

## read/only root filesystem strategy

**Source:** <http://linux.derkeiler.com/Newsgroups/comp.os.linux.development.system/2005-03/0477.html>

---

*phil-news-nospam\_at\_ipal.net*

**Date:** 03/19/05

Date: 19 Mar 2005 03:49:03 GMT

This little project I am working on has the goal of making as much of the installed system as possible be running under a read/only mount. There are two aspects to this. One is to be able to run on read/only media such as a CDRom/DVD, or a widely shared network filesystem. The other is to simply provide additional "accident prevention" for systems running from ordinary hard disks. And I want to do it the same both ways.

I believe I should have no problems with the following subdirectories being read/only:

`/bin /boot /lib /mnt /opt /sbin /usr`

Certain problems with exist with these:

`/dev /etc`

So part of the strategies I'm investigating involve setting these up as copies loaded into a ramfs or tmpfs filesystem. Other subdirectories do need to be writable and persistent across reboots:

`/home /tmp /var`

I did not include /etc in the latter group because I believe it can be run in a non-persistent way. Any direct changes to it get lost at reboot, unless those changes get copied into wherever the persistent copy is kept (undecided).

I'm looking at several strategies do set this up. But they all involve ending up with / being a non-device filesystem, such as rootfs or tmpfs. The subdirectories would be either bind mounted from some partition, or ramfs/tmpfs. Among the procedures to set this up I am exploring:

1. Modifying the kernel to perform the setup.
2. Using early userspace /init to perform the setup.
3. Substituting /sbin/init to perform the setup.

## comp.os.linux.development.system: read/only root filesystem strategy

Methods 1 and 2 may end up leaving the initial rootfs filesystem the kernel starts with, before mounting the requested root filesystem, exposed at the / mount point. I would probably remount it read/only. Once the setup is (mostly) complete, each method would invoke a normal init program in PID 1 to complete the startup of the system and run it.

Probably the biggest area of difficulty is deciding the best way to populate the /dev /etc subdirectories when these are rootfs or tmpfs. It would likely be most cumbersome to do this in the kernel itself, so method number 1 looks unattractive (though it seems to work well for the rest of the chores in early prototypes in init/main.c). Method number 2 shares another problem with number 1 in that both require a special kernel image. Method 3 will be a bit more twisted because it has to do a pivot\_root() and probably an invocation of a 2nd executable file to release the root filesystem so it can be remounted explicit (since it isn't technically a root filesystem anymore, I want to get rid of the /dev/root mount source info).

Here's one filesystem layout I'm working on trying to achieve in the above way:

mount point source device directory within device

```
/ tmpfs
/bin hda2 /bin
/boot hda1 /boot
/dev tmpfs
/dev/shm tmpfs
/etc tmpfs
/home hda4 /home
/lib hda2 /lib
/mnt hda2 /mnt
/opt hda2 /opt
/proc proc
/root hda4 /root
/sbin hda2 /sbin
/tmp hda5
/usr hda2 /usr
/usr/local hda4 /usr/local
/var hda6 /var
/var/lock tmpfs
/var/run tmpfs
```

Partition hda1 would house the kernel and a small (240MB) rescue system for administrative purposes. Partition hda2 would be the principle system partition, and where everything is installed. Partition hda3 would have the extended partitions which would include /tmp /var and swap space. Partition hda4 would then have everything from there to the end of the drive (the big space). Partition hda2 would be mounted read/only and stay that way until the administrator has the need to update something. The other

## comp.os.linux.development.system: read/only root filesystem strategy

partitions would be writable except for hda1, which doesn't really even need to be mounted most of the time.

What is hda2 here might be the CDROM for a CDROM bootable system. Then the only way to change the system would be to burn a new CDROM.

I've already asked about how I might get a process disconnected from the mapped file it starts from. It's not necessarily essential to do that, but I would like to know that just to keep options open.

But this posting is more to solicit feedback on ways to do this. I'm sure there are plenty of people who will say "why bother". Maybe you wouldn't do this at all. But I'm convinced that once it is worked out, it could make for a reasonably reliable way to manage a system in careful and practical manner, emphasizing reliability and taking what little bit of added security comes along with it.

--

---

Phil Howard KA9WGN	<a href="http://linuxhomepage.com/">http://linuxhomepage.com/</a>	<a href="http://ham.org/">http://ham.org/</a>
(first name) at ipal.net	<a href="http://phil.ipal.org/">http://phil.ipal.org/</a>	<a href="http://ka9wqn.ham.org/">http://ka9wqn.ham.org/</a>

---