西北工業大學

NORTHWESTERN POLYTECHNICAL UNIVERSITY

Lab | Report

Report Subject: OS Experiment - Lab 8

Student ID	**:**	2019380141
Student Name	**:**	ABID ALI
Experiment Name	**:**	File system

Objective:

- Gain practice in working with the Linux file system interface: This lab will have you work with functions to inspect various attributes of files, as you modify a program for traversing a directory tree.
- Gain more practice with the higher-level file I/O interface: By now, you have used the C/Linux file I/O APIs for lower level syscalls (open, read, etc.) and for higher level streams (fopen, fread, etc.). It pays off to think critically about the differences between the two kinds of interface, so that you understand a little more about the design of the operating system.

Equipment:

VMware with Ubuntu Linux

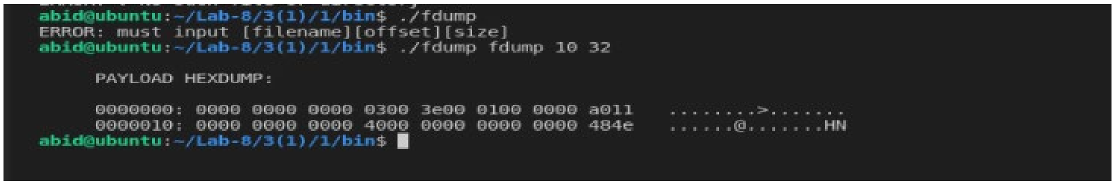
2.1 Experiment 1: practice the Linux file system interface

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).

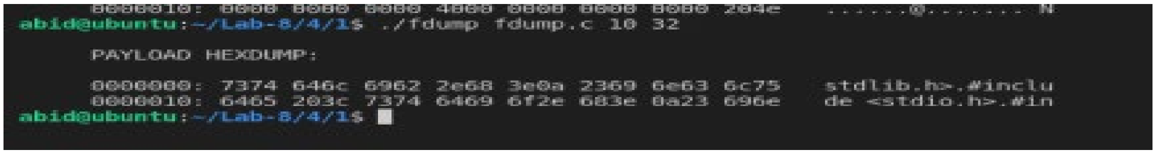
2) The program opens the file indicated by filename with **fopen**, moves forward the file position indicator by the number of bytes indicated by offset, and reads size bytes from filename into a buffer. (Hint: you will need to make a call to a random-access library function to get to the right read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



```
abid@ubuntu:~/Lab-8/3(1)/1/bin$ ./fdump
ERROR: must input [filename][offset][size]
abid@ubuntu:~/Lab-8/3(1)/1/bin$ ./fdump fdump 10 32

    PAYLOAD HEXDUMP:

    0000000: 0000 0000 0000 0300 3e00 0100 0000 a011   ........>.......
    0000010: 0000 0000 0000 4000 0000 0000 0000 484e   ......@.......HN
abid@ubuntu:~/Lab-8/3(1)/1/bin$ 
```

./fdump fdump 10 32 (Executable)



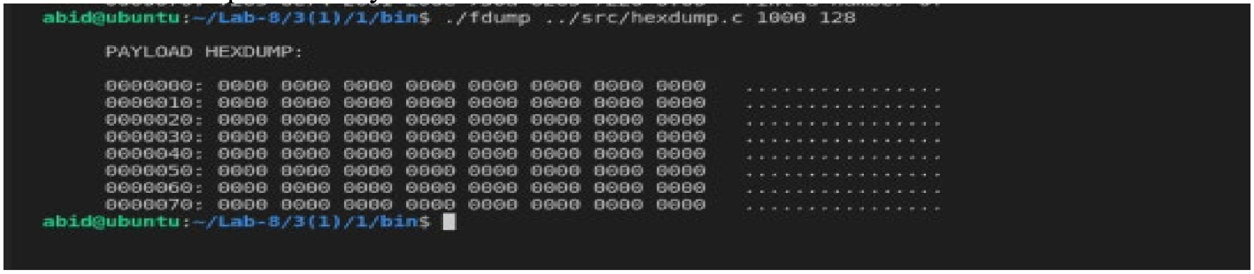
```
      0000010: 0000 0000 0000 4000 0000 0000 0000 204e   ......@....... N
abid@ubuntu:~/Lab-8/4/1$ ./fdump fdump.c 10 32

      PAYLOAD HEXDUMP:

      0000000: 7374 646c 6962 2e68 3e0a 2369 6e63 6c75   stdlib.h>.#inclu
      0000010: 6465 203c 7374 6469 6f2e 683e 0a23 696e   de <stdio.h>.#in
abid@ubuntu:~/Lab-8/4/1$ 
```

./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.....
...
```

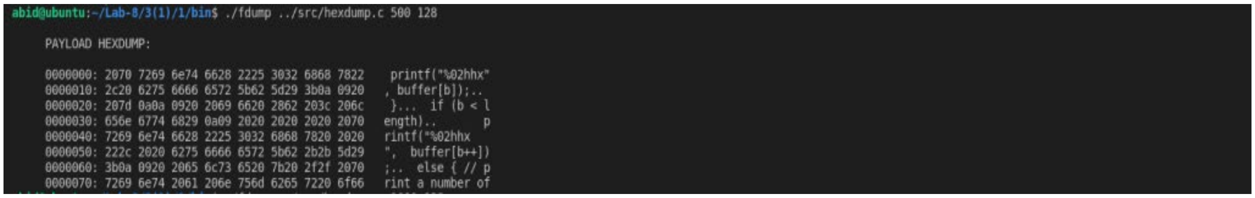
Requirements:

1) Create a Makefile to generate your fdump executable. You should compile hexdump.c separately into an object that gets linked with the compilation of fdump.c at a later point.
2) Run your fdump program with the following parameters: filename = hexdump.c, offset=1000, size=128. Explain what you see.



```
abid@ubuntu:~/Lab-8/3(1)/1/bin$ ./fdump ../src/hexdump.c 1000 128

      PAYLOAD HEXDUMP:

      0000000: 0000 0000 0000 0000 0000 0000 0000 0000   ................
      0000010: 0000 0000 0000 0000 0000 0000 0000 0000   ................
      0000020: 0000 0000 0000 0000 0000 0000 0000 0000   ................
      0000030: 0000 0000 0000 0000 0000 0000 0000 0000   ................
      0000040: 0000 0000 0000 0000 0000 0000 0000 0000   ................
      0000050: 0000 0000 0000 0000 0000 0000 0000 0000   ................
      0000060: 0000 0000 0000 0000 0000 0000 0000 0000   ................
      0000070: 0000 0000 0000 0000 0000 0000 0000 0000   ................
abid@ubuntu:~/Lab-8/3(1)/1/bin$ 
```

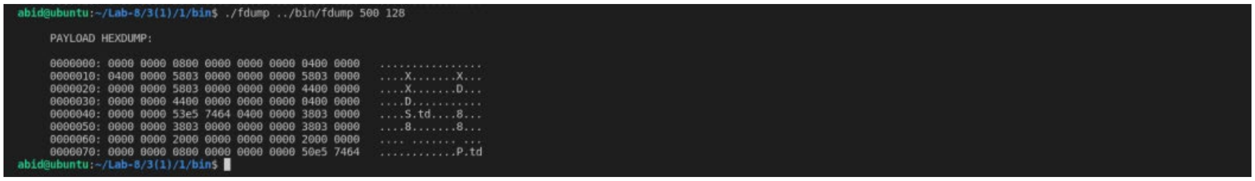
When the offset is too much bigger.I think, I see output is out of range.



```
abid@ubuntu:~/Lab-8/3(1)/1/bin$ ./fdump ../src/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
```

I see that ,output is matching the source code.

3) Run your fdump program with the following parameters: filename = fdump, offset=500, size=128. Explain what you see.



```
abid@ubuntu:~/Lab-8/3(1)/1/bin$ ./fdump ../bin/fdump 500 128

    PAYLOAD HEXDUMP:

    0000000: 0000 0000 0800 0000 0000 0000 0400 0000   ................
    0000010: 0400 0000 5803 0000 0000 0000 5803 0000   ....X.......X...
    0000020: 0000 0000 5803 0000 0000 0000 4400 0000   ....X.......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 2000 0000 0000 0000 2000 0000   .... ....... ...
    0000070: 0000 0000 0800 0000 0000 0000 50e5 7464   ............P.td
abid@ubuntu:~/Lab-8/3(1)/1/bin$ 
```

Here, the file is fdump is an excutable file, the code is converted to elf file

4) Compare the output you produced for answers (2) and (3). Looking at the hexadecimal dump on the left, in both cases, makes it clear that inside both files, you store information in binary encoding. However, the data to the right of the hexadecimal dump shows something human-readable for (2) and non-human readable for (3). In this answer, you are asked to explain why this is the case.

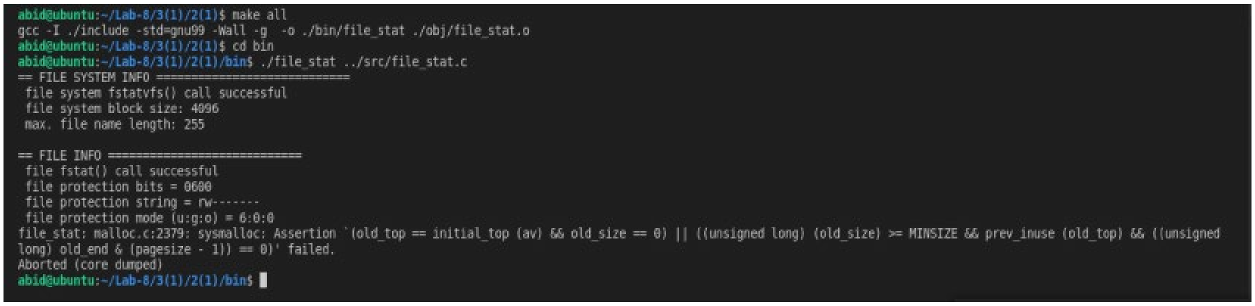
Solution:

For question 2, hexdump.c is a source file when we execute the code.We can see the part of source code and that is human readable.

For question 3, fdump is excutable file, the code is converted to elf file. So, the content is none human-readable.

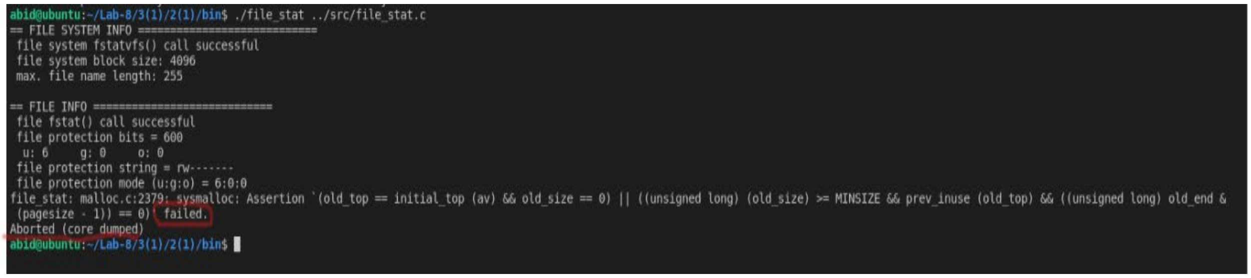
An elf file contains the bin information but it is surrounded by lots of other information, possible debug info, symbols, can distinguish code from data within the binary. Allows for more than one chunk of binary data.

2.2 Experiment 2: obtain the file information



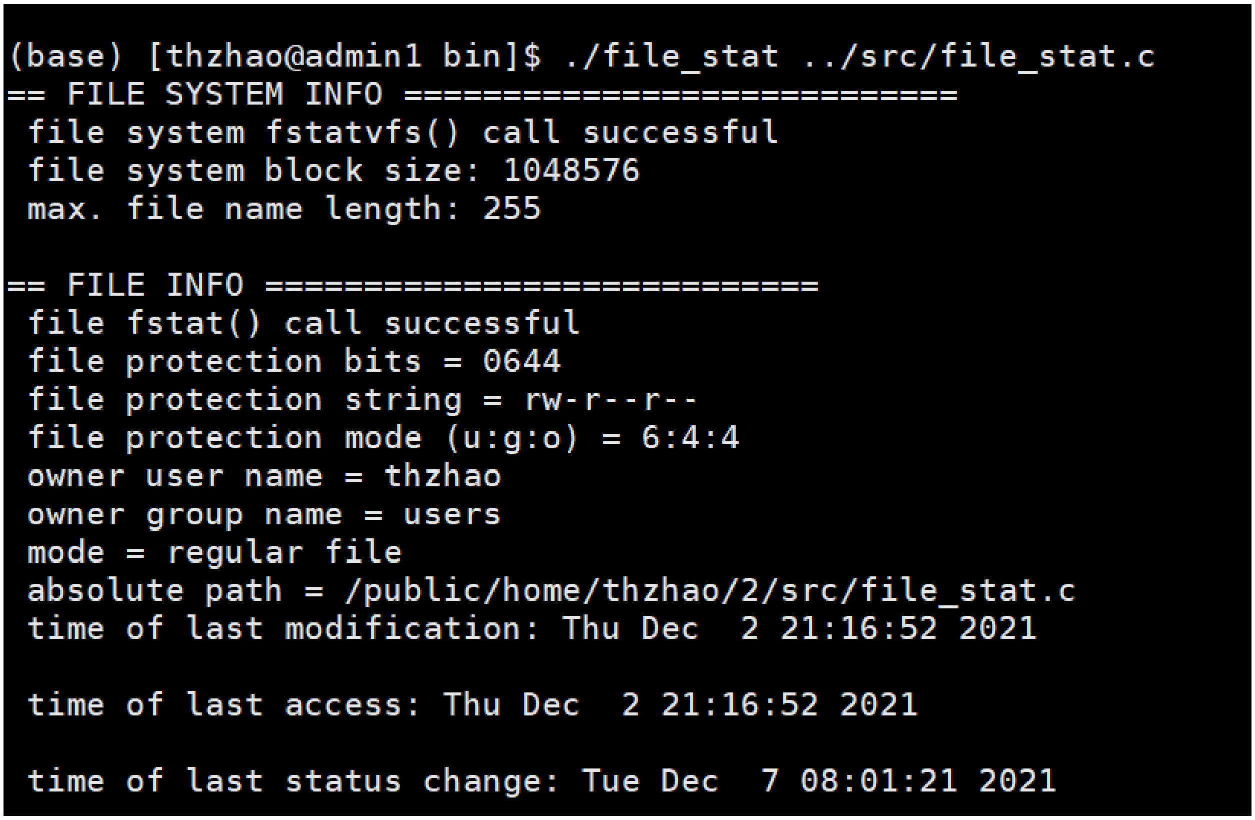
```
abid@ubuntu:~/Lab-8/3(1)/2(1)$ make all
gcc -I ./include -std=gnu99 -Wall -g  -o ./bin/file_stat ./obj/file_stat.o
abid@ubuntu:~/Lab-8/3(1)/2(1)$ cd bin
abid@ubuntu:~/Lab-8/3(1)/2(1)/bin$ ./file_stat ../src/file_stat.c
== FILE SYSTEM INFO ============================
 file system fstatvfs() call successful
 file system block size: 4096
 max. file name length: 255

== FILE INFO ============================
 file fstat() call successful
 file protection bits = 0600
 file protection string = rw-------
 file protection mode (u:g:o) = 6:0:0
file_stat: malloc.c:2379: sysmalloc: Assertion `(old_top == initial_top (av) && old_size == 0) || ((unsigned long) (old_size) >= MINSIZE && prev_inuse (old_top) && ((unsigned long) old_end & (pagesize - 1)) == 0)' failed.
Aborted (core dumped)
abid@ubuntu:~/Lab-8/3(1)/2(1)/bin$ 
```

I got this answer



```
abid@ubuntu:~/Lab-8/3(1)/2(1)/bin$ ./file_stat ../src/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 = 600
  u: 6     g: 0     o: 0
 file protection string = rw-------
 file protection mode (u:g:o) = 6:0:0
file_stat: malloc.c:2379: sysmalloc: Assertion `(old_top == initial_top (av) && old_size == 0) || ((unsigned long) (old_size) >= MINSIZE && prev_inuse (old_top) && ((unsigned long) old_end &
 (pagesize - 1)) == 0)' failed.
Aborted (core dumped)
abid@ubuntu:~/Lab-8/3(1)/2(1)/bin$ 
```

But,teacher got this code using this my code.

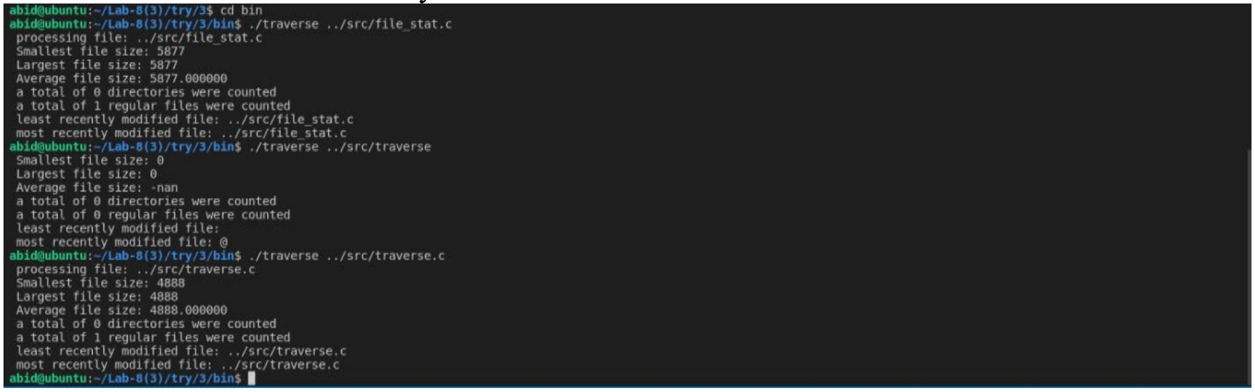


```
(base) [thzhao@admin1 bin]$ ./file_stat ../src/file_stat.c
== FILE SYSTEM INFO ============================
 file system fstatvfs() call successful
 file system block size: 1048576
 max. file name length: 255

== FILE INFO ============================
 file fstat() call successful
 file protection bits = 0644
 file protection string = rw-r--r--
 file protection mode (u:g:o) = 6:4:4
 owner user name = thzhao
 owner group name = users
 mode = regular file
 absolute path = /public/home/thzhao/2/src/file_stat.c
 time of last modification: Thu Dec  2 21:16:52 2021

 time of last access: Thu Dec  2 21:16:52 2021

 time of last status change: Tue Dec  7 08:01:21 2021
```

2.3 Experiment 3: traverse directory

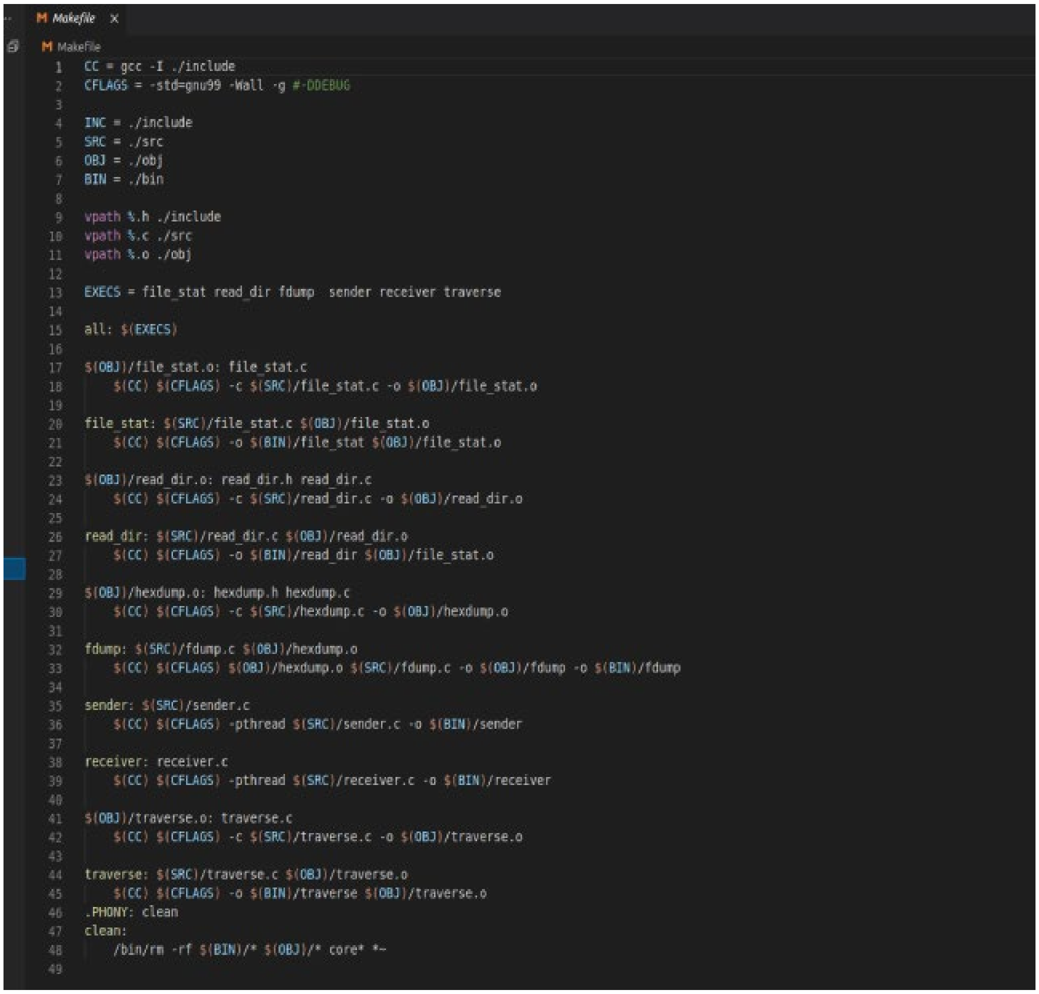
You need to create a new program called **traverse.c**, which will traverse a given directory tree, printing to the standard output the following information:

- 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.



```
abid@ubuntu:~/Lab-8(3)/try/3$ cd bin
abid@ubuntu:~/Lab-8(3)/try/3/bin$ ./traverse ../src/file_stat.c
 processing file: ../src/file_stat.c
 Smallest file size: 5877
 Largest file size: 5877
 Average file size: 5877.000000
 a total of 0 directories were counted
 a total of 1 regular files were counted
 least recently modified file: ../src/file_stat.c
 most recently modified file: ../src/file_stat.c
abid@ubuntu:~/Lab-8(3)/try/3/bin$ ./traverse ../src/traverse
 Smallest file size: 0
 Largest file size: 0
 Average file size: -nan
 a total of 0 directories were counted
 a total of 0 regular files were counted
 least recently modified file:
 most recently modified file: @
abid@ubuntu:~/Lab-8(3)/try/3/bin$ ./traverse ../src/traverse.c
 processing file: ../src/traverse.c
 Smallest file size: 4888
 Largest file size: 4888
 Average file size: 4888.000000
 a total of 0 directories were counted
 a total of 1 regular files were counted
 least recently modified file: ../src/traverse.c
 most recently modified file: ../src/traverse.c
abid@ubuntu:~/Lab-8(3)/try/3/bin$ 
```

Makefile was created



```
CC = gcc -I ./include
CFLAGS = -std=gnu99 -Wall -g #-DDEBUG

INC = ./include
SRC = ./src
OBJ = ./obj
BIN = ./bin

vpath %.h ./include
vpath %.c ./src
vpath %.o ./obj

EXECS = file_stat read_dir fdump  sender receiver traverse

all: $(EXECS)

$(OBJ)/file_stat.o: file_stat.c
    $(CC) $(CFLAGS) -c $(SRC)/file_stat.c -o $(OBJ)/file_stat.o

file_stat: $(SRC)/file_stat.c $(OBJ)/file_stat.o
    $(CC) $(CFLAGS) -o $(BIN)/file_stat $(OBJ)/file_stat.o

$(OBJ)/read_dir.o: read_dir.h read_dir.c
    $(CC) $(CFLAGS) -c $(SRC)/read_dir.c -o $(OBJ)/read_dir.o

read_dir: $(SRC)/read_dir.c $(OBJ)/read_dir.o
    $(CC) $(CFLAGS) -o $(BIN)/read_dir $(OBJ)/file_stat.o

$(OBJ)/hexdump.o: hexdump.h hexdump.c
    $(CC) $(CFLAGS) -c $(SRC)/hexdump.c -o $(OBJ)/hexdump.o

fdump: $(SRC)/fdump.c $(OBJ)/hexdump.o
    $(CC) $(CFLAGS) $(OBJ)/hexdump.o $(SRC)/fdump.c -o $(OBJ)/fdump -o $(BIN)/fdump

sender: $(SRC)/sender.c
    $(CC) $(CFLAGS) -pthread $(SRC)/sender.c -o $(BIN)/sender

receiver: receiver.c
    $(CC) $(CFLAGS) -pthread $(SRC)/receiver.c -o $(BIN)/receiver

$(OBJ)/traverse.o: traverse.c
    $(CC) $(CFLAGS) -c $(SRC)/traverse.c -o $(OBJ)/traverse.o

traverse: $(SRC)/traverse.c $(OBJ)/traverse.o
    $(CC) $(CFLAGS) -o $(BIN)/traverse $(OBJ)/traverse.o
.PHONY: clean
clean:
    /bin/rm -rf $(BIN)/* $(OBJ)/* core* *~

```

Solution:

To solve those problems I looked for information in internet. In order to understand some questions and procedure I also asked the teacher to help me understand them. And provided instructions helped to solve some of my errors during the experiment.

Problems:

The problem that I faced during was how to use Linux file system interface: This lab will have to work with functions to inspect various attributes of files, as you modify a program for traversing a directory tree..I was having problem to understand the question at the beginning .

Conclusion:

At the beginning, I was unfamiliar with the Linux file system interface: This lab will have you work with functions to inspect various attributes of files, as you modify a program for traversing a directory tree. I learned about the higher-level file I/O interface

Gradually, reading lot of article and reading teachers ppt then I solved those problem one by one. In this experiment,small helps and suggestions from the teacher was very helpful that saved my time.I enjoyed the practical and learned lot of interesting things.

Attachments:

1) ABID ALI_2019380141_OS(Lab 8).docx
2) ABID ALI_2019380141_OS(Lab 8).pdf
3)Code_ABID ALI_2019380141(Lab-8).zip