Operating Systems

Motivation
- Processes store, retrieve information
- Process capacity restricted to main mem size
- When process terminates, memory lost
- Multiple processes share information
- Requirements:
  - large
  - persistent
  - concurrent access
Solution? File System!

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, B lah
  - 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()

```c
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
- **success, returns index**

Unix open() - Under the Hood

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

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 conseq 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

<table>
<thead>
<tr>
<th>Physical Block</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block 0</td>
</tr>
<tr>
<td>Block 1</td>
</tr>
<tr>
<td>Block 2</td>
</tr>
<tr>
<td>Block 0</td>
</tr>
<tr>
<td>Block 1</td>
</tr>
</tbody>
</table>

- 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
- Directories
- Disk space management
- Mac

(done)
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, cd)
- Organized for:
  - efficiency - locating file quickly
  - convenience - user patterns
    - groups (. , .exe), same names
- Tree structure directory the most flexible
  - aliases allow files to appear at more than one location

Simple Directory

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

Hierarchical Directory (MS-DOS)

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

Hierarchical Directory (Unix)

- Tree
- Entry:
  - name
  - inode number (try “Is -I” or “ls -ld.”)
- Example:
  /usr/bob/mbox

Unix Directory Example

Root Directory

Block 132

Block 26

Block 406

Looking up/usr gives 1-node 6
Looking up /usr/bob gives 1-node 6
Looking up /usr/bob/mbox gives 1-node 60

What 1-node 60 has contents of mbox
Storing Files

- Possibilities:
  a) Directory entry contains disk blocks?
  b) Directory entry points to attributes structure?
  c) Have new type of file “link”?
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 (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
  - 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), mkfs (Linux)

- **"Bad blocks"**
  - most disks have some
  - scandisk (Win) or badblocks (Linux)
  - add to "bad-block" 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: directories

- **Special file with names and inodes**

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
  - copy of super block

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" (NT has perfmon)
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