FSO – November 25, 2013

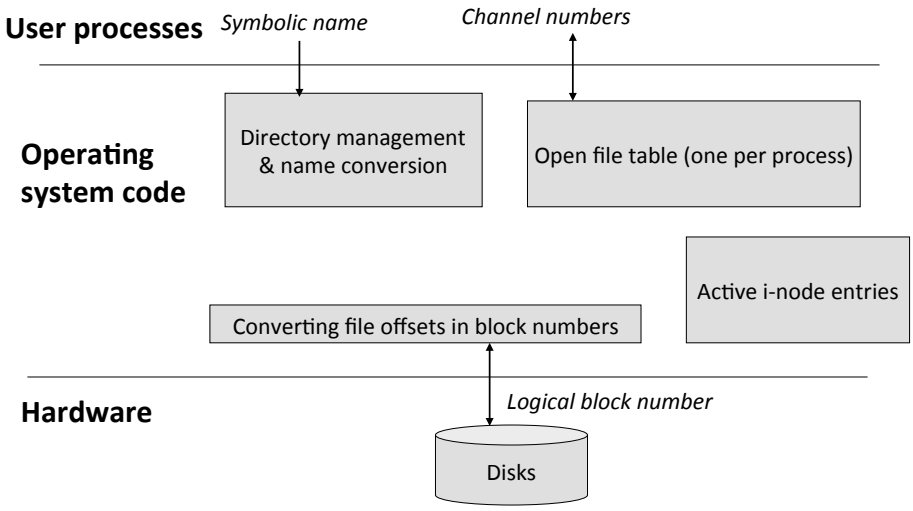
Mounting file systems.
Allocating disk blocks to files.
Block caching.

Bibliography: OSTEP book
- Chap 38: section 15
- Chap 39: sections 3 and 7

Main questions about file systems

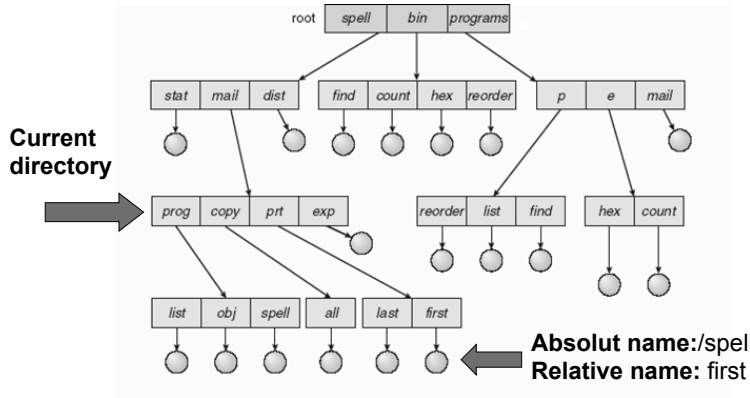
- How users name files
  - Name space organized as a tree; directories convert symbolic names
  - Integrating more than one disk in a single file system
- How files and directories are suported
  - Internal names are indexes in the i-node table stored in the disk
  - How to support directories
  - How to allocate disk blocks to files
  - How to accelerate read and write operations

File system organization



Symbolic file system [1]

- Organized as a tree
- Files have absolute and relative names



Symbolic file system [2]

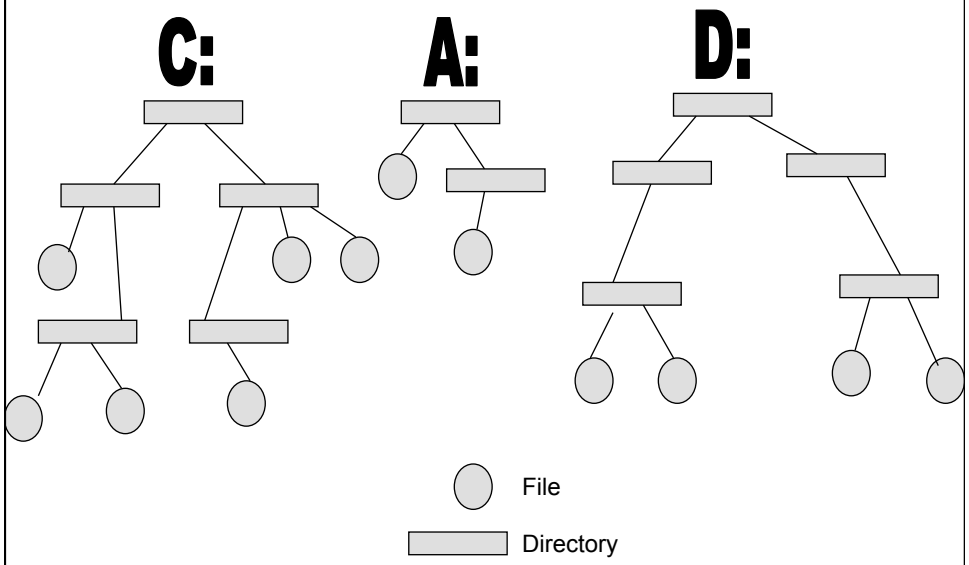
- Directories are stored in the disk as ordinary files (and have an index in the i-node table)
- Used to convert names into indexes in the i-node table

Type	Name	I-node
F	x.c	1209
D	progs	133
D	music	14551

Symbolic file system [3]

- Volume or logic disk: corresponds to a physical disk or to a part of it (partition)
- “Raw disk”: physically formatted (surfaces, tracks, sectors) but not logically. To be used one needs to put there management information (meta-data): format (windows), mkfs (unix/linux)
- Naming: A:, ... Windows; /dev/hda1, ... Linux
- How users view several logical disks ?

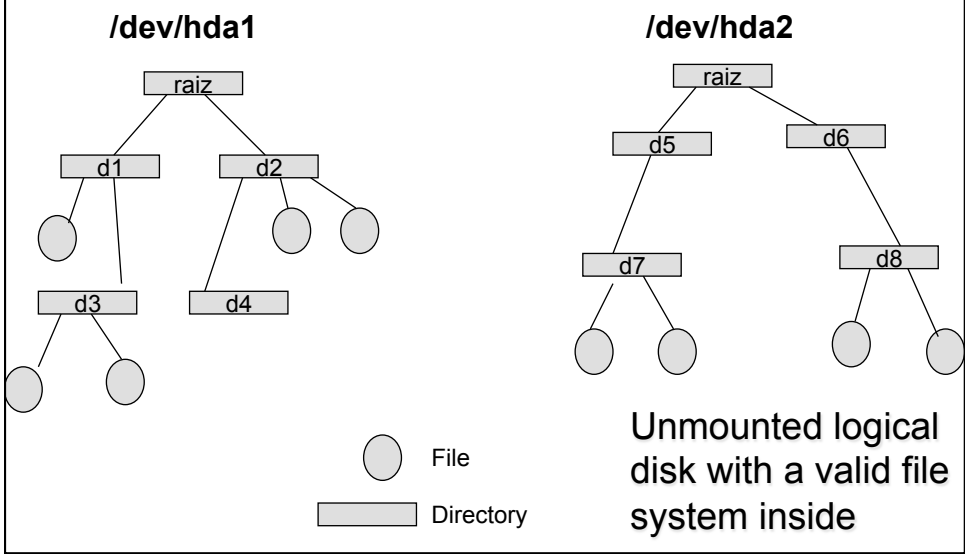
A tree for each logical disk



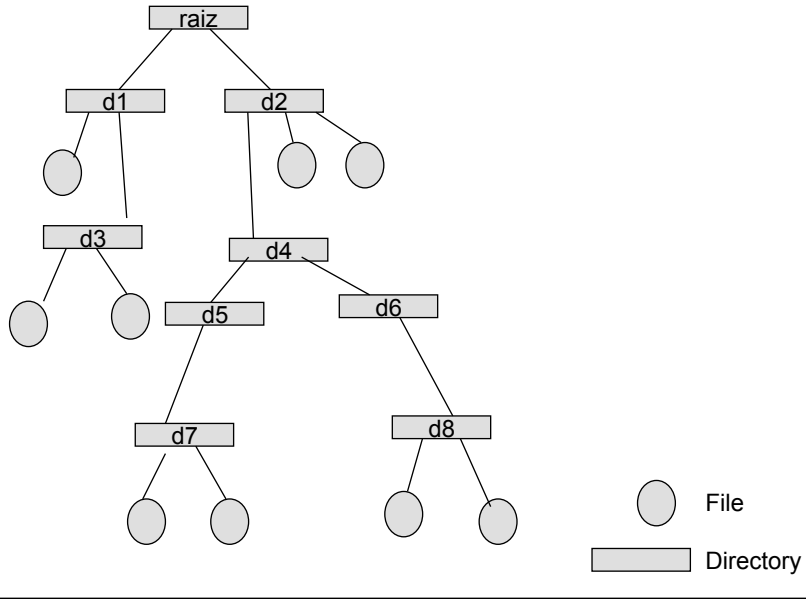
File System Mounting or a single tree

- There is a main file system (root file system)
- A file system (FS) must be **mounted** before becoming accessible.
- A non-mounted FS is placed (mounted) in a **mount point**.

Root FS /dev/hda1



After *mount /dev/hda2 /d2/d4*

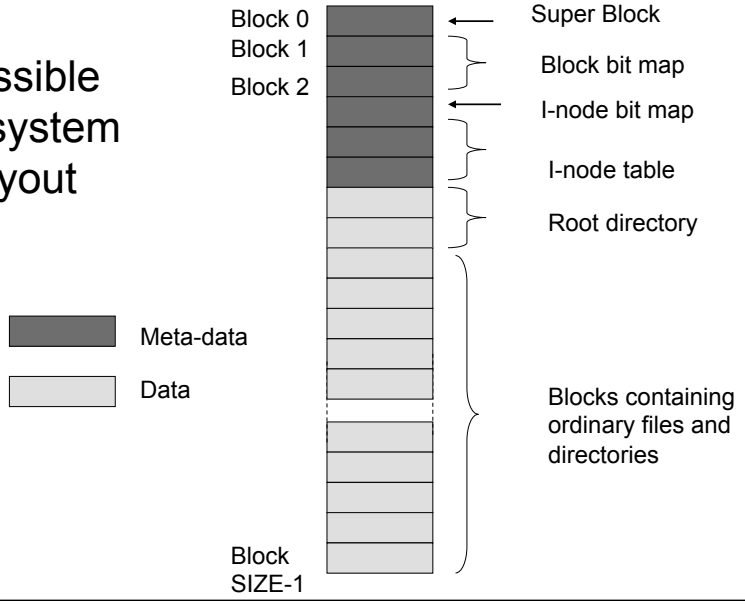


Implementing a file system

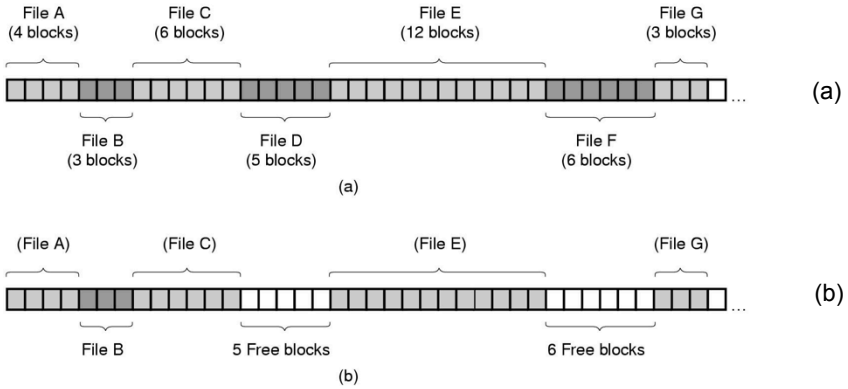
- How to support in the disk files and directories
- How to allocate blocks to files
- How to speed up file access

File System Implementation

A possible file system layout



Supporting Files



(a) Contiguous allocation of disk space for 7 files

(b) State of the disk after files *D* and *E* have been removed

As we will see later, it is not mandatory to allocate blocks sequentially.

Directory Implementation

Symbolic name	**Information associated**: **most important is index in i-node table**

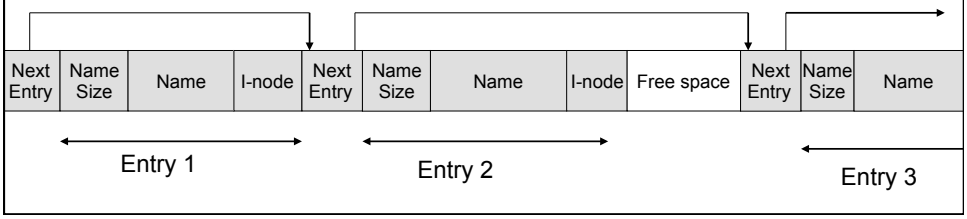
- **Stored in a file**
- **Table** of file names with corresponding i-node number
  - Linear search time
  - Could keep ordered alphabetically via linked list or use B+ tree
- How to handle large and small names ?

Directory

Name	Information

Name length

- Fixed-size entries
  - Inflexible
  - Works if name size is limited, or
  - Large space allocated, that in most cases is wasted
- Variable-size entries
  - More flexible
  - Harder to manipulate
  - A possible solution



Allocating blocks to files

- File sizes
  - Are most files small or large?
    - SMALL
  - Which accounts for more total storage: small or large files?
    - LARGE

File System Workload

- File access
  - Are most accesses to small or large files?
    - SMALL
  - Which accounts for more total I/O bytes: small or large files?
    - LARGE

File System Workload

- How are files used?
  - Most files are read/written sequentially
  - Some files are read/written randomly
    - Ex: database files, swap files
  - Some files have a pre-defined size at creation
  - Some files start small and grow over time
    - Ex: program stdout, system logs

File System Design

- For small files:
  - Small blocks for storage efficiency
  - Files used together should be stored together
- For large files:
  - Storage efficient (large blocks)
  - Contiguous allocation for sequential access
  - Efficient lookup for random access
- May not know at file creation
  - Whether file will become small or large
  - Whether file is persistent or temporary
  - Whether file will be used sequentially or randomly

Allocation Methods

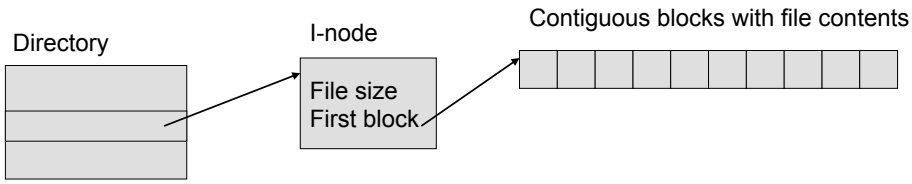
- An allocation method refers to how disk blocks are allocated for files
- 3 main methods
  - Contiguous allocation
  - Linked allocation
  - Indexed allocation

Contiguous Allocation

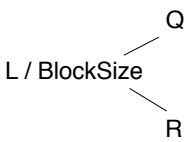
Contiguous allocation – each file occupies set of contiguous blocks

- Best performance in most cases
- Simple – only starting location (block #) and length (number of blocks) are required
- Problems include finding space for file, knowing file size, external fragmentation, need for **compaction off-line** (**downtime**) or **on-line**

Contiguous Allocation



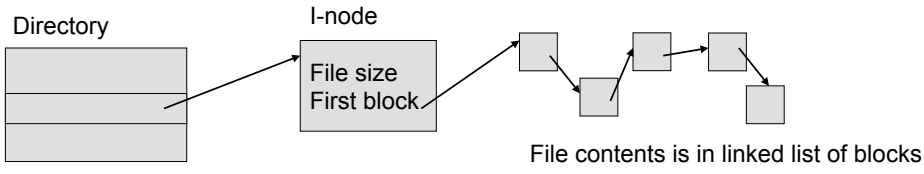
- Mapping from logical to physical: in which block is the byte with offset L ?



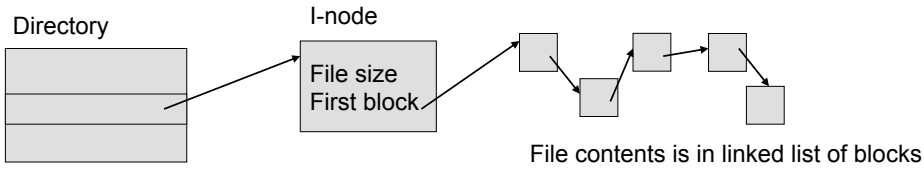
Block to be accessed = Q + first block
Displacement into block = R

Linked Allocation

- Each file a linked list of blocks
  - File ends at null pointer
  - No external fragmentation
  - Each block contains pointer to next block
  - No compaction, external fragmentation



Linked Allocation

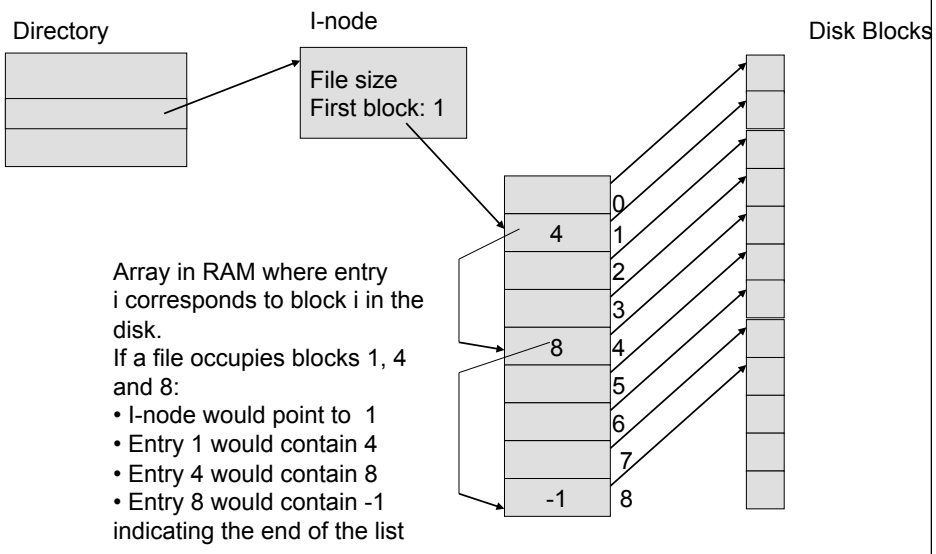


- Mapping from logical to physical:
  in which block is the byte with offset L ?

L / (BlockSize – sizePointerNextBlock) → Q, R

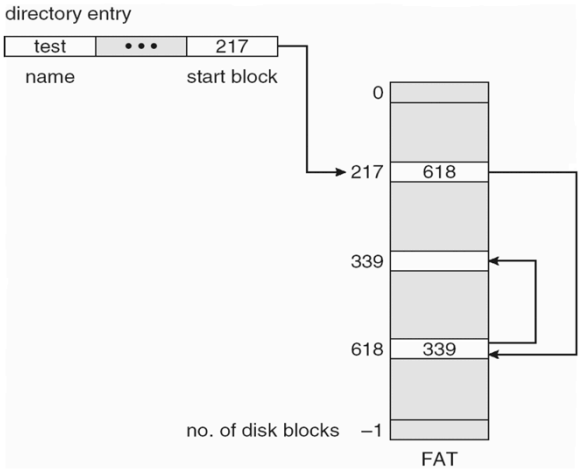
Block to be accessed is in Qth position in the list
Displacement into block = R

Linked Allocation with list in RAM



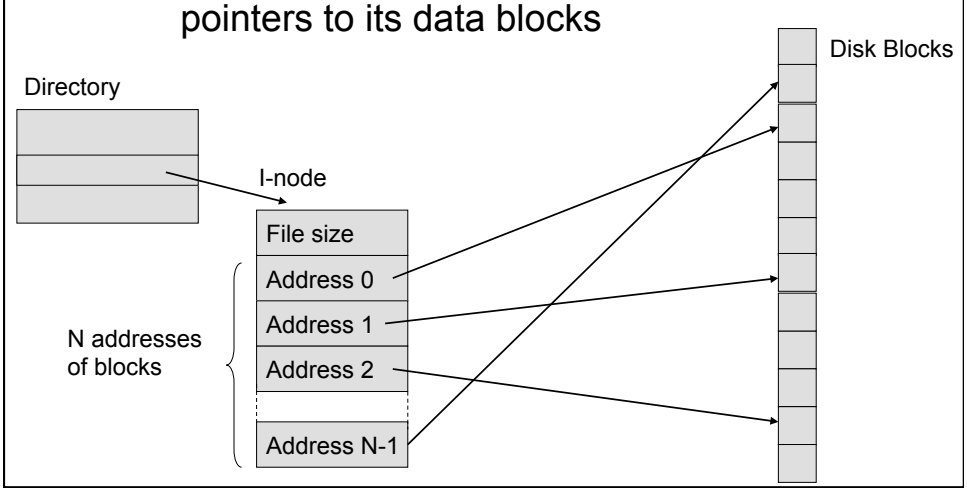
Array in RAM where entry i corresponds to block i in the disk.
If a file occupies blocks 1, 4 and 8:
- I-node would point to 1
- Entry 1 would contain 4
- Entry 4 would contain 8
- Entry 8 would contain -1 indicating the end of the list

File-Allocation Table (Windows FAT)



Allocation Methods - Indexed

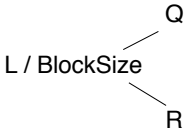
- **Indexed allocation**
  - Each file has its own **index block**(s) of pointers to its data blocks



Indexed Allocation (Cont.)

- Need index table
- Direct access (lseek) is easy
- Without external fragmentation, but have overhead of index block

Mapping from logical to physical: in which block is the byte with offset L ?

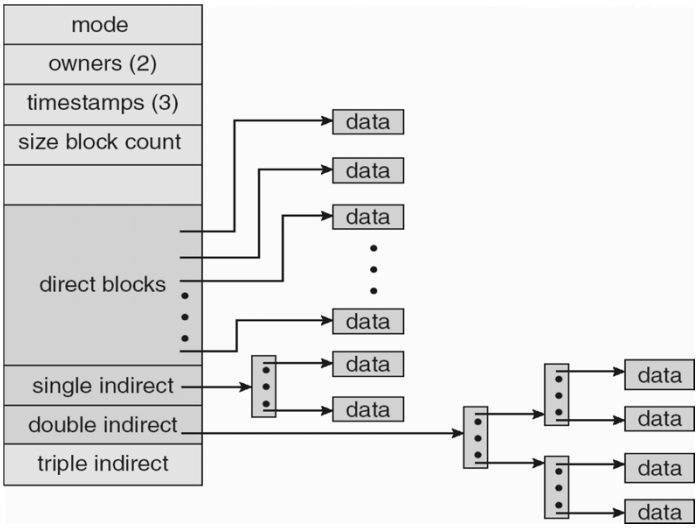


Q = displacement into index table
R = displacement into block

Indexed Allocation (Cont.)

- One of the addresses can point to a block with addresses (1-level indirect)
  - 1 block can contain BlockSize/AddressSize addresses. For example with blocks of 4096 bytes and addresses of 4 bytes, one can have 1024 addresses
- More levels of indirection are possible (see the following example of UNIX)
- Good compromise
  - Small files use only direct addresses
  - It is possible to support very large files (see the following example)

Unix/Linux: 3 levels of indirection



Maximum size of file = blockSize*(10 + 1024 + 1024*1024 + 1024*1024*1024)

Using this approach file size is bigger than 2^32; one needs more than 32-bit for the file offset

Performance

- Best method depends on file access type
  - Contiguous great for sequential and random
- Linked good for sequential, not random
- Indexed more complex
  - Single block access could require 2 index block reads then data block read

Efficiency and Performance

- Efficiency dependent on:
  - Disk allocation and directory algorithms
    - For each file, keeping data and meta-data as close as possible
  - Types of data kept in file's directory entry
  - Pre-allocation or as-needed allocation of metadata structures
  - Fixed-size or varying-size data structures

Efficiency and Performance (Cont.)

Buffer cache – separate section of main memory for frequently used blocks

- **read-ahead** – techniques to optimize sequential access
- **Delayed write** – writing to the buffer cache; eventually data changed in RAM will arrive to disk. The flush of data to the disk happens:
  - When the file is closed
  - When the user calls fsync()
  - Every 5-10 seconds by a system daemon; this value is just indicative

Block cache (cont)

- Delayed write advantages

  When writing in a block, there is a high probability of writing again in the same block shortly. Instead of writing in the disk, the system updates a copy of the block in RAM; the subsequent writes will also be made in RAM. If there N changes to the same block one will have to write to the disk only one time instead of N

- Delayed write disadvantages

  When changes are only in RAM the file system can be incoherent, i.e. part of the changes are already in the disk, others aren´t. This incoherence must not exist for long