

190031187

NERELLA VENKATA RADHAKRISHNA

**IMPLEMENTATION OF SHELL, EDITOR, LS, Porting xv6 with POSIX compliance +
VFS + ulibc + ACPI, AND Mirrored RAID feature IN XV6.**

A SKILLING PROJECT REPORT

Submitted towards the professional course

19CS2106A Operating System Design

Nerella Venkata Radhakrishna (190031187)

BACHELOR OF TECHNOLOGY

Branch: Computer Science and Engineering



NOVEMBER 2020

Department of Computer Science and Engineering

K L Deemed to be University,

Green Fields, Vaddeswaram,

Guntur District, A.P., 522502.

TABLE OF CONTENTS

CHAPTER NO.	TITLE
1	What is XV6?
2	SRS
3	Data Flow Diagram
4	Modules Description
5	Codes with output
6	Conclusion

WHAT IS XV6?

xv6 is a modern reimplementation of Sixth Edition Unix in ANSI C for multiprocessor x86 and RISC-V systems.

Purpose: Unlike Linux or BSD, xv6 is simple enough to cover in a semester, yet still contains the important concepts and organization of Unix.[2] Rather than study the original V6 code, the course uses xv6 since PDP-11 machines are not widely available and the original operating system was written in archaic pre-ANSI C.

Xv6 can be cloned into your system with the below command

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

System Requirements Specification

1 Introduction

1.1 Purpose

- Run an improvised version of the MIT XV6 basic OS
- Implement most common Command Line Interface functionalities in XV6
- Enhance smooth operation of the XV6
- Ensure security for the all the documents which will be saved in XV6

1.2 Scope

With the decrease in the number of people actually learning to work with the base OS like XV6 due to its lack of functionality even for educational purposes. We took it upon ourselves to create a Shell in XV6 with all the functionalities which we think is absolutely necessary for us someone to use it properly without any problem.

1.3 Overview of the system

The system focuses on improving the already existing open source XV6-public OS distribution by MIT on GitHub and use create the basic shell functionalities like Copying, Moving and Editing files and also to display all running process. This means we create a basic working Editor and add extra functionalities into it while at the same time implementing all missing common Linux commands.

2 General Requirements

- Basic XV6 – use the MIT XV6 as a base code and make it run
- Copy – Implement a copy function to copy files from one location to another
- Move – enable moving a file from one location to another using the function
- Head – display first 10 lines of any file
- Tail – Display last 10 lines of any file
- Editor – Create a basic editor to create and modify files
- Process Display – display all running process

3 Functional Requirements

3.1 Necessary requirements

- The user should have general computer knowledge
- The users should have a popular Linux Distribution
- User should have a virtualization command like Qemu or Qemu-KVD
- User should be comfortable with working on a sole Command-Line Interface without any mouse usage

3.2 Technical requirements

- **Linux Distro with QEMU or any other Virtualization support must be installed.**

4 Interface requirements

4.1 Software Requirements

Visual Studio Code – A basic editor for modifying the code

4.2 Hardware Requirements

- Intel core i3 processor at 2.0 GHz or higher
- 256 MB RAM or higher
- 256 GB of Hard disk

6 Performance Requirements

- Response time of the system should be as quick as possible.
- In case of technical issues, the system should try to handle it without entering Panic State

7 References

- XV6 MIT PDOS
- COL331/COL633 Operating Systems Course Lecture Videos.
- XV6 Survival Guide
- XV6 Code Sheet

Data Flow Models

Level 0 DFD

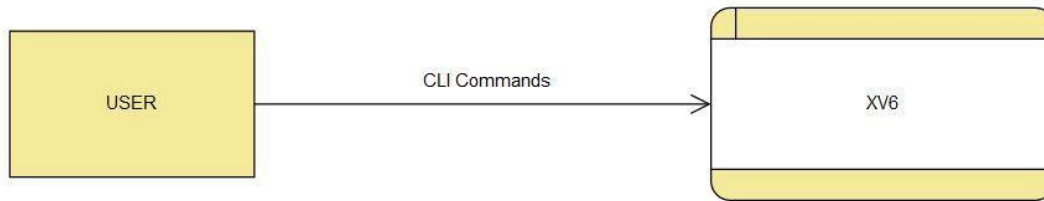


TABLE DESCRIPTIONS

Main Memory

The RAM and HDD/SSD parts of an OS where all data is finally stored. It does not lose any data even when the OS enters a panic state or is shut down. It has a logical memory address or physical memory address. The RAM houses all files which are for immediate access while the HDD/SSD houses the rest.

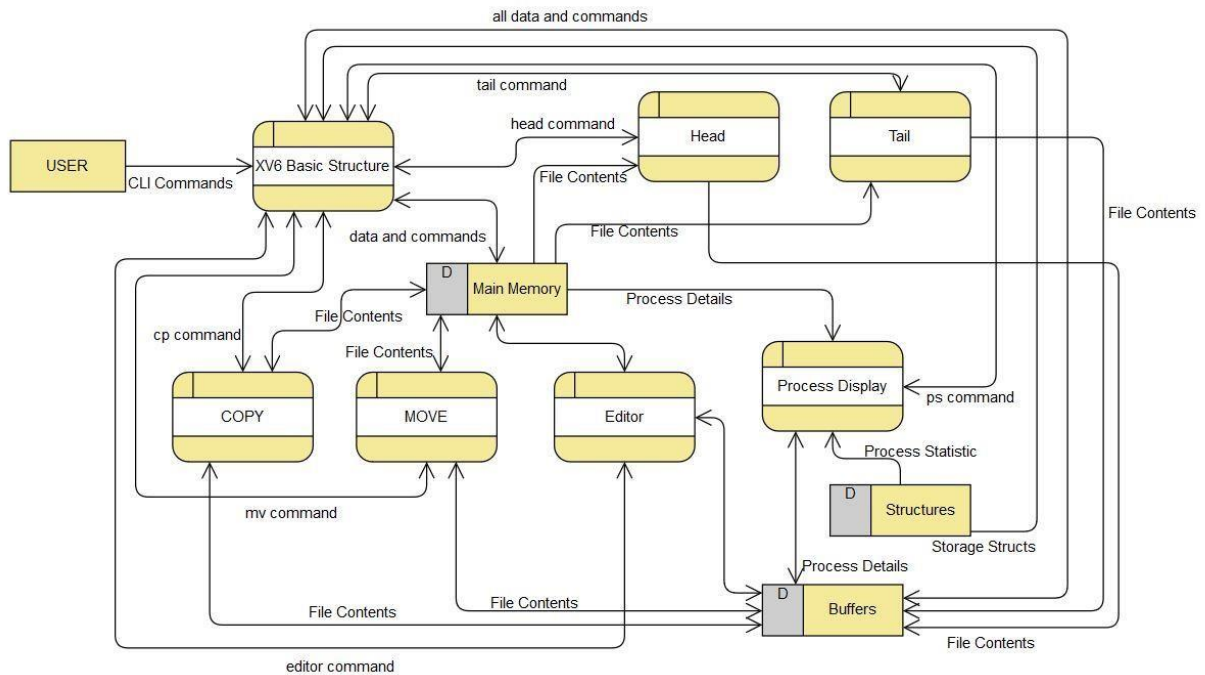
Buffers

The buffers are streams or intermediate storages that house all data for display or modification. The stream 2 is connected straight to the output terminal and is used for displaying in the Terminal. The other streams are used to carry around information and commands from all devices and the CPU.

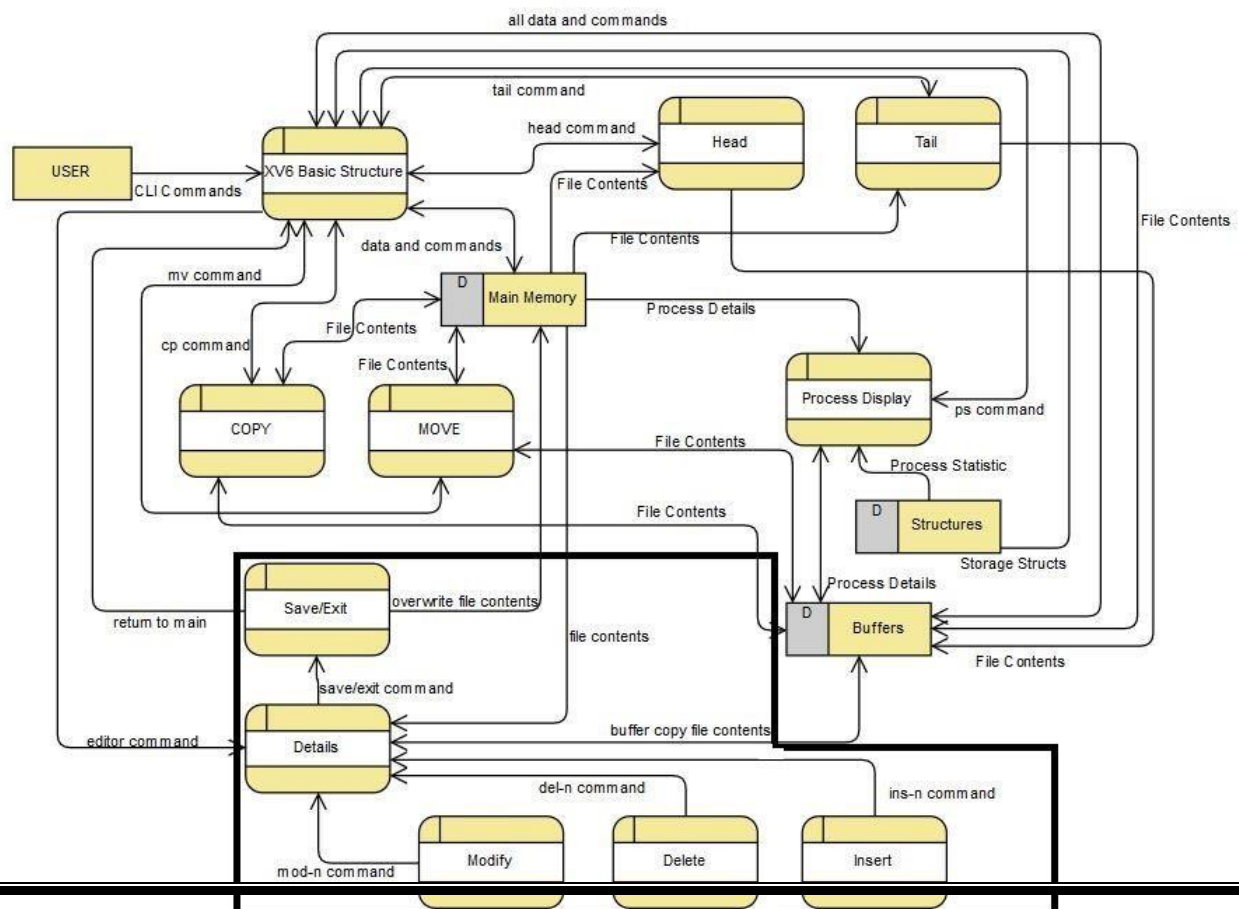
Structures

This Data Store stores all necessary structures required for functioning of a CPU. This table has predefined structures and cannot be modified unless the change is done directly to the source code. This data store houses the structures of Process Statistics or File Structures and is used for initiation of all core functionalities of a system.

Level 1 DFD



Level 2 DFD



MODULES DESCRIPTION**Shell**

Shell is the primary user interface to traditional Unix – like systems including XV6. The shell is an ordinary program (user level) that reads commands from the user and executes them. The xv6 shell is a simple implementation of the essence of the Unix Bourne shell. And It is a program that executes the other programs. It is user interface for access to an operating system service.

Editor

Syntax: editor file1 or bedit file1 mode

Mandatory Parameters: file1

This module is used to open a basic editor that can be used to create a new file or view and modify an existing file. The editor can be used to insert, modify or delete a particular line. It can also be used to insert a huge block of text. The editor can also be used to add lines at end of the file. The editor displays the number of lines at each line and that can be used to specify after which line you need to insert or modify. When invoked, the editor goes to fetch the filename and if its non-existent, it then goes on to create a file of the given name. It then prints the whole text along with line numbers and then shows all possible options to choose from and execute. At the end, you can choose to exit with or without saving all changes.

POSIX

The Portable Operating System Interface (POSIX) is an IEEE standard that helps compatibility and portability between operating systems. Theoretically, POSIX compliant source code should be seamlessly portable. In the real world, application transition often runs into system specific issues.

VFS

The Virtual File System (also known as the Virtual Filesystem Switch) is the software layer in the kernel that provides the filesystem interface to user space programs. It also provides an abstraction within the kernel which allows different filesystem implementations to coexist.

Ulibc

It is a small C standard library intended for Linux kernel-based operating systems for embedded systems and mobile devices.

ACPI

ACPI, known as a Hardware Abstraction Layer (HAL) in embedded computing, is an abstraction layer between the operating system, platform firmware and hardware. This allows the OS and the platform to evolve independently. The core of the Linux ACPI implementation comes from ACPICA (ACPI Component Architecture).

These are basically enchantments (special features) to help in improvising a better a xv6.

Mirrored Raid

RAID Mirroring means an exact clone (or mirror) of the same data writing to two drives. A minimum two number of disks are more required in an array to create RAID1 and it's useful only, when read performance or reliability is more precise than the data storage capacity.

Here we perform two tests for Mirrored Raid Feature.

Shell Code | Task - 1:

```
// Shell.
#include "types.h"
#include "user.h"
#include "param.h"
#include "mmu.h"
#include "fcntl.h"
#include "proc.h"
#include "spinlock.h"
#define NELEM(x) (sizeof(x)/sizeof((x)[0]))
// Parsed command representation
#define EXEC 1
#define REDIR 2
#define PIPE 3
#define LIST 4
#define BACK 5
#define NULL 0
#define MAXARGS 10
// #define INT_MAX 2147483647

/// By Us
char *strcat(char *strg1, char *strg2)
{
    char *start = strg1;
    while(*strg1 != '\0')
    {
        strg1++;
    }
    while(*strg2 != '\0')
    {
        *strg1 = *strg2;
        strg1++;
        strg2++;
    }

    *strg1 = '\0';
    return start;
}
```

```
}
struct cmd {
int type;
};
struct execcmd {
int type;
char *argv[MAXARGS];
char *eargv[MAXARGS];
};
struct redircmd {
int type;
struct cmd *cmd;
char *file;
char *efile;
int mode;
int fd;
};
struct pipecmd {
int type;
struct cmd *left;
struct cmd *right;
};
struct listcmd {
int type;
struct cmd *left;
struct cmd *right;
};
struct backcmd {
int type;
struct cmd *cmd;
};
/// pwd
struct directory{
char string[100];
struct directory *Next;
struct directory *Before;
};
```

```
int fork1(void); // Fork but panics on failure.
void panic(char*);
struct cmd *parsecmd(char*);
struct {
    struct spinlock lock;
    struct proc proc[NPROC];
} ptable;
///Build Directory
struct directory* CreateNode(char *Str)
{
    struct directory* Temp = malloc(sizeof(struct directory));
    //Temp->string = malloc(sizeof(Str));
    strcpy(Temp->string, Str);
    Temp->Before = Temp->Next = NULL;
    return Temp;
}
// Execute cmd. Never returns.
void
runcmd(struct cmd *cmd)
{
    int p[2];
    struct backcmd *bcmd;
    struct execcmd *ecmd;
    struct listcmd *lcmd;
    struct pipecmd *pcmd;
    struct redircmd *rcmd;
    char Point[] = "/";
    if(cmd == 0)
        exit();
    switch(cmd->type){
    default:
        panic("runcmd");
    case EXEC:
        ecmd = (struct execcmd*)cmd;
        if(ecmd->argv[0] == 0)
            exit();
        exec(strcat(Point, ecmd->argv[0]), ecmd->argv);
```

```
printf(2, "exec %s failed\n", ecmd->argv[0]);
break;
case REDIR:
rcmd = (struct redircmd*)cmd;
close(rcmd->fd);
if(open(rcmd->file, rcmd->mode) < 0){
printf(2, "open %s failed\n", rcmd->file);
exit();
}
runcmd(rcmd->cmd);
break;
case LIST:
lcmd = (struct listcmd*)cmd;
if(fork1() == 0)
runcmd(lcmd->left);
wait();
runcmd(lcmd->right);
break;
case PIPE:
pcmd = (struct pipecmd*)cmd;
if(pipe(p) < 0)
panic("pipe");
if(fork1() == 0){
close(1);
dup(p[1]);
close(p[0]);
close(p[1]);
runcmd(pcmd->left);
}
if(fork1() == 0){
close(0);
dup(p[0]);
close(p[0]);
close(p[1]);
runcmd(pcmd->right);
}
close(p[0]);
```

```
close(p[1]);
wait();
wait();
break;
case BACK:
    bcmd = (struct backcmd*)cmd;
    if(fork1() == 0)
        runcmd(bcmd->cmd);
    break;
}
exit();
}
int
getcmd(char *buf, int nbuf)
{
    printf(2, "$ ");
    memset(buf, 0, nbuf);
    gets(buf, nbuf);
    if(buf[0] == 0) // EOF
        return -1;
    return 0;
}
int
main(void)
{
    static char buf[100];
    int fd;
    // Assumes three file descriptors open.
    while((fd = open("console", O_RDWR)) >= 0){
        if(fd >= 3){
            close(fd);
            break;
        }
    }
    struct directory *Head_Directory = CreateNode("/");
    struct directory *Curr = Head_Directory;
    struct directory *prev = NULL;
```

```
// Read and run input commands.
while(getcmd(buf, sizeof(buf)) >= 0){
    if(buf[0] == 'c' && buf[1] == 'd' && buf[2] == ' '){
        // Clumsy but will have to do for now.
        // Chdir has no effect on the parent if run in the child.
        buf[strlen(buf)-1] = 0; // chop \n
        int returnStatus = chdir(buf+3);
        if(returnStatus < 0) {
            printf(2, "cannot cd %s\n", buf + 3);
        } else {/// By US
            if(buf[3] == '/' && buf[4] == NULL)
            {
                Curr = Head_Directory;
                Curr->Next = NULL;
                prev = NULL;
                continue;
            }
            if(buf[3] == '.' && buf[4] == '.')
            {
                if(Curr != Head_Directory)
                {
                    if(Curr->Before == Head_Directory)
                    {
                        Curr = Head_Directory;
                        Curr->Next = NULL;
                        prev = NULL;
                        continue;
                    }
                    Curr = Curr->Before->Before;
                    Curr->Next = NULL;
                    prev = Curr->Before;
                }
                continue;
            }
            if(buf[3] == '.' && buf[4] == NULL)
            {
                continue;
            }
        }
    }
}
```



```
}
int Flag = 0,i,k;
for(i = 4;i<strlen(buf);i++)
{
if(buf[i] == '/')
{
Flag = 1;
break;
}
}
struct directory *Next;
if(Flag){
char buffer[100];
if (buf[3] == '/')
{
Curr = Head_Directory;
Curr->Next = NULL;
prev = NULL;
}
for(i=3,k=0;i<strlen(buf);i++)// YET TO BE PERFECTED(several directory climb)
{
if ((strlen(buf) == i || i == 3) && buf[i] == '/'){
continue;
}
if (buffer[k-1] == '\0')
k=0;
if(buf[i] != '/'){
buffer[k++] = buf[i];
printf(1,"%s\n",buffer);
continue;
}
else
{
buffer[k++] = '\0';
if((i != 3 && buf[i] == '/') && Curr != Head_Directory)
{
Next = CreateNode("/");
```

```
Curr->Next = Next;
Curr->Before = prev;
prev = Curr;
Next->Before = Curr;
Curr = Curr->Next;
}
Next = CreateNode(buffer);
Curr->Next = Next;
Curr->Before = prev;
prev = Curr;
Next->Before = Curr;
Curr = Curr->Next;
}
}
if (buf[strlen(buf)] != '/') {
Next = CreateNode("/");
Curr->Next = Next;
Curr->Before = prev;
prev = Curr;
Next->Before = Curr;
Curr = Curr->Next;
Next = CreateNode(buffer);
Curr->Next = Next;
Curr->Before = prev;
prev = Curr;
Next->Before = Curr;
Curr = Curr->Next;
}
continue;
}
if (buf[3] == '/' )
{
Curr = Head_Directory;
Curr->Next = NULL;
prev = NULL;
Next = CreateNode(buf+4);
Curr->Next = Next;
```

```
Curr->Before = prev;
prev = Curr;
Next->Before = Curr;
Curr = Curr->Next;
continue;
}
if (Curr != Head_Directory && buf[3] != '/'){
Next = CreateNode("/");
Curr->Next = Next;
Curr->Before = prev;
prev = Curr;
Next->Before = Curr;
Curr = Curr->Next;
}
Next = CreateNode(buf+3);
Curr->Next = Next;
Curr->Before = prev;
prev = Curr;
Next->Before = Curr;
Curr = Curr->Next;
}
continue;
}
if(buf[0] == 'p' && buf[1] == 'w' && buf[2] == 'd')
{
struct directory *iter = Head_Directory;
while(iter)
{
printf(1,iter->string);
iter = iter->Next;
}
printf(1,"\n");
continue;
}
if(buf[0] == 'p' && buf[1] == 's')
{
static char *states[] = {
```

```
[UNUSED] "unused",
[EMBRYO] "embryo",
[SLEEPING] "sleep ",
[RUNNABLE] "runble",
[RUNNING] "run ",
[ZOMBIE] "zombie"
};
char *state;
struct proc *p;
printf(1, "F S UID PID PPID SZ WCHAN COMD\n");
for (p = ptable.proc; p < &ptable.proc[NPROC]; p++) {
    if (p->state == UNUSED)
        continue;
    if (p->state >= 0 && p->state < NELEM(states) && states[p->state])
        state = states[p->state];
    else
        state = "???";
    printf(1, "2 %s Root %d %d %d %d %s\n", state, p->pid, p->parent->pid, p->sz, p->chan, p->name);
}
continue;
}
if(fork1() == 0)
    runcmd(parsecmd(buf));
wait();
}
exit();
}
void
panic(char *s)
{
    printf(2, "%s\n", s);
    exit();
}
int
fork1(void)
{
```

```
int pid;
pid = fork();
if(pid == -1)
panic("fork");
return pid;
}
//PAGEBREAK!
// Constructors
struct cmd*
execcmd(void)
{
struct execcmd *cmd;
cmd = malloc(sizeof(*cmd));
memset(cmd, 0, sizeof(*cmd));
cmd->type = EXEC;
return (struct cmd*)cmd;
}
struct cmd*
redircmd(struct cmd *subcmd, char *file, char *efile, int mode, int fd){
struct redircmd *cmd;
cmd = malloc(sizeof(*cmd));
memset(cmd, 0, sizeof(*cmd));
cmd->type = REDIR;
cmd->cmd = subcmd;
cmd->file = file;
cmd->efile = efile;
cmd->mode = mode;
cmd->fd = fd;
return (struct cmd*)cmd;
}
struct cmd*
pipecmd(struct cmd *left, struct cmd *right)
{
struct pipecmd *cmd;

cmd = malloc(sizeof(*cmd));
memset(cmd, 0, sizeof(*cmd));
```

```
cmd->type = PIPE;
cmd->left = left;
cmd->right = right;
return (struct cmd*)cmd;
}
struct cmd*
listcmd(struct cmd *left, struct cmd *right)
{
    struct listcmd *cmd;
    cmd = malloc(sizeof(*cmd));
    memset(cmd, 0, sizeof(*cmd));
    cmd->type = LIST;
    cmd->left = left;
    cmd->right = right;
    return (struct cmd*)cmd;
}
struct cmd*
backcmd(struct cmd *subcmd)
{
    struct backcmd *cmd;
    cmd = malloc(sizeof(*cmd));
    memset(cmd, 0, sizeof(*cmd));
    cmd->type = BACK;
    cmd->cmd = subcmd;
    return (struct cmd*)cmd;
}
//PAGEBREAK!
// Parsing
char whitespace[] = " \t\r\n\v";
char symbols[] = "<|>&;()";
int
gettoken(char **ps, char *es, char **q, char **eq)
{
    char *s;
    int ret;
    s = *ps;
    while(s < es && strchr(whitespace, *s))
```

```
s++;
if(q)
*q = s;
ret = *s;
switch(*s){
case 0:
break;
case '|':
case '(':
case ')':
case ';':
case '&':
case '<':
s++;
break;
case '>':
s++;
if(*s == '>'){
ret = '+';
s++;
}
break;
default:
ret = 'a';
while(s < es && !strchr(whitespace, *s) && !strchr(symbols, *s))
s++;
break;
}
if(eq)
*eq = s;
while(s < es && strchr(whitespace, *s))
s++;
*ps = s;
return ret;
}
int
peek(char **ps, char *es, char *toks)
```

```
{
char *s;
s = *ps;
while(s < es && strchr(whitespace, *s))
s++;
*ps = s;
return *s && strchr(toks, *s);
}

struct cmd *parseline(char**, char*);
struct cmd *parsepipe(char**, char*);
struct cmd *parseexec(char**, char*);
struct cmd *nulterminate(struct cmd*);
struct cmd*
parsecmd(char *s)
{
char *es;
struct cmd *cmd;
es = s + strlen(s);
cmd = parseline(&s, es);
peek(&s, es, "");
if(s != es){
printf(2, "leftovers: %s\n", s);
panic("syntax");
}
nulterminate(cmd);
return cmd;
}

struct cmd*
parseline(char **ps, char *es)
{
struct cmd *cmd;
cmd = parsepipe(ps, es);
while(peek(ps, es, "&")){
gettoken(ps, es, 0, 0);
cmd = backcmd(cmd);
}
if(peek(ps, es, ";")){
```




```
gettoken(ps, es, 0, 0);
cmd = listcmd(cmd, parseline(ps, es));
}
return cmd;
}
struct cmd*
parsepipe(char **ps, char *es)
{
    struct cmd *cmd;
    cmd = parseexec(ps, es);
    if(peek(ps, es, "|")){
        gettoken(ps, es, 0, 0);
        cmd = pipecmd(cmd, parsepipe(ps, es));
    }
    return cmd;
}
struct cmd*
parseredirs(struct cmd *cmd, char **ps, char *es)
{
    int tok;
    char *q, *eq;
    while(peek(ps, es, "<>")){
        tok = gettoken(ps, es, 0, 0);
        if(gettoken(ps, es, &q, &eq) != 'a')
            panic("missing file for redirection");
        switch(tok){
            case '<':
                cmd = redircmd(cmd, q, eq, O_RDONLY, 0); break;
            case '>':
                cmd = redircmd(cmd, q, eq, O_WRONLY|O_CREATE, 1);
                break;
            case '+': // >>
                cmd = redircmd(cmd, q, eq, O_WRONLY|O_CREATE, 1);
                break;
        }
    }
    return cmd;
}
```

```
}
struct cmd*
parseblock(char **ps, char *es)
{
    struct cmd *cmd;
    if(!peek(ps, es, "("))
        panic("parseblock");
    gettoken(ps, es, 0, 0);
    cmd = parseline(ps, es);
    if(!peek(ps, es, ")"))
        panic("syntax - missing )");
    gettoken(ps, es, 0, 0);
    cmd = parseredirs(cmd, ps, es);
    return cmd;
}
struct cmd*
parseexec(char **ps, char *es)
{
    char *q, *eq;
    int tok, argc;
    struct execcmd *cmd;
    struct cmd *ret;
    if(peek(ps, es, "("))
        return parseblock(ps, es);
    ret = execcmd();
    cmd = (struct execcmd*)ret;
    argc = 0;
    ret = parseredirs(ret, ps, es);
    while(!peek(ps, es, "|)&")){
        if((tok=gettoken(ps, es, &q, &eq)) == 0)
            break;
        if(tok != 'a')
            panic("syntax");
        cmd->argv[argc] = q;
        cmd->eargv[argc] = eq;
        argc++;
        if(argc >= MAXARGS)
```

```
panic("too many args");
ret = parseredirs(ret, ps, es);
}
cmd->argv[argc] = 0;
cmd->eargv[argc] = 0;
return ret;
}
// NUL-terminate all the counted strings.
struct cmd*
nulterminate(struct cmd *cmd)
{
    int i;
    struct backcmd *bcmd;
    struct execcmd *ecmd;
    struct listcmd *lcmd;
    struct pipecmd *pcmd;
    struct redircmd *rcmd;
    if(cmd == 0)
        return 0;
    switch(cmd->type){
    case EXEC:
        ecmd = (struct execcmd*)cmd;
        for(i=0; ecmd->argv[i]; i++)
            *ecmd->eargv[i] = 0;
        break;
    case REDIR:
        rcmd = (struct redircmd*)cmd;
        nulterminate(rcmd->cmd);
        *rcmd->efile = 0;
        break;
    case PIPE:
        pcmd = (struct pipecmd*)cmd;
        nulterminate(pcmd->left);
        nulterminate(pcmd->right);
        break;
    case LIST:
        lcmd = (struct listcmd*)cmd;
```

```
nulterminate(lcmd->left);
nulterminate(lcmd->right);
break;
case BACK:
bcmd = (struct backcmd*)cmd;
nulterminate(bcmd->cmd);
break;
}
return cmd;
}
```

OUTPUT:

 osd-190031187@team-osd:~/190031187-xv6

```
SeaBIOS (version 1.11.0-2.el7)

iPXE (http://ipxe.org) 00:03.0 C980 PCI2.10 PnP PMM+1FF94780+1FED4780 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 start 58
init: starting sh
190031187$ ls
.          1 1 512
..         1 1 512
README    2 2 2286
cat       2 3 14480
echo      2 4 13336
forktest  2 5 8160
grep      2 6 16016
init      2 7 14224
kill      2 8 13368
ln        2 9 13308
ls        2 10 16168
mkdir     2 11 13400
rm        2 12 13376
sh        2 13 31064
stressfs  2 14 14324
usertests 2 15 67224
wc        2 16 15144
zombie    2 17 13036
editor    2 18 26800
console   3 19 0
OSD       1 20 48
f1.txt    2 21 16
190031187$ cd OSD
190031187$ pwd
/OSD
190031187$ ls
.          1 20 48
..         1 1 512
f2.txt    2 22 13
190031187$ █
```

Observation and Analysis:

In Old shell pwd and ls was not working inside a directory. Here we observe, that an improvised shell is being used and was checked if properly working or not by executing one of the commands from list above.

→mkdir OSD

→cd OSD

→pwd

→ls

Editor Code | Task-2:

```
#include "types.h"
#include "stat.h"
#include "user.h"
#include "fcntl.h"
#include "fs.h"
#define BUF_SIZE 256
#define MAX_LINE_NUMBER 256
#define MAX_LINE_LENGTH 256
#define MAX_ROLLBACK_STEP 20
#define NULL 0
// define some color variables
#define NONE          "\e[0m"
#define BLACK         "\e[0;30m"
#define L_BLACK       "\e[1;30m"
#define RED           "\e[0;31m"
#define L_RED         "\e[1;31m"
#define GREEN         "\e[0;32m"
#define L_GREEN       "\e[1;32m"
#define YELLOW        "\e[0;33m"
#define L_YELLOW      "\e[1;33m"
#define BLUE          "\e[0;34m"
#define L_BLUE        "\e[1;34m"
#define PURPLE        "\e[0;35m"
#define L_PURPLE      "\e[1;35m"
#define CYAN          "\e[0;36m"
#define L_CYAN        "\e[1;36m"
#define GRAY          "\e[0;37m"
#define WHITE         "\e[1;37m"

char* strcat_n(char* dest, char* src, int len);
int get_line_number(char *text[]);
void show_text(char *text[]);
void com_ins(char *text[], int n, char *extra, int flag);
```

```
void com_mod(char *text[], int n, char *extra, int flag);
void com_del(char *text[], int n, int flag);
void com_help(char *text[]);
void com_save(char *text[], char *path);
void com_exit(char *text[], char *path);
void com_create_new_file(char *text[], char *path);
void com_display_color_demo();
void com_init_file(char *text[], char *path);
void show_text_syntax_highlighting(char *text[]);
void com_rollback(char *text[], int n);
void record_command(char *command);
int stringtonumber(char* src);
void number2string(int num, char array[]);
```

```
int changed = 0;
int auto_show = 1;
int line_number = 0;
char *command_set[MAX_ROLLBAKC_STEP] = {};
int upper_bound = -1;
int main(int argc, char *argv[]) {
    int is_new_file = 0;
    if (argc == 1) {
        printf(1, ">>> \e[1;31mplease input the command as [editor
file_name]\n\e[0m");
        exit();
    }
    char *text[MAX_LINE_NUMBER] = {};
    text[0] = malloc(MAX_LINE_LENGTH);
    memset(text[0], 0, MAX_LINE_LENGTH);
    int fd = open(argv[1], O_RDONLY);
    if (fd != -1) {
        //printf(1, ">>> \e[1;33mfile exist\n\e[0m");
        char buf[BUF_SIZE] = {};
        int len = 0;
```

```
while ((len = read(fd, buf, BUF_SIZE)) > 0) {
    int i = 0;
    int next = 0;
    int is_full = 0;
    while (i < len) {
        for (i = next; i < len && buf[i] != '\n'; i++)
            ;
        strcat_n(text[line_number], buf+next, i-next);
        if (i < len && buf[i] == '\n') {
            if (line_number >= MAX_LINE_NUMBER - 1)
                is_full = 1;
            else {
                line_number++;
                text[line_number] =
malloc(MAX_LINE_LENGTH);
                memset(text[line_number], 0,
MAX_LINE_LENGTH);
            }
        }
        if (is_full == 1 || i >= len - 1)
            break;
        else
            next = i + 1;
    }
    if (is_full == 1)
        break;
}
close(fd);
} else{
    com_create_new_file(text, argv[1]);
    is_new_file = 1;
}
if(!is_new_file && auto_show){
    show_text_syntax_highlighting(text);
}
```



```
char input[MAX_LINE_LENGTH] = {};
while (1)
{
    //printf(1, ">>> \e[1;33mplease input command:\n\e[0m");
    printf(1, ">>> \e[1;33m\e[0m");
    memset(input, 0, MAX_LINE_LENGTH);
    gets(input, MAX_LINE_LENGTH);
    int len = strlen(input);
    input[len-1] = '\0';
    len--;
    int pos = MAX_LINE_LENGTH - 1;
    int j = 0;
    for (; j < 8; j++)
    {
        if (input[j] == ' ')
        {
            pos = j + 1;
            break;
        }
    }
    if (input[0] == 'i' && input[1] == 'n' && input[2] == 's')
    {
        if (input[3] == '-' && stringtonumber(&input[4]) >= 0)
        {
            com_ins(text, stringtonumber(&input[4]), &input[pos], 1);
            line_number = get_line_number(text);
        }
        else if(input[3] == ' ' | input[3] == '\0')
        {
            com_ins(text, line_number+1+1, &input[pos], 1);
            line_number = get_line_number(text);
        }
        else
        {

```

```
                                //printf(1, ">>> \033[1m\e[43;31minvalid
command.\e[0m\n");
                                printf(1, ">>> \033[1m\e[41;33minvalid
command.\e[0m\n");
                                //com_help(text);
                                }
                                }
                                //mod
                                else if (input[0] == 'm' && input[1] == 'o' && input[2] == 'd')
                                {
                                    if (input[3] == '-'&&stringtonumber(&input[4])>=0)
                                        com_mod(text, atoi(&input[4]), &input[pos], 1);
                                    else if(input[3] == ' ' | input[3] == '\0')
                                        com_mod(text, line_number + 1, &input[pos], 1);
                                    else {
                                        printf(1, ">>> \033[1m\e[41;33minvalid
command.\e[0m\n");
                                        //com_help(text);
                                    }
                                }
                                //del
                                else if (input[0] == 'd' && input[1] == 'e' && input[2] == 'l') {
                                    if (input[3] == '-'&&stringtonumber(&input[4])>=0)
                                    {
                                        com_del(text, stringtonumber(&input[4]), 1);
                                        line_number = get_line_number(text);
                                    }
                                    else if(input[3]=='\0')
                                    {
                                        com_del(text, line_number + 1, 1);
                                        line_number = get_line_number(text);
                                    }
                                    else
                                    {
```

```
printf(1, ">>> \033[1m\e[41;33minvalid
command.\e[0m\n");
    //com_help(text);
}

}
else if (strcmp(input, "show") == 0)
{
    auto_show = 1;
    printf(1, ">>> \e[1;33menable show current contents after text
changed.\n\e[0m");
}
else if (strcmp(input, "hide") == 0)
{
    auto_show = 0;
    printf(1, ">>> \e[1;33mdisable show current contents after
text changed.\n\e[0m");
}
// rollback
else if(strcmp(input, "rb") == 0){
    com_rollback(text, 1);
}
else if (strcmp(input, "help") == 0)
    com_help(text);
// save
else if (strcmp(input, "save") == 0 || strcmp(input, "CTRL+S\n") == 0)
    com_save(text, argv[1]);
else if (strcmp(input, "exit") == 0)
    com_exit(text, argv[1]);
else if (strcmp(input, "demo") == 0)
    com_display_color_demo();
else if (strcmp(input, "init") == 0)
    com_init_file(text, argv[1]);
else if(strcmp(input, "disp") == 0){
    show_text_syntax_highlighting(text);
```

```
    }
    else if(strcmp(input, "normaldisp") == 0){
        show_text(text);
    }
    else
    {
        printf(1, ">>> \033[1m\e[41;33minvalid command.\e[0m\n");
        //com_help(text);
    }
}
exit();
}
char* strcat_n(char* dest, char* src, int len)
{
    if (len <= 0)
        return dest;
    int pos = strlen(dest);
    if (len + pos >= MAX_LINE_LENGTH)
        return dest;
    int i = 0;
    for (; i < len; i++)
        dest[i+pos] = src[i];
    dest[len+pos] = '\0';
    return dest;
}
void show_text(char *text[])
{
    printf(1, ">>> \033[1m\e[45;33mthe contents of the file are:\e[0m\n");
    printf(1, "\n");
    int j = 0;
    for (; text[j] != NULL; j++)
        if(strcmp(text[j], "\n") == 0){
            printf(1, "\e[1;30m%d%d%d\e[0m\e[0;32m|\e[0m\n",
(j+1)/100, ((j+1)%100)/10, (j+1)%10);
        }
}
```

```
        else{
            printf(1, "\\e[1;30m%d%d%d\\e[0m\\e[0;32m|\\e[0m%s\\n",
(j+1)/100, ((j+1)%100)/10, (j+1)%10, text[j]);
        }
        printf(1, "\\n");
    }
int get_line_number(char *text[])
{
    int i = 0;
    for (; i < MAX_LINE_NUMBER; i++)
        if (text[i] == NULL)
            return i - 1;
    return i - 1;
}
int stringtonumber(char* src)
{
    int number = 0;
    int i=0;
    int pos = strlen(src);
    for(;i<pos;i++)
    {
        if(src[i]==' ') break;
        if(src[i]>57 || src[i]<48) return -1;
        number=10*number+(src[i]-48);
    }
    return number;
}
void number2string(int num, char array[]) {
    char array_rvs[20] = {};
    int i, sign;
    if ((sign = num)<0) // record the sign
        num = -num;          // make num into positive number
    i = 0;
    do {
        array_rvs[i++] = num % 10 + '0'; // fatch the next number
```

```
} while ((num /= 10)>0);                // delete this number
if (sign<0)
    array_rvs[i++] = '-';
array_rvs[i] = '\0';
int length = strlen(array_rvs);
int j;
for (j = 0; j < length; j++) {
    array[j] = array_rvs[length - 1 - j];
}
array[length] = '\0';
}

void com_ins(char *text[], int n, char *extra, int flag)
{
    if (n <= 0 || n > get_line_number(text) + 1 + 1)
    {
        printf(1, ">>> \033[1m\e[41;33minvalid line number\e[0m\n");
        return;
    }
    char input[MAX_LINE_LENGTH] = {};
    if (*extra == '\0')
    {
        printf(1, "... \e[1;35minput content:\e[0m");
        gets(input, MAX_LINE_LENGTH);
        input[strlen(input)-1] = '\0';
    }
    else
        strcpy(input, extra);

    char *part4 = malloc(MAX_LINE_LENGTH);
    if(flag){
        strcpy(part4, text[n-1]);
    }

    int i = MAX_LINE_NUMBER - 1;
```

```
for (; i > n-1; i--)
{
    if (text[i-1] == NULL)
        continue;
    else if (text[i] == NULL && text[i-1] != NULL)
    {
        text[i] = malloc(MAX_LINE_LENGTH);
        memset(text[i], 0, MAX_LINE_LENGTH);
        strcpy(text[i], text[i-1]);
    }
    else if (text[i] != NULL && text[i-1] != NULL)
    {
        memset(text[i], 0, MAX_LINE_LENGTH);
        strcpy(text[i], text[i-1]);
    }
}
// couldn't understand what this code block means
// maybe it allocates space for text[n-1] to avoid none space of text[n-1]
if (text[n-1] == NULL)
{
    text[n-1] = malloc(MAX_LINE_LENGTH);
    if (text[n-2][0] == '\0')
    {
        memset(text[n-1], 0, MAX_LINE_LENGTH);
        strcpy(text[n-2], input);
        changed = 1;
        if (auto_show == 1)
            show_text_syntax_highlighting(text);
        return;
    }
}
memset(text[n-1], 0, MAX_LINE_LENGTH);
strcpy(text[n-1], input);
changed = 1;
if(flag){
```

```
// record the command into command_set
char *command = malloc(MAX_LINE_LENGTH);
char part1[] = "ins-";
char part2[10];
number2string(n, part2);
char part3[] = " \0";
strcat_n(part1, part2, strlen(part2));
strcat_n(part1, part3, strlen(part3));
strcat_n(part1, part4, strlen(part4));
strcpy(command, part1);
record_command(command);
}

if (auto_show == 1)
    show_text_syntax_highlighting(text);
}

void com_mod(char *text[], int n, char *extra, int flag)
{
    if (n <= 0 || n > get_line_number(text) + 1)
    {
        printf(1, ">>> \033[1m\e[41;33minvalid line number\e[0m\n");
        return;
    }
    char input[MAX_LINE_LENGTH] = {};
    if (*extra == '\0')
    {
        printf(1, "... \e[1;35minput content:\e[0m");
        gets(input, MAX_LINE_LENGTH);
        input[strlen(input)-1] = '\0';
    }
    else
        strcpy(input, extra);

    char *part4 = malloc(MAX_LINE_LENGTH);
    if(flag){
```



```
        strcpy(part4, text[n-1]);
    }
    memset(text[n-1], 0, MAX_LINE_LENGTH);
    strcpy(text[n-1], input);
    changed = 1;
    if(flag){
        // record the command into command_set
        char *command = malloc(MAX_LINE_LENGTH);
        char part1[] = "mod-";
        char part2[10];
        number2string(n, part2);
        char part3[] = " \0";
        strcat_n(part1, part2, strlen(part2));
        strcat_n(part1, part3, strlen(part3));
        strcat_n(part1, part4, strlen(part4));
        strcpy(command, part1);
        record_command(command);
    }
    if (auto_show == 1)
        show_text_syntax_highlighting(text);
}

void com_del(char *text[], int n, int flag)
{
    if (n <= 0 || n > get_line_number(text) + 1)
    {
        //printf(1, "n: %d\n", n);
        printf(1, ">>> \033[1m\e[41;33minvalid line number\e[0m\n");
        return;
    }
    char *part4 = malloc(MAX_LINE_LENGTH);
    if(flag){
        strcpy(part4, text[n-1]);
    }
    memset(text[n-1], 0, MAX_LINE_LENGTH);
    int i = n - 1;
```

```
for (; text[i+1] != NULL; i++)
{
    strcpy(text[i], text[i+1]);
    memset(text[i+1], 0, MAX_LINE_LENGTH);
}
if (i != 0)
{
    free(text[i]);
    text[i] = 0;
}
changed = 1;
if(0){
    char part1[] = "del-";
    char part2[10];
    number2string(n, part2);
    char part3[] = " \0";
    strcat_n(part1, part2, strlen(part2));
    strcat_n(part1, part3, strlen(part3));
    char* part5 = malloc(MAX_LINE_LENGTH);
    memset(part5, 0, MAX_LINE_LENGTH);
    strcat_n(part5, part1, strlen(part1));
    strcat_n(part5, part4, strlen(part4));
    record_command(part5);
}
if (auto_show == 1)
    show_text_syntax_highlighting(text);
}
void com_save(char *text[], char *path)
{
    unlink(path);
    int fd = open(path, O_WRONLY|O_CREATE);
    if (fd == -1)
    {
        printf(1, ">>> \033[1m\e[41;33msave failed, file can't
open:\e[0m\n");
    }
}
```

```
        //setProgramStatus(SHELL);
        exit();
    }
    if (text[0] == NULL)
    {
        close(fd);
        return;
    }
    write(fd, text[0], strlen(text[0]));
    int i = 1;
    for (; text[i] != NULL; i++)
    {
        printf(fd, "\n");
        write(fd, text[i], strlen(text[i]));
    }
    close(fd);
    printf(1, ">>> \e[1;32msaved successfully\e[0m\n");
    changed = 0;
    return;
}

void com_exit(char *text[], char *path)
{
    while (changed == 1)
    {
        printf(1, ">>> \e[1;33msave the file?\e[0m\n");
        char input[MAX_LINE_LENGTH] = {};
        gets(input, MAX_LINE_LENGTH);
        input[strlen(input)-1] = '\0';
        if (strcmp(input, "y") == 0)
            com_save(text, path);
        else if (strcmp(input, "n") == 0)
            break;
        else
            printf(2, ">>> \e[1;31mwrong answer?\e[0m\n");
    }
}
```

```
    }
    int i = 0;
    for (; text[i] != NULL; i++)
    {
        free(text[i]);
        text[i] = 0;
    }
    exit();
}

// create new file
void com_create_new_file(char *text[], char *path){
    int fd = open(path, O_WRONLY|O_CREATE);
    if(fd == -1){
        printf(1, ">>> \e[1;31mcreate file failed\e[0m\n");
        exit();
    }
}

void com_display_color_demo(){
    printf(1, ">>> \e[1;33mcolor demo:\n\e[0m");
    printf(1, "-----+-----\n");
    printf(1, "L_BLACK:      | \e[1;30ml am happy.\e[0m   |
\e[1;30m\\e[1;30m\e[0m\n");
    printf(1, "BLACK:        | \e[0;30ml am happy.\e[0m   |
\e[0;30m\\e[0;30m\e[0m\n");
    printf(1, "RED:          | \e[0;31ml am happy.\e[0m   |
\e[0;31m\\e[0;31m\e[0m\n");
    printf(1, "L_RED:        | \e[1;31ml am happy.\e[0m   |
\e[1;31m\\e[1;31m\e[0m\n");
    printf(1, "GREEN:        | \e[0;32ml am happy.\e[0m   |
\e[0;32m\\e[0;32m\e[0m\n");
    printf(1, "L_GREEN:      | \e[1;32ml am happy.\e[0m   |
\e[1;32m\\e[1;32m\e[0m\n");
    printf(1, "YELLOW:       | \e[0;33ml am happy. \e[0m   |
\e[0;33m\\e[0;33m\e[0m\n");
```

```

printf(1, "L_YELLOW:      | \e[1;33ml am happy. \e[0m  |
\e[1;33m\\e[1;33m\e[0m\n");
printf(1, "BLUE:          | \e[0;34ml am happy. \e[0m  |
\e[0;34m\\e[0;34m\e[0m\n");
printf(1, "L_BLUE:        | \e[1;34ml am happy. \e[0m  |
\e[1;34m\\e[1;34m\e[0m\n");
printf(1, "PURPLE:        | \e[0;35ml am happy. \e[0m  |
\e[0;35m\\e[0;35m\e[0m\n");
printf(1, "L_PURPLE:      | \e[1;35ml am happy. \e[0m  |
\e[1;35m\\e[1;35m\e[0m\n");
printf(1, "CYAN:          | \e[0;36ml am happy. \e[0m  |
\e[0;36m\\e[0;36m\e[0m\n");
printf(1, "L_CYAN:        | \e[1;36ml am happy. \e[0m  |
\e[1;36m\\e[1;36m\e[0m\n");
printf(1, "GRAY:          | \e[0;37ml am happy. \e[0m  |
\e[0;37m\\e[0;37m\e[0m\n");
printf(1, "WHITE:         | \e[1;37ml am happy. \e[0m  |
\e[1;37m\\e[1;37m\e[0m\n");
printf(1, "-----+-----+-----\n");
}
void com_help(char *text[])
{
    printf(1, ">>> \e[1;33minstructions for use:\n\e[0m");
    printf(1, "-----+-----\n");
    printf(1, "\e[1;32mins-n:\e[0m  | insert a line after line n\n");
    printf(1, "\e[1;32mmod-n:\e[0m    | modify line n\n");
    printf(1, "\e[1;32mdel-n:\e[0m   | delete line n\n");
    printf(1, "\e[1;32mins:\e[0m    | insert a line after the last line\n");
    printf(1, "\e[1;32mmod:\e[0m    | modify the last line\n");
    printf(1, "\e[1;32mdel:\e[0m   | delete the last line\n");
    printf(1, "\e[1;32mshow:\e[0m    | enable show current contents after
executing a command.\n");
    printf(1, "\e[1;32mhide:\e[0m  | disable show current contents after
executing a command.\n");
    printf(1, "\e[1;32msave:\e[0m  | save the file\n");
}

```

```

printf(1, "\e[1;32mexit:\e[0m  | exit editor\n");
printf(1, "\e[1;32mhelp:\e[0m  | help info\n");
printf(1, "\e[1;32mdemo:\e[0m  | color demo\n");
printf(1, "\e[1;32minit:\e[0m  | initial file\n");
printf(1, "\e[1;32mdisp:\e[0m  | display with highlighting\n");
printf(1, "\e[1;32mrb:\e[0m    | rollback the file\n");
printf(1, "-----+-----\n");
}
void com_init_file(char *text[], char *path){
    char *buf[MAX_LINE_NUMBER] = {};
    int i;
    for(i = 0; i < MAX_LINE_NUMBER; i++){
        buf[i] = malloc(MAX_LINE_LENGTH);
    }
    strcpy(buf[0], "// Create a NULL-terminated string by reading the provided
file");
    strcpy(buf[1], "static char* readShaderSource(const char* shaderFile");
    strcpy(buf[2], "{");
    strcpy(buf[3], "    int flag = 24;");
    strcpy(buf[4], "    double ways = 100.43;");
    strcpy(buf[5], "    if ( fp == NULL ) {");
    strcpy(buf[6], "        return NULL;");
    strcpy(buf[7], "    }");
    strcpy(buf[8], "    fseek(fp, 0L, SEEK_END); //search something");
    strcpy(buf[9], "    long size = ftell(fp);");
    strcpy(buf[10], "    fseek(fp, 0L, SEEK_SET);");
    strcpy(buf[11], "    char* buf = new char[size + 1];");
    strcpy(buf[12], "    memset(buf, 0, size + 1); //Initiate every bit of buf as
0");
    strcpy(buf[13], "    fread(buf, 1, size, fp);");
    strcpy(buf[14], "    buf[size] = '\\0';");
    strcpy(buf[15], "    fclose(fp);                // close 'fp' stream.");
    strcpy(buf[16], "    return buf;");
    strcpy(buf[17], "    while (flag != 0){");
    strcpy(buf[18], "        ways = ways + ways * 12;");

```

```
        strcpy(buf[19], "    }");
        strcpy(buf[20], "    for (int a = 10; a >= 0; a--){");
        strcpy(buf[21], "                float tmp_value = 20.5;");
        strcpy(buf[22], "                printf(\"the real value of variable tmp_value
is:%f\\", tmp_value);");
        strcpy(buf[23], "                continue;");
        strcpy(buf[24], "    }");
        strcpy(buf[25], "});
        strcpy(buf[26], "// This is the Example for highlight of C – Language ");
        for(i = 0; i <= 26; i++){
            text[i] = malloc(MAX_LINE_LENGTH);
            strcpy(text[i], buf[i]);
        }
        line_number = get_line_number(text);
        show_text_syntax_highlighting(text);
        changed = 1;
    }
void show_text_syntax_highlighting(char *text[]){
    printf(1, ">>> \\033[1m\\e[45;33mthe contents of the file are:\\e[0m\\n");
    printf(1, "\\n");
    int j = 0;
    for (; text[j] != NULL; j++){
        printf(1, "\\e[1;30m%d%d%d\\e[0m\\e[0;32m|\\e[0m", (j+1)/100,
((j+1)%100)/10, (j+1)%10);
        int pos = 0;
        int a;
        for(a = 0; a < MAX_LINE_LENGTH; a++){
            if(text[j][a] != ' '){
                pos = a;
                break;
            }
        }

        if(strcmp(text[j], "\\n") == 0){
            printf(1, "\\n");
        }
    }
}
```

```
}
else if(text[j][pos] == '/' && text[j][pos+1] == '/'){
    printf(1, "\e[1;32m%s\n\e[0m", text[j]);
}
else{
    int mark = 0;
    int flag_annotation = 0;
    while(mark < MAX_LINE_LENGTH && text[j][mark] != NULL){
        // do something with 'mark' and print all the statements
        // by the way of one letter by one letter
        // judge annotation
        if(flag_annotation){
            printf(1, "\e[1;32m%c\e[0m", text[j][mark++]);
            //mark++;
            continue;
        }

        // numbers
        if(text[j][mark] >= '0' && text[j][mark] <= '9'){
            printf(1, "\033[0;33m%c\033[0m", text[j][mark]);
            mark++;
        }
        // printf
        else if((mark+5)<MAX_LINE_LENGTH && text[j][mark] ==
'p' && text[j][mark+1] == 'r'
&& text[j][mark+2] == 'i' && text[j][mark+3] == 'n'
&& text[j][mark+4] == 't'
&& text[j][mark+5] == 'f'){
            printf(1, "\e[1;36m%c\e[0m", text[j][mark]);
            printf(1, "\e[1;36m%c\e[0m", text[j][mark+1]);
            printf(1, "\e[1;36m%c\e[0m", text[j][mark+2]);
            printf(1, "\e[1;36m%c\e[0m", text[j][mark+3]);
            printf(1, "\e[1;36m%c\e[0m", text[j][mark+4]);
            printf(1, "\e[1;36m%c\e[0m", text[j][mark+5]);
            mark = mark + 6;
        }
    }
}
```



```
    }
    // int
    else if((mark+2)<MAX_LINE_LENGTH && text[j][mark] ==
'i' && text[j][mark+1] == 'n'
        && text[j][mark+2] == 't'){
        // highlighting 'int' string
        printf(1, "\e[1;34m%c\e[0m", text[j][mark]);
        printf(1, "\e[1;34m%c\e[0m", text[j][mark+1]);
        printf(1, "\e[1;34m%c\e[0m", text[j][mark+2]);
        mark = mark + 3;
    }
    // float
    else if((mark+4)<MAX_LINE_LENGTH && text[j][mark] ==
'f' && text[j][mark+1] == 'l' && text[j][mark+2] == 'o'
        && text[j][mark+3] == 'a' && text[j][mark+4] ==
't'){
        printf(1, "\e[1;34m%c\e[0m", text[j][mark]);
        printf(1, "\e[1;34m%c\e[0m", text[j][mark+1]);
        printf(1, "\e[1;34m%c\e[0m", text[j][mark+2]);
        printf(1, "\e[1;34m%c\e[0m", text[j][mark+3]);
        printf(1, "\e[1;34m%c\e[0m", text[j][mark+4]);
        mark = mark + 5;
    }
    // double
    else if((mark+5)<MAX_LINE_LENGTH && text[j][mark] ==
'd' && text[j][mark+1] == 'o' && text[j][mark+2] == 'u'
        && text[j][mark+3] == 'b' && text[j][mark+4] == 'l'
&& text[j][mark+5] == 'e'){
        printf(1, "\e[1;34m%c\e[0m", text[j][mark]);
        printf(1, "\e[1;34m%c\e[0m", text[j][mark+1]);
        printf(1, "\e[1;34m%c\e[0m", text[j][mark+2]);
        printf(1, "\e[1;34m%c\e[0m", text[j][mark+3]);
        printf(1, "\e[1;34m%c\e[0m", text[j][mark+4]);
        printf(1, "\e[1;34m%c\e[0m", text[j][mark+5]);
        mark = mark + 6;
```

```
    }
    // char
    else if((mark+3)<MAX_LINE_LENGTH && text[j][mark] ==
'c' && text[j][mark+1] == 'h' && text[j][mark+2] == 'a'
&& text[j][mark+3] == 'r'){
        printf(1, "\e[1;34m%c\e[0m", text[j][mark]);
        printf(1, "\e[1;34m%c\e[0m", text[j][mark+1]);
        printf(1, "\e[1;34m%c\e[0m", text[j][mark+2]);
        printf(1, "\e[1;34m%c\e[0m", text[j][mark+3]);
        mark = mark + 4;
    }
    // if
    else if((mark+1)<MAX_LINE_LENGTH && text[j][mark] ==
'i' && text[j][mark+1] == 'f'){
        printf(1, "\e[1;35m%c\e[0m", text[j][mark]);
        printf(1, "\e[1;35m%c\e[0m", text[j][mark+1]);
        mark = mark + 2;
    }
    // else
    else if((mark+3)<MAX_LINE_LENGTH && text[j][mark] ==
'e' && text[j][mark+1] == 'l' && text[j][mark+2] == 's'
&& text[j][mark+3] == 'e'){
        printf(1, "\e[1;35m%c\e[0m", text[j][mark]);
        printf(1, "\e[1;35m%c\e[0m", text[j][mark+1]);
        printf(1, "\e[1;35m%c\e[0m", text[j][mark+2]);
        printf(1, "\e[1;35m%c\e[0m", text[j][mark+3]);
        mark = mark + 4;
    }
    // else if
    else if((mark+5)<MAX_LINE_LENGTH && text[j][mark] ==
'e' && text[j][mark+1] == 'l' && text[j][mark+2] == 's'
&& text[j][mark+3] == 'e' && text[j][mark+4] == ' '
&& text[j][mark+5] == 'i' && text[j][mark+6] == 'f'){
        printf(1, "\e[1;35m%c\e[0m", text[j][mark]);
        printf(1, "\e[1;35m%c\e[0m", text[j][mark+1]);
```

```
printf(1, "\\e[1;35m%c\\e[0m", text[j][mark+2]);
printf(1, "\\e[1;35m%c\\e[0m", text[j][mark+3]);
printf(1, "\\e[1;35m%c\\e[0m", text[j][mark+4]);
printf(1, "\\e[1;35m%c\\e[0m", text[j][mark+5]);
printf(1, "\\e[1;35m%c\\e[0m", text[j][mark+6]);
mark = mark + 7;
}
else if((mark+2)<MAX_LINE_LENGTH && text[j][mark] ==
'f' && text[j][mark+1] == 'o' && text[j][mark+2] == 'r'){
    // highlighting 'int' string
    printf(1, "\\e[1;35m%c\\e[0m", text[j][mark]);
    printf(1, "\\e[1;35m%c\\e[0m", text[j][mark+1]);
    printf(1, "\\e[1;35m%c\\e[0m", text[j][mark+2]);
    mark = mark + 3;
}
else if((mark+4)<MAX_LINE_LENGTH && text[j][mark] ==
'w' && text[j][mark+1] == 'h' && text[j][mark+2] == 'i'
&& text[j][mark+3] == 'l' && text[j][mark+4] ==
'e'){
    printf(1, "\\e[1;35m%c\\e[0m", text[j][mark]);
    printf(1, "\\e[1;35m%c\\e[0m", text[j][mark+1]);
    printf(1, "\\e[1;35m%c\\e[0m", text[j][mark+2]);
    printf(1, "\\e[1;35m%c\\e[0m", text[j][mark+3]);
    printf(1, "\\e[1;35m%c\\e[0m", text[j][mark+4]);
    mark = mark + 5;
}
// long
else if((mark+3)<MAX_LINE_LENGTH && text[j][mark] ==
'l' && text[j][mark+1] == 'o' && text[j][mark+2] == 'n'
&& text[j][mark+3] == 'g'){
    printf(1, "\\e[1;34m%c\\e[0m", text[j][mark]);
    printf(1, "\\e[1;34m%c\\e[0m", text[j][mark+1]);
    printf(1, "\\e[1;34m%c\\e[0m", text[j][mark+2]);
    printf(1, "\\e[1;34m%c\\e[0m", text[j][mark+3]);
    mark = mark + 4;
```

```
    }
    // {}[]()
    else if(text[j][mark] == '{' || text[j][mark] == '}' ||
text[j][mark] == '[' || text[j][mark] == ']'
        || text[j][mark] == '(' || text[j][mark] == ')'){
        printf(1, "\e[1;35m%c\e[0m", text[j][mark]);
        mark++;
    }
    // static
    else if((mark+5)<MAX_LINE_LENGTH && text[j][mark] ==
's' && text[j][mark+1] == 't' && text[j][mark+2] == 'a'
        && text[j][mark+3] == 't' && text[j][mark+4] == 'i'
&& text[j][mark+5] == 'c'){
        printf(1, "\e[1;34m%c\e[0m", text[j][mark]);
        printf(1, "\e[1;34m%c\e[0m", text[j][mark+1]);
        printf(1, "\e[1;34m%c\e[0m", text[j][mark+2]);
        printf(1, "\e[1;34m%c\e[0m", text[j][mark+3]);
        printf(1, "\e[1;34m%c\e[0m", text[j][mark+4]);
        printf(1, "\e[1;34m%c\e[0m", text[j][mark+5]);
        mark = mark + 6;
    }
    // const
    else if((mark+4)<MAX_LINE_LENGTH && text[j][mark] ==
'c' && text[j][mark+1] == 'o' && text[j][mark+2] == 'n'
        && text[j][mark+3] == 's' && text[j][mark+4] ==
't'){
        printf(1, "\e[1;34m%c\e[0m", text[j][mark]);
        printf(1, "\e[1;34m%c\e[0m", text[j][mark+1]);
        printf(1, "\e[1;34m%c\e[0m", text[j][mark+2]);
        printf(1, "\e[1;34m%c\e[0m", text[j][mark+3]);
        printf(1, "\e[1;34m%c\e[0m", text[j][mark+4]);
        mark = mark + 5;
    }
    // memset
```

```
else if((mark+5)<MAX_LINE_LENGTH && text[j][mark] ==
'm' && text[j][mark+1] == 'e' && text[j][mark+2] == 'm'
&& text[j][mark+3] == 's' && text[j][mark+4] == 'e'
&& text[j][mark+5] == 't'){
    printf(1, "\\e[1;36m%c\\e[0m", text[j][mark]);
    printf(1, "\\e[1;36m%c\\e[0m", text[j][mark+1]);
    printf(1, "\\e[1;36m%c\\e[0m", text[j][mark+2]);
    printf(1, "\\e[1;36m%c\\e[0m", text[j][mark+3]);
    printf(1, "\\e[1;36m%c\\e[0m", text[j][mark+4]);
    printf(1, "\\e[1;36m%c\\e[0m", text[j][mark+5]);
    mark = mark + 6;
}
// //
else if((mark+1)<MAX_LINE_LENGTH && text[j][mark] ==
'/' && text[j][mark] == '/'){
    printf(1, "\\e[1;32m%c\\e[0m", text[j][mark]);
    printf(1, "\\e[1;32m%c\\e[0m", text[j][mark+1]);
    mark = mark + 2;
    flag_annotation = 1;
}
// NULL
else if((mark+3)<MAX_LINE_LENGTH && text[j][mark] ==
'N' && text[j][mark+1] == 'U' && text[j][mark+2] == 'L'
&& text[j][mark+3] == 'L'){
    printf(1, "\\e[1;35m%c\\e[0m", text[j][mark]);
    printf(1, "\\e[1;35m%c\\e[0m", text[j][mark+1]);
    printf(1, "\\e[1;35m%c\\e[0m", text[j][mark+2]);
    printf(1, "\\e[1;35m%c\\e[0m", text[j][mark+3]);
    mark = mark + 4;
}
// character string
else if(text[j][mark] == ""){
    int tmp_pos = mark+1;
    int end = -1;
    while(tmp_pos < MAX_LINE_LENGTH){
```

```
        if(text[j][tmp_pos] == ""){
            end = tmp_pos;
            break;
        }
        else{
            tmp_pos++;
        }
    }
    int inter;
    for(inter = mark; inter <= end; inter++){
        printf(1, "\e[1;33m%c\e[0m", text[j][inter]);
    }
    mark = end + 1;
}
// single character
else if(text[j][mark] == '\\'){
    int tmp_pos = mark+1;
    int end = -1;
    while(tmp_pos < MAX_LINE_LENGTH){
        if(text[j][tmp_pos] == '\\'){
            end = tmp_pos;
            break;
        }
        else{
            tmp_pos++;
        }
    }
    int inter;
    for(inter = mark; inter <= end; inter++){
        printf(1, "\e[1;33m%c\e[0m", text[j][inter]);
    }
    mark = end + 1;
}
// continue
```

```
        else if((mark+5)<MAX_LINE_LENGTH && text[j][mark] ==
'c' && text[j][mark+1] == 'o' && text[j][mark+2] == 'n'
        && text[j][mark+3] == 't' && text[j][mark+4] == 'i'
&& text[j][mark+5] == 'n' && text[j][mark+6] == 'u'
        && text[j][mark+7] == 'e'){
            printf(1, "\\e[1;35m%c\\e[0m", text[j][mark]);
            printf(1, "\\e[1;35m%c\\e[0m", text[j][mark+1]);
            printf(1, "\\e[1;35m%c\\e[0m", text[j][mark+2]);
            printf(1, "\\e[1;35m%c\\e[0m", text[j][mark+3]);
            printf(1, "\\e[1;35m%c\\e[0m", text[j][mark+4]);
            printf(1, "\\e[1;35m%c\\e[0m", text[j][mark+5]);
            printf(1, "\\e[1;35m%c\\e[0m", text[j][mark+6]);
            printf(1, "\\e[1;35m%c\\e[0m", text[j][mark+7]);
            mark = mark + 8;
        }
        // return
        else if((mark+5)<MAX_LINE_LENGTH && text[j][mark] ==
'r' && text[j][mark+1] == 'e' && text[j][mark+2] == 't'
        && text[j][mark+3] == 'u' && text[j][mark+4] == 'r'
&& text[j][mark+5] == 'n'){
            printf(1, "\\e[1;34m%c\\e[0m", text[j][mark]);
            printf(1, "\\e[1;34m%c\\e[0m", text[j][mark+1]);
            printf(1, "\\e[1;34m%c\\e[0m", text[j][mark+2]);
            printf(1, "\\e[1;34m%c\\e[0m", text[j][mark+3]);
            printf(1, "\\e[1;34m%c\\e[0m", text[j][mark+4]);
            printf(1, "\\e[1;34m%c\\e[0m", text[j][mark+5]);
            mark = mark + 6;
        }
        else{
            printf(1, "%c\\e[0m", text[j][mark]);
            mark++;
        }
    }
    printf(1, "\\n");
}
```

```
    }
    printf(1, "\n");
}
void com_rollback(char *text[], int n){
    // rollback the command
    if(upper_bound < 0){
        printf(1, ">>> \033[1m\e[41;33mcouldn't rollback\e[0m\n");
        return;
    }
    char *input = malloc(MAX_LINE_LENGTH);
    strcpy(input, command_set[upper_bound]);
    upper_bound--;
    // searching the first space of command
    int pos = MAX_LINE_LENGTH - 1;
    int j = 0;
    for (; j < 10; j++) {
        if (input[j] == ' ')
        {
            pos = j + 1;
            break;
        }
    }
    // deal 'ins' command
    if (input[0] == 'i' && input[1] == 'n' && input[2] == 's')
    {
        // the line to be deleted
        int line = stringtonumber(&input[4]);
        com_del(text, line, 0);
        line_number = get_line_number(text);
    }
    // deal 'mod' command
    else if (input[0] == 'm' && input[1] == 'o' && input[2] == 'd')
    {
        // the line to be modified
        int line = stringtonumber(&input[4]);
```



```
        // the content of mod
        char *content = &input[pos];
        com_mod(text, line, content, 0);
        line_number = get_line_number(text);
    }
    // deal 'del' command
    else if (input[0] == 'd' && input[1] == 'e' && input[2] == 'l')
    {
        // the line to be deleted
        int line = stringtonumber(&input[4]);
        // the content of deletion
        char *content = &input[pos];
        com_ins(text, line, content, 0);
        line_number = get_line_number(text);
    }
}

void record_command(char *command){
    if((upper_bound+1) < MAX_ROLLBAKC_STEP){
        command_set[upper_bound+1] = malloc(MAX_LINE_LENGTH);
        strcpy(command_set[upper_bound+1], command);
        upper_bound++;
    }
    else{
        int i;
        for(i = 1; i < MAX_ROLLBAKC_STEP; i++){
            strcpy(command_set[i-1], command_set[i]);
        }
        strcpy(command_set[upper_bound], command);
        upper_bound = MAX_ROLLBAKC_STEP - 1;
    }
}
```

OUTPUT:

osd-190031187@team-osd:~/190031187-xv6

```

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
190031187$ editor sample
>>> ins
... input content:hello sample
>>> the contents of the file are:

001|hello sample
002|

>>> help
>>> instructions for use:
-----
ins-n: | insert a line after line n
mod-n: | modify line n
del-n: | delete line n
ins:   | insert a line after the last line
mod:   | modify the last line
del:   | delete the last line
show:  | enable show current contents after executing a command.
hide:  | disable show current contents after executing a command.
save:  | save the file
exit:  | exit editor
help:  | help info
demo:  | color demo
init:  | initial file
disp:  | display with highlighting
rb:    | rollback the file
-----
>>> del-1
>>> the contents of the file are:

001|

>>> disp
>>> the contents of the file are:

001|

>>> init
>>> the contents of the file are:

001|// Create a NULL-terminated string by reading the provided file
002|static char* readShaderSource(const char* shaderFile)
003|{
004|    int flag = 24;
005|    double ways = 100.43;
006|    if ( fp == NULL ) {
007|        return NULL;
008|    }
009|    fseek(fp, 0L, SEEK_END);           //search something
010|    long size = ftell(fp);
011|    fseek(fp, 0L, SEEK_SET);
012|    char* buf = new char[size + 1];
013|    memset(buf, 0, size + 1);         //Initiate every bit of buf as 0
014|    fread(buf, 1, size, fp);
015|    buf[size] = '\0';
016|    fclose(fp);                       // close 'fp' stream.
017|    return buf;
018|    while (flag != 0){
019|        ways = ways + ways * 12;
020|    }
021|    for (int a = 10; a >= 0; a--){
022|        float tmp_value = 20.5;
023|        printf("the real value of variable tmp_value is:%f", tmp_value);
024|        continue;
025|    }
026|}
027|// This is the Example for highlight of C - Language

>>> exit
>>> save the file? y/n
y

>>> saved successfully
190031187$ 190031187$

```

Observation and Analysis:

Here a file **sample** is created by using the editor command. After execution, the editor also asks whether to **save** the file or not and exits the editor by command **exit**. We can also try the **ins-n** command which takes a parameter called line from user to insert or delete in a particular line. If you need any command description **help** command displays all the commands along with their description. **disp** command displays the content in the file. It also highlights c language.

Is.c Code | Task-3:

```
/*ls.c*/
#include "types.h"
#include "stat.h"
#include "user.h"
#include "fs.h"

char*
fmtname(char *path)
{
    static char buf[DIRSIZ+1];
    char *p;

    // Find first character after last slash.
    for(p=path+strlen(path); p >= path && *p != '/'; p--)
        ;
    p++;

    // Return blank-padded name.
    if(strlen(p) >= DIRSIZ)
        return p;
    memmove(buf, p, strlen(p));
    memset(buf+strlen(p), ' ', DIRSIZ-strlen(p));
    return buf;
}

void
ls(char *path)
{
    char buf[512], *p;
    int fd;
    struct dirent de;
    struct stat st;

    if((fd = open(path, 0)) < 0){
        printf(2, "ls: cannot open %s\n", path);
        return;
    }

    if(fstat(fd, &st) < 0){
```

```
    printf(2, "ls: cannot stat %s\n", path);
    close(fd);
    return;
}

switch(st.type){
case T_FILE:
    printf(1, "%s %d %d %d\n", fmtname(path), st.type, st.ino, st.size);
    break;

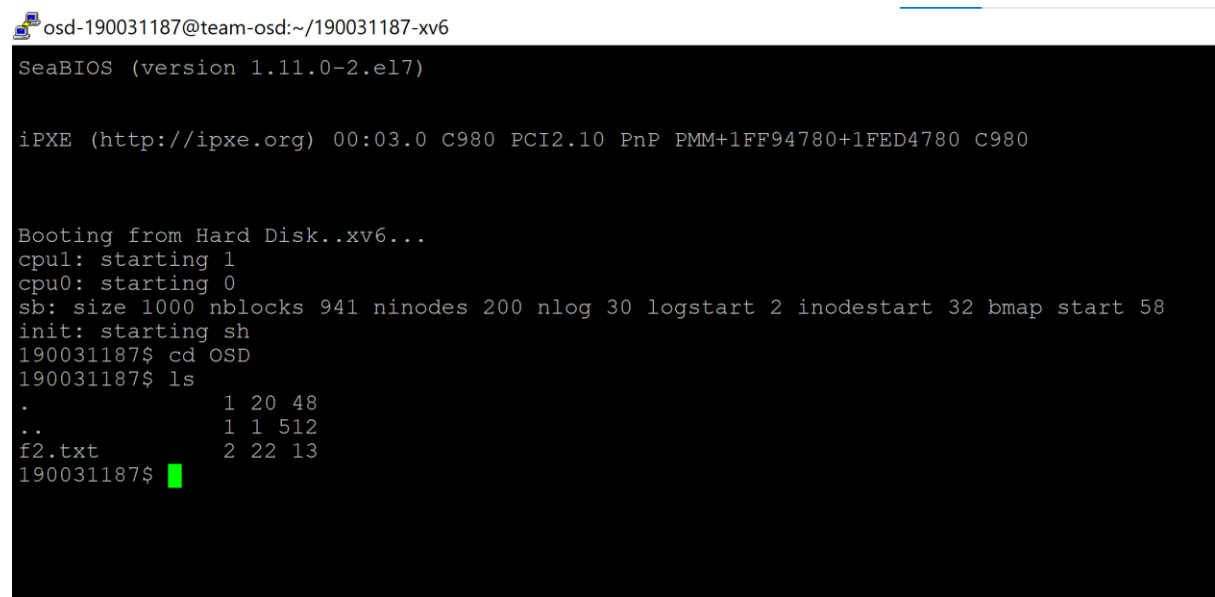
case T_DIR:
    if(strlen(path) + 1 + DIRSIZ + 1 > sizeof buf){
        printf(1, "ls: path too long\n");
        break;
    }
    strcpy(buf, path);
    p = buf+strlen(buf);
    *p++ = '/';
    while(read(fd, &de, sizeof(de)) == sizeof(de)){
        if(de.inum == 0)
            continue;
        memmove(p, de.name, DIRSIZ);
        p[DIRSIZ] = 0;
        if(stat(buf, &st) < 0){
            printf(1, "ls: cannot stat %s\n", buf);
            continue;
        }
        printf(1, "%s %d %d %d\n", fmtname(buf), st.type, st.ino, st.size);
    }
    break;
}
close(fd);
}

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

    if(argc < 2){
```

```
ls(".");  
exit();  
}  
for(i=1; i<argc; i++)  
    ls(argv[i]);  
exit();  
}
```

OUTPUT:



```
osd-190031187@team-osd:~/190031187-xv6  
SeaBIOS (version 1.11.0-2.el7)  
  
iPXE (http://ipxe.org) 00:03.0 C980 PCI2.10 PnP PMM+1FF94780+1FED4780 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 start 58  
init: starting sh  
190031187$ cd OSD  
190031187$ ls  
.  
..  
f2.txt  
190031187$
```

Observation and Analysis:

Here we observe the ls command working by creating a folder/directory by using 'mkdir' command and creating a sample.txt file in the directory and use ls to list the text file.

Porting xv6 with POSIX compliance + VFS + ulibc + ACPI | Task-4:

1. First clone the repository from GitHub to your local repository.

-> git clone https://github.com/NewbiZ/xv6

2. Open Makefile and make following changes

line 61; QEMU = qemu-system-i386

line 62; qemu: distrib/distrib.img disk.img

line 63; \$(LOG_CMD) \$(QEMU) -nographic -serial mon:stdio -hdb distrib/distrib.img
disk.img -smp 2 -m 512 \$(QEMUEXTRA)

3. Type make qemu-nox

OUTPUT

osd-190031187@team-osd:~/190031187-xv6/xv6

```
SeaBIOS (version 1.11.0-2.el7)

iPXE (http://ipxe.org) 00:03.0 C980 PCI2.10 PnP PMM+1FF94780+1FED4780 C980

Booting from Hard Disk...
xv6...
cpu1: starting
cpu0: starting
[init] executing rc.local

Welcome to Xv6!

[init] starting shell
$ ls
.          1 1 512
..         1 1 512
bin        1 2 560
boot       1 3 32
dev        1 4 112
etc        1 5 64
home       1 6 32
lib        1 7 32
media      1 8 32
mnt        1 9 32
opt        1 10 32
root       1 11 32
sbin       1 12 64
srv        1 13 32
tmp        1 14 32
usr        1 15 112
var        1 16 176
$
```

Mirrored Raid Feature | Task 5:

4. First clone the repository from GitHub to your local repository.
-> git clone https://github.com/aditvenk/xv6_file_system
5. Now change the directory to the created or clone directory. Now we have to modify the code in the **Makefile**. In **Makefile**, from lines 54 to 72 **replace the old code with new code**.

If the makefile can't find QEMU, specify its path here

#QEMU :

Try to infer the correct QEMU if not specified

ifndef QEMU

QEMU := \$(shell if which qemu 1> /dev/null 2> /dev/null; \

then echo qemu; exit; \

else \

qemu=/u/c/s/cs537-2/ta/tools/qemu; \

if test -x \$\$qemu; then echo \$\$qemu; exit; fi; \

echo "****" 1>&2; \

echo "**** Error: Couldn't find a working QEMU executable." 1>&2; \

echo "**** Is the directory containing the qemu binary in your " 1>&2; \

echo "**** PATH or have you tried setting the QEMU variable in " 1>&2; \

echo "**** Makefile?" 1>&2; \

echo "****" 1>&2; exit 1)

endif

//Add the following code in place of above code

QEMU = qemu-system-i386

Try to infer the correct QEMU

ifndef QEMU

QEMU = \$(shell if which qemu > /dev/null; \

then echo qemu; exit; \

elif which qemu-system-i386 > /dev/null; \

then echo qemu-system-i386; exit; \

elif which qemu-system-x86_64 > /dev/null; \

then echo qemu-system-x86_64; exit; \

else \


```
qemu=/Applications/Q.app/Contents/MacOS/i386-softmmu.app/Contents/MacOS/i386-softmmu; \
if test -x $$qemu; then echo $$qemu; exit; fi; \
echo "****" 1>&2; \
echo "**** Error: Couldn't find a working QEMU executable." 1>&2; \
echo "**** Is the directory containing the qemu binary in your PATH" 1>&2; \
echo "**** or have you tried setting the QEMU variable in Makefile?" 1>&2; \
echo "****" 1>&2; exit 1)
endif
```

6. Now go to user directory. And create a new file named, **tester.c**. And write the **below code** in it.

```
// Do not modify this file. It will be replaced by the grading scripts
// when checking your project.
```

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

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

    printf(1, "mirrored fs test \n");
    /*
    int fd;
    fd = open("echo", O_CREATE);
    if (fd < 0)
        exit();
    close(fd);
    */

    /*
    struct stat st;
    if (stat("echo", &st) < 0) {
```

```
    printf(1, "stat failure" );
    exit();
}
printf(1, "stat output: \n");
printf(1, "type = %d, dev = %d, inode-num = %d, nlink = %d, size = %d \n",
st.type, st.dev, st.ino, st.nlink, st.size);
printf(1, "logical_size = %d, physical_size = %d \n", st.logical_size,
st.physical_size);
*/
```

```
int fd;
char buf[128];
fd = open ( "temp.txt", O_CREATE | O_MIRRORED | O_RDWR);
if (fd < 0) {
    printf(1, "open failed \n");
    exit();
}
int j=0;
char temp[512];
for (j=0; j<512; j++){
    temp[j] = 'x';
}
for (j=0;j<7;j++) {
    if (write(fd, temp, 512) != 512) {
        printf(1, "Write failed \n");
        exit();
    }
}
printf(1, "write done\n");
close(fd);
fd = open ( "temp.txt", O_MIRRORED | O_RDONLY);
if (fd < 0)
    exit();
```

```
int i = read(fd, buf, 10);
```

```
if (i != 10) {
    printf(1, "read failed \n");
    exit();
}
```

```
printf(1, "read text = %s ", buf);
close(fd);
//unlink("temp.txt");

exit();
}
```

7. create another file named, **tester.c**. And write the **below code** in it.

```
#include "types.h"
#include "stat.h"
#include "user.h"
#include "fcntl.h"
#include "fs.h"
int ppid;
#define assert(x) if (x) {} else {\
    printf(1, "%s: %d ", __FILE__, __LINE__); \
    printf(1, "assert failed (%s)\n", # x); \
    printf(1, "TEST FAILED\n"); \
    kill(ppid); \
    exit(); \
}

Int main(int argc, char *argv[])
{
    ppid = getpid();
    int fd;
    int size = 512;
    int n = (MAXFILE * BSIZE) / size / 2;
    int i, j;
    char buf[size];
    int r;

    //printf(1, "buffer size: %d\n", size);
    //printf(1, "file size: %d\n", n * size);

    //printf(1, "create mirrored file\n");
    fd = open("out", O_CREATE | O_MIRRORED | O_RDWR);
```


```
assert(fd >= 0);

memset(buf, 0, size);

//printf(1, "writing file\n");
for (i = 0; i < n; i++) {
    buf[0] = (char)('A' + i);
    printf(1, "Writing %s \n", buf);
    r = write(fd, buf, size);
    assert(r == size);
}
//printf(1, "reopening read only\n");
r = close(fd);
assert(r == 0);
fd = open("out", O_RDONLY);
assert(fd >= 0);

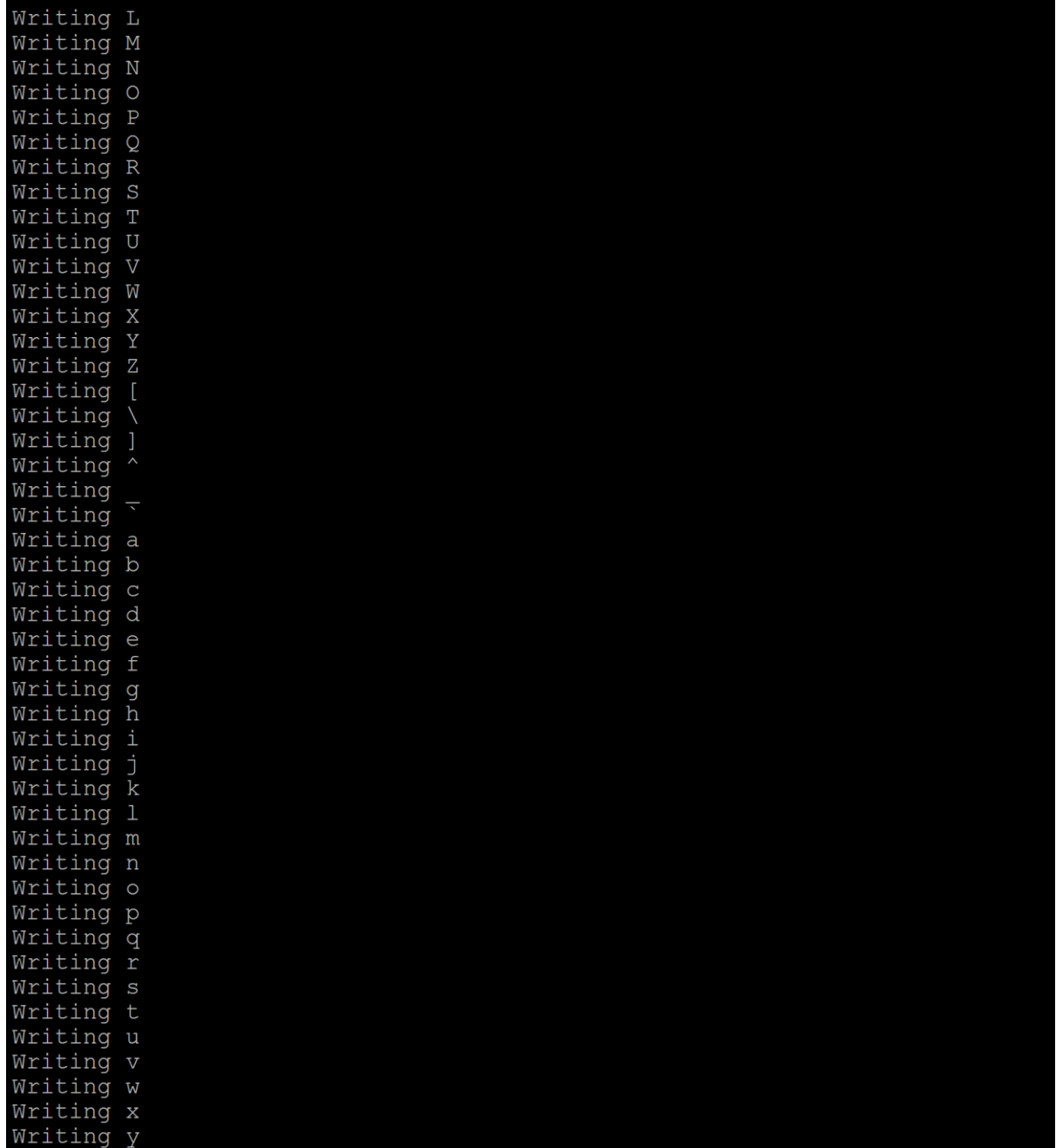
//printf(1, "reading file\n");
for (i = 0; i < n; i++) {
    r = read(fd, buf, size);
    assert(r == size);
    printf(1, "Reading %s \n", buf);
    assert(buf[0] == (char)('A' + i));
    for (j = 1; j < size; j++) {
        assert(buf[j] == 0);
    }
}
printf(1, "TEST PASSED\n");
exit();
}
```

8. Type make qemu-nox

OUTPUT: osd-190031187@team-osd:~/190031187-xv6/task5

```
Booting from Hard Disk..xv6...
lapicinit: 1 0xf000000
cpu1: starting
cpu0: starting
init: starting sh
190031187$ tester
mirrored fs test
write done
read text = xxxxxxxxxxx 190031187$
190031187$ write2
Writing A
Writing B
Writing C
Writing D
Writing E
Writing F
Writing G
Writing H
Writing I
Writing J
Writing K
Writing L
Writing M
Writing N
Writing O
Writing P
Writing Q
Writing R
Writing S
Writing T
Writing U
Writing V
Writing W
Writing X
Writing Y
Writing Z
Writing [
Writing \
Writing ]
Writing ^
Writing `
Writing a
Writing b
Writing c
```

Writing d
Writing e
Writing f
Writing g
Writing h
Writing i
Writing j
Writing k
Writing l
Writing m
Writing n
Writing o
Writing p
Writing q
Writing r
Writing s
Writing t
Writing u
Writing v
Writing w
Writing x
Writing y
Writing z
Writing {
Writing |
Writing }
Writing ~
Writing
Writing
Writing
Writing
Writing
Writing
Writing
Reading A
Reading B
Reading C
Reading D
Reading E
Reading F
Reading G
Reading H
Reading I
Reading J
Reading K

A large black rectangular area representing a mirrored RAID test result. The text 'Writing L' through 'Writing y' is visible on the left side of the black area, indicating the input data. The mirrored data is not visible within the black area.

Writing L
Writing M
Writing N
Writing O
Writing P
Writing Q
Writing R
Writing S
Writing T
Writing U
Writing V
Writing W
Writing X
Writing Y
Writing Z
Writing [
Writing \
Writing]
Writing ^
Writing \wedge
Writing a
Writing b
Writing c
Writing d
Writing e
Writing f
Writing g
Writing h
Writing i
Writing j
Writing k
Writing l
Writing m
Writing n
Writing o
Writing p
Writing q
Writing r
Writing s
Writing t
Writing u
Writing v
Writing w
Writing x
Writing y

Observation and Analysis:

Here the first test writes a series of X of form XXXXXXXX and the same is read below. Hence a mirrored raid feature. Next, we write all the alphabets and symbols, and the mirrored raid feature reads all the written alphabets and symbols in same order. Hence both the above tests prove our Mirrored Raid feature by performing tests.

CONCLUSION

I successfully created a basic XV6 shell with what I believe to be necessary for a common usage. I learnt a lot from working with a basic Operating System and would like to thank everyone for this opportunity. The journey to modifying the XV6 and implementing my own shell was a very interesting and eventful one and even though sometimes, my code was like a shot in the dark, I believe that I achieved what I wanted to in the end.