Pid Eins

レナート
لينارت

POSTED ON DI 12 APRIL 2011

systemd for Administrators, Part VII

Here's yet another installment of my ongoing series on systemd for Administrators:

The Blame Game

Fedora 15[1] is the first Fedora release to sport systemd. Our primary goal for F15 was to get everything integrated and working well. One focus for Fedora 16 will be to further polish and speed up what we have in the distribution now. To prepare for this cycle we have implemented a few tools (which are already available in F15), which can help us pinpoint where exactly the biggest problems in our boot-up remain. With this blog story I hope to shed some light on how to figure out what to blame for your slow boot-up, and what to do about it. We want to allow you to put the blame where the blame belongs: on the system component responsible.

The first utility is a very simple one: systemd will automatically write a log message with the time it needed to syslog/kmsg when it finished booting up.

```
systemd[1]: Startup finished in 2s 65ms 924us (kernel) + 2s 828ms 195us (in
```

And here's how you read this: 2s have been spent for kernel initialization, until the time where the initial RAM disk (initrd, i.e. dracut) was started. A bit less than 3s have then been spent in the initrd. Finally, a bit less than 12s have been spent after the actual system init daemon (systemd) has been invoked by the initrd to bring up userspace. Summing this up the time that passed since the boot loader jumped into the kernel code until systemd was finished doing everything it needed to do at boot was a bit less than 17s. This number is nice and simple to understand -- and also easy to misunderstand: it does not include the time that is spent initializing your GNOME session, as that is outside of the scope of the init system. Also, in many cases this is just where systemd finished doing everything it needed

to do. Very likely some daemons are still busy doing whatever *they* need to do to finish startup when this time is elapsed. Hence: while the time logged here is a good indication on the general boot speed, it is not the time the user might *feel* the boot actually takes.

Also, it is a pretty superficial value: it gives no insight which system component systemd was waiting for all the time. To break this up, we introduced the tool `systemd-analyze blame`:

```
$ systemd-analyze blame
  6207ms udev-settle.service
  5228ms cryptsetup@luks\x2d9899b85d\x2df790\x2d4d2a\x2da650\x2d8b7d2fb92cc
   735ms NetworkManager.service
   642ms avahi-daemon.service
   600ms abrtd.service
   517ms rtkit-daemon.service
   478ms fedora-storage-init.service
   396ms dbus.service
   390ms rpcidmapd.service
   346ms systemd-tmpfiles-setup.service
   322ms fedora-sysinit-unhack.service
   316ms cups.service
   310ms console-kit-log-system-start.service
   309ms libvirtd.service
   303ms rpcbind.service
   298ms ksmtuned.service
   288ms lvm2-monitor.service
   281ms rpcgssd.service
   277ms sshd.service
   276ms livesys.service
   267ms iscsid.service
   236ms mdmonitor.service
   234ms nfslock.service
   223ms ksm.service
   218ms mcelog.service
...
```

This tool lists which systemd unit needed how much time to finish initialization at boot, the worst offenders listed first. What we can see here is that on this boot two services required more than 1s of boot time: `udev-settle.service` and `cryptsetup@luks\x2d9899b85d\x2df790\x2d4d2a\x2da650\x2d8b7d2fb92cc3.service`. This tool's output is easily misunderstood as well, it does not shed any light on why the services in question actually need this much time, it just determines that they did. Also note that the times listed here might be spent "in parallel", i.e. two services might be initializing at the same time and thus the time spent to initialize them both is much less than the sum of both individual times combined.

Let's have a closer look at the worst offender on this boot: a service by the name of `udev-settle.service`. So why does it take that much time to initialize, and what can we do about it? This service actually does very little: it just waits for the device probing being done by udev to finish and then exits. Device probing can be slow. In this instance for example, the reason for the device probing to take more than 6s is the 3G modem built into the machine, which when not having an inserted SIM card takes this long to respond to software probe requests. The software probing is part of the logic that makes ModemManager work and enables NetworkManager to offer easy 3G setup. An obvious reflex might now be to blame ModemManager for having such a slow prober. But that's actually ill-directed: hardware probing quite frequently is this slow, and in the case of ModemManager it's a simple fact that the 3G hardware takes this long. It is an essential requirement for a proper hardware probing solution that individual probers can take this much time to finish probing. The actual culprit is something else: the fact that we actually wait for the probing, in other words: that `udev-settle.service` is part of our boot process.

So, why is `udev-settle.service` part of our boot process? Well, it actually

doesn't need to be. It is pulled in by the storage setup logic of Fedora: to be precise, by the LVM, RAID and Multipath setup script. These storage services have not been implemented in the way hardware detection and probing work today: they expect to be initialized at a point in time where "all devices have been probed", so that they can simply iterate through the list of available disks and do their work on it. However, on modern machinery this is not how things actually work: hardware can come and hardware can go all the time, during boot and during runtime. For some technologies it is not even possible to know when the device enumeration is complete (example: USB, or iSCSI), thus waiting for all storage devices to show up and be probed must necessarily include a fixed delay when it is assumed that all devices that can show up have shown up, and got probed. In this case all this shows very negatively in the boot time: the storage scripts force us to delay bootup until all potential devices have shown up and all devices that did got probed -- and all that even though we don't actually need most devices for anything. In particular since this machine actually does not make use of LVM, RAID or Multipath![2]

Knowing what we know now we can go and disable `udev-settle.service` for the next boots: since neither LVM, RAID nor Multipath is used we can mask the services in question and thus speed up our boot a little:

```
# ln -s /dev/null /etc/systemd/system/udev-settle.service
# ln -s /dev/null /etc/systemd/system/fedora-wait-storage.service
# ln -s /dev/null /etc/systemd/system/fedora-storage-init.service
# systemctl daemon-reload
```

After restarting we can measure that the boot is now about 1s faster. Why just 1s? Well, the second worst offender is cryptsetup here: the machine in question has an encrypted `/home` directory. For testing purposes I have stored the passphrase in a

file on disk, so that the boot-up is not delayed because I as the user am a slow typer. The cryptsetup tool unfortunately still takes more han 5s to set up the encrypted partition. Being lazy instead of trying to fix cryptsetup[3] we'll just tape over it here [4]: systemd will normally wait for all file systems not marked with the `noauto` option in /etc/fstab to show up, to be fscked and to be mounted before proceeding bootup and starting the usual system services. In the case of `/home` (unlike for example `/var`) we know that it is needed only very late (i.e. when the user actually logs in). An easy fix is hence to make the mount point available already during boot, but not actually wait until cryptsetup, fsck and mount finished running for it. You ask how we can make a mount point available before actually mounting the file system behind it? Well, systemd possesses magic powers, in form of the `comment=systemd.automount` mount option in `/etc/fstab`. If you specify it, systemd will create an automount point at `/home` and when at the time of the first access to the file system it still isn't backed by a proper file system systemd will wait for the device, fsck and mount it.

And here's the result with this change to `/etc/fstab` made:

```
systemd[1]: Startup finished in 2s 47ms 112us (kernel) + 2s 663ms 942us (in
```

Nice! With a few fixes we took almost 7s off our boot-time. And these two changes are only fixes for the two most superficial problems. With a bit of love and detail work there's a lot of additional room for improvements. In fact, on a different machine, a more than two year old X300 laptop (which even back then wasn't the fastest machine on earth) and a bit of decrufting we have boot times of around 4s (total) now, with a resonably complete GNOME system. And there's still a lot of room in it.

`systemd-analyze blame` is a nice and simple tool for tracking down slow services. However, it suffers by a big problem: it does not visualize how the parallel execution of the services actually diminishes the price one pays for slow starting services. For that we have prepared `systemd-analyize plot` for you. Use it like this:

```
$ systemd-analyze plot > plot.svg
$ eog plot.svg
```

It creates pretty graphs, showing the time services spent to start up in relation to the other services. It currently doesn't visualize explicitly which services wait for which ones, but with a bit of guess work this is easily seen nonetheless.

To see the effect of our two little optimizations here are two graphs generated with `systemd-analyze plot`, the first before and the other after our change:

(For the sake of completeness, here are the two complete outputs of `systemd-analyze blame` for these two boots: before and after.)

The well-informed reader probably wonders how this relates to Michael Meeks' bootchart. This plot and bootchart do show similar graphs, that is true. Bootchart is by far the more powerful tool. It plots in all detail what is happening during the boot, how much CPU and IO is used. `systemd-analyze plot` shows more high-level data: which service took how much time to initialize, and what needed to wait for it. If you use them both together you'll have a wonderful toolset to figure out why your boot is not as fast as it could be.

Now, before you now take these tools and start filing bugs against the worst boot-up time offenders on your system: think twice. These tools give you raw data, don't misread it. As my optimization example above hopefully shows, the blame for the slow bootup was not actually with `udev-settle.service`, and not with the ModemManager prober run by it either. It is with the subsystem that pulled this service in in the first place. And that's where the problem needs to be fixed. So, file the bugs at the right places. Put the blame where the blame belongs.

As mentioned, these three utilities are available on your Fedora 15 system out-of-the-box.

And here's what to take home from this little blog story:

- `systemd-analyze` is a wonderful tool and systemd comes with profiling built in.
- Don't misread the data these tools generate!
- With two simple changes you might be able to speed up your system by 7s!
- Fix your software if it can't handle dynamic hardware properly!

- The Fedora default of installing the OS on an enterprise-level storage managing system might be something to rethink.

And that's all for now. Thank you for your interest.

Footnotes

[1] Also known as the greatest Free Software OS release ever.

[2] The right fix here is to improve the services in question to actively listen to hotplug events via libudev or similar and act on the devices showing up as they show up, so that we can continue with the bootup the instant everything we really need to go on has shown up. To get a quick bootup we should wait for what we actually need to proceed, not for everything. Also note that the storage services are not the only services which do not cope well with modern dynamic hardware, and assume that the device list is static and stays unchanged. For example, in this example the reason the initrd is actually as slow as it is is mostly due to the fact that Plymouth expects to be executed when all video devices have shown up and have been probed. For an unknown reason (at least unknown to me) loading the video kernel modules for my Intel graphics cards takes multiple seconds, and hence the entire boot is delayed unnecessarily. (Here too I'd not put the blame on the probing but on the fact that we wait for it to complete before going on.)

[3] Well, to be precise, I actually did try to get this fixed. Most of the delay of crypsetup stems from the -- in my eyes -- unnecessarily high default values for `--iter-time` in cryptsetup. I tried to convince our cryptsetup maintainers that 100ms as a default here are not really less secure than 1s, but well, I failed.

[4] Of course, it's usually not our style to just tape over problems instead of fixing them, but this is such a nice occasion to show off yet another cool systemd feature...

Category: projects

← BACK TO INDEX