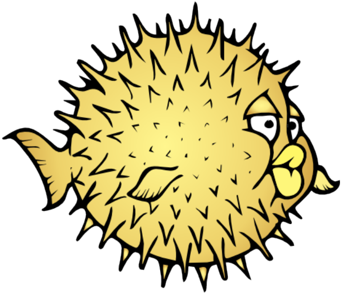
Your scheduler is not the problem



Martin Pieuchot
mpi@openbsd.org

EuroBSDcon, Paris

September 2017

Your OS doesn’t work

A consulting generally begins with:

- It’s OpenBSD fault
- It doesn’t scale
- The scheduler sucks
- I’ll switch to Linux

Fine, let’s take an example.

Agenda

Major Firefox regression

First little hacks

Real solution

Conclusion

Agenda

Major Firefox regression

First little hacks

Real solution

Conclusion

Firefox 40

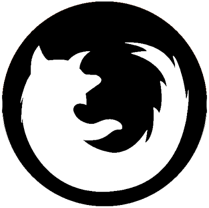
- Released in August 2015
- Multiple complaints of regression
- Nothing obvious in the Changelog
- Switched to ESR then Chrome
- Problem fixed?

Complaints

```
On 06/01/16(Wed) 11:19, Landry Breuil wrote:
> [...]
> i’ve had multiple ppl coming to me privately about this - Yes,
> performance with firefox has been steadily degrading [...]
```

When you complain, don’t forget relevant information.

Black box analysis



Different metrics between old and new?

fstat(1), **ifconfig(8)**, **iostat(8)**, **lsusb(8)**, **netstat(1)**, **nfsstat(1)**, **pfctl(8)**, **ps(1)**, **pstat(8)**, **route(8)**, **systat(1)**, **vmstat(8)**, …

Different metrics

- **vmstat(8)** reported 30K+ IPIs

```
$ vmstat -i
interrupt                       total     rate
irq0/ipi                     182906012   31511
...
```

- **top(1)** showed that CPUs play ping-pong

ktrace or it didn’t happen

```
13288/1032189 firefox-bin RET   sched_yield 0
13288/1032189 firefox-bin CALL  sched_yield()
13288/1010095 firefox-bin CALL  sched_yield()
13288/1010095 firefox-bin RET   sched_yield 0
13288/1010095 firefox-bin CALL  sched_yield()
13288/1027370 firefox-bin CALL  sched_yield()
13288/1032189 firefox-bin RET   sched_yield 0
13288/1032189 firefox-bin CALL  sched_yield()
13288/1027370 firefox-bin RET   sched_yield 0
13288/1027370 firefox-bin CALL  sched_yield()
13288/1032189 firefox-bin RET   sched_yield 0
13288/1032189 firefox-bin CALL  sched_yield()
13288/1027370 firefox-bin RET   sched_yield 0
13288/1010095 firefox-bin RET   sched_yield 0
13288/1027370 firefox-bin CALL  sched_yield()
13288/1010095 firefox-bin CALL  sched_yield()
13288/1032189 firefox-bin RET   sched_yield 0
13288/1032189 firefox-bin CALL  sched_yield()
13288/1027370 firefox-bin RET   sched_yield 0
13288/1010095 firefox-bin RET   sched_yield 0
```

Problem isolation

Difference between ESR and Nightly:

```
$ grep sched_yield kdump-esr.txt |wc -l
       4
$ grep sched_yield kdump-nightly.txt  |wc -l
    89418
```

Limite the scope of research

Which code is being executed?

- Search **sched_yield(2)** on bxr.su and dxr.mozilla.org
  - □ used by Firefox directly
  - □ used by librthread
- Let's use **ltrace(1)**

```
$ LD_TRACE_PLT="" LD_TRACE_PLTSPEC="libpthread" DISPLAY=:0 firefox &
$ ltrace -p $pid -t cu -u libpthread ; sleep 2; ktrace -C
```

_spinlock()

```
$ less kdump-nightly.txt
13288/1027370 firefox-bin USER  .plt symbol: 11 bytes        "_spinunlock"
13288/1010095 firefox-bin USER  .plt symbol: 9 bytes        "_spinlock"
13288/1010095 firefox-bin USER  .plt symbol: 12 bytes        "_atomic_lock"
13288/1010095 firefox-bin CALL  sched_yield()
13288/1010095 firefox-bin RET   sched_yield 0
13288/1027370 firefox-bin USER  .plt symbol: 9 bytes        "_spinlock"
13288/1010095 firefox-bin USER  .plt symbol: 12 bytes        "_atomic_lock"
13288/1027370 firefox-bin USER  .plt symbol: 12 bytes        "_atomic_lock"
13288/1010095 firefox-bin CALL  sched_yield()
13288/1027370 firefox-bin CALL  sched_yield()
13288/1032189 firefox-bin RET   sched_yield 0
13288/1032189 firefox-bin USER  .plt symbol: 12 bytes        "_atomic_lock"
13288/1032189 firefox-bin CALL  sched_yield()
13288/1027370 firefox-bin RET   sched_yield 0
13288/1027370 firefox-bin USER  .plt symbol: 12 bytes        "_atomic_lock"
13288/1027370 firefox-bin CALL  sched_yield()
13288/1032189 firefox-bin RET   sched_yield 0
```

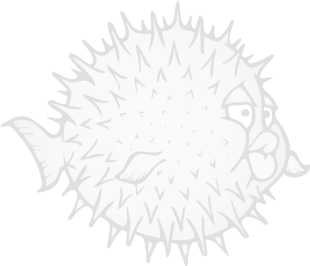
Agenda

Major Firefox regression

First little hacks

Real solution

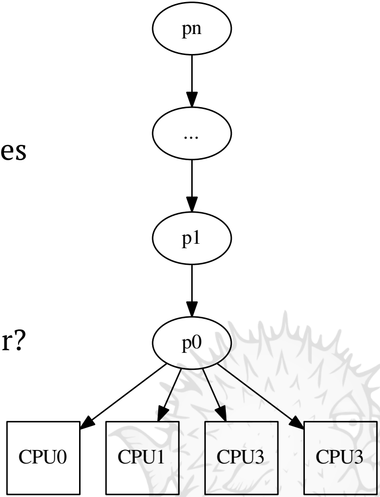
Conclusion



Don’t guess

I started by ripping out per-CPU queues

- It worked
  - □ I could watch HD videos again
  - □ but why?
- Is this problem inside the scheduler?



Deeper inspection

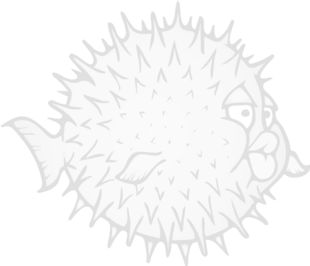
- **gdb(1)**
  - needs debug symbols for ports
  - needs better support for threaded programs
- **printf** debugging

```
0x13da04988d00 called yield() 900 times from <_rthread_mutex_lock+0x58>
0x13da8a19de00 called yield() 1000 times from <pthread_cond_timedwait+0x363>
0x13da04988d00 called yield() 1000 times from <_rthread_mutex_lock+0x58>
0x13da8a19de00 called yield() 1100 times from <pthread_cond_timedwait+0x363>
0x13da04988d00 called yield() 1100 times from <_rthread_mutex_lock+0x58>
0x13da8a19de00 called yield() 1200 times from <pthread_cond_timedwait+0x363>
0x13da04988d00 called yield() 1200 times from <_rthread_mutex_lock+0x58>
```

Read some code

Scheduling priorities are:

- Inherited from 4.4BSD
- Recalculated when sleeping
- Decreased when running
- sched_yield(2) doesn’t guarantee progress
  - □ Keep running until your priority drops



Thread yield hack

Overwrite priority of yielding thread:

```
/*
 * If one of the threads of a multi-threaded process called
 * sched_yield(2), drop its priority to ensure its siblings
 * can make some progress.
 */
p->p_priority = p->p_usrpri;
TAILQ_FOREACH(q, &p->p_p->ps_threads, p_thr_link)
        p->p_priority = max(p->p_priority, q->p_priority);
```

- Improve 3rd party: ffmpeg, Java, chromium, MariaDB…
    - no matter if they use **sched_yield(2)** directly or not

Agenda

Major Firefox regression

First little hacks

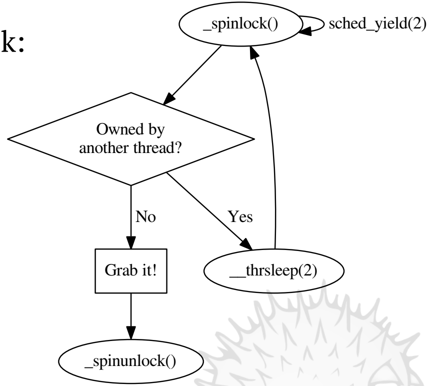
Real solution

Conclusion

6.1 pthread_mutex_lock(3)

- Internal state protected by a lock:
  - based on a Spinlock, and
  - **__trhsleep(2)**:
    - atomically *release* a lock
    - go to sleep

- In the contented case:
  - spin before & after sleeping
  - N atomic operations
  - N syscalls



_spinlock()

sched_yield(2)

Owned by another thread?

No

Yes

Grab it!

__thrsleep(2)

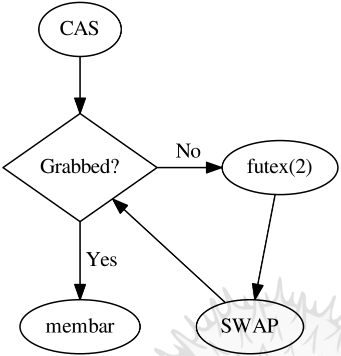
_spinunlock()

Snowball effect with **sched_yield(2)** & Scheduler.

6.2 pthread_mutex_lock(3)

- Internal state is the lock:
  - based on a **C**ompare **A**nd **S**wap,
  - an atomic Swap,
  - a memory barrier, and
  - **futex(2)**:
    - sleep until unlock

- In the contented case:
  - no spinning
  - 1+1 atomic operations
  - 1+1 syscall



Improve latency of threaded programs: git, chrome, GNOME…

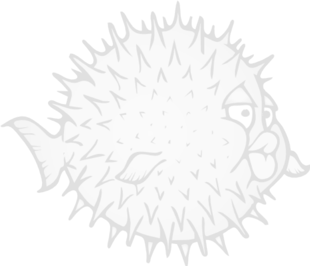
Why futex(2)?

Make it easier for others to contribute. **NIH**, so we can rely on:

- Existing literature, blogs, papers
  - well described in Futexes Are Tricky from U. Drepper
- Multiple kernel implementations
- Multiple libc implementations
  - glibc, musl, bionic
- Existing regression tests

Software is never finished

- Test & convert more architectures
  - enabled on x86 and mips64 for the moment.
  - take care of hardware not providing CAS
- Get rid of the remaining spinning bits
  - **pthread_mutex_*()** and **pthread_convar_*()** for the moment
  - **sched_yield(2)**-free libpthread
- Continue improving the scheduler
  - current bottleneck is in the kernel



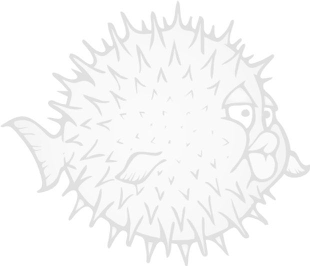
Agenda

Major Firefox regression

First little hacks

Real solution

Conclusion



Conclusion

- OSes will always have problems, complaining will not help
- Gathering basic information is trivial and helps
  - **top(1)** & **systat(1)**
  - **ktrace(1)** or it didn’t happen
- Be sure you understand the bottleneck, guesses are dangerous
  - A change might hide the real problem
  - The Scheduler wasn’t the problem here
- Finding where the bottleneck is, that’s hard
  - Fixing it, that’s generally easier & fun
- Yes, a dynamic tracer would help and I’m working on that

Questions?

Slides on https://www.openbsd.org/papers/

More stories on http://www.grenadille.net

You have a similar problem? Come talk to me!