

CS162 – Section # 11, 04/15/2003

Rodrigo Fonseca

Announcements

- Proj. 3 Initial Design due Today
 - For Phases 3 and 4
 - Test cases! They should be described more specifically as to how you implemented them, and what the expected result was. Also, there should be a distinction between glass box and black box tests, and unit/integration tests when applicable.
 - You will also submit test cases with your code. (The code per se should not be in the design doc)
 - For phase 3, there should be a test comparing your replacement policy to a random replacement policy.
 - Design review meetings: these will be held on Thursday and Friday, see the signup page at <http://inst.eecs.berkeley.edu/~cs162-tc/cgi/design.cgi>
 - Submitting design documents:
 - Initial: submit projX-design
 - Final: submit projX-final-design

We will take points off for these two next phases
 - Today: 2 questions, Directories, Filesystem performance
-

Practice Question: File system organization

We want to design a UNIX-like file system with 4KB blocks and 32-bit disk addresses. An i-node contains 10 direct entries, 2 indirect entries, and 1 doubly indirect entry. Assume that an i-node takes up 100B of space.

1. What is the maximum size of a file in this file system?

$$(10 + 2048 + 1048576) * 4096B = 4303396864 \text{ ~just over 4GB}$$

2. How much overhead storage is required to store a file of size 50MB?

First, we need the 100B for the i-node.

Now, the first 10 direct entries hold up to 40Kb, so we need more.

*The two indirect entries let us go to $4KB * (10 + 2048) = 8MB + 40KB$. The overhead for these is one block per entry, so 8KB.*

We still need more space, and get it from the double indirect entry.

*The first level is a block (more 4KB of overhead). Each second level block will give us extra $4KB * 1024$, or 4MB. So, we need 11, to give us 44MB.*

The total overhead is then: 100B + (2 + 1 + 11)*4KB, or 56KB and 100 Bytes.

3. What would be the advantages and disadvantages of having
 - a. the block size be 1KB
 - + less internal fragmentation
 - smaller files (only 256 pointers per block, for example)
 - lots of seeks
 - high overhead per block on disk
 - b. the block size be 40KB?
 - +larger files

- +fewer seeks
- internal fragmentation

BSD 4.3 uses 4K blocks, and fragments of ¼ of a block if needed.

The optimal parameters depend on the pattern of usage of the filesystem. For example, in [1], most files accessed are smaller than 10K, but some are very large. Thus, per file cost must be low, but must have good performance for large files.

[1] M.G. Baker, J.H. Hartman, M.D. Kupfer, K.W. Shirriff, and J.K. Ousterhout, "Measurements of a Distributed File System", in proceeding of the 13th ACM Symposium on Operating Systems Principles, 1991.

Question: Network Performance

You and your friend (who lives in San Jose) want to share a movie file of 700MB. This file is on your hard disk. It takes you 10 minutes to record the file to a CD, and 1 ½ hours to drive to your friend's place.

1. What are the overhead, latency, and throughput? (Consider recording the CD to be the overhead in this case)

Answer:

Overhead: 10 minutes

Latency: 90 minutes

Throughput delay: almost negligible, just watch the movie...

2. Suppose you have a connection to your friend over DSL, running at 256Kbps, with a latency of 200ms. The overhead per packet is 1ms. How long does it take to transfer the file?

Answer:

*The latency is 200ms, you must wait for this until the first packet arrives there. Assuming you then send at 256Kbps, it takes $700MB * 8Mb/MB / 0.25Mb/s = 22400s = 373m20s > 6h!$, which will dominate the transfer time.*

3. How fast would your connection have to be (bandwidth) to transfer the file faster than with the 'car network'?

Answer:

You want to transfer the file in less than 100 minutes.

*$700MB * 8Mb/MB \text{ bit} / x \text{ Mb/s} = 6000s$. $x = 5600Mb/6000s = 0.933Mbps$*

An interesting question would be: How many CD's fit in your car...

Directory Hierarchy

Directories are special files with names, index pair
Organized hierarchically: one entry can be another directory
Allows to mapping from names to indices.

root is at a known location in the filesystem, find everything from there.

Caching of directories is crucial to performance. Why?

In DOS, directory tree.

In Unix, more general, because of links.

A Hard Link is a directory entry that points to the same file.

A Soft (or Symbolic) Link is shortcut: a special file whose contents are the symbolic name of the linked file.

Say you create file a. You then issue 'ln a b'; 'ln -s a c'. You then delete a. What happens to b and c?

File System Performance

File Caching

Caching is our old friend! There is great locality in file access patterns. The buffer cache stores disk blocks in main memory.

Many I-nodes should be in the cache, frequently accessed.

How to size the cache?

- If it is too small, and we are using LRU, you know what may happen!
- If it is too large, you are taking up memory that you could use to 'cache' VM pages.

See Silberchatz 12.6.2

Prefetching

Most reads are sequential, so, you may want to read ahead, with the hope that those blocks will indeed be requested.

→ Tradeoff:

- if you read too much, waste disk bandwidth.
- if you don't read enough, you will pay extra seeks and rotations (you want to amortize these, remember last section)

Delayed Writes

See *prof. Joseph's Fall 99 Midterm #2, question 2*

Write-behind: used in Unix

write syscall copies data to kernel buffer

periodically the kernel writes data to disk

+ more room for disk scheduler to play around

+ lots of temporary files are short-lived

- loss of data in the event of a crash

More:

Take a look at Access Control (ACLs vs Capabilities)