CS 537 Nov 5 Handout

- Interface
  - Metadata
  - Symbolic and hard links
  - Mounting a file system
- Implementation: A Simple File System
- On-disk structures
  - data region
    - blocksize?
  - metadata
    - inode table
  - allocation structures
    - bitmaps
  - directories
  - superblock
    - magic number
    - fs type
    - # of inodes, data blocks
  - mkfs
  - mounting a file system
- Access methods
  - creating a file
  - read()
  - write()
- Long pathnames?
  - Resolve dir by dir
- Large files?
  - Indirect pointers
- Optimizing performance
  - unified page cache

Research

A Five-Year Study of File-System Metadata (TOS 2007)

Questions

1. Why does the simple file system have so many direct pointers and so few indirect pointers? Why not simply have indirect (and double indirect, etc.) pointers?
2. Write down the list of inodes and blocks accessed to read block of /tmp/a.
3. Note that the simple file system has a limited number of inodes. What happens when all the inodes get used? What operations can no longer be performed?
4. What happens if a corruption (a 1 switches to 0 or a 0 switches to 1) happens in the data or inode bitmap? Which is worse -- corruption in the data or the inode bitmap?
5. In the simple file system, the locations of the inode table and the data region are static -- fixed at mkfs time. What would be a good reason to make it dynamic?
6. Imagine a file system that never wrote anything to disk -- it is purely in memory. Would such a file system be useful? Why?
7. If a file-system layout is static (inode table and data region are fixed), why do we need a superblock?
8. What happens on a file system if the root inode is not accessible?
9. Why do we need to “mount” a file system?
10. What does mkfs need to be run before we mount a file system?
11. Given a block number, how do you read it off the disk?
12. Given an inode number, how do you read it off the disk?

Use the simulator (vsfs.py) given at the end of the book chapter!