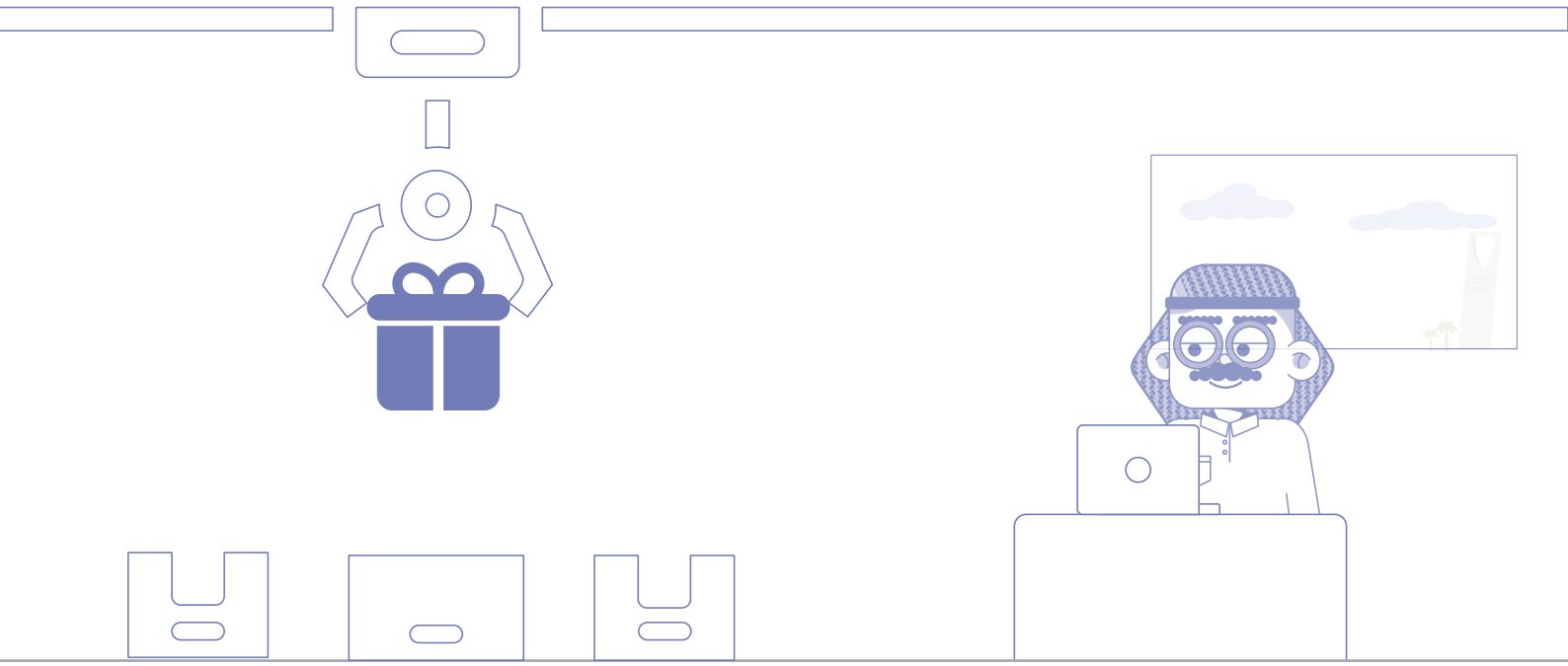
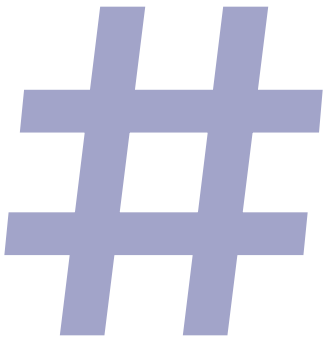
OPERATING SYSTEM

File Management



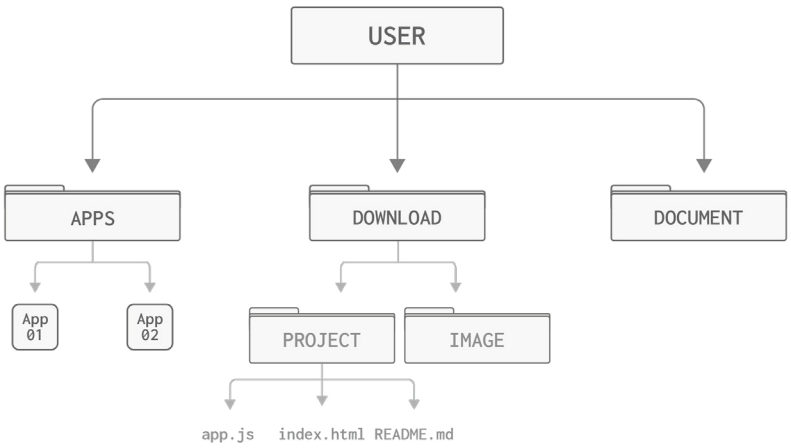


Introduction to File Management

Step 01 Introduction

Everything in a computer is stored as a file. The long-term storage device in a computer, such as a hard disk drive (HDD) or solid-state drive (SSD), stores files permanently. The disk organizes files in directories, which can be visualized as a tree data structure.

Files can include normal text files, images, and executable files (which are programs).



> **NOTE**
> Even though files are considered the primary unit of storing data in computers, at a fundamental level, all data in a computer, whether it's a program, image, or document, is represented as binary data (0s and 1s). This data is stored in files.

Operating systems allows us to manage files by providing us with a list of services (syscalls) to read, write, delete, and updated files and directories.

- Examples of file management syscalls
  - open()
  - write()
  - read()
  - close()

Step 02 File System

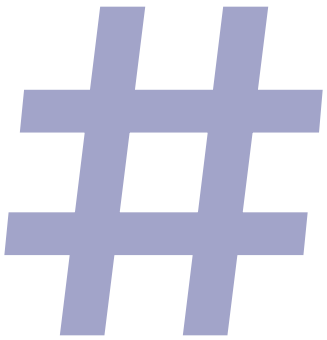
A file system is the software that manages file storage in a computer. It typically operates at the level of the operating system, serving as an integral part of the OS. Its primary functions include storing, organizing, and accessing data in files, enabling users and applications to efficiently interact with their data. The file system defines how files are named, how they are organized into directories, and how permissions and access control are managed.

The file system stores the following **information** of a file.

- **File**, which contains the actual data stored.
- **Directory**, which is used to structure files and organize them.
- **Path**, which is the way to specify the location of a file.
- **File Attribute/Metadata**, which can be used to understand how to interact with the file (such as reading or writing). Examples of metadata is the size, permissions, or the creation and last modification dates.
- **File Naming Rules**, which can be used to govern how to name files in the computer.

It also provides the following **functionalities**.

- **Storage Management**, organizes and allocates space on storage devices.
- **File Accessing**, which is done by providing methods for reading, writing, deleting, or updating files.
- **Data Integrity**, which ensures data reliability.
- **Access Control/Security**, by managing permissions and access rights.
- **File Organization**, organizes files for easier retrieval. It can be done using data structures to organize and access files efficiently.



File Management

Step 01 File Management

As explained earlier, we can manage files on the computer using the syscalls provided by the operating system. It will allow us to read, write, update, and delete files. Let us have some examples of each operation in C.

Step 02 Examples

In C, to access file functionalities, you will need to create a **FILE** pointer to point to the file we are working with and perform operations on it. Let us have a simple example of opening a file before doing anything else.

01 Open a file

To open a file, we will use **fopen** function as the following.

> NOTE
> Our project has two files in the same level (directory). The **open.c** and **file.txt.** To open **file.txt** from our C program, we will have the following code in **open.c**.

```c
#include <stdio.h>


int main(){
    FILE* fptr; // Create FILE pointer

    // fopen( <file-path>, <mode>);
    fptr = fopen("file.txt","r"); //r: read mode

    if(fptr != NULL){
        printf("File opened successfully! \n");
    }else{
        perror("There was an issue opening the file \n");
    }

    return 0;
}
```

OUTPUT

```
File opened successfully!
```

> NOTE
> The perror function in C is used to print a descriptive error message

Since the code is correct, the output displayed is **File opened successfully!**. Let's change the file name to an incorrect name for checking if it will open it or display the error message above.

```c
#include <stdio.h>


int main(){
    FILE* fptr;

    // fopen( <file-path>, <mode>);
    fptr = fopen("wrong.txt","r"); //r: read mode

    if(fptr != NULL){
        printf("File opened successfully! \n");
    }else{
        perror("There was an issue opening the file \n");
    }

    return 0;
}
```

OUTPUT

There was an issue opening the file

: No such file or directory

fopen function has the following modes to open a file.

Mode	Description
`"r"`	Read (text), open the file for reading.
`"w"`	Write (text), if the file does not exist, it will create it. If exists, it will overwrite its content.
`"a"`	Append (text), append a text to the file content.
`"rb"`	Read (binary)
`"wb"`	Write (binary)
`"ab"`	Append (binary)
`"r+"` or `rb+` or `r+b`	Read and write (open a file for update by reading or writing without discarding previous data)
`"w+"` or `wb+` or `w+b`	Read and write (discards existing data)
`"a+"` or `ab+` or `a+b`	Read and append

02 Close a file

After accessing any file, you should close the connection to remove the overhead of maintaining the file from the OS.

```c
#include <stdio.h>


int main(){
    FILE* fptr;

    fptr = fopen("file.txt","r");

    if(fptr != NULL){
        printf("File opened successfully! \n");
        fclose(fptr); // close the file
    }else{
        perror("There was an issue opening the file \n");
    }

    return 0;
}
```

OUTPUT

File opened successfully!

You can also have an if statement to check the return value from **`fclose`**, if equal to **0** then the file has been closed successfully. Else, indicate that an error occurred during the closing of the file.

```c
#include <stdio.h>


int main(){
    FILE* fptr;

    fptr = fopen("file.txt","r");

    if(fptr == NULL) { // If NULL then exit with error of opening the file.
        perror("There was an issue opening the file \n");
        return -1;
    }

    printf("File opened successfully! \n");


// If not equal to 0 then exit with error of closing the file.
    if(fclose(fptr) != 0){
        perror("Error while closing the file\n");
        return -1;
    }
        printf("File closed successfully! \n");

    return 0;
}
```

> **NOTE**
> We added some changes to the code.

OUTPUT

File opened successfully!

File closed successfully!

03 Read

To read the content of out text file, we will use the **r** mode for reading and **fgets** function to read each line of the text file as the following.

```c
#include <stdio.h>

int main(){
    FILE* fptr;

    fptr = fopen("file.txt","r"); //r: read mode

    if(fptr == NULL){
        perror("Error ocurred while opening the file.");
        return -1;
    }
    // Read every line in the text file
    char line[256]; // Act as a buffer to hold every 256 character
    fgets(line, sizeof(line), fptr);

    // Print the content
    printf("%s\n", line);

    fclose(fptr);

    return 0;
}
```

OUTPUT

```
Hello, this is my file to work with.
```

04 Write

To write into a file, we will use **w** mode to indicate writing mode. And the **fprintf** function to write text into our file.

```c
#include <stdio.h>

int main(){
    FILE* fptr;

    fptr = fopen("file.txt","w"); //w: write mode

    if(fptr == NULL){
        perror("There was an issue opening the file \n");
        return -1;
    }

    // Write into the file
    fprintf(fptr, "Hello from my C program!\nBy DEV");

    fclose(fptr);
    return 0;
}
```

OUTPUT

file.txt content

Hello from my C program!

By DEV

05 Read and Write

We can combine both read and write functionalities on a file by using **w+** or **r+** for indicating reading and writing modes.

> NOTE
> **w+** will discard the old content of the file while **r+** will not discard it. Be aware to which one you will use.

```c
#include <stdio.h>

int main(){
    FILE* fptr;

//w: write mode
    fptr = fopen("file.txt","w+");

    if(fptr == NULL){
        perror("There was an issue opening the file \n");
        return -1;
    }

// Write into the file
    fprintf(fptr, "Hello from my C program!\nBy DEV");

// Move the file pointer (position of an arrow in the file) to
the start for reading now
    rewind(fptr);

    char line[256];
    fgets(line, sizeof(line), fptr );
    printf("%s", line);

    fclose(fptr);
    return 0;
}
```

OUTPUT

```
Hello from my C program!


file.txt content
Hello from my C program!
By DEV
```

As you can see, only the first line was printed. However, we want to print the complete file content. To do so, we will include **fgets** in a loop to print each line until the content is completed.

```c
#include <stdio.h>

int main(){
    FILE* fptr;

//w: write mode
    fptr = fopen("file.txt","w+");

    if(fptr == NULL){
        perror("There was an issue opening the file \n");
        return -1;
    }

// Write into the file
    fprintf(fptr, "Hello from my C program!\nBy DEV");

// Move the file pointer (position of an arrow in the file)
to the start for reading now
    rewind(fptr);

    char line[256];
    while(fgets(line, sizeof(line), fptr )){
        printf("%s", line);
    }

    fclose(fptr);
    return 0;
}
```

OUTPUT

```
Hello from my C program!
By DEV
```

06 Update

To update the content of a file, we can use **a** which is the append mode, it will help us append a text to the file content (you can also use **w** to write into the file completely).

```c
#include <stdio.h>

int main(){
    FILE* fptr;

    fptr = fopen("file.txt","a"); //a : append mode

    if(fptr == NULL){
        perror("There was an issue opening the file \n");
        return -1;
    }

    // Write into the file
    fprintf(fptr, "\nDeveloper");

    fclose(fptr);
    return 0;
}
```

OUTPUT

file.txt content **before** running the code

```
Hello, please add your name below.
```

file.txt content **after** running the code

```
Hello, please add your name below.
Developer
```

07 Delete

To delete a file, we will pass the file name to the **remove** function as the following.

```c

#include <stdio.h>

int main(){

    if (remove("file.txt") == 0) {
        printf("File deleted successfully.\n");
    } else {
        perror("Error deleting the file");
        return -1;
    }

    return 0;
}
```

OUTPUT

```
File deleted successfully.
```