1 Open System call

Open system call is used for opening a file.
int open(const char ****pathname*, int** *flags***, mode_t** *mode***);**

1. pathname is a file name
2. The argument *flags* must include one of the following *access modes* **O_RDONLY**, **O_WRONLY**, or **O_RDWR**. These request opening the file in read-only, write-only, or read/write modes, respectively. Apart from above, flags can also have any of the following:

(A) O_APPEND (file is opened in append mode)
(B) O_CREAT (If *pathname* does not exist, create it as a regular file.)
(C) O_EXCL Ensure that this call creates the file: if this flag is specified in conjunction with **O_CREAT**, and *pathname* already exists, then **open**() fails.

Note: to use two flags at once use bitwise OR operator, i.e., O_WRONLY | O_CREAT

3. **Mode is only required when a new file is created and is used to set permissions on the new file**

```
S_IRWXU  00700 user (file owner) has read, write, and execute
         permission

S_IRUSR  00400 user has read permission

S_IWUSR  00200 user has write permission

S_IXUSR  00100 user has execute permission

S_IRWXG  00070 group has read, write, and execute permission

S_IRGRP  00040 group has read permission

S_IWGRP  00020 group has write permission

S_IXGRP  00010 group has execute permission

S_IRWXO  00007 others have read, write, and execute permission

S_IROTH  00004 others have read permission

S_IWOTH  00002 others have write permission

S_IXOTH  00001 others have execute permission
```

On success, the open system call returns a file descriptor. File reading and writing then can be done using read and write system calls, respectively. On failure, -1 is returned.

2 Read System call read - read from a file descriptor

```
#include <unistd.h>
ssize_t read(int fd, void *buf, size_t count);
```

read() attempts to read up to count bytes from file descriptor fd into the buffer starting at buf. On files that support seeking, the read operation commences at the file offset, and the file offset is incremented by the number of bytes read. If the file offset is at or past the end of file, no bytes are read, and read() returns zero.
On success, the number of bytes read is returned (zero indicates end of file), and the file position is advanced by this number. It is not an error if this number is smaller than the number of bytes requested; this may happen for example because fewer bytes are actually available right now. On error, -1 is returned.

3 Write System Call

```
#include <unistd.h>
ssize_t write(int fd, const void *buf, size_t count);
```

write() writes up to count bytes from the buffer starting at buf to the file referred to by the file descriptor fd. The number of bytes written may be less than count if, for example, there is insufficient space on the underlying physical medium. On success, the number of bytes written is returned. On error -1 is returned

4 Close System Call

It closes the file descriptor so that the file descriptor no longer refers to any file anymore and can be reused.

```
#include <unistd.h>
int close(int fd);
```