Preliminary of Files in Linux

- Each file have below file permissions.

  Read(r),write(w),execute(x)

  Read has ‘4’ units(2 power 2)

  Write has ‘2’ units (2 power 1)

  Execute has ‘1’ units (2 power 0)
- Each file can be accessed by different users as follows

  User, group, others
- Each user type can have r/w/x access for file

Types of Files

- - : regular file (data files).
- d : directory.
- c : character device file.
- b : block device file.
- s : local socket file.
- p : named pipe.
- l : symbolic link.

File Input Output

The focus of this topic is exploring the system calls used for performing file input and output operations Below concepts will be covered with examples.

1. File descriptors
2. System calls involved in file operations
3. Open()
4. Read()
5. Write()
6. Close()
7. Lseek()

File descriptor

- All system calls for performing I/O refer to open files using a file descriptor, a (usually small) nonnegative integer.

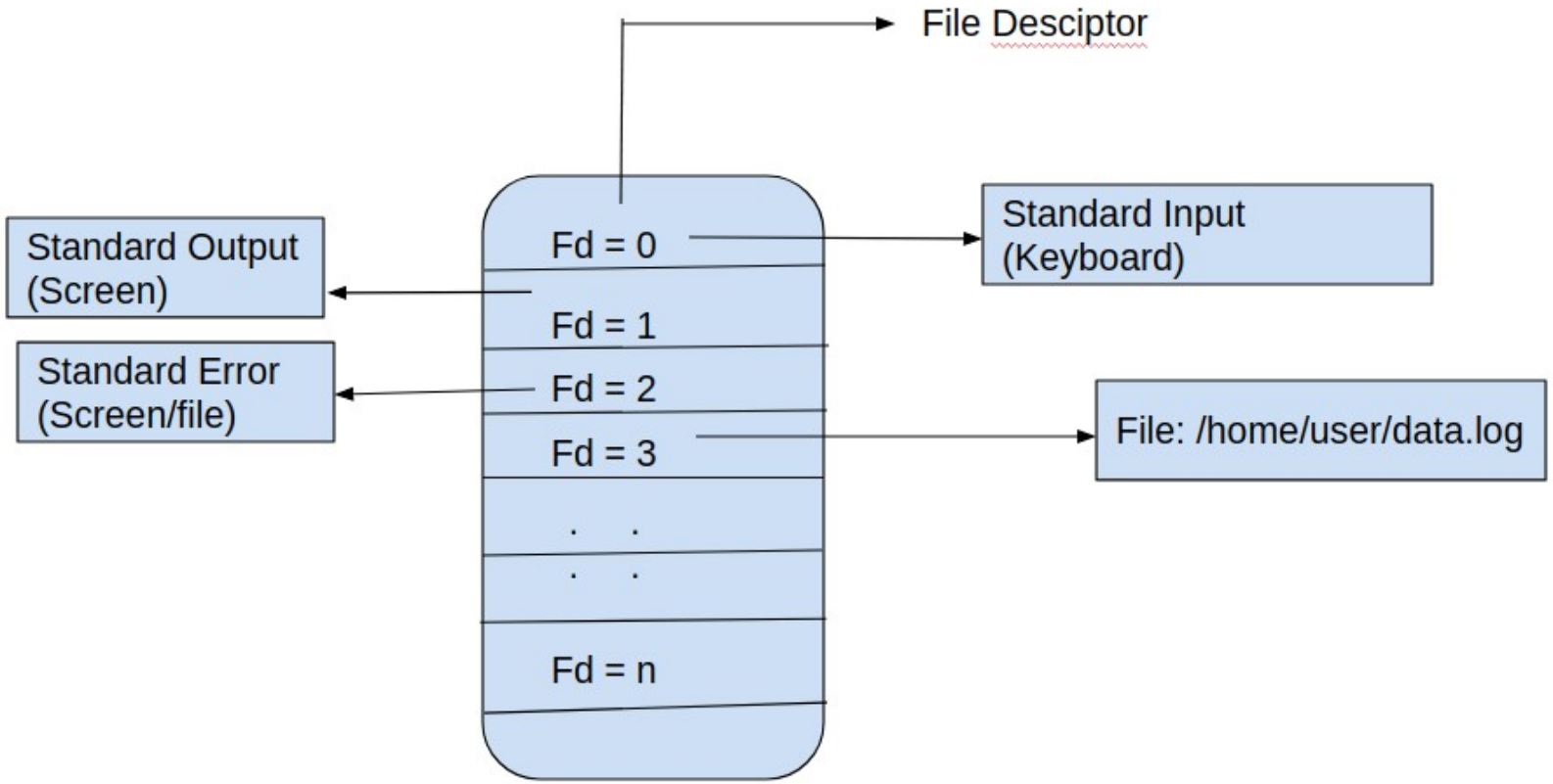
- All file related operations are performed via file descriptor(fd)

- There are 3 standard file descriptor by default available in Linux.

File Descriptor	Description	Posix name	Stdio stream
0	Standard input	STDIN_FILENO	stdin
1	Standard output	STDOUT_FILENO	stdout
2	Standard error	STDERR_FILENO	stderr

Open() - System call

- In order to perform any operation like reading/writing a file, we need to open the file and provide a handle to the kernel. Any further operations on this file, will be done using this handle.



Open()

- Syntax of open() in C language

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

Or

```
int open(const char * pathname , int flags , ... /* mode_t mode */);  // generic description
```

Parameters of Open()

- Path : path to file which you want to open the file.
- Flags :

  1. The Flag should contain atleast one of O_RDONLY,
     O_WRONLY , O_RDWR along with other flags.
     O_RDONLY : read only,
     O_WRONLY : write only,
     O_RDWR : read and write.
  2. O_CREAT: create file if it doesn't exist.
     When open() is used to create a new file.
     If the open() call doesn't specify O_CREAT , mode can be omitted.

- Mode: Specifies the file creation with access permissions.

Open() in C

Open() - return value

- If an error occurs, open() returns –1.
- errno is set accordingly

The errors that occur which are most common are

EACCES, EEXIST, EISDIR

Fd returned by open()

- If open() succeeds, it returns least non zero value. This value is called file descriptor.
- All operations(read,write) on this file opened is made via this descriptor.

- Lets us open the ‘Linux Manual’ page for open().

Creat() system call

- In early implementations, open() system call was not used to create new file, instead creat() system call was used.

  #include <fcntl.h>

  int creat(const char * pathname , mode_t mode);

  Returns file descriptor, or –1 on error

- Calling creat() is equivalent to the following open() call:
- fd = open(pathname, O_WRONLY | O_CREAT | O_TRUNC, mode);

  Note : creat() is now obsolete, although it may still be seen in older programs.

Read() - system call

- The read() system call reads data from the open file referred to by the descriptor fd.

  #include <unistd.h>

  ssize_t read(int fd , void * buffer , size_t count);

  (size_t - unsigned integer)

- The buffer argument supplies the address of

  the memory buffer into which the input data is to be placed. This buffer must be at least ‘count’ bytes long

- Note : Read() does not allocate any memory, user must allocate memory and pass to read().

- A successful call to read() returns the number of bytes actually read, or 0 if end-of-file is encountered. On error, the usual –1 is returned.

  Refer to C program on read() func call

- Note : read() system calls are applied on files like regular files, PIPES, sockets, FIFO

Write() - system call

- The write() system call writes data to an opened file.

  #include <unistd.h>

  ssize_t write(int fd, void * buffer , size_t count);

  Returns number of bytes written, or –1 on error
- The arguments to write() are similar to those for read(): buffer is the address of the data to be written on File.
- count is the number of bytes to write to file from buffer; and fd is a file descriptor referring to the file to which data is to be written.

- The buffer argument supplies the address of the memory buffer containing input data. This data is written to file specified by ‘fd’. This buffer must be at least ‘count’ bytes long.

- A successful call to write() returns the number of bytes actually written to file, On error –1 is returned.

  Note: The actual bytes written to file may also be lesser than ‘count’ bytes specified in 3rd parameter of write() system call.

Close() - system call

- The close() system call closes an open file descriptor, freeing it for subsequent reuse by the process.
- When a process terminates, all of its open file descriptors are auto-matically closed.

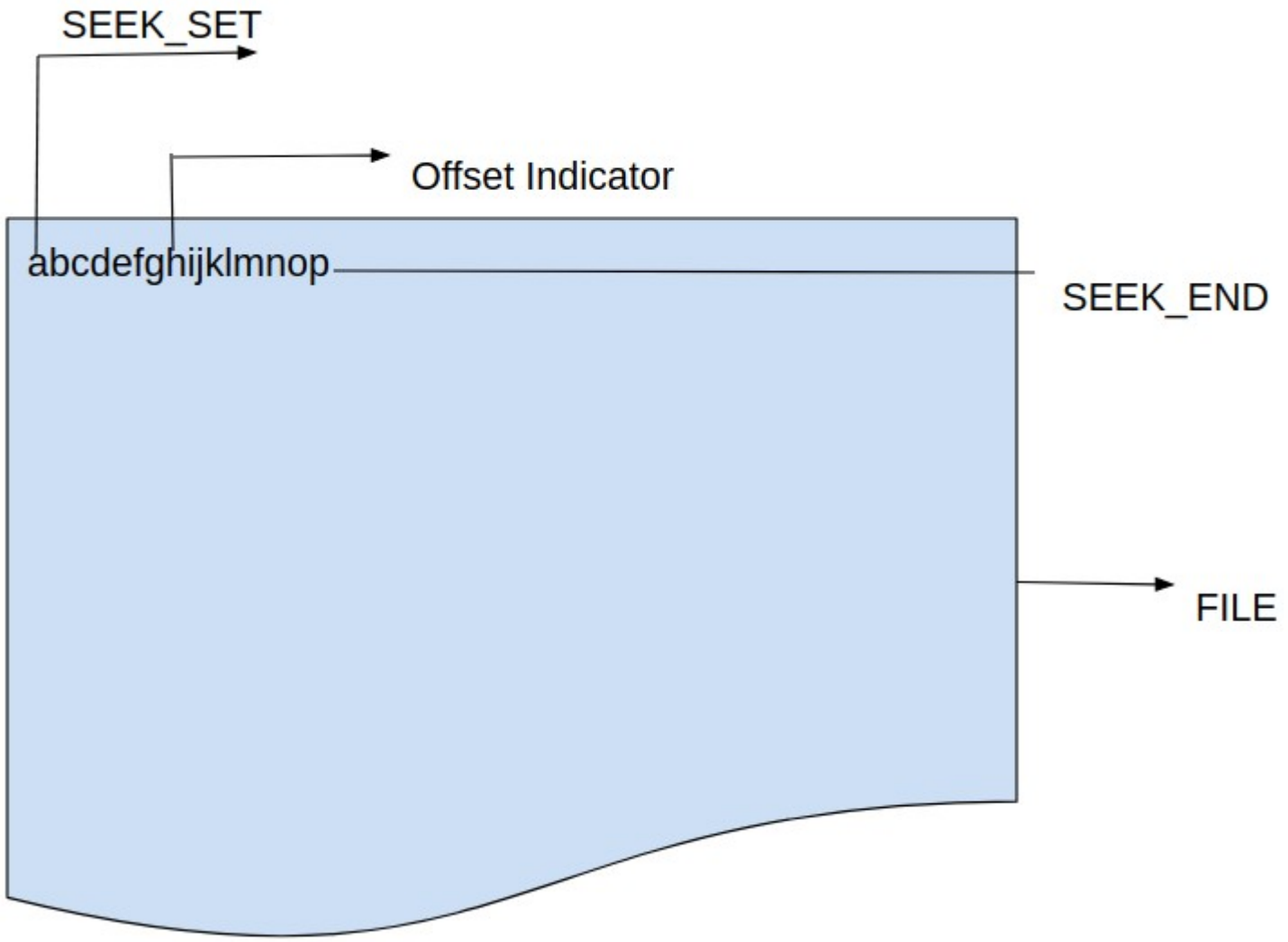
Lseek() - system call

- A file can be considered as a continuous set of bytes.
- There is a internal indicator present, which points to the offset byte of the file. This offset is used to read / write the next set of bytes/data from file. This indicator is updated when we do any file operation like read() or write().
- lseek is a system call that is used to change the location of the read/write pointer of a file descriptor. The location can be set either in absolute or relative terms.

- #include <unistd.h>

  off_t lseek(int fd , off_t offset , int whence);

  Returns new file offset if successful, or –1 on error


SEEK_SET
Offset Indicator
abcdefghijklmnop
SEEK_END
FILE

Lseek()

- fd : The file descriptor of the file.
- off_t offset : The offset of the pointer (measured in bytes).
- int whence : The method in which the offset is to be interpreted

  SEEK_SET - The offset is set to offset bytes.

  SEEK_CUR - The offset is set to its current location plus offset bytes.

  SEEK_END - The offset is set to the size of the file plus offset bytes.
- return value : Returns the offset of the pointer (in bytes) from the beginning of the file. If the return value is -1, then there was an error moving the pointer.