First of all, this machine problem only simply contains mainly two parts, one is for file system, and one is for file. Basically, the file system offers the functions for file class to implement and use. The simple idea would be implemented FileSystem class with inode to implement File class (FileSystem->inode->File), which would make the whole machine problem not that complicated. In the inode class, I made up the inode list that contains the file id, file information, and construct the inode into single list. In the file system, just keeping tracking the block size, free blocks, and inode list. The file class contains size, position, block information, and file id.

Simply, the inode just need to store file information and written it to the disk, and wu need to update all the information here.

For the file system, Mount function would associate the file system with a disk. Limit to at most one file system per disk. Basically, it would update all the information of the disk, which would be very similar to the function of constructors. I simply update the disk, and disk size at the beginning, then I set free blocks and initialize inode here. the format function just assigns the disk information to wipe the file system from the disk and install an empty file system of given size. The lookup function is just simply checking if the given file has already been existed, just see if they have the same file id, and return the inode in formation. CreateFile and DeleteFile is very simply to create file with the file id, delete all the assign new value.

```
bool FileSystem::CreateFile(int _file_id) {
    Console::puts("creating file with id:"); Console::puti(_file_id); Console::puts("\n");
    /* Here you check if the file exists already. If so, throw an error.
       Then get yourself a free inode and initialize all the data needed for the
       new file. After this function there will be a new file on disk. */
    // assert(false);
    Inode* cur = inodes;
    while (cur->next != NULL){
        cur = cur->next;
    }
    cur->next = new Inode(_file_id);
    cur = cur->next;
    cur->file = new File(this, _file_id);
    return true;
}
```

```
bool FileSystem::DeleteFile(int _file_id) {
    Console::puts("deleting file with id:"); Console::puti(_file_id); Console::puts("\n");
    /* First, check if the file exists. If not, throw an error.
       Then free all blocks that belong to the file and delete/invalidate
       (depending on your implementation of the inode list) the inode. */
    Inode* cur = inodes->next;
    Inode* pre = inodes;
    while (cur != NULL && cur->id != _file_id) {
        cur = cur->next;
        pre = pre->next;
    }
    if (cur == NULL) {
        return true;
    }
    File* f = cur->file;
    Block* current_block = f->head->next;
    while (current_block != NULL){
        free_blocks[current_block->b_no] = 0;
        current_block = current_block->next;
    }
    pre->next = cur->next;
    delete cur;
    delete f;
    return true;
}
```

For the file class, after finishing inode and file system, this just implements function to read buffer, write buffer, check EOF, and reset etc. The file constructor would initialize all the variables of file. In the read, I constructed 512 bytes buffer to issue the read operation with the current position of the file. For the write, I checked the EOF first, and write into the buffer that I need to update the block data. The main read and write codes are screen shot here.

Read here

```
unsigned int begin = position;
unsigned char* buffer = new unsigned char[512];
// read file until nothing to read
while (position < size && position - begin < _n) {
    int block_num = position / 512;
    int offset = position - block_num * 512;
    Block* cur = head->next;
    for (int i = 0; i < block_num; i++) {
        cur = cur->next;
    }
    FS->disk->read(cur->b_no, buffer);
    int j = 0;
    while (position < size && position - begin < _n && j < 512) {
        // _buf[position - begin] = buffer[j];
        // position++;
        // j++;
        _buf[position++ - begin] = buffer[j++];
    }
}
```

Write here

```
// write here
while (position - begin < _n) {
    // new block number
    unsigned long new_block_num = FS->assignBlock(id);
    Block* cur = head;
    while (cur->next != NULL) {
        cur= cur->next;
    }
    cur->next = new Block(new_block_num);
    cur->next->next = NULL;
    int j = 0;
    while (position - begin < _n && j < 512) {
        buffer[j++] = _buf[position++ - begin];
    }
    size += j;
    // Console::puts("L118:");
    // Console::puti(size);
    // Console::puts("\n");

    FS->disk->write(new_block_num, buffer);
}
```