CENG328
Operating Systems

Laboratory XI
Files and Directories

1.1 File Types

- **stat**, **fstat**, and **lstat** functions; code48.c
  - Given a pathname, the **stat** function returns a structure of information about the named file. The **fstat** function obtains information about the file that is already open on the descriptor (file descriptor). The **lstat** function is similar to **stat**, but when the named file is a symbolic link, **lstat** returns information about the symbolic link, not the file referenced by the symbolic link.
  - Study the following command in detail;

    man 2 stat
  - The type of a file is encoded in the **st_mode** member of the stat structure.
  - We can determine the file type with the macros in **<sys/stat.h>** (**S_IS***()**).
  - Compile the code and execute as the following;

    ./code48 code48.c /etc /dev/random /dev/sda /vmlinuz /dev/log
  - Study the output.
  - **Exercise:** Modify the code such that it will be able to print out the information (present status, values, etc.) about all the fields in the stat structure (man 2 stat; see struct stat) not only the field protection.

1.2 File Access

- **chmod** and **fchmod** functions; code49.c
  - These two functions allow us to change the file access permissions for an existing file. The **chmod** function operates on the specified file, whereas the **fchmod** function operates on a file that has already been opened.
  - To change the permission bits of a file, the effective user ID of the process must be equal to the owner ID of the file, or the process must have superuser permissions (we will not discuss the effective user ID and superuser concepts).
  - The mode constants for chmod functions, from **<sys/stat.h>** are **S_IRWXU**, **S_IRUSR**, **S_IWUSR**, **S_IXUSR**, **S_IRWXG**, **S_IRGRP**, **S_IWGRP**, **S_IXGRP**, **S_IRWXO**, **S_IROTH**, **S_IWOTH**, **S_IXOTH**.
  - They are self-explanatory, but as an example **S_IRWXU** stands for <u>r</u>ead, <u>w</u>rite and e<u>x</u>ecute by <u>u</u>ser (owner).

1.2 File Access

- **chmod** and **fchmod** functions; code49.c
  - Compile the code.
  - Before execution, create two files as the following;

    ```
    -rw------- 1 ozdogan ozdogan 0 May 9 15:10 foo
    -rw------- 1 ozdogan ozdogan 0 May 9 15:10 bar
    ```

    by using **touch** and **chmod** commands.
  - Execute the code. Observe how the final state of the two files are changed after running the program.
  - Note that the time and date listed by the **ls** command did not change after we ran the program. By default, the **ls -l** lists the time when the contents of the file were last modified.

1.3 Reading Directories

- File tree walk code50.c, code51.c, code52.c
  - Directories can be read by anyone who has access permission to read the directory. (But only the kernel can write to a directory, to preserve file system sanity)
  - We'll use these directory routines (**opendir**, **readdir**, etc.) to write a program that traverses a file hierarchy. The goal is to produce the count of the various types of files.
  - The program takes a single argument the starting pathname and recursively descends the hierarchy from that point.
  - Apply the following commands;

```
gcc -c code52.c
gcc -c code51.c
gcc -o code50 code50.c code51.o code52.o
./code50 .
./code50 ..
```

  - Also try with different paths.
  - You can also use this program to see the file profile in your system by executing with root privileges and with the pathname "/".
  - Also see **man nftw**.

1.4 File I/O Performance

- A program to show the effect of the buffer. code53.c (also download ourhdr.h),
  - Many applications assume that standard input is file descriptor 0 and standard output is file descriptor 1. In this code we use the two defined names **STDIN_FILENO** and **STDOUT_FILENO** from **<unistd.h>**. The program does not close the input file or output file. Instead it uses the fact that whenever a process terminates, all open file descriptors are closed.
  - This program works for both text file and binary files, since there is no difference between the two to the kernel.
  - First create an input file by

    dd if=/dev/zero of=inputfile bs=8K count=100

    - The output is:

      ```
      100+0 records in
      100+0 records out
      819200 bytes (819 kB) copied, 0.00343696 s, 238 MB/s
      ```

    - Timing for the writing speed is given as the last item (as 238 MB/s),
    - What are the parameters **bs** and **count**? (**man dd**)
    - Observe the writing speed by changing the value of **bs**,
    - Observe the writing speed by changing the value of **count**.

1.4 File I/O Performance

- Download the table week11l_table.ods.
- Using dd, create a 100 MiB file named as **inputfile**.
- Now, execute the program as:

  ```
  time ./code53 < inputfile > outputfile
  ```

- Run the program using **BUFFSIZE** as 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384, 32768, 65536 and 131072. Fill the table according to the values you see on screen.