux Kernel xv6 developed by MIT University.

➢To create a system call that can be used to change the priority of the system call.

➢To create a system call that is used for dummy calculations by creating a parent-child processes.

➢To create a system call that shows the state and priority of process.

# KEYWORDS

➢ **System Call**

➢ **Linux Kernel xv6**

➢ **QEMU**

# INTRODUCTION

A System call is a mechanism that provides the interface between a process and the operating system.

A computer program makes a system call when it makes a request to the operating system's kernel.

System call **provides** the services of the operating system to the user programs via Application Program Interface(API).System calls are the only entry points into the kernel system.

**Services Provided by System Calls :**

1. Process creation and management

2. Main memory management

3. File Access, Directory and File system management

4. Device handling(I/O)

5. Protection

6. Networking, etc.

# SYSTEM REQUIREMENTS

**➢ Operating System=> Linux xv6 developed by MIT University. ➢ Virtual Box**

**➢ Quick Emulator (QEMU)**

**➢ Language Used=> C**

# PREREQUISITES

There are certain prerequisites for adding a system call in Linux kernel XV6:

● Use a sudo user, so as to avoid unnecessary issues.

● Use the most recent updates of Ubuntu.
To check version:
lsb_release –a
To update ububtu:

● The following packages are required for compilation and is a must. Run commands:

● Git is a version control. It is required to clone repositories. To install:
sudo apt-get install git

● Get the latest version of qemu based on computer's hardware.

● Get XV6 OS from official MIT Github link.

# LINUX  XV6 OS

- **xv6 is a modern reimplementation of Sixth Edition Unix in ANSI C for multiprocessor x86 and RISC-V systems**
- **This is a lightweight operating system where the time to compile is very low.**
- **Xv6 is a teaching operating system developed in the summer of 2006 for MIT**

| | |
|---|---|
| **Developer** | **MIT** |
| **Written in** | **C and assembly** |
| **OS family** | **Unix-like** |
| **Source model** | **Open source** |

# METHODOLOGY

## Installation of QEMU

QEMU (short for Quick Emulator) 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.

**Run the following command to install qemu:**

**sudo apt-get install qemu**

## Installation of xv6

Xv6 is a re-implementation of the Unix sixth edition in order to use as a learning tool. xv6 was developed by MIT as a teaching operating system for
their “6.828” course. A vital fact about xv6 is that it contains all the core Unix concepts and has a similar structure to Unix even though it lacks some functionality that you would expect from a modern operating system.
This is a lightweight operating system where the time to compile is very low
and it also allow remote debugging.

## Clone the official repository:

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

**Running xv6:**

**● cdxv6**

**● make**

**● make qemu**

**In order to define our own system call in xv6, we need to make changes to 5 files. Namely, these files are as follows.**

## Steps Followed To Add SYSTEM CALLS In LINUX KERNEL (XV6 OS):

• Created files named ps.c, clear.c, shutdown.c, nice.c, foo.c, cp.c within the folder xv6 containing all its files.

• Added the names and paths of all above mentioned files in Makefile.

• Added a new system call to the system call table in file syscall.h, usys.S, syscall.c.

• Added the relevant function call in sysproc.c.

• Added a piece of code that will determine what a system call will perform after it is called
to proc.c.

• Finally we can run our system calls and see the outcome on the terminal screen.

```
1. syscall.h
2. syscall.c
3. sysproc.c
4. usys.S
5. user.h
```

## syscall.h

```c
// 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_total  22
#define SYS_states    23
#define SYS_changeprS 24
```

## syscall.c

```
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_total(void);
extern int total_calls;
extern int sys_states(void);
extern int sys_changepr(void);
```

# Usys.S

```
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(total)
SYSCALL(states)
SYSCALL(changepr)
```

# User.h

```c
// 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 total(void);
int states(void);
int changepr(int pid,int priority);
```

# Default list of system calls in xv6

```
cpu1: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 n
init: starting sh
$ ls
.              1 1 512
..             1 1 512
README         2 2 2170
cat            2 3 13436
echo           2 4 12508
forktest       2 5 8232
grep           2 6 15260
init           2 7 13100
kill           2 8 12552
ln             2 9 12460
ls             2 10 14676
mkdir          2 11 12572
rm             2 12 12556
sh             2 13 23196
stressfs       2 14 13228
usertests      2 15 56112
wc             2 16 14088
number         2 17 12744
zombie         2 18 12284
hello          2 19 12484
console        3 20 0
$
```

# System calls that are added: -

**1.total_sys** - System Call which return the total number of method Calls in xv6 .

Sysproc.c

```c
int sys_total(void)
{

if(total_calls==-1) return total_calls;
else return total_calls + 1;
}
```

Syscall.c

```c
void
syscall(void)
{
total_calls++;
  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);
```

Total_sys.c

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

int main(void){

printf(1,"Total Number of System Calls till now are : %d\n",total());
exit();
}
```

**2.pstat** - Shows the current status of process (Running , Sleeping , Runnable)

Pstat.c

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

int main(void){

states();
exit();
}
```

Proc.c

```
int states()
{

struct proc *p;

sti();



acquire(&ptable.lock);
cprintf("name \t pid \t state \t \tpriority\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);
if(p->state==RUNNING)
cprintf("%s \t %d \t RUNNING \t %d \n" ,p->name,p->pid,p->priority);
if(p->state==RUNNABLE)
cprintf("%s \t %d \t RUNNABLE \t %d \n" ,p->name,p->pid,p->priority);
}
release(&ptable.lock);

return 23;
}
```

**3.dmy** - used for dummy calculations by creating a parent-child processes.

**Dmy.c**

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

int main(int argc, char *argv[]) {
  int pid;
  int k, n;
  int x, z;

        n = atoi(argv[1]);

  x = 0;
  pid = 0;

  for ( k = 0; k < n; k++ ) {
    pid = fork ();
    if ( pid < 0 ) {
      printf(1, "%d failed in fork!\n", getpid());
    } else if (pid > 0) {
      // parent
      printf(1, "Parent %d creating child %d\n",getpid(), pid);
      wait();
      }
      else{
        printf(1,"Child %d created\n",getpid());
        for(z = 0; z < 4000000000; z+=1)
            x = x + 3.14*89.64; //Useless calculation to consume CPU Time
        break;
      }
```

**4.chgp** - can be used to change the priority of the system call.

Proc.c

```c
int changepr(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;
}
```

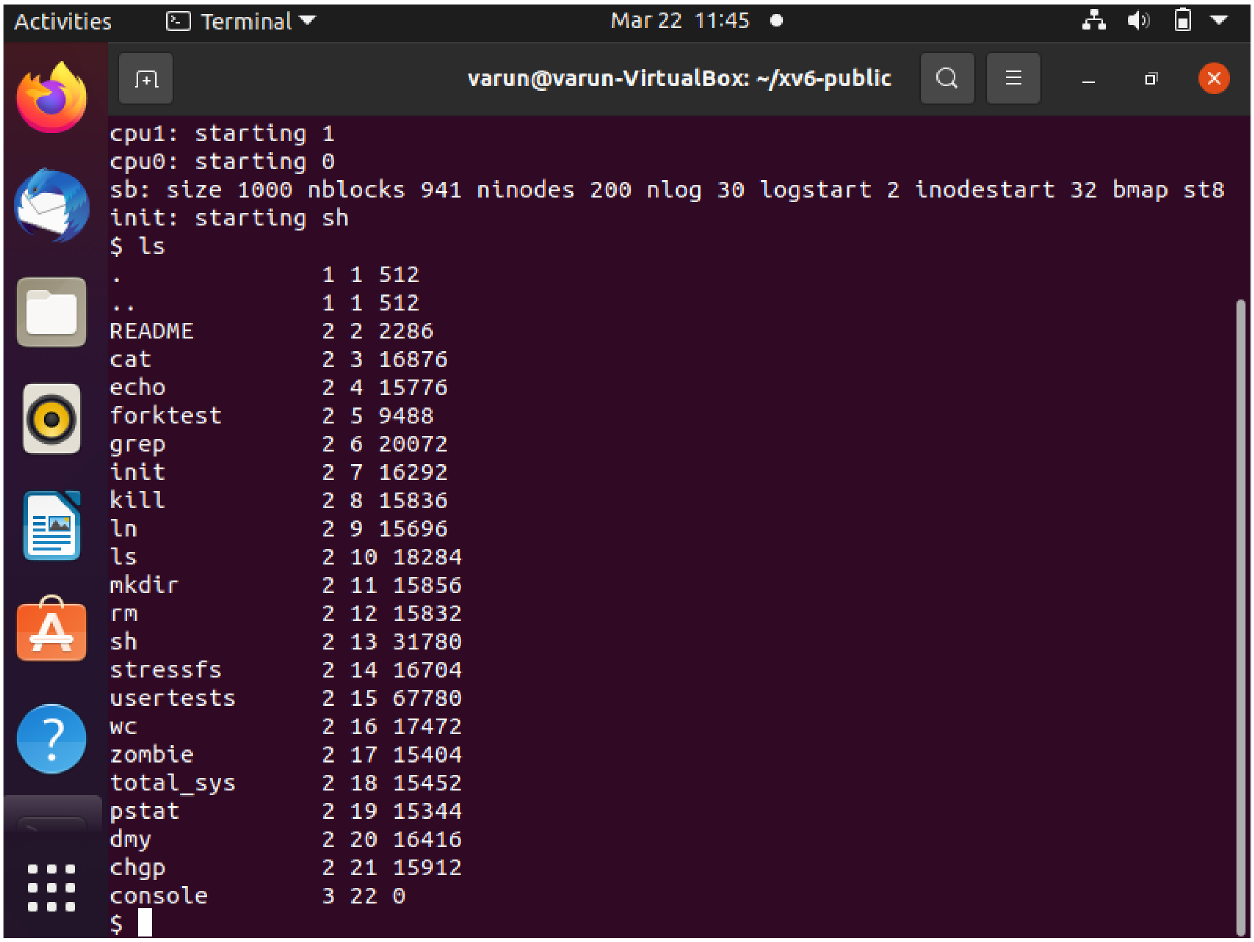
# Sysproc.c

```
int sys_changepr(void){

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

if(argint(1,&pr)<0)
return -1;

return changepr(pid,pr);
```

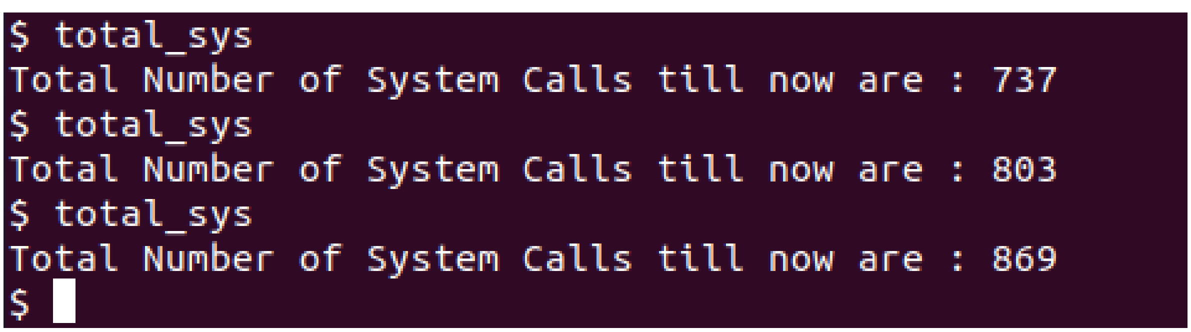
# Chgp.c

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

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

  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);
  changepr(pid, priority);
  exit();
}
```

# OUTPUTS




## Total_sys system call



**Dmy system call**

```
$ dmy 3 &
$ Parent 8 creating child 9
Child 9 created
```

**Pstat system call**

```
pstat
name      pid      state        priority
init      1        SLEEPING      10
sh        2        SLEEPING      10
dmy       9        RUNNABLE      10
dmy       8        SLEEPING      10
dmy       12       RUNNING       10
dmy       11       SLEEPING      10
pstat     13       RUNNING       10
$
```

**Chgp system call**

```
$ chgp 8 6
 pid=8, pr=6
$ pstat
name      pid      state          priority
init      1        SLEEPING        10
sh        2        SLEEPING        10
dmy       9        RUNNING         10
dmy       8        SLEEPING        6
dmy       12       RUNNABLE        10
dmy       11       SLEEPING        10
pstat     15       RUNNING         10
$ chgp 11 4
 pid=11, pr=4
$ pstat
name      pid      state          priority
init      1        SLEEPING        10
sh        2        SLEEPING        10
dmy       9        RUNNABLE        10
dmy       8        SLEEPING        6
dmy       12       RUNNING         10
dmy       11       SLEEPING        4
pstat     17       RUNNING         10
$ S
```

```
$ pstat
name      pid      state          priority
init      1        SLEEPING        10
sh        2        SLEEPING        10
dmy       9        RUNNING         8
dmy       8        SLEEPING        6
dmy       12       RUNNABLE        10
dmy       11       SLEEPING        4
pstat     20       RUNNING         10
```

# CONCLUSION

➢ We can add system calls such as **create child processes** with our own fork system call and also control its **run time** before it becomes zombie.

➢ We deep dive into the very basics of an Operating system and make changes in it at root level.We can add features to any Open Source OS similarly.

➢ Some system calls do not take any arguments and return just an integer value (e.g., uptime in sysproc.c). Some other system calls take in multiple arguments like strings and integers (e.g., open system call in sysfile.c), and return a simple integer value. Further, more complex system calls return a lot of information back to the user program in a user-defined structure.

# REFERENCES


➢https://medium.com/@silvamatteus/adding-user-programs-to-xv6-ba9896605942

➢https://en.wikipedia.org/wiki/Xv6

➢https://github.com/mit-pdos/xv6-public

➢https://www.ijsr.net/archive/v6i1/5011702.pdf