西北工業大學
NORTHWESTERN POLYTECHNICAL UNIVERSITY

Lab 8 Report

Report Subject: OS Experiment - Lab 8

Student ID: 2018380039

Student Name: Dikshya Kafle

Submission Date: 2021/12/1

Computer Operating System Experiment

Computer Operating System Experiment

Laboratory 8

File system

1) Objective:

- ❖ Gain practice in working with the Linux file system interface.
- ❖ Gain more practice with the higher-level file I/O interface.

2) Equipment:

- ❖ VirtualBox with Ubuntu Linux

3) Experiments:

Experiment 1: The Linux file system interface.

Create a C program called fdump.c that works as described below.

1) The program must accept the following command line parameters, in the order given: filename (a C array), offset (unsigned integer), and size (unsigned integer). If the three command line parameters are not provided by the user, the program must terminate immediately with an error message and indicate the proper usage (that is, the order and the types of command line parameters expected).

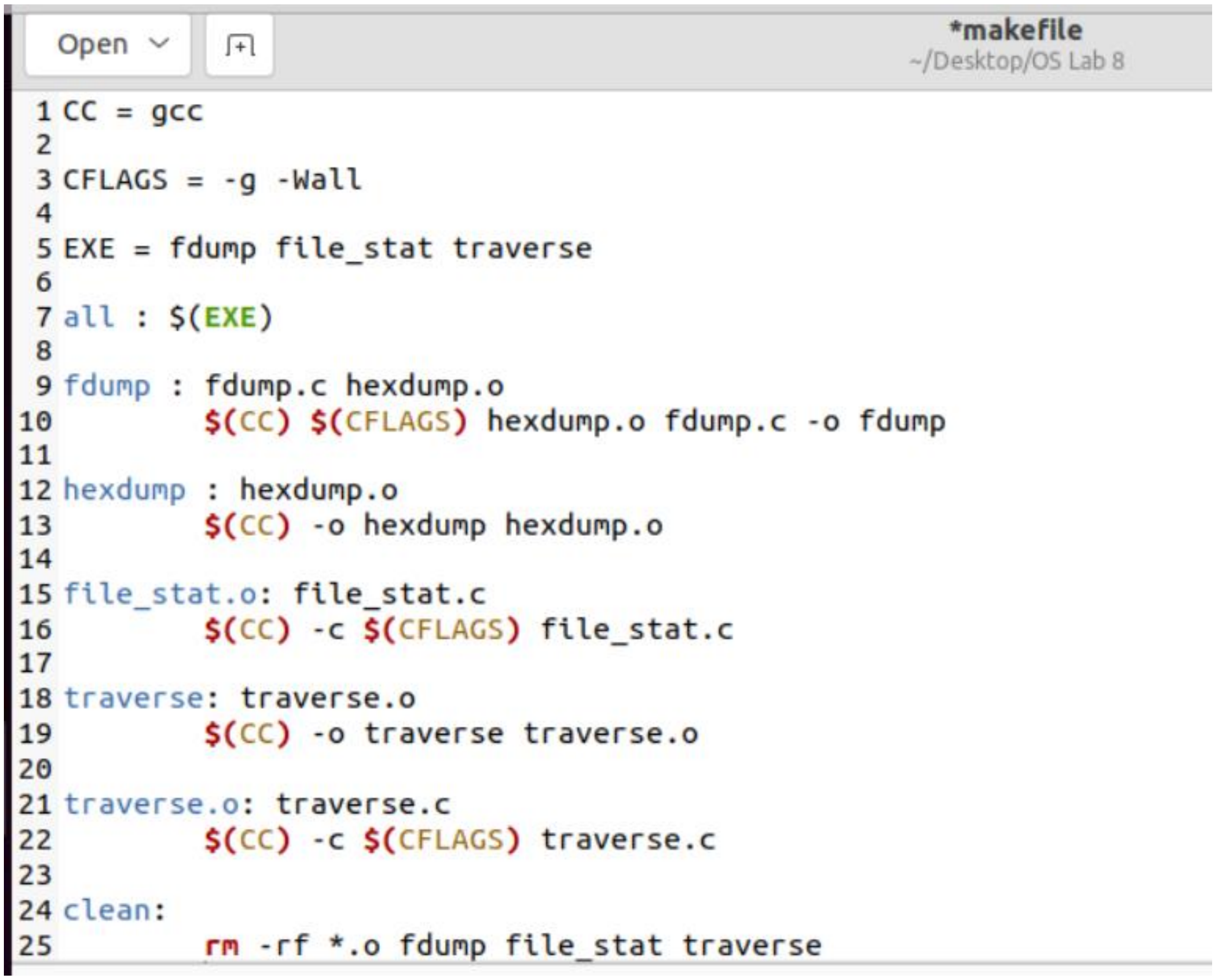
2) The program opens the file indicated by filename with **fopen**, moves forward the file position indicator by the number of bytes indicated by offset, and reads size bytes from filename into a buffer. (Hint: you will need to make a call to a random-access library function to get to the right read location into the user-specified file.)

3) Once that data is read into your program's buffer, make it call the function **hexdump** provided to you in files **hexdump.h** and **hexdump.c**. The output generated will resemble the example below.
   fdump [fileName: char[]] [offset: int] [size: int]

for example: ./fdump fdump.c 10 32

```
0000000: d4c3 b2a1 0200 0400 0000 0000 0000 0000  ................
0000010: ffff 0000 0100 0000 47c5 a943 fd14 0300  ........G..C....
0000020: 2a00 0000 2a00 0000 ffff ffff ffff 0001  *...*...........
0000030: 039c ffbd 0806 0001 0800 0604 0001 0001  ................
0000040: 039c ffbd c0a8 0166 0000 0000 0000 c0a8  .......f........
0000050: 0101 49c5 a943 5eed 0d00 4000 0000 4000  ..I..C^...@...@.
0000060: 0000 0001 039c ffbd 000c 41a1 f5da 0806  ..........A.....
0000070: 0001 0800 0604 0002 000c 41a1 f5da c0a8  ..........A.....
...
```

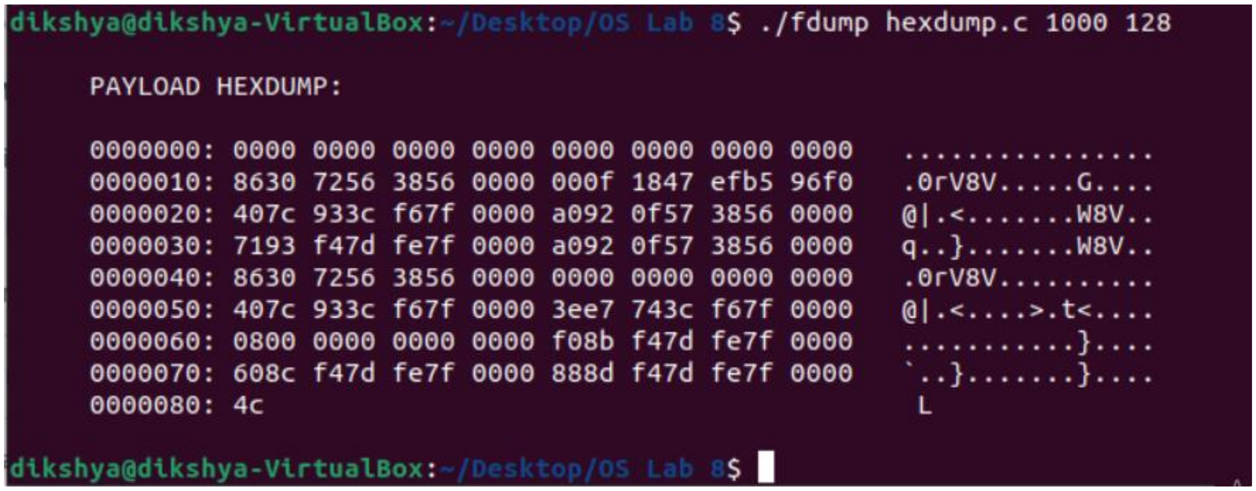
1) We create a Makefile to generate the fdump executable. The hexdump.c is compiled separately into an object that gets linked with the compilation of fdump.c.


Open
*makefile
~/Desktop/OS Lab 8

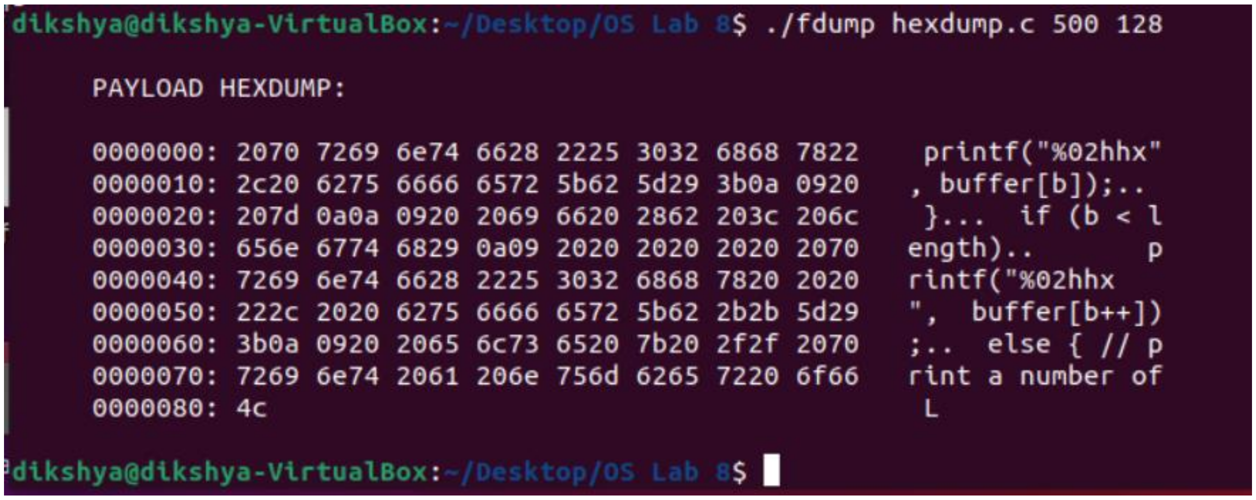
```
CC = gcc

CFLAGS = -g -Wall

EXE = fdump file_stat traverse

all : $(EXE)

fdump : fdump.c hexdump.o
	$(CC) $(CFLAGS) hexdump.o fdump.c -o fdump

hexdump : hexdump.o
	$(CC) -o hexdump hexdump.o

file_stat.o: file_stat.c
	$(CC) -c $(CFLAGS) file_stat.c

traverse: traverse.o
	$(CC) -o traverse traverse.o

traverse.o: traverse.c
	$(CC) -c $(CFLAGS) traverse.c

clean:
	rm -rf *.o fdump file_stat traverse
```

2) We run your fdump program with the following parameters: filename = hexdump.c, offset=1000, size=128. To do this we run the command: ***"./fdump hexdump.c 1000 128".***

The output shows hex values in an array format. To the right of the hex values, there is a text line that says "counter for the number of bytes (half the number of hex digits) ...".



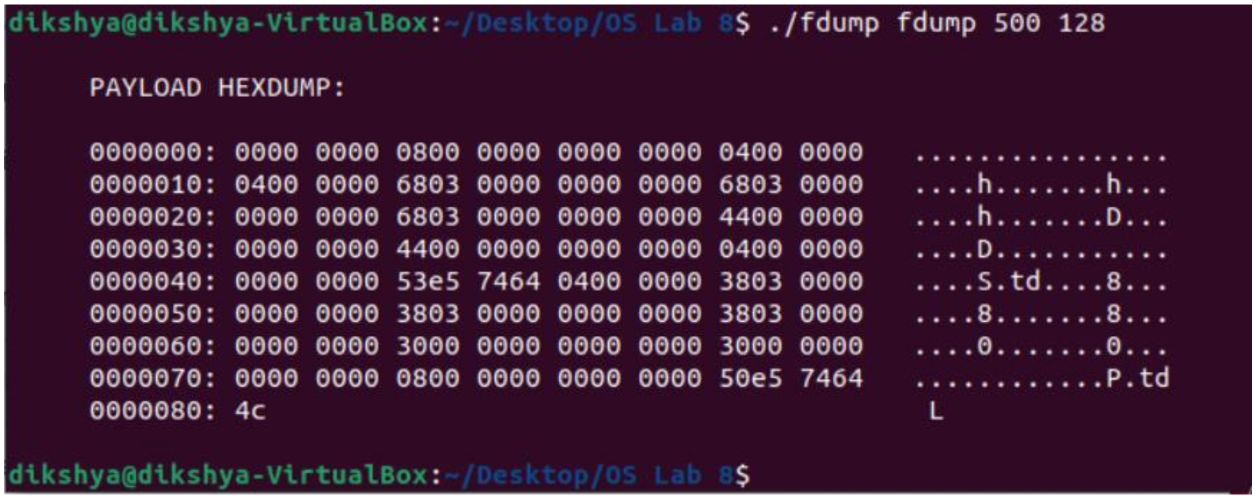
```
dikshya@dikshya-VirtualBox:~/Desktop/OS Lab 8$ ./fdump hexdump.c 1000 128

     PAYLOAD HEXDUMP:

     0000000: 0000 0000 0000 0000 0000 0000 0000 0000   ................
     0000010: 8630 7256 3856 0000 000f 1847 efb5 96f0   .0rV8V.....G....
     0000020: 407c 933c f67f 0000 a092 0f57 3856 0000   @|.<.......W8V..
     0000030: 7193 f47d fe7f 0000 a092 0f57 3856 0000   q..}.......W8V..
     0000040: 8630 7256 3856 0000 0000 0000 0000 0000   .0rV8V..........
     0000050: 407c 933c f67f 0000 3ee7 743c f67f 0000   @|.<....>.t<....
     0000060: 0800 0000 0000 0000 f08b f47d fe7f 0000   ...........}....
     0000070: 608c f47d fe7f 0000 888d f47d fe7f 0000   `..}.......}....
     0000080: 4c                                         L

dikshya@dikshya-VirtualBox:~/Desktop/OS Lab 8$
```



```
dikshya@dikshya-VirtualBox:~/Desktop/OS Lab 8$ ./fdump hexdump.c 500 128

     PAYLOAD HEXDUMP:

     0000000: 2070 7269 6e74 6628 2225 3032 6868 7822    printf("%02hhx"
     0000010: 2c20 6275 6666 6572 5b62 5d29 3b0a 0920   , buffer[b]);..
     0000020: 207d 0a0a 0920 2069 6620 2862 203c 206c    }...  if (b < l
     0000030: 656e 6774 6829 0a09 2020 2020 2020 2070   ength)..       p
     0000040: 7269 6e74 6628 2225 3032 6868 7820 2020   rintf("%02hhx
     0000050: 222c 2020 6275 6666 6572 5b62 2b2b 5d29   ",  buffer[b++])
     0000060: 3b0a 0920 2065 6c73 6520 7b20 2f2f 2070   ;..  else { // p
     0000070: 7269 6e74 2061 206e 756d 6265 7220 6f66   rint a number of
     0000080: 4c                                          L

dikshya@dikshya-VirtualBox:~/Desktop/OS Lab 8$
```

3) Run your fdump program with the following parameters: filename = fdump, offset=500, size=128. To do this we run the command: ***"./fdump fdump 1000 128".***

The output shows the hexcode of the fdump file and to the right of it is a message that contains a lot of periods and some random letters.



```
dikshya@dikshya-VirtualBox:~/Desktop/OS Lab 8$ ./fdump fdump 500 128

     PAYLOAD HEXDUMP:

     0000000: 0000 0000 0800 0000 0000 0000 0400 0000   ................
     0000010: 0400 0000 6803 0000 0000 0000 6803 0000   ....h.......h...
     0000020: 0000 0000 6803 0000 0000 0000 4400 0000   ....h.......D...
     0000030: 0000 0000 4400 0000 0000 0000 0400 0000   ....D...........
     0000040: 0000 0000 53e5 7464 0400 0000 3803 0000   ....S.td....8...
     0000050: 0000 0000 3803 0000 0000 0000 3803 0000   ....8.......8...
     0000060: 0000 0000 3000 0000 0000 0000 3000 0000   ....0.......0...
     0000070: 0000 0000 0800 0000 0000 0000 50e5 7464   ............P.td
     0000080: 4c                                         L

dikshya@dikshya-VirtualBox:~/Desktop/OS Lab 8$
```

4) In question 2.1, the data to the right of the hexadecimal dump shows human-readable text while the data to the right of the hex dump in 2.2 shows a non-human readable output.

This is because, in 2.1, we dump a file written in C, <hexdump.c>, which contains human-readable code and text. In 2.2, we dump an executable file, <fdump>, that consists of mostly non-human readable language which is why we see the difference in the format of the data to the right of the dump.

Experiment 2: Output the file information.

The manual page for **fstatvfs** shows that it returns various pieces of information on the underlying file system in which the given file resides. Using this call, you can learn the block size for the file system, the type of the file system, and the maximum length for a file name. The call returns all these data and more in an instance of struct **statvfs** (a pre-defined type), which you must have allocated previously. **fstatvfs** will receive a pointer to your instance of struct statvfs and fill up its various fields of file system information.

```
/*The function statvfs() returns information about a mounted file system. path is the pathname of any file
within the mounted file system. buf is a pointer to a statvfs structure defined approximately as follows:

*/

#include <sys/statvfs.h>

int fstatvfs(int fd, struct statvfs *buf);



struct statvfs {

    unsigned long  f_bsize;    /* file system block size */

    unsigned long  f_frsize;   /* fragment size */

    fsblkcnt_t     f_blocks;   /* size of fs in f_frsize units */

    fsblkcnt_t     f_bfree;    /* # free blocks */

    fsblkcnt_t     f_bavail;   /* # free blocks for unprivileged users */

    fsfilcnt_t     f_files;    /* # inodes */

    fsfilcnt_t     f_ffree;    /* # free inodes */

    fsfilcnt_t     f_favail;   /* # free inodes for unprivileged users */

    unsigned long  f_fsid;     /* file system ID */

    unsigned long  f_flag;     /* mount flags */

    unsigned long  f_namemax;  /* maximum filename length */

};
```

Reading about **fstat**, you will see that it returns a stat struct, which is another pre-defined type. This struct contains information such as the user id and the group id for the owner of the file, its protection bits (for user, group and other), the file size in numbers of blocks, and the times when it was last accessed, modified, and created. You must pass to **fstat** the pointer to an instance of stat struct, which you allocated previously; the system call will fill up the various fields with information on the specific file.

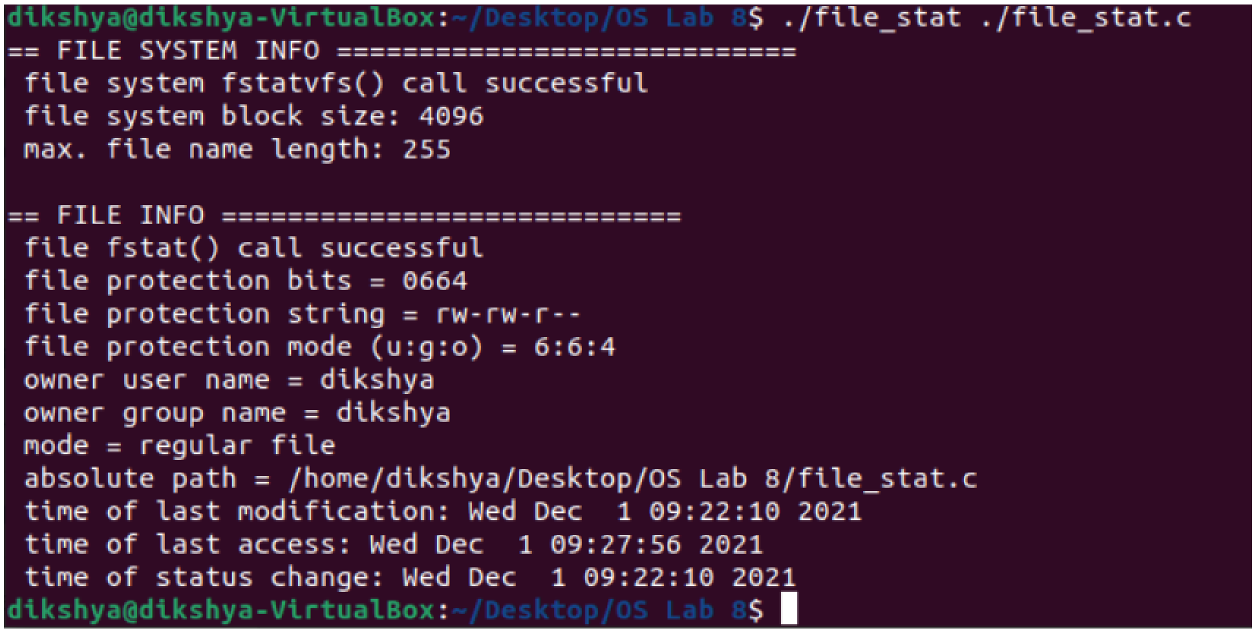
```
//stat, fstat, lstat - get file status


#include <sys/types.h>

#include <sys/stat.h>

#include <unistd.h>


int stat(const char *path, struct stat *buf);

int fstat(int fd, struct stat *buf);

int lstat(const char *path, struct stat *buf);

struct stat {

   dev_t     st_dev;     /* ID of device containing file */

   ino_t     st_ino;     /* inode number */

   mode_t    st_mode;    /* protection */

   nlink_t   st_nlink;   /* number of hard links */

   uid_t     st_uid;     /* user ID of owner */

   gid_t     st_gid;     /* group ID of owner */

   dev_t     st_rdev;    /* device ID (if special file) */

   off_t     st_size;    /* total size, in bytes */

   blksize_t st_blksize; /* blocksize for file system I/O */

   blkcnt_t  st_blocks;  /* number of 512B blocks allocated */

   time_t    st_atime;   /* time of last access */

   time_t    st_mtime;   /* time of last modification */

   time_t    st_ctime;   /* time of last status change */

};
```

Reading the source code given to you, you will notice that you must open the file before you call either **fstatvfs** or **fstat**. The code contains missing portions that you will construct in this lab. Look for them where you find comments containing the string TO-DO. Once you have filled in the missing pieces, the output produced by your file_stat executable on file file_stat.c should be somewhat similar to what is presented below (minus the text in red).

Augment the program to print the time of last access and the time of status change for the file given as command-line parameter. Note that you will need to read the man pages for **getpwuid** and **getgrgid** to learn how to translate numeric USER ID and GROUP IP to strings, respectively.

```
$ file_stat file_stat.c

== FILE SYSTEM INFO ============================

file system fstatvfs() call successful

file system block size: 65536 <———————— For you to do

max. file name length: 255 <———————— For you to do

== FILE INFO ============================

file fstat() call successful

file protection bits = 0644

file protection string = rw-r–r– <—————— For you to do

file protection mode (u:g:o) = 6:4:4  <—————— For you to do

owner user name = perrone <———————— For you to do

owner group name = cs <—————————— For you to do

mode = x <—— For you to do (x may be file, link, directory, socket, etc.)

time of last modification: Thu Nov 14 15:04:21 2014

time of last access: <—————————— For you to do

time of status change: <—————————— For you to do
```

After filling the missing portions, the output produced by the file_stat executable on file file_stat.c is as following:



```
dikshya@dikshya-VirtualBox:~/Desktop/OS Lab 8$ ./file_stat ./file_stat.c
== FILE SYSTEM INFO ============================
 file system fstatvfs() call successful
 file system block size: 4096
 max. file name length: 255

== FILE INFO ============================
 file fstat() call successful
 file protection bits = 0664
 file protection string = rw-rw-r--
 file protection mode (u:g:o) = 6:6:4
 owner user name = dikshya
 owner group name = dikshya
 mode = regular file
 absolute path = /home/dikshya/Desktop/OS Lab 8/file_stat.c
 time of last modification: Wed Dec  1 09:22:10 2021
 time of last access: Wed Dec  1 09:27:56 2021
 time of status change: Wed Dec  1 09:22:10 2021
dikshya@dikshya-VirtualBox:~/Desktop/OS Lab 8$ 
```

Experiment 3: Traverse files.

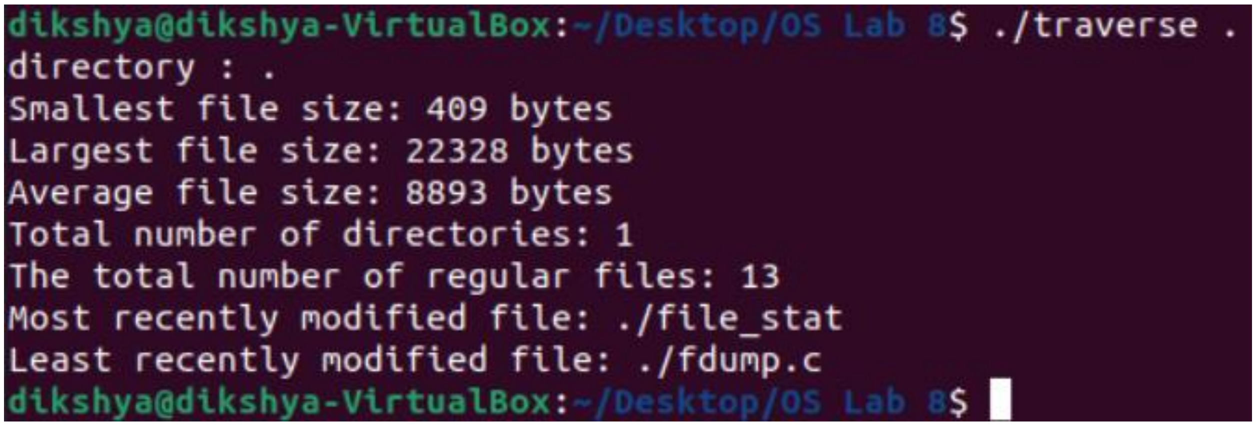
You need to create a new program called **traverse.c**, which will traverse a given directory tree, printing to the standard output the following information:

- The value of the smallest, the largest, and the average file size.
- Total number of directories.
- Total number of regular files, that is, those which are not directories, symbolic links, devices, sockets, or fifos.
- The name of the file that was most recently modified, and the one that was least recently modified in the directory tree.

Note that the size of a file can be accessed from the struct stat returned by calling **fstat**. Read the program **file_stat.c** and the manual pages **fstat** and **lstat** for more information. Make sure to modify the Makefile given to you so that it will build the newly created traverse program.

When you complete the previous step, run your program in a directory tree where there is no symbolic link and observe its behavior. Next, in a directory of your own, create a symbolic link which links it to its parent directory. Note that you are creating a loop in the directory graph. Run the program again and note what happens.

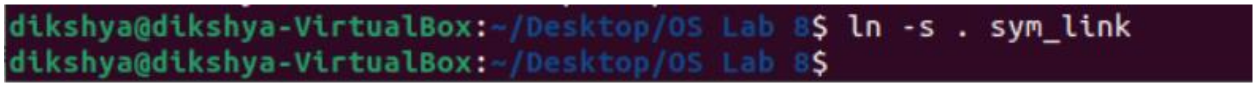
- ❖ Running the program in a directory tree where there is no symbolic link:



```
dikshya@dikshya-VirtualBox:~/Desktop/OS Lab 8$ ./traverse .
directory : .
Smallest file size: 409 bytes
Largest file size: 22328 bytes
Average file size: 8893 bytes
Total number of directories: 1
The total number of regular files: 13
Most recently modified file: ./file_stat
Least recently modified file: ./fdump.c
dikshya@dikshya-VirtualBox:~/Desktop/OS Lab 8$
```

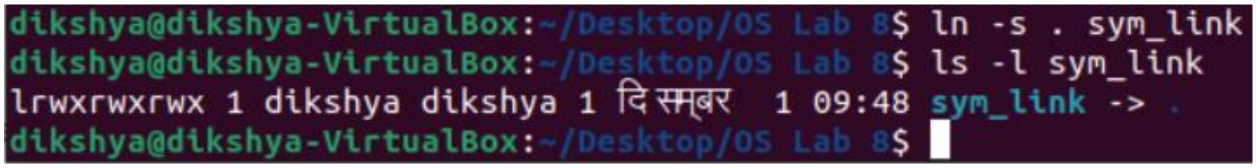
- ❖ Running the program in a directory tree where there is a symbolic link:

Creating the symbolic link:



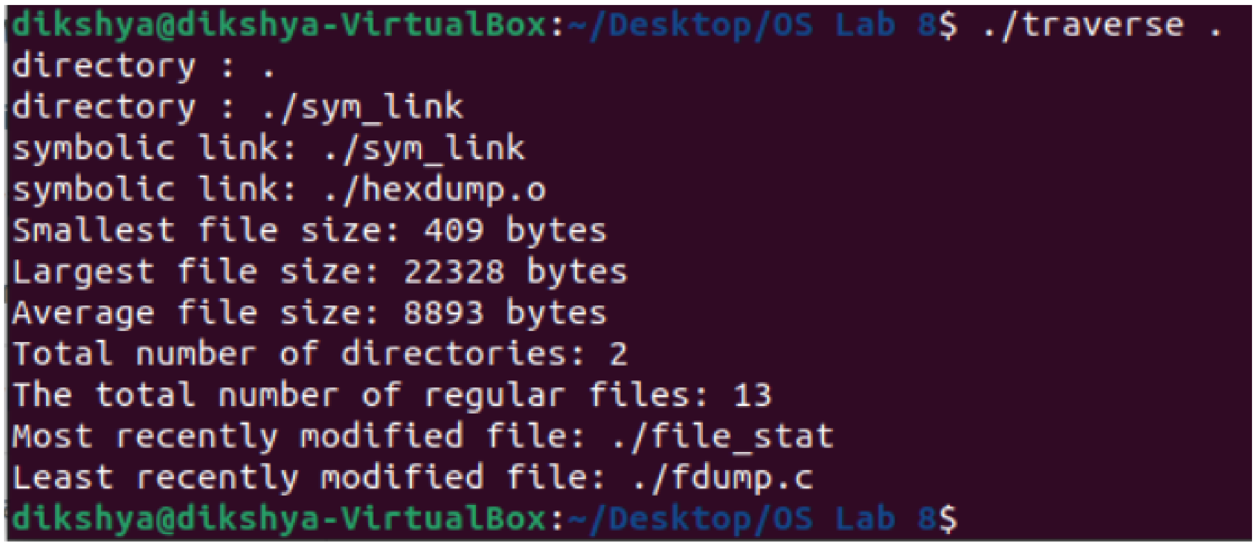
```
dikshya@dikshya-VirtualBox:~/Desktop/OS Lab 8$ ln -s . sym_link
dikshya@dikshya-VirtualBox:~/Desktop/OS Lab 8$
```

Testing the symbolic link:



```
dikshya@dikshya-VirtualBox:~/Desktop/OS Lab 8$ ln -s . sym_link
dikshya@dikshya-VirtualBox:~/Desktop/OS Lab 8$ ls -l sym_link
lrwxrwxrwx 1 dikshya dikshya 1 दिसम्बर  1 09:48 sym_link -> .
dikshya@dikshya-VirtualBox:~/Desktop/OS Lab 8$
```

Traversing



```
dikshya@dikshya-VirtualBox:~/Desktop/OS Lab 8$ ./traverse .
directory : .
directory : ./sym_link
symbolic link: ./sym_link
symbolic link: ./hexdump.o
Smallest file size: 409 bytes
Largest file size: 22328 bytes
Average file size: 8893 bytes
Total number of directories: 2
The total number of regular files: 13
Most recently modified file: ./file_stat
Least recently modified file: ./fdump.c
dikshya@dikshya-VirtualBox:~/Desktop/OS Lab 8$
```