Versioning Extensions for Linux

CS736 Spring 1999

J. Adam Butts

Paramjit Oberoi

Linux Versioning Layer (LVL)

- Transparently create copies of files prior to modification
  - Copies are sequentially numbered old versions
  - Allow user access to these versions
- Why use versioning?
  - Recover deleted files
  - Revert to older versions
  - Record modification history
- Versioning in other OS's
  - TOPS (later T[W]ENEX)
  - VMS (DEC VAX OS)
  - Cedar (Xerox PARC)

```
$ ls -al
-r--r--r-- 1167 Apr 9 3:31 .file,2
-r--r--r-- 1459 May 3 0:54 .file,3
-rw-r--r-- 1556 May 5 9:17 file

$ cat >> file
Adding one more line.^D
$ ls -al
-r--r--r-- 1459 May 3 0:54 .file,3
-r--r--r-- 1556 May 5 9:17 .file,4
-rw-r--r-- 1577 May 9 4:30 file

$ rm file
$ ls -al
-r--r--r-- 1556 May 5 9:17 .file,4
-r--r--r-- 1577 May 9 4:30 .file,5
```

Goals and Resulting Challenges

- LVL Goals
  - No OS modification required (runs “out of the box”)
  - Compatibility (forward and backward)
  - Useful for real users
    - Easy to use
    - Customizable
  - Low memory and processing overhead
- Design and Implementation Challenges
  - Naming of old versions
  - Versioning policies
  - Extending the functionality of the Linux kernel

Naming Version Files

- What are desirable requirements for version filenames?
  - Name derived from original name of file
  - Explicit identification of version number
  - Visible to user *on demand*
- What do version numbers mean?
  - Absolute version numbers
    - Gaps can occur if users remove old versions
  - Relative version numbers
    - Filename to file mapping changes over time
  - Both?
- Other Issues
  - Will current file *also* be represented as a numbered version file?
  - How are version numbers affected if old versions are modified?
    - Nested versioning, become latest version, modification not allowed?
  - Links

Version Files in LVL

- Format: ***.filename,number***
  - Examples: **`.736report.fm,4`** and **`.thesis,591`**
  - Original filename and version number explicit
  - Dot is UNIX convention for hidden files
  - Version files created in same directory as file
- Absolute version numbering scheme
  - Users expect a fixed filename to file mapping
  - Version number is one more than highest existing on same file
  - Relative version numbering to identify files may be added
- Most current version identified only with original filename
- Modification of old versions allowed, but not handled differently
  - Modification of **`.thesis,591`** causes **`..thesis,591,0`** to be created

Versioning Policies

- Versioning policies allow users to customize behavior of LVL
- At what granularity may policies be specified?
  - Per: file, directory, arbitrary group of files, user, file system, system...
- What properties of versioning are controlled by policies?
  - What files will be versioned?
  - When will files be versioned?
    - On write?
    - Specific time/date? Time interval?
  - How are version files stored?
    - Exact copy?
    - **`diff`** with previous version? Next version? Latest version?
    - Compressed?
  - How is disk usage limited?
    - Number of versions per: file, directory, arbitrary group of files, user, device…
    - Disk space occupied

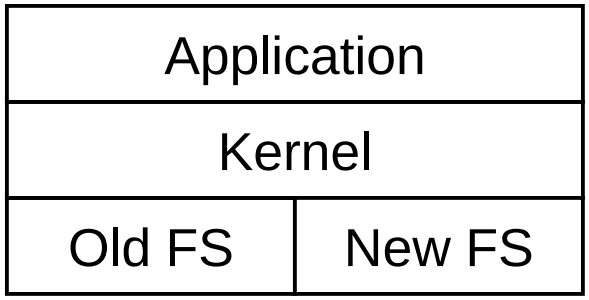
LVL Policies

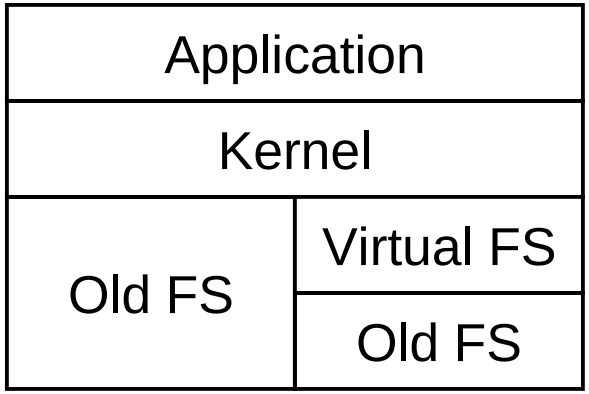
- Policies specified per directory with special file **`.version`**
  - No versioning if policy file not present
- New versions created on first modification after each open
- Version files are exact copy
- Versions limited by number per file

Policy	Explanation	Example
OnWrite	**Create new versions on modification**	**Yes/No**
OnUnlink	**Create new versions on deletion**	**Yes/No**
MaxVersions	**The maximum number of old versions**	**1, 2, 10, ...**
RdOnly	**Old versions are made read-only**	**Yes/No**
Exclude	**Files to be excluded from versioning**	**`*swp|*.o|.*`**
Include	**Files to be versioned even if they match *Exclude***	**`*thesis*`**

Versioning as a File System

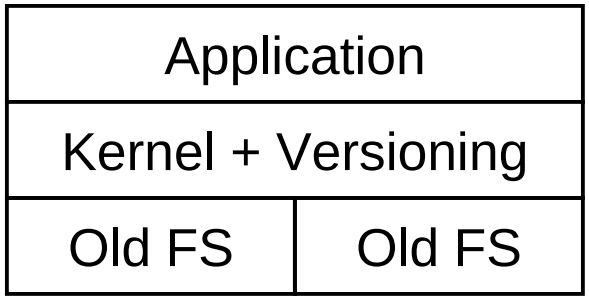
- New versioning file system
  - ✔ Dynamically installable
  - ✔ Complete control over files
  - ✔ Very efficient
  - ✘ Complex to implement
  - ✘ Incompatible with existing file systems
- Virtual file system
  - ✔ Dynamically installable
  - ✔ Utilizes existing file systems
  - ✘ Less efficient
  - ✘ Difficult to ensure transparency
    - Shared data structures between FS & kernel
  - ✘ Per file system functionality



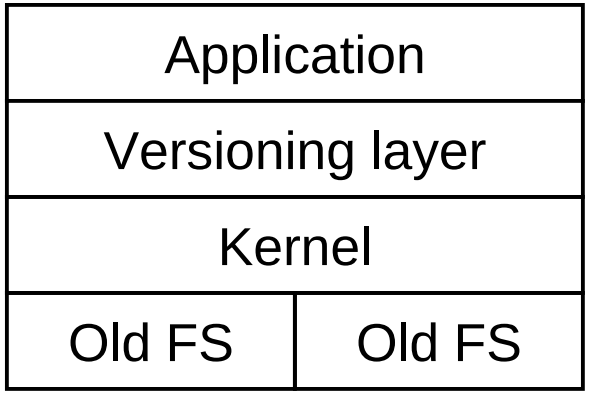


Versioning within the Kernel

- Modified kernel
  - ✔ Versioning on all file systems
  - ✔ Maximum flexibility
  - ✔ Low overhead
  - ✘ Requires rebuild of OS

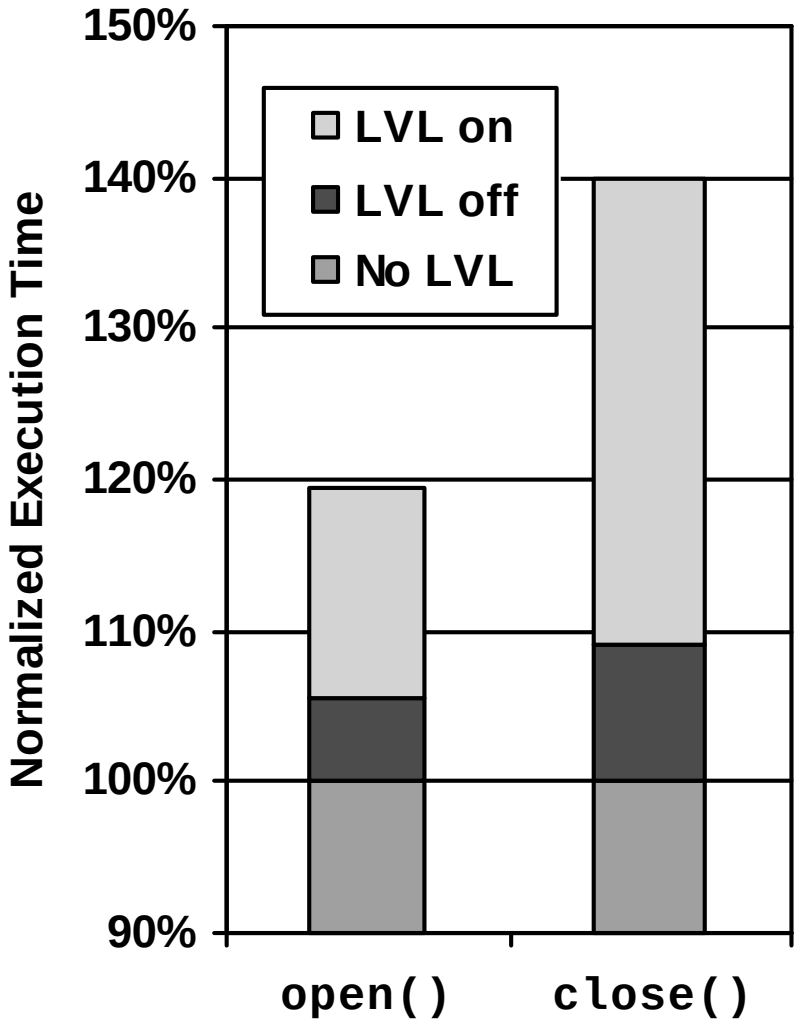


- Modified system calls
  - ✔ Dynamically installable
  - ✔ Applies to all file systems
  - ✔ File level semantics
  - ✘ Reduced flexibility
  - ✘ Increased system call execution time



Performance

- LVL adds minimal overhead
  - ~5% on **`open()`**
  - ~10% on **`close()`**
  - Overhead increases by ~4x when file is versioned
    - Still small absolute times
- **`write()`** overhead variable
  - ~20% if will not be or already versioned
  - Otherwise size dependent because of copy to old version
    - ~10x for files < 10k
    - Then linear increase with file size



Summary

- No modification of kernel required to add versioning
  - Kernel module dynamically adds and removes functionality
- Versioning features are file system independent
  - System call semantics are independent of underlying file system
  - Assumes no limitations on filename format (i.e. no MS-DOS!)
- Many policies and naming options are possible
  - Only a small number of policies are required for usability
- Overhead of versioning within acceptable limits
  - Large files are expensive to copy, but users probably will not require versioning of the largest files
    - audio, video, executable files