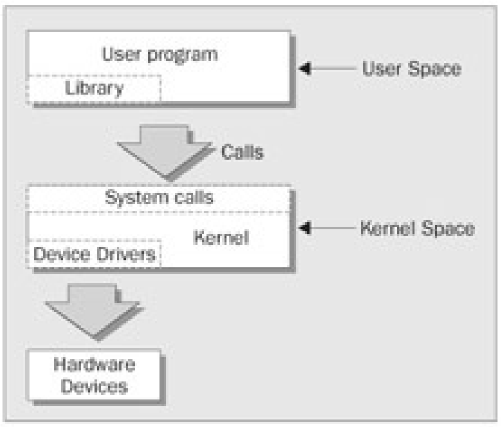
File Input Ouput Operations

System Calls :

The files and devices can be accessed using a small functions, These functions are known as system calls. They are provided by Linux directly and they are the interface to the operating system itself.



Low−level File Access

Each running program, called a process, has associated with it a number of file descriptors. These are small integers that can be used to access open files or devices. How many of these are available will vary depending on how the Linux system has been configured. When a program starts, it usually has three of these descriptors already opened. These are:

0	Standard Input
1	Standard Output
2	Standard Error

Other file descriptors are associated with files and devices by using the open system call

open

To create a new file descriptor the open system call is used.

```
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
int open(const char *path, int oflags);
int open(const char *path, int oflags, mode_t mode);
```

In simple terms, open establishes an access path to a file or device. If successful, it returns a file descriptor that can be used in read, write and other system calls.

Open returns the new file descriptor (always a non−negative integer) if successful, or −1 if it fails, open also sets the global variable errno to indicate the reason for the failure.

The file descriptor is unique and isn't shared by any other processes that may be running. If two programs have a file open at the same time, they maintain distinct file descriptors. If they both write to the file, they will continue to write where they left off. Their data isn't interleaved, but one will overwrite the other. Each keeps its own idea of how far into the file (the offset) it has read or written.

The name of the file or device to be opened is passed as a parameter, path, and the oflags parameter is used to specify actions to be taken on opening the file. The oflags are specified as a bitwise OR of a mandatory file access mode and other optional modes. The open call must specify one of the following file access modes:

Mode	**Description**
O_RDONLY	Open for read−only
O_WRONLY	Open for write−only
O_RDWR	Open for reading and writing

The call may also include a combination (bitwise OR) of the following optional modes in the oflags parameter:

O_APPEND	Place written data at the end of the file.
O_TRUNC	Set the length of the file to zero, discarding existing contents.
O_CREAT	Creates the file, if necessary, with permissions given in mode.
O_EXCL	Used with O_CREAT, ensures that the caller creates the file. The open is atomic, i.e. it's performed with just one function call. This protects against two programs creating the file at the same time. If the file already exists, open will fail.

Other possible values for oflags are documented in the open manual page, found in section 2 of the manual (use man 2 open).

Initial Permissions:

When a file is created using the O_CREAT flag with open, three parameter form is used. mode, the third parameter, is made from a bitwise OR of the flags defined in the header file sys/stat.h. These are:

- **S_IRUSR -** Read permission, owner.
- **S_IWUSR** - Write permission, owner.
- **S_IXUSR** - Execute permission, owner.
- **S_IRGRP** - Read permission, group.
- **S_IWGRP** - Write permission, group.
- **S_IXGRP** - Execute permission, group.
- **S_IROTH** - Read permission, others.

- **S_IWOTH** - Write permission, others.
- **S_IXOTH** - Execute permission, others.

Write:

```
#include <unistd.h>
size_t write(int fildes, const void *buf, size_t nbytes);
```

The write system call arranges for the first nbytes bytes from buf to be written to the file associated with the file descriptor fildes. It returns the number of bytes actually written. This may be less than nbytes if there has been an error in the file descriptor. If the function returns 0, it means no data was written, if −1, there has been an error in the write call and the error will be specified in the errno global variable.

Read:

```
#include <unistd.h>
size_t read(int fildes, void *buf, size_t nbytes);
```

The read system call reads up to nbytes bytes of data from the file associated with the file descriptor fildes and places them in the data area buf. It returns the number of data bytes actually read, which may be less than the number requested.
If a read call returns 0, it had nothing to read; it reached the end of the file. Again, an error on the call will cause it to return −1.

Close:

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

close is used to terminate the association between a file descriptor, fildes, and its file. The file descriptor becomes available for reuse. It returns 0 if successful and −1 on error. Note that it can be important to check the return result from close.

Ioctl:

```
#include <unistd.h>
int ioctl(int fildes, int cmd, ...);
```

ioctl provides an interface for controlling the behavior of devices, their descriptors and configuring underlying services. Terminals, file descriptors, sockets, even tape drives may have ioctl calls defined for them and you need to refer to the specific device's man page for details.
ioctl performs the function indicated by cmd on the object referenced by the descriptor fildes. It may take an optional third argument depending on the functions supported by a particular device.