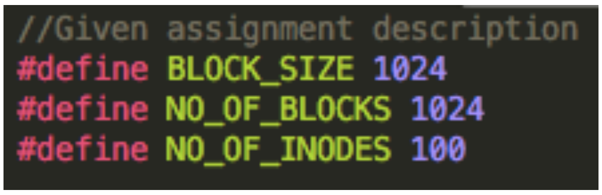
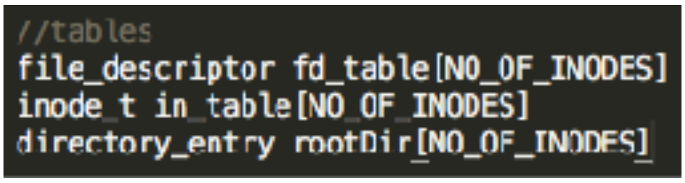
Void mksfs (int fresh)



```
//Given assignment description
#define BLOCK_SIZE 1024
#define NO_OF_BLOCKS 1024
#define NO_OF_INODES 100
```



```
//tables
file_descriptor fd_table[NO_OF_INODES]
inode_t in_table[NO_OF_INODES]
directory_entry rootDir[NO_OF_INODES]
```

There 2 tables that are very important , namely the file descriptor table and the Inode table.These tables can be implemented as by creating global arrays of their particular types present in the sfs_api.h . The size of array can be capped off with maximum number of Inodes that are allowed by the file system ie.100.

The **file descriptor table** always **resides in memory** ie its never written to the disk.Everytime the mksfs() function is called it should be initialised by **setting some member** (make wise decision) of the file_descriptor struct of every element in fd_table to -1. It would be better to design this as a new function say "init_fdt()" which initialises the global array.

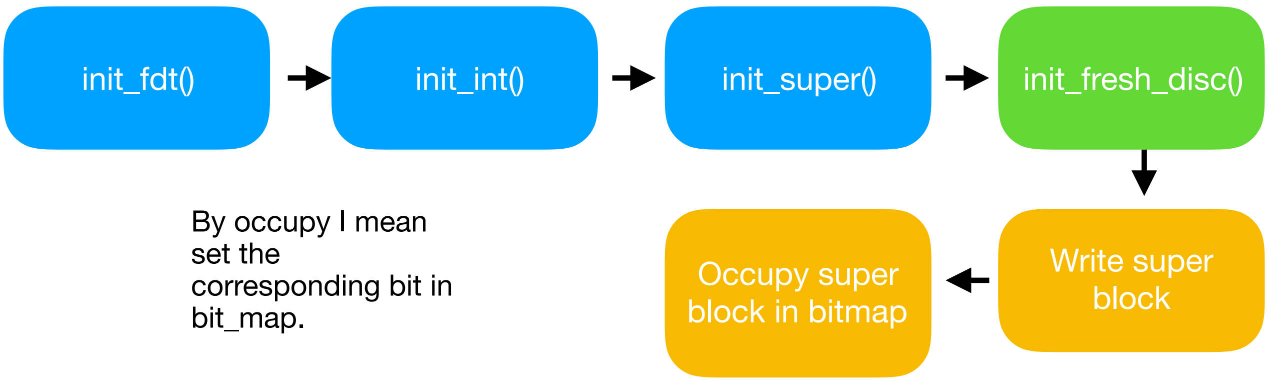
The **Inode table** is something that **resides both, in the memory and on the disk.** Similar to file_descriptor table even this table needs to be initialised with -1 . Be careful this is an Inode and it would be a better decision to set all the members of the inode struct to -1, for each element of the in_table global array.Again, you can have a function say "init_int()" for this.

The **rootDir** can have as many files as there are nodes so it makes sense to have a global array for this as well with a fixed size.Now this can be used to find out the number of blocks that the directory could take.Use the ciel value while calculating.

The **superblock** has a separate structure in sfs_api.h. It is beneficial to declare a global super block and a function say "init_super()" which initialised it structure with values you know.Donot worry about the magic number , you can set it 0xACBD0005.

1.If fresh is true

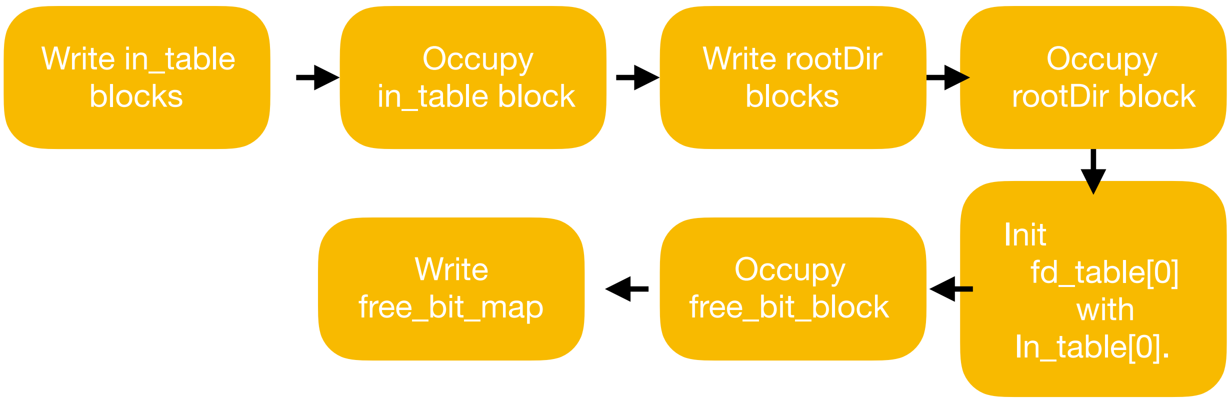
(Green functions are given , blue is func() you have to implement, yellow are steps)



By occupy I mean set the corresponding bit in bit_map.

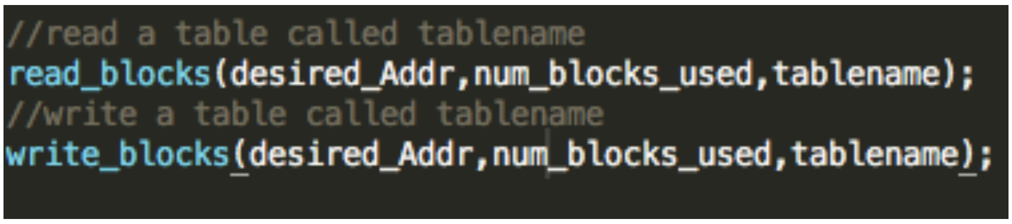
Once this is done , its time to prepare the first inode ie in_table[0] for root dir. Do not worry about different fields in inode structure ,setting only size and data pointer should be enough. The data pointers are basically the block number to which they point to . Run a loop for the number of blocks that rootDir will occupy and set to its corresponding location. Be careful to consider the initial offset ie your first block for root dir comes after blocks taken by superblock+ in_table(to be taken when we write it to disk).Donot fiddle with indirect pointer as of now. Directory easily fits in less than 12 blocks.Note that as you initialise these data pointer with block numbers .

Aakash Nandi Rola Harmouche



By this time you have a file_descriptor table , superblock , in_table and root dir in memory and a copy of superblock , in_table and root directory on the disk.

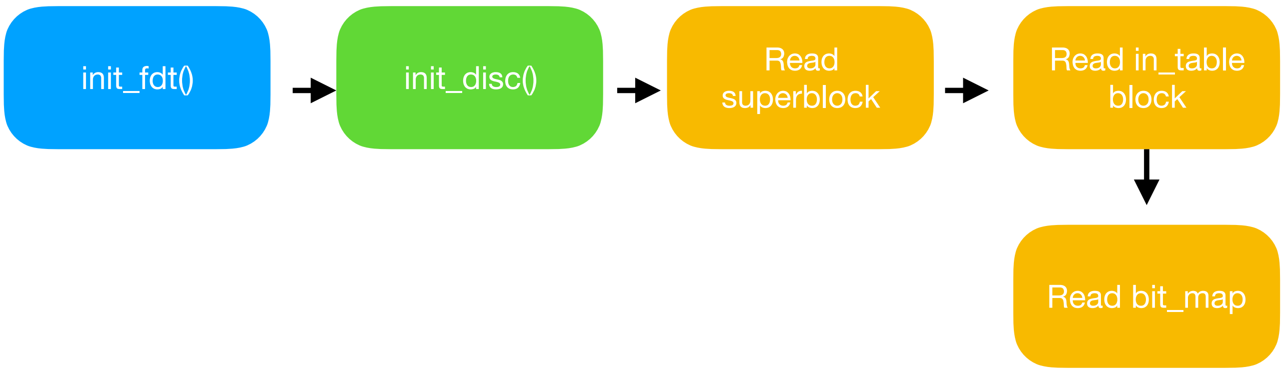
A common question asked is how to write/read the tables on to disk in form of blocks.
Writing a table say in_table is easy.Your table may not be of the size of integral multiples of blocks but specify the integral block size and your write_blocks and read_blocks takes care of it.Read blocks will read the integral blocks but take only enough bytes from buffer required to fill the table.It simply slices the extra garbage value due to left over block space.Same goes for the superblock as well.



```
//read a table called tablename
read_blocks(desired_Addr,num_blocks_used,tablename);
//write a table called tablename
write_blocks(desired_Addr,num_blocks_used,tablename);
```

2.If fresh is false

In short when fresh was true , take the burden of creating an superblock, empty directory,inode table and bitmaps etc and push it to disk but when fresh is **false** simple read superblock,in_table and bitmap into their corresponding global variables.



Aakash Nandi Rola Harmouche