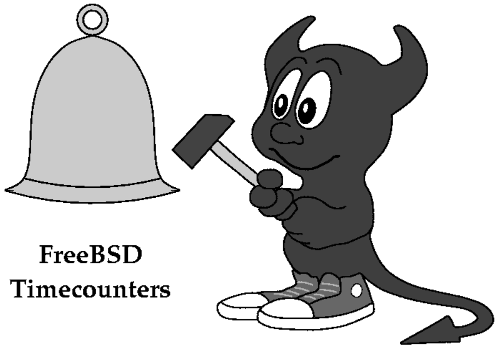


FreeBSD Timecounters

Poul-Henning Kamp

phk@FreeBSD.org

The FreeBSD Project

At the sound of the tone...

- We don't know what time actually is.
- We can measure it very, <u>very</u> precisely.
- We define length and other fundamental units of measurement using time.
- Everybody has a very good mental model of time.
- ... but we still don't know what time <u>is</u>.

Counting oscillations

- All relevant timekeeping is based on counting oscillations of some sort.
- Linear phenomena has higher losses than rotational phenomena.
- Linear phenomena has more boundary conditions than rotational phenomena.
- Suitable rotational phenomena available.

Build your own clock.

- A clock consists of:
  - Oscillator.
  - Counter.
- Counting is trivial, forget the counter.
- Three important properties of the oscillator:
  - Stability.
  - Resolution.
  - Precision.

Oscillators

- Mechanical oscillators are ok, but they wear out and generally are a lot of work.
- Quartz is a fantastic good oscillator due to a lucky mix of special properties.
- Hyperfine atomic emission lines are probably as good as it gets, since nothing much can disturb the phenomena, only our measurement of it.

Timescales

- A timescale consists of an “origo” and some well defined timeinterval repeated thereafter.
- We cannot “go back to the origo and measure again”
- We must rely on “dead reckoning” and count very carefully.

TAI

- “Time Atomic International”
- Sequence of SI-seconds starting 1958.
- SI-seconds defined as:
  - The duration of 9,192,631,770 periods of the radiation corresponding to the transition between the two hyperfine levels of the ground state of the caesium 133 atom.

UTC

- Universal Time Coordinated.
- Also known as “Zulu-time”
- Same second as TAI, counts from random middle-east event, inserts or deletes “leap-seconds” to match local astronomy.
- The Earth is not a very precise clock.
- Leap-seconds are a pain, often just ignored.

UNIX Time

- Like UTC.
- Counts SI seconds since 1970, ignoring leap-seconds when they happen.
- Timeintervals are wrong if they span a leapsecond.
- NTP has an interesting task coping with this braindamage.

UNIX timestamps.

- time_t
  - Seconds since 1970
- Struct timeval
  - time_t + microseconds.
- Struct timespec
  - time_t + nanoseconds.

POSIX BRAINDAMAGE

```
tv->tv_sec = tv1->tv_sec + tv2->tv_sec;
tv->tv_usec = tv1->tv_usec + tv2->tv_usec;
if (tv->tv_usec > 1000000) {
        tv->tv_sec++;
        tv->tv_usec -= 1000000;
}
```

Struct bintime

-
- time_t with 64 bit <u>binary</u> fractional seconds.
- Resolution = 5.421E-20 seconds.
- Simple arithmetic.
- Simple conversion:
  - nanosec = (fraction * 1000000000) >> 64;
  - Fraction = nanosec * (2^64/1000000000);

CPU frequency vs. Resolution

- 32 bits is just not enough:
  - 2^32 Hz = 4.294... GHz.
  - 2^32 Hz = 7cm wavelength
-
- 64 bits <u>is</u> enough:
  - 2^64 Hz = 18 GigaGiga Hz
  - 2^64 Hz = 16 pico-meter wavelength

Timecounter hardware support

- Requirements:
  - A binary counter of sufficient width.
  - Running at a constant known frequency.
- Optional:
  - Readable in single atomic operation.
  - External event latch

Figuring out the time:

- Read hardware count.
- Subtract reference count.
- Scale to “bintime” resolution & format.
- Add reference timestamp
- (done)

Avoiding overrun

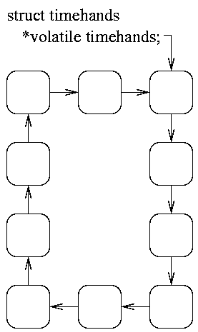
- At regular intervals:
  - Read hardware count.
  - Calculate timestamp (as on previous slide)
  - (Do seconds-rollover NTP/PLL/FLL routine.)
  - Count and timestamp becomes new reference.
  - Update cached timestamps.
  - (done)

Avoiding locks

- Acquiring a free lock is still expensive.
- Time has specific predictable properties.
- Use stable-storage with generation-number.

A ring of clocks...

- All structures are valid
- One of them is “current”
- Periodic update makes the next in turn the “current”
- Timestamping always starts with “current”
- Generation number to spot any races.
-



Periodic update.

-

```
th = timehands->next;
gen= th->generation;
th->generation = 0;
/* update things */
if (++gen == 0)
    gen = 1;
th->generation = gen;
timehands = th;
```

Timestamps, once more

-
```
do {
    gen = tc->generation;
    /* the timestamp math */
} while (tc->generation != gen || gen == 0);
```

"Update things"

-
-
```
count = read_counter(th->th_counter);
th->timestamps = math(th, count);
th->offset = count;
if (new second)
  Call NTP/PLL/FLL
  calc_factors(th)
```

Changing hardware.

-
```
th = timehands->next;
ocount = read_counter(th->th_counter);
th->timestamp = math(ocount);
th->th_counter = newhardware;
th->offset = read_counter(th->th_counter);
calc_factors(th);
timehands = th;
```
- Generation stuff elided for clarity.

Changing frequency.

-
```
th = timehands->next;
th->offset = read_counter(th->th_counter);
th->timestamp = math(th->offset);
calc_factors(th);
timehands = th;
```
- Generation stuff elided for clarity.

Hardware interface (1)

- Struct timecounter {
```
        tsc_get_timecount,  /* get function */
        0,                  /* No poll_pps */
        ~0u,                /* Counter mask */
        0,                  /* Frequency */
        "TSC"               /* Name */
} tsc_timecounter;
```
- Frequency calibrated and filled in by boot code.

Hardware interface (2)

- static unsigned
```
tsc_get_timecount(struct timecounter *tc)
{
    return (rdtsc());
}
```
- [...]
```
if (tsc_freq != 0 && !tsc_broken) {
   tsc_timecounter.tc_frequency = tsc_freq;
   tc_init(&tsc_timecounter);
}
```

Timestamps API

- [get]{bin,nano,micro}[up]time();
  - "get" -> low resolution, approx 1-10 msec.
  - "bin" -> struct bintime
  - "nano" -> struct timespec
  - "micro" -> struct timeval
  - "up" -> time since boot (else POSIX/UTC)
- "time_second" and "time_uptime" globals.
  - for very low granularity needs.

Conclusion.

- Timecounters work in FreeBSD.
- They have exposed a fair bit of code which didn't.
  - "microuptime went backwards"
- Hardware limited performance.
- Rich API for delivering time.
- Adding new hardware takes very little code
  - (Not counting code to deal with broken HW).