Support for Radio Clocks in OpenBSD

Marc Balmer <mbalmer@openbsd.org>
The OpenBSD Project, Micro Systems Marc Balmer

Abstract

Every computer is equipped with at least a clock chip or a general purpose device to provide a timer function. While these timers are certainly precise enough for measuring relatively short periods of time, they are not well suited for keeping the correct time and date over a longer period, since almost every chip drifts by a few seconds per day. Even so called real-time clocks only approximately meet the real time.

External time sources can be used to synchronize the local clock with a much preciser time information. Time signals are disseminated over various systems, the best known are the US american GPS (Global Positioning System) and Time Signal Stations. Time signal stations are available in many countries; while the coding schemes vary from time signal station to time signal station, the decoding principles are similar.

This paper outlines the general problems of setting a computers time at runtime; it will then give an overview about time signal stations and the GPS system. In the last sections the OpenBSD implementation is detailed.

1 Introduction

1.1 Adjusting the System Time

While receiving and decoding the time information is comparatively simple, introducing the current time into the computer system may be a complex task. Generally, it is recommended to set the system time during the boot procedure, for example as part of the startup procedure. In this case, everything is simple and no problems will arise. If it is, however, intended to update the system time while the computer is running and possibly executing programs that rely on absolute time or on time intervals, serious problems may occur.

There are two generally different concepts to change the system time at runtime. The first concept gives maximum priority to the continuity of the time, i.e. the time may be compressed or streched, but under no circumstances may a discrete time value get lost. The second concept regards time as a sequence of time units with fixed length which can neither be stretched nor compressed, but is allowed to miss or insert a time unit.

The distinction between the two different methods is necessary as in every environment the time must not be changed without prior consideration of the software that is running. Imagine a daemon program that has to start other programs at a given time: If the continuity of the time is broken up, a particular program may never be started. Such software would only run properly if time adjustment is done by stretching or compressing the time axis.

Other software may not rely on the absolute time but on the accuracy of the system clock (tick) rate. If, in this case, the time is adjusted by speeding-up or slowing-down the tick rate (i.e. stretching or compressing the time axis), this software will fail. Such software would only run properly if time adjustment is done just by changing the time settings.

If both types of software simultaneously run on the same system, the time cannot be adjusted without producing unpredictable results. In this case, the system time should better not be adjusted at runtime.

2 Time Signal Stations

In the following sections the focus is on time signal stations that emit official time using longwave transmitters.

2.1 Germany: DCF77

An ultra-precise time mark transmitter in Germany, called DCF77, emits a 77.5 kilohertz signal modulated by the encoded current time. This time signal can be used to adjust a computer's real-time clock and to ensure

accurate calendar day and time of the system. An easy-to-build receiver and decoder can be directly attached to a free port; a special driver is needed to decode the incoming time information and to update the system clock whenever needed and desired.

Principally, there are two different possibilities to synchronize the system clock with the DCF77 signal. First, the system clock can be set every time a valid time information is received from the time-mark transmitter; secondly, the update can be done in predefined time periods, for example every 5 minutes. Since the accuracy of the real-time clock device is normally good enough to ensure precise system time and date over a short time period, the second possibility may not only suffice but also minimize system overhead.

2.1.1 The DCF77 Timecode

The DCF77 signal not only provides a very stable frequency, but is also continuously modulated with the current date and time information. The bit codes to provide date and time information are transmitted during the 20th and 58th second of a minute, each bit using a 1-second window. The transmitter signal is reduced to 30the beginning of each second. This reduction lasts for 100 or 200 milliseconds to encode a bit value of 0 or 1, respectively. There is no power reduction in the 59th second; this encodes the beginning of a new minute, so that the time information transmitted during the last minute may be regarded as valid. In consequence, the encoded time information has a maximum precision of one minute. The current second can only be determined by counting the bits since the last minute gap. The following additional information is included in the DCF77 code: daylight saving time, announcement of a leap second at midnight for time adjustments, spare antenna usage and others.

The DCF77 time signal station uses the following encoding scheme to transmit time information:

Bit 15	Call bit
Bit 16	Announcement of DST change
Bit 17-18	Indication of DST
Bit 19	Announcement of a leap second
Bit 20	Start of encoded time information
Bits 21-27	Minute
Bit 28	Parity
Bits 29-34	Hour
Bit 35	Parity
Bits 36-41	Day of month
Bits 42-44	Day of week
Bits 45-49	Month
Bits 50-57	Year
Bit 58	Parity

The time information is in German legal time, that is UTC+1 (or UTC+2 during daylight saving time).

2.2 Switzerland: HBG

The Swiss HBG time signal stations emits the official Swiss time on a frequency of 75 kHz from a transmitter located in Prangins near Geneva. Since 2001 it uses an encoding scheme that is compatible with the German DCF77 station. The only difference to the DCF77 code occurs during the first second of a minute: While there is only one reduction of the emission power in the DCF77 signal, there are two reductions of 100 ms with a gap of 100 ms in the HBG signal. This additional reduction of power can be used to differentiate between the two stations. During the first second of a new hour, we see three such power reductions, at noon and midnight, even four reductions are used.

2.3 Japan: JJY

The official japanese time is disseminated using two longwave time signal station transmitter, one is located on Mount Otakadoy near Fukushima and the second on Mount Hagane on Kyushu Island. The code is different from the Swiss and German codes and due to the lack of a suited receiver and for obvious geographical constraints, no driver support has yet been writte for JJY.

2.4 Connecting the Receiver

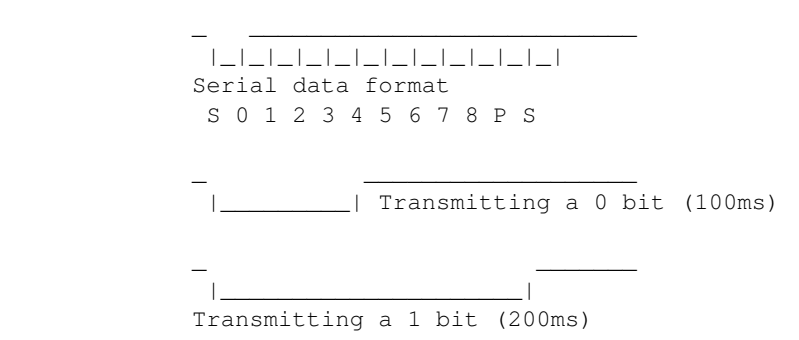
Various techniques are applicable to connect a DCF77 receiver to a computer system: The most obvious way is to convert the two states of the amplitude to a square wave signal, i.e. to a stream of zeroes and ones. In this case, the length of the reduction is determined to define the logical bit state.

A more sophisticated way that is used by most commercially available receivers is to provide a data line with the decoded time information and a separate clock line. These lines can be used to load a shift register or to enter the data on an input port bit using the clock line as an interrupt generating strobe. A device driver is required to read the data line and to combine the bits to the complete time information. In this case, the time of the amplitude reduction is measured by the receiver.

Most DCF77 receivers, however, not only provide the data and clock line but also the undecoded square wave signal on the clock line as additional information. This allows for a rather unconventional, but easy, method to both connect the receiver to the computer hardware and decode the time information. In essence, this method is based on interpreting the undecoded square wave signal as if it were an RS-232 bit stream signal so that a standard serial interface and the standard read command can be used.

2.4.1 Using the Serial Interface

Before the bit stream signal can be used as an RS-232 input signal, the required communication parameters must be determined and the controller be programmed accordingly: The longest duration of the low state of the square wave signal (200 ms) is taken as ten consecutive serial bits (one start bit, eight data bits, one parity bit) each of 20ms, so that a total of 50 bits would be transfered per second. Consequently, if the serial interface is set to 50 Baud, 8 data bits and even-parity, a 20ms section of the square wave signal represents one bit in the controllers input byte; the first 20ms section, however, is not considered since it is interpreted as a start bit.



A logical 0 is encoded from the time signal station as a low-level pulse of 100 ms duration and therefore causes the serial controller's input byte to contain binary 11110000 (hexadecimal F0). A logical 1 encoded as a 200 ms low-level pulse simply causes an input byte of 0.

The only hardware requirement to connect the square wave signal from the DCF77 receiver to a serial RS-232 port of a computer is a TTL-to-V.24 level shifter. Only the receive data (RxD) and ground (GND) pins are used, all other pins remain unused.

2.5 Decoding the Time

The very low level functions do nothing more than collecting the time bits and storing them in an appropriate structure. As the low-level interface may need to synchronize with the DCF77 transmitter, it cannot be expected to return very quickly. Under worst-case condition, i.e. if the function is entered just after the first bit has been transmitted, it may take up to two minutes until the function has completely collected the time information.

2.5.1 Collecting the Bits

There is, however, still a problem when using the serial interface to decode the time information. This problem is due to the evaluation of the parity bit. Both input byte values, hexadecimal F0 and 0, would need the parity bit to have the even (high level) state. This is alright in the first case; but in the second case (200 ms low-level state), the input signal has still low-level when the parity bit is expected so that a parity error is generated. The decoding routine has, therefore, to consider this condition.

2.5.2 The Decoding Algorithm

At any time the decoder is started, it must synchronize with the DCF77 time signal station. To do so, it waits for a delay between two bits that is significantly longer than one second, e.g. 1.5 seconds.

After this prolonged gap, the next bit to receive is bit zero of the subsequent time information packet. The algorithm then picks up the bits until all 59 data bits are collected. In case the algorithm falls out of synchronization and misses some bits, perhaps due to bad reception of the radio signal, it detects the data loss by the time elapsed between two bits which must not be more than one second. In this case, the algorithm synchronizes with the DCF77 signal again and restarts.

Once the complete time information is received, it is immediately decoded and marked valid as soon as bit 0 of the following minute starts. The time obtained can then be used to adjust the system time.

If the time decoding function is re-entered within less than one second interval, it does not need to resynchronize. In this case, the function waits for the next data bit and stores it. In consequence, such a function needs at least two minutes only for the first call; if, however, called at regular and short intervals, the function returns after about one second so that, for example, a display can continuously be updated.

2.6 We Are Always Late

The method described above to decode the DCF77 data bits has one disadvantage: We are always late. When the read function returns, indicating that a new second just started, we are already late for 100 or 200 milliseconds which is the time that is needed to transmit the data bit. This delay must be taken into consideration when the exact time of the starting of a new second is needed.

3 GPS

The American Global Positioning System, or GPS for short, works completely different than time signal stations and its primary purpose is not the dissemination of time, but accurate three-dimensional position information all over the world. To determine the exact position with the help of satellites, very high presion timing is used. This makes the GPS system a very interesting op-

tion to receive time information: It is available globally and it inherently carries very precise time information.

GPS receivers can be very cheap and are available as USB connected devices, serially attached receivers, even PC-Card or CF attached devices exist.

Professional GPS receivers are available as PCI cards, e.g. from the German manufacturor Meinberg Funkuhren.

OpenBSD currently has support for GPS in the nmea(4) and mbg(4) codes which are described in the following section.

4 OpenBSD Implementation

All time signal stations use their own code. All have some properties of their own, like the announcement of leap seconds or the announcement of a daylight saving time change in the DCF77 code. Time is encoded in a local timezone for most stations. In consequence, all drivers that decode a particular time signal station code should follow a common, yet minimal, protocol: Report the time in coordinated universal time (UTC), not the local time.

4.1 The Sensor Framework

OpenBSD has a sensor framework with sensors being read-only values that report various environmental values like CPU temparature, fan speeds or even acceleration values if the computer is equipped with accelerometers. Sensor values can be read using the sysctl(8) interface from the commandline or in userland programs.

To support radio clocks, a new type of sensor has been introduced, the timedelta sensor that reports the error (or offset) of the local clock in nanoseconds. A radio clock device driver provides a timedelta sensor by comparing the local time with the time information received.

A userland daemon like ntpd(8) can then pick up this time information and adjust the local clock accordingly.

Timedelta sensors not only report the error of the local clock, but they also have a timestamp value indicating when exactly the last valid time information has been decoded and a status flags indicating the quality of the time information. Initially, this status is set to UNKNOW, it will then change to OK once proper time information has been received. Some radio clock drivers, e.g. udcf(4), will degrade to the WARNING state if not valid time information has been recived for more than 5 minutes. If not time is received for a longer period, the state will eventually degrade to ERROR.

4.2 udcf(4)

During the development the actual driver implementation, I have used various Expert mouseCLOCK devices manufactured in Germany by Gude Analog und Digital Systeme.

The Expert mouseCLOCK USB and the Expert mouseCLOCK are inexpensive devices available in several variations. They can be interfaced serially or using USB and decode either the German DCF77 station, the Swiss HBG station, or the British MSF time signal station.

Although the serial devices use standard V.24 signal levels, they are not serial devices in the usual sense. They use the signal pins to provide a serial stream of bits that can be decoded with the method outlined above.

The USB attached devices interestingly contain an ISDN controller with USB interface that controls the receiving part using auxillary ports of the ISDN controller.

The implemented driver, udcf, attaches to a uhub device. When the device is plugged in to a USB hub, the driver programs the device to provide power to the receiver part. It then sets up a timer in 2000 ms to let the receiver stabilize. When this timer expires, the driver starts its normal operation by polling the device over the USB bus for the falling edge of the signal in a tight loop. Once the falling edge is detected, this fast polling stops and a set of four timers is used to decode the signal.

When the device is removed from the USB hub, all timers are stopped.

On the falling edge of the signal, i.e. at the beginning of a second, fast polling is stopped and some timers are started using timeout(9). Any pending timers are first reset using timeout_del(9).

The first timer expires in 150 ms. Its purpose is to detect the bit being transmitted. The current signal level is measured — if the low power emission was 100 ms long, we have a high level again;, if it is a 200 ms emission, we still have a low level. The bit detected is stored.

The second timer expires in 250 ms and is used to detect if we decode the German DCF77 or the Swiss HBG signal.

The third timer expires in 800 ms. It is used to restart fast polling over the USB bus to detect the falling edge of the signal at the start of the next second. Note that there might not be a falling edge for more than one second during the minute gap after the 58th second. This situation is detected using a third timer.

The fourth timer expires in 1500 ms after the falling edge. When it expires, we have detected the 59th second. Note that this timer will not expire during seconds 0-58 as all timers are reset when the falling edge is detected using the fast polling loop.

In the 59th second we decode and validate the com-

plete time information just received and at the beginning of the next second we stamp the time information with microtime(9) and mark it as valid. A userland program can get at this information and knowing the received time information and the exact system time when it was valid, the userland program can calculate the exact time.

The fifth timer expires in 3000 ms. If it ever expires, we have lost reception. We set an error bit and stop the whole decoding process for some time.

The four timers need not be very precise (10% tolerance is very acceptable) - the precision of the time decoding is solely determined by the detection of the falling edge at the start of a second. All means should be taken to make this detection as precise as possible.

When the algorithm is started we do not know in which second we are, so we first must synchronize to the DCF77 signal. This is done by setting the state to synchronizing, in which we don't store the bits collected, but rather wait for the minute gap. At the minute gap, the state is changed from synchronizing to receiving. Whenever we lose reception or time information gets invalid for other reasons, we fall back to the synchronizing state.

4.2.1 Using Interrupts

The driver described above is very easy to use. But it has limitations as polling over the USB bus has to be done to detect the falling edge at the beginning of a second. It is basically this polling loop that limits the precision of the time information. Higher precision can be obtained when the falling edge of the signal causes an interrupt. No polling is needed then and the decoding driver needs only some slight adjustments.

When the falling edge is detected, further interrupts from the device are disabled and the second timer, used to restart fast polling in the udcf driver, is used to reenable interrupts from the time signal receiver, thus debouncing the signal.

The serial versions of the time signal receivers can be rather easily used to generate these interrupts. Instead of using the standard wiring, the data line that provides the signal level is attached to an interrupt generating pin of the serial port.

The default serial driver must of course be disabled and the time signal station driver must program the UART accordingly.

4.3 nmea(4)

To use GPS receivers as time source, nmea(4) has been added to OpenBSD. Unlike the other implementations presented in this paper, nmea(4) is not a device driver, but a tty line discipline. A tty line discipline consists of a set of functions that are called by the tty driver on events like a character has been received, a character is to be sent etc. Thus a line discipline can look at (and manipulate) a serial data stream on a tty device.

The purpose of the nmea(4) line discipline is to decode a serial NMEA 0183 data stream originating from an attached GPS device. NMEA is rather simple, ASCII based protocols where a NMEA speaking device emits so called sentences that always start with a $ character and extend to a CR-LF pair. No sentence is longer than 82 characters and there is an optional checksum. To decode the time information, it is sufficied to decode the GPRMC sentence, the "Recommended Minimum Soecific GPS/TRANSI Data".

nmea(4) supports all GPS devices that emit NMEA sentences and that attach to a serial port of some sort (RS-232, USB, or PC-Card).

There is a problem, however, with simply decoding the NMEA sentence. We have no indication **when** exactly the time information just received was actually valid. The nmea(4) line discipline takes a local timestamp when it receives the initial $ character and uses this timestamp as the base for the calculation of the local clock offset. This automatically leads to jitter but nevertheless this method gives us accurate date and time information.

4.3.1 TTY Timestamping

To address this problem, tty timestamping has been added to the OpenBSD tty driver.

Some GPS receivers provide a highly accurate pulse-per-second, PPS, signal. A PPS signal typically has microsecond accuracy and with PPS enabled, the GPRMC sentence indicates the time (and position) of the last PPS pulse. So if we can measure the exact local time when the pulse occurs, we can later, when we received the GPRMC sentence, calculate the local offset with very high precision.

This is done in the tty driver when tty timestamping is enabled. Once enabled, the tty driver will take a local timestamp at the very moment the PPS signal occurs (which must be wired to the serial ports RTS or DCD line). The nmea(4) line discipline will then use this timestamp as the base for its calculations once the GPRMC sentence is received.

To attach the nmea(4) line discipline to a tty device, the utility program nmeaattach(8) can be used which can also enable tty timestamping.

Userland programs that want to use the NMEA data as well can do so as nmea(4) does not consume the data, it only looks at it. So with the proper setup, a general GPS software like gpsd can be used to do whatever you want with e.g. the position data while the running kernel just uses the time information to keep the clock adjusted.

4.4 mbg(4)

The mbg(4) driver for radio-clocks supports the professional radio-clocks manufactured by Meinberg Funkuhren in Bad Pyrmont, Germany. Meinberg produces a range of industrial grade receivers for the German DCF77 time signal station and the global GPS system. All cards have a local real-time clock that can be free-running on a local oscillator, which on request is temperature compensated.

The mbg(4) currently supports the PCI32 and PCI511 DCF77 receiver cards and the GPS170 GPS receiver card. All cards provide the exact time information which is available to the driver at any time, plus status information.

Especially with the newer cards PCI511 and GPS170 a very high precision can be achieved, as these cards take the internal timestamp at the very moment the first command byte is written to the card over the PCI bus. The mbg(4) driver uses a very small critical section, protected by splhigh(9), to first take the local timestamp and then send the command to the card. The critical section is immediately left and the driver waits then for the card to return the time information.

Using a kernel timeout(9), the card is queried for time information every ten seconds.

As of the time of this writing, the mbg(4) driver is still under active development, so we expect to achieve higher precision with this driver in the future.

5 Conclusion

With the advent of timedelta sensors, tty timestamping and the drivers presented in this paper, OpenBSD now has complete support for precise time acquisition, keeping and distribution.

The elegant concept of timedelta sensors, an idea by Theo de Raadt, provides a very thin layer of abstraction that allows to provide time information in a uniform way to the sytem from devices as different as a time signal station receiver that is polled over the USB bus to a PCI based GPS receiver card.

The OpenNTPD daemon ntpd(8) can then be used to distribute the time information in the network.

All this makes OpenBSD an ideal platform for time servers.

6 Acknowledgments

Meinberg Funkuhren donated several PCI based GPS and time signal station receiver cards for the development of mbg(4).
Gude ADS donated several Expert Mouse CLOCK devices for the development of the udcf(4) driver.
The concept of timedelta sensors was an idea of Theo de Raadt who also did the implementation of the tty timestamping code.
Several OpenBSD users donated radio clocks of any kind to help with time related development, which was much appreciated.
Many OpenBSD developers helped in various ways, be it by testing the code or by pushing me in the right direction.

7 Availability

nmea(4) and udcf(4) are included in OpenBSD since the 4.0 release. The newer mbg(4) driver will be included in the upcoming 4.1 release.

`http://www.openbsd.org/`

About the Author

After working for Atari Corp. in Switzerland where he was responsible for Unix and Transputer systems, Marc Balmer founded his company micro systems in 1990 which first specialised in real-time operating systems and later Unix. During his studies at the University of Basel, he worked as a part time Unix system administrator.

He led the IT-research department of a large Swiss insurance company and he was a lecturor and member of the board of Hyperwerk, an Institute of the Basel University of Applied Sciences.

Today he fully concentrates on micro systems, which provides custom programming and IT outsourcing services mostly in the Unix environment.

Marc Balmer is an active OpenBSD developer; he was chair of the 2005 EuroBSDCon conference that was held at the University of Basel.

In his spare time he likes to travel, to photograph and to take rides on his motorbike. He is a licensed radioamateur with the call sign HB9SSB.