

CSE 451: Operating Systems

Winter 2017

Module 16

BSD UNIX Fast File System

Mark Zbikowski
mzbik@cs.washington.edu
Allen Center 476

File system implementations

- We've looked at disks
- We've looked at file systems generically
- We've looked in detail at the implementation of the original Bell Labs UNIX file system
 - a great *simple yet practical* design
 - exemplifies engineering tradeoffs that are pervasive in system design
- Now we'll look at some more advanced file systems
 - First, the Berkeley Software Distribution (BSD) UNIX Fast File System (FFS)
 - enhanced performance for the UNIX file system
 - at the heart of most UNIX file systems today

BSD UNIX FFS

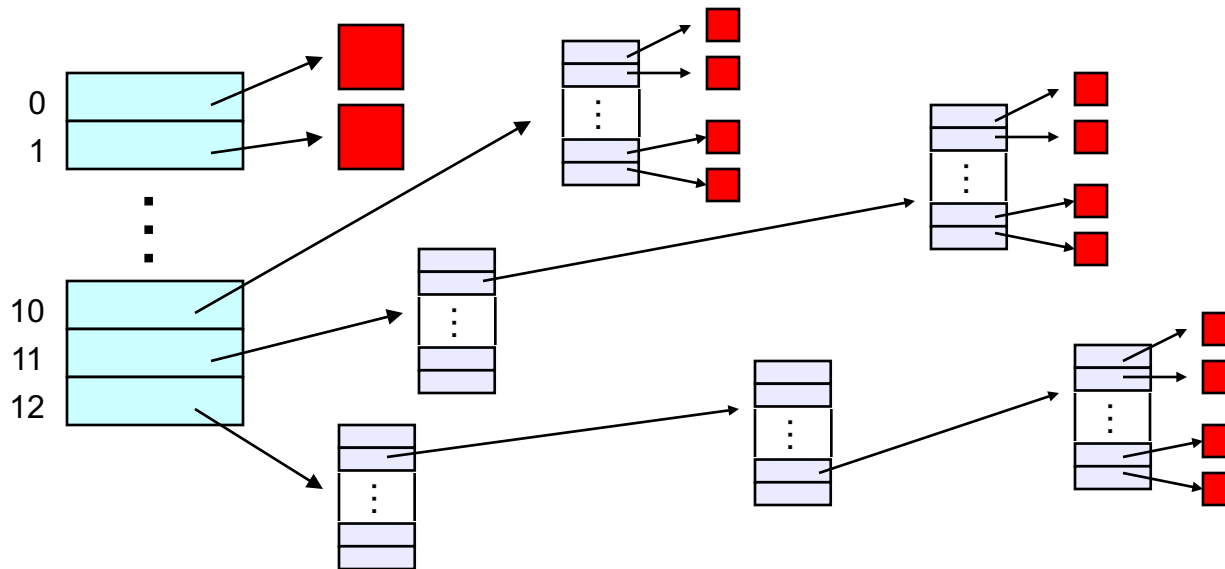
- Original (1970) UNIX file system was elegant but slow
 - poor disk throughput
 - far too many seeks, on average
- Berkeley UNIX project did a redesign in the mid '80's
 - McKusick, Joy, Fabry, and Leffler
 - improved disk throughput, decreased average request response time
 - principal idea is that FFS is aware of disk structure
 - it places related things on nearby cylinders to reduce seeks

Recall the UNIX disk layout

- Boot block
 - can boot the system by loading from this block
- Superblock
 - specifies boundaries of next 3 areas, and contains head of freelists of inodes and file blocks
- i-node area
 - contains descriptors (i-nodes) for each file on the disk; all i-nodes are the same size; head of freelist is in the superblock
- File contents area
 - fixed-size blocks; head of freelist is in the superblock
- Swap area
 - holds processes that have been swapped out of memory

Recall the UNIX block list / file content structure

- directory entries point to i-nodes – file headers
- each i-node contains a bunch of stuff including 13 block pointers
 - first 10 point to file blocks (i.e., 512B blocks of file data)
 - then single, double, and triple indirect indexes



UNIX FS data and i-node placement

- Original UNIX FS had three major performance problems:
 - data blocks are allocated randomly in aging file systems
 - blocks for the same file allocated sequentially when FS is new
 - as FS “ages” and fills, it needs to allocate blocks freed up when other files are deleted
 - deleted files are essentially randomly placed
 - so, blocks for new files become scattered across the disk!
 - data blocks are relatively small
 - reduces fragmentation, but exacerbates the problem above
 - i-nodes are allocated far from blocks
 - all i-nodes at beginning of disk, far from data
 - traversing file name paths, manipulating files, directories requires going back and forth from i-nodes to data blocks
- All three of these generate many long seeks!
 - gets worse as disks get bigger

FFS: Cylinder groups

- FFS addressed the first and third problems using the notion of a cylinder group
 - disk is partitioned into groups of cylinders
 - data blocks from a file are all placed in the same cylinder group
 - files in same directory are placed in the same cylinder group
 - i-node for file placed in same cylinder group as file's data
- Introduces a free space requirement
 - to be able to allocate according to cylinder group, the disk must have free space scattered across all cylinders
 - in FFS, 10% of the disk is reserved just for this purpose!
 - good insight: keep disk partially free at all times!
 - this is why it may be possible for df to report >100% full!

FFS: Increased block size, fragments

- The original UNIX FS had 512B blocks
 - even more seeking
 - small maximum file size (~1GB maximum file size)
- Then a version had 1KB blocks
 - still pretty puny
- FFS uses a 4KB blocksize
 - big improvement in disk throughput - fewer seeks when transferring large files
 - allows for very large files (4TB) – exponential impact of indirect index
 - but, introduces internal fragmentation
 - on average, each file wastes 2K!
 - why?
 - worse, the average Unix file size is only about 1K!
 - why?
 - fix: introduce “fragments”
 - 1KB pieces of a block

FFS: Aggressive File Buffer Cache

- Exploit locality by caching file blocks in memory
 - the cache is system wide, shared by all processes
 - even a small (4MB) cache can be very effective (why?)
 - many FS's "read-ahead" into buffer cache
- What about writes?
 - some apps assume data is on disk after write
 - either "write-through" the buffer cache
 - or "write-behind"
 - maintain queue of uncommitted blocks, periodically flush. Unreliable!
 - NVRAM: write into battery-backed RAM. Expensive!
 - LFS, JFS: we'll talk about this soon!
- Buffer cache issues:
 - competes with VM for physical frames
 - integrated VM/buffer cache?
 - need replacement algorithms here
 - LRU usually

FFS: Awareness of hardware characteristics

- Original UNIX FS was unaware of disk parameters
- FFS parameterizes the FS according to disk and CPU characteristics
 - e.g., account for CPU interrupt and processing time, plus disk characteristics, in deciding where to lay out sequential blocks of a file, to reduce rotational latency

FFS: Performance

- This was a long time ago – look at the relative performance, not the absolute performance!

Type of File System	Processor and Bus Measured	Speed	Read Bandwidth	% CPU
<u>old 1024</u>	750/UNIBUS	29 Kbytes/sec	<u>29/983 3%</u>	11%
<u>new 4096/1024</u>	750/UNIBUS	221 Kbytes/sec	<u>221/983 22%</u>	43%
<u>new 8192/1024</u>	750/UNIBUS	233 Kbytes/sec	<u>233/983 24%</u>	29%
<u>new 4096/1024</u>	750/MASSBUS	466 Kbytes/sec	<u>466/983 47%</u>	73%
<u>new 8192/1024</u>	750/MASSBUS	466 Kbytes/sec	<u>466/983 47%</u>	54%

(983KB/s is theoretical disk throughput)

(block size / fragment size)

Table 2a – Reading rates of the old and new UNIX file systems.

Type of File System	Processor and Bus Measured	Speed	Write Bandwidth	% CPU
<u>old 1024</u>	750/UNIBUS	48 Kbytes/sec	48/983 5%	29%
<u>new 4096/1024</u>	750/UNIBUS	142 Kbytes/sec	142/983 14%	43%
<u>new 8192/1024</u>	750/UNIBUS	215 Kbytes/sec	215/983 22%	46%
<u>new 4096/1024</u>	750/MASSBUS	323 Kbytes/sec	<u>323/983 33%</u>	94%
<u>new 8192/1024</u>	750/MASSBUS	466 Kbytes/sec	<u>466/983 47%</u>	95%

(CPU maxed doing block allocation!)

Table 2b – Writing rates of the old and new UNIX file systems.

FFS: Faster, but less elegant (warts make it faster but ugly)

- Multiple cylinder groups
 - effectively, treat a single big disk as multiple small disks
 - additional free space requirement (this is cheap, though)
- Bigger blocks
 - but fragments, to avoid excessive fragmentation
- Aggressive File Buffer Cache
- Aware of hardware characteristics
 - ugh!