ECE 598 – Advanced Operating Systems
Lecture 19

Vince Weaver

`http://web.eece.maine.edu/~vweaver`

`vincent.weaver@maine.edu`

10 April 2018

Announcements

- Homework #9 was posted
- Don't forget project update is due Thursday

Homework 8 Review

- Code: any problems? Rounding? Lack of floating point an issue. Can we use floating point, did anyone try it?
- Meaningful timestamp, I meant birthday or something, but really anything is fine
- Questions
  - ext2: directory entry has filename, inode does not have it. Multiple can point to same inode
  - Indirect blocks:

    No-indirect: **12k** (can you store contents *in* the inode?)

One indirect: 1k/4=(256*1k)+12k = **268k**

Two indirect: 12k+256k+(256*256*1k)= roughly **67MB**

Three indirect: 12k+256k+ (256*256*1k)+ (256*256*256*1k)=**16GB**

Overhead: 1k (for inode)+ 1k (for indirect) + (1k+256k) + 1k+256k+256*256k = roughly **64M**

- How is ext2 better?

  faster? (if you say this, say how, when, why)

  better filenames, less fragmentation, larger filesizes,

  permissions
  - How is FAT better?
    simple to implement, small mem requirements, compatibility
  - sparse file/file with holes
  - Filesystems: UFS (bsd), cramfs/squashfs, coda/AFS, Fuse based (plan9 mount the www?), cryptofs, cephfs, NTFS, sysfs, xfs, hfs+
- Filesizes. Megabytes (MB) vs Mebibyte (MiB)

Advanced Filesystems

btrfs

- Butter-fs? Butter-fuss? B-tree fs?
- Started in 2007 at Oracle (by Chris Mason, who had worked on Reiserfs)
- Address scaling
- Lack of pooling, snapshots, checksums in Linux
  - Pooling – preallocate resources so they can be quickly handed out when needed
  - Snapshots – instead of taking full backup (long) just take a snapshot of current state and then keep using

filesystem

  - Checksums – mathematically check to make sure values in files are what they should be

- $2^{64} = 16$ Exabyte file size limit (Linux VFS limits you to 8EB)
- Space-efficient packing small files
- Dynamic inode allocation

btrfs details

- Primary data structure is a copy-on-write B-tree
  - B-tree similar to a binary tree, but with pages full of leaves
    allow searches in logarithmic time
  - Btrees also used by ext4, NTFS, HFS+
  - Goal is to be able to quickly find disk block X
  - Copy-on-write when writing to file, rather than over-write (which is what traditional filesystems do)
  - Copy on write. When write to a file, old data not

overwritten. Since old data not over-written, crash recovery better

Eventually old data garbage collected

- Data in extents
- Copy-on-write
- Forest of trees:
  - sub-volumes
  - extent-allocation
  - checksum tree
  - chunk device
  - reloc

- On-line defragmentation
- On-line volume growth
- Built-in RAID
- Transparent compression
- Snapshots
- Checksums on data and meta-data, on-line data scrubbing
- De-duplication
- Cloning, reflinks
  - can make an exact snapshot of file, copy-on-write
  - different inodes, initially point to same blocks

  - different from hardlink (different dir entry, point to same inode)
- In-place conversion from ext3/ext4
- Superblock mirrors – at 64k, 64MB,256GB, and 1PB. All updated at same time. Has generation number. Newest one is used.

ZFS

- Advanced OS from Sun/Oracle
- 128-bit filesystem (opposed to btrfs which is 64-bit)
  Running out of space would require 10^{24} 3TB hard drives
- Not really included in Linux due to licensing issues (CDDL vs GPL2)
  Was originally proprietary, then open source, then proprietary again (with open fork)
- Vaguely similar in idea to btrfs
- indirect still, not extent based?

- Acts as both the filesystem *and* the volume manager (RAID array)
- Aim is to be super reliable, to know the state of underlying disks, make sure files stay valid, drives stay healthy
- Can take snapshots. Can roll back if something goes wrong.
- Checksums. Stored in parent. Other fs stores with file metadata so if that lost then checksum also lost
- Limitations: needs lots of RAM and lots of free disk space (due to copies and snapshots). If less than 80%

free then goes to space-conserve mode rather than high-performance

- Supports encryption (btrfs doesn't yet)

ReFS

- Resilient FS, codename “Protogon”
- Microsoft’s answer to btrfs and zfs
- Windows 8.1
- Initially removed features such as disk quotas, alt data streams, extended attributes (added later?)
- Uses B+ trees (not same as b-trees), similar to relational database
- All structures 64-bit
- Windows cannot be booted from ReFS

APFS

- New Apple OS for High Sierra and later, iOS 10.3 later
- Fix core problems of HFS+
- Optimized for solid-state drive, encryption
- 64-bit inode numbers
- checksums
- Crash protection: instead of overwriting metadata, creates new metadata, points to it, and only then removes old
- No hard-links to directories (most other OSes are like

this) but this breaks “Time Machine” backup

- HighSierra auto-converts flash-based drives

Networked File Systems

- Allow a centralized file server to export a filesystem to multiple clients.
- Provide file level access, not just raw blocks (NBD)
- Clustered filesystems also exist, where multiple servers work in conjunction.

NFS – Network File System (NFS2/3/4)

- Developed by Sun in the 80s.
- Stateless. Means server and client can reboot without the other noticing.
- A server, nfsd, exports filesystems as described in `/etc/exports`. The server can be in userspace or in the kernel
- Needs some sort of “file handle” unique value to specify value. Often cheat and use inode value. Problem with older version of protocol with only 32-bit handles.

- UDP vs TDP
- Read-ahead can help performance
- Cache consistency a problem. One way is to just have timeouts that flush data regularly (3-30s)
- List of operations (sort of like syscalls) sent to server
  read sends a packet with file-handle, offset, and length
  No open syscall; server has no list of open files. This way there is no state needed, can handle reboots.
- nfsroot

CIFS/SMB

- Windows file sharing.
- Poorly documented
- Samba reimplements it, originally reverse-engineered.

Virtual/Pseudo Filesystems

- Files do not exist on disk; they are virtual, fake files that the kernel creates dynamically in memory
- proc
- sys
- debugfs
- usbfs

Distributed / Cluster Filesystems

procfs

- Originally process filesystem. Each process gets a directory (named by the process id (pid)) under /proc Tools like `top` and `ps` use this info.
  - cmdline
  - cwd
  - environ
  - exe
  - fd
  - maps

- Eventually other arbitrary files were also included under proc, providing system information
  - cpuinfo
  - meminfo
  - interrupts
  - mounts
  - filesystems
  - uptime
- ABI issues – these files are part of the kernel, and even though the intention was that they could come and go at will, enough people write programs that depend

on them, the values cannot be easily changed without breaking the ABI

sysfs

- procfs was getting too cluttered, so sysfs was created
- intended to provide tree with information on devices
- one-item per file and strict documentation rule
- also hoped that it would replace sysctl() and ioctl() but that hasn't happened

Other Filesystem Features

- Holes – why store blocks of zeros in a file? Why not instead note when a file has a "hole" in it? This lets large files that are mostly zeros not take up much space on disk.

- Compression – transparently compress files. Does have some performance issues, write issues (do you have to decompress, write, then recompress?) and also files rarely compress to nice power-of-two sizes.

- Online fsck
- Defragmentation
- Undelete
- Secure Delete
- Snapshots
- Journaling
- De-dup

- Quotas – especially an issue on multi-user machines, you want to keep any one user from filling up the disk.
- Encryption
- Locking – may want to prevent more than one person writing a file at a time as it can get corrupted

Linux VFS

- VFS interface - VFS / Virtual Filesystem / Virtual Filesystem Switch
- Makes all filesystems look like Linux filesystems. Might need hacks; i.e. for FAT have to fake a superblock, directory entries, and inodes (generate on the fly). Can be important having consistent inode numbers as filesystems like NFS use them even across reboots.
- Objects
  - superblock

  - inode object (corresponds to file on disk)
  - file object – info on an open file (only exists in memory)
  - dentry object – directory entry.
- Can use default versions, such as default_llseek
- dentries are cached. As they get older they are freed.
- dentry operations tale. hash. compare (how you handle case sensitive filesystems)

Linux Filesystem Interface

- linux/fs.h
- Module. Entry point init_romfs_fs(), exit_romfs_fs()
  - init_romfs_fs() – register_filesystem()
    name, romfs_mount, romfs_kill_sb
  - romfs_mount – mount_bdev(), romfs_fill_super
  - sb$-$ $>$s_op$=$&romfs_super_ops();
  - romfs_iget() $- >$ i_op struct, gets pointed to in each inode

mounting

- Opens superblock
- Inserts into linked list of opened filesystems

pathname lookup

- If begins with /, starts with current$-$ >fs$-$ >root
- otherwise, relative path, starts with current$-$ >fs$-$ >path
- looks up inode for starting directory, then traverses until it gets to the one wanted
- the dentry cache caches directory entries so the above can happen without having to do any disk reads if the directory was used recently before

- the access rights of intervening directories must be checked (execute, etc)

- symbolic links can be involved

- you might enter a different filesystem

- Should you cache invalid file lookups?

open syscall

- `getname()` – safely copies name we want to open from userspace process
- get_unused_fd() to get the file descriptor
- calls filp_open()
  - creates new file structure
  - open_namei() – checks dentry cache first, otherwise hits disk and looks up dentry
  - lookup_dentry()
- validates and sets up the file

- returns a fd

FUSE

- Allows creating filesystem drivers in userspace
- Works on various OSes