3. What happens when you switch on a computer?

A computer without a program running is just an inert hunk of electronics. The first thing a computer has to do when it is turned on is start up a special program called an *operating system*. The operating system's job is to help other computer programs to work by handling the messy details of controlling the computer's hardware.

The process of bringing up the operating system is called *booting* (originally this was *bootstrapping* and alluded to the process of pulling yourself up "by your bootstraps"). Your computer knows how to boot because instructions for booting are built into one of its chips, the BIOS (or Basic Input/Output System) chip.

The BIOS chip tells it to look in a fixed place, usually on the lowest-numbered hard disk (the *boot disk*) for a special program called a *boot loader* (under Linux the boot loader is called Grub or LILO). The boot loader is pulled into memory and started. The boot loader's job is to start the real operating system.

The loader does this by looking for a *kernel*, loading it into memory, and starting it. If you Linux and see "LILO" on the screen followed by a bunch of dots, it is loading the kernel. (Each dot means it has loaded another *disk block* of kernel code.)

(You may wonder why the BIOS doesn't load the kernel directly — why the two-step process with the boot loader? Well, the BIOS isn't very smart. In fact it's very stupid, and Linux doesn't use it at all after boot time. It was originally written for primitive 8-bit PCs with tiny disks, and literally can't access enough of the disk to load the kernel directly. The boot loader step also lets you start one of several operating systems off different places on your disk, in the unlikely event that Unix isn't good enough for you.)

Once the kernel starts, it has to look around, find the rest of the hardware, and get ready to run programs. It does this by poking not at ordinary memory locations but rather at *I/O ports* — special bus addresses that are likely to have device controller cards listening at them for commands. The kernel doesn't poke at random; it has a lot of built-in knowledge about what it's likely to find where, and how controllers will respond if they're present. This process is called *autoprobing*.

You may or may not be able to see any of this going on. Back when Unix systems used text consoles, you'd see boot messages scroll by on your screen as the system started up. Nowawadays, Unixes often hide the boot messages behind a graphical splash screen. You may be able to see them by switching to a text console view with the key combination Ctrl-Shift-F1. If this works, you should be able to switch back to the graphical boot screen with a different Ctrl-Shift sequence; try F7, F8, and F9.

Most of the messages emitted boot time are the kernel autoprobing your hardware through the I/O ports, figuring out what it has available to it and adapting itself to your machine. The Linux kernel is extremely good at this, better than most other Unixes and *much* better than DOS or Windows. In fact, many Linux old-timers think the cleverness of Linux's boot-time probes (which made it relatively easy to install) was a major reason it broke out of the pack of free-Unix experiments to attract a critical mass of users.

But getting the kernel fully loaded and running isn't the end of the boot process; it's just the first stage

(sometimes called *run level 1*). After this first stage, the kernel hands control to a special process called ‘init’ which spawns several housekeeping processes. (Some recent Linuxes use a different program called ‘upstart’ that does similar things)

The init process's first job is usually to check to make sure your disks are OK. Disk file systems are fragile things; if they've been damaged by a hardware failure or a sudden power outage, there are good reasons to take recovery steps before your Unix is all the way up. We'll go into some of this later on when we talk about how file systems can go wrong.

Init's next step is to start several *daemons*. A daemon is a program like a print spooler, a mail listener or a WWW server that lurks in the background, waiting for things to do. These special programs often have to coordinate several requests that could conflict. They are daemons because it's often easier to write one program that runs constantly and knows about all requests than it would be to try to make sure that a flock of copies (each processing one request and all running at the same time) don't step on each other. The particular collection of daemons your system starts may vary, but will almost always include a print spooler (a gatekeeper daemon for your printer).

The next step is to prepare for users. Init starts a copy of a program called **getty** to watch your screen and keyboard (and maybe more copies to watch dial-in serial ports). Actually, nowadays it usually starts multiple copies of **getty** so you have several (usually 7 or 8) virtual consoles, with your screen and keyboards connected to one of them at a time. But you likely won't see any of these, because one of your consoles will be taken over by the X server (about which more in a bit).

We're not done yet. The next step is to start up various daemons that support networking and other services. The most important of these is your X server. X is a daemon that manages your display, keyboard, and mouse. Its main job is to produce the color pixel graphics you normally see on your screen.

When the X server comes up, during the last part of your machine's boot process, it effectively takes over the hardware from whatever virtual console was previously in control. That's when you'll see a graphical login screen, produced for you by a program called a *display manager*.

Prev Home Next
Basic anatomy of your computer What happens when you log in?