Operating Systems

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

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!

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" 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
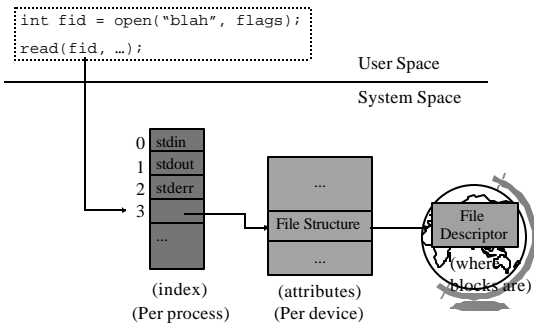- Get attributes
- Set attributes

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_CREATE then use `mode` for perms
- success, returns index

Unix `open()` - Under the Hood

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

User Space

System Space

0	stdin
1	stdout
2	stderr
3	
...	

...

File Structure

...

File Descriptor (where blocks are)

(index)
(Per process)

(attributes)
(Per device)

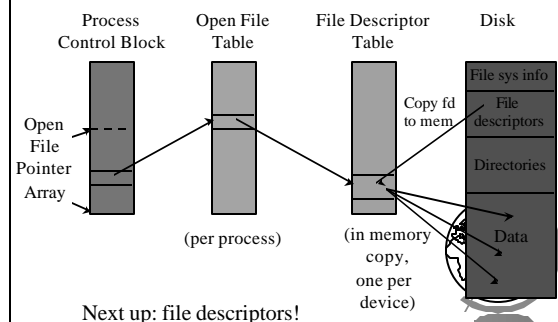## Example: WinNT `CreateFile()`

- 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

Process Control Block

Open File Table

File Descriptor Table

Disk

Open File Pointer Array

Copy fd to mem

File sys info

File descriptors

Directories

Data

(per process)

(in memory copy, one per device)

Next up: file descriptors!

File System Implementation

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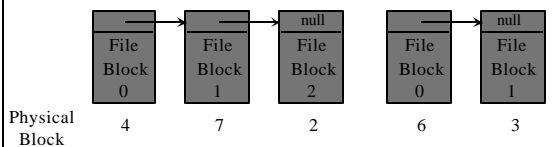 - Linked List
  - Linked List with Index
  - I-nodes

File Descriptor
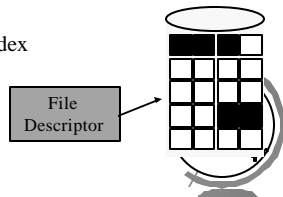
Contiguous Allocation

- 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

| File Block 0 | File Block 1 | File Block 2 | File Block 0 | File Block 1 |

Physical Block   4   7   2   6   3

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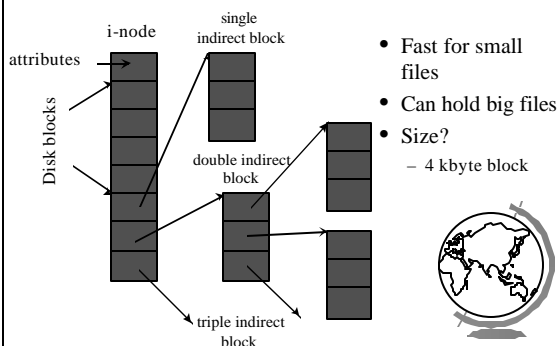

Linked List Allocation with Index

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, 500K disk
    + = 2MB!
  - MS-DOS FAT, Win98 VFAT

I-nodes

i-node
single indirect block
attributes
Disk blocks
double indirect block
triple indirect block

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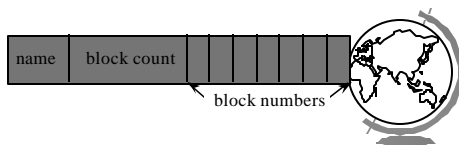 – aliases allow files to appear at more than one location

Directories

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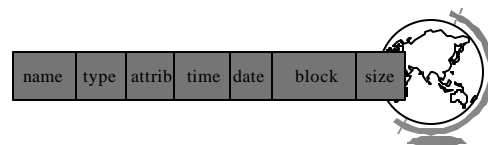 – i-node number
- Map ascii name to the *file descriptor*

Simple Directory

- No hierarchy (all "root")
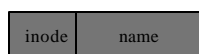- Entry
  – name
  – block count
  – block numbers

name	block count							

block numbers

Hierarchical Directory (MS-DOS)

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

name	type	attrib	time	date	block	size

Hierarchical Directory (Unix)

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

inode	name

Unix Directory Example

Root Directory

1	.
1	..
4	bin
7	dev
14	lib
9	etc
6	usr
8	tmp

Looking up usr gives I-node 6

I-node 6

132

Relevant data (/usr) is in block 132

Block 132

6	.
1	..
26	bob
17	jeff
14	sue
51	sam
29	mark

Looking up bob gives I-node 26

I-node 26

406

/usr/bob is in block 406

Block 406

26	.
6	..
12	grants
81	books
60	mbox
17	Linux

And I-node 60 has contents of mbox

Storing Files

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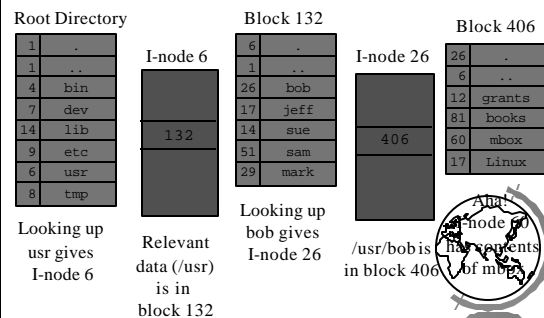
"alias"
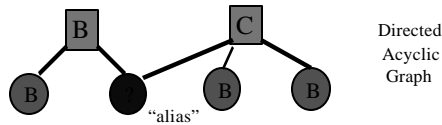
- Possibilities:
  - a) Directory entry contains disk blocks?
  - b) Directory entry points to attributes structure?
  - c) Have new type of file "link"?

Problems

- a) Directory entry contains disk blocks?
  - contents (blocks) may change
- b) Directory entry points to 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)

Outline

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

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)

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

Disk Space Utilization

Data Rate

Block size →
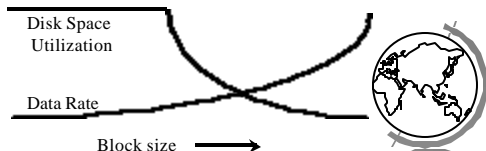
Keeping Track of Free Blocks

- Two methods   (note, these are
  - linked list of disk blocks   stored on the disk)
    + 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
    + 30 blocks = 30k
    + 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 (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?
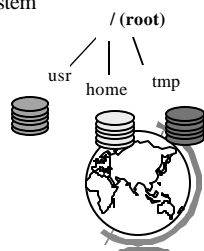 - write immediately
- Occasionally write out all
  - sync

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

/ (root)

usr   home   tmp

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)
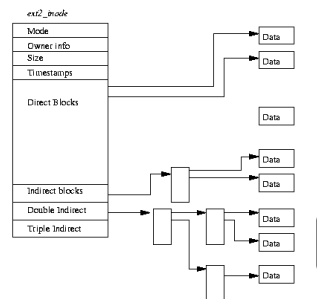 - lost+found, correcting file descriptors...

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
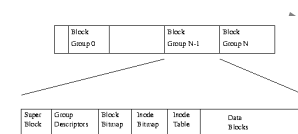- Limit: blocks, files, i-nodes

Linux Filesystem: ext2fs

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

ext2_inode

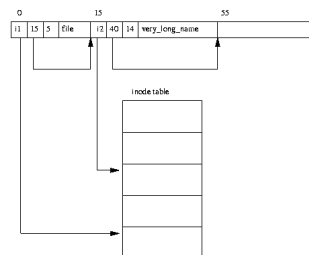| Mode |
| Owner info |
| Size |
| Timestamps |
| Direct Blocks |
| Indirect blocks |
| Double Indirect |
| Triple Indirect |

Data

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 I-nodes
 - copy of super block

| Block Group 0 | | Block Group N-1 | Block Group N |

| Super Block | Group Descriptors | Block Bitmap | Inode Bitmap | Inode Table | Data Blocks |

Linux Filesystem: directories

- Special file with names and inodes

| i1 | 15 | 5 | file | i2 | 40 | 14 | very_long_name | |

inode table

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`)

VFS — Inode Cache
MINIX — EXT2 — Directory Cache
Buffer Cache
Disk Drivers

7
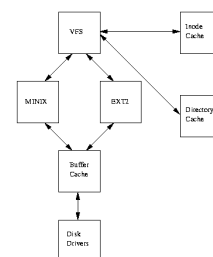
WinNT Filesystem: NTFS

- 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