File Descriptors and Dup system call

Objectives:

- Define what a file descriptor is in Unix-like operating systems.
- Explain how the OS uses file descriptors to manage I/O resources.
- Differentiate between file descriptors and C file pointers (FILE *).
- Understand the role of dup and dup2 in file descriptor duplication and redirection.

File descriptors:

When a program opens a file, the operating system (OS) needs a way to keep track of that open file.
To do this, the OS creates an internal entry (inside the kernel) that stores information about the opened file, such as:

- where the file is,
- how it was opened (read/write),
- and the current position in the file.

Each of these entries is identified by a number. This number is called a **file descriptor**.

What is a file descriptor?

A file descriptor is a small integer number that uniquely identifies an open file or I/O resource for a specific process. Instead of using the file name again and again, the program uses this number to work with the file. For example, if a process opens 10 files, the OS creates 10 entries. Each entry has a different integer (for example: 3, 4, 5, …). These integers are the file descriptors for that process.

In Unix-like operating systems (Linux, macOS, FreeBSD), file descriptors are used for:

- Files
- Keyboards and screens
- Pipes
- Network sockets

So, a file descriptor represents any input/output (I/O) resource, not just files.

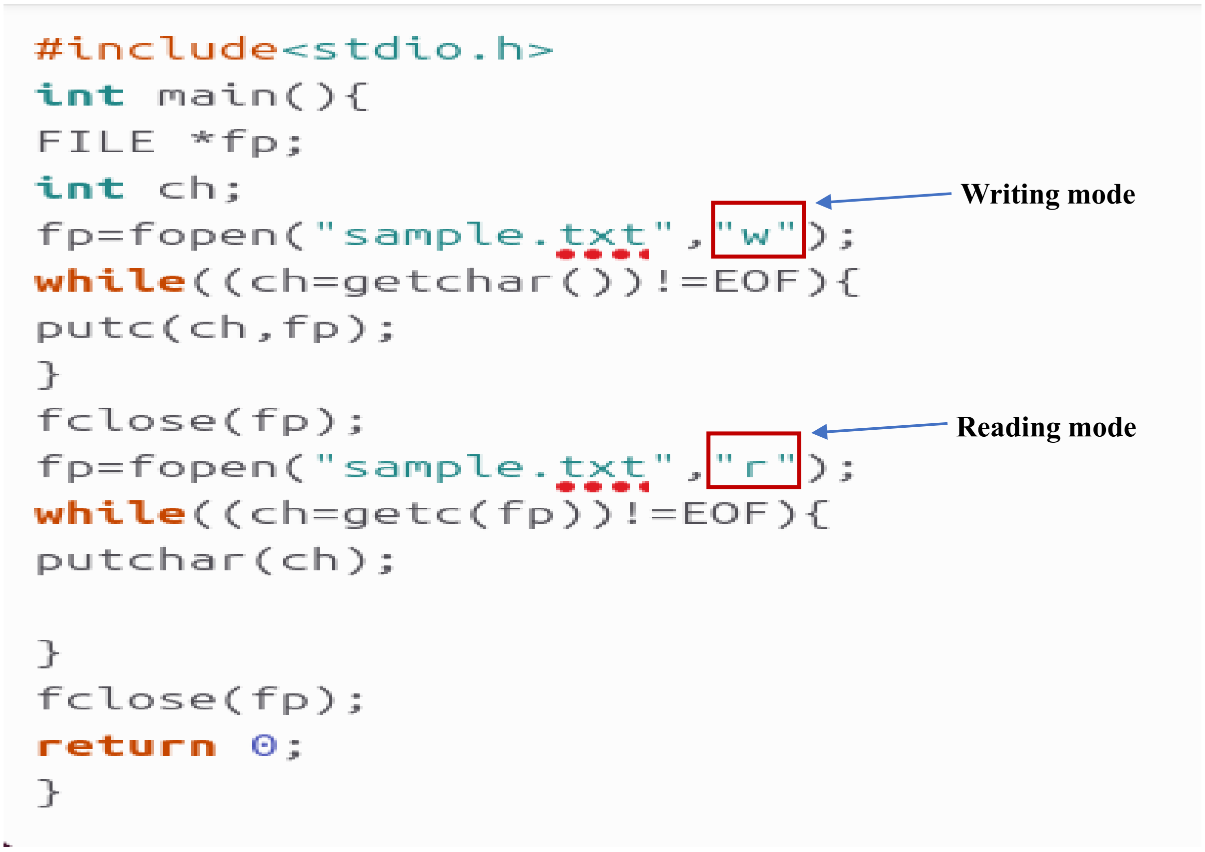
How a process uses a file descriptor?

Once a file descriptor is assigned:

1. the process uses it to read from the resource
2. write to the resource
3. or close the resource

The process does not need to know how the resource is implemented internally.The file descriptor acts like a handle or reference. In Unix-like systems, file descriptors are typically represented as small integers, and **the first three descriptors, 0, 1, and 2, are reserved for the standard input, standard output, and standard error streams,** respectively. Additional file descriptors are allocated as needed when files or other I/O resources are opened by the process.

While the Operating System uses File Descriptors to track resources like keyboards and files, C programmers use File Pointers (FILE *fp) as a "smart wrapper" to make data handling easier and more efficient.



```
#include<stdio.h>
int main(){
FILE *fp;
int ch;
fp=fopen("sample.txt","w");
while((ch=getchar())!=EOF){
putc(ch,fp);
}
fclose(fp);
fp=fopen("sample.txt","r");
while((ch=getc(fp))!=EOF){
putchar(ch);

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

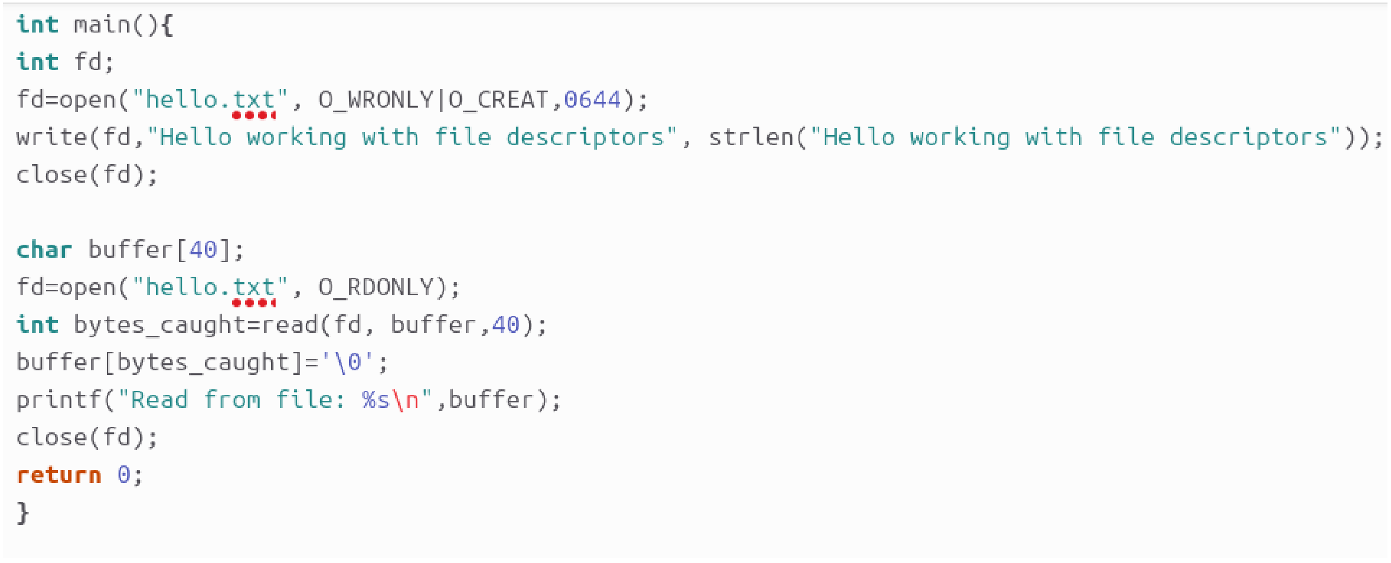
getchar(): Reads a single character from the standard input (usually your keyboard).

putchar(ch): Writes a single character to the standard output (usually your screen).

getc(fp): Reads a single character from a specific file stream identified by the file pointer fp.

putc(ch, fp): Writes a single character to a specific file stream identified by the file pointer fp.

Under the Hood:



```
int main(){
int fd;
fd=open("hello.txt", O_WRONLY|O_CREAT,0644);
write(fd,"Hello working with file descriptors", strlen("Hello working with file descriptors"));
close(fd);

char buffer[40];
fd=open("hello.txt", O_RDONLY);
int bytes_caught=read(fd, buffer,40);
buffer[bytes_caught]='\0';
printf("Read from file: %s\n",buffer);
close(fd);
return 0;
}
```

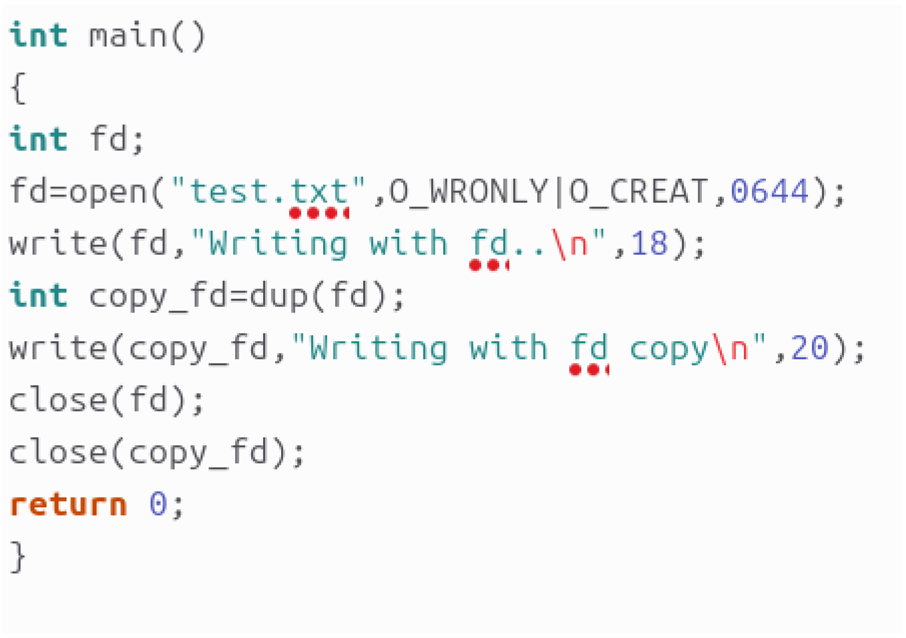
Dup and Dup2 system calls:

The **dup** system call in Unix-like operating systems is used to duplicate a file descriptor. This means it creates a new file descriptor that refers to the same open file description as the original one. The new file descriptor is the lowest-numbered available descriptor.

```
int new_fd = dup(old_fd);
```

- it accepts the old file descriptor to be duplicated
- and returns the new file descriptor on success or -1 on error.

It's like making a copy of a key; both keys open the same door.



```
int main()
{
int fd;
fd=open("test.txt",O_WRONLY|O_CREAT,0644);
write(fd,"Writing with fd..\n",18);
int copy_fd=dup(fd);
write(copy_fd,"Writing with fd copy\n",20);
close(fd);
close(copy_fd);
return 0;
}
```

The dup2 system call duplicates the file descriptor oldfd to newfd, allowing you to specify a particular file descriptor number for the duplication. If newfd is already in use, it is closed before the duplication occurs.

This is particularly useful for redirecting standard input/output or managing file descriptors in a controlled manner.

```
dup2(int oldfd, int newfd);
```

- It takes oldfd (the file descriptor to be duplicated) and newfd (the file descriptor to which oldfd should be duplicated) and
- returns the new file descriptor on success, or -1 on error.

```
int file_desc = open("file.txt", O_WRONLY | O_APPEND);
dup2(file_desc, STDOUT_FILENO);  // Redirects stdout to file.txt
```

```
printf("%d\n", someData);  //Now, printf doesn't go to the screen. It goes to file.txt!
```