CS 537 Handout Nov 10

- Unix File System (UFS)
    - similar to simple file system developed in class
    - uses a free list to manage free blocks on disk (not bitmaps)
- Key problem: UFS treated disk as if it was memory
    - data spread all over the disk
    - file data far away from file inode
    - related files may be far away from each other
    - free list resulted in fragmentation
    - small block size: 512 bytes
- Fast File System (FFS)
    - key approach: make file system “disk-aware”
    - Same interface as UFS, but different implementation
- Main structure: cylinder group (CG)
    - divide disk into regions called cylinder groups
- Main policy: put related things nearby each other
    - Put new directory in CG with lowest num of directories and highest num of free inodes
    - Data blocks in same CG as inode of file
    - Files in same directory in same CG
- Handling large files
    - Why not just fill up a CG with a file?
    - Divide up large file into chunks: each chunk in a different CG
        - This will reduce performance due to seeking
        - Amortize performance reduction
        - Direct blocks in one CG
        - Each indirect block in another CG
- Handling internal fragmentation
    - Sub blocks: 512 bytes in size
    - Blocks: 4096 bytes in size
    - When file is getting created from size zero, first allocate sub blocks
    - When size >= 4096, copy out data into a block, free sub blocks
    - Modify libc: buffer writes until enough for a full block
- Optimized disk layout
    - Old disks read block by block
    - Disk reads block 0, transfers to operating system, oops block 1 has rotated away
    - Technique: parametrization
        - How many blocks to skip so that sequential reads dont have rotational delay
    - Handled on modern disks using a track buffer
- Long file names, symbolic links!

- Atomic rename()

Questions:

1. What was the main problem with the old unix file system?
2. How does the old unix file system differ from the simple file system we developed in class?
3. How does the UFS perform over time? Does it performance increase/decrease? Why?
4. Why is a bitmap better than a free list to manage free space? What does it perform better?
5. Look at the SEER trace analysis in the book chapter. What is a type of locality that FFS does not capture?
6. How big does the chunk size of a large file need to be so that we send 80% of the time transferring data and the rest 20% in seeking? (ignore rotational delays).
7. When libc buffers a write, is this visible to the file system? Is it in the page/buffer cache?
8. Why is the atomic rename() important for applications?
9. Why aren’t disk blocks bigger than 4 KB? Bigger size helps get better transfer rates, so why not make disk blocks huge like 4 MB?
10. In relation to the previous question, suppose the only change we made to UFS was to introduce bigger blocks. Would this solve all its problems? Why or why not?