New Advances in the Filesystem Space

Grant Miner

Abstract

The filesystem is a database, but it has always been unsuitable as the computer's primary one. Programmers have to write specialized programs to get the functionality they need. Now, new advances in software like Plan 9, the Reiser 4 filesystem and Linux are making the improvements the filesystem needs to become viable. Plan 9 is using the filesystem as the integral system interface, and the Reiser filesystem is unifying pointlessly different but equivalent namespaces. For operating systems to improve for users (that always includes programmers), they need to incorporate these new ideas.

History

The POSIX filesystem is unsuitable for anything but storage of large files. Since it does not support transactions, efficient storage of small files (smaller than a kilobyte), inheritance, files as directories, and plugins, the filesystem is not capable enough to serve for anything other than simple, unreliable file storage and retrieval. It is not suitable for databases, which need transactions, huge data fields, and tiny data fields (dates, integers, characters, etc.).

Unifying The Namespaces

Attributes Schmattributes

In POSIX, attributes are attached to files to describe things such as time of creation, time of last access, time of last modification, and permissions. However, as more people want more attributes (such as support for access control lists), these must be added on in the kernel. In addition, new commands must be implemented to manipulate each new attribute. In POSIX we have `touch` to modify time stamps, `chown` to change a file's owner, `chgrp` to change its group owner, and `chmod` to change its permissions. As more attributes were added, we had to add `chacl` to set access control lists, and `attr` to set other attributes (like immutable and append-only). However, we already have everything we need to implement an infinity of attributes: files.

We can implement attributes as files. The Reiser 4 filesystem supports files-as-directories. This means when you access a file, say, `/foo`, it functions normally, and when you access it as `/foo/`, it acts as a directory. Inside this directory, you can put whatever you want. Reiser 4 gives you some default pseudo files—attributes--as well (2003b): `/foo/..uid` and `/foo/..gid` contain the numerical id of the host owner and group, respectively; `/foo/..rwx` returns the mode of foo; `..nlink` returns the number of hard links, etc. With these files, you can change the ownership of a file simply with

```
cp a/..uid b/..uid
```

eliminating the need for `chown`, `chgrp`, and `chmod`[1]. This means that we have reduced the number of primitives in our system, which increases its flexibility:

1 Simpler variants of these commands would still be desired, to resolve group and user names to ID's, and to handle symbolic permissions. More importantly, these commands would be simple enough to write as simple shell scripts.

instead of our primitives being the set of files, directories and attributes, it is now just the set of files and directories (2003c).

Storage

The filesystem has to be able to efficiently store these small files. Most filesystems allocate files in large blocks. A file like ..uid is 4 bytes. If it were stored in a standard filesystem, it would occupy a whole block—it could be ½, 1, 2, 3, or 4 kilobytes. That wastes a lot of space with many files. Reiser4 uses a fast B+tree algorithm to store small files efficiently with high performance (2003c).

The GConf Problem

GConf[2] is a configuration program, part of the GNOME platform, written to easily store user preference. This is how the authors describe GConf (2003a):

> Essentially GConf provides a preferences database, which is like a simple filesystem. The filesystem contains keys organized into a hierarchy. Each key is either a directory containing more keys, or has a value. For example, the key /apps/metacity/general/titlebar_font contains an integer value giving the size of the titlebar font for the Metacity window manager.

When you use GConf, it keeps a directory in your home directory specified in `/etc/gconf/<version>/path` (the default is `.gconf`). Your `.gconf` acts as the root, `/`, of your GConf database. Thus, `/apps` (in GConf) is kept in the directory `/home/you/.gconf/apps` on the filesystem. For efficiency reasons, in each directory, an XML file[3] stores the actual settings. Furthermore,

> System wide, GConf's path file comes configured as follows:

```
xml:readonly:/etc/gconf/gconf.xml.mandatory
include "$(HOME)/.gconf.path"
xml:readwrite:$(HOME)/.gconf
xml:readonly:/etc/gconf/gconf.xml.defaults
```

> If a value is set in the first source, which is read only, then users can't delete that value, and thus can't set a value of their own. These settings become mandatory for all users. The "include" line allows users to insert their own configuration sources into the search path by creating a file in their home directory called .gconf.path. The readwrite source ~/.gconf is where user settings are normally written....
>
> In addition to $(HOME), the variable $(USER) can be used, and any variable $(ENV_FOO) will be replaced by the value of the environment variable FOO. So $(ENV_DISPLAY) might be handy, for example.

There is a way to make the filesystem powerful enough so that it can perform the features we need GConf for. The first move is obvious: once we have efficient small files, we do not need the XML files. We can just store `/apps/metacity/general/titlebar_uses_system_font` in the file named `/home/you/.gconf/apps/metacity/general/titlebar_uses_system_font`.

But we still need to know where the user's home directory is (so we can access his `.gconf` directory). How can eliminate that lookup? Traditionally, POSIX uses the HOME environment variable. Interestingly, in Plan 9, environment variables are a filesystem mounted on `/mnt/env`, and this filesystem allows only files and not creation of directories. To get $HOME on Plan 9, one would simply cat `/mnt/env/HOME`. But, the environment is not even needed, as will be shown in the next paragraph.

Next, we need to do what Plan 9 does: each process gets its own filesystem namespace, and mounts can be layered (stacked). This means that my filesystem root (`/`) can be different from someone else's root on the same machine.

---

2 GConf is a particularly excellent example, because it illustrates all of the shortcomings of the filesystem and how programmers have to expend too much effort to get around these deficiencies.

3 According to the GConf documentation, the backend is pluggable, so that flat files or a relational database could also be used, but these have yet to be implemented.

Furthermore, we can bind `/home/me` to my root as read-write. So if we do that, I can write files in `/` (and they are really stored in `/home/me`). Finally, the system root can be layered on (optionally read-only) to my personal root.

Assume there is a per-directory file, named ..mount, that is responsible for the mount layer management (the OS watches this file for changes and acts accordingly). Each line specifies the mount source and mount options (such as read only, filesystem type, etc) much like `/etc/fstab`. The higher up a line is, the higher priority it is, i.e. when a call to access a file is made, the OS searches from the top-most mounted filesystem first, then the next, until the file is found or the list is exhausted. So lets see what my `/..mounts` file looks like:

```
/dev/hda1 read-only
/home/me read-write
```

Assume `/dev/hda1` contains the system-wide root directory, with the directories `/gconf` and `/bin`, and no files in `/`. In this configuration, I can now access all the system-wide programs, which will be in my `/bin` because `/dev/hda1` is mounted on my root. If I feel like writing a file */gconf/screen_size*, I cannot write it, because `/dev/hda1` is read-only. But, if my `/..mounts` file were

```
/home/me read-write
/dev/hda1 read-only
```

then I could write the file `/gconf/screen_size` (assuming I had permission in my home directory), because a read-write filesystem is first on the stack. Also, if `/etc/resolution` existed only on `/dev/hda1`, and I edited and then saved it, that file would transparently be copied (with the modifications) to my `/home/me`.

Incidentally, with this setup, environment variables are unnecessary. It is easier just to create a file.

Metacity now knows that `/gconf/apps/metacity/general/titlebar_uses_system_font`[4] contains the option. to use the system font for its titlebar. I have eliminated the environment variable lookup for $HOME, the gconf lookup for where `.gconf` is, and gconf's lookup to `/etc/gconf/`<version>`/path` to find a setting in a list of directories. But I am not quite done yet. Programs need to be able to get notified of changes to a file, so that when you change the `titlebar_uses_system_font` option, metacity instantly adjusts itself accordingly. FAM handles this on Linux. Here, if programs used files for their environment variables and watched those files for changes, nobody would have to kill all her programs after changing her $PATH. Finally, the filesystem should report back an error if an invalid option is put in `/gconf/apps/metacity/general/titlebar_uses_system_font`.

Reiser4 handles this problem with filesystem plugins, specifically a boolean plugin. Having /`gconf/apps/metacity/general/titlebar_uses_system_font` use the boolean plugin enforces a constraint: that file can only have the value `true` or `false`, and this value would be compressed down to a single bit on the filesystem. Reiser4 will have numerous plugins for constraints as well as security (2003c).

Now, as a system administrator, you ensure that `/etc/system-wide-gconf` is in your users' `/gconf/..mounts`. If you want to lock those settings down, make sure the line specifying the locked-down settings is above the one specifying their home directory (so that when they try to write a new setting, the OS finds the system directory first, which would deny them write access). If you just want to give them some default settings, have that line below their home directory's line. Voila! Lock down framework.

This is nice progress. All the functionality GConf needs the filesystem provides, which eliminates the need to install the GConf program and the required XML libraries. Better yet, it makes application preferences more manageable for

4 This directory, as well as /bin, /lib, and /share should probably go in /system or some other directory so that users don't have lots of system directories cluttering their root directory, where they'll put their files.

system administrators[5], because they can use all their real filesystem tools (editors, find, cat, cp) instead of looking up the syntax for gconftool each time they need to do something.

Application to Common Usage

Sysfs

The Linux kernel, version 2.5, has a new filesystem, sysfs, which exports kernel objects as files. Most drivers have converted to use sysfs for their configuration. These files are text-based and allow easy configuration of system parameters like the SCSI subsystem and the elevator.

Tagging your MP3s

Once we see the GConf example, other possibilities immediately spring to mind. In almost all multimedia formats--such as MP3, MPEG, and OGG--there is a tagging system for storing things such as the author and title. Instead of storing those attributes in the tag--yet another namespace--store it in the file-as-a-directory. I have a file `/music/Millencolin/PennybridgePioneers/RightAboutNow.ogg`. I let `RightAboutNow.ogg/title` be "Right About Now", `/artist` be "Millencolin". I could add a file `RightAboutNow.ogg/genre` and put "punk" in it. Or have it be a list of genres:

```
Swedish
punk
pop
```

Furthermore, I could strip the .ogg extension, so I get

```
#cat
"/music/Millencolin/PennybridgePioneers/RightAboutNow/type"
ogg
```

Putting metadata like this in the filesystem is much more flexible, as there are no arbitrary size and character restrictions. Most importantly, it eliminates the need for the supporting libraries and programs for manipulating the tag formats for the hundreds of multimedia file formats.

/etc/passwd

The `passwd` file can be decomposed into a file hierarchy, and recomposed with a filesystem plugin (2003c). In `/etc/passwd/` go numbered files, corresponding to the uid of each user. So we can have a session like this:

```
#cat /etc/passwd/0/name
root
#cat /etc/passwd/0/shell
/bin/bash
#echo "/bin/scsh" > /etc/passwd/0/shell
```

Now users get to change their shells and credentials based on per-file ACL's, not with insecure setuid root programs. The filesystem plugin puts everything back together with delimiters so that editing /etc/passwd remains the same.

With this hierarchy, it is easy to have sub-users. For example, let us say Joe has an account is in `/etc/passwd/500`. Lets say he wants to run an untrusted program. He creates a new sub-user (of himself) in `/etc/passwd/500/5`. After, and only after Joe logs in, he can login in as a sub-user, i.e. user #5 (optionally with

5 This is also the main goal of GConf, whose "root motivation ... is to make application preferences more manageable for system administrators" (2003a).

a password, but it would not be necessary, since he would have to supply a password to get into the parent (Joe's) account first). He is allowed to make arbitrary nested users for himself without having to ask the system administrator. Now users can make a separate sub-user for each program, therefore protecting their files from malicious or buggy programs.

LDAP

You do not need it. Layer mount your passwd hierarchy over the network transport of your choice. Bonus: if the server goes down, and you have a local filesystem layer, you might still be able to login.

E-Mail

Plan 9 (2000) has a program that collects e-mail and makes it accessible through the filesystem, in `/mail/box`. This makes the mail collection program separate from the mail viewer and composer programs. Also, the filesystem is particularly well suited to the MIME mail attachment hierarchy. You can read your e-mail from your favorite file manager, text editor, and HTML viewer.

Environment Variables

Have your programs use files instead of environment variables, and monitor those files for changes. That way users can update settings in those files without having to restart all of their programs.

Setuid

Setuid and setgrp need to go away. If Apache needs port 80, I should be able to say “the apache user gets read and write access to port 80” to the OS. If a process needs a capability, that capability should be expressible as a file so that an access control list can be applied to it. SELinux has been doing a lot of work in this area.

References

Pike, Rob. 2000. The Use of Name Spaces in Plan 9.
2003a. http://www.gnome.org/projects/gconf/
Danilov, Nikita. 2003b. Standard Reiser4 Psuedo Files.
http://www.namesys.com/v4/pseudo.html
2003c. Reiser 4. http://www.namesys.com/v4/v4.html