


CS4513 Distributed Computer Systems


File Systems



Motivation


- Processes store, retrieve information
- Process capacity restricted to vmem size
- When process terminates, memory lost
- Multiple processes share information
- Requirements:
 - large
 - ***persistent***
 - concurrent access

Solution? File System!




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




Outline

- Files
 - User's point of view
 - Example "Under the Hood"
 - File system implementation
- Directories
- Disk space management
- Misc



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" or extension (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 abstraction 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 ()

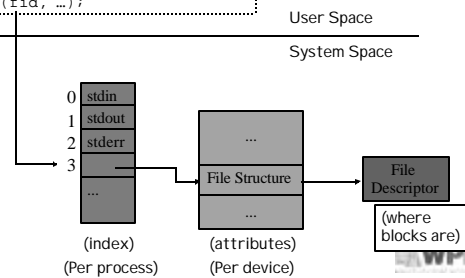
```
int open(char *path, int flags [, int mode])
```

- path is name of file
- flags is bitmap to set switch
 - O_RDONLY, O_WRONLY...
 - O_CREAT then use mode for perms
- on success, returns index
- on failure, returns -1



Unix open () - Under the Hood

```
int fid = open("blah", flags);
read(fid, ...);
```



Example: Windows CreateFile ()

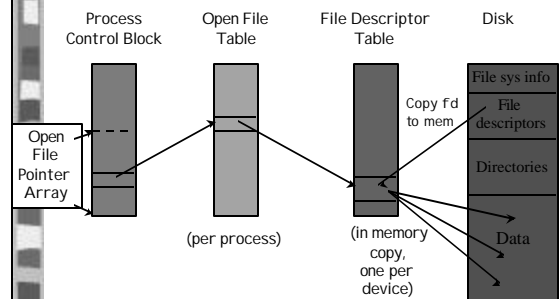
- Returns file object handle:

```
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

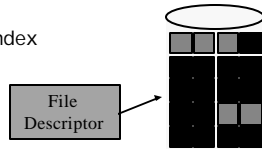


Next up: file descriptors!



File System Implementation

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



Contiguous Allocation

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

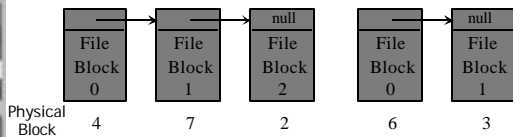
File A: start 0, length 2
File B: start 14, length 3

- Good:
 - Easy: remember location with 1 number
 - Fast: read entire file in 1 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

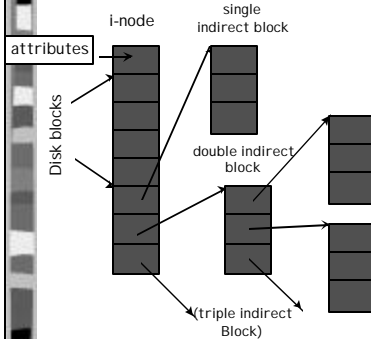
Physical Block

0	
1	
2	null
3	null
4	7
5	
6	3
7	2

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



I-nodes



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



Outline

- Files (done)
- 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
 - modify through system call
 - (See `ls.c` sample)
- 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 (more on this later)



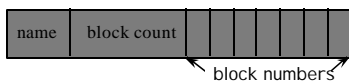
Directories

- 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

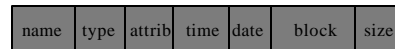


(What are the drawbacks?)



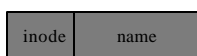
Hierarchical Directory (MS-DOS)

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

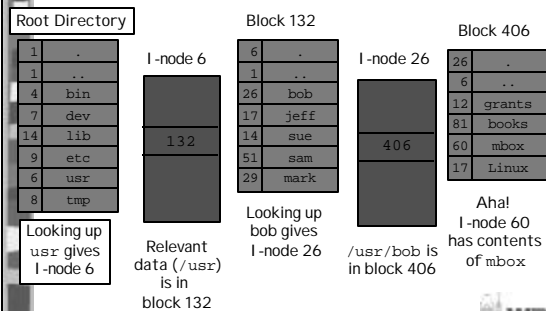


Hierarchical Directory (Unix)

- Tree
- Entry:
 - name
 - inode number (try "`ls -l`" or "`ls -liad .`")
- example:
 - `/usr/bob/mbox`



Unix Directory Example



Storing Files

(Not really a tree
- Directed Acyclic Graph)

- What if ...
 - a) Directory entry contains disk blocks?
 - b) Directory entry points to file descriptor?
 - c) Have new type of file "link"?

WPI

Issues

- a) Directory entry contains disk blocks?
 - contents (blocks) may change
- b) Directory entry points to file descriptor?
 - if removed, refers to non-existent file
 - must keep count, remove only if 0
 - *hard link*
 - Similar if delete file in use (show example)
- c) Have new type of file "link"?
 - contains alternate name for file
 - overhead, must parse tree second time
 - *soft link*
 - often have max link count in case loop (show example)

WPI

Outline

- Files (done)
- Directories (done)
- Disk space management →
- Misc

WPI

Disk Space Management

- n bytes
 - contiguous
 - blocks
- Similarities with memory management
 - contiguous is like variable-sized partitions
 - but moving on disk very slow!
 - so use blocks
 - blocks are like paging
 - how to choose block size?
- (Note, disk block size typically 512 bytes, but file system logical block size chosen when formatting)

WPI

Choosing Block Size

- Large blocks
 - faster throughput, less seek time
 - wasted space (internal fragmentation)
- Small blocks
 - less wasted space
 - more seek time since more blocks

WPI

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 (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
 - With linked list, 30 blocks = 30k
 - 1 bit vs. 16 bits

WPI

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
- Sometimes, combine both (Linux)



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 ...
 - Unlike in VM, exact LRU can be done (why?)
- 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



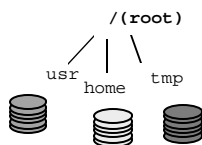
Outline

- Files (done)
- Directories (done)
- Disk space management (done)
- Misc \rightarrow
 - partitions (fdisk, mount)
 - maintenance
 - quotas
- Linux and WinNT/2000



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)



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...
 - `e2fsck` (Linux) or `fsck` (Win)



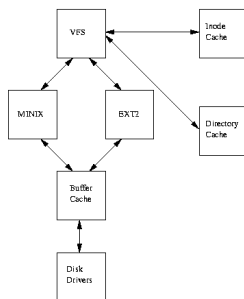
Disk Quotas

- 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, so relatively fast
- Limit: blocks, files, i-nodes



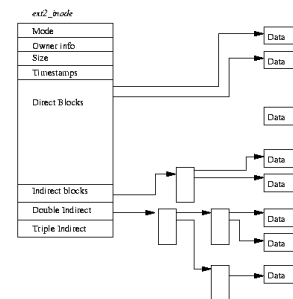
Linux: Virtual File System

- File system independent layer
- Generic inode and directory entry for all
 - Even if not inode based
- Specific file systems register with VFS



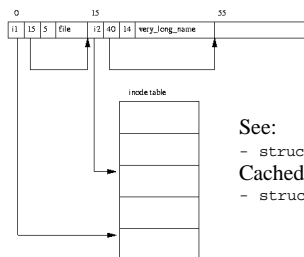
Linux File System: ext2fs

- "Extended (from Minix) file system, v2"
- Uses inodes
 - `mode` for file, directory, symbolic link ...
- (See: `struct inode`)



Linux File System: ext2 directories

- Special file with names (+ length) and inodes

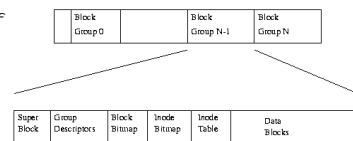


See:
- `struct ext2_dir_entry_2`
Cached. See:
- `struct dentry`



Linux File System: ext2 blocks

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



Linux File System: ext2 Superblock

- **Magic Number**
 - allow mounting check that is an EXT2 file system
- **Revision Level**
 - major and minor revision levels for compatibility check
- **Mount Count and Maximum Mount Count**
 - Run `esfsck` if reach max
- **Block Size**
 - Block size in bytes, for example 1024 bytes
- **Blocks per Group**
 - Blocks in a group. Fixed when the file system is created
- **Free Blocks**
 - Free blocks in the file system
- **Free Inodes**
 - Free inodes in the file system
- **First Inode**
 - First inode in the file system. Points to root dir
(See `struct superblock`)



Linux Filesystem: /proc

- Contents of "files" not stored, but computed
- Provide interface to kernel statistics
- Allows access to "text" using Unix tools
- Again, enabled by "virtual file system"
- (NT/2000 has `perfmon` to access registry)
- (show example in `/proc`)
- (show `biteMe` module example)



WinNT/2000 Filesystem: NTFS

- *Volume* (partition) can cover part, all or multiple disks
- Basic allocation unit called a *cluster* (block)
- Each file has structure, made up of *attributes*
 - Examples: time modified, permissions, author...
 - attributes are a stream of bytes
 - stored in *Master File Table*, 1 entry per file
 - Metadata (free blocks, etc) kept in MFT for volume
 - each has unique ID
 - part for MFT index, part for "version" of file for caching and consistency
- Hierarchical directory with internal structure stored as B+ tree (for efficiency)
- Supports compression plus encryption



WinNT/2000 Filesystem: Recovery

- Avoid the need to `fdisk`
- Use database notion of "transaction" (all or none)
 - Before data committed, record start in log
 - Also contain redo or undo information
 - After data written, write to log that done
- If a crash, redo or undo ops that did not finish
- Periodically (5 sec by default) record checkpoint
 - Can the discard log
- Note, does not guarantee data is ok, only metadata
- Linux has:
 - `resiserfs` (journaling of metadata) + `ext3` (journaling metadata + data)
- (See "samples" for journaling + file system stuff)

