Thorsten Leemhuis

The Linux Kernel: We have to finish this thing one day ;)

Solving big problems in small steps for more than two decades

twentieth (F)OSDEM already?

time flies…

big round of applause please:

for organizers and all other volunteers!

you made and make this great conference happen! many thx!

warning: this talk is part of the history track

but no, won't be a boring history class

I promise!

everything I mention is kinda relevant for today and tomorrow

there will be a moral of the story in the end

so let's get started…

= the stage =

the first (F)OSDEM happened in 2001

Linux 2.4 had just been released

had about all important features it needed back then

all needed to conquer the world!

proper Posix support

X was running (0.95)

arch portability (1.2 & 2.0)

SMP (2.0)

proper performance

this and many other important things

since then it got tons of improvements…

this talk will only give a glimpse into what happened

= growing up =

2.4 likely would not run too well on today's computers

due to missing drivers, obviously, but also…

numbers of CPU cores would be problematic

back then, uniprocessor systems were the norm

today, we have CPUs with 12 or 16 cores not that expensive

and even smartphones often have at least four cores

Linux was SMP capable since 2.0 (Jun 1996)

was realized with the help of a big hammer

Big Kernel Lock / BKL

only one CPU core is allowed to execute kernel code at any time

with obvious performance impact ;-)

finer graded locking followed in 2.2

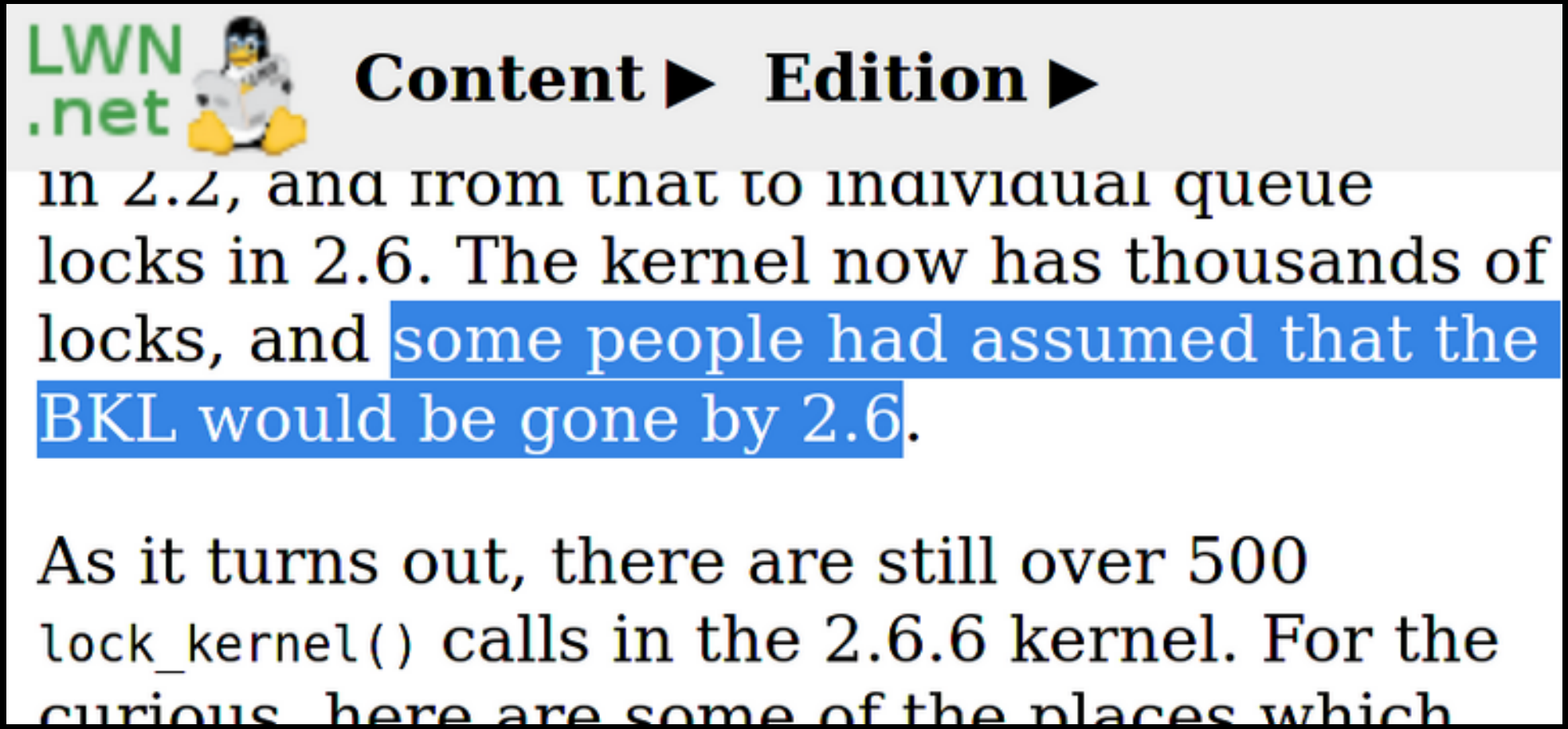
even more in 2.4

that made Linux
better at scaling

still: in the 2.4.x days, other Unixes were known to scale better

by 2.6 (Dec 2003):

Linux got thousands of finer-grained locks



https://lwn.net/Articles/86859/ (May 2004)

2.6.6 still had about 500 lock_kernel() calls :-/

many more steps where needed and taken

https://lwn.net/Kernel/Index/#Big_kernel_lock

Linux finally got rid of the BKL in 2011

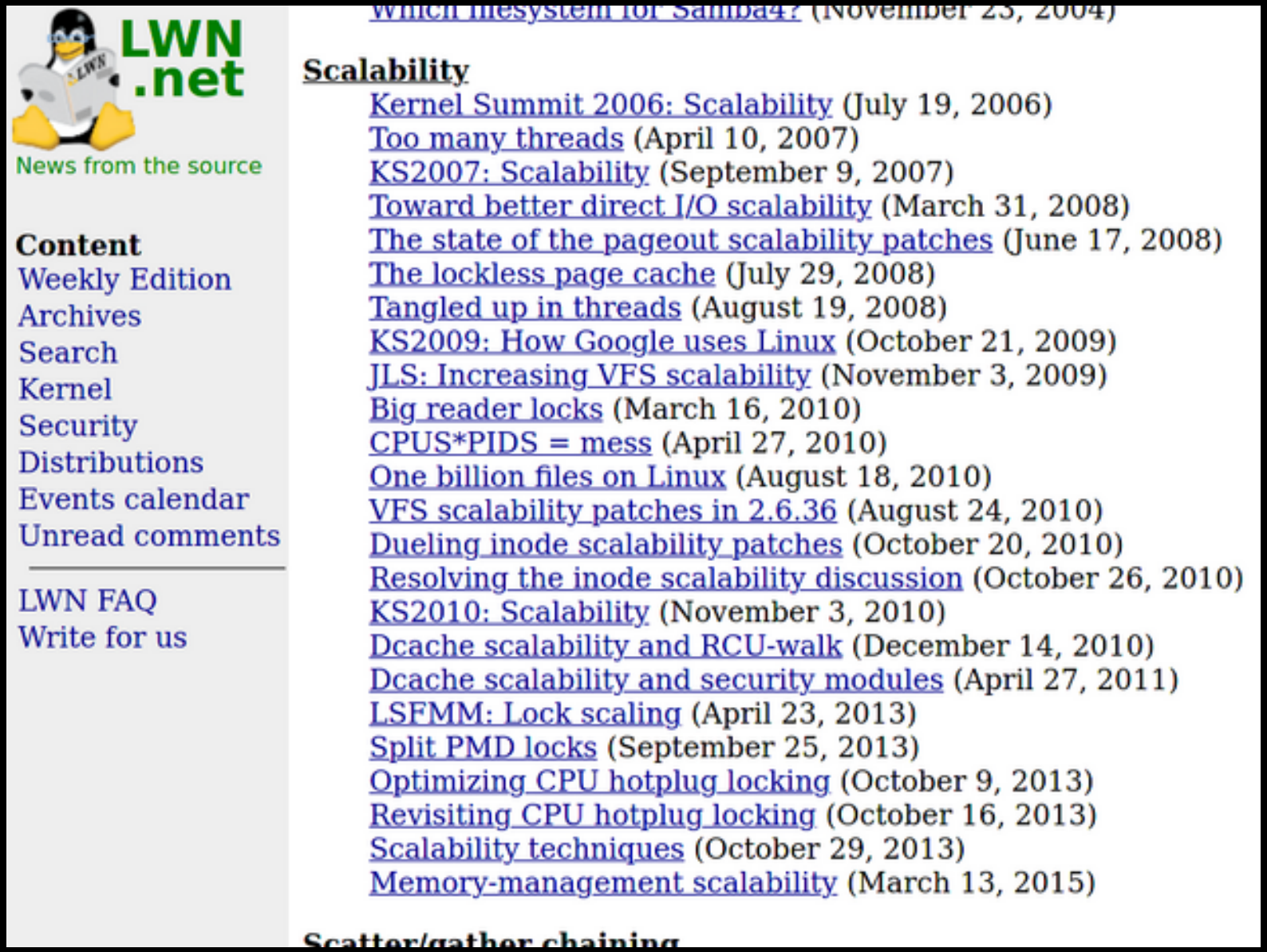
after about 15 years

thx to heroic efforts by various developers

esp. Arnd Bergmann, who took on the task of eliminating the BKL entirely!

the BKL might be history, but…

scalability is something still being worked on



https://lwn.net/Kernel/Index/#Scalability

many small improvements over time

never ending story

quite a few mm optimizations lately

new scheduler load balancing core in Linux 5.5

scheduling for asymmetric systems got improved recently

most people do not notice any of this

mostly flies under the radar

thx to all these small steps

Linux is and stays
one of the best scaling
OS kernels

= being a good host =

getting rid of the BKL was one of the first big achievements

reached in many small steps

something everybody worked towards

not always like that

more often, there is some competition

which can lead to interesting results

something important today was absent in the early FOSDEM days:

builtin virtualization capabilities

in the mid 2000s:

virtualization with x86 Linux got famous

Xen (~2005) made it popular and x86 processors started getting virtualization capabilities (2006)

Xen looked like the obvious and fitting solution the Linux world

one that everyone seemed to agree on

only problem:

support for running as Host (Dom0) or Guest (DomU) was out-of-tree

and Xen was a Kernel underneath the Linux kernel

then suddenly, out of nowhere,
in Oct 2006:

KVM

merged already into 2.6.20
in Feb 2007

because it was so small

in the beginning compared to Xen

worse performance,
less features,
CPU support required

a toy?

KVM was quickly improved in small steps

various people and companies made it better and better

a we know today:

turned out to be
a game changer

used basically everywhere these days and made Linux rule the cloud

Xen still around

Dom0 and DomU support only merged in 3.0 days (2011!) and small when compared to KVM

why did KVM succeed?

some might say:

because it took Xensource too long to upstream their code

definitely a factor, but I doubt it would have changed much

the real reason: KVM had a better, more flexible, and future-proof design

built into Linux, not underneath it

reuse things already there

that suited Linux more and left it in control

which obviously is in the interest of Linux developers

that's why a lot of people were willing to help

which in the end resulted in a better solution

history lesson relevant today, as every now and then we have

similar situations like Xen vs KVM

DPDK (Data Plane Development Kit)

a technique to make network packages bypass the Linux kernel

Linux developers started to fight back

with the eXpress Data Path (XDP), whereupon the AF_XDP socket (XSK) builds

seems XDP & AF_XDP can mostly keep up with DPDK these days

likely more future proof

another similar situation

Asynchronous I/O (AIO)

common in the Windows world, unusual in Linux

these days

io_uring finally brings proper AIO to Linux

an answer to the SPDK

Storage Performance Development Kit – a I/O bypass technique that started to gain territory

https://twitter.com/mjpt777/status/1215209572681515008

just as KVM:

both XDP/AF_XDP and io_uring started small

and got and get improved in small steps

= hosting differently =

another thing Linux still lacked during the early days of FOSDEM

support for Containers

other Unixes supported them already

FreeBSD jails (1999),
Solaris Zones (2004)

Linux containers only became famous ~2014

so why did it take so long?

kernel simply lacked required features

impossible to build something like Jails or Zones easily & reliable

features got built,
one step at a time

took years…

some for exactly this use case

various namespaces (2002 - now)

some for nearly this use case

cgroups (2007)

(initially often used for Virtualization with KVM)

some for different use cases

capabilities (~2003),
seccomp (2005),
...

Docker combined features in a new, more attractive way

…and made Linux containers popular

these small steps thus in the end changed the computer world

funny detail:

LXC was designed to become the preferred container solution

Virtuozzo/OpenVZ became small; Linux-Vserver nearly forgotten

they came earlier, but used out-of-tree patches

LXC still around, but not as big as Docker

ChromeOS and Canonical use it

imagine for a moment

what if just one company had been working towards LXC?

might have been
a pretty bad return of
investment…

those things show companies

investing money into developing complex new features bears risks…

a problem for the kernel, but still

Linux, the OS, got a better and more flexible solution

thx to the small steps

as they lead to features that Docker could combine in new, attractive way

= unexpected, but welcomed surprise =

docker shows:

sometimes things surface nobody aimed for

thx to kernel improvements in small steps, that lead to individual features you can recombine in various ways

Linux recently started a trip into the unknown

since ~2014 and 3.15+

people improved the Berkeley Packet Filter

(BPF, these days often called Classic BPF/cBPF)

the in-kernel mini-VM

(like a Java VM,
not an emulated computer)

tcpdump relied on it to only get the packets it was interested in

for performance reasons

(copying everything over to userland first is way too much work…)

improved cBPF
got called eBPF

called BPF for short these says :@

faster and much more powerful VM

run small programs in kernel mode

20 years ago, this idea would likely have been shot down immediately

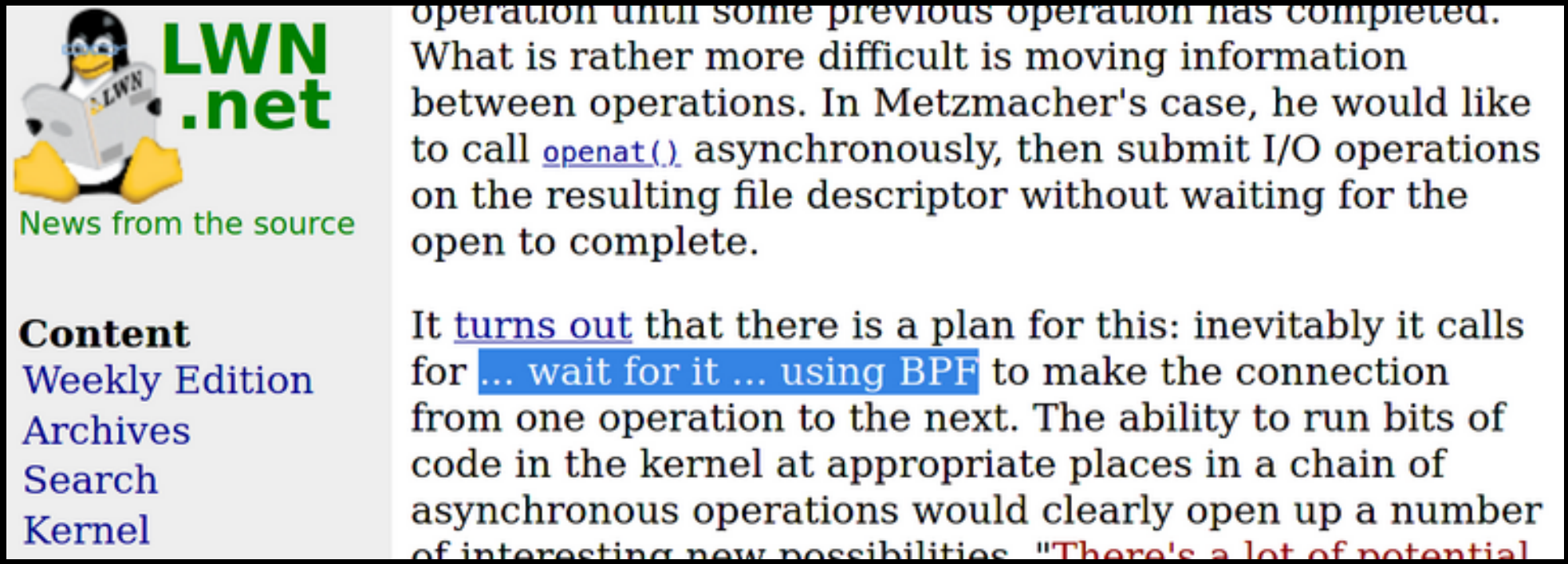
network devs scratched itches with eBPF

and improved it again and again

XDP & AF_XDP
build upon it

other kernel subsystems started to use it, too

and more and more will soon



https://lwn.net/Articles/810414/

eBPF still gets improved a lot with each new version

starts to change the kernel fundamentally

Linux gains more aspects of a microkernel

that's what Europe's biggest computer magazine wrote

the German c't magazine

c't magazin für computer technik
7.7.2018 15

Den optimalen Kopfhörer finden

Gratis-Bordmittel aktivieren und optimieren

Windows absichern

Virenschutz, Privatsphäre, Extra-Schutz für unterwegs

Kfz-Diagnose, Fahrtenbuch, Notruf, Hotspot

OBD2-Dongles: Nützliche Spione

Profi-Geräte mit 15 und 17 Zoll

Erste Notebooks mit Hexa-Core

IM TEST

- Turbo-LAN: NBase-T-Switches
- Lüfterloser Mini-PC mit Power
- Falt-Drohne Parrot Anafi
- Handwerker-Handy Cat S61

Desinfec't: Fotos und andere Daten retten

Webdienste per REST anzapfen

Flexible Heimautomation mit Node-Red

Fotos optimieren mit Gimp 2.10

Luxus-Boards für AMD Ryzen

Vollausstattung für Gamer, Übertakter und Kreative

Flexibler filtern

Neue Firewall-Technik für Linux bringt Elemente von Microkernels

Der Linux-Kernel erhält eine weitere Firewall-Technik, bei der zur Laufzeit maßgeschneiderter Code den Netzwerkverkehr filtert. Zum sicheren und abgeschirmten Erzeugen dieses Codes hat Linux eine Infrastruktur bekommen, die den Kernel erheblich verändern könnte.

Von Thorsten Leemhuis

Anzeige

Der im August erwartete Linux-Kernel 4.18 bringt erste Teile des Bpfilter, einer neuen Paket-Filter-Technik für Firewalls. Admins brauchen sich allerdings nicht um ihr Iptables- oder Nftables-Know-how zu sorgen: Bpfilter soll lediglich den Unterbau ersetzen, den das altbekannte Iptables und sein designierter Nachfolger nutzen. Das soll sie deutlich schneller machen. Noch liegt dieses Ziel aber in weiter Ferne, denn bei 4.18 wurden nur Teile des Fundaments gelegt. Das hat dem eher monolithischen Linux-Kernel ganz nebenbei zu einer Infrastruktur verholfen, die prinzipiell eine Modularisierung in der Art von Microkernels ermöglicht.

Genau passender Code

Der Clou von Bpfilter: Die Entscheidung über die Handhabung von Netzwerkpaketen erfolgt durch lokal erzeugten Programmcode, den der Kernel mit dem BPF ausführt. Dabei handelt es sich um eine prozessbasierte Virtual Machine, die allerdings simpler gestrickt ist als jene von .NET oder Java. Die VM ist ist auch als eBPF bekannt, da sie in den letzten Jahren aus dem Berkeley Packet Filter (BPF) hervorgegangen ist; mit ihm hat der aktuelle und bereits an vielen Stellen des Kernels genutzte BPF aber kaum noch etwas gemein.

Der BPF-Programmcode zum Filtern von Netzwerkpaketen mit dem Bpfilter wird für die jeweiligen Anforderungen maßgeschneidert. Das reduziert Verzweigungen im Code und verspricht, Overhead zu vermeiden. Damit der Prozessor den BPF-Code möglichst schnell ausführt, übersetzt der Kernel ihn auf gängigen Prozessor-Architekturen noch per Just-in-Time (JIT) in Maschinenbefehle. Darüber hinaus kann der Bpfilter den Netzwerkverkehr schon beim eXpress Data Path (XDP) abgreifen; der BPF-Code kann die Pakete dadurch schon kurz nach Empfang durch die Netzwerkhardware verarbeiten, bevor der mächtigere und daher trägere Netzwerkstack von Linux übernimmt.

Durch diese und andere Vorteile verspricht der neue Ansatz, effizienter zu arbeiten als die bislang von Iptables und Nftables genutzte Infrastruktur des Netfilter-Subsystems. Die ist zwar in C geschrieben, aber als universeller und flexibler Paket-Filter ausgelegt. Beim Abarbeiten des Regelsatzes muss der Code daher auch auf Eventualitäten gefasst sein, die lokal verwendete Regeln gar nicht nutzen. Das macht den Codepfad komplex, schließlich bietet der Netfilter immens viele Möglichkeiten, von denen Firewalls meist nur einen Bruchteil nutzen.

Userspace-Helferlein

Der Umstieg auf Bpfilter ist aber noch Zukunftsmusik, denn in 4.18 flossen lediglich Vorarbeiten ein. Sie schaffen eine Infrastruktur, mit der der Kernel den Filtercode lokal erzeugen soll. Das erfordert einen komplexen Übersetzungsvorgang, daher wollen die Entwickler die dazu nötigen Funktionen nicht im Code sehen, der mit Kernel-Rechten ausgeführt wird. Da es um eine sehr sicherheitskritische Kernel-Funktion geht, wollen sie die Aufgabe aber auch nicht an Werkzeuge delegieren, die unabhängig vom Kernel entwickelt werden und als normale Anwendungen laufen.

Die Programmierer haben daher für den Bpfilter einen Zwischenweg geschaffen: Erweiterung am User Mode Helper (UMH) und Module-Code, um den Kernel-Quellen beiliegende

Disclaimer: it was me who wrote that ;-)

Kernel regression tracking, part 2

By **Jonathan Corbet**
November 6, 2017

2017 Maintainers Summit

The tracking of kernel regressions was discussed at the 2017 Kernel Summit; the topic made a second appearance at the first-ever Maintainers Summit two days later. This session was partly a repeat of what came before for the benefit of those (including Linus Torvalds) who weren't at the first discussion, but some new ground was covered as well.

Thorsten Leemhuis started with a reprise of the Kernel Summit discussion, noting that he has been doing regression tracking for the last year and has found it to be rather harder than he had expected. The core of the problem, he said, is that nobody tells him anything about outstanding regressions or the progress that has been made in fixing them, forcing him to dig through the lists to discover that information on his own. He had, though, come to a few conclusions on how he wants to proceed.

First, he will try again to establish the use of special tags to identify regressions. His first attempt had failed to gain traction, but he agreed that he perhaps had not tried hard enough to publicize the scheme and get developers to use it. He will be looking into using the kernel Bugzilla again, even though it still seems like unpleasant work to him. He'll try to improve the documentation of how regressions should be tracked and handled. There is a plan to create a new mailing list on vger.kernel.org, with the idea that regression reports would be copied there. He will put more effort into poking maintainers about open regressions.

The discussion quickly turned to the problem (as seen by some) of the many kernel subsystems that do not use the kernel.org Bugzilla instance for tracking bugs. Peter Anvin said that many developers don't see much value in that system. Reported bugs tend to say something like "my laptop doesn't boot" with no further information; that tends not to be useful for the identification of any actual bugs. Beyond that, many bugs reported against the core kernel or x86 architecture turn out to be driver bugs in the end.

Users, it was suggested, should be explicitly directed to the mailing lists when reporting bugs for the subsystems that do not use Bugzilla. Laura Abbott said

Content
Weekly Edition
Archives
Search
Kernel
Security
Distributions
Events calendar
Unread comments

LWN FAQ
Write for us

Edition
Return to the Front page

others compared it to microkernels, too

Bpfilter (and user-mode blobs) for 4.18

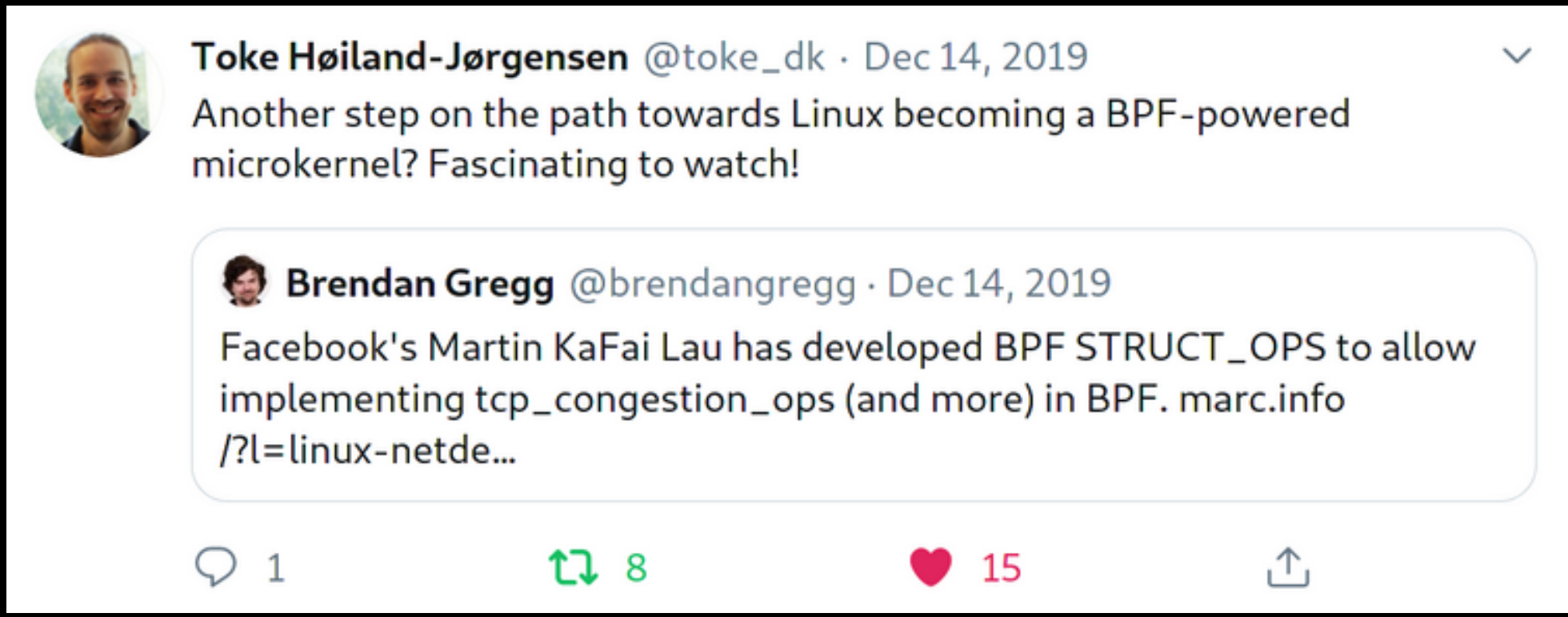
By **Jonathan Corbet**
May 30, 2018

In February, the bpfilter mechanism was first posted to the mailing lists. Bpfilter is meant to be a replacement for the current in-kernel firewall/packet-filtering code. It provides little functionality itself; instead, it creates a set of hooks that can run BPF programs to make the packet-filtering decisions. A version of that patch set has been merged into the net-next tree for 4.18. It will not be replacing any existing packet filters in its current form, but it does feature a significant change to one of its more controversial features: the new user-mode helper mechanism.

[...]

The replacement of netfilter, even if it happens as expected, will take years to play out, but we may see a number of interesting uses of the new user-mode helper mechanism before then. The kernel has just gained a way to easily sandbox code that is carrying out complex tasks and which does not need to be running in a privileged mode; it doesn't take much effort to think of other settings where this ability could be used to isolate scary code. Just be careful not to call the result a "microkernel" or people might get upset.

https://twitter.com/srostedt/status/1177147373283418112



https://twitter.com/toke_dk/status/1205824686426378240

maybe the beginning or middle of a small revolution

makes Linux more error-resistant, flexible, and powerful

and most people don't
notice anything

happening in
a lot of small steps

= longstanding wishes =

another area where Linux was behind

from the early FOSDEM days
until recently

a proper tracing solution similar to DTrace

published 2005, built for Solaris

Linux finally got something better quite recently:

BCC and bpftrace

www.brendangregg.com/blog/2018-10-08/dtrace-for-linux-2018.html

called "DTrace 2.0" by Brendan Gregg

"one of the leading experts on DTrace" (Wikipedia)

BCC and bpftrace can do more than DTrace

pretty cool, see Brendan website, his talks, or his book

www.brendangregg.com

BPF Performance Tools (book)

This is the official site for the book BPF Performance Tools: Linux System and Application Observability, published by Addison Wesley (2019). This book can help you get the most out of your systems and applications, helping you improve performance, reduce costs, and solve software issues. Here I'll describe the book, link to related content, and list errata and updates.

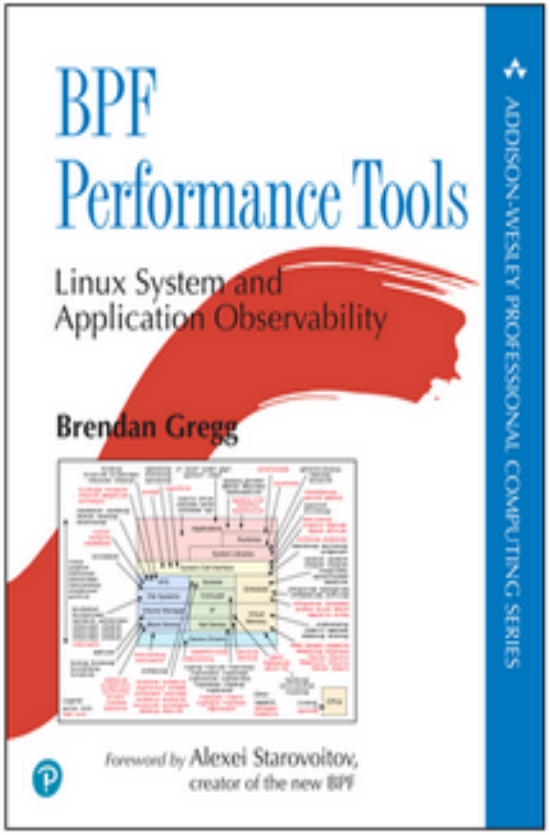
The book is available on Amazon.com (paperback, kindle), InformIT (paperback, PDF, etc), and Safari (here and here). The paper book was released in December 2019 **but sold out immediately**; more copies printed soon. ISBN-13: 9780136554820.

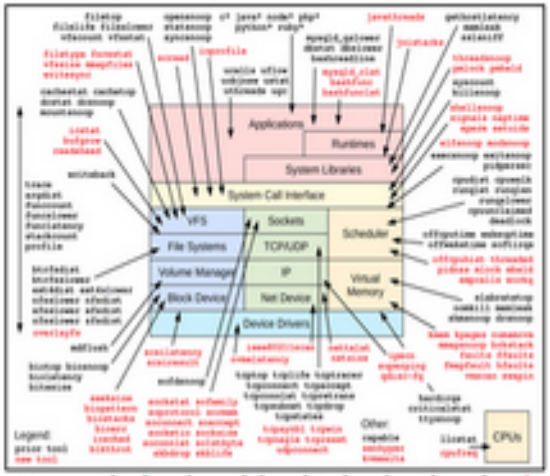
The Amazon Kindle preview shows the first 100 pages out of this 880 page book.

As an example new tool from the book, readahead.bt provides a new view of file system read ahead performance: the age of read-ahead pages when they are finally referenced, and unused read-ahead pages while tracing:

```
# readahead.bt
Attaching 5 probes...
^C
Readahead unused pages: 128

Readahead used page age (ms):
@age_ms:
[1]               2455 |@@@@@@@@@@@@@@@@                                    |
[2, 4)            8424 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@|
[4, 8)            4417 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@                       |
[8, 16)           7680 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ |
[16, 32)          4352 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@                       |
[32, 64)             0 |                                                    |
[64, 128)            0 |                                                    |
[128, 256)         384 |@@                                                  |
```





New tools developed for this book colored red.

The book covers many of the existing tools as well, for example, tcplife for efficiently logging TCP

just like containers:

took 10 to 15 years to build everything into the Linux kernel

the cool thing:

happened without a design that had exactly BCC or bpftrace in mind

they emerged thx to evolution

various building blocks got developed in the past 10 to 15 years

with smaller goals

perf, ftrace, tracepoints, kprobes, uprobes, kretprobes, uprobes, ...

features someone developed to scratch a specific itch

those are one part
of the solution;
the other:

eBPF ;-)

eBPF and tracing/perf tools got combined

and people developed BCC and bpftrace

and "ta ta", finally, after many years
and many small steps

Linux got a DTrace 2.0

15 years after people called for it…

= something impossible =

Linux soon will offer an important new feature

one almost nobody would have expected in the early FOSDEM days

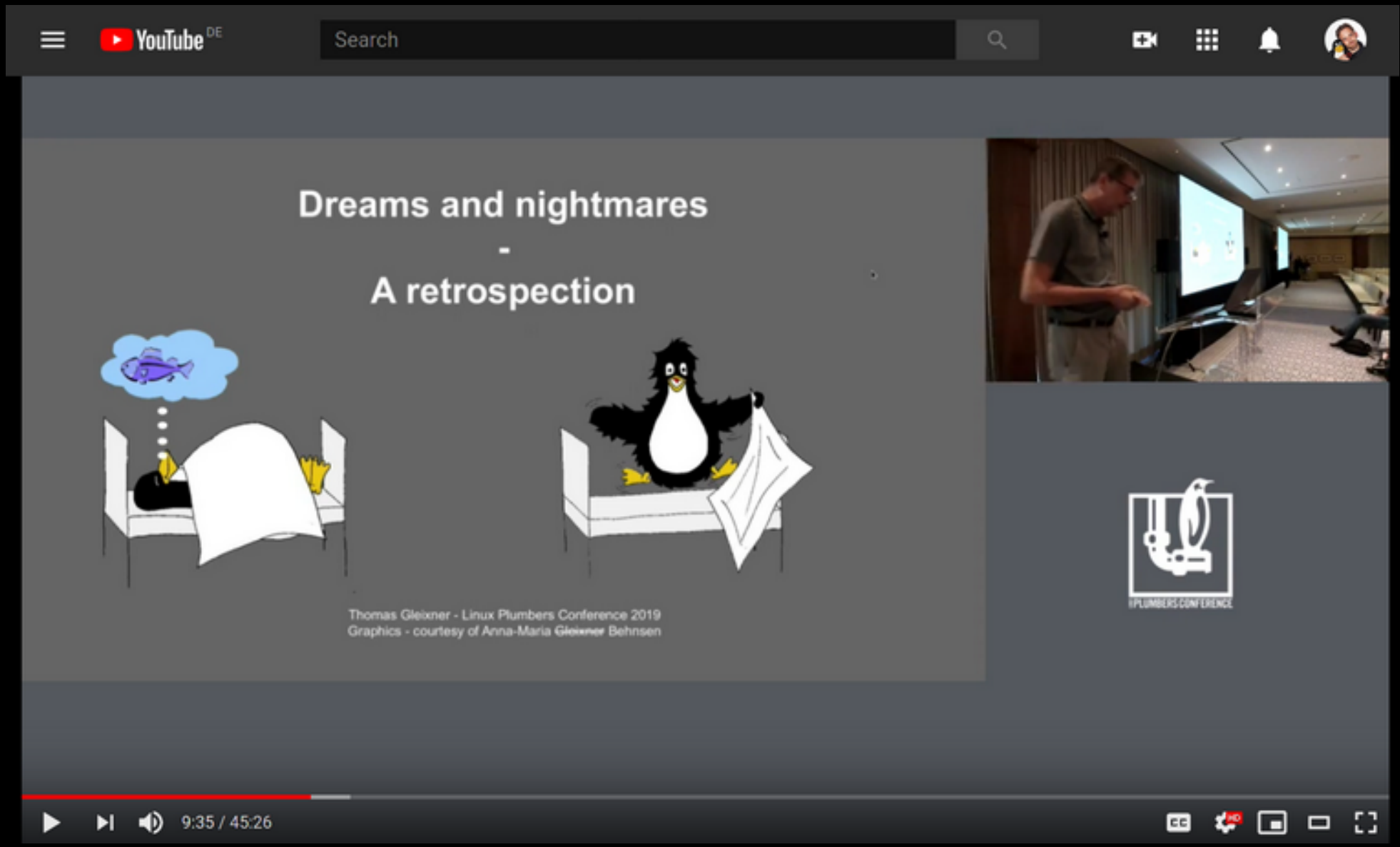
realtime capabilities

control your Laser cutter with Linux

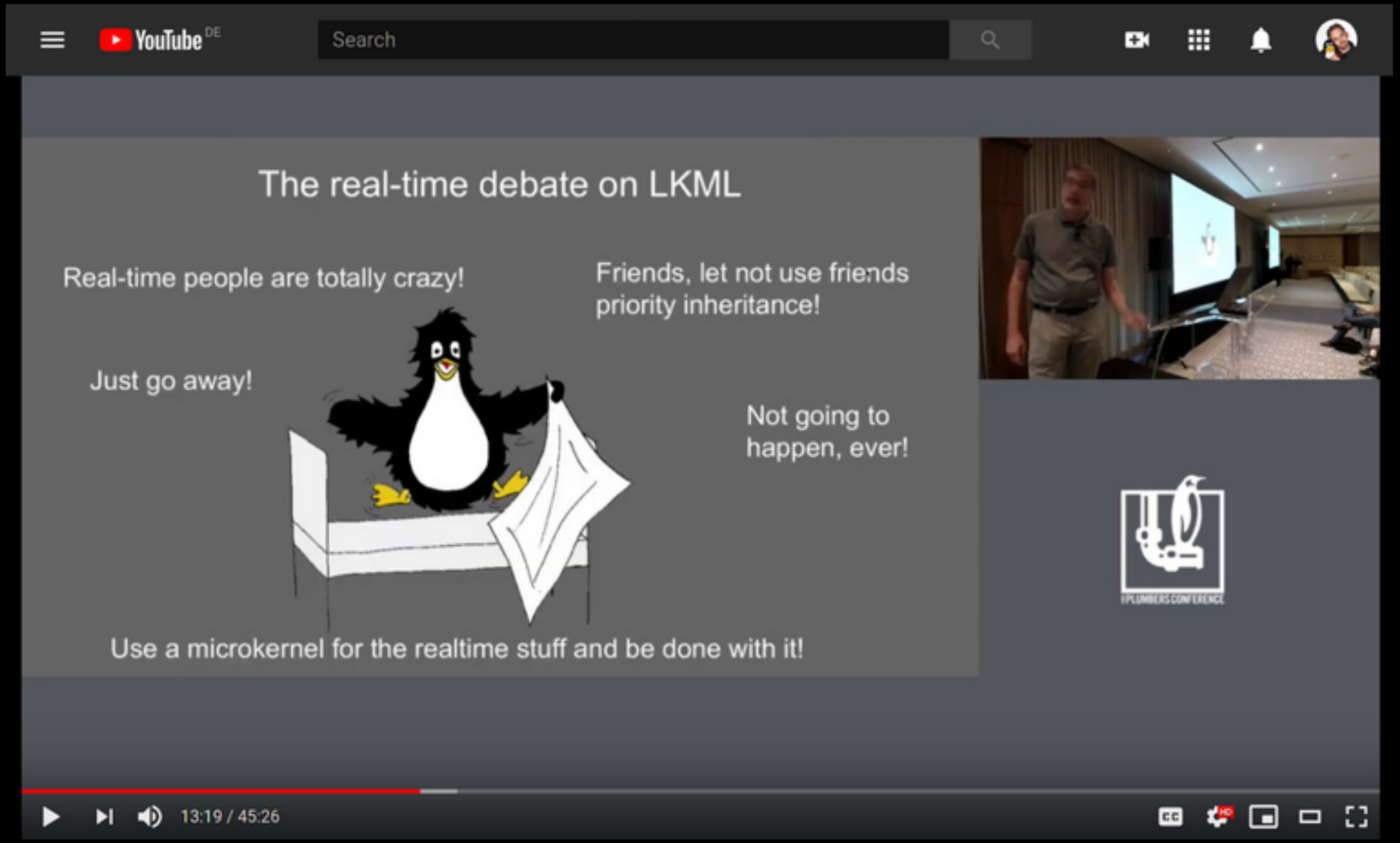
reminder: Realtime is primary about predictability, not performance

very vague and kinda crazy idea back then

by a few people



https://youtu.be/BTak9U6vuc0?t=512



https://youtu.be/BTak9U6vuc0?t=799

still

the developers behind the idea didn't give up

worked towards realizing the idea ever since in small steps

they made Linux
better for all of us

realtime systems hit many problems and scalability issues first

RT developers had lots of body blows

one of the worst afaics:

after going 90 to 95% of the route, they needed money for the rest

most of those that used RT patches didn't help much with development

luckily, the RT people were successful

Linux Foundation helped and founded a project

2015

soon the main trip will finally be finished

CONFIG_PREEMPT_RT already in mainline

but not exposed yet!

main thing missing

a printk() rework

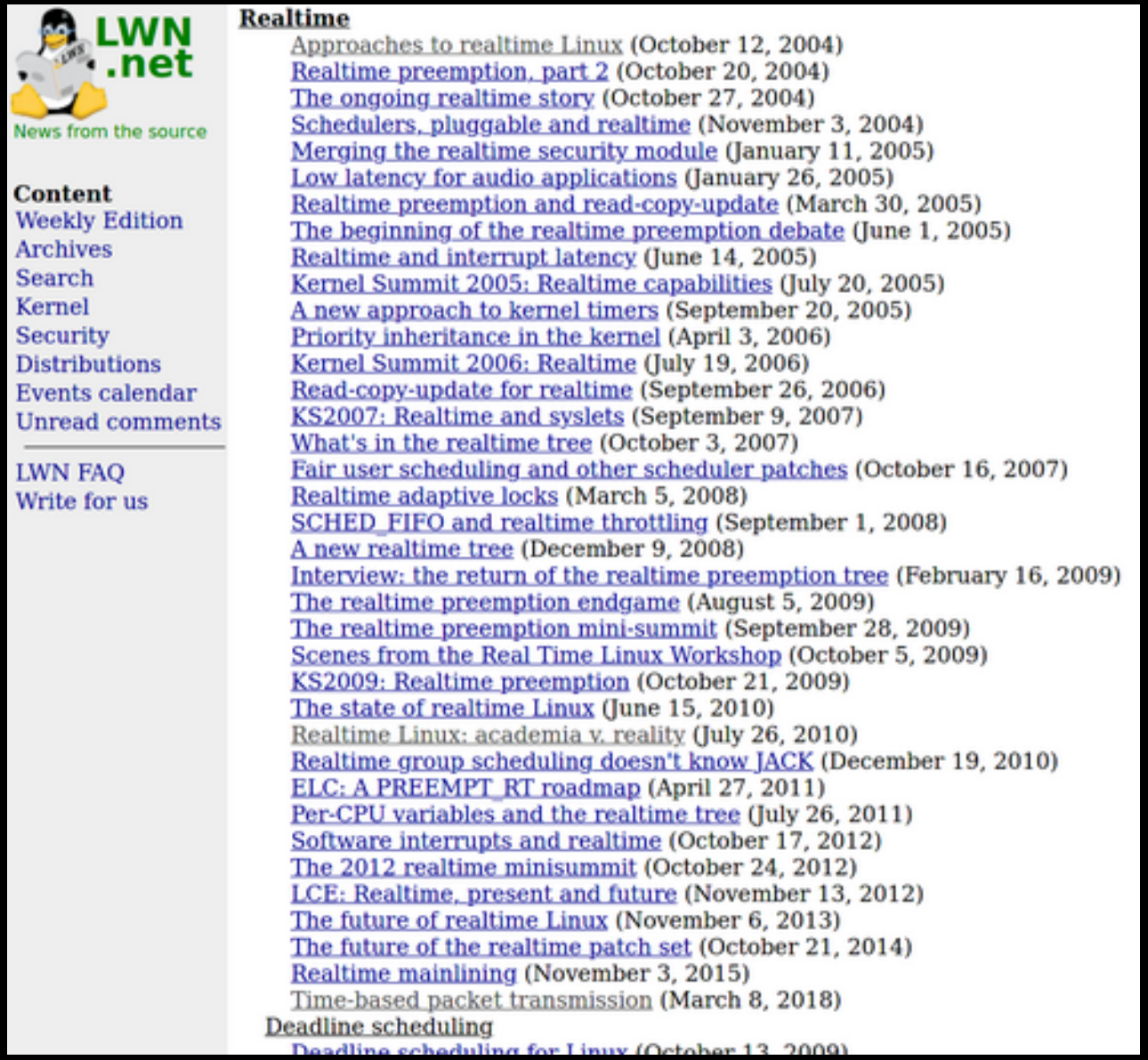
https://lwn.net/Articles/800946/

differences got settled recently,
just need to be implemented

looks like it will
be ready this year

realtime, for real, this year, too?

describing all the steps taken would fill hours



https://lwn.net/Kernel/Index/#Realtime

shows:

crazy goals that look unreachable can be achieved in small steps

that's how most kernel big features evolve

as new kernel features often are not designed by some company

often it are individuals
that want to realize
an idea or a dream

they might have to (ab)use companies to realize their ideas

or find money
in other places

but with a good idea and commitment big & crazy dreams can be realized

= working differently =

containers, bpftrace, realtime, …

Linux learned a lot since the early FOSDEM days

it took quite long to get those features realized

that's just how the Linux world is

you can't just hire
~50 developers

and make them build a feature you want in two or three years

like Sun could for Zones, DTrace or ZFS

bears costly risks

Linux developers might reject the outcome

they want to see small incremental, steps

which take more work, time, and might have a bad return of investment

served them very well

as often lead to one of the best or the best solution on the market

but it has
disadvantages, too

political and licensing issues aside

Is ZFS (2005) the most sophisticated filesystem in the *nix world?

hands up if, you agree!

work on "ZFS for Linux" already started in ~2008

Btrfs

but hasn't reached that goal yet

doesn't look like it will become a Linux-ZFS anytime soon

- Ability to handle swap files and swap partitions

Implemented but not recommended for production use [edit]

- Hierarchical per-subvolume quotas[45]
- RAID 5, RAID 6[46]

Planned but not yet implemented [edit]

- In-band data deduplication[32]
- Online filesystem check[47]
- RAID with up to six parity devices, surpassing the reliability of RAID 5 and RAID 6[48]
- Object-level RAID 0, RAID 1, and RAID 10
- Encryption[8][49]

In 2009, Btrfs was expected to offer a feature set comparable to ZFS, developed by Sun Microsystems.[50] After Oracle's acquisition of Sun in 2009, Mason and Oracle decided to continue with Btrfs development.[51]

Cloning [edit]

Btrfs provides a *clone* operation that atomically creates a copy-on-write snapshot of a file. Such cloned files are

https://en.wikipedia.org/wiki/Btrfs#Implemented_but_not_recommended_for_production_use
see also: https://btrfs.wiki.kernel.org/index.php/Status

so what went wrong?

one thing for sure

it was overhyped

still needed a lot of improvements
after the groundwork was done

and that as always, was…

done in small steps that took (and take) a lot of time

shows

how quick things improve mainly depends on…

(1) how complex the problem is and

(2) how many individuals or companies back development

turned out:

problem scope is really complex…

and companies did not care too much

some companies helped quite a bit

Oracle, Suse, Facebook, and a few others

but some didn't help much or at all

(no complaint)

big question

will Linux get something to compete with ZFS?

I'm pretty sure:

sooner or later it will!

it might just take 10 more years…

will it be bcachefs?

a lot of people have high expectation

I'd say:

wait and see

and keep your expectations under control

history shows:

it's a hard problem that takes a lot of effort

bcachefs right now is nearly a one-man show and not even submitted to upstream inclusion yet...

unlikely to fly soon

will take many years, even if big companies would start to back it

= lifestyle =

before coming to an end, let's switch gears

stop talking about features and look how the Linux kernel is developed

during the early FOSDEM days

Linux kernel development looked odd to outsiders

no central
development forge

like sourceforge, gitlab or github

development driven
by mail

Dozens of mailing lists

no tracker for patch submissions

quite a few fall through the cracks

no central issue tracker

for neither developers nor users

long unstable development phases

new features lingered in unstable tree for long

no predictable release cadence

no driver database

no way to easily look up if Linux contains a driver for your particular hardware and see what features it supports

we had a overworked lead developer

one reason for that:

we did not even have a version control system (VCS)

there were more
odd aspects

the kernel development model improved somewhat since then

after a short bitkeeper journey

we got git in 2005!

changed the world for the better;
thanks Linus!

unstable/stable model left behind

we got a mostly predictable release cycle (2005/~2.6.13)

new releases every 9 or 10 weeks

a lot called it crazy back then, but

turned out very well!

browsers picked scheme up

we also got Stable and Longterm kernels

~2005: 2.6.11.y, 2.6.16.y

but to be honest

many of the other odd things are still around

some even got worse…

we now have hundreds of mailing lists

instead of a few Dozen

there is a bugzilla, which
a lot of developer
do not look at at all

hint: official place to report a bug in most cases is a mailing list!

security became much more important, but we

still have no automated code checking in a central place

a lot of room for improvements here

switch to a central forge like gitlab or github?

could be a major step forward, as this brings CI, issue tracker, code review, and many more things

but no, that won't happen anytime soon

just as with features:

developers demand small steps here, too

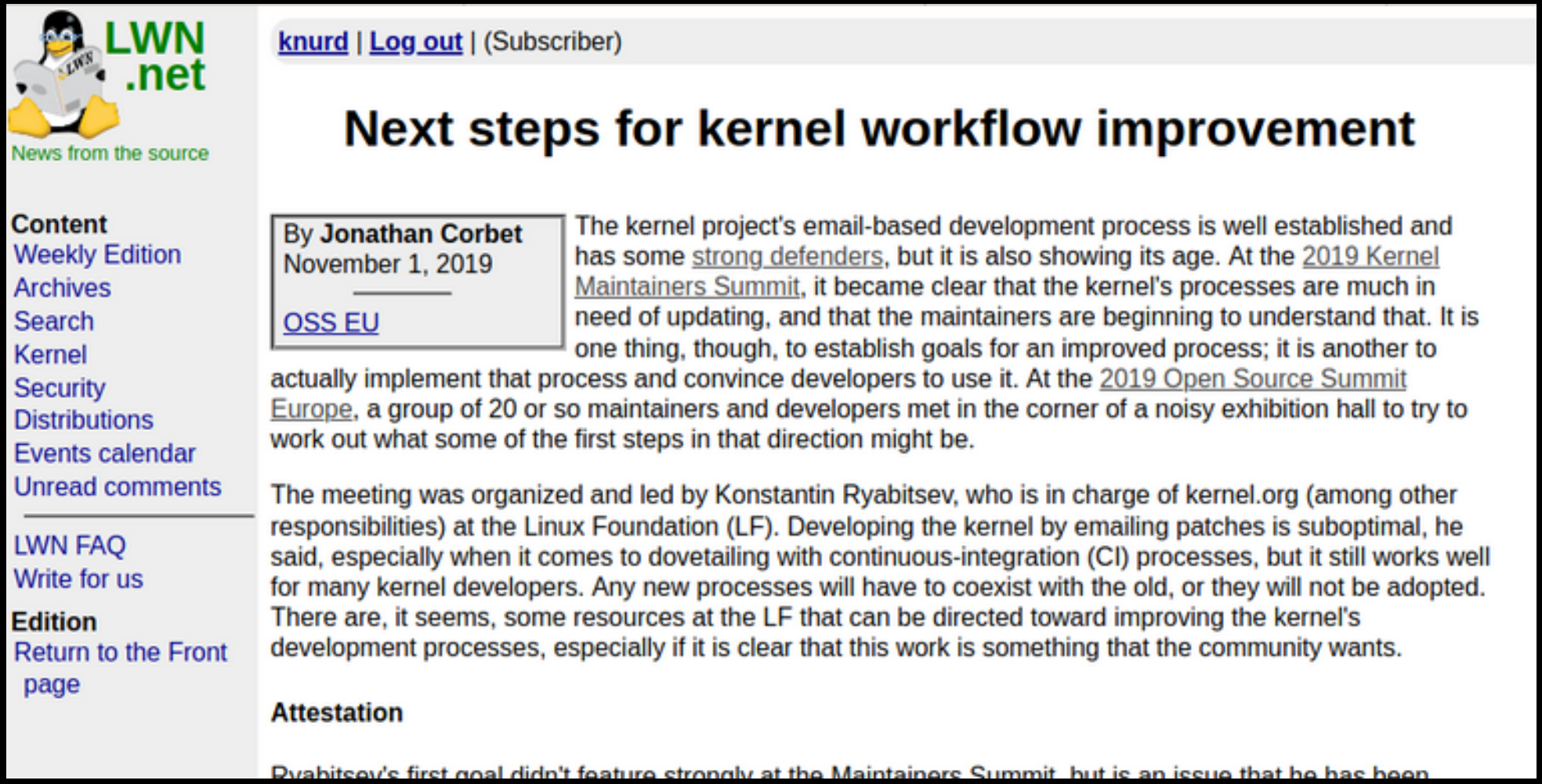
needs someone motivated enough to drive small, boring things forward

without an immediate return of investment

as that's why quite a few things are still kinda archaic

which becomes more and more of a problem…

lwn.net/Articles/799134/ (links to slides)
www.youtube.com/watch?v=iAfrrNdl2f4



https://lwn.net/Articles/803619/

https://twitter.com/dvyukov/status/1220410272755671043

just like with features

small steps are taken

and it will take time; you can help!

should the Linux Foundation help more?

not sure about that

Linux developers likely would prefer not to be governed like OpenStack or Kubernetes are

nevertheless

Linux development meanwhile runs at the usual pace

a new kernel version
every 9 or 10 weeks

for many years now

each with ~13.500
commits these days

diffstat:

bringing round about +650.000 insertions and -350.000 deletions

growth: ~1,5 million lines per year

about 15 years after Andrew Morton wrote:

(who back then was #2 in the hierarchy)

From: Andrew Morton <akpm@osdl.org>
To: ebiederm@xmission.com (Eric W. Biederman)
Cc: torvalds@osdl.org, pavel@suse.cz, len.brown@intel.com,
drzeus-list@drzeus.cx, acpi-devel@lists.sourceforge.net,
ncunningham@cyclades.com, masouds@masoud.ir,
linux-kernel@vger.kernel.org
Subject: Re: [PATCH 2/2] suspend: Cleanup calling of power off methods.
Date: Wed, 21 Sep 2005 11:24:48 -0700
Message-ID: <20050921112448.0e121a3d.akpm@osdl.org> (raw)
In-Reply-To: <m1ll1qcmzr.fsf@ebiederm.dsl.xmission.com>

ebiederm@xmission.com (Eric W. Biederman) wrote:
>
> > Famous last words, but the actual patch volume _has_ to drop off one day.
> > In fact there doesn't seem to much happening out there wrt 2.6.15.
>
> Due to changes coming through git or that there will simply be fewer

From: Andrew Morton <akpm@osdl.org>
To: ebiederm@xmission.com (Eric W. Biederman)
Cc: torvalds@osdl.org, pavel@suse.cz, len.brown@intel.com,
drzeus-list@drzeus.cx, acpi-devel@lists.sourceforge.net,
ncunningham@cyclades.com, masouds@masoud.ir,
linux-kernel@vger.kernel.org
Subject: Re: [PATCH 2/2] suspend: Cleanup calling of power off methods.
Date: Wed, 21 Sep 2005 11:24:48 -0700
Message-ID: <20050921112448.0e121a3d.akpm@osdl.org> (raw)
In-Reply-To: <m1ll1qcmzr.fsf@ebiederm.dsl.xmission.com>

ebiederm@xmission.com (Eric W. Biederman) wrote:
>
> > Famous last words, but the actual patch volume _has_ to drop off one day.
> > In fact there doesn't seem to much happening out there wrt 2.6.15.
>
> Due to changes coming through git or that there will simply be fewer
> things that need to be patched?

We're at -rc2 and I only have only maybe 100 patches tagged for 2.6.15 at
this time. The number of actual major features lined up for 2.6.15 looks
relatively small too.

As I said, famous last words. But we have to finish this thing one day ;)

> As for 2.6.15 I know I have patches in the queue that I intend to send

= summing things up =

Linux developers
solve big problems
in small steps

#bigkernellock

small steps lead to better and more flexible solutions

#kvm vs #xen

sometimes make new, groundbreaking technologies possible

#docker

building blocks build in small steps can even help fulfilling old wishes

#DTrace_2.0

process can lead to quite unexpected, disrupting results

#bpf (keep an eye on it!)

that's what made and makes Linux so great

reaching big goals with small steps takes time

and thus money

they thus need someone really committed

ideally and individual that wants to realize a dream

that worked great
in a lot of areas

#realtime – but also #BKL, #KVM, #DTace_2.0, #BPF, …

in some areas, we are not there yet :-/

to improve things,
become an individual
that is committed

and find money to get
the dream realized

then Linux will get a filesystem even better than ZFS

and developer tools and schemes even better than what we have

or other things that will have a positive impact on the world

like Linux and Git
had and have

which once
were just a dream
in somebody's head

that's it – questions?

(TWIMC: this is slide #234)

feedback

please provide feedback

feedback welcomed, even if negative; talk to me!

mail: linux@leemhuis.info, thl@ct.de
GPG Key: 0x72B6E6EF4C583D2D

social media: @kernellogger, @knurd42 on #twitter & #friendica

4 more social media accounts, see www.leemhuis.info/me/

#EOF