2. How Linux Keeps Track of Time

2.1 Basic Strategies

A Linux system actually has two clocks: One is the battery powered "Real Time Clock" (also known as the "RTC", "CMOS clock", or "Hardware clock") which keeps track of time when the system is turned off but is not used when the system is running. The other is the "system clock" (sometimes called the "kernel clock" or "software clock") which is a software counter based on the timer interrupt. It does not exist when the system is not running, so it has to be initialized from the RTC (or some other time source) at boot time. References to "the clock" in the `ntpd` documentation refer to the system clock, not the RTC.

The two clocks will drift at different rates, so they will gradually drift apart from each other, and also away from the "real" time. The simplest way to keep them on time is to measure their drift rates and apply correction factors in software. Since the RTC is only used when the system is not running, the correction factor is applied when the clock is read at boot time, using `clock(8)` or `hwclock(8)`. The system clock is corrected by adjusting the rate at which the system time is advanced with each timer interrupt, using `adjtimex(8)`.

A crude alternative to `adjtimex(8)` is to have `chron` run `clock(8)` or `hwclock(8)` periodically to sync the system time to the (corrected) RTC. This was recommended in the `clock(8)` man page, and it works if you do it often enough that you don't cause large "jumps" in the system time, but `adjtimex(8)` is a more elegant solution. Some applications may complain if the time jumps backwards.

The next step up in accuracy is to use a program like `ntpd` to read the time periodically from a network time server or radio clock, and continuously adjust the rate of the system clock so that the times always match, without causing sudden "jumps" in the system time. If you always have a network connection at boot time, you can ignore the RTC completely and use `ntpdate` (which comes with the `ntpd` package) to initialize the system clock from a time server-- either a local server on a LAN, or a remote server on the internet. But if you sometimes don't have a network connection, or if you need the time to be accurate during the boot sequence before the network is active, then you need to maintain the time in the RTC as well.

2.2 Potential Conflicts

It might seem obvious that if you're using a program like `ntpd`, you would want to sync the RTC to the (corrected) system clock. But this turns out to be a bad idea if the system is going to stay shut down longer than a few minutes, because it interferes with the programs that apply the correction factor to the RTC at boot time.

If the system runs 24/7 and is always rebooted immediately whenever it's shut down, then you can just set the RTC from the system clock right before you reboot. The RTC won't drift enough to make a difference in the time it takes to reboot, so you don't need to know its drift rate.

Of course the system may go down unexpectedly, so some versions of the kernel sync the RTC to the system clock every 11 minutes if the system clock has been adjusted by another program. The RTC won't drift enough in 11 minutes to make any difference, but if the system is down long enough for the RTC to drift significantly, then you have a problem: the programs that apply the drift correction to the RTC need to know **exactly** when it was last reset, and the kernel doesn't record that information anywhere.

Some unix "traditionalists" might wonder why anyone would run a linux system less than 24/7, but some of us run dual-boot systems with another OS running some of the time, or run Linux on laptops that have to be shut down to conserve battery power when they're not being used. Other people just don't like to leave machines running unattended for long periods of time (even though we've heard all the arguments in favor of it). So the "every 11 minutes" feature becomes a bug.

This "feature/bug" appears to behave differently in different versions of the kernel (and possibly in different versions of `xntpd` and `ntpd` as well), so if you're running both `ntpd` and `hwclock` you may need to test your system to see what it actually does. If you can't keep the kernel from resetting the RTC, you might have to run without a correction factor on the RTC.

The part of the kernel that controls this can be found in `/usr/src/linux-2.0.34/arch/i386/kernel/time.c` (where the version number in the path will be the version of the kernel you're running). If the variable `time_status` is set to `TIME_OK` then the kernel will write the system time to the RTC every 11 minutes, otherwise it leaves the RTC alone. Calls to `adjtimex(2)` (as used by `ntpd` and `timed`, for example) may turn this on. Calls to `settimeofday(2)` will set `time_status` to `TIME_UNSYNC`, which tells the kernel not to adjust the RTC. I have not found any real documentation on this.

I've heard reports that some versions of the kernel may have problems with "sleep modes" that shut down the CPU to save energy. The best solution is to keep your kernel up to date, and refer any problems to the people who maintain the kernel.

If you get bizarre results from the RTC you may have a hardware problem. Some RTC chips include a lithium battery that can run down, and some motherboards have an option for an external battery (be sure the jumper is set correctly). The same battery maintains the CMOS RAM, but the clock takes more power and is likely to fail first. Bizarre results from the system clock may mean there is a problem with interrupts.

2.3 Should the RTC use Local Time or UTC, and What About DST?

The Linux "system clock" actually just counts the number of seconds past Jan 1, 1970, and is always in UTC (or GMT, which is technically different but close enough that casual users tend to use both terms interchangeably). UTC does not change as DST comes and goes--what changes is the *conversion* between UTC and local time. The translation to local time is done by library functions that are linked into the application programs.

This has two consequences: First, any application that needs to know the local time also needs to know what time zone you're in, and whether DST is in effect or not (see the next section for more on time zones). Second, there is no provision in the kernel to change either the system clock or the RTC as DST comes and goes, because UTC doesn't change. Therefore, machines that only run Linux should have the RTC set to UTC, not local time.

However, many people run dual-boot systems with other OS's that expect the RTC to contain the local time, so `hwclock` needs to know whether your RTC is in local time or UTC, which it then converts to seconds past Jan 1, 1970 (UTC). This still does not provide for seasonal changes to the RTC, so the change must be made by the other OS (this is the one exception to the rule against letting more than one program change the time in the RTC).

Unfortunately, there are no flags in the RTC or the CMOS RAM to indicate standard time vs DST, so each OS stores this information someplace where the other OS's can't find it. This means that `hwclock` must assume that the RTC always contains the correct local time, even if the other OS has not been run since the most recent seasonal time change.

If Linux is running when the seasonal time change occurs, the system clock is unaffected and applications will make the correct conversion. But if linux has to be rebooted for any reason, the system clock will be set to the time in the RTC, which will be off by one hour until the other OS (usually Windows) has a chance to run.

There is no way around this, but Linux doesn't crash very often, so the most likely reason to reboot on a dual-boot system is to run the other OS anyway. But beware if you're one of those people who shuts down Linux whenever you won't be using it for a while-- if you haven't had a chance to run the other OS since the last time change, the RTC will be off by an hour until you do.

Some other documents have stated that setting the RTC to UTC allows Linux to take care of DST properly. This is not really wrong, but it doesn't tell the whole story-- as long as you don't reboot, it does not matter which time is in the RTC (or even if the RTC's battery dies). Linux will maintain the correct time either way, until the next reboot. In theory, if you only reboot once a year (which is not unreasonable for Linux), DST could come and go and you'd never notice that the RTC had been wrong for several months, because the system clock would have stayed correct all along. But since you can't predict when you'll want to reboot, it's better to have the RTC set to UTC if you're not running another OS that requires local time.

The Dallas Semiconductor RTC chip (which is a drop-in replacement for the Motorola chip used in the IBM AT and clones) actually has the ability to do the DST conversion by itself, but this feature is not used because the changeover dates are hard-wired into the chip and can't be changed. Current versions change on the first Sunday in April and the last Sunday in October, but earlier versions used different dates (and obviously this doesn't work in countries that use other dates). Also, the RTC is often integrated into the motherboard's "chipset" (rather than being a separate chip) and I don't know if they all have this ability.

2.4 How Linux keeps Track of Time Zones

You probably set your time zone correctly when you installed Linux. But if you have to change it for some reason, or if the local laws regarding DST have changed (as they do frequently in some countries), then you'll need to know how to change it. If your system time is off by some exact number of hours, you may have a time zone problem (or a DST problem).

Time zone and DST information is stored in `/usr/share/zoneinfo` (or `/usr/lib/zoneinfo` on older systems). The local time zone is determined by a symbolic link from `/etc/localtime` to one of these files. The way to change your timezone is to change the link. If your local DST dates have changed, you'll have to edit the file.

You can also use the `TZ` environment variable to change the current time zone, which is handy of you're logged in remotely to a machine in another time zone. Also see the man pages for `tzset` and `tzfile`.

This is nicely summarized at http://www.linuxsa.org.au/tips/time.html

2.5 The Bottom Line

If you don't need sub-second accuracy, `hwclock(8)` and `adjtimex(8)` may be all you need. It's easy to get enthused about time servers and radio clocks and so on, but I ran the old `clock(8)` program for years with excellent results. On the other hand, if you have several machines on a LAN it can be handy (and sometimes essential) to have them automatically sync their clocks to each other. And the other stuff can be fun to play with even if you don't really need it.

On machines that only run Linux, set the RTC to UTC (or GMT). On dual-boot systems that require local time in the RTC, be aware that if you have to reboot Linux after the seasonal time change, the clock may be temporarily off by one hour, until you have a chance to run the other OS. If you run more than two OS's, be sure only one of them is trying to adjust for DST.

Next Previous Contents