```
int open(const char* path_name, int flags, ...);
ssize_t read(int fd, void* buffer, ssize_t buffer_size);
ssize_t write(int fd, void* buffer, ssize_t buffer_size);
int close(int fd);
#----------------------------
const char* path_name = "/home/yogeshwar/src/abc.txt"
int fd = open(path_name, O_RDONLY);
assert(fd != -1);
#---------------
pqr.txt does not exist in /home/yogeshwar/src
const char* path_name = "/home/yogeshwar/src/pqr.txt";
int fd = open(path_name, O_WRONLY | O_CREAT, S_IRUSR | S_IWUSR | S_IRGRP);
assert(fd != -1);
#---------------
pqr.txt may or may not exist in /home/yogeshwar/src
if does not exist -> creat it
if exists -> truncate the existing content and open it

int fd = open(path_name, O_WRONLY | O_CREAT | O_TRUNC, S_IRUSR | S_IWUSR |
              S_IRGRP | S_IROTH);
assert(fd != -1)
#---------------------
hello.c is present in src.
You want to open it to overwrite its content
const char* path_name = "/home/yogeshwar/src/hello.c"
int fd = open(path_name, O_WRONLY);
#---------------------
hello.c is present in src.
You want to append data to hello.c
You want to ensure that whenever a write system call is issued, it (= write)
will make sure to seek the next read write offset to the end of the file
and then commit a write
int fd = open(path_name, O_WRONLY | O_APPEND);
assert(fd != -1)
#----------------
ssize_t pread(int fd, void* buffer, size_t size, off_t offset);
ssize_t pwrite(int fd, void* buffer, size_t size, off_t offset);

OFFSET RANGE -> offset to offset + size -1 -> bmap()

off_t start_offset = 8 * 1024;
ssize_t size = 4 * 1024;
char buffer[4096];
```

```
pread(fd, buffer, size, start_offset);
buffer will contain a data from 8K to 12K-1 (assuming file is at least as long
as 12K bytes)

void copy(const char* src_path, const char* dest_path){
    int fd_src, fd_dest;
    off_t src_offset = 0, dest_offset = 0;
    ssize_t read_bytes;
    ssize_t written_bytes;

    fd_src = open(src_path, O_RDONLY);
    fd_dest = open(dest_path, O_RDWR | O_CREAT | O_TRUNC | O_APPEND,
                    S_IRUSR | S_IWUSR | S_IGRP | S_IWGRP);
    char* buffer = (char*)malloc(4096);

    while((read_bytes = pread(fd_src, buffer, 4096, src_offset)) > 0){
        written_bytes = pwrite(fd_dest, buffer, read_bytes, dest_offset);
        src_offset += read_bytes;
        dest_offset += written_bytes;
    }

    close(fd_src);
    close(fd_dest);
    free(buffer);
}
#-------------------------------

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

whence -> SEEK_SET, SEEK_CUR, SEEK_END

If N bytes is the size of file in bytes.
SEEK_SET -> OFFSET 0
SEEK_END -> OFFSET N
SEEK_CUR -> CURRENT POSITION
offset positive -> RIGHTWARDS
offset negative -> LEFTWARDS

START                CUR                END
[ ------>    <------ | ------>    <------ ] ------>
```

LSEEK PATTERNS:

Move the next read write offset at the start of the file.

```
lseek(fd, 0, SEEK_SET);
```

Position next read write offset at the end of the file.

```
lseek(fd, 0, SEEK_END);
```

(better use O_APPEND so that job of this lseek call is internally performed by write() avoiding the race condition)

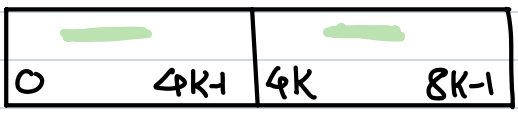
Find out where the current offset is

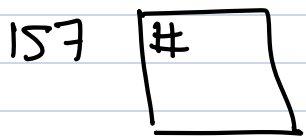
```
off_t offset = lseek(fd, 0, SEEK_CUR);
```

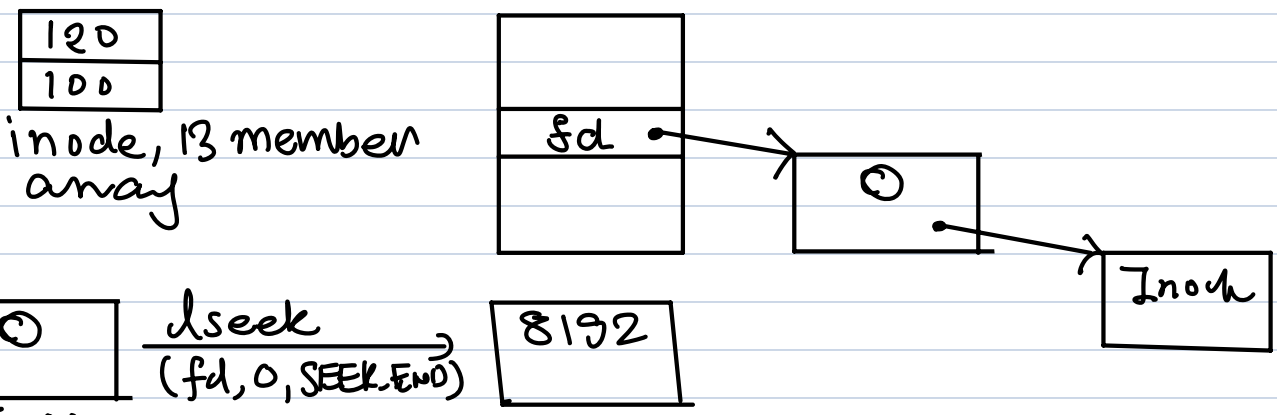
```
void read_test(const char* path_name){
    int fd = open(path_name, O_RDONLY);
    off_t current_offset;
    char* buffer = (char*)malloc(4096);
    ssize_t read_bytes;
    while((read_bytes = read(fd, buffer, 4096)) > 0){
        current_offset = lseek(fd, 0, SEEK_CUR);
        printf("Current offset = %lu\n", current_offset);
        write(STDOUT_FILENO, buffer, read_bytes);
    }
}
```

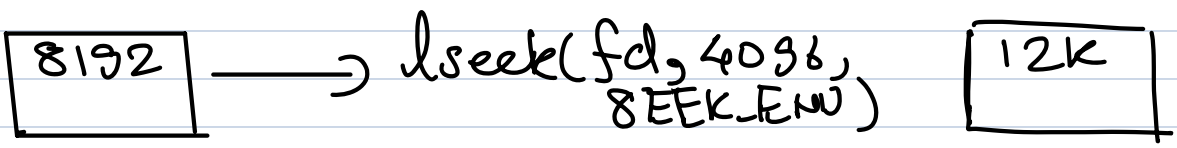
```
void read_test(const char* path_name){
    int fd = open(path_name, O_RDONLY);
    off_t current_offset;
    char* buffer = (char*)malloc(4096);
    ssize_t read_bytes;
    off_t off_read = 0;
    while((read_bytes = pread(fd, buffer, 4096, off_read)) > 0){
        current_offset = lseek(fd, 0, SEEK_CUR);
        printf("Current offset = %lu\n", current_offset);
        off_read += read_bytes;
        write(STDOUT_FILENO, buffer, read_bytes);
    }
}
```

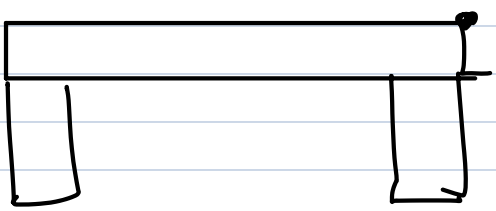
```
/* Assume that
    1) LOGICAL BLOCK SIZE == 4K
    2) File is being processed only by this application (no race condition)
    3) Initial size of file = 8K
*/
void test(const char* path_name){
    int fd = open(path_name, O_RDWR);
    off_t offset = lseek(fd, 0, SEEK_END);
    printf("END OFFSET == SIZE OF FILE = %lu\n", offset);
    offset = lseek(fd, 4096, SEEK_END);
    write(fd, "#", 1); /* Write # character to file */
    close(fd);

}
```



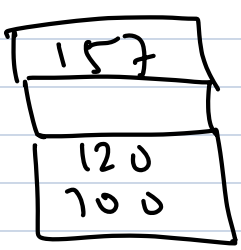








#



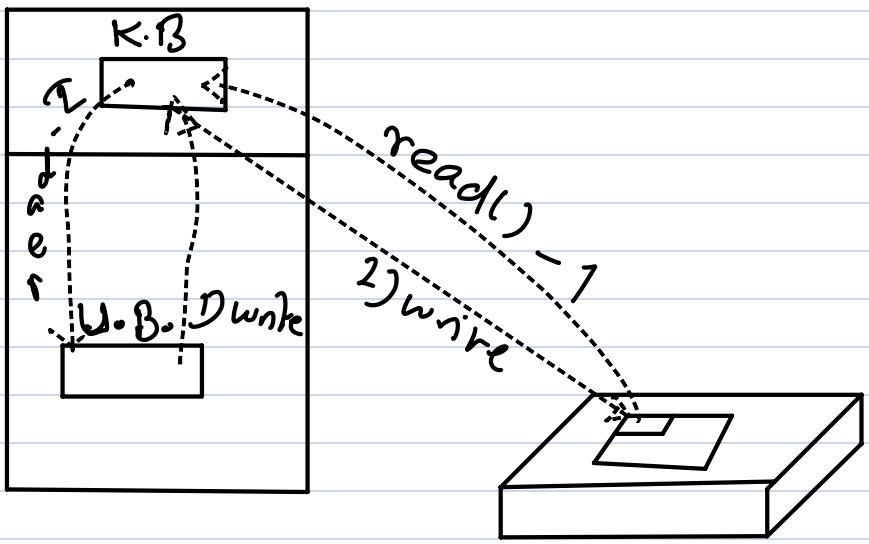
12K = #

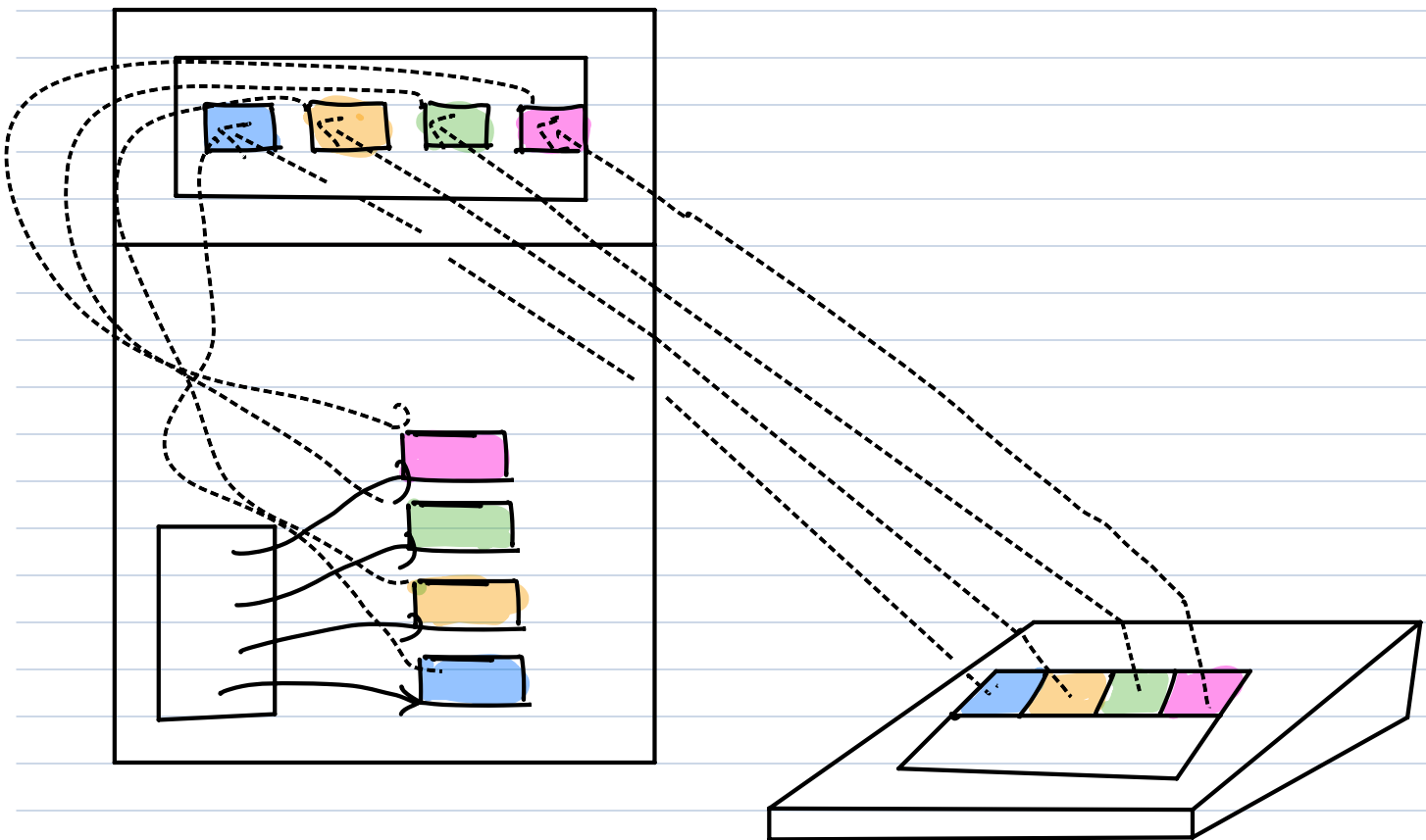
---

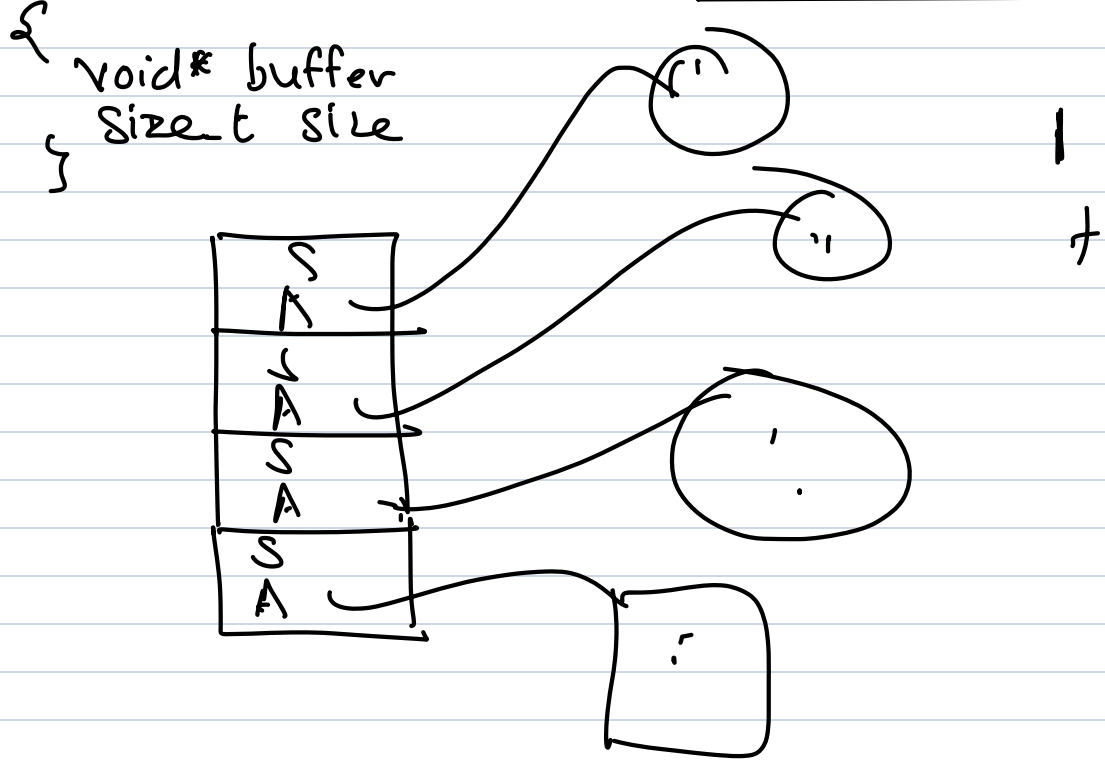
readv()/ writev()

SCATTER / GATHER - I/O

```
ssize_t readv(int fd, const struct iovec* iov, int iovcnt);
ssize_t writev(int fd, const struct iovec* iov, int iovcnt);
```








```
// BIG FILE
void test(const char* path_name){
   static struct iovec iov_arr[4];
   int i;

   for(i = 0; i < 4; ++i){
      iov_arr[i].iov_base = (char*)malloc(4096);
      iov_arr[i].iov_len = 4096;
   }

   int fd = open(path_name, O_RDONLY);
   assert(fd != -1);

   int read_bytes = readv(fd, iov_arr, 4);
   // 16 KB have been read
   // first 4 KB -> iov_arr[0].iov_base
   // second 4 KB -> iov_arr[1].iov_base
   // third 4 KB -> iov_arr[2].iov_base
   // fourth 4 KB -> iov_arr[3].iov_base
   for(i = 0; i < 4; ++i)
      printf("ARRAY(%d):%s\n", i, (char*)iov_arr[i].iov_base);
   for(i = 0; i < 4; ++i)
      free(iov_arr[i].iov_base);
}
```

open(), read(), write(), lseek(), close(), pread(), pwrite(),
readv(), writev(), preadv(), pwritev()
#----------------------------

fcntl(), ioctl(),
dup(), dup2(), dup3()

#--------------------------

fork(), execve(), wait(), _exit()

#----------------------

Basic signal