memmove() function:

The C library function void \*memmove(void \*str1, const void \*str2, size\_t n) copies n characters from str2 to str1, but for overlapping memory blocks, memmove() is a safer approach than memcpy().

Declaration

Following is the declaration for memmove() function.

void \*memmove(void \*str1, const void \*str2, size\_t n)

Parameters

str1 − This is a pointer to the destination array where the content is to be copied, type-casted to a pointer of type void\*.

str2 − This is a pointer to the source of data to be copied, type-casted to a pointer of type void\*.

n − This is the number of bytes to be copied.

Return Value

This function returns a pointer to the destination, which is str1.

Example

The following example shows the usage of memmove() function.

#include <stdio.h>

#include <string.h>

int main () {

char dest[] = "oldstring";

const char src[] = "newstring";

printf("Before memmove dest = %s, src = %s\n", dest, src);

memmove(dest, src, 9);

printf("After memmove dest = %s, src = %s\n", dest, src);

return(0);

}

Memset () function:

Description

The C library function void \*memset(void \*str, int c, size\_t n) copies the character c (an unsigned char) to the first n characters of the string pointed to, by the argument str.

Declaration

Following is the declaration for memset() function.

void \*memset(void \*str, int c, size\_t n)

Parameters

str − This is a pointer to the block of memory to fill.

c − This is the value to be set. The value is passed as an int, but the function fills the block of memory using the unsigned char conversion of this value.

n − This is the number of bytes to be set to the value.

Return Value

This function returns a pointer to the memory area str.

Example

The following example shows the usage of memset() function.

#include <stdio.h>

#include <string.h>

int main () {

char str[50];

strcpy(str,"This is string.h library function");

puts(str);

memset(str,'$',7);

puts(str);

return(0);

}

Source: https://www.tutorialspoint.com/c\_standard\_library/c\_function\_memset.htm

**xv6 customization myls.c**

/\*program to list files and directories implementation of ls command \*/

#include "types.h"

#include "stat.h"

#include "user.h"

#include "fs.h"

int lo=0; int dot=0; int help=0;

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;

if(dot==0 && de.name[0]=='.')

continue;

memmove(p, de.name, DIRSIZ);

p[DIRSIZ] = 0;

if(stat(buf, &st) < 0){

printf(1, "ls: cannot stat %s\n", buf);

continue;

}

if(lo==1)

{

if(st.type==T\_DIR) printf(1, "\033[1m\x1B[34m%s\x1B[0m %d %d %d\n", fmtname(buf), st.type, st.ino, st.size);

else if(st.type==T\_DEV) printf(1, "\033[1m\x1B[31m%s\x1B[0m %d %d %d\n", fmtname(buf), st.type, st.ino, st.size);

else printf(1, "%s %d %d %d\n", fmtname(buf), st.type, st.ino, st.size);

}

else

{

if(st.type==T\_DIR) printf(1, "\033[1m\x1B[34m%s\x1B[0m\n", fmtname(buf));

else if(st.type==T\_DEV) printf(1, "\033[1m\x1B[31m%s\x1B[0m\n", fmtname(buf));

else printf(1, "%s\n", fmtname(buf));

}

}

break;

}

close(fd);

}

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

{

int i;

for(i=1;i<argc;i++)

{

if(argv[i][0]=='-')

{

if(strcmp(argv[i],"-l")==0) lo=1;

else if(strcmp(argv[i],"-a")==0) dot=1;

else if(strcmp(argv[i],"--help")==0) help=1;

else printf(1,"invalid OPTIONS try 'ls --help' for more information ");

}

}

if(help)

{

printf(1,"Usage : \033[1mls\x1B[0m [OPTION]... [FILE]...\n");

printf(1,"List information about the FILEs (the current directory by default).\n");

printf(1,"OPTION:\n");

printf(1,"\t\033[1m-a\x1B[0m do not ignore entries starting with .\n");

printf(1,"\t\033[1m-l\x1B[0m use a long listing format\n");

}

if(argc < 2)

{

ls(".");

exit();

}

else if((lo==1 || dot==1 || help==1) && argc < 3)

{

ls(".");

exit();

}

else if((lo==1 || dot==1 || help==1) && argc < 4)

{

ls(argv[2]);

exit();

}

else

{

ls(argv[1]);

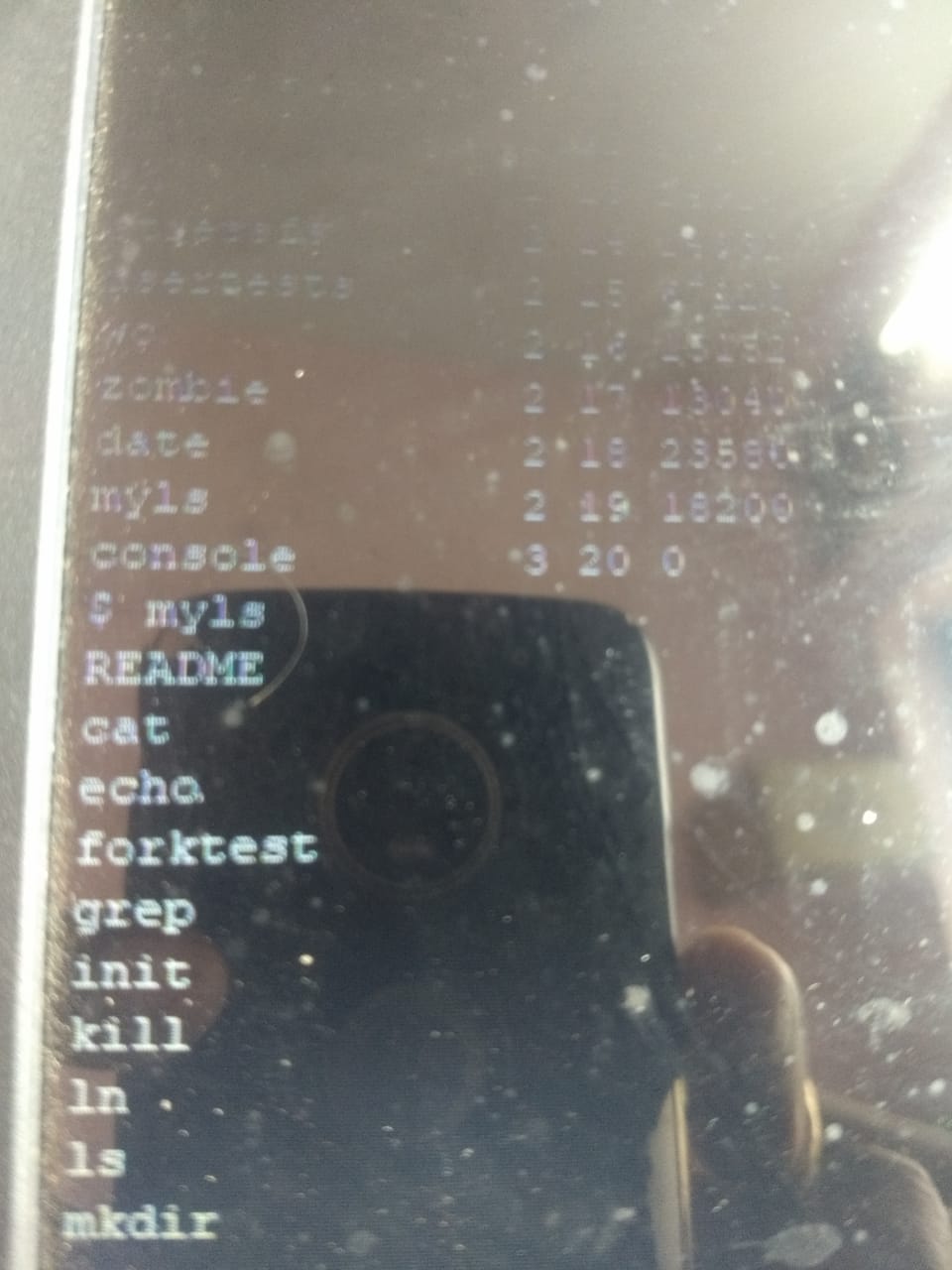
exit();

}

exit();

}

OUTPUT:



**ls.c source code xv6**

#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();

}

**Xv6 customization date.c**

#include "types.h" // this file has all the datatypes

#include "user.h" // this file has the prototypes of all the system calls

#include "date.h" // this file contains definition of struct rtcdate

// prototypes of all the functions used

long long power(int,int);

int check\_leap(int);

void month\_name(int);

void day\_name(int,int,int);

void time(void);

void yesterday(void);

void today(void);

void tomorrow(void);

void particular\_day(char \*);

void utc\_day(void);

void day(char \*);

// main

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

{

// if user only types date in the command prompt

if(argc==1)

today(); // this function prints today's date and current time (IST format)

else

{

// if user uses -d option with the date command

if((argc==3)&&(strcmp("-d",\*(argv+1))==0))

day(\*(argv+2)); // this function checks which option is chosen by user

// it then calls a suitable function to implement that option

// if user uses -u option

else if((argc==2)&&(strcmp("-u",\*(argv+1))==0))

utc\_day(); // this function prints today's date and current time (UTC format)

// if the user types an invalid command

else

printf(1,"Invalid command. Please try again.\n");

}

exit();

}

// this function calculate a power b

long long power(int x,int y)

{

long long res = 1;

int i;

for(i=0;i<y;i++)

{

res = res \* x;

}

return(res);

}

// this function whether the current year is a leap year

int check\_leap(int x)

{

int flag = 0;

if(x%400==0)

flag = 1;

else if(x%100==0)

flag = 0;

else if(x%4==0)

flag = 1;

else

flag = 0;

return(flag);

}

// this function prints the name of the month of the year

void month\_name(int x)

{

switch(x)

{

case 1:printf(1," Jan");

break;

case 2:printf(1," Feb");

break;

case 3:printf(1," Mar");

break;

case 4:printf(1," Apr");

break;

case 5:printf(1," May");

break;

case 6:printf(1," Jun");

break;

case 7:printf(1," Jul");

break;

case 8:printf(1," Aug");

break;

case 9:printf(1," Sep");

break;

case 10:printf(1," Oct");

break;

case 11:printf(1," Nov");

break;

case 12:printf(1," Dec");

break;

}

}

// this function prints the name of the day of the week

void day\_name(int x,int y,int z)

{

int initial\_day = 4;

int count = 0;

int i;

if(x>1970)

{

for(i=1970;i<x;i++)

{

if(check\_leap(i))

count += 366;

else

count += 365;

}

}

for(i=1;i<y;i++)

{

if(i==2)

{

if(check\_leap(x))

count += 29;

else

count += 28;

}

else if((i<8)&&(i%2==1))

count += 31;

else if((i<8)&&(i%2==0))

count += 30;

else if((i>=8)&&(i%2==0))

count += 31;

else

count += 30;

}

int final\_day = (initial\_day+count+z-1)%7;

switch(final\_day)

{

case 0:printf(1,"Sun");

break;

case 1:printf(1,"Mon");

break;

case 2:printf(1,"Tue");

break;

case 3:printf(1,"Wed");

break;

case 4:printf(1,"Thur");

break;

case 5:printf(1,"Fri");

break;

case 6:printf(1,"Sat");

break;

}

}

// this function prints the current time in IST format

void time()

{

struct rtcdate r;

if (date(&r))

{

printf(2, "date failed\n");

exit();

}

if(r.minute+30>59)

{

r.hour += 6;

r.minute = r.minute+30-59;

}

else

{

r.hour += 5;

r.minute += 30;

}

if(r.hour>=24)

r.hour -= 24;

printf(1," %d:%d:%d",r.hour,r.minute,r.second);

}

void yesterday()

{

struct rtcdate r;

if (date(&r))

{

printf(2, "date failed\n");

exit();

}

// if month is march

if(r.month == 3)

{

if(check\_leap(r.year))

{

if(r.day==1)

{

r.month -= 1;

r.day = 29;

}

else

r.day -= 1;

}

else

{

if(r.day==1)

{

r.month -= 1;

r.day = 28;

}

else

r.day -= 1;

}

}

// if date is 1st Jan

else if((r.day==1)&&(r.month==1))

{

r.month = 12;

r.day = 31;

r.year -= 1;

}

else

{

if(r.month<9)

{

if(r.month%2==0)

{

if(r.day==1)

{

r.month -= 1;

r.day = 31;

}

else

r.day -= 1;

}

else

{

if(r.day==1)

{

r.month -= 1;

r.day = 30;

}

else

r.day -= 1;

}

}

else

{

if(r.month%2==1)

{

if(r.day==1)

{

r.month -= 1;

r.day = 31;

}

else

r.day -= 1;

}

else

{

if(r.day==1)

{

r.month -= 1;

r.day = 30;

}

else

r.day -= 1;

}

}

}

day\_name(r.year,r.month,r.day); // prints the name of yesterday's day of the week

month\_name(r.month); // prints the name of the yesterday's month of the year

printf(1," %d",r.day); // prints yesterday's date

time(); // prints the current time (IST format)

printf(1," IST");

printf(1," %d\n",r.year); // prints yesterday's year

}

// this function prints today's date and current time (IST format)

void today()

{

struct rtcdate r;

if (date(&r))

{

printf(2, "date failed\n");

exit();

}

day\_name(r.year,r.month,r.day); // prints the name of day of the week

month\_name(r.month); // prints the name of the month of the year

printf(1," %d",r.day); // prints the today's date

time(); // prints the current time (IST format)

printf(1," IST");

printf(1," %d\n",r.year); // prints the current year

}

// this function prints tomorrow's date and time (IST format)

void tomorrow()

{

struct rtcdate r;

if (date(&r))

{

printf(2, "date failed\n");

exit();

}

// if month is Feb

if(r.month == 2)

{

if(check\_leap(r.year))

{

if(r.day==29)

{

r.month += 1;

r.day = 1;

}

else

r.day += 1;

}

else

{

if(r.day==28)

{

r.month += 1;

r.day = 1;

}

else

r.day += 1;

}

}

// if the date is 31st Dec

else if((r.day==31)&&(r.month==12))

{

r.month = 1;

r.day = 1;

r.year += 1;

}

else

{

if(r.month<8)

{

if(r.month%2==1)

{

if(r.day==31)

{

r.month += 1;

r.day = 1;

}

else

r.day += 1;

}

else

{

if(r.day==30)

{

r.month += 1;

r.day = 1;

}

else

r.day += 1;

}

}

else

{

if(r.month%2==1)

{

if(r.day==30)

{

r.month += 1;

r.day = 1;

}

else

r.day += 1;

}

else

{

if(r.day==31)

{

r.month += 1;

r.day = 1;

}

else

r.day += 1;

}

}

}

day\_name(r.year,r.month,r.day); // prints the name of tomorrow's day of the week

month\_name(r.month); // prints the name of the tomorrow's month of the year

printf(1," %d",r.day); // prints tomorrow's date

time(); // prints the current time (IST format)

printf(1," IST");

printf(1," %d\n",r.year); // prints tomorrow's year

}

// if user uses -d option with a particular date

void particular\_day(char \*x)

{

int flag = 1;

int i;

for(i=0;i<4;i++)

{

if(\*(x+i)=='-')

{

flag = 0;

break;

}

}

for(i=5;i<7;i++)

{

if(\*(x+i)=='-')

{

flag = 0;

break;

}

}

for( i=8;i<10;i++)

{

if(\*(x+i)=='-')

{

flag = 0;

break;

}

}

if(flag==0)

{

printf(1,"date: invalid date %s\n",x);

exit();

}

int y=0;

int m=0;

int d=0;

for( i=0;i<4;i++)

{

y += (\*(x+i) - '0')\*power(10,3-i);

}

for( i=5;i<7;i++)

{

m += (\*(x+i) - '0')\*power(10,6-i);

}

for( i=8;i<10;i++)

{

d += (\*(x+i) - '0')\*power(10,9-i);

}

if((m>12)||(d>31)||(m<1)||(d<1)||(y<1970))

{

printf(1,"date: invalid date %s\n",x);

exit();

}

else if(m==2)

{

if(d>28)

{

if(check\_leap(y))

{

if(d>29)

{

printf(1,"date: invalid date %s\n",x);

exit();

}

}

else

{

printf(1,"date: invalid date %s\n",x);

exit();

}

}

}

day\_name(y,m,d);

month\_name(m);

printf(1," %d",d);

printf(1," 00:00:00 IST");

printf(1," %d\n",y);

}

// this function prints today's date and current time (UTC format)

void utc\_day()

{

struct rtcdate r;

if (date(&r))

{

printf(2, "date failed\n");

exit();

}

day\_name(r.year,r.month,r.day); // prints the name of today's day of the week

month\_name(r.month); // prints the name of the today's month of the year

printf(1," %d",r.day); // prints today's date

printf(1," %d:%d:%d",r.hour,r.minute,r.second); // prints the current time (UTC format)

printf(1," UTC");

printf(1," %d\n",r.year); // prints current year

}

// this function checks which option is chosen by user

// it then calls a suitable function to implement that option

void day(char \*x)

{

switch(\*(x+2))

{

case 'd':

case 'w':today();

break;

case 'm':tomorrow();

break;

case 's':yesterday();

break;

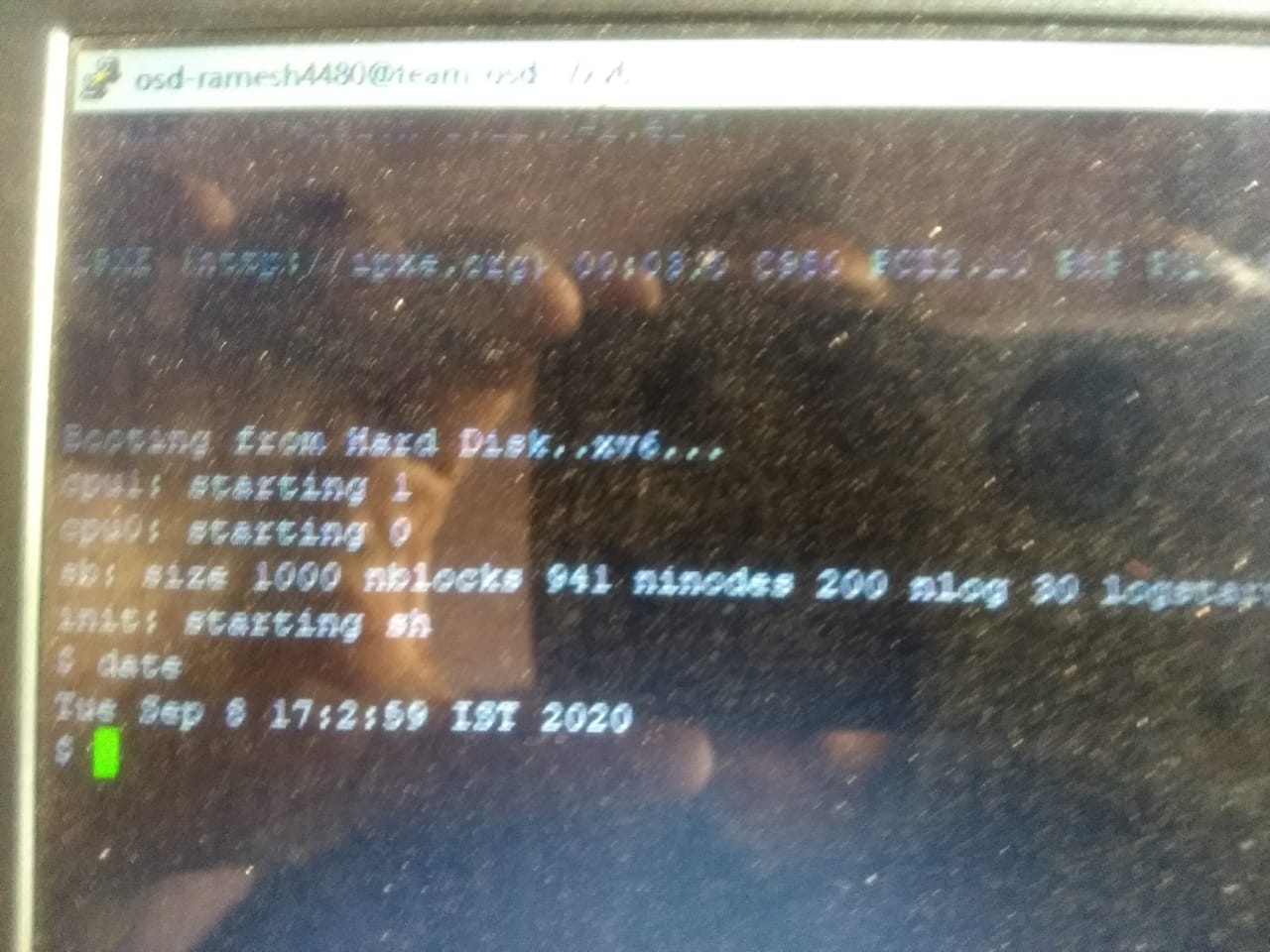
default:particular\_day(x);

break;

}

}

OUTPUT:



**Xv6 customization head.c**

#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);

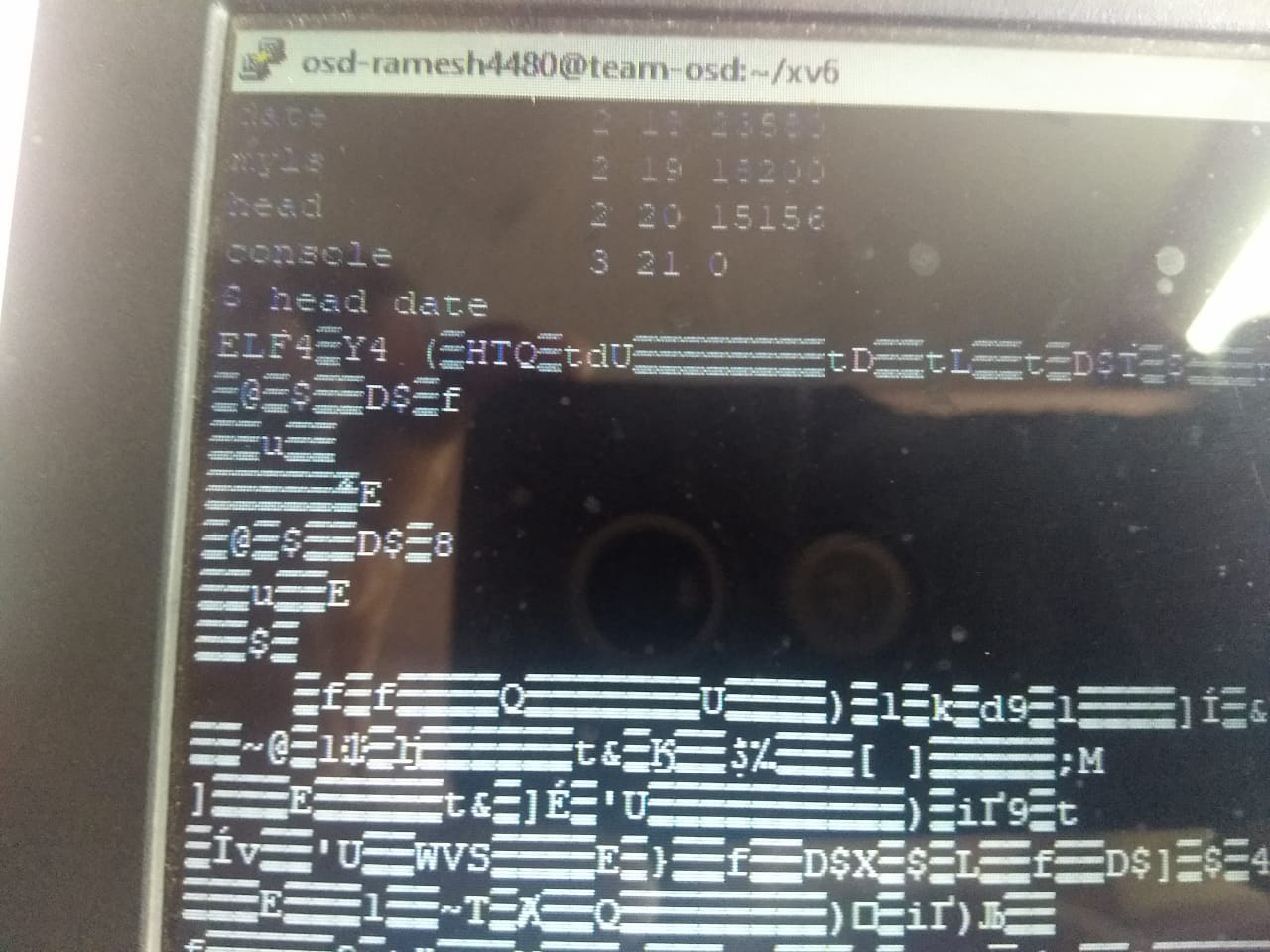
close(fd);

exit();

}

}

OUTPUT:



Out is the ascii values of the original file . date is the file name