Block 0

<table>
<thead>
<tr>
<th>name</th>
<th>block#</th>
<th>len</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>afile.txt</td>
<td>1</td>
<td>1000</td>
<td>F</td>
</tr>
<tr>
<td>mydir</td>
<td>3</td>
<td>512</td>
<td>D</td>
</tr>
</tbody>
</table>

 undergone.

```
root
+-- mydir
    +-- fileb.txt
```

Step 1: Read the root directory
Block 0
then look up 'my dir'

Step 2: Read 'my dir'
Block 3
then look up fileb.txt
then you can read 17 bytes of fileb.txt

→ look up "my dir"

(1) read root
(block 0)
→ look up "my dir"

(2) read "my dir"
(block 3)
→ look up fileb.txt

What will happen if we want to read:

cat /my dir/fileb.txt
How the file system handle their free space, how they associate and handle blocks.

Review: Old Unix file system
- Causes file system fragmentation.
- Linked free list (org. Unix) Approach 1
- High fragmentation & hence not efficient.

> Bitmap Approach 2
- When writing a file search for longer run of contiguous blocks.
- So bitmap search is another way of writing files, but when file system becomes big, bitmap method is not very efficient.

Berkeley Fast File System (FFS)
- MS-DOS (Sort of)

Extent list - Approach 3
- Keep a list of free ranges.
- Keep an ordered list
- When we free a block of range 25-35
- Then list will be:
  - 0-10
  - 20-25
  - 50-60
  - 75-150

We implemented this via a tree
- B-tree for ordered insert/lookup.

Efficient way of keeping track of regions in a free list.
Free List Handling
- linked list
  - bitmap
  - extemp list

How to keep track of blocks within a file?

* File Organization
  1. contiguous (most simple)

Practically it is never done in file syst, coz of overhead in paining also waste bytes to store this information, so we generally have separate array of pointers. Else we be playing havoc with virtual memory.

2. Linked List (File Allocation Table)
   - blocks
   - If you want to write a file then simply link free blocks to each other.
   - But a lot of pointer chasing and reading needs to be done if it's a big file syst.
   - Big file reading from beginning to end will be required to read many pointers to read the blocks of data

INDIRECT-BLOCK TREE
- i-node + indirect blocks — used in org. UNIX
- instead of just having just a pointer, you have something complex as in directory entry
4
5) Extent lists. 
(a, b) (c, d) ----
- we can represent a file as a list of contiguous
  ranges of blocks
- The ordering is by logical address
  logical [0 ... 199], [200 .. 299]
  physical [500 .. 699], [0 ... 99]
- This will be stored in a B-tree (representation)

File vs Name

weather there is distinction
between the name of a file
and the file itself

- when multiple locations
  point to the same file
  - hardlinks
File Attributes

- Length
- Timestamps
- Ownership
- Permissions