ECE 598 – Advanced Operating Systems
Lecture 16

Vince Weaver

http://www.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

26 March 2015

Announcements

- I'll be at ISPASS on Tuesday the 31st, so class is canceled.
- HW#5 will be posted soon.

Filesystems Continued

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 wihtout 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

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
- Inerts 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 (excute, etc)
- symbolic links can be involved
- you might enter a different filesystem
- Should you cache invalid file lookups?

open syscall

- getname() to get name from process
- get_unused_fd() to get the file descriptor
- calls filp_open()
  - creates new file structure
  - open_namei()
  - lookup_dentry()
- validates and sets up the file

- returns a fd

FUSE

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