Operating Systems

File Systems
(Select parts of Ch 11-12)

Outline

- Files ←
- Directories
- Partitions

File Systems

- Abstraction to disk (convenience)
  - "The only thing friendly about a disk is that it has persistent storage."
  - Devices may be different: tape, IDE/SCSI, NFS
- Users
  - don't care about detail
  - care about interface (won't cover, assumed knowledge)
- OS
  - cares about implementation (efficiency)

File System Structures

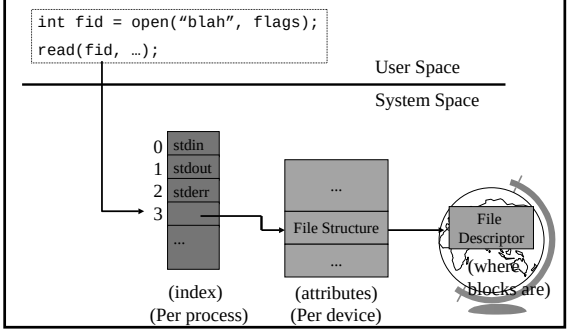
- *Files* - store the data
- *Directories* - organize files
- *Partitions* - separate collections of directories (also called "volumes")
  - all directory information kept in partition
  - `mount` file system to access

Example: Unix `open()`

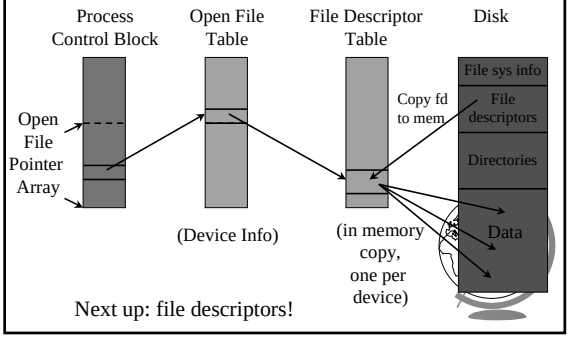
```
int open(char *path, int flags [, int mode])
```

- `path` is name of file
- `flags` is bitmap to set switch
  - O_RDONLY, O_WRONLY…
  - O_CREATE then use `mode` for perms
- success, returns index

Unix `open()` - Under the Hood



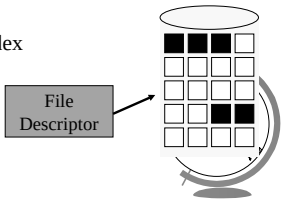
File System Implementation



Next up: file descriptors!

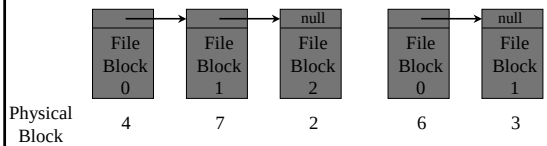
File System Implementation

- Which blocks with which file?
- File descriptor implementations:
  - Contiguous
  - Linked List
  - Linked List with Index
  - I-nodes



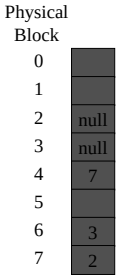
Linked List Allocation

- Keep a linked list with disk blocks



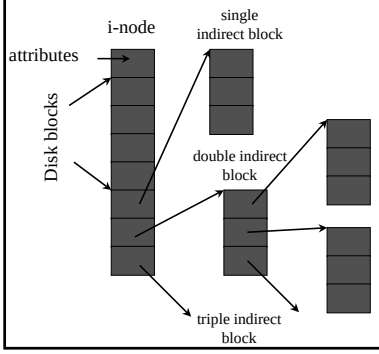
- Good:
  - Easy: remember 1 number (location)
  - Efficient: no space lost in fragmentation
- Bad:
  - Slow: random access bad

Linked List Allocation with Index



- Table in memory
  - faster random access
  - can be large!
    - 1k blocks, 500K disk
    - = 2MB!
  - MS-DOS FAT, Win98 VFAT

I-nodes



- Fast for small files
- Can hold big files
- Size?
  - 4 kbyte block

Outline

- Files (done)
- Directories ←
- Partitions

Directories

- Before reading file, must be opened
- Directory entry provides information to get blocks
  - disk location (block, address)
  - i-node number
- Map `ascii` name to the *file descriptor*

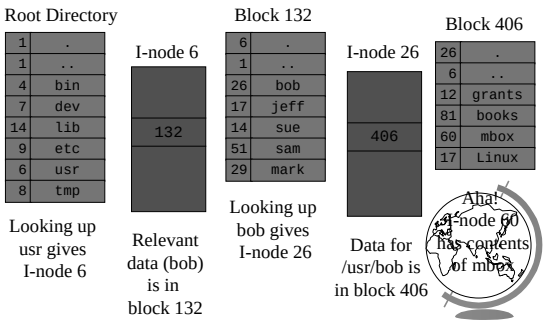
Hierarchical Directory (Unix)

- Tree
- Entry:
  - name
  - inode number (try "ls –I" or "ls –iad .")
- example:

```
60  /usr/bob/mbox
```

inode	name

Unix Directory Example

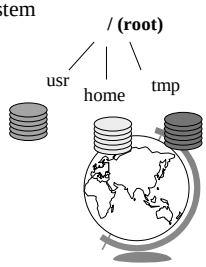


Outline

- Files (done)
- Directories (done)
- Partitions ←

Partitions

- `mount`, `unmount`
  - load "super-block" from disk
  - pick "access point" in file-system
- Super-block
  - file system type
  - block size
  - free blocks
  - free I-nodes



Partitions: `fdisk`

- Partition is large group of sectors allocated for a specific purpose
  - IDE disks limited to 4 physical partitions
  - logical (extended) partition inside physical partition
- Specify number of cylinders to use
- Specify type
  - magic number recognized by OS

(Hey, show example)

Keeping Track of Free Blocks

- Two methods
  - linked list of disk blocks
    - one per block or many per block
  - bitmap of disk blocks

(note, these are stored on the disk)

- Linked List of Free Blocks (many per block)
  - 1K block, 16 bit disk block number
    - = 511 free blocks/block
    - 200 MB disk needs 400 free blocks = 400k
- Bit Map
  - 200 MB disk needs 20 Mbits
  - 30 blocks = 30k
  - 1 bit vs. 16 bits

File System Maintenance

- Format:
  - create file system structure: super block, I-nodes
  - `format` (Win), `mke2fs` (Linux)
- "Bad blocks"
  - most disks have some
  - `scandisk` (Win) or `badblocks` (Linux)
  - add to "bad-blocks" list (file system can ignore)
- Defragment
  - arrange blocks efficiently
- Scanning (when system crashes)
  - lost+found, correcting file descriptors...