Operating Systems

File Systems
(Ch 10.1-10.4, Ch 11.1-11.5)

Motivation

- Process store, retrieve information
- Process capacity restricted to vmem size
- When process terminates, memory lost
- Multiple processes share information

- Requirements:
  - large
  - ***persistent***
  - concurrent access

Solution? Files!

Outline

- Files ¬
- Directories
- Disk space management
- Misc

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
- OS
  - cares about implementation (efficiency)

File System Concepts

- *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
- *Protection* - allow/restrict access for files, directories, partitions

Files: The User's Point of View

- Naming: how do I refer to it?
  - blah, BLAH, Blah
  - file.c, file.com
- Structure: what's inside?
  - Sequence of bytes (most modern OSes)
  - Records - some internal structure
  - Tree - organized records

Files: The User's Point of View

- Type:
  - ascii - human readable
  - binary - computer only readable
  - "magic number" (executable, c-file …)
- Access Method:
  - sequential (for *character* files, an abstraction of I/O of serial device such as a modem)
  - random (for *block* files, an absraction of I/O to block device such as a disk)
- Attributes:
  - time, protection, owner, hidden, lock, size …

File Operations

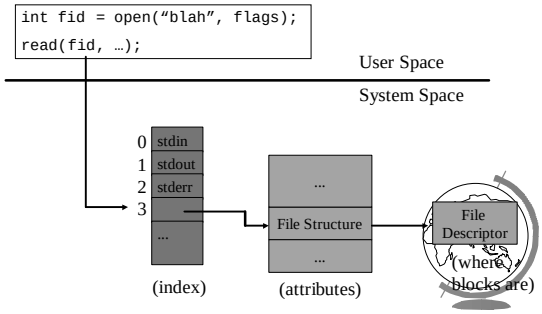
- Create
- Delete
- Truncate
- Open
- Read
- Write
- Append
- Seek - for random access
- Get attributes
- Set attributes

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



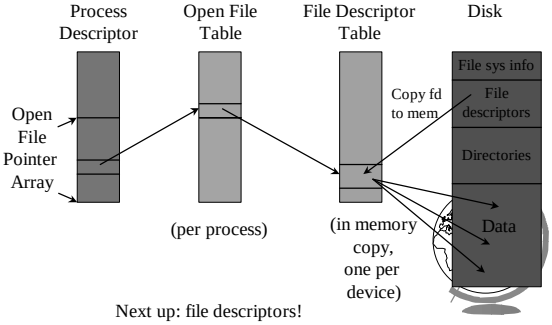
Example: WinNT `CreateFile()`

- Returns file object:

```
HANDLE CreateFile (
  lpFileName, // name of file
  dwDesiredAccess, // read-write
  dwShareMode,  // shared or not
  lpSecurity, // permissions
  ...
)
```

- File objects used for all: files, directories, disk drives, ports, pipes, sockets and console

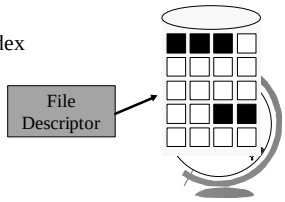
File System Implementation



Next up: file descriptors!

File System Implementation

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



Contiguous Allocation

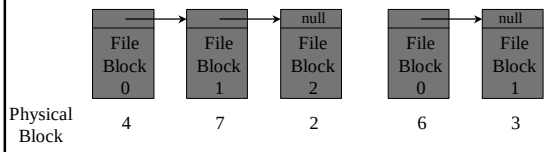
- Store file as contiguous block
  - ex: w/ 1K block, 50K file has 50 coneq blocks

```
File A: start 0, length 2
File B: start 14, length 3
```

- Good:
  - Easy: remember 1 number (location)
  - Fast: read entire file in one operation (length)
- Bad:
  - Static: need to know file size at creation
    - or tough to grow!
  - Fragmentation: remember why we had paging?

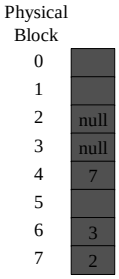
Linked List Allocation

- Keep a linked list with disk blocks



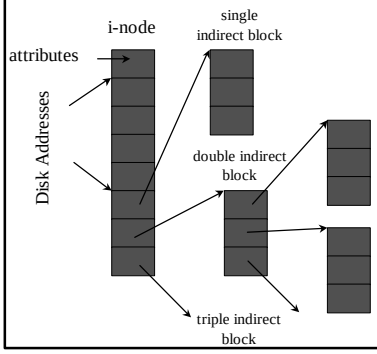
- Good:
  - Easy: remember 1 number (location)
  - Efficient: no space lost in fragmentation
- Bad:
  - Slow: random access bad

Linked List Allocation with Index



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

I-nodes



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

Outline

- Files ✓
- Directories ¬
- Disk space management
- Misc

Directories

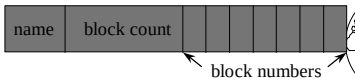
- Just like files, only have special bit set so you cannot modify them (*what?!*)
  - data in directory is information / links to files
- Organized for:
  - efficiency - locating file quickly
  - convenience - user patterns
    - groups (.c, .exe), same names
- Tree structure directory the most flexible
  - aliases allow files to appear at more than one location

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*

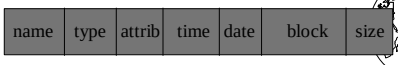
Simple Directory

- No hierarchy (all “root”)
- Entry
  - name
  - block count
  - block numbers



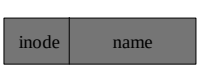
Hierarchical Directory (MS-DOS)

- Tree
- Entry:
  - name
  - type (extension)
  - time
  - date
  - block number (w/FAT)

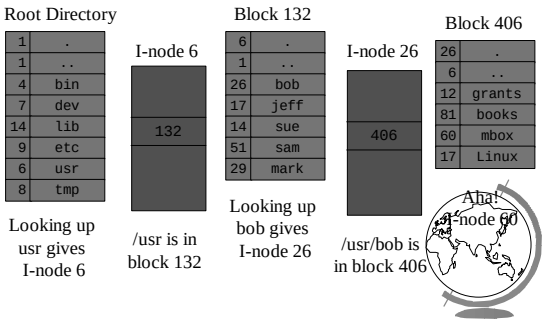


Hierarchical Directory (Unix)

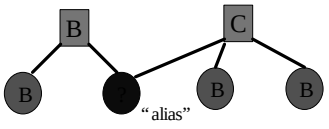
- Tree
- Entry:
  - name
  - inode number
- example:
  `/usr/bob/mbox`



Unix Directory Example



Storing Files



Directed Acyclic Graph

- Possibilities:
  - a) Directory entry contains disk blocks?
  - b) Directory entry points to attributes structure?
  - c) Have new type of file "link"?

Problems

- a) Directory entry contains disk blocks?
  - contents (blocks) may change
- b) Directory entry points to attributes structure?
  - if removed, refers to non-existent file
  - must keep count, remove only if 0
  - *hard link*
- c) Have new type of file "link"?
  - overhead, must parse tree second time
  - *soft link*

Outline

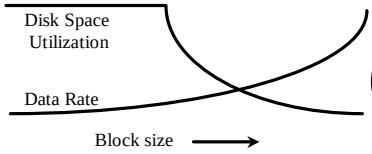
- Files ✓
- Directories ✓
- Disk space management ¬
- Misc

Disk Space Management

- *n* bytes
  - contiguous
  - blocks
- Similarities with memory management
  - contiguous is like segmentation
    - but moving on disk very slow!
    - so use blocks
  - blocks are like paging
    - how to choose block size?

Choosing Block Size

- Large blocks
  - wasted space (internal fragmentation)
- Small blocks
  - more seek time since more blocks



Keeping Track of Free Blocks

- Two methods (note, these are stored on the disk)
  - linked list of disk blocks
    - one per block or many per block
  - bitmap of disk blocks
- Linked List of Free Blocks (man per block)
  - 1K block, 16 bit disk block number
    - = 511 free blocks/block
    - 200 MB disk needs 400 blocks = 400k
- Bit Map
  - 200 MB disk needs 20 Mbits
  - 30 blocks = 30 K
  - 1 bit vs. 16 bits

Tradeoffs

- Only if the disk is nearly full does linked list scheme require fewer blocks
- If enough RAM, bitmap method preferred
- If only 1 "block" of RAM, and disk is full, bitmap method may be inefficient since have to load multiple blocks
  - linked list can take first in line

File System Performance

- Disk access 100,000x slower than memory
  - reduce number of disk accesses needed!
- Block/buffer cache
  - cache to memory
- Full cache? FIFO, LRU, 2nd chance …
  - exact LRU can be done
- LRU inappropriate sometimes
  - crash w/i-node can lead to inconsistent state
  - some rarely referenced (double indirect block)

Modified LRU

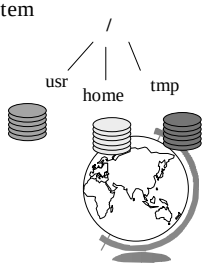
- Is the block likely to be needed soon?
  - if no, put at beginning of list
- Is the block essential for consistency of file system?
  - write immediately
- Occasionally write out all
  - sync

Outline

- Files ✓
- Directories ✓
- Disk space management ✓
- Misc ¬
  - partitions (`fdisk, mount`)
  - maintenance
  - quotas
  - Linux
  - WinNT

Partitions

- `mount`, `unmount`
  - load "super-block"
  - pick "access point" in file-system
- Super-block
  - file system type
  - block Size
  - free blocks
  - free inodes



Partitions: `fdisk`

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

File System Maintenance

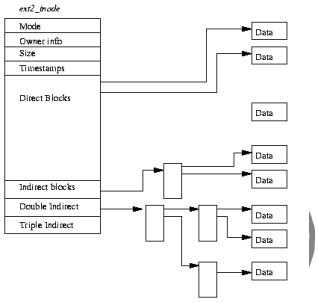
- Format:
  - create file system structure: super block, inodes
  - `format` (Win), `mke2fs` (Linux)
- "Bad blocks"
  - most disks have some
  - `scandisk` (Win) or `badblocks` (Linux)
  - add to "bad-blocks" list (file system can ignore)
- Defragment
  - arrange blocks efficiently
- Scanning (when system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

Disk Quotas

- Table 1: Open file table in memory
  - when file size changed, charged to user
  - user index to table 2
- Table 2: quota record
  - soft limit checked, exceed allowed w/warning
  - hard limit never exceeded
- Overhead? Again, in memory
- Limit: blocks, files, i-nodes

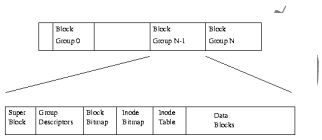
Linux Filesystem: ext2fs

- "Extended (from minix) file system vers 2"
- Uses inodes
  - *mode* for file, directory, symbolic link ...



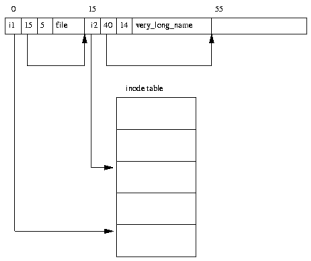
Linux filesystem: blocks

- Default is 1 Kb blocks
  - small!
- For higher performance
  - performs I/O in chunks (reduce requests)
  - clusters adjacent requests (block *groups*)
- Group has:
  - bit-map of free blocks and inodes
  - copy of super block



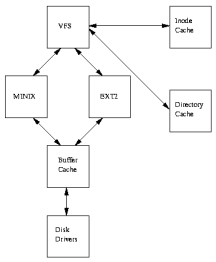
Linux Filesystem: directories

- Special file with names and inodes



Linux filesystem: proc

- contents of "files" not stored, but computed
- provide interface to kernel statistics
- allows access to "text" using Unix tools
- enabled by "virtual file system"



WinNT Filesystem: NTFS

- Basic allocation unit called a *cluster* (block)
- Each file has structure, made up of *attributes*
  - attributes are a stream of bytes
  - stored in *Master File Table*, 1 entry per file
  - each has unique ID
    - part for MFT index, part for "version" of file for caching and consistency
- Recover via "transaction" where they have a log file to restore redo and undo information