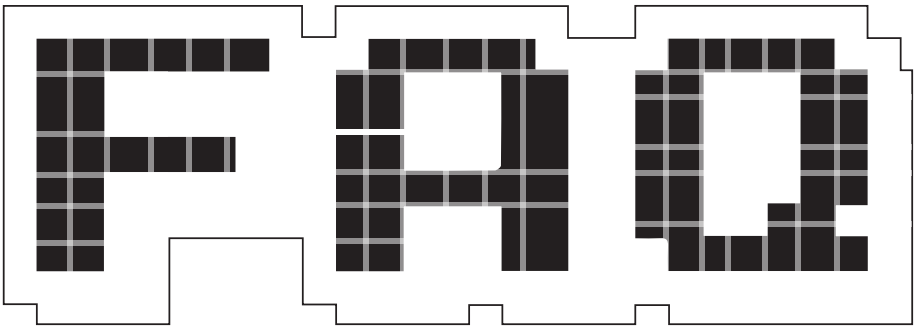

SYSTEMD

It's complicated but it's probably already booting your computer.

GRAHAM MORRISON

Q Surely the 'd' in Systemd is a typo?

A No –it's a form of Unix notation used to signify a daemon.

Q You mean like those little devils inhabiting Dante's underworld?

A There is a link in that Unix usage of the term daemon supposedly comes from Greek mythology, where daemons invisibly wove their magic and benign influence. The word is today more commonly spelt 'demon', which leaves the archaic form to us. And that's perhaps why it's been subverted for software that silently weaves its benign influence over your system.

Q Does this mean that, for ordinary users, Unix daemons are an archaic idea?

A Not at all. Daemons are essential for performing many essential background tasks. They could be likened to your brain doing many things while you're driving a car, without you having to consciously think about them all. And there are lots of daemons you may already be aware of. *Crond*, for example, is used to run scripts at a specific time or interval. *dhcpd* is used by most of our computers to dynamically connect to your network, while *syslogd* pools all the system messages together to create a log of everything important. Another daemon, though it lacks the 'd', is *init* – famous for being the first process that runs on your system.

Q Isn't *init* used to switch between the command-line and the graphical desktop?

A For many of us, yes. This was the main way of going from the desktop to a command line and back again without trying to figure out which processes to kill or start manually. Typing **init 3** would typically close any graphical environment you had running, kill the X server and drop you to a simple login prompt. This is because the **init** command knew where to find a group of scripts that were created for switching between different operating environments. These were called 'runlevels' and in the old days of System-V Unix they could be used to tell your system to boot into single-user mode, multi-user mode as well as shutdown and reboot. This whole system is often abbreviated to 'SysV init', and it's this we mean when we refer to 'init'.

Many Linux distributions inherited the same system, even if they didn't use the same runlevels. Debian, for example, used a runlevel of 0 to stop the system, a runlevel of 1 for a single-user mode, runlevel 3 for the same command prompt we described earlier, and runlevel 5 to launch a graphical environment. Changing this for your next boot often involved editing the **/etc/inittab** file, and you'd soon get used to manually starting and stopping your own services simply by executing the scripts you found.

Q You seem to be using the past tense for all this talk about the init daemon...

A That's because the aforementioned *Systemd* wants to put *init* in the past. *Systemd* is basically a replacement for all those scripts launched by the *init* process, and a replacement for *init* itself. But as the name suggests, it's got lofty ambitions for being the 'system daemon', and has the potential to handle far more than starting and stopping your system. And in replacing a few scripts, it's needed to come up with a completely different mechanism for handling boot processes.

Q What was so bad with 'init' that it needed replacing?

A *Init* was a product of its time. Everything that needed to be managed or launched did so from its own set of scripts, keeping things relatively simple in concept. Those

scripts called upon whatever utilities were needed and installed within your system, and it also meant that almost anyone could tweak the scripts to get their boot process doing exactly what they wanted. It was an extension of the old Unix philosophy of small, simple tools being used in preference to complex, over-engineered solutions. The command line is a perfect example of this philosophy, because while you can build very complex solutions within its confines, most of the commands you execute perform only a simple task. It's the modularity of these commands, and the control you have over managing their input and output, that creates the flexibility we all love. *init* does this too.

> **"Systemd launches processes in parallel and avoids the bottlenecks that blight init."**

But *init* has suffered in several ways. First, because it's so easy to modify and has no formal system of standardisation, almost every family of Linux distribution tweaked it in some way. Red Hat moved files into **/etc/rc.d/init.d**, for example, and that meant that while installing the same start process should be straightforward, it wasn't. You had to take into account all these changes. And as booting requirements became ever more complicated, maintaining all these scripts and their positions within the boot procedure became very difficult, making things harder for developers, users and system administrators when they used more than one distribution.

Q That sounds more like an organisational problem than a technical one. Surely there's another reason to replace *init*?

A Indeed there is, and it's our old friend performance. The world was becoming obsessed with boot times – despite most of us only having to go through the procedure maybe once a day, or that for servers it's a non-issue. But for a while boot times became the machismo statistic of an operating system's vitality. *init* is fundamentally sequential. That means it waits for one task to finish before it attempts another, and it doesn't take much to imagine the delays this could cause in the boot process. Your boot procedure could spend ages waiting for a network connection, even if you didn't need one, or waiting for an unrelated hard disk to be scanned. These are both functions better suited to being run in parallel, but this introduces problems familiar to anyone who's had to deal with concurrency or forking processes; you need to know which processes are dependent on which other processes. A Samba process that needed to access remote files would be dependent on the networking process, for example, and calculating the relationship between all the things that make a running system is difficult.

But as new computers have become faster and multi-cored, more and more of these resources were being under-utilised by *init*, and many people thought it was time for an alternative.

Q Aha! The appearance of *Systemd*!

A Exactly. Although *Systemd* wasn't the only contender in the fight to replace *init*. Ubuntu had switched to its own alternative, called *Upstart*, and for a while it looked like many other distributions would do the same. Red Hat, Fedora and OpenSUSE all used *Upstart* at various times in the past, and it had the huge advantage of being backward compatible with *init* while addressing some of its shortcomings.

Q You seem to be using the past tense again.

A The end of *Upstart* came with a very public and contentious debate, as Debian decided how to replace *init*. *Upstart* and *Systemd* were the two favourites, and the Debian Technical Committee eventually voted for *Systemd*. In a humble blog post, Mark Shuttleworth announced that Ubuntu would be dropping *Upstart* in favour of *Systemd* after committing itself to whatever its upstream partner (Debian)decided. These events have led us to the point where *Systemd* is now considered the default booting daemon.

Q So what makes *Systemd* better than *init* or *Upstart*?

A During the great Debian debate, the infrastructure team behind the music streaming service Spotify offered support for *Systemd* after saying "We have some 5,000 physical servers and well over a thousand virtual servers using both public and private clouds running Debian GNU/Linux serving millions of songs to our users every day." Their arguments for *Systemd* encapsulate the reasons many people think *Systemd* is the way forward:

- Its dependency model is easier to understand than *Upstart*'s.
- Features built on top of *Systemd* are very useful.
- *Systemd* has the stronger community momentum.

Additionally, *Systemd* launches processes in parallel and avoids the bottlenecks that blight *init*. It does this by using sockets for services so that daemons can talk to each other and manages those sockets on behalf of daemons that haven't started yet. It's a clever idea.

Q Are there any disadvantages to distros adopting *Systemd*?

A The main problem is that *Systemd* is different, and breaks the simple tool Unix tradition. It's a huge and comprehensive project, and this has caused friction when people have to learn both a new system and throw away hard-earned skills.

It's also complex and, some argue, over engineered for its purpose. Perhaps more importantly, it's Linux-only. But – and this is the important point – *Systemd* has been proven to work, and it works well enough that many distributions have already moved to *Systemd*. Significantly, the Gnome team have announced it will become a dependency for installing the latest versions of their desktop environment, entrenching the technology deeper into our operating systems.

Q Where can we find more details about how to use it?

A Not completely by coincidence, our very own Mike Saunders is currently working on a tutorial that should answer all your practical questions in our next issue, including how to create and modify boot processes, in the way you may already be used to from *init*, and how to use many of those weird *Systemd* tools that are now part of your distribution. LV