IMPLEMENTATION OF SHELL, EDITOR,HFILE SYSTEM CHECKER, **HEAD,ADDING A NEW SYSTEM CALL getprocs()** IN XV6

A SKILLING PROJECT REPORT

Submitted towards the professional course

20CS2103S Operating Systems Design

P.Manogyna Sai(2000030787)

BACHELOR OF TECHNOLOGY

Branch: Computer Science and Engineering

NOVEMBER 2021

Department of Computer Science and Engineering

K L Deemed to be University,

Green Fields, Vaddeswaram,

Guntur District, A.P., 522 502.

TABLE OF CONTENTS

WHAT IS XV6?

Xv6 is a teaching operating system developed in the summer of 2006 for MIT's operating systems course, 6.828: Operating System Engineering. We hope that xv6 will be useful in other courses too. This page collects resources to aid the use of xv6 in other courses, including a commentary on the source code itself.

History and Background

For many years, MIT had no operating systems course. In the fall of 2002, one was created to teach operating systems engineering. In the course lectures, the class worked through Sixth Edition Unix (aka V6) using John Lions's famous commentary. In the lab assignments, students wrote most of an exokernel operating system, eventually named Jos, for the Intel x86. Exposing students to multiple systems–V6 and Jos–helped develop a sense of the spectrum of operating system designs.

V6 presented pedagogic challenges from the start. Students doubted the relevance of an obsolete 30-year-old operating system written in an obsolete programming language (pre-K&R C) running on obsolete hardware (the PDP-11). Students also struggled to learn the low-level details of two different architectures (the PDP-11 and the Intel x86) at the same time. By the summer of 2006, we had decided to replace V6 with a new operating system, xv6, modeled on V6 but written in ANSI C and running on multiprocessor Intel x86 machines. Xv6's use of the x86 makes it more relevant to students' experience than V6 was and unifies the course around a single architecture. Adding multiprocessor support requires handling concurrency head on with locks and threads (instead of using special-case solutions for uniprocessors such as enabling/disabling interrupts) and helps relevance. Finally, writing a new system allowed us to write cleaner versions of the rougher parts of V6, like the scheduler and file system. 6.828 substituted xv6 for V6 in the fall of 2006.

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 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

Data Flow Models

Level 0 DFD

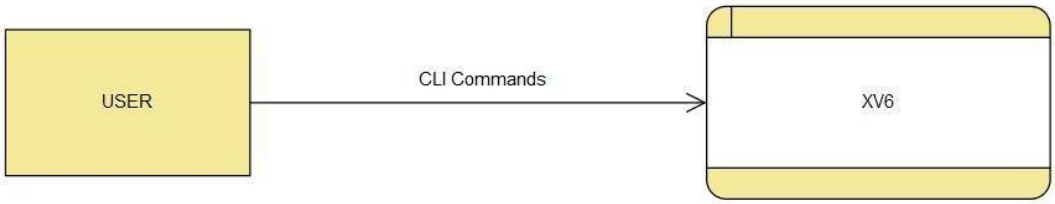


TABLE DESCRIPTIONS

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.

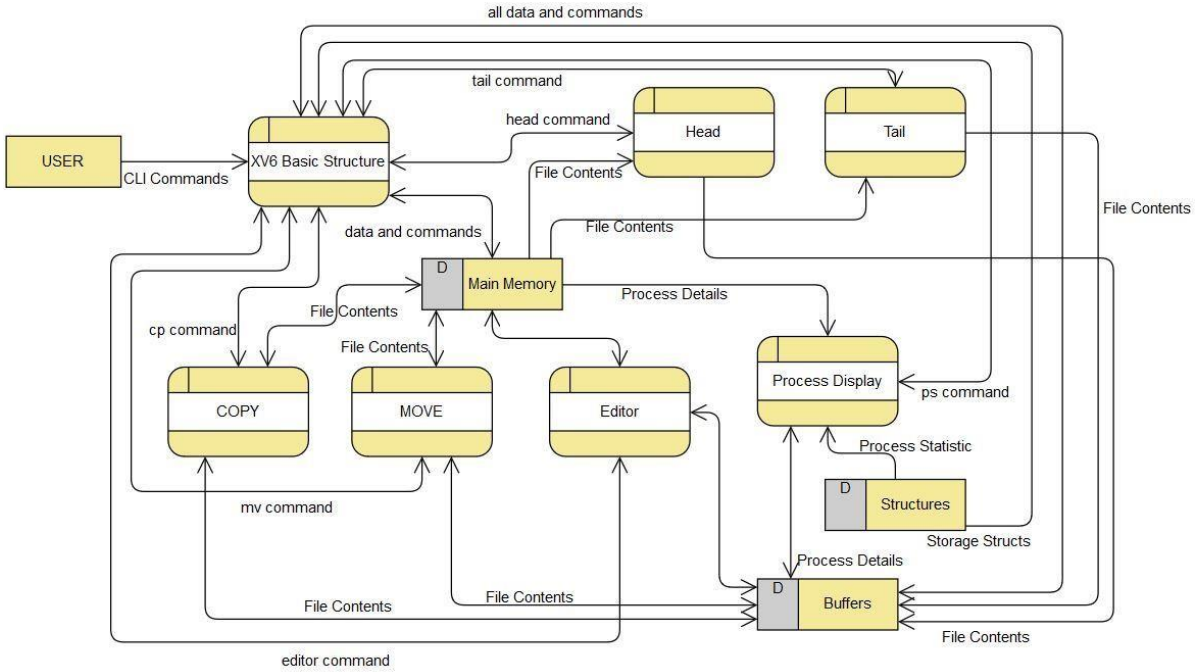
Buffers

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.

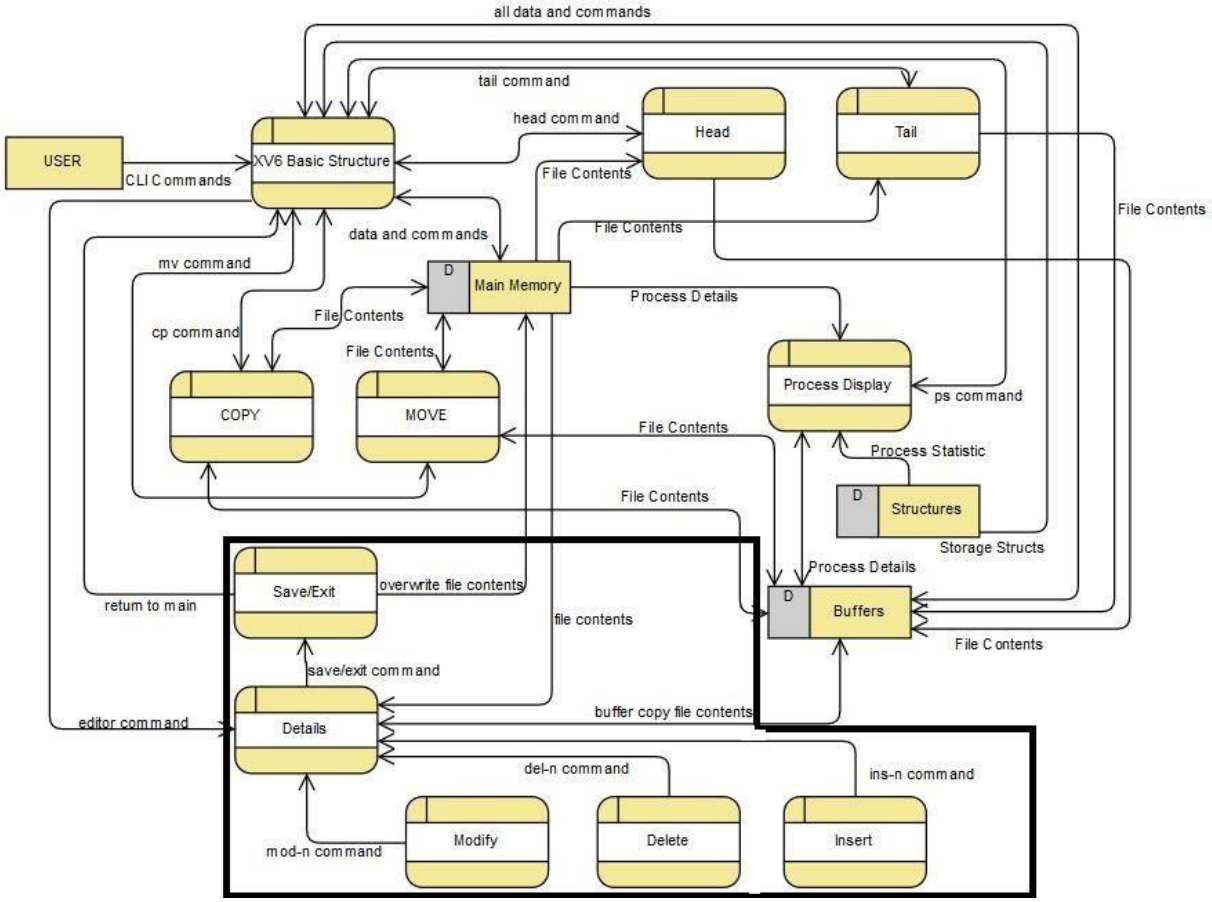
Structures

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


all data and commands
tail command
head command
USER
CLI Commands
XV6 Basic Structure
Head
Tail
File Contents
File Contents
File Contents
data and commands
D
Main Memory
Process Details
File Contents
cp command
File Contents
COPY
MOVE
Editor
Process Display
ps command
Process Statistic
D
Structures
Storage Structs
mv command
Process Details
D
Buffers
File Contents
File Contents
File Contents
editor command

Level 2 DFD


all data and commands
tail command
head command
Head
Tail
USER
CLI Commands
XV6 Basic Structure
File Contents
File Contents
data and commands
File Contents
mv command
D
Main Memory
Process Details
cp command
File Contents
File Contents
Process Display
ps command
COPY
MOVE
File Contents
Process Statistic
File Contents
D
Structures
Storage Structs
Process Details
Save/Exit
overwrite file contents
return to main
file contents
D
Buffers
File Contents
save/exit command
editor command
Details
buffer copy file contents
del-n command
ins-n command
mod-n command
Modify
Delete
Insert

MODULES DESCRIPTION

Copy

Syntax: cp file1 file2
Mandatory Parameters: file1, file2

This module is invoked with help of the command cp. this command copies the contents of file1 to file2. but internally what happens is that it reads the contents of file1 in a buffer and writes the content in file 2 from the buffer continuously until the end of file1, here the file2 is created if it doesn't exist else it is overwritten on the existing file specified. The contents of the file1 is unharmed. Since a copy of file1 is created, more space will be occupied in the memory Here all parameters are mandatory for the invocation of the module. This module is based on command line arguments where the inputs are passed as arguments to the module when it is invoked.

Move

Syntax: mv file1 file2
Mandatory Parameters: file1, file2

This module is invoked with help of the command mv. This command moves the contents of file1 to file2, but internally what happens is that it reads the contents of file1 in a buffer and writes the content in file 2 from the buffer continuously until the end of file1 and then in the end file1 is deleted from the memory, here the file2 is created if it doesn't exist else it is overwritten on the existing file specified. There will be no change in space of the memory since the file1 is deleted. Here all parameters are mandatory for the invocation of the module. This module is based on command line arguments where the inputs are passed as arguments to the module when it is invoked.

Head
Syntax: head file1 n

Mandatory Parameter: file1

This module is invoked when the command head is passed. This command prints the contents at the start of the file. Here the “n” parameter in the command specifies the number of lines to be printed from the start of file, by default it is taken as 10, it’s an optional parameter which means that the user doesn’t have to mention the value of “n” each time when the module has to be invoked. The internal working of the module is that it read the contents of the file1 through a buffer and then writes it to the terminal so that the user can read the first “n” lines if the value has been provided else the value 10 will be assigned for “n”. Here not all parameters are mandatory for the invocation of the module. This module is based on command line arguments where the inputs are passed as arguments to the module when it is invoked.

Tail
Syntax: tail file1 n

Mandatory Parameter: file1

This module is invoked when the command tail is passed. This command prints the contents at the end of the file. Here the “n” parameter in the command specifies the number of lines to be printed end of the file, by default it is taken as 10, it’s an optional parameter which means that the user doesn’t have to mention the value of “n” each time when the module has to be invoked. The internal working of the module is that it read the contents of the file1 through a buffer and then writes it to the terminal so that the user can read the last “n” lines if the value has been provided else the value 10 will be assigned for “n”. Here not all parameters are mandatory for the invocation of the module. This module is based on command line arguments where the inputs are passed as arguments to the module when it is invoked

Process display
Syntax: ps

Mandatory Parameters: None

This module is invoked when the command ps is passed. This is invoked when the user demands to see the currently running processes, Memory allocation to each process, total run time of a process, starting address of a process, size of a process, total CPU utilization of a process, id of a process, parent id for each process, current state of the process, name of the process and total no flags used by a process. The internal working of ps module is that it accesses the process structure and then it fetches the data it needs and then checks whether a process is "UNUSED" or not, if the former then the data is ignored and then the next set of data is fetched from the memory and checked else if it's the latter then the details mentioned above will be written to the output buffer and then onto the terminal. No parameters are required to invoke this module. This module is based on command line arguments where the inputs are passed as arguments to the module when it is invoked.

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.

CODES WITH OUTPUT

MODIFIED SH.C for cd and pwd

CODE

```
// 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;
```

```
117 char Point[] = "/" ;
118
119 if(cmd == 0)
120 exit();
121
122 switch(cmd->type){
123 default:
124 panic("runcmd");
125
126 case EXEC:
127 ecmd = (struct execcmd*)cmd;
128 if(ecmd->argv[0] == 0)
129 exit();
130 exec(strcat(Point,ecmd->argv[0]), ecmd->argv);
131 printf(2, "exec %s failed\n", ecmd->argv[0]);
132 break;
133
134 case REDIR:
135 rcmd = (struct redircmd*)cmd;
136 close(rcmd->fd);
137 if(open(rcmd->file, rcmd->mode) < 0){
138 printf(2, "open %s failed\n", rcmd->file);
139 exit();
140 }
141 runcmd(rcmd->cmd);
142 break;
143
144 case LIST:
145 lcmd = (struct listcmd*)cmd;
146 if(fork1() == 0)
147 runcmd(lcmd->left);
148 wait();
149 runcmd(lcmd->right);
150 break;
151
152 case PIPE:
153 pcmd = (struct pipecmd*)cmd;
154 if(pipe(p) < 0)
155 panic("pipe");
156 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;
for(int 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(int 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;
```

```
316 }
317 continue;
318 }
319 if (buf[3] == '/' )
320 {
321 Curr = Head_Directory;
322 Curr->Next = NULL;
323 prev = NULL;
324 Next = CreateNode(buf+4);
325 Curr->Next = Next;
326 Curr->Before = prev;
327 prev = Curr;
328 Next->Before = Curr;
329 Curr = Curr->Next;
330 continue;
331 }
332 if (Curr != Head_Directory && buf[3] != '/'){
333 Next = CreateNode("/");
334 Curr->Next = Next;
335 Curr->Before = prev;
336 prev = Curr;
337 Next->Before = Curr;
338 Curr = Curr->Next;
339 }
340 Next = CreateNode(buf+3);
341 Curr->Next = Next;
342 Curr->Before = prev;
343 prev = Curr;
344 Next->Before = Curr;
345 Curr = Curr->Next;
346 }
347 continue;
348 }
349 if(buf[0] == 'p' && buf[1] == 'w' && buf[2] == 'd')
350 {
351 struct directory *iter = Head_Directory;
352 while(iter)
353 {
354 printf(1,iter->string);
355 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;
```

```
435 return (struct cmd*)cmd;
436 }
437
438 struct cmd*
439 pipecmd(struct cmd *left, struct cmd *right)
440 {
441 struct pipecmd *cmd;
442
443 cmd = malloc(sizeof(*cmd));
444 memset(cmd, 0, sizeof(*cmd));
445 cmd->type = PIPE;
446 cmd->left = left;
447 cmd->right = right;
448 return (struct cmd*)cmd;
449 }
450
451 struct cmd*
452 listcmd(struct cmd *left, struct cmd *right)
453 {
454 struct listcmd *cmd;
455
456 cmd = malloc(sizeof(*cmd));
457 memset(cmd, 0, sizeof(*cmd));
458 cmd->type = LIST;
459 cmd->left = left;
460 cmd->right = right;
461 return (struct cmd*)cmd;
462 }
463
464 struct cmd*
465 backcmd(struct cmd *subcmd)
466 {
467 struct backcmd *cmd;
468
469 cmd = malloc(sizeof(*cmd));
470 memset(cmd, 0, sizeof(*cmd));
471 cmd->type = BACK;
472 cmd->cmd = subcmd;
473 return (struct cmd*)cmd;
474 }
```

```
//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;
}

```

OUTPUTS

osd-190031554@team-osd:~/190031554-xv6

```
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 27348
stressfs       2 14 14324
usertests      2 15 67224
wc             2 16 15144
zombie         2 17 13036
rename         2 18 23756
editor         2 19 26800
console        3 20 0
temp.pwd       2 21 1
abc            2 22 18
190031554$ mkdir sss
190031554$ cd sss
190031554$ mkdir ddd
190031554$ ls
.              1 23 48
..             1 1 512
ddd            1 24 32
190031554$
```

EDITOR

CODE

```c
#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 NULL 0

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);
void com_mod(char *text[], int n, char *extra);
void com_del(char *text[], int n);
void com_help(char *text[]);
void com_save(char *text[], char *path);
void com_exit(char *text[], char *path);
int stringtonumber(char* src);

int changed = 0;
int auto_show = 1;

int main(int argc, char *argv[])
{
    if (argc == 1)
    {
        printf(1, "please input the command as [editor file_name]\n");
        exit();
    }
```

```c

    char *text[MAX_LINE_NUMBER] = {};
    text[0] = malloc(MAX_LINE_LENGTH);
    memset(text[0], 0, MAX_LINE_LENGTH);
    int line_number = 0;
    int fd = open(argv[1], O_RDONLY);
    if (fd != -1)
    {
        printf(1, "file exist\n");
        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
{
    printf(1,"File do not exist\n");
    unlink(argv[1]);
    fd=open(argv[1],O_CREATE | O_WRONLY);
}

show_text(text);
com_help(text);

char input[MAX_LINE_LENGTH] = {};
while (1)
{
    printf(1, "\nplease input command:\n");
    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;
```

```c
        int j = 0;
        for (; j < 8; j++)
        {
            if (input[j] == ' ')
            {
                pos = j + 1;
                break;
            }
        }
        //ins
        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]);
                line_number = get_line_number(text);
            }
            else if(input[3] == ' '||input[3] == '\0')
            {
                com_ins(text, line_number+1, &input[pos]);
                                line_number = get_line_number(text);
            }
            else
            {
                printf(1, "invalid command.\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]);
        else if(input[3] == ' '||input[3] == '\0')
            com_mod(text, line_number + 1, &input[pos]);
        else
        {
            printf(1, "invalid command.\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, atoi(&input[4]));
            line_number = get_line_number(text);
        }
        else if(input[3]=='\0')
        {
            com_del(text, line_number + 1);
            line_number = get_line_number(text);
        }
        else
        {
            printf(1, "invalid command.\n");
            com_help(text);
        }

    }
    else if (strcmp(input, "show") == 0)
    {
        auto_show = 1;
```

```
            printf(1, "enable show current contents after text changed.\n");
        }
        else if (strcmp(input, "hide") == 0)
        {
            auto_show = 0;
            printf(1, "disable show current contents after text changed.\n");
        }
        else if (strcmp(input, "help") == 0)
            com_help(text);
        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
        {
            printf(1, "invalid command.\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];
```

```c
    dest[len+pos] = '\0';
    return dest;
}

void show_text(char *text[])
{
    printf(1, "****************************************\n");
    printf(1, "the contents of the file are:\n");
    int j = 0;
    for (; text[j] != NULL; j++)
        printf(1, "%d%d%d:%s\n", (j+1)/100, ((j+1)%100)/10, (j+1)%10, text[j]);
}

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);
```

```c
    }
    return number;
}

void com_ins(char *text[], int n, char *extra)
{
    if (n < 0 || n > get_line_number(text) + 1)
    {
        printf(1, "invalid line number\n");
        return;
    }
    char input[MAX_LINE_LENGTH] = {};
    if (*extra == '\0')
    {
        printf(1, "please input content:\n");
        gets(input, MAX_LINE_LENGTH);
        input[strlen(input)-1] = '\0';
    }
    else
        strcpy(input, extra);
    int i = MAX_LINE_NUMBER - 1;
    for (; i > n; 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)
```

```c
        {
            memset(text[i], 0, MAX_LINE_LENGTH);
            strcpy(text[i], text[i-1]);
        }
    }
    if (text[n] == NULL)
    {
        text[n] = malloc(MAX_LINE_LENGTH);
        if (text[n-1][0] == '\0')
        {
            memset(text[n], 0, MAX_LINE_LENGTH);
            strcpy(text[n-1], input);
            changed = 1;
            if (auto_show == 1)
                show_text(text);
            return;
        }
    }
    memset(text[n], 0, MAX_LINE_LENGTH);
    strcpy(text[n], input);
    changed = 1;
    if (auto_show == 1)
        show_text(text);
}

void com_mod(char *text[], int n, char *extra)
{
    if (n <= 0 || n > get_line_number(text) + 1)
    {
        printf(1, "invalid line number\n");
        return;
    }
```

```c
    char input[MAX_LINE_LENGTH] = {};
    if (*extra == '\0')
    {
        printf(1, "please input content:\n");
        gets(input, MAX_LINE_LENGTH);
        input[strlen(input)-1] = '\0';
    }
    else
        strcpy(input, extra);
    memset(text[n-1], 0, MAX_LINE_LENGTH);
    strcpy(text[n-1], input);
    changed = 1;
    if (auto_show == 1)
        show_text(text);
}

void com_del(char *text[], int n)
{
    if (n <= 0 || n > get_line_number(text) + 1)
    {
        printf(1, "invalid line number\n");
        return;
    }
    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 (auto_show == 1)
        show_text(text);
}

void com_help(char *text[])
{
    printf(1, "****************************************\n");
    printf(1, "show, enable show current contents after executing a command.\n");
    printf(1, "hide, disable show current contents after executing a command.\n");
    printf(1, "instructions for use:\n");
    printf(1, "ins-n, insert a line after line n\n");
    printf(1, "mod-n, modify line n\n");
    printf(1, "del-n, delete line n\n");
    printf(1, "ins, insert a line after the last line\n");
    printf(1, "mod, modify the last line\n");
    printf(1, "del, delete the last line\n");
    printf(1, "save, save the file\n");
    printf(1, "exit, exit editor\n");
}

void com_save(char *text[], char *path)
{
    unlink(path);
    int fd = open(path, O_WRONLY|O_CREATE);
    if (fd == -1)
    {
        printf(1, "save failed, file can't open:\n");
```

```
        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, "saved successfully\n");
    changed = 0;
    return;
}

void com_exit(char *text[], char *path)
{
    while (changed == 1)
    {
        printf(1, "save the file? y/n\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, "wrong answer?\n");
    }
    int i = 0;
    for (; text[i] != NULL; i++)
    {
        free(text[i]);
        text[i] = 0;
    }
    exit();
}
```

OUTPUTS

```
osd-190031554@team-osd:~/190031554-xv6
190031554$ editor abc
file exist
****************************************
the contents of the file are:
001:aaaaaaaaaa
002:aaaaaaaaaa
003:dddddddddd
004:bbbbbbbbbb
005:
****************************************
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

please input command:
```

Head.c

Code:

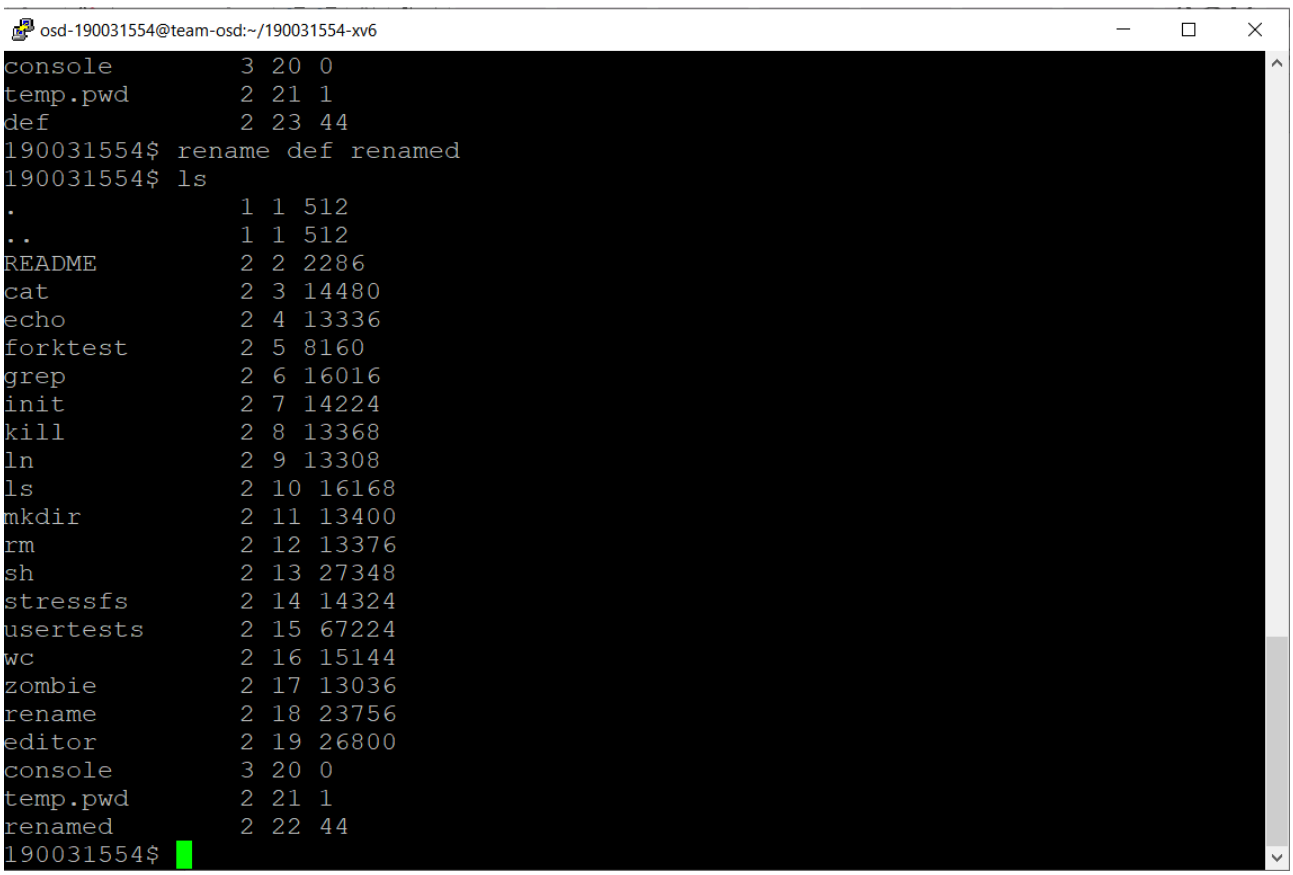
```
#include "types.h"

#include "stat.h"

#include "user.h"

char buf[512];

void head(int fd, char *name, int line)

{

 int i, n; //here the size of the read chunk is defined by n, and i is

used to keep a track of the chunk index

 int l, c; // here line number is defined by l, and the character count

in the string is defined by c

 l = c = 0;

 while((n = read(fd, buf, sizeof(buf))) > 0 )
```

```
 {
 for(i=0;i<=n ;i++)
 { //print the characters in the line
 if(buf[i] !='\n'){
 printf(1,"%c",buf[i]);
 }
//if the number of lines is equal to l, then
exit
 else if (l == (line-1)){
printf(1,"\n");
exit();
 }
//if the number of lines is not equal to l,
then jump to next line and increment the value of l
 else{
 printf(1,"\n");
 l++;
 }
 }
 }
 if(n < 0){
 printf(1, "head: read error\n");
 exit();
 }
}
int
main(int argc, char *argv[]) {
```

```
 int i;
 int fd = 0; // when the file is not specified, then it will take
input from the user
 int x = 10; // will read the first 10 lines by default
 char *file; // pointer to the name of the file
 char a;
 file = ""; // in the case when no file name is specified, it will
take input from the user
 if (argc <= 1) {
 head(0, "", 10); // handles the default case of taking input
from user for 10 lines
 exit();
 }
 else {
 for (i = 1; i < argc; i++) {
a = *argv[i]; // assigns the char value of the argv to the
var a
 if (a == '-') { // it means that -NUM is provided, hence
limited number of lines are to be printed
argv[i]++;
 x = atoi(argv[i]++);
 }
 else { // if a !='-' then it implies that number of
lines are not defined and hence default lines will print
if ((fd = open(argv[i], 0)) < 0) {// this will
execute if the file is unable to open
 printf(1, "head: cannot open %s\n", argv[i]);
```

```
exit();
}
 }
 }
 head(fd, file, x);
 close(fd);
 exit();
 }
}
```

OUTPUT:



File system checker code:

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <fcntl.h>
#include <string.h>
#define stat xv6_custom_stat // avoid clash with host (struct stat in stat.h)
#include "types.h"
#include "fs.h"
#include "stat.h"
#include "spinlock.h"
#include "sleeplock.h"
#include "buf.h"
int fsfd;
struct superblock sb;
/*Iterate over all inodes. For those of type file, iterate over its direct and
indirect links,
return the count of number of files addressed (directly or indirectly) */
// Helper for error 11
int traverse_dir_by_inum(uint addr, ushort inum)
{
 lseek(fsfd, addr*BSIZE, SEEK_SET);
 struct dirent buf;
 int i;
 for(i=0;i<BSIZE/sizeof(struct dirent);i++)
 {
 read(fsfd,&buf,sizeof(struct dirent));
 if(buf.inum==inum)
 {
 return 0;
 }
 }
 return 1;
}
//Related to error 11
int check_links(struct dinode current_inode, uint current_inum)
{
 int inum;
```

```
int count = 0;
struct dinode in;
for(inum = 0; inum < sb.ninodes; inum++)
{
if(inum == current_inum && inum != 1)
{
continue;
}
lseek(fsfd, sb.inodestart * BSIZE + inum * sizeof(struct dinode), SEEK_SET);
read(fsfd, &in, sizeof(struct dinode));
if(in.type != T_DIR)
{
continue;
}
int x;
for(x = 0; x <NDIRECT; x++)
{
if(in.addrs[x] == 0)
{
continue;
}
if(traverse_dir_by_inum(in.addrs[x], current_inum)== 0)
{
count++;
}
}
int y;
uint directory_address;
if(in.addrs[NDIRECT] != 0)
{
for(y = 0; y <NINDIRECT; y++)
{
lseek (fsfd, in.addrs[NDIRECT] * BSIZE + y*sizeof(uint), SEEK_SET);
read(fsfd, &directory_address, sizeof(uint));
if( directory_address == 0)
{
continue;
}
if(traverse_dir_by_inum(directory_address, current_inum) == 0)
{
```

```
 count++;
 }
 }
 }
 }
 return count;
}
/*
For every inode check whether it belongs to one of the given three catogeries
or not.
If not print error.
Check if nlinks to a directory inode, if it is greater than 1 print error.
For file inode compare nlinks to number of time it is referenced in directories,
if there is a mismatch print error.*/
int corrupted_inode()
{int i;
 struct dinode inode;
 char buf[sizeof(struct dinode)];
 lseek(fsfd,sb.inodestart*BSIZE,SEEK_SET);
 for (i=0;i<(int)sb.ninodes;i++)
 {
 lseek(fsfd,sb.inodestart*BSIZE+i*sizeof(struct dinode),SEEK_SET);
 read(fsfd,buf,sizeof(inode));
 memmove(&inode, buf, sizeof(inode));
 if(inode.type!=0 && inode.type!=T_FILE && inode.type!=T_DIR &&
inode.type!=T_DEV)
 {
 printf("ERROR: bad inode.\n");
 close(fsfd);
 return 1;
 }
 //This checks for error 11
 //Reference counts (number of links) for regular files match the number of
times
 //file is referred to in directories (i.e., hard links work correctly)
 //ERROR: bad refrence count for file
 if(inode.type == T_FILE)
 {
 if(inode.nlink != check_links(inode, i))
 {
```

```
 printf("ERROR: bad reference count for file.");
 close(fsfd);
 return 1;
 }
 }
 //This checks for error 12
 //No extra links allowed for directories, each directory only appears in one
other directory
 if(inode.type == T_DIR && inode.nlink > 1)
 {
 printf("ERROR: directory appears more than once in file system.\n");
 close(fsfd);
 return 1;
 }
 }
 return 0;
}
/*Function to find directory inode by name. This takes an address and name as
input and
searches for that directory entry in that particular address*/
int find_directory_by_name(uint addr, char *name)
{int i;
 struct dinode inode;
 struct dirent buf;
 lseek(fsfd,addr*BSIZE,SEEK_SET);
 //read(fsfd,&buf,sizeof(struct dirent));
 for(i=0;i<BSIZE/sizeof(struct dirent);i++)
 {
 read(fsfd,&buf,sizeof(struct dirent));
 if(buf.inum==0)
 continue;
 if(strncmp(name,buf.name,DIRSIZ)==0)
 {
 return buf.inum;
 }
 }
 return -1;
}
/*Error 7/8 each address must be used only once. Error 2 bad direct or indirect
address*/
```

```
/*THIS function traveses through all the addresses pointed to by an inode and
if that address
has not been previously encountered it sets corresponding address array
valuer to 1.
If it has been encountered print error
Also, checks the range of the addresses.*/
int check_address(uint* address, struct dinode inode)
{ int i;
 int dstart=sb.bmapstart+1;
 for(i=0;i<NDIRECT;i++)
 {
 if(inode.addrs[i] == 0) {continue;}
 if(inode.addrs[i]<dstart || inode.addrs[i]>=dstart+sb.nblocks)
 {
 printf("ERROR: bad direct address in inode.\n");
 return 1;
 }
 if(address[inode.addrs[i]] == 1)
 {
 printf("ERROR: direct address used more than once\n");
 return 1;
 }
 address[inode.addrs[i]]=1;
 }
 uint addr;
 if(inode.addrs[NDIRECT] != 0)
 {int j;
 for(j=0; j<NINDIRECT; j++)
 {
 lseek(fsfd, inode.addrs[NDIRECT] * BSIZE + j*sizeof(uint), SEEK_SET);
 read(fsfd, &addr, sizeof(uint));
 if(addr==0) {continue;}
 if(addr<dstart || addr>=dstart+sb.nblocks)
 {
 printf("ERROR: bad indirect address in inode.\n");
 return 1;
 }
 if(address[addr] == 1)
 {
 printf("ERROR: indirect address used more than once\n");
```

```
 return 1;
 }
 address[addr]=1;
 }
 }
 return 0;
}
/*ERROR 4: Check the directories for errors*/
/*This function loops through all the inodes and if it is a directory, it searches
for . and ..
If they don't exist or . points to something else prints error.*/
int check_directory(uint *address)
{int i;
 struct dinode inode;
 int dot_inode=-1;
 int ddot_inode=-1;
 char buf[sizeof(struct dinode)];
 //lseek(fsfd,sb.inodestart*BSIZE,SEEK_SET);
 for (i=0;i<sb.ninodes;i++)
 {
 lseek(fsfd, sb.inodestart*BSIZE + i*sizeof(struct dinode), SEEK_SET);
 read(fsfd,buf,sizeof(struct dinode));
 memmove(&inode, buf, sizeof(struct dinode));
 if(inode.type!=0)
 {
 if (check_address(address, inode))
 {
 return 1;
 }
 }
 if(inode.type==T_DIR)
 {int j;
 for(j=0;j<NDIRECT;j++)
 {
 if(inode.addrs[j]==0)
 continue;
 if(dot_inode==-1)
 dot_inode=find_directory_by_name(inode.addrs[j],".");
 if(ddot_inode==-1)
 ddot_inode=find_directory_by_name(inode.addrs[j],"..");
```

```
}
if(dot_inode==-1 || ddot_inode==-1)
{
if (dot_inode!=-1 && dot_inode!=i+1)
{
printf("ERROR: directory not properly formatted.\n");
return 1;
}
else if(inode.addrs[NDIRECT]!=0)
{int k;
lseek(fsfd,inode.addrs[NDIRECT]*BSIZE,SEEK_SET);
uint indbuf;
for(k=0;k<NINDIRECT;k++)
{
read(fsfd,&indbuf,sizeof(uint));
if(dot_inode==-1)
dot_inode=find_directory_by_name(indbuf,".");
if(ddot_inode==-1)
ddot_inode=find_directory_by_name(indbuf,"..");
if(dot_inode!=-1 && ddot_inode!=-1)
break;
}
if(dot_inode!=i+1 || dot_inode==-1 || ddot_inode==-1)
{
printf("ERROR: directory not properly formatted.\n");
return 1;
}
}
else
{
printf("ERROR: directory not properly formatted.\n");
// printf("%d %d %d\n", dot_inode, ddot_inode,
inode.addrs[NDIRECT]);
return 1;
}
}
if(dot_inode!=i || dot_inode==-1 || ddot_inode==-1)
{
printf("ERROR: directory not properly formatted.\n");
return 1;
```

```
 }
 }
 }
 return 0;
}
/*Error 3: this function checks if inode 1 is a directory and looks for . and .. in it.
here . and .. should point to inode 1. If not, print error*/
int check_root()
{
 struct dinode inode;
 char buf[sizeof(struct dinode)];
 lseek(fsfd,sb.inodestart*BSIZE+sizeof(struct dinode),SEEK_SET);
 read(fsfd,buf,sizeof(struct dinode));
 memmove(&inode,buf,sizeof(inode));
 //printf("%d",inode.type);
 if(inode.type!=1)
 {
 printf("ERROR: root directory does not exist.\n");
 close(fsfd);
 return 1;
 }
 else
 {int j;
 int dot_inode=-1;
 int ddot_inode=-1;
 //char buf[sizeof(struct dinode)];
 for(j=0;j<NDIRECT;j++)
 {
 if(inode.addrs[j]==0)
 continue;
 if(dot_inode==-1)
 dot_inode=find_directory_by_name(inode.addrs[j],".");
 if(ddot_inode==-1)
 ddot_inode=find_directory_by_name(inode.addrs[j],"..");
 }
 if(dot_inode!=-1 && ddot_inode!=-1)
 {
 if(dot_inode!=1 && ddot_inode!=1)
 {
 printf("ERROR: root directory does not exist.\n");
```

```
 return 1;
 }
 }
 else if(inode.addrs[NDIRECT]!=0)
 {int k;
 lseek(fsfd,inode.addrs[NDIRECT]*BSIZE,SEEK_SET);
 uint indbuf;
 for(k=0;k<NINDIRECT;k++)
 {
 read(fsfd,&indbuf,sizeof(uint));
 if(dot_inode==-1)
 dot_inode=find_directory_by_name(indbuf,".");
 if(ddot_inode==-1)
 ddot_inode=find_directory_by_name(indbuf,"..");
 if(dot_inode!=-1 && ddot_inode!=-1)
 break;
 }
 if(dot_inode!=1 || dot_inode==-1 || ddot_inode==-1 || ddot_inode!=1)
 {
 printf("ERROR: root directory does not exist.\n");
 return 1;
 }
 }
 else
 {
 printf("ERROR: root directory does not exist.\n");
 return 1;
 }
 }
 return 0;
}
/*Error 6: check address array entry and corresponding bitmap entry, both
must hold same value.
If not print error*/
int check_block_inuse(uint* address){
 // Position od datablock in Bitmap
 // Bitmapstart(in Bytes) + number of bytes in Bitmap(# total blocks%8) -
number of bytes of dataBlocks(# dataBlocks%8)
 int db_inbmap =sb.bmapstart*BSIZE + sb.size/8 - sb.nblocks/8;
 // Current address
```

```
 int current_block=(sb.bmapstart + 1);
 // Seeking cursor to the first byte of DataBlock in BitMap
 lseek(fsfd, db_inbmap, SEEK_SET);
 uint bit_to_check;
 int byte_to_check;
 int i;
 // taking Bytewise addresses from BitMap
 for (i=current_block; i<sb.size; i+=8){
 int x;
 // reading 1 Byte => it will contain usage info. about 8 DataBlocks
 read(fsfd, &byte_to_check, 1);
 for (x=0; x<8; x++){
 // Reading last bit step-by-step each time in the corresponding Byte
 bit_to_check = (byte_to_check >> x)%2;
 // bit !=0 => when DataBlock marked as in-use in BitMap
 if (bit_to_check!=0){
 // address[current_block] is 0 when it is not in use
 if(address[current_block]==0){
 printf("ERROR: bitmap marks block in use but it is not in use.\n");
 return 1;
 }
 }
 current_block++;
 }
 }
 return 0;
}
/*For error 9, given a directory inode address go through all the entries
to see if the given inode is referenced or not*/
int check_inum_indir(uint addr, ushort inum){
 lseek(fsfd, addr*BSIZE, SEEK_SET);
 struct dirent buf;
int i;
 for(i=0; i < BSIZE / sizeof(struct dirent); i++){
 read(fsfd, &buf, sizeof(struct dirent));
 if(buf.inum==inum){
 return 0;
 }
 }
 return 1;
```

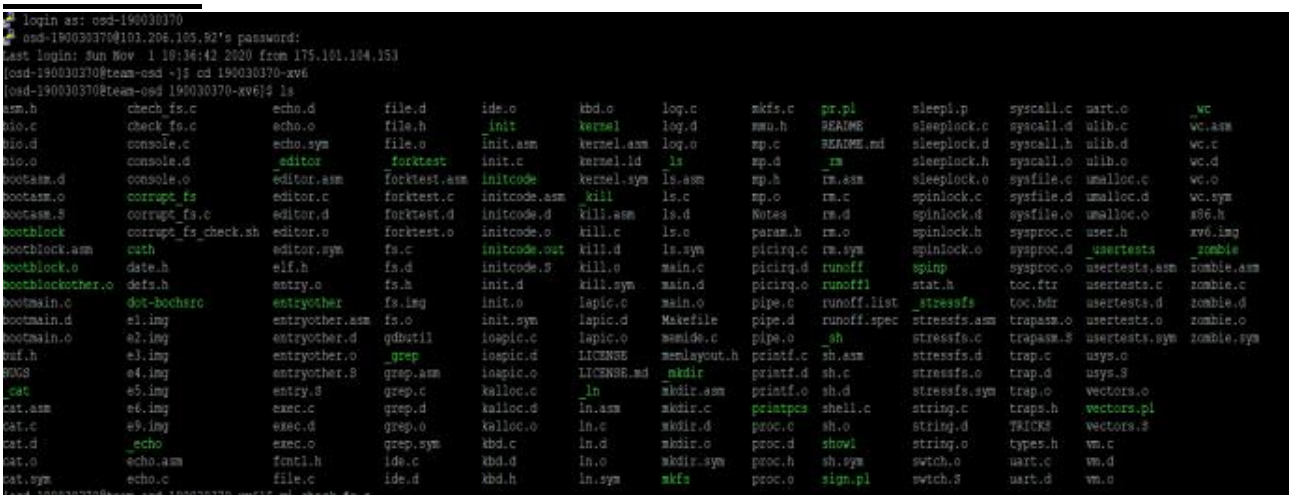
```
}
/*Error 9: For every file inode this function does a directory lookup, if the
inode is not refernd
anywhere, print error*/
int inode_check_directory(uint target_inum){
 // DIR-inode to compare
 struct dinode compare_inode;
int compare_inum;
 // loop over all the DIR inodes to find the reference to the in-use target
inode
 for (compare_inum=0; compare_inum<sb.ninodes; compare_inum++){
 // storing the current inode for comparison
 lseek(fsfd, sb.inodestart*BSIZE + compare_inum * sizeof(struct dinode),
SEEK_SET);
 read(fsfd, &compare_inode, sizeof(struct dinode));
 // 1-> T_DIR
 // skipping if it is not a Directory
 if (compare_inode.type!=1) {continue;}
 int d_ptr=0;
 // looping through all the direct-pointers
 for(d_ptr=0; d_ptr<NDIRECT; d_ptr++){
 // skipping if it is empty
 if(compare_inode.addrs[d_ptr]==0) {continue;}
 //checking if in-use target inode number present in the compare DIR
inode
 else if(check_inum_indir(compare_inode.addrs[d_ptr],
target_inum)==0){return 0;}
 }
 // For all the indirect addresses
 uint ind_DIR_address;
 // looping through all the indirect-pointers
 int ind_ptr;
 for(ind_ptr=0; ind_ptr<NINDIRECT; ind_ptr++){
 lseek(fsfd, compare_inode.addrs[NDIRECT] * BSIZE +
ind_ptr*sizeof(uint), SEEK_SET);
 read(fsfd, &ind_DIR_address, sizeof(uint));
 // skipping if empty
 if(ind_DIR_address==0) {continue;}
 // checking if in-use target inode number present in the indirect address
 if(check_inum_indir(ind_DIR_address, target_inum)==0) {return 0;}
```

```
 }
 }
 // in-use inode not found in any directory.
 printf("ERROR: inode marked use but not found in a directory\n");
 return 1;
}
/*for every address in a given inode check corresponding bitmap entry if any of
the
corresponding bitmap entry is 0, print error*/
int check_inode_addr(struct dinode current_inode){
 uint addr, byte;
 int i;
 for (i=0; i < NDIRECT+1; i++){
 if (current_inode.addrs[i]==0) {continue;}
 lseek(fsfd, sb.bmapstart*BSIZE + current_inode.addrs[i]/8,SEEK_SET);
 read(fsfd, &byte, 1);
 byte=byte >> current_inode.addrs[i]%8;
 byte=byte%2;
 if(byte==0) {
 printf("ERROR: address used by inode but marked free in bitmap.\n");
 return 1;
 }
 }
 if(current_inode.addrs[NDIRECT] != 0){
 int x;
 for(x=0; x<NINDIRECT; x++){
 lseek(fsfd, current_inode.addrs[NDIRECT] * BSIZE + x*sizeof(uint),
SEEK_SET);
 read(fsfd, &addr, sizeof(uint)) != sizeof(uint);
 if (addr!=0){
 lseek(fsfd, sb.bmapstart*BSIZE + addr/8,SEEK_SET);
 read(fsfd, &byte, 1);
 byte=byte >> current_inode.addrs[x]%8;
 byte=byte%2;
 if(byte==0) {
 printf("ERROR: address used by inode but marked free in
bitmap.\n");
 return 1;
 }
 }
```

```
 }
 }
 return 0;
}
int main(int argc, char *argv[])
{
 int i;
 uint address[sb.size];
 // initializing array address with zeores
 // For error involving blocks-in-use
 for(i=0;i<sb.size;i++) {
 address[i]=0;
 }
 if(argc<2)
 {
 printf("usage $fsck file_system_image.img\n");
 exit(1);
 }
 fsfd = open(argv[1],O_RDONLY);
 lseek(fsfd,BSIZE,SEEK_SET);
 uchar buf[BSIZE];
 read(fsfd,buf,BSIZE);
 memmove(&sb,buf,sizeof(sb));
 //error 1
 if(corrupted_inode()==1)
 return 1;
 // if(check_directory(address)==1){
 // return 1;
 // }
 // //error3 starts
 // if(check_root()==1)
 // return 1;
 //error3 ends
 // error6 starts
 if (check_block_inuse(address)){
 return 1;
 }
 // error6 ends
 //error 9
 struct dinode current_inode;
```

```
int current_inum;
 // looping through all inodes to check all the in-use inodes
 for (current_inum = 0; current_inum < ((int) sb.ninodes); current_inum++){
 lseek(fsfd, sb.inodestart * BSIZE + current_inum * sizeof(struct dinode),
SEEK_SET);
 read(fsfd, buf, sizeof(struct dinode))!=sizeof(struct dinode);
 memmove(&current_inode, buf, sizeof(current_inode));
 // only checking if the inode in use
 if (current_inode.type!=0){
 if (inode_check_directory(current_inum)){
 return 1;
 }
 if (check_inode_addr(current_inode)){
 return 1;
 }
 }
 }
 if(check_directory(address)==1){
 return 1;
 }
 //error3 starts
 if(check_root()==1)
 return 1;
 printf("Your File System is intact\n");
 return 0;
}
```

OUTPUT:



```
vi check_fs.c
nano check_fs.c
gcc -o check_fs check_fs.c
./check_fs el.img

./check_fs fs.img
```

HEAD

CODE

```c
#include "types.h"
#include "stat.h"
#include "user.h"
#define INT_MAX 2147483647

char buf[512];

void
Head(int fd,int Limit)
{
    int Current_Read = 0;
    int n;
    while((n = read(fd, buf, 1)) > 0 && Limit-1>= Current_Read ) {
        if (write(1, buf, n) != n) {
            printf(1, "Head: write error\n");
            exit();
        }
        if (buf[0] == '\n')
            Current_Read += 1;
    }
    if(n < 0){
        printf(1, "Head: read error\n");
        exit();
    }
}
```

```c
int
main(int argc, char *argv[])
{
    int fd;

    if(argc <= 1){
        Head(0,INT_MAX);
        exit();
    }


    if((fd = open(argv[1], 0)) < 0){
        printf(1, "Head: cannot open %s\n", argv[1]);
        exit();
    }
    if (argc == 2)
        Head(fd,10);
    else if (argc == 3)
        Head(fd,atoi(argv[2]));
    else
        printf(2,"Head: Too many args");
    close(fd);
    exit();
}

```

OUTPUT:

```
$ head README
xv6 is a re-implementation of Dennis Ritchie's and Ken Thompson's Unix
Version 6 (v6).  xv6 loosely follows the structure and style of v6,
but is implemented for a modern x86-based multiprocessor using ANSI C.

ACKNOWLEDGMENTS

xv6 is inspired by John Lions's Commentary on UNIX 6th Edition (Peer
to Peer Communications; ISBN: 1-57398-013-7; 1st edition (June 14,
2000)). See also https://pdos.csail.mit.edu/6.828/, which
provides pointers to on-line resources for v6.
```

ADDING A NEW SYSTEM CALL TO XV6 getprocs():-

1.nano getprocs.c :

```
#include "types.h"
#include "stat.h"
#include "user.h"
int
main(int argc, char *argv[])
{
 printf(1, "test for function getprocs()!\n");
 printf(1, "\n");
 int pNum, ppid;
 pNum = getprocs();
 printf(1, "getprocs() returns result: %d\n", pNum);
 ppid = getpid();
 printf(1, "current pid is: %d\n", ppid);
 printf(1, "\n");
 printf(1, "getprocs() returns result: %d\n", getprocs());
 printf(1, "current pid is: %d\n", getpid());
 printf(1, "\n");
 kill(ppid);
 printf(1, "getprocs() returns result: %d\n", getprocs());
 exit();
}
```

2.nano sysproc.c :

```
#include "spinlock.h"
struct {
 struct spinlock lock;
 struct proc proc[NPROC];
} ptable;
int sys_getprocs(void) {
 struct proc *p;
 int count = 0;
 for(p = ptable.proc; p < &ptable.proc[NPROC]; p++) {
 if(p->state != UNUSED) {
 count++;
 }
 }
 return count;
}
```

3.nano syscall.c :

```
extern int sys_getprocs(void);
[SYS_getprocs] sys_getprocs,
```

4.nano syscall.h :

```
#define SYS_getprocs 22
```

5.nano user.h :

```
int getprocs(void);
```

6.nano usys.S :

```
SYSCALL(getprocs)
```

7.nano Makefile :

```
UPROGS:
 _getprocs\
EXTRA:
getprocs.c\
```

OUTPUT:

```
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
```

```
.              1 1 512
..             1 1 512
README         2 2 2286
cat            2 3 14508
echo           2 4 13364
forktest       2 5 8192
grep           2 6 16044
init           2 7 14256
kill           2 8 13392
ln             2 9 13336
ls             2 10 16192
mkdir          2 11 13424
rm             2 12 13400
sh             2 13 31092
stressfs       2 14 14348
usertests      2 15 67248
wc             2 16 15172
zombie         2 17 13060
editor         2 18 26824
getprocs       2 19 13960
console        3 20 0
190030370$ getprocs
test for function getprocs()!

getprocs() returns result: 3
current pid is: 4

getprocs() returns result: 3
current pid is: 4
```

CONCLUSION

We successfully created a basic XV6 shell with what our team believes to be necessary for a common usage. We 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 our own shell was a very interesting and eventful one and even though sometimes, our code was like a shot in the dark, we believe that we achieved what we wanted to in the end.