The concept of translucency

overlaying made easy (work in progress)

Bernhard M. Wiedemann, and others

August 25, 2002

1 In the Beginning

...the earth was void and without form... err... linux had no VFS... but as Linus saw that it was needed, he created one... and everybody cheered. Files could be accessed and devices, named pipes, sockets and kernel variables too.

But as more applications came to peoples minds, they wanted to do sophisticated operations as to have files that have different content depending on who looks at it (i.e. clusterNFS does so for NFS)

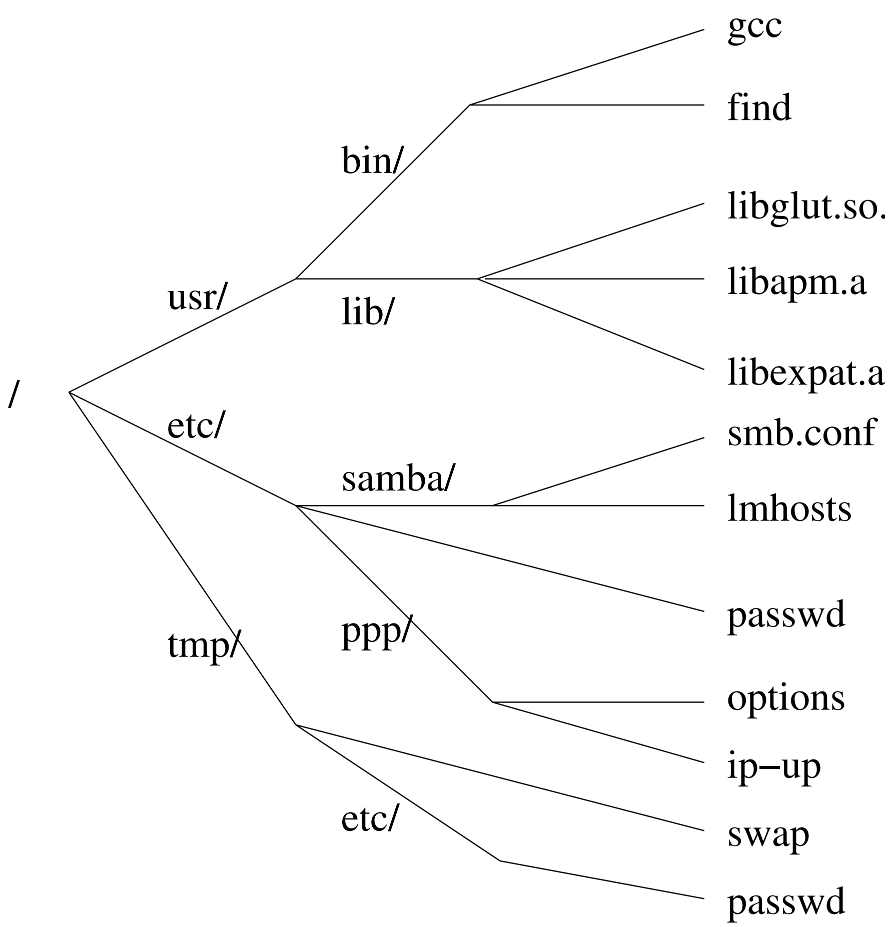
Or to do what everyone easily does with transparencies: put several of them in a stack and write on the topmost of them while everything below remained readable as long as it was not overwritten above. You can even take a lower one out and change it directly, because logically the data belonged there.

People thought of a modular package-management for disk-less nodes and distributed computing environments, writing on CD-ROM, and compiling large projects from clean sources, putting output files somewhere else.

Of course many tried (see `http://translucency.sf.net/dl/implementations/` and `http://www.informatik.hu-berlin.de/~ wiedeman/development/links.html`) and some succeeded. But still current linux did not seem to have this...

2 Paths among trees

As everyone knows there is such a thing called directory tree. Here you can see some of the 100 to 40000 files:



Now, what if we (the kernel) told everyone who wanted to take a path into /etc just to go into /tmp/etc instead. He would no more be able to see or use files in /etc but have those in /tmp/etc there (`mount -bind /tmp/etc /etc` does that)

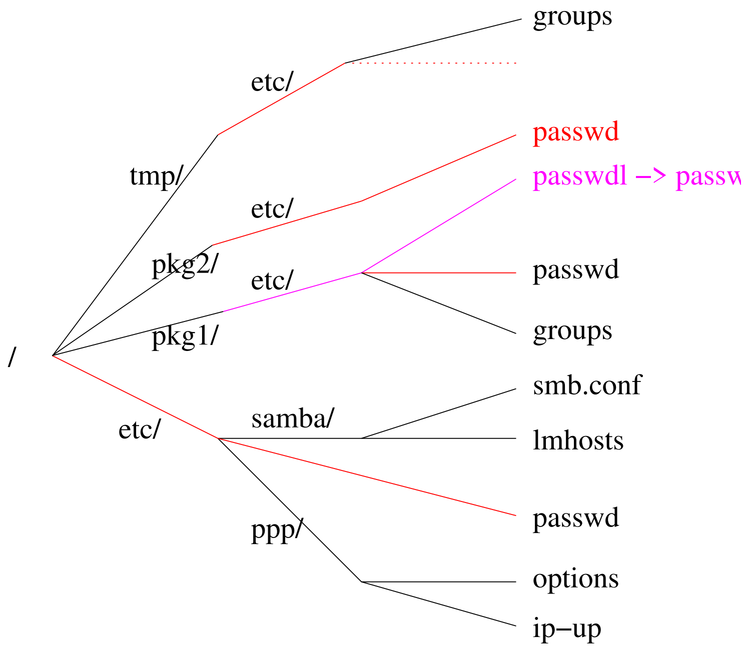
But if we do not want to copy all files and sub-directories to /tmp/etc, because we do not have much free space? Current translucency lkm does just that for you at need: it takes two dirs and if an entry can not be found in the upper one it lets the lower one shine through... or returns -ENOENT if there was none either. If a write is going on it writes to the upper layer. The user says "Hey kernel, write to /etc/passwd" we tell the kernel "a user wants to write to /tmp/etc/passwd"

If we are thinking of doing package-management by translucency this could mean 1000 packages and 1000 times asking "is that file here? not? Oh, well, I will try somewhere else..." We don't care yet that the time needed to open a file is $O(n)$ - we want that functionality really much.

Still there has to be coding to

1. (mostly done: have more than two layers (transparencies) in a stack)
2. have several such stacks
3. let it all work with those translucent directories randomly distributed in VFS (a being within b or b in a with c a sub-directory of b being overlaid by d)

additional complexity gets into it if we have such:



lowest layer is /etc above is /pkg1/etc then /pkg2/etc and /tmp/etc at the top

`open("/etc/passwdl")` opens a symlink. This means wandering through those four directories twice and if it was symlinked to `/etc/passwdl` it could produce an infinite loop in the kernel which means that it can no more do anything useful (read: crash). To avoid this, one has to increment a counter on each step and return -ELOOP if it passes some limit.

3 getdents

`getdents` is a syscall. It was introduced some years ago as a replacement for the linux-specific `readdir` syscall that could only read one directory entry at a time. If you do `ls` it calls getdents to get directory entries and usally sorts them and gets more information by means of the `stat` syscall.

The problem with translucency and getdents is that redirection is done within the `open` syscall, but it does not remember what was redirected where. When a program calls getdents later, there is no status information. Current implementation just checks if that dir to be listed is a subdir of the lower layer. If so, it does another redirection which returns the upper layer. This produces double-entries for overlaid files and does not easily work for layers>2 but it should be overhauled anyway.

One solution could be to merge, sort, uniquify and store directory listings of all layers when an overlaid directory is opened. Later getdents on that handle would just return this prepared listing. It is a relatively clean approach. This would require some amount (5-200K) of memory (think of `ls /dev`), some time for sorting, and reliable bookkeeping logic.

For storing I had the idea of a temporary file. It would be created and unlinked the next moment. This means that though there was no more a directory entry we can `write` to that file-handle, `seek` to the beginning again, and `read` from it serving getdents but as soon as the file gets closed, it is deleted and the allocated space freed. But when is it closed? When getdents reaches its EOF we can close it - this may not happen sometimes. When the owning(opening) process exits. That is fine as long as it is `ls` - but when non-forking demon processes do getdents years without stopping it could fill up your RAM or harddisk. So at some point we may want to hook into `close` syscalls and check for each closed dir if we still keep a temporary file. If so close it too.

Still there has to be some way to find associated data(filehandle) for a given open dir (struct file *) - probably a hash-table.

Another (uncleaner) way would be to create that merged temporary file as above but return this handle... mhh... getdents would result in -ENOTDIR then. Isn't there a "temporary directory" thing?

3.1 merging

merging multiple directory listings to a single one works thus:

1. start at lowest layer with empty result list
2. do getdents on current layer
3. sort those new entries lexicographically
4. walk new list and result list entry by entry:
   (a) if new > result next result
   (b) if new == result if is-whiteout-entry(new) delete else replace
   (c) if new < result insert-before-current(result)
   (d) if end-of(result) append-remaining-entries(new)
5. if not-at-top goto step 2 with layer above current

4 Alternatives

Some people mentioned that the cleaner way would be to implement a **file-system** which directs writes to the topmost layer and reads to all layers beginning it's search at the top. Symlinks (relative and absolute) would just point somewhere below the translucent mount-point and mean the same as a filename above.

Files beeing accessed on the fs need not produce -EBUSY on unmount - it just stops overlaying (is this possible?).

The filesystem does not need a dev.

since several layers (directories) need to be specified at mount-time one could use such (`binfmt_misc` like) syntax: `mount -o layers=:/dir:/other:/more nodev /mnt` or something like `echo :/dir:/other:/more > /proc/sys/fs/translucency/0` to add them later dynamically

Block-level translucency needs a writable file-system and a lower layer which does not change in the life-time of the upper one. Also it has size limited to the lower layer's size and does not easily allow overlaying directories or overlaying more than two layers.

On the other hand it allows to append to large files or change some bytes of it while only allocating a few blocks. Additionally it is very clean and consistent with symlinks, directory-listings, appending, executing, case-insensitivity and all the like.

The cloop module (`http://www.knopper.net/download/knoppix/`) could be useful to have a compressed 2GB read-only ext2 image with empty space in it without wasting bandwidth or disk-space.

e2compr may help too.