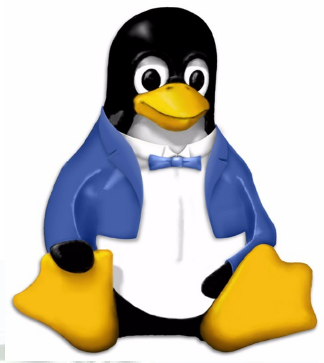


Linux Filesystems in ~~21 days~~ 45 minutes ...

A Step by Step Introduction to Writing (or Understanding) a Linux Filesystem

Steve French
Linux Filesystems/Samba Design
IBM Linux Technology Center

http://svn.samba.org/samba/ftp/cifs-cvs/samplefs.tar.gz

Legal Statement

Outline

- Who am I?
- FS background info
- Days 1 through 7, basic ops
- FS Data Structures
- Days 8 and 9
- FS Operations
- Days 10 - 11 finished our sample
- Days 12 - 21 – advanced ops

Who Am I?

- Author and maintainer of Linux cifs vfs (for accessing Samba, Windows and various SMB/CIFS based NAS appliances)
- Member of the Samba team, coauthor of CIFS Technical Reference and former SNIA CIFS Working Group chair
- Architect for Filesystems/NFS/Samba in IBM LTC

Note ...

- Linux is constantly evolving, interfaces change (even those used by these small samples)!
  - See Documentation directory of kernel (e.g. feature-removal-schedule.txt) or even better kernel source itself if unsure about call
  - The kernel page of lwn.net is particularly useful: summarizing kernel API changes
  - Examples were built and/or lightly tested on 2.6.16.11 and 2.6.18-rc1

What is a Filesystem ["file system"]?

- "a file system is a set of abstract data types that are implemented for the storage, hierarchical organization, manipulation, navigation, access, and retrieval of data" [http://en.wikipedia.org/wiki/Filesystem]
- A Linux kernel module used to access files and directories. A filesystem provides access to this data for applications and system programs through consistent, standard interfaces exported by the VFS, and enables access to data that may be stored persistently on local media or on remote network servers/devices, or even transient data (such as debug data or kernel status) stored temporarily in RAM or special devices.

Linux ... perfect fs experimental platform?

- Linux is easily available and usable filesystems can be smaller in Linux than many other OS
  - e.g. Ramfs is working 390 LOC example (simpler shmem/tmpfs) and stubs in fs/libfs.c make starting easy
  - Filesystems average under 30KLOC
- Lots of samples in kernel:
  - >12 (general) local filesystems (e.g. ext2/3, reiserfs, xfs, UDF, ...)
  - >16 special purpose local fs
  - 8 network/cluster fs (nfs, cifs, ocfs2...)

Some common Linux FS

FS Name	Type	Approx. size (1000 LOC)
Ramfs	Local	0.4
Sysfs	Spec. purp.	2
Proc	Spec. purp.	4
FUSE	Spec. purp.	4
Smbfs (obsol)	network	6
Ext3	Local	12
NTFS	Local	17
JFS	Local	18
CIFS	Network	21
Reiserfs	Local	22
NFS	Network	24
OCFS2	Cluster	31
XFS	Local	71

samplefs

- Goals for samplefs
  - Small, understandable (yet extend existing ramfs and rkfs samples)
  - Easy to compile on reasonably current (e.g. 2.6.9 or later) kernels
  - Demonstrate basic data structures and concepts that would help one:
    - Implementing a new fs for Linux
    - Experimenting with fs Linux
    - Learning Linux fs concepts to help debugging and/or tuning Linux

Day 1: Basic Module 101...

- A Linux filesystem kernel driver:
  - Can be built as distinct module or
  - Can be built into vmlinuz itself
- Kernel modules usually have:
  - Entry in Kconfig (./fs/Kconfig)
  - New directory (fs/samplefs)
  - Makefile (fs/samplefs/Makefile)
  - C Code to do module init/remove

Day 1 – key data structure

- We fill in struct file_system_type and information describing our filesystem kernel module
  - Name of fs
  - Method to allocate/kill superblock

Day 1: cont

- gunzip samplefs.tar.gz
- cd to root of kernel source tree
- If you have not built before and .config file does not exist (create config file that matches your installed kernel or do “make oldconfig”)
- make menuconfig (or equivalent)
  - Select “File systems” then [M] for “Sample Filesystem (Experimental)”
- make M=~/samplefs/day1

Day 1 - status

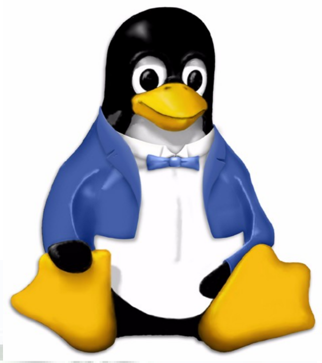
- What can we do with our little sample now?
- (as root) /sbin/insmod ~/samplefs/day1/samplefs.ko
- /sbin/lsmod shows our module loaded
- /sbin/rmmod will remove it

Day 2 Mount – What is a superblock

- Each mount has a superblock which contains information about the mount
- Key data: "struct super_block" and "struct vfsmount"
- Fill in superblock operations and create the first (root) inode

Day 2 - continued

- make M=~/samplefs/day2
- Note:
  - Addition of routines to allocate/free superblock and parse mount options
  - Your filesystem (like the case of fs/ramfs) may not need to have fs specific sb info or mount options but this is provided as an example



Day 2 - status

- Mount code added, but mount would oops in get_sb_nodev if attempted
- Time to add debug code

Day 3 Control and debug our filesystem

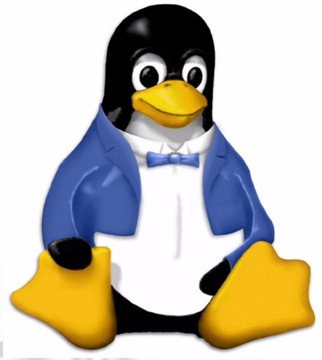
- “dmesg” displays kernel error/warning messages
- Lets fix msg: “samplefs: module license 'unspecified' taints kernel.”
- Also will add debug messages (errors, warnings and/or information messages) for various conditions
- Add optional enablement for /proc which allows debug information and settings to be easily displayed (may move to sysfs and/or debugfs)
- make M=~/samplefs/day3

Day 3 - status

- Loading and unloading module now displays inform. messages to dmesg
- Added an entry in /proc/fs for future e.g. runtime debugging or status information
- Added ability to parse parms passed on module init
- Debug code allowed us to isolate problem with iget line
- Example of debug statement depending on menuconfig setting

Day 4 What is an inode?

- An inode is a representation of a file and its metadata (timestamps, type, size, attributes) but not its name
- Inodes can represent files, directories (containers of files), symlinks and special files
- Fill in function pointers to inode and file (open file) operations
- Fill in inode metadata (uid owner, mode, file timestamps, etc.)



Day 4 - status

- We can mount now (e.g. “mount -t samplefs any /mnt”)
- “cat /proc/mounts”
- “stat -f /mnt”
- “stat /mnt”

Day 5 What is a dentry?

- The dcache contains dentries, and provides a fast way to lookup inodes based on a specific pathname
- The dentries for individual path components (parent directory, parent of parent, etc.) of a file name form a hierarchy
- A file inode can have multiple different dentries pointing to it (e.g. Due to hardlinks)

Day 5 - continued

- Our dentry operations do not save negative dentries as most fs do
- If a filesystem exports same data through other mechanisms (e.g. Other nodes in a network or cluster filesystem), then d_revalidate would normally be exported
- Our example adds case insensitive support (mount option ignorecase)

Day 5 - status

- Still need to add some operations, time to work on inode operations

Day 6 simple inode operations

- Time to add
  - create
  - mkdir
  - unlink (delete)
  - rmdir
  - mknod

Day 6 - status

- Now … like magic, we can do simple operations like mkdir and mknod
- But readdir (“ls /mnt”) yields “Not a directory”
- And open (“touch /mnt/file ; cat /mnt/file”) gets “Invalid argument

Day 7 simple file operations

- Files AND directories have file operations
- Those ops for the directory (to support e.g. Readdir) are easy in this type of Linux fs ... but what about for files?
- We could add the simplest set of file operations
  - Read
  - Write
  - (some fs also need seek and fsync)
- But Linux has a powerful page cache that is not much harder to use for this

Day 7 - status

- We can now do “ls /mnt1” (after creating some test files) and get expected output
- Now for some review of the various structures we have talked about and their relationship …

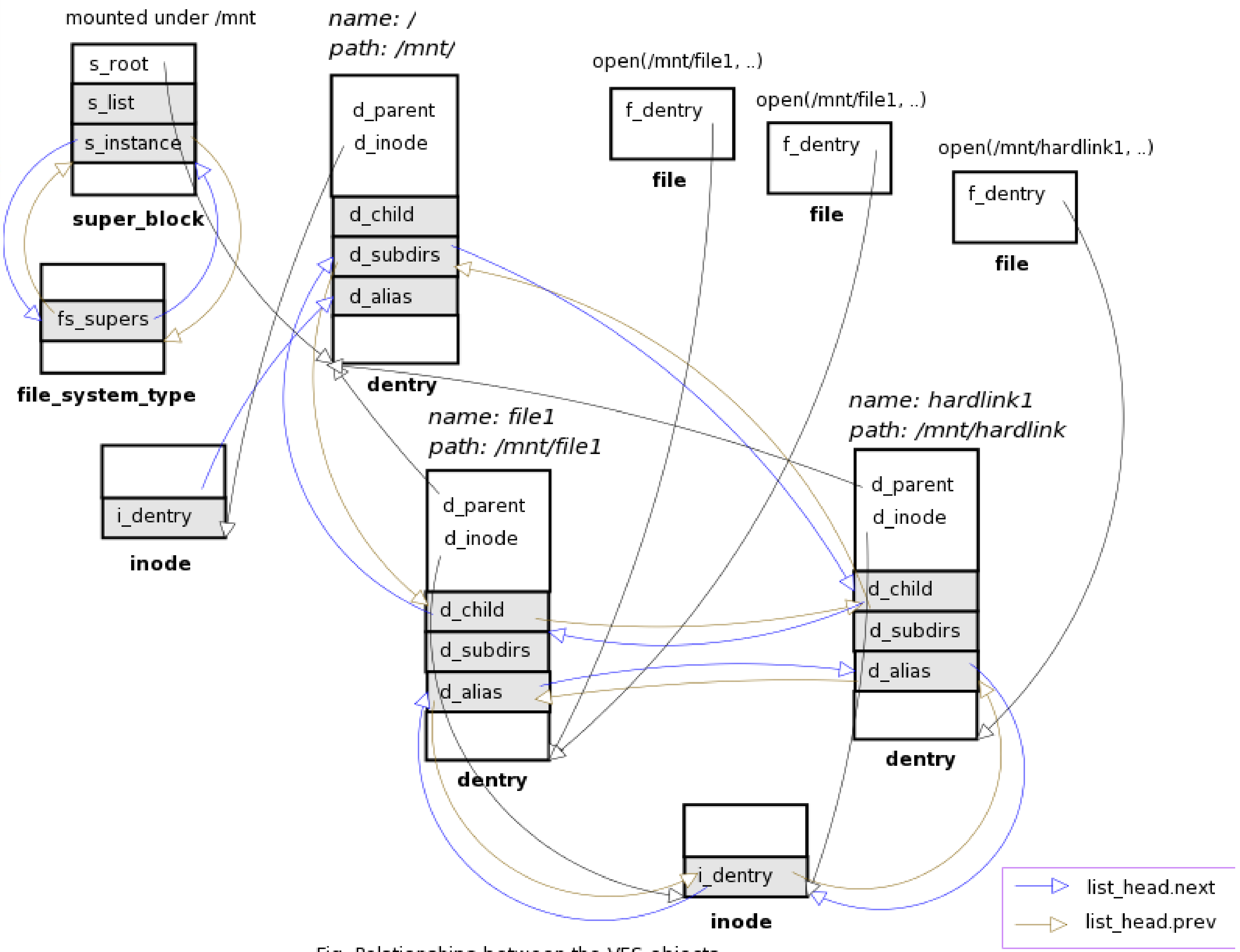


Fig: Relationships between the VFS objects

Source: http://www.geocities.com/ravikiran_uvs/articles/rkfs.html

Day 8 Opening a file

- A “File” (struct file) represents an open instance of an inode (for a particular pathname, with particular open flags)
- The call for “open” (or “create” or “mkdir” etc.) in userspace do not map atomically into just one corresponding call into the fs unfortunately, but Linux filesystems can use the “intent” fields on lookup and create to improve efficiency

Day 9 Introducing the page cache

- Lets add calls to our driver to use the generic page cache
  - File operations map via
    - Generic_file_read
    - generic_file_write
  - To readpage
  - And writepage
  - With or without mmap

Day 9 - status

- We can create files, write data and read data and do most common file operations
- So lets review how these operations tie together ....

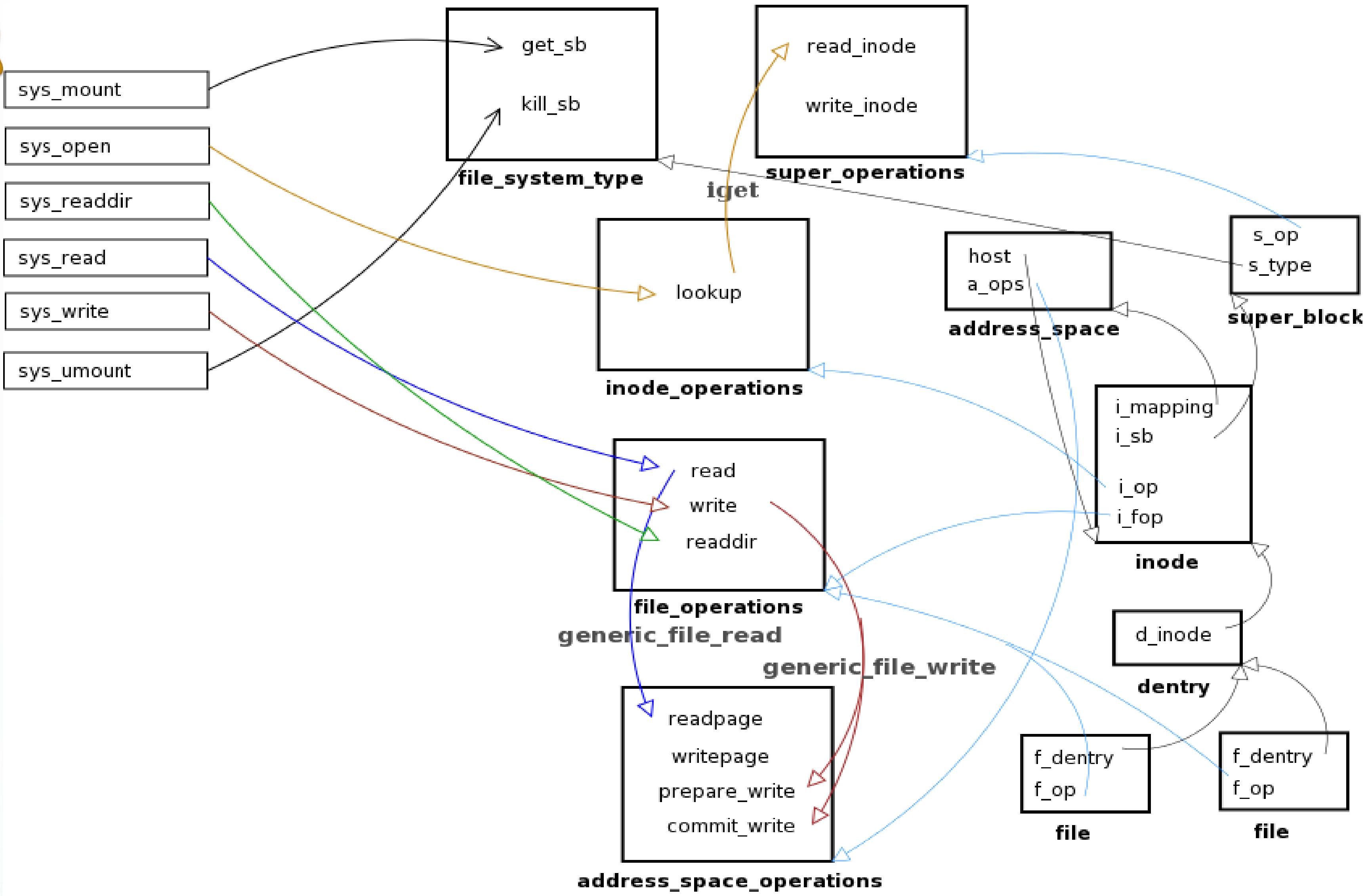


Fig: System call mapping and data structure relationships

Source: http://www.geocities.com/ravikiran_uvs/articles/rkfs.html

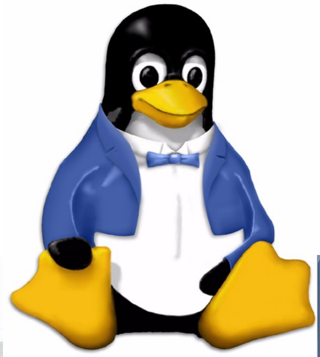
Day 10 Multipage operations

- Linux has two interesting high performance page cache read/write op
  - Readpages (nine filesystems use)
  - Writepages (eight filesystems use including ext3, nfs and cifs)
- Useful for coalescing reads/writes together (NB Some network filesystems like cifs typically negotiate buffer sizes much larger than a 4K page so this allows more efficiency)
- For cifs (unlike for samplefs) using writepages dramatically improved write performance over GigE

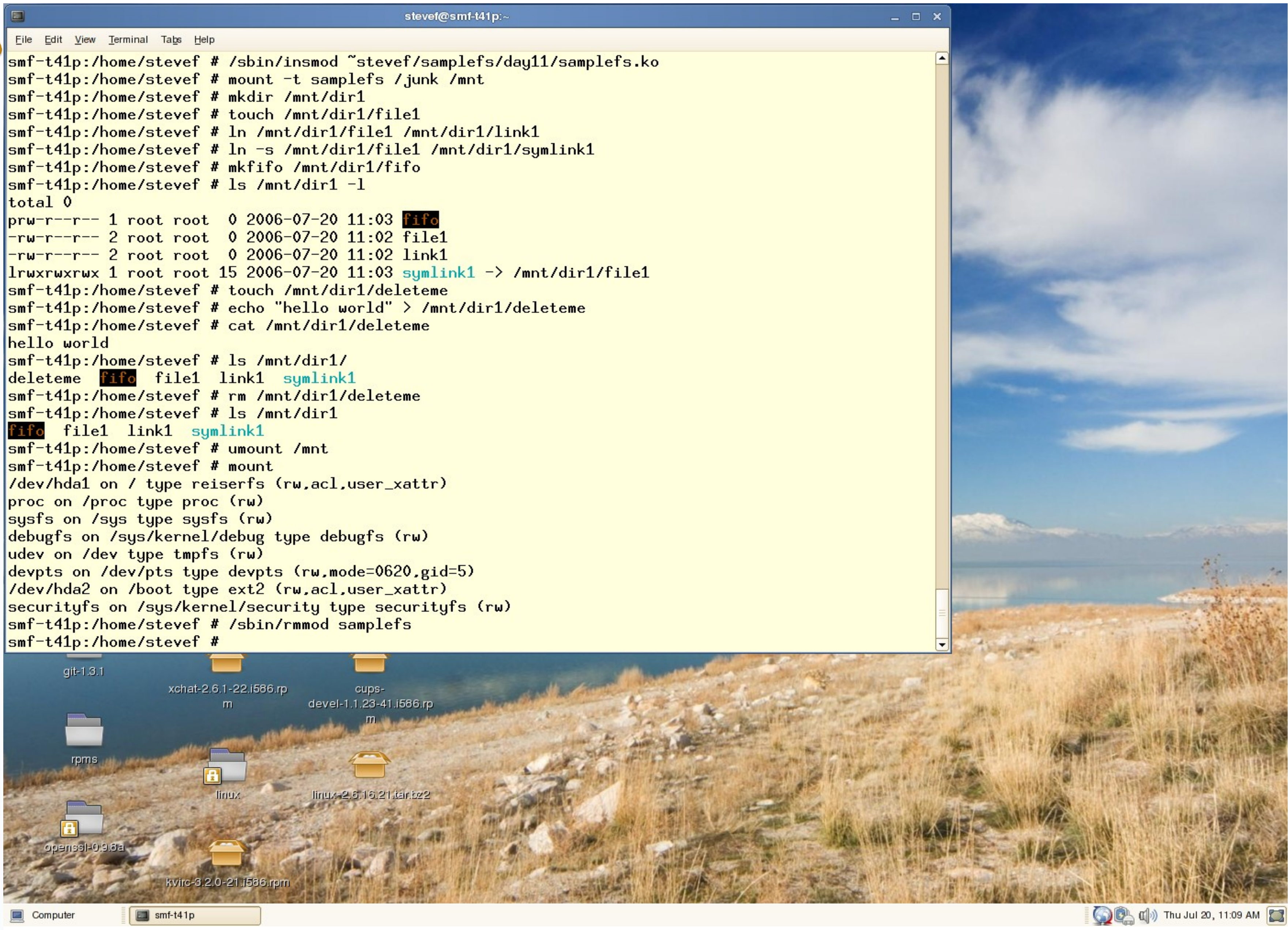
Day 11 Hardlinks and symlinks

- Adding hardlinks & symlinks easy
- Status after day 11 ... most important operations work:

```
smf-t41p:/usr/src/linux-2.6.16.11-7 # ls /mnt1 -l
total 0
drwxr-xr-x 2 root root  0 2006-07-18 23:53 dir
drwxr-xr-x 2 root root  0 2006-07-18 23:53 dir1
drwxr-xr-x 2 root root  0 2006-07-18 23:55 fifo1
-rw-r--r-- 1 root root  0 2006-07-18 23:53 file
-rw-r--r-- 3 root root  0 2006-07-18 23:53 file1
-rw-r--r-- 3 root root  0 2006-07-18 23:53 file2
lrwxrwxrwx 1 root root 11 2006-07-18 23:54 file3 -> /mnt1/file1
-rw-r--r-- 3 root root  0 2006-07-18 23:53 hardlinktofile1
```



It works!



```
smf-t41p:/home/stevef # /sbin/insmod ~stevef/samplefs/day11/samplefs.ko
smf-t41p:/home/stevef # mount -t samplefs /junk /mnt
smf-t41p:/home/stevef # mkdir /mnt/dir1
smf-t41p:/home/stevef # touch /mnt/dir1/file1
smf-t41p:/home/stevef # ln /mnt/dir1/file1 /mnt/dir1/link1
smf-t41p:/home/stevef # ln -s /mnt/dir1/file1 /mnt/dir1/symlink1
smf-t41p:/home/stevef # mkfifo /mnt/dir1/fifo
smf-t41p:/home/stevef # ls /mnt/dir1 -l
total 0
prw-r--r-- 1 root root  0 2006-07-20 11:03 fifo
-rw-r--r-- 2 root root  0 2006-07-20 11:02 file1
-rw-r--r-- 2 root root  0 2006-07-20 11:02 link1
lrwxrwxrwx 1 root root 15 2006-07-20 11:03 symlink1 -> /mnt/dir1/file1
smf-t41p:/home/stevef # touch /mnt/dir1/deleteme
smf-t41p:/home/stevef # echo "hello world" > /mnt/dir1/deleteme
smf-t41p:/home/stevef # cat /mnt/dir1/deleteme
hello world
smf-t41p:/home/stevef # ls /mnt/dir1/
deleteme  fifo  file1  link1  symlink1
smf-t41p:/home/stevef # rm /mnt/dir1/deleteme
smf-t41p:/home/stevef # ls /mnt/dir1
fifo  file1  link1  symlink1
smf-t41p:/home/stevef # umount /mnt
smf-t41p:/home/stevef # mount
/dev/hda1 on / type reiserfs (rw,acl,user_xattr)
proc on /proc type proc (rw)
sysfs on /sys type sysfs (rw)
debugfs on /sys/kernel/debug type debugfs (rw)
udev on /dev type tmpfs (rw)
devpts on /dev/pts type devpts (rw,mode=0620,gid=5)
/dev/hda2 on /boot type ext2 (rw,acl,user_xattr)
securityfs on /sys/kernel/security type securityfs (rw)
smf-t41p:/home/stevef # /sbin/rmmod samplefs
smf-t41p:/home/stevef #
```

Halfway through 21 days we have a complete FS!

- Good stopping point for our code sample ... (otherwise the sample could get too complex, or require a different backend storage mechanism)
- But
  - Some additional fs concepts may be helpful
  - many Linux filesystems implement more advanced features that are worth discussing in more detail

Day 12 inodes continued - Changing inode metadata

- Setattr is a key call
  - Based on the iattr struct passed in can change
    - Uid, gid, mode, timestamps, size,
    - Linux has three timestamps (atime, mtime, ctime) does not have a way of returning create timestamp
    - Linux allows an fs to specify a time granularity via s_time_gran mask (cifs e.g. reports time in 100 nanosecond units, jfs uses even better 1 ns, but ext2/ext3 timestamps much worse)
- Other “attributes” are changeable through xattrs and ioctls

Day 13 readdir

- Especially for network filesystems "ls" can cause "readdir" storms (hurting performance) by immediately following readdir with lots of expensive stat calls (unless the stat results are requested together, or cached)

Day 14 byte range locks, leases/distributed caching

- Linux supports the standard POSIX byte range locking but also supports “leases”
- F_SETLEASE, F_GETLEASE, used by programs like Samba server, help allow servers to offer safe distributed caching (e.g. “Oplock” [cifs] and “delegations” [nfs4]) for network/cluster filesystems

Day 15 Dir change tracking – inotify, d_notify

- There are two distinct mechanisms for change notification
  - Fcntl F_NOTIFY
  - And the newer, more general inotify

Day 16 kernel memory, object tracking

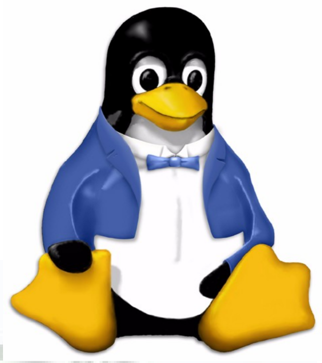
- Filesystems can take advantage of key mm features of Linux
  - Memory pools
  - Slab allocations (named, easy to track) for fixed size allocations
  - Kmalloc
  - Various optional Linux memory leak debugging facilities

Day 17 Address space mapping

- The few places where filesystems are not passed kernel memory, extra care must be taken to read/write to user memory
  - copy_to_user
  - copy_from_user
- Operations:
  - Read, Write (not using page cache)
  - Readlink
  - Some fcntls

Day 18 xattrs

- Xatts, similar in some ways to OS/2 “EAs” allow additional inode metadata to be stored
- This is particular helpful to Samba to store inode information that has no direct equivalent in POSIX, but a different category (namespace) also is helpful for storing security information (e.g. SELinux) and ACLs



Day 19 POSIX ACLs, permissions

- Since Unix Mode bits are primitive, richer access control facility was implemented (based on an expired POSIX draft for ACLs).
- Now working w/CITI, Andreas et al to offer optional standard NFS4 ACLs (NFSv4 ACLs loosely based on CIFS)
- POSIX ACLs are handled via xattr interface, but can be stored differently internal to the filesystem. A few filesystems (including CIFS and NFS) can get/set them natively to Linux servers

Day 19 continued

- ACL mapping works ok for:
  - POSIX ACL <-> Mode
- Being researched are the mappings to/from:
  - POSIX ACL <-> CIFS/NFSv4 ACL
  - CIFS/NFSv4 <->Mode
- Some Linux (out of kernel) fs such as GPFS already include NFSv4 ACL support in their Linux client

Day 20 Misc entry points: fcntl, ioctl

- Fcntl useful not just for get/setlease
- Ioctl includes two “semi-standard” calls which fs should consider implementing
  - getflags/setflags (chattr, lsattr on some other platforms)

Day 21 FS communication with userspace

- Various options including
  - Notify/read/write on a pseudofile
  - “connector” upcall (via cn.ko) which is nice in order to avoid netlink
  - Dbus
  - Ioctl (not recommended)

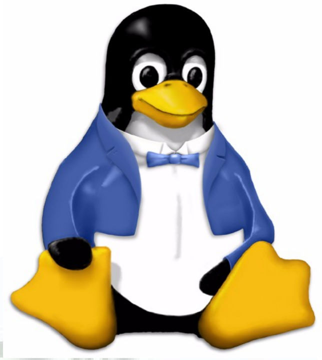
Future ...

- That is why we are here
- Linux filesystems continue to evolve and improve
- Research continues across the community in many key areas important for fs:
  - Large page, perf improvements
  - Duplicate mount optimizations
  - Offline caching
  - Credential management
  - Improved network and cluster fs ...

Thank you for your time!

For further reading ...

- General kernel module bckgrnd
  - O'Reilly books:
    - Linux Device Drivers 3rd edition (also online at http://lwn.net/Kernel/LDD3/)
    - Understanding the Linux Kernel 3rd edition
  - Prentice Hall book:
    - Linux Debugging and Performance Tuning: Tips and Techniques (http://vig.prenhall.com/catalog/academic/product/0,1144,0131492470,00.html)
  - Kernel API http://lwn.net/Articles/2.6-kernel-api/
  - Linux Kernel Module Programming Guide (http://www.tldp.org/LDP/lkmpg/2.6/html/)
  - Linux kernel documentation (Documentation directory of kernel source tree)



For further reading (cont)

- Linux Filesystems
  - Documentation/filesystems/vfs.txt
  - http://www.geocities.com/ravikiran_uvs/articles/rkfs.html
  - This presentation and samples:
    - http://linux-cifs.samba.org/cifs/samplefs.html
    - http://svn.samba.org/samba/ftp/cifs-cvs/samplefs.tar.gz
    - http://svn.samba.org/samba/ftp/cifs-cvs/ols2006-fs-tutorial-smf.odp