Lab02

# E/19/166

# Jayathunga W.W.K.

**Exercise 1.1**

1. **O\_WRONLY:** This flag opens the file for writing, retains the existing file contents, and puts the file pointer at the start of the file. Any writes overwrite existing content.

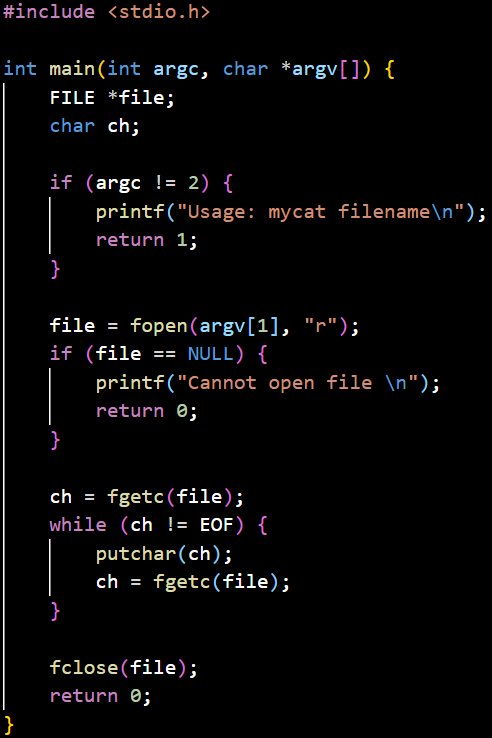
**O\_APPEND:** This flag causes writes to append to the end of the file instead of overwriting at the start. This flag is persistent. If we move the cursor elsewhere to read data, it’s always repositioned to the end of the file before each write.

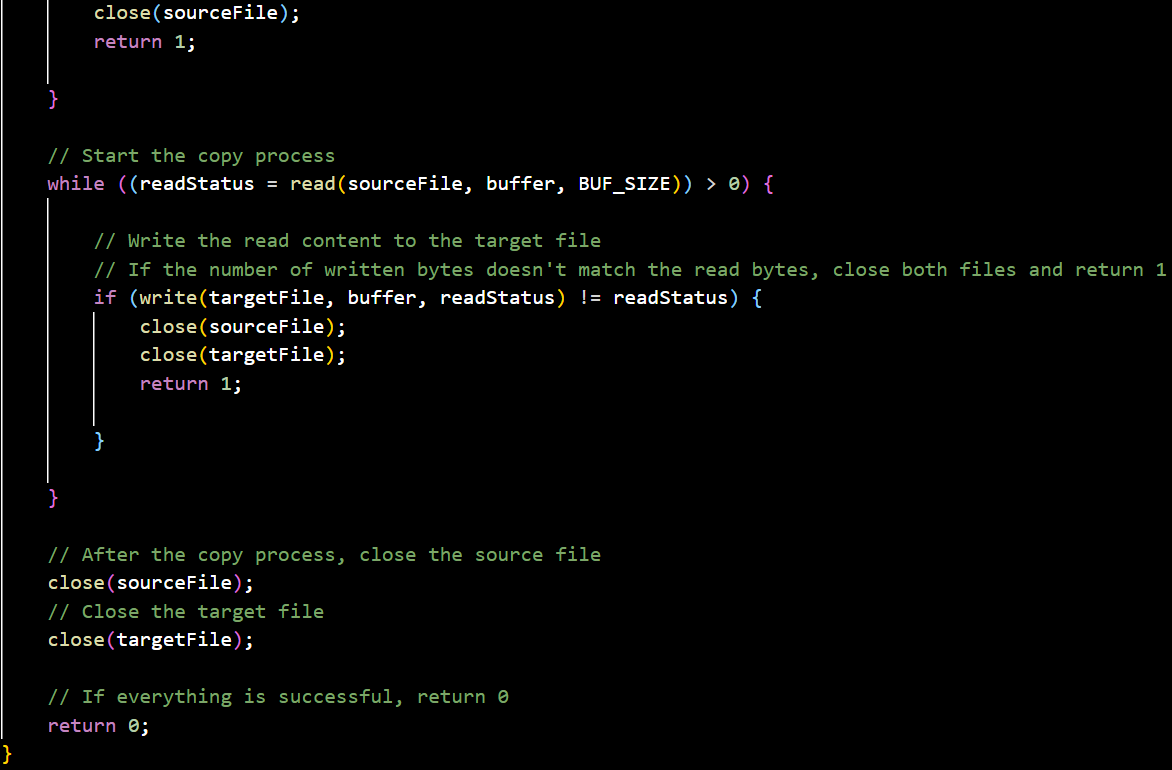
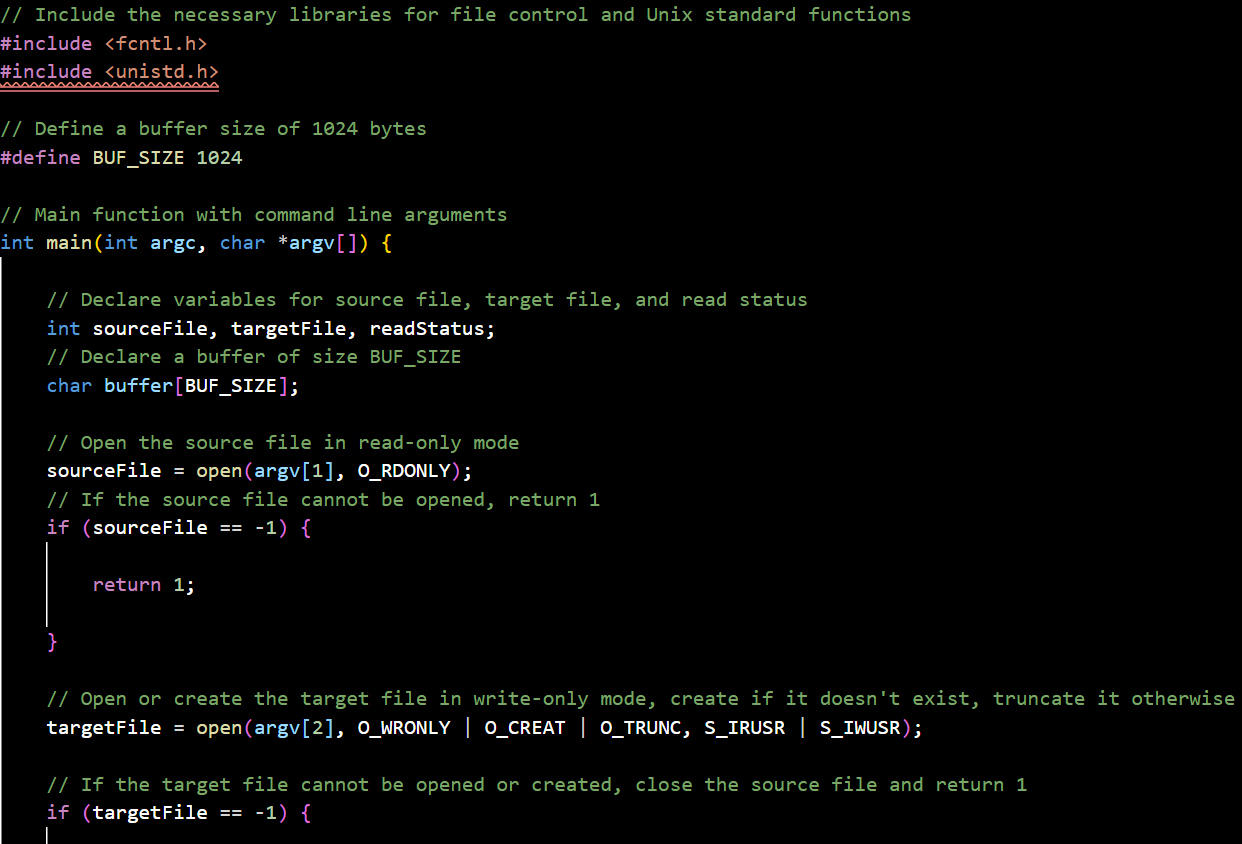
**O\_CREAT:** This flag is used to create a file if it does not exist.

1. **S\_IRUSR:** This mode sets read permission for the owner of the file. On many systems, this bit is 0400.

**S\_IWUSR:** This mode sets write permission for the owner of the file. Usually, this bit is 0200.

**Exercise 1.2**





**Exercise 2.1**

1. writes ‘count’ bytes from the buffer ‘buff’ directly to the standard output (usually the terminal). This system call bypasses any additional formatting or interpretation that higher-level functions like ‘printf’ might apply, making it suitable for raw data output.
2. Using a single unnamed pipe for bidirectional communication is generally not possible due to the inherent design of pipes. A pipe, created with the pipe(int pipefd[2]) function, is unidirectional. This means data can only flow in one direction—from the write-end (pipefd[1]) to the read-end (pipefd[0]).

Attempting to use the same pipe for both reading and writing in both directions would lead to confusion and possible deadlocks, as there’s no built-in mechanism to distinguish whether data is meant for reading or writing.

1. Unnamed pipes cannot be used to communicate between unrelated processes due to the following reasons:
2. Creation and Inheritance Mechanism:

**Pipe Creation:**

Unnamed pipes are created using the pipe(int pipefd[2]) system call, which provides two file descriptors: one for reading and one for writing.

**File Descriptor Inheritance:**

These file descriptors are only available in the creating process and its children.

When a process forks, the child process inherits the file descriptors from the parent process, allowing both processes to use the pipe for communication.

1. File Descriptor Scope:

**Process-Specific:**

File descriptors in Unix-like systems are specific to the process in which they are created. They are not globally accessible and cannot be shared directly with unrelated processes.

**Lack of a Common Handle:**

Unnamed pipes lack a global identifier or handle that can be passed to other unrelated processes to access the pipe.

1. Inter-Process Communication (IPC) Requirements:

**Shared Memory or Named Objects:**

Unrelated processes typically need to use named IPC mechanisms such as named pipes (FIFOs), message queues, shared memory, or sockets.

Named pipes (created using mkfifo) provide a file path that unrelated processes can use to open and communicate through the pipe.