Lecture 13: File Systems (Mar 8, 10, 22, 2005)
Yap

March 24, 2005

1 ADMIN

- READING GUIDE FOR Chap.6 on File Systems. Read up to page 424 (Section 6.3.6).

2 Review

Question FOR NEXT CLASS:

- Q: Suppose you have a 40 GB disk, and your block size is 2KB. What is the size of your FAT table if each block entry in the table takes 6 bytes?

  A:

  40 GB = $40x10^9$. So the number of entries is $40x10^9/2000 = 20x10^6$ i.e., 20 million entries. So we need 120 Megabytes of FAT table.

  Q: Suppose you use a bit map. What is the size of the bit map table?

  A:

- Q: What is the relative advantage/disadvantage of FAT tables versus I-Nodes?

  A:

- Q: What are typical block sizes?

  A: 1-2 KB.

3 Overview of File Systems

- Overall organization:
  - rooted tree hierarchy (basically)
  - leaves are "files"
  - internal nodes are "directories" (or "folders")

  - concepts: root, path
  - relative or absolute path
  - special paths: "dot" and "dotdot" and "tilde"
  - current directory (cwd)
- File Names
  - Firstname.SecondName
  - SecondName = "extension" (e.g., .ps, .pdf, .txt)
  - as convention (unix) or enforced (windows)
  - special chars in file name
  - length of file names
- Types of Files
  - regular files ("actual files" and directory files)
  - but Unix also has irregular files ("char special files" to represent serial I/O devices, and "block special files" to represent disks).
  - regular non-directory files are either ASCII or binary.
- File Attributes
  - access: sequential/random (today: mostly random)
  - passwd
  - permission (r, w, x)
  - creator
  - owner
  - time of creation, last access, last change
  - size
- Nondirectory File Operations
  - create, delete, open, close, read, write
  - rename, get/set attributes
  - link/unlink
- Directory File Operations
  - create, delete, opendir, closedir
  - readdir, rename

4 Physical File Implementation

- storage in disk (most common case)
- simple solution: contiguous allocation
  – useful in CDROM and read only media
- GENERAL SOLUTION: disk blocks
  – What is ideal block size? Experimentally, 1-2 Kb.
- SOLUTION 1: linked list of blocks for each file
  – disadvantage: slow sequential access
- SOLUTION 2: FAT (= linked list + table)
  – store File Allocation Table in main memory
  – Disadvantage: space.
  – E.g., 20 GB disk, 1 Kb blocks implies 20 million entries in FAT. If each entry takes 4 bytes, we need 80 MB for FAT
- SOLUTION 3: I-Nodes (index nodes)
  – Each entry in directory stores a fixed number of I-nodes. If more are needed, the last entry points to another fixed size list.
  – Advantage: space for files depends only on the actual file sizes, not on the entire physical disk size.
- Tracking Free Blocks:
  –SOLUTION A: free list (can used empty blocks to do this!)
  –SOLUTION B: bitmaps
- Issue: Trashing in Free Blocks
  – SOLUTION: introduce some hysterisis

5 Implementing Directories

- Main function: mapping an ascii name (+path) to the blocks containing the file
- SOLUTION 1 (Windows):
  – Each file has an entry in the directory file, storing all the file attributes, including the file name.
- SOLUTION 2 (Unix):
  – Each file entry in directory is just the file name PLUS I-node.
  – the first I-node contains all the file attributes

6 File Consistency

- Unix (fsck), Windows (scandisk)
- Consistency problem:

  A block that has been modified but not yet written before the system crashes.

  Different problems arise depending on whether this block is

  (1) an i-node, or (2) directory block, or (3) an free-list block, or (4) an ordinary file block.

- Suppose we want to check the consistency of the current file system (say after a system reboot).

  – We must first ensure that our directory system is consistent. Next we ensure that the block within the files are consistent.

  – This gives rise to two kinds of consistency checks: block consistency and file consistency

- **File Consistency Check**:

  – This may also be called "Directory Consistency Check"

  – We maintain a counter on a "per file" basis

  ALGORITHM:

  – Start at the Root Directory, to recursively inspect each directory.

  – For each file in a directory, we increment a counter for that files usage. Initially, that counter value is 0.

  – RECALL: hard links cause this counter value to be greater than 1.

  – SOFT links has no such effects.

  – FINALLY, we verify that this count is consistent with the number of link counts stored as a file attribute. If so, the system is CONSISTENT.

  – IF NOT, we just reset the link count to the counter value.

  – PROBLEM 1: If the counter value is LARGER than the link count in I-node. What could happen if not fixed?

  – PROBLEM 2: If the counter value is LESS than the link count in I-node. What could happen if not fixed?

- **Block Consistency Check**:

  ALGORITHM:

  –Build up 2 tables for all blocks

  –Table 1: each block has a counter, initially 0.

  This counts the number of times each block appears in a file

  –Table 2: each block has a counter, initially 0.

This counts the number of times each block appears in free list

–First, read all I-nodes of files in directories

–For each I-node, follow links to all the I-nodes in this file. This updates Table 1.

–Then it goes into the free list, and locate all the blocks in this list. The count in Table 2 is updates.

–If file system is consistent, each block has a TOTAL count of 1 in Tables 1 and 2.

- BLOCK CONSISTENCY PROBLEMS:

  – Missing blocks: in neither table!

  (SOLUTION: add to free list)

  – Duplicate free blocks: twice in Table 2 (free list)!

  (SOLUTION: rebuild free list)

  – Duplicated in files: twice in Table 1 (file list)!

  WHY IS THIS BAD? If we delete both files, the block will put in free list twice, etc.

  (SOLUTION: make a copy of block for each file)

  – Free and used at the same time: used in both Tables!

  SOLUTION: remove from free list