Minix File System – File Layout

- The Minix FS is a logical, self-contained entity with i-nodes, directories and data blocks. Example of a 360K layout with 127 i-nodes and 1K block size. Figure 5.30.

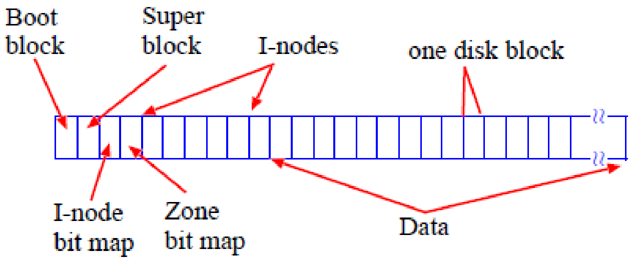


Figure 5.30: Minix FS Layout

- **Boot block**. When a computer is turned on, the hardware reads the boot block into memory and jump to it.

Super Block

- **super-block** contains information about file system layout. Figure 5.31.

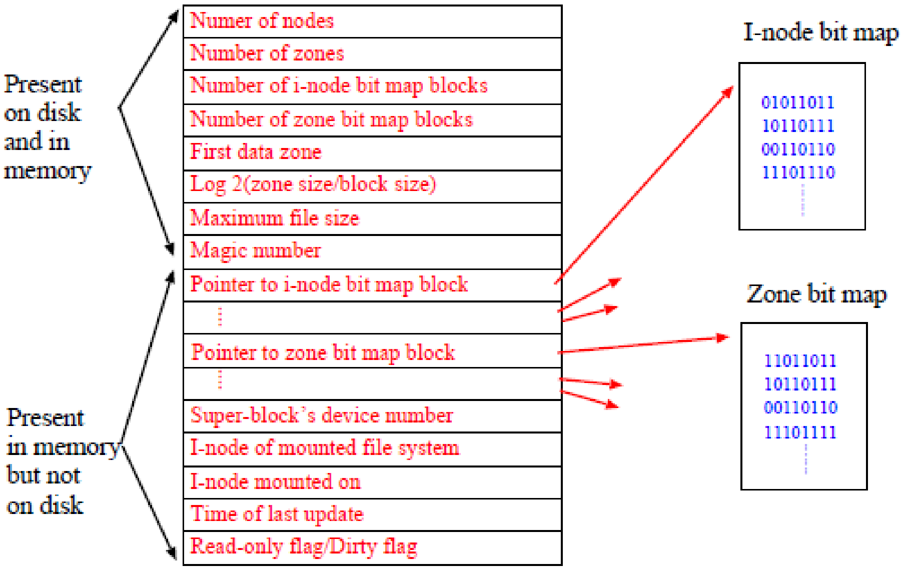
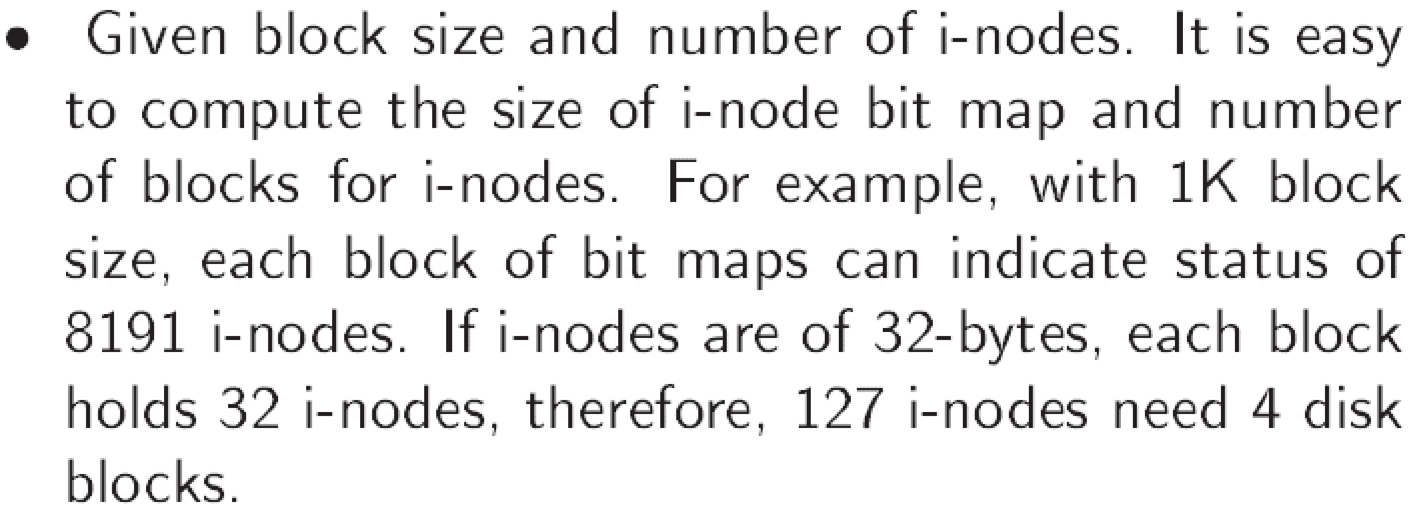

Present on disk and in memory
Present in memory but not on disk
Numer of nodes
Number of zones
Number of i-node bit map blocks
Number of zone bit map blocks
First data zone
Log 2(zone size/block size)
Maximum file size
Magic number
Pointer to i-node bit map block
Pointer to zone bit map block
Super-block's device number
I-node of mounted file system
I-node mounted on
Time of last update
Read-only flag/Dirty flag
I-node bit map
01011011
10110111
00110110
11101110
Zone bit map
11011011
10110111
00110110
11101111

Figure 5.31: The MINIX super-block

Super Block



- Given block size and number of i-nodes. It is easy to compute the size of i-node bit map and number of blocks for i-nodes. For example, with 1K block size, each block of bit maps can indicate status of 8191 i-nodes. If i-nodes are of 32-bytes, each block holds 32 i-nodes, therefore, 127 i-nodes need 4 disk blocks.

Super Block

- disk storage is allocated in units (zones) of 1,2,4,8 or in general, 2^n blocks.

- mapping from zone to block or vice versa can be done by algorithm. For example, with 8 blocks per zone, to find zone containing block 128, just shift 128 right by 3 bits.

- when Minix is booted, the super-block for the root device is read into a table called super-blocks table. New parameters are then derived and stored in this table like whether it has been mounted read-only, modified status field.

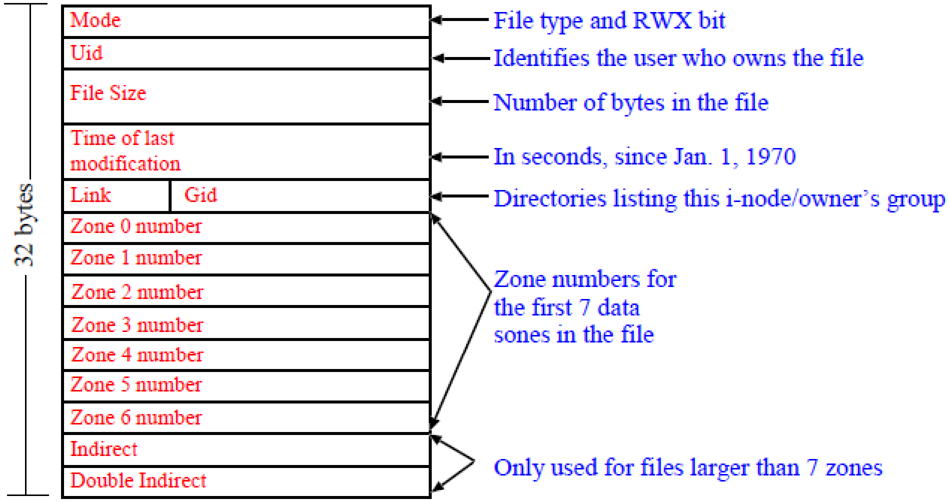
- To enforce a known structure, the utility program *mkfs* is provided to build a file system.

Bitmap

- i-nodes and zone bit maps keep track of status.
- upon boot up, super-block and bit maps for the root device are loaded into memory.
- to remove a file, based on the i-node number, we will know which pointer in in the super-block to go to and access the i-node bit map and then clear the corresponding bit.
- to create a file, sequentially search through the bit map blocks until it finds a free i-node. This i-node is then allocated for the new file. If no available i-node, return 0.
- Rational for zone is to improve performance so that disk blocks that belong to the same file are located on the same cylinder as much as possible.

Inode

- in Minix, i-node is a 32-bytes structure. Figure 5.32.



32 bytes

Mode — File type and RWX bit

Uid — Identifies the user who owns the file

File Size — Number of bytes in the file

Time of last modification — In seconds, since Jan. 1, 1970

Link | Gid — Directories listing this i-node/owner's group

Zone 0 number

Zone 1 number

Zone 2 number

Zone 3 number

Zone 4 number

Zone 5 number

Zone 6 number

Zone numbers for the first 7 data sones in the file

Indirect

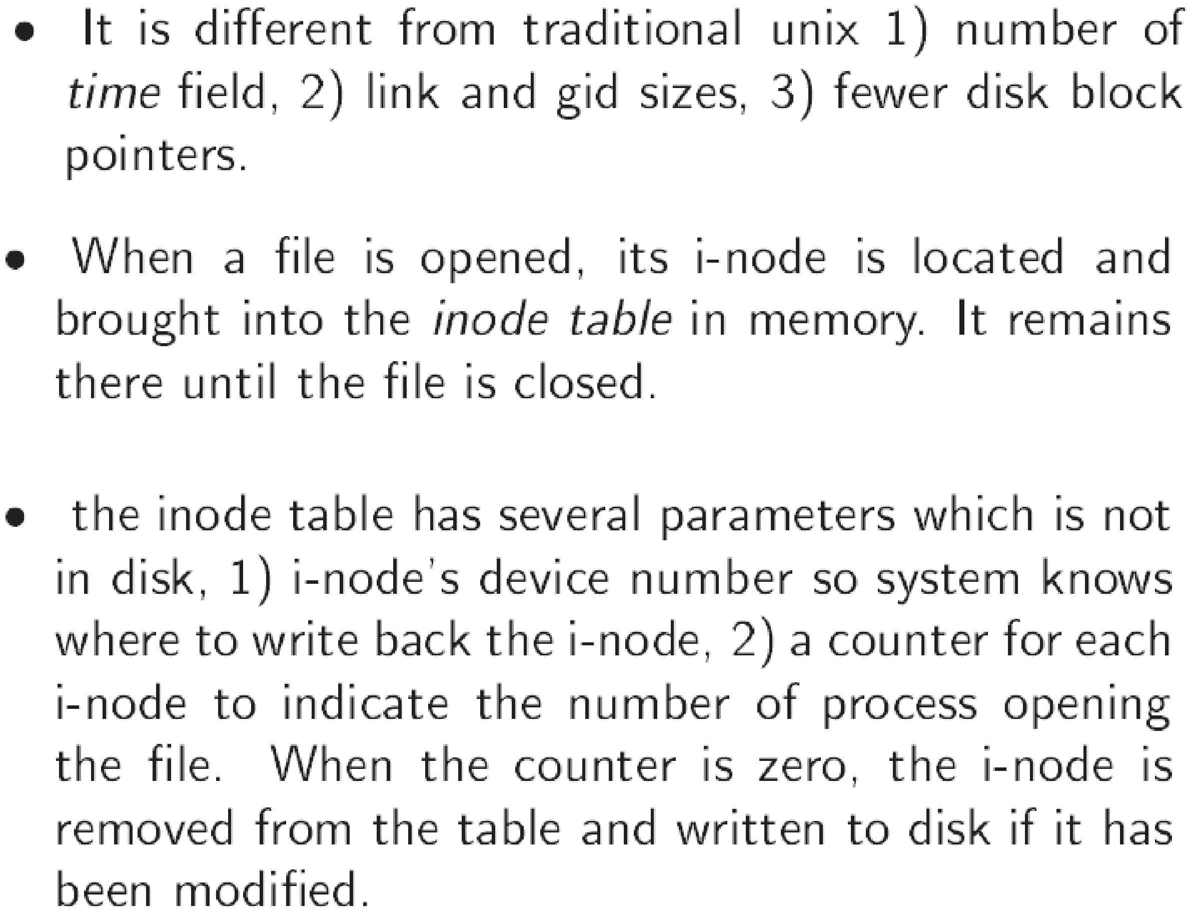
Double Indirect

Only used for files larger than 7 zones

Figure 5.32: The MINIX i-node

- *mode* indicates the type of file (regular, directory, block special, character special or pipe), protection bits.

Inode



- It is different from traditional unix 1) number of *time* field, 2) link and gid sizes, 3) fewer disk block pointers.
- When a file is opened, its i-node is located and brought into the *inode table* in memory. It remains there until the file is closed.
- the inode table has several parameters which is not in disk, 1) i-node's device number so system knows where to write back the i-node, 2) a counter for each i-node to indicate the number of process opening the file. When the counter is zero, the i-node is removed from the table and written to disk if it has been modified.

Directories and Path

- as mentioned before, when the system wants to open a file, it is actually accessing an i-node.

- file name look up is the same as we described before. The important thing is to find the root directory i-node, which is located in the fixed position of the disk.

- When user types in command
  % mount /def/fd1 /user
  this implies to mount the file system on top of /user.

Directories

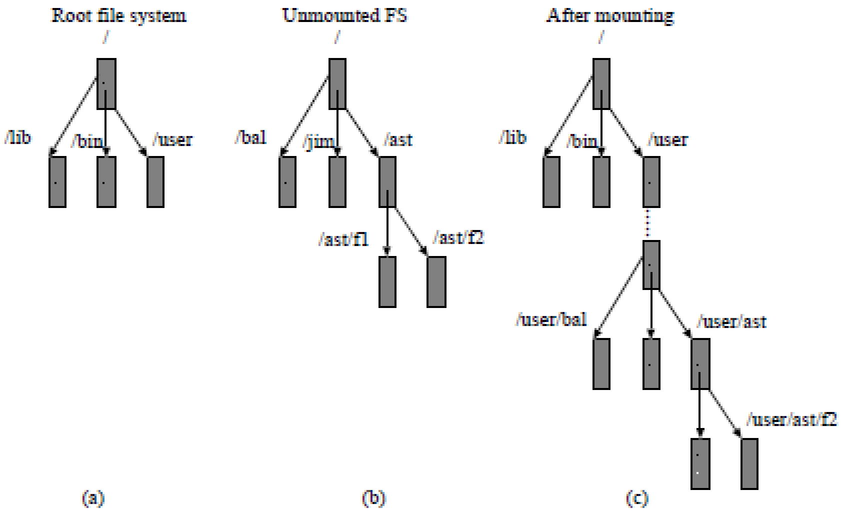


Figure 5.34: (a) Root file system. (b) An unmounted file system.
(c) The result of mounting the file system of (b) on /user

Directories

- What does it really mean for mounting? That is, during file name lookup, how does the system know when and where to switch?

- OS has to 1) set a flag in ”/user” to indicate a successful mount, 2) load the super-block and i-node of the newly mounted file system into super-block table and i-node table (this is accomplished by MOUNT call), 3) set the *i-node-of-the-mounted-file-system* to point to the root i-node of the newly mounted system, 4) set the *i-node-mounted-on* to point to the i-node of ”/user”.

- to answer the previous question, it is straight forward.

Superblock

```
struct minix_super_block {
        unsigned short s_ninodes;
        unsigned short s_nzones;
        unsigned short s_imap_blocks;
        unsigned short s_zmap_blocks;
        unsigned short s_firstdatazone;
        unsigned short s_log_zone_size;
        unsigned int s_max_size;
        unsigned short s_magic;
        unsigned short s_state;
        unsigned int s_zones;
};
```

Inode

```
struct minix_inode {
        unsigned short i_mode;
        unsigned short i_uid;
        unsigned int i_size;
        unsigned int  i_time;
        unsigned char i_gid;
        unsigned char i_nlinks;
        unsigned short i_zone[9];
};
```

Directory Entry

```
struct minix_dir_entry {
  unsigned short inode;
  char name[0];
};
```