Valeria Frolova
V00920747
CSC 360, Assignment 3 – April 6, 2020

File System Idea

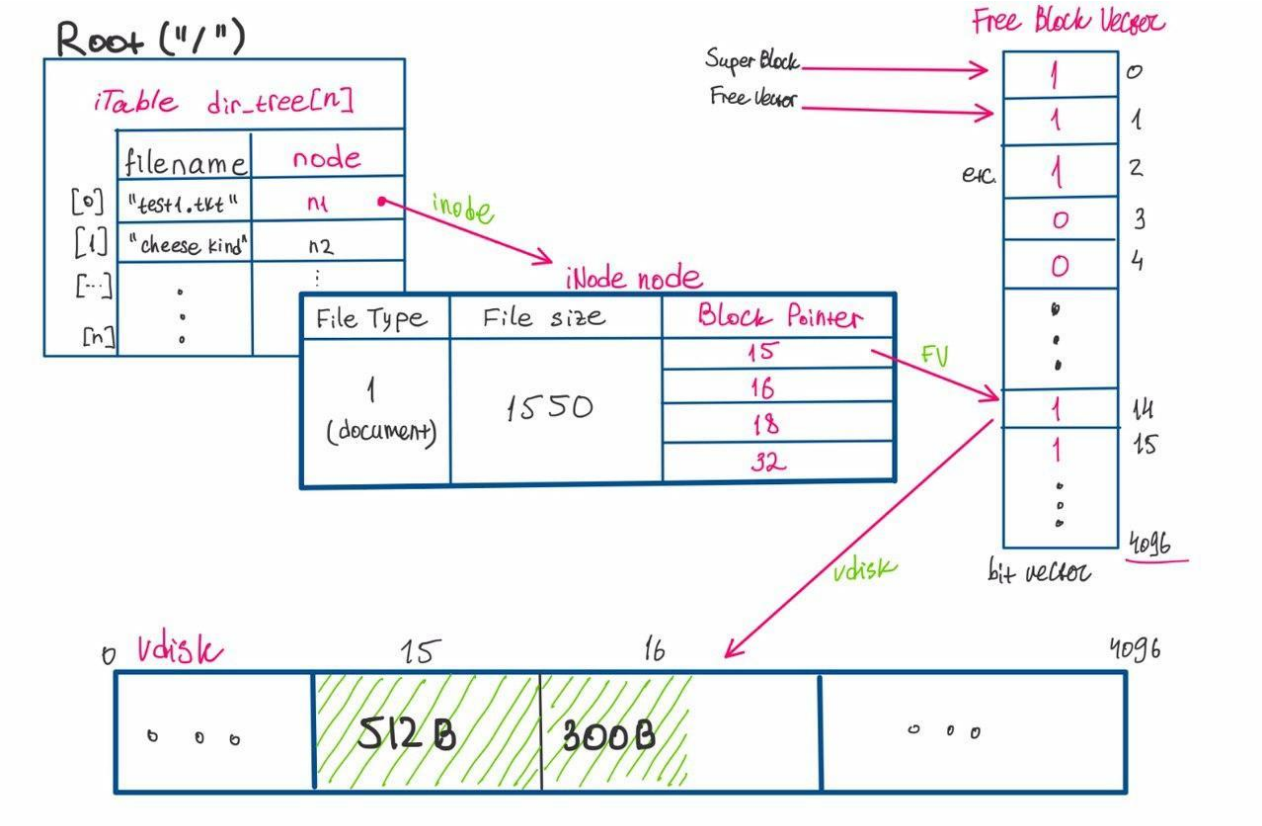
Disk Implementation (*disk.c*)

The ***disk.c*** creates ***vdisk*** if it doesn't exist *or* opens the existing one. Aside from reading and writing blocks methods, there is a disk checker that prevents wrong usage or allocation. Disk termination method simply closes the disk.

Reading and writing is performed via ***fseek*** where global variable for starting pointer begins its segmented 512-byte read or write. To separate blocks within ***vdisk***, the pointer is multiplied by the number of bytes. This way disk allocation is delivered and accessed in chunks.

File System Implementation (*File.c*)

The functionality is mainly delivered through structs the following way:



There is a checker flag in test files that finds if there exists such a file called ***vdisk***. If the disk flag is new, several blocks are pre-allocated for SuperBlock, root, inodes, free block vector, etc. These allocations are reflected on block vector. If the disk has been created before s.t. its file already exists, we simply open that file for manipulations. Next, the writing or reading of user content begins.

The **root** contains an index table **iTable dir_tree[n]** which is a mapping between an inode and a file name. Each inode in this directory table has a field:

- file type – 0 for directory, 1 for document
- file size, in bytes
- block pointer: if file is larger than 512 bytes, pointers are assigned for each data segment of a single file

In turn, this block pointer also serves as a starting pointer for read or write. Upon writing, this pointer is incremented such that a new, *next free available* block, may be located on the free block vector. Then, the pointer is mapped on the disk. Recall, a single *vdisk*'s 512B block has been converted to a 4096-bit **free block vector (FBV)** that serves as mapping (shown above).

The reading is done similar way, although we may only read allocated blocks, and those that belong to a file. This 'sanity check' is done via mapping in inode and FBV.

Although log-based system was not implemented, the idea was to keep updates buffered and copied onto a separate (say, undirected) block. This undirected block pointer would also be present in inode struct and mapped onto FBV. We cannot keep *all* updates on a disk except for the information about it being done. Keeping all deletions or additions result in a huge amount of information, although modern systems are capable of it. In our implementation we could keep at least 2 most recent updates and let them be written to a disk if our main data segment gets corrupted at some point. How would a system know that it is corrupted? My assumption is that buffered segment may not match main segment. Obviously, there was some intervention, so the system copies everything from that buffer to a corresponding main block on the disk.