Linux File systems in 3 hours

Aneesh Kumar K.V– aneesh.kumar@linux.vnet.ibm.com
Kernel Team
IBM Linux Technology Center

19-Jan-2009

Agenda

- FS background info
- Basic Module
- Mount
- Support for file creation/deletion
- Support for reading directory contents
- Support for reading from/writing to files
- Support for hardlinks and symlinks

What is a File system ["filesystem"]?

- "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 file system 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
  - 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

Step1: Basic Module

- A Linux file system 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

Step1: 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

Step1: Howto Build and Test

- patch -p1 < patches/step1.diff
- make all
- make clean
- insmod ./samplefs.ko
- cat /proc/filesystems

  nodev   samplefs

Step1: Status

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

```
mount -t samplefs none /test/
        mount: wrong fs type, bad option, bad superblock on none,
        missing codepage or helper program, or other error
        (for several filesystems (e.g. nfs, cifs) you might
        need a /sbin/mount.<type> helper program)
        In some cases useful info is found in syslog - try
        dmesg | tail  or so
```

Step2: mount- what is a superblock

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

Step2: status

- Mount code added
- We can't create any directories or files
- We can't read the directory contents (no ls)

```
root@:/test# mount
none on /test type samplefs (rw)
root@:/test# ls
ls: reading directory .: Not a directory
root@:/test# mkdir p
mkdir: cannot create directory `p': Operation not permitted
root@:/test# touch a
touch: cannot touch `a': Permission denied
root@:/test#
```

Step3: What is an inode?

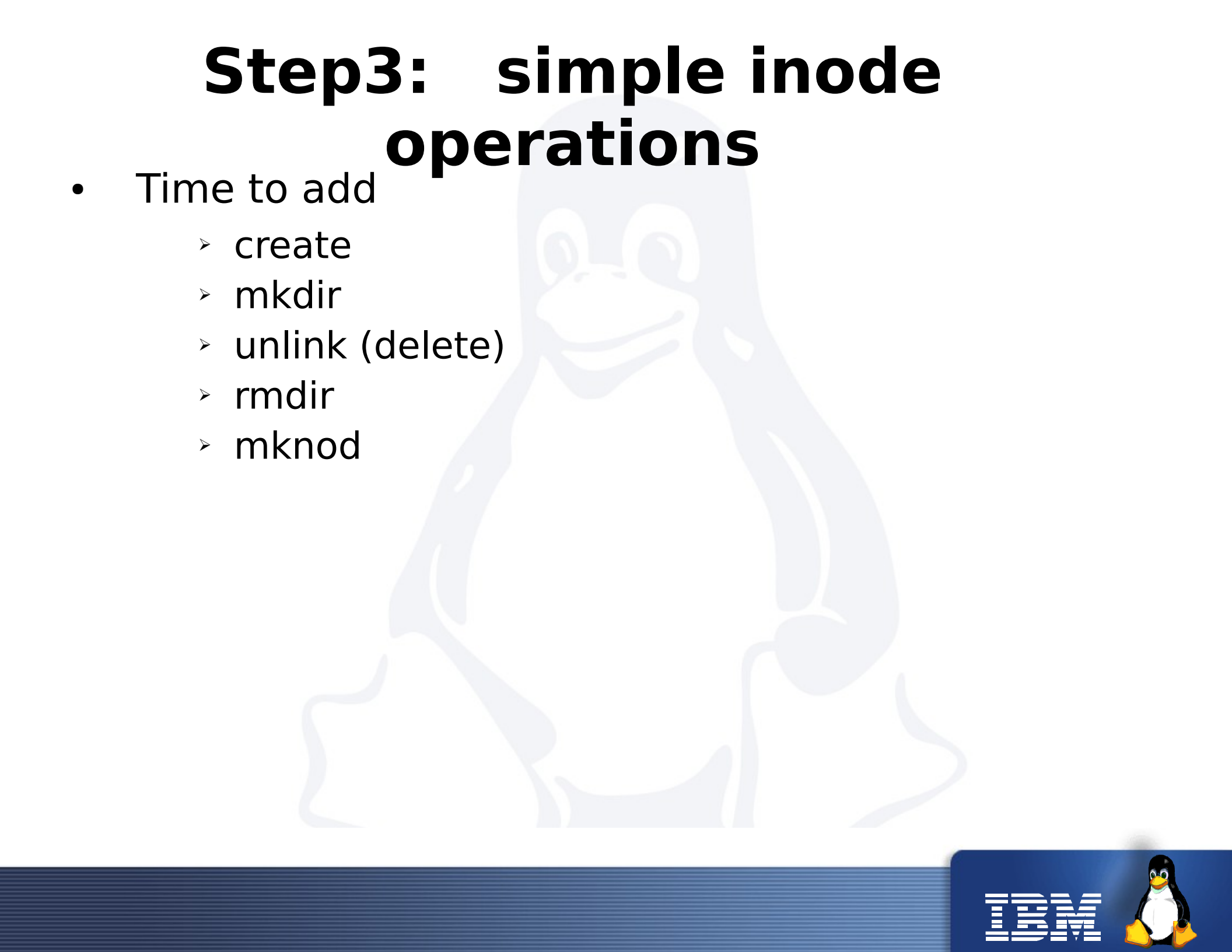
- An inode is a representation of a file and its metadata (timestamps, type, size, attributes) but not its name
- Inodes can represent files, directories (containers of files), symlinks and special files
- Fill in function pointers to inode and file (open file) operations
- Fill in inode metadata (uid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

Step3: 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)

Step3: simple inode operations

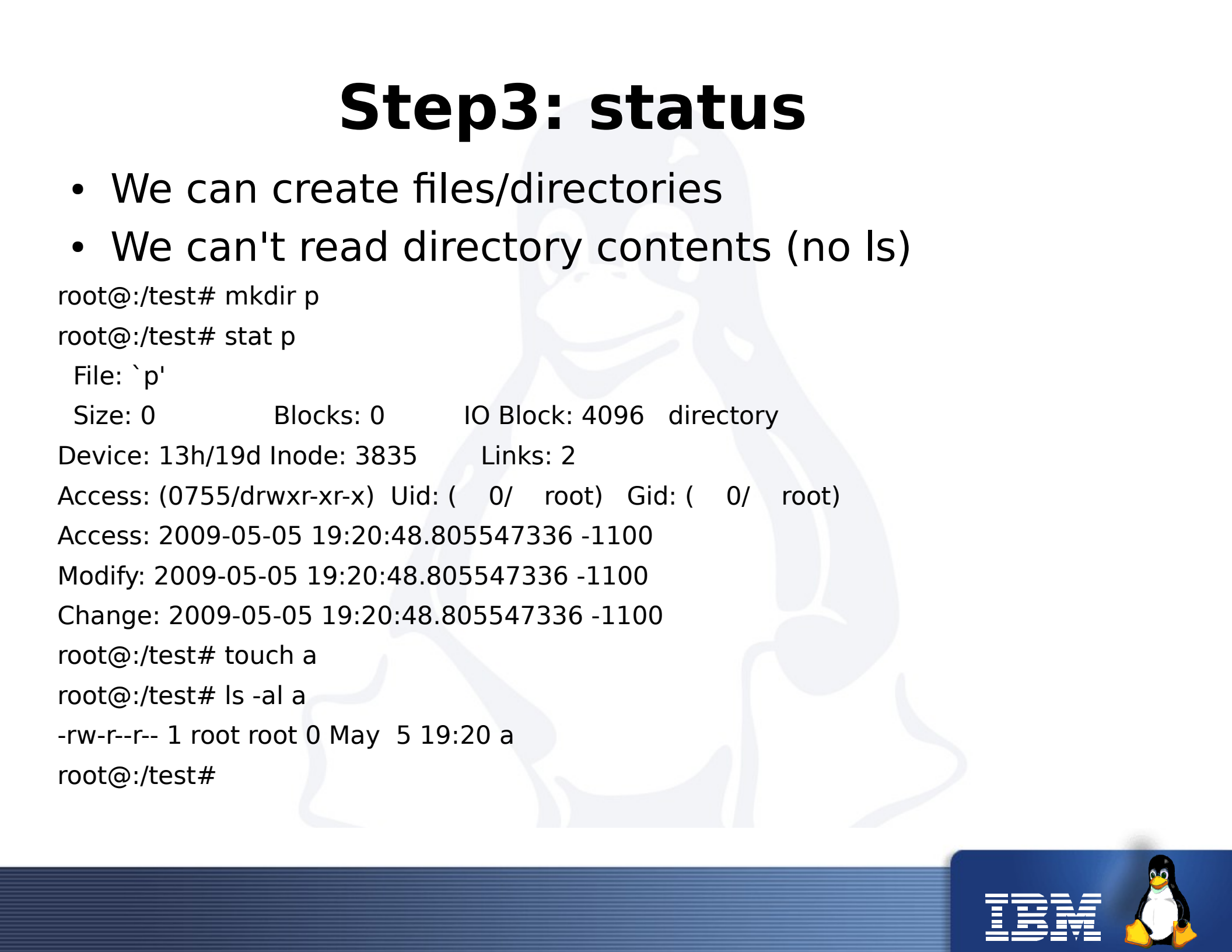
- Time to add
  - create
  - mkdir
  - unlink (delete)
  - rmdir
  - mknod



Step3: status

- We can create files/directories
- We can't read directory contents (no ls)

```
root@:/test# mkdir p
root@:/test# stat p
  File: `p'
  Size: 0               Blocks: 0          IO Block: 4096   directory
Device: 13h/19d Inode: 3835        Links: 2
Access: (0755/drwxr-xr-x)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 2009-05-05 19:20:48.805547336 -1100
Modify: 2009-05-05 19:20:48.805547336 -1100
Change: 2009-05-05 19:20:48.805547336 -1100
root@:/test# touch a
root@:/test# ls -al a
-rw-r--r-- 1 root root 0 May  5 19:20 a
root@:/test#
```



Step4: 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 .

Step4: status

- We can now read directory contents

```
root@:/test# mkdir p
root@:/test# touch t
root@:/test# ls -al
total 4
drwxr-xr-x  3 root root  224 May  5 19:52 .
drwxr-xr-x 21 root root 4096 May  4 22:28 ..
drwxr-xr-x  2 root root    0 May  5 19:52 p
-rw-r--r--  1 root root    0 May  5 19:52 t
root@:/test#
```

Step5: 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

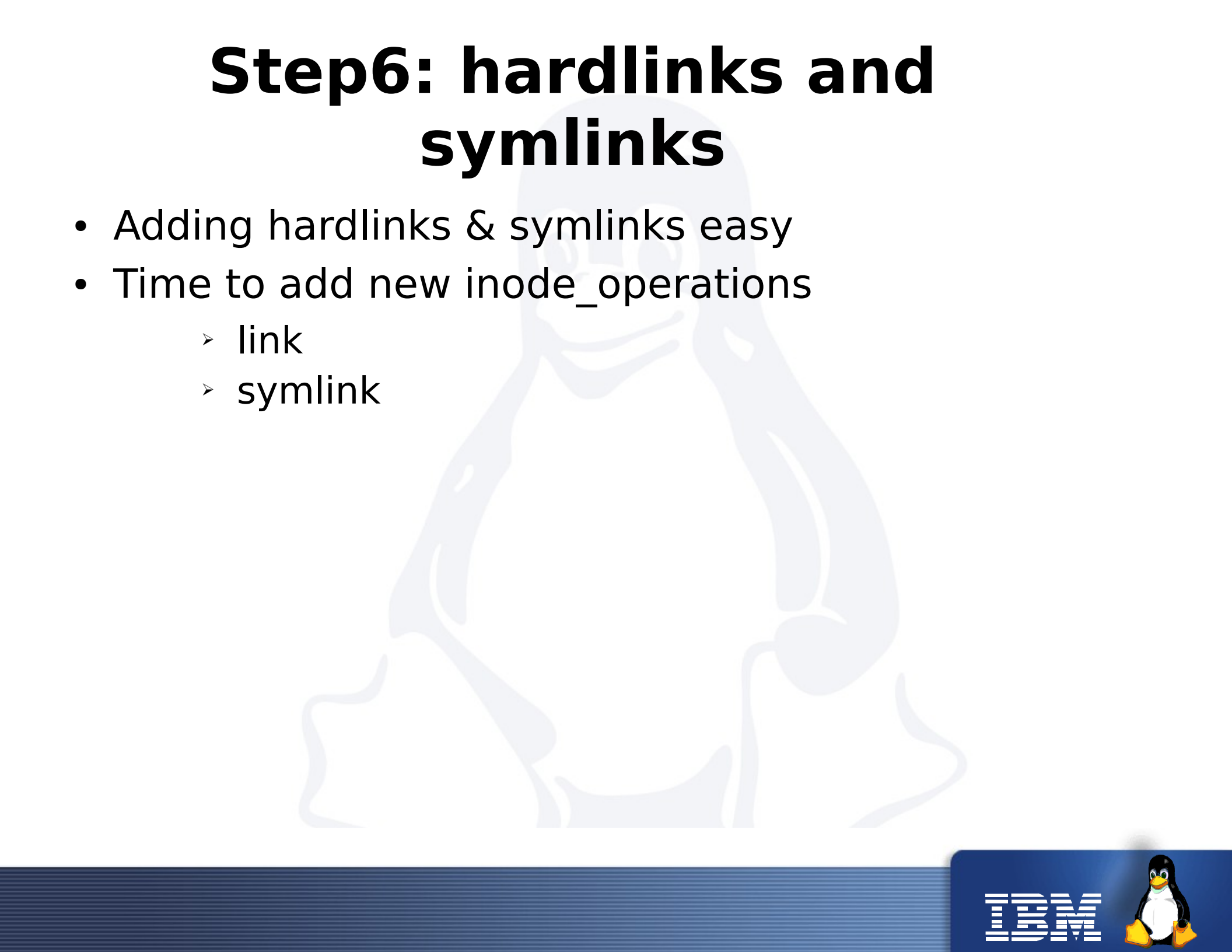
Step5: status

- We can create files, write data and read data and do most common file operations

```
root@:/test# touch t
root@:/test# echo ppp > t
root@:/test# cat t
ppp
root@:/test#
```

Step6: hardlinks and symlinks

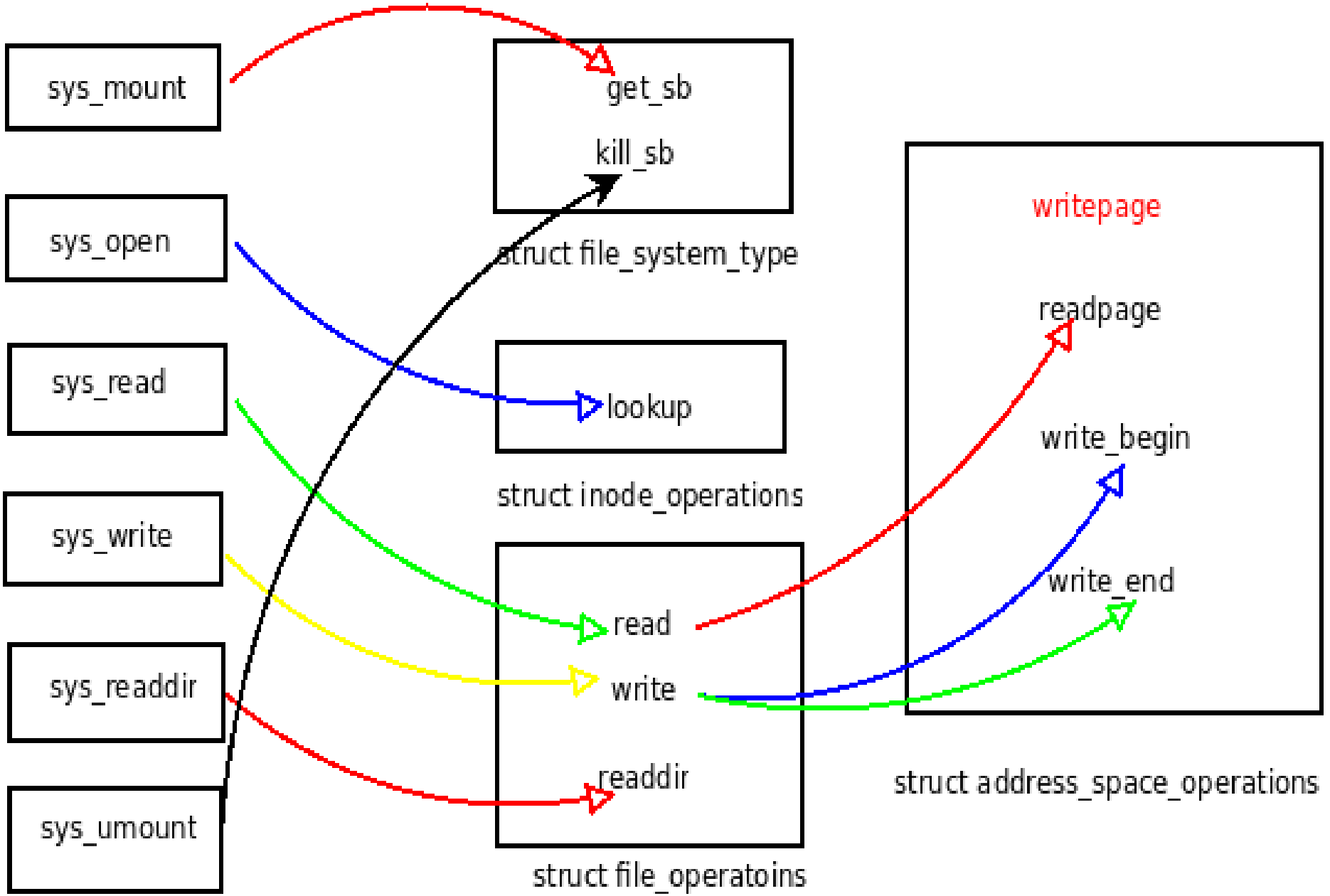
- Adding hardlinks & symlinks easy
- Time to add new inode_operations
  - link
  - symlink



Step6: status

```
root@:/test# ln -s a b
root@:/test# ls -al
total 4
drwxr-xr-x  2 root root    0 May  5 21:53 .
drwxr-xr-x 21 root root 4096 May  4 22:28 ..
lrwxrwxrwx  1 root root    1 May  5 21:53 b -> a
root@:/test# echo dddd > a
root@:/test# cat b
dddd
root@:/test# cat a
dddd
root@:/test#
```

- So lets review how these operations tie together ..


sys_mount
sys_open
sys_read
sys_write
sys_readdir
sys_umount
get_sb
kill_sb
struct file_system_type
lookup
struct inode_operations
read
write
readdir
struct file_operatoins
writepage
readpage
write_begin
write_end
struct address_space_operations

Acknowledgements

- The slides are based on the presentation done by Steve French at OLS 2006
  - http://*us3.samba.org/samba/ftp/cifs-cvs/ols2006-fs-tutorial-smf.pdf*

Legal Statement

- This work represents the view of the authors and does not necessarily represent the view of IBM.
- IBM is a registered trademark of International Business Machines Corporation in the United States and/or other countries.
- Linux is a registered trademark of Linus Torvalds.
- Other company, product, and service names may be trademarks or service marks of others.