Lab 8 – File System Management

OBJECTIVE

Learn to use open, read, write, close system for file management.

TIME REQUIRED : 1 hrs

PROGRAMMING LANGUAGE : C/C++

SOFTWARE REQUIRED : Ubuntu/Fedora, gcc/gc, Text Editor, Terminal, Windows, Dev

HARDWARE REQUIRED : Core i5 in Computer Labs

FILE SYSTEM MANAGEMENT IN LINUX

File management system calls handle file manipulation jobs like creating a file, reading, and writing, etc. The Linux System calls under this are **open(), read(), write(), close().**

- **open():**
  - It is the system call to open a file.
  - This system call just opens the file, to perform operations such as read and write, we need to execute different system call to perform the operations.
  - **Syntax:**

```
fd = open (file_name, mode, permission);
Example:
fd = open ("file", O_CREAT | O_RDWR, 0777);
```

  Here,
  - file_name is the name to the file to open.
  - mode is used to define the file opening modes such as create, read, write modes.
  - permission is used to define the file permissions.

Return value: Function returns the file descriptor.

- **read():**
  - This system call opens the file in reading mode
  - We can not edit the files with this system call.
  - Multiple processes can execute the read() system call on the same file simultaneously.

Syntax:

```
length = read(file_descriptor , buffer, max_len);
```

```
Example:
n = read(0, buff, 50);
```

Here,

- file_descriptor is the file descriptor of the file.
- buffer is the name of the buffer where data is to be stored.
- max_len is the number specifying the maximum amount of that data can be read.

Return value: If successful read returns the number of bytes actually read.

- **write():**
    - This system call opens the file in writing mode
    - We can edit the files with this system call.
    - Multiple processes can not execute the write() system call on the same file simultaneously.

  Syntax:

  ```
  length = write(file_descriptor , buffer, len);
  Example:
  n = write(fd, "Hello world!", 12);
  ```

  Here,

    - file_descriptor is the file descriptor of the file.
    - buffer is the name of the buffer to be stored.
    - len is the length of the data to be written.

  Return value: If successful write() returns the number of bytes actually written.

- **close():**
    - This system call closes the opened file.

  Syntax:

  ```
  int close(int fd);
  ```

  Here,

    - fd is the file descriptor of the file to be closed.

  Return value: If file closed successfully it returns 0, else it returns -1.

C code to demonstrate example of System call:

Run the following code and write down the outcome of the programs.

Activity 8.1

The following program will create a new file and read input from the terminal. Later this will read from the file and display output from the data in the file.

```
#include<unistd.h>
#include<fcntl.h>
#include<sys/stat.h>
#include<sys/types.h>
#include<stdio.h>

int main()
{
        int n,fd;
        char buff[50];  // declaring buffer

        //message printing on the display
        printf("Enter text to write in the file:\n");
        //read from keyboard, specifying 0 as fd for std input device
        //Here, n stores the number of characters
        n= read(0, buff, 50);

        // creating a new file using open.
        fd=open("file",O_CREAT | O_RDWR, 0777);

        //writting input data to file (fd)
        write(fd, buff, n);
        //Write to display (1 is standard fd for output device)
        write(1, buff, n);

        //closing the file
        int close(int fd);

        return 0;
}
```

Answer: