What is the Time? Part 3

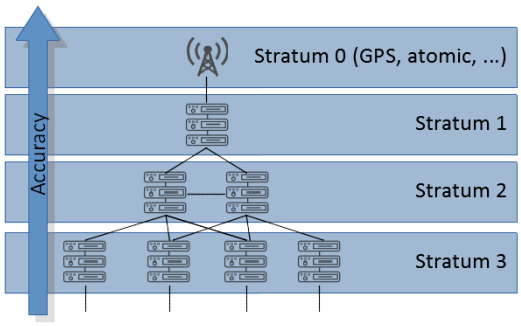
Call: +44 (0)23 8098 8890
E-mail: sales@itdev.co.uk (mailto:sales@itdev.co.uk)

Posted 11th January 2019, By James H (/company/about/team/james-h)

Making a Stratum 1 Linux Time Server: Part 3

We have found out in the previous two parts of this series why distributed devices might need a method to synchronise their time to a reference. One method for doing this is through a time server. In this article I will show you how to setup a stratum 1 NTP time server running on a *Debian* or *Ubuntu* based *Linux* system. For this you will need a GPS receiver that supports RS232 and a 1 pulse per second (PPS) output and a Linux server with a serial port.

The first thing you might wonder is, do I need my own time server? The answer is yes. Most “cloud” compute power you can buy is hosted on virtual machines where you can be allocated more resource the more you pay. This is great if you want to crunch numbers, serve databases and so on. It’s not so good if you want to serve time. First it is unlikely that you can find someone who is maintaining a stratum 1 server and second if it is virtual it almost certainly can’t be stratum 1 because the OS will use virtual timer interrupts to count time. Given that the machine is virtual the hypervisor can cause a virtual OS to miss ticks (https://www.dasher.com/time-synchronization-in-a-virtual-environment/). Not good for a time server.



It’s also quite hard to find a hosting solution that will feed you a GPS source but it can be done. Then you will want to feed the GPS into a *dedicated* bit of hardware that is *not* virtualised. This way you can create your own stratum-1 server. When setting up your own time server be aware of security. You may want to consider setting up a firewall such as UFW, a security module like SELinux or AppArmor, a log monitor like Fail2Ban and generally hardening your system. However, this is outside the scope of this blog.

We can now talk about synchronising your system time to your GPS reference. Your GPS module should output a 1 pulse per second (PPS) signal and a time string in one of the standard formats, like the National Marine Electronics Association standard (NMEA).

The rising edge of the 1 PPS normally indicates the top-of-second, or the very start of the second. The next NMEA string received after this pulse, and before the next pulse, associates a time with this pulse edge. Thus, if only 1 PPS is used you can only frequency align (syntonise) with the reference. This is because you will be able to tell when the second starts, but not which second it is. With both the 1 PPS and NMEA strings you can align both phase and frequency (synchronise) because you know when the second starts and what second you are receiving.

You should note that you probably shouldn’t be using a USB to serial device to receive the 1 PPS and NMEA strings as this will introduce a lot of jitter and latency which must be accounted for. We don’t address this situation here.

To begin we need to make sure we have some libraries/utilities installed:

```
# Get/set Linux serial port information
$ sudo apt-get install --yes setserial

# Get everything you need to build the ntpd
$ sudo apt-get install --yes build-essential
$ sudo apt-get install --yes libcap2
$ sudo apt-get install --yes libcap-dev

# Linux PPS API and support tools.
$ sudo apt-get install --yes pps-tools
```

Next, make sure that your kernel is configured with the following settings (https://www.itdev.co.uk/blog/building-linux-kernel-introduction):

```
CONFIG_PPS=m
CONFIG_PPS_CLIENT_LDISC=m
CONFIG_PPS_CLIENT_PARPOR
```

Next - we are assuming the use of systemd - you will have to disable its built-in NTP daemon. We don’t want this.

```
$ sudo systemctl stop systemd-timesyncd
$ sudo systemctl disable systemd-timesyncd
```

Now we are ready to download the NTP daemon and recompile it. We need to do this because the default package that can be retrieved through an “apt-get” doesn’t know that kernel-based clock disciplining is available.

Create a temporary directory and change directory into it. Then download the source and compile as so:

```
$ wget https://www.eecis.udel.edu/~ntp/ntp_spool/ntp4/ntp-4.2/ntp-4.2.8p11.tar.gz (https://www.eecis.udel.edu/~ntp/ntp_spool/ntp4/ntp-4.2/ntp-4.2.8p1
$ tar xvf ntp-4.2.8p11.tar.gz
$ cd ntp-4.2.8p11/
$ ./configure CFLAGS="-O2 -g -fPIC" \
            --prefix=/usr --bindir=/usr/sbin \
            --sysconfdir=/etc --enable-linuxcaps \
            --with-lineeditlibs=readline \
            --docdir=/usr/share/doc/ntp-4.2.8p11
$ make
$ make check
```

I downloaded the latest version at the time I was doing this. You can check the URL above to see if there are more recent versions and substitute in as appropriate.

Next, create a user and group for the NTP daemon. We don’t want this to be an account that can be logged into. It exists only to set and limit the Network Time Protocol daemon’s (ntpd) permissions.

```
$ sudo useradd --user-group -M \
             -c "Network Time Protocol" \
             -d /var/lib/ntp \
             -s /bin/false ntp
```

Install the NTP daemon:

```
$ sudo make install
$ sudo install -v -o ntp -g ntp -d /var/lib/ntp
```

We’re getting there but we now need to tell the NTP daemon to look for our GPS source and lock to that. You need to edit the file `/etc/ntp.conf` and make sure the only server specified is as follows:

```
server 127.127.20.0 mode 40 minpoll 4 prefer
fudge 127.127.20.0 flag1 1 flag2 0 flag3 0 flag4 0
```

What does all this mean? The most confusing thing here is that we appear to have a local IP address as our server but we’re locking to a 1 PPS and serial connection. These IP addresses are special and do not address network devices but “reference clock drivers”. Addresses of the form “127.127.type.unit” indicate a clock source that has a specific type and unit number. The unit number is used in case there are multiple clocks of the same type.

In our case we can see that the clock type is 20, which is the “Generic NMEA GPS Receiver” clock driver as defined by the NTP documentation (http://doc.ntp.org/4.2.8/refclock.html).

Following the pseudo IP address are some configuration parameters (http://doc.ntp.org/4.2.8/drivers/driver20.html). The mode bit we have set indicates a line speed of 115200 bits per second and that we are using GPRMC NMEA sentences.

Fudge flag 1 enables the use of the PPSAPI, which is a PPS API for Unix-like operating systems. This enables the kernel to do the PPS processing, which is more accurate than leaving it to an application. Flag 3 uses ntpd discipline rather than kernel clock discipline when PPS processing is available i.e. Flag 1 lets the kernel timestamp PPS events, and flag 3 decides who processes the timestamps. Ntpd clock discipline is used as recommended by ntp.org because “there are reasons to disable the use of the kernel discipline … [as] ntpd performs more complex computations than the kernel clock does (http://www.ntp.org/ntpfaq/NTP-s-algo-kernel.htm)”.

What we haven’t shown here is the time fudge factors as these will have to be calibrated for your specific device. The first fudge factor, referred to as “time1” in the ntpd documents, is the time it takes from transmission of the PPS signal to it being time-stamped by the kernel. This could be used to factor out propagation delay if the PPS signal is delivered over a very long cable. What you’re unlikely to calibrate out is the jitter between the PPS signal arriving at the host and the host’s interrupt routine time-stamping the edge.

The second time calibration factor is described as follows: “time2 time specifies the serial end of line time offset calibration factor”. The offset is the time between the PPS pulse edge and the terminating serial character. This is important because the NMEA sentence needs to be matched to a PPS pulse edge. If the sentence generation is too slow, such that the end of the sentence is closer to the next second, it will be wrongly associated. The fudge factor is used to “shift” the NMEA end towards the correct PPS edge (https://github.com/benegon/ntp/blob/master/ntpd/refclock_nmea.c).

We have told the NTP daemon what to sync to, but it must access the correct serial port and 1PPS. How does it do this? The answer is also in the docs. We need to setup a device for the serial link at `'/dev/gps0'` that is a symbolic link to the serial port we are using. For our PPS device we need to setup a symbolic link to it under the name `'/dev/gpspps0'`.

To create the PPS device, we must load the PPS line discipline module. We’ll configure our system to load it on boot. Create the file `'/etc/modules-load.d/pps-source.conf'` and add to it the following line:

```
pps_ldisc
```

This will load the pps ldisc module on boot. This module registers a PPS source in the kernel using the LinuxPPS API (https://www.kernel.org/doc/Documentation/pps/pps.txt) and is what will provide a `'/dev/ppsx'` file, when the PPS line discipline is attached to our serial line via ldattch below.

Having sorted our PPS device, now we need to make sure the symbolic links that ntpd expects always exist. We'll create them on boot by using a udev rule.

Create the file `/etc/udev/rules.d/pps-sources.rules` and add the following lines:

```
KERNEL=="pps0", OWNER="root", GROUP="ntp", MODE="0660", SYMLINK+="gpspps0"
KERNEL=="ttyS2", MODE="0660", RUN+="/bin/setserial -v /dev/%k low_latency irq 17", SYMLINK+="gps0"
```

In the above example the serial port we are using is ttyS2. You'll need to configure this for your serial device. The lines instruct udev to create a symlink `'/dev/gpsppps0'` for `'/dev/pps0'` and `'/dev/gps0'` for `/dev/ttyS2` . It also sets up the serial device to use low latency interrupts.

Next, we need to make our system attach the PPS line discipline to the serial port on boot. Create the file `'/etc/systemd/system/ldattach@.service'` , with the following contents:

```
[Unit]
Description=Line Discipline for GPS Timekeeping for %i
Before=network.target

[Service]
ExecStart=/usr/sbin/ldattach PPS /dev/%i
Type=forking

[Install]
WantedBy=multi-user.target
```

This sets up the NTP daemon to run after the network subsystem is setup.

Now we enable the line discipline as a service so that system will start it every time we boot.

```
$ systemctl enable ldattach@ttyS2.service
$ systemctl start ldattach@ttyS2.service
```

Now reload the udev rules and start the NTP service.

```
$ udevadm control --reload-rules && udevadm trigger
$ service ntp start
```

Give the NTP server a little time to "see" the PPS source and then use ntpstat to see if it has synchronised to your GPS clock source. You should see something like the following:

```
$ ntpstat
synchronised to UHF radio at stratum 1
   time correct to within 1 ms
   polling server every 16 s
```

Debugging NTP & The GPS Reference

Setting up ntpd can be a little "fiddly". If you run into trouble, using some of the techniques in this section can be invaluable.

If you run ntpstat and don't see the above, there are several things you can do. First check that your `'/dev/ttyS2'` device and `'/dev/pps0'` exist i.e. your serial card has been "seen" by the system and bound with a driver and the same for the PPS device. If these don't exist your best bet it to check the system log for errors and debug from there.

If they do exist, however, you could double check that the pps ldisc module has been loaded:

```
$ lsmod | grep pps
pps_ldisc              16384  1
pps_core               16384  2 pps_ldisc,ptp
```

If it isn't, go back over the setup described above and make sure all the steps were completed. Failing that, searching the system log for error messages is your next stop.

If the pps_ldisc module is loaded, make sure that the ldattach daemon is running:

```
$ pgrep -a ldattach
1214 ldattach PPS /dev/ttyS2
```

You should see something like the above. If you don't, try running `'sudo systemctl start ldattach@ttyS2'` and re-running the above. If this does not fix things then you will need to consult your system logs and grep for "ldattach" to see what errors have occurred, if any.

If `'ldattach'` is running, then you should be able to use `'sudo ppstest /dev/pps0'` to verify that PPS pulses are being received:

```
$ sudo ppstest /dev/pps0
[sudo] password for user-account:
trying PPS source "/dev/pps0"
found PPS source "/dev/pps0"
ok, found 1 source(s), now start fetching data...
source 0 - assert 1530543724.980177934, sequence: 8333 - clear  1530535775.181483216, sequence: 384
source 0 - assert 1530543725.980176523, sequence: 8334 - clear  1530535775.181483216, sequence: 384
source 0 - assert 1530543726.980175909, sequence: 8335 - clear  1530535775.181483216, sequence: 384
```

If you're still having trouble, make sure that the symlink `'/dev/gpspps0'` exists and points to `'/dev/pps0'`. Check that the symlink `'/dev/gps0'` exists and points to `'/dev/ttyS2'`.

You can also stop the ntp service (`'sudo service ntp stop'`) and then run it in the foreground with verbose output enabled to get a better idea as to what's going on.

```
$ sudo /usr/sbin/ntpd -n -D 100
```

Alternatively configure ntpd setup to log debug to a file by editing `'/etc/ntp.conf'` to add the following lines and then restarting the service.

```
logfile /var/log/ntp
logconfig=syncall +sysevents
```

Doing it this way allows you to keep the NTP daemon in the background and just look at the logs it outputs.

Hopefully by following the steps in this article you have been able to setup your own stratum-1 time server that is locked to a GPS reference. You should now be able to run all your devices so they agree on the same time to within a typical accuracy of between 10 – 100 milliseconds. If you require better accuracy you might consider using PTP (Precision Time Protocol).

How ITDev Can Help

As a provider of software and electronics design services, we are often working with the Linux and Android operating systems. We have extensive experience in developing, modifying, building, debugging and bring up of these operating systems be it for new developments or working with existing kernels to demonstrate new device technologies and features. We offer advice and assistance to companies considering to use or already using Linux or Android on their embedded system. Initial discussions are always free of charge, so if you have any questions, or would like to find out more, we would be delighted to speak to you. If you are interested in attending a workshop on embedded Linux/Android, or receiving more information on this topic, please sign up to our Embedded Linux Interest Group (https://itdev.us14.list-manage.com/subscribe?u=a0b4fcd76cd92114585f73087&id=f09869841b).

(https://itdev.us14.list-manage.com/subscribe?u=a0b4fcd76cd92114585f73087&id=f09869841b)

- Follow us on Linkedin (https://www.linkedin.com/company/itdev/), Twitter (https://twitter.com/ITDevLtd) or Facebook (https://www.facebook.com/itdevltd/)to find out about our future blog posts.

< prev (/blog/what-time-part-2) next >