1. fd = creat("Fillename" , permissions);  
   fd = creat("Marvellous.txt" , 0777);

if fd == -1 failed to create a file;

1. fd = open(filename , mode);  
   fd = open(Marvellous.txt , O\_RDONLY);  
   modes = O\_RDONLY readonly  
   O\_WRONLY write only  
   O\_RDWR read and write.  
    if fd == -1 failed to create a file.
2. Close(fd);
3. Write(fd , ArrayName or string to be written in the file , How many bytes to write in the file)  
   return how many bytes are written inside the file successfully.

**OPEN MODES:**

**🔸 O\_APPEND**

**Meaning:**  
When you write to the file, data is always added at the **end**, even if you move the file pointer.

**Example:**  
File data.txt contains:

Hello

Code:

int fd = open("data.txt", O\_RDWR | O\_APPEND);

write(fd, "World", 5);

**Result in file:**

HelloWorld

✅ Appended **at the end**, not overwritten.

**🔸 O\_CREAT**

**Meaning:**  
Creates the file **if it doesn't exist**.

**Example:**

int fd = open("newfile.txt", O\_RDWR | O\_CREAT, 0644);

* If newfile.txt doesn’t exist → it gets created.
* 0644: File permission (owner can read/write; others read only).

**✅ Combined Example**

int fd = open("log.txt", O\_RDWR | O\_APPEND | O\_CREAT, 0644);

write(fd, "Log Entry\n", 10);

* Creates log.txt if missing.
* Appends "Log Entry" to the end.

Let me know if you want a working code with file creation and append logic.

**fd = open(Name , O\_RDWR | O\_APPEND);**

if we opened the file in append mode the data which we are going to write in that file will be written after the previous contents of the file .

if we don’t open file in append mode the previous data will be overwritten by the new data.

**iRet = read(fd ,Data , sizeof(Data));//1**

**printf("%d bytes read succesfully\n",iRet);//2**

**Data[iRet] = '\0';// setting the end //3**

Here when we are read () system call is used to read the data form the file and we have to pass 3 parameters first the fd(file discriptor) of the opened file 2nd the empty array to store the read data and 3rd parameter specifies how many bytes we want to read;

Read system call returns the numbers of bytes successfully read from the file which we have stored int iRet

So at the 3rd line we are explicitly adding the delimiter at the end of string so that while printing the string no garbage values should be displayed.

**unlink(Name);**

this is unlink system call use to delete the file we just have to pass the name of the file as a parameter.

**fd = open(Name, O\_RDONLY); Name.txt = 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16**

**lseek(fd , 5 , SEEK\_SET);**

**read(fd , Arr , 10);**

**Short Explanation:**

* fd = open(Name, O\_RDONLY);  
  ➤ Opens the file **Name.txt** in **read-only** mode.
* lseek(fd, 5, SEEK\_SET);  
  ➤ Moves the file pointer to **5th byte** from the beginning.
* read(fd, Arr, 10);  
  ➤ Reads **10 bytes** from that position into Arr.

**Example Output from File Content:**

If file has: 1 2 3 4 5 6 7 8 9 10 11...  
Pointer at 5th byte → likely at character '6'  
So, it reads: '6 7 8 9 10 11 12 13 14 15' (10 bytes, depends on spacing).

**lseek(fd , 5 , SEEK\_CUR);**

**read(fd , Brr , 10);**

**printf("Data from the file is : %s\n",Brr)**

**Short Explanation:**

* lseek(fd, 5, SEEK\_CUR);  
  ➤ Moves the file pointer **5 bytes ahead** from the **current position**.
* read(fd, Brr, 10);  
  ➤ Reads **10 bytes** from that new position into Brr.
* printf("Data from the file is : %s\n", Brr);  
  ➤ Prints the 10 bytes as a string.

**Example:**

Assume file: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

If the file pointer was at byte 0 initially, then:

* After lseek(fd, 5, SEEK\_CUR); → pointer now at **byte 5** (likely character '6')
* read() reads 10 bytes → '6 7 8 9 10 11' (depending on spacing)
* Output:
* Data from the file is : 6 7 8 9 10 11

(*Note: exact characters depend on actual file content and spacing.*)

Here is the **list of system calls used**, along with their **modes/flags**:

**🔹 System Calls and Their Usage Modes**

1. **creat(filename, permissions)**
   * Creates a new file or truncates if exists.
   * **Permissions:** e.g., 0777, 0644
2. **open(filename, mode)**
   * Opens a file.
   * **Modes:**
     + O\_RDONLY – Read only
     + O\_WRONLY – Write only
     + O\_RDWR – Read and write
     + O\_APPEND – Append mode (writes at end)
     + O\_CREAT – Create file if it doesn't exist
     + Combined: O\_RDWR | O\_APPEND | O\_CREAT
3. **close(fd)**
   * Closes the opened file descriptor
4. **write(fd, buffer, size)**
   * Writes data to file
   * Returns number of bytes written
5. **read(fd, buffer, size)**
   * Reads data from file
   * Returns number of bytes read
6. **lseek(fd, offset, whence)**
   * Moves the file pointer
   * **Whence Modes:**
     + SEEK\_SET – From beginning
     + SEEK\_CUR – From current position
     + SEEK\_END – From end
7. **unlink(filename)**
   * Deletes the file

Let me know if you want a summary table too.

Welcome Mr. Shinde, I'm here for you.

Here’s a **complete list of system calls used in File Handling in C (Linux/Unix-based systems)** along with their brief descriptions and usage modes:

**🔧 Basic File Handling System Calls in C**

| **System Call** | **Description** | **Example Use** |
| --- | --- | --- |
| creat() | Creates a new file or rewrites an existing one | fd = creat("file.txt", 0777); |
| open() | Opens a file for reading/writing | fd = open("file.txt", O\_RDONLY); |
| read() | Reads data from a file | read(fd, buffer, size); |
| write() | Writes data to a file | write(fd, buffer, size); |
| close() | Closes an opened file | close(fd); |
| lseek() | Moves the file pointer to a desired position | lseek(fd, offset, SEEK\_SET); |
| unlink() | Deletes a file | unlink("file.txt"); |
| ftruncate() | Truncates a file to a specific length | ftruncate(fd, length); |
| dup() | Duplicates a file descriptor | new\_fd = dup(fd); |
| dup2() | Duplicates to a specific file descriptor | dup2(oldfd, newfd); |
| fsync() | Forces a flush of file buffers to disk | fsync(fd); |
| fcntl() | Performs various control operations on a file | fcntl(fd, F\_GETFL); |
| stat() | Retrieves file attributes | stat("file.txt", &statbuf); |
| fstat() | Retrieves attributes of a file using file descriptor | fstat(fd, &statbuf); |
| chmod() | Changes permissions of a file | chmod("file.txt", 0644); |
| chown() | Changes ownership of a file | chown("file.txt", new\_owner, new\_group); |
| access() | Checks file accessibility | `access("file.txt", F\_OK |

**📁 Common Flags Used in open() System Call**

| **Flag** | **Description** |
| --- | --- |
| O\_RDONLY | Open for reading only |
| O\_WRONLY | Open for writing only |
| O\_RDWR | Open for reading and writing |
| O\_CREAT | Create file if it doesn't exist |
| O\_APPEND | Append on each write |
| O\_TRUNC | Truncate file to zero length |
| O\_EXCL | Error if file already exists |

**🔍 Example Snippet:**

#include <fcntl.h>

#include <unistd.h>

#include <stdio.h>

int main() {

int fd = open("example.txt", O\_CREAT | O\_WRONLY, 0777);

if (fd < 0) {

perror("Open failed");

return 1;

}

write(fd, "Hello, Mr. Shinde\n", 18);

close(fd);

return 0;

}