Live blog day 1: morning

Sep 25, 2023

Let's start this 10th edition of Kernel Recipes !

This is the anniversary edition of Kernel Recipes, 12 years after the first edition.

David Miller is the godfather of this edition.

This is the live stream from those who prefer video over text.

A quarter century of Linux open source – Jens Axboe's personal journey

In this non-technical talk, Jens explains how he got his start in Open Source, and Linux in particular.

In 1993, Jens discovered slackware in article, and downloaded it from a Swedish BBS, exploding the home Internet phone bill. He played with it Linux, and found the coolest thing was the kernel part, and started following LKML very closely.

After seeing the cdrom maintainer looking for a replacement, he offered help, and that's how he became a maintainer. He learned a lot in the process, from optical media burning, to IDE and then the block layer.

The responsibilities of the Kernel Maintainer are numerous: from setting the general direction of the subsystem, reviewing changes, doing stable backports, and many others. One of those is dealing with bug reports.

Many bug reports include zero details, and that is very bad. But what is a good report ? A good bug report must include specifics on versions, what broke, relevant details, and the steps to reproduce it. Even better: include a fixing patch. A git bisect is also very helpful.

The kernel rate of change is one release every two months. That's both fast and slow: while a patch can reach the upstream in 3 months, it might take a year or more to reach the user distributions.

As maintainer, it's easy to have fatigue. There is no "off switch": emails arrive every day, from all time zones; security issues need to be fixed during the weekend.

A solution to this is to find new people to help, and share the workload as group. But, Jens says, it's very hard to find the right people.

A new contributor needs to understand how to use the email infrastructure the kernel relies on: tools like b4 and lei help; `git send-email` is good for sending patches.

When upstreaming, it's good to do something good for the project as a preparation: not taking only your needs into account. If nobody pays attention to the patch, a trick is to "half-ass" the change, which attracts many reviewers. But it's a difficult balance to strike.

Sometimes there might issues when submitting complex patchsets. A trick to work around that is to have multiple smaller series. When sending new versions of those patchsets, always wait until reviews on previous versions have calmed down.

Before submitting a patch, or even a new version of patch, always verify that it compiles, with no new warnings, and test it. A good commit message should explain the why; if it's a fix explain what is broken, and what are the consequences of the patch. It should explain how the patch was tested. Many of those things should be doubled checked.

The Arm laptop project – Johan Hovold

This talk about Linaro's Arm laptop project, on which Johan has been working for the past year and half. The goal is to run Linux on Windows On Arm Qualcomm laptops.

These laptop have a non-standard boot process, and previous work to get Linux to boot on them was already done.

Recently, the third-generation Lenovo ThinkPad X13s changed the game by providing a much more powerful platform to work on.

After boot EFI runtime services won't be available. The custom ACPI implementation also made things more complicated.

The process make them work with mainline Linux included multiple phases, from reverse engineering, refactoring, adding features, reviewing patches, etc.

Johan worked with its own regularly rebased tree, and caught a few regressions upstream in the process.

For device firmware, the initial work used the Windows firmware, with work done in the background to have Lenovo and Qualcomm release redistributable firmware.

Over the past 5 kernel releases, many features were added. There are many that are still a work in progress: Camera, EFI variables, Fingerprint reader, Video acceleration, etc.

The power consumption situation is still being improved, but it's already at 15h of idle time, and 29h of suspend time.

GPU support has been worked on by Rob Clark, the freedreno maintainer. The support is already in Mesa and the kernel; but it needs an extracted GMU firmware which is not yet in linux-firmware.

Bluetooth support was added, but the device address needs to be set manually, since it is stored externally, and this part still needs to be reverse engineered.

The touchpad on the laptop is dual-sourced, so there might be different controllers; the device tree should be updated by the boot firmware to tell which is present. Some other devices, like the touchscreen, are optionally present. Those devices can't be all in the device tree, because of the way device probing works: they might be probed in parallel, which induces mapping races in the IRQ core (this was fixed in 6.4). Shared resources (like a reset GPIO) might be claimed temporarily; this caused an interrupt regression in 6.6-rc1. Pin configuration might be claimed before HID driver probes; so those might have to be moved to the parent bus node.

To fix all this, Johan thinks the devicetree specification should be extended to be able to mark devices as mutually exclusive, and even force sequential probing.

Other issues might arise when doing handover from the bootloader to the kernel, or from the early kernel, to late kernel if drivers are built as modules.

For future work, Johan says that support for the Camera ISP can only be done by Qualcomm. Same for virtualization, which needs early firmware fixes. Other areas planned include Bluetooth MAC address (BD_ADDR), Thermal throttling, special keyboard keys, TPM and USB PD.

It's possible to get started with the X13s today: it needs a recent kernel (6.5+), up-to-date linux-firmware and alsa-ucm-conf, as well as two qualcomm userspace daemons. There are a few guides for available Fedora, Ubuntu or Debian. Support is good enough for everyday use of a few kernel maintainers, Johan says.

The maintainer's POV – Borislav Petkov

Borislav wants to share his 10 year experience as maintainer in x86 land; he was once wondering why maintainers were so grumpy — now he knows.

The development process, he says is like a fence; you have stuff flying over the fence. Patches from submitters, most employed by businesses interested in getting code upstream. There are users and testers; unfortunately not enough testers.

Code submitters send code without necessarily helping with other tasks — that falls to the maintainer. Many people want to rush stuff, which leads to bad quality; things shouldn't be rushed, but done properly: test properly, address feedback, etc.

On the other side of the fence, there are the maintainers. They don't scale, so maintainer groups are a better solution, but scaling them is still hard. Maintainers need to handle bug fixes, code integration, reviews, their day job, and the new thing: embargoed hardware issues. Those are done behind closed doors, and often break things because they can't benefit from the same testing. The last ones even broke clang builds for example.

New hardware features often need to be added in the kernel as well, and might not be well-thought from a software point of view.

Maintainers aren't born grumpy, they become grumpy. The fix for that, according to Borislav, is to make everyone a review and maintainer. If everyone participates in reviews, and enhancement of the source code maintainability. The goal is to have everyone think about the big picture.

After sending patches, one should start reviewing other patches while waiting for their own review, Borislav says. Getting a code review is priceless. It requires thinking as maintainer: does the code make

sense? would I accept it ? and would I maintain it for many years ?

Why are maintainers nitpicking about many small details ? There are many reasons. Good, concise and imperative commit titles save a loot of time when looking at a patch queue.

The commit messages should aim for clarity, and have all the necessary context for anyone to be able to understand what's happening. Even yourself in a few years from now. This helps because maintainers are always doing git archeology. For example, when pondering if code can be removed, having the reason why it was added can help making a decision.

Structuring a good commit message is not rocket science, and Borislav shared a simple method that can be applied; when in doubt, read the Linux kernel documentation on submitting patches.

Adding tags like `Fixes: <sha1-of-bug-introduction>` is very helpful, because it helps when backporting. The `Signed-off-By` chain is mandatory. `Link:` is a pointer to the mailing list discussion on lore, also very helpful for code archeology.

Continue on the afternoon live blog.

Live blog day 1: afternoon

Sep 25, 2023

Fast by Friday: Why kernel superpowers are essential – Brendan Gregg

What would it take to fix any performance issue in 5 days ?

Improving performance is good for the environment, companies, etc. Brendan's motto is that any performance issue reported on Monday should be solved by Friday. By solved, it means that the root cause is identified. It's both a vision and a way of thinking.

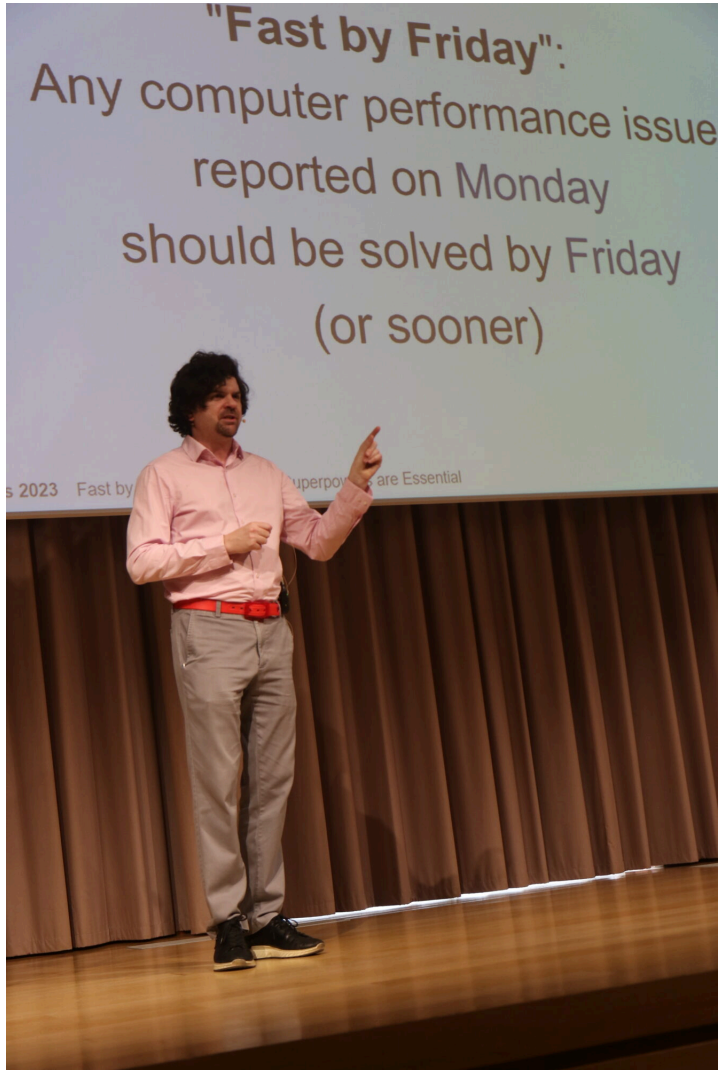
The focus here is on finding the performance issue root cause. And for many people, this is the hardest part, and what matters.

Often, working on performance is an issue of time. Performance regressions aren't tracked down, and hardware and software are getting more and more complex.

A common scenario at product vendors is that there's often a tunable or configuration that might prevent the product from being the fastest.

The first step has to be done in the prior weeks: Preparation. Everything must work and be already installed. Performance "crisis tools" should already be installed (sysperf, bpftrace, systat, ftrace etc.), and stacktraces should work.

On Monday, the, the problem should be quantified, using the problem statement method. The problem should be understood by the end of day.



On Tuesday, checklists should help eliminate subsystems where the issue is **not** coming from. Most new observability tools need kernel superpowers, like ftrace and eBPF. eBPF tools are very helpful; future eBPF tools should live in the kernel tree itself, and be reviewed and ideally written by the developers themselves. Brendan gave an example for a ZFS L2ARC health tool, which he wrote. A health tool doesn't have to be done by the developers, but at least sharing a document explaining how one would assess the health can let others use that as a guide to write one.

On Wednesday, profiling. CPU flame graphs, CPI flame graphs and Off-cpu analysis.

On Thursday, Latency analysis, logs (events...) and hardware counters with perf.

Finally, on Friday: efficiency and algorithms. It's the hardest part, Brendan says. Checking the Big O notation of the algorithms. From the audience, Matthew Wilcox says that even the Big O notation is not sufficient: one should look at constant overhead, and cache performance for example.

If on Friday, the performance issue isn't root-caused, it might be time to move-on, and accept the code is efficient enough.

Installing the crisis tools by default is the easiest part. Stack walking should also work by default. Should the frame pointers be disabled for performance reasons ? Brendan says, there's a probably middle ground. Steven Rostedt in the audience says that s-frames can also help. For higher level languages like Java, the stack walkers should be shipped as open source with the runtime code.

Without eBPF, Fast by Friday would not be possible, it would take longer.

io_uring meets network – Pavel Begunkov

io_uring is not only for storage; it also has support for networking; it supports sendmsg/recvmsg operations too. In io_uring, submissions and completions are asynchronous.

In the early days, this relied on a worker pool, which can be slow. This is why one might need to do polling instead, with the appropriate io_uring operation. It was first step for operations that wanted to convert an epoll workload to io_uring. The best option is to instead have a syscall batching system, otherwise, the app won't get the best performance from io_uring.

Pavel says, one should dig a little bit deeper int the application architecture in order to be able to use io_uring. As a first step, coupling sending the request and polling is simple, and makes for a simple api. io_uring suports MSG_WAITALL to workaround shourt reads or writes.

The memory consumption can quickly grows because slow connections might lock buffers and consume too much memory. A solution is to use provided buffers to have a kernel buffer pool. This way the buffers are only locked when the data becomes available.

The first version of buffer pools is unofficially deprecated for performance reasons; the second one is recommend one.

To improve polling, one can use multishot polling combined with multishot accept and recv with a buffer pool. With multishot requests can be cancelled, and can fail. The Completion Queue is finite, so having it overflow will create performance issues (allocations).

Fixed files optimize per request file refcounting; it mostly makes sense for send requests.

Close can be done with io_uring as well. But for in-flight requests, it is recommended to use shutdown first to ensure cleanup of resources before the recv is done.

Zercopy send is already available, and there are patches for multishot zerocopy recv.

Chosing the appropriate task run mode is important.

Using all those tips can help improve performance if assembled correctly.

Update on Landlock: Audit, Debugging and Metrics – Mickaël Salaun

Landlock has been in mainline since Linux 5.13, and is now in multiple Linux distributions. It's a solution for unprivileged processes sandboxing.

Landlock is dedicated to protect against untrusted applications and exploitable bugs in trusted applications.

Landlock allows putting restrictions on filesystem access. Mickael gave an example use case, where limiting the access of an application can limit the damage it can do in case of issues.

Landlock explicitely does not want to track access requests. Since it's a Linux Security Module (LSM); they can be stacked; and once an action is denied, following stacked LSM won't see any event.

But a Landlock wants its own denials to be logged, and the reason why. This helps app developers writing a sandboxing policy, power users, sysadmins, and security experts to detect attacks.

When there are multiple, dynamic, nested and unprivileged security policies, tracking denies in logs might be an issue. Logs need to track denied requests's domain hierarchy, follow the lifetime of rulesets.

Those logs should not be available to unprivileged users. The audit framework is the perfect tool for this.

Mickaël did a demo with sandboxer, a tool from the linux kernel samples directory, showing how denies would appear in the audit log.

In an upcoming patch series, new syscall flags will be added to opt-in or opt-out of logging, and to have permissive mode to be able to log all actions that would be denied.

In the future, a new filesystem to get a view of Landlock domain information could be added, once a few challenges are overcome, like unique and global IDs that currently leak information about layering. Checkpoint-Restore In Userspace (CRIU, a snapshotting solution) support will be worked on in the future.

On the roadmap, there are new access-control types for ioctl and tcp.

Linux and gaming: the road to performance – Andre Almeida

Linux has no roadmap, Andre says. It is driven by users. Different companies are pushing for different things. Cloud providers might care about storage performance, Android about responsiveness, and Embedded about resource usage. The result in a kernel that is versatile.

Gaming can bring other things to the table.

Most games were developed for Windows; Valve changed the landscape as it brought Steam to Linux, invested in Proton (Wine), and shipped a Linux portable game console: Steam Deck. Valve sponsors the development of all the open source stack needed for gaming, from the kernel to Mesa, and many others.

Proton is bundle of projects and patches, and relies mainly on Wine, which is a way to translate Win32 API calls to Linux.

Nowadays, the Linux kernel has seen a lot of gaming workloads, leading to a few patches, done by multiple companies (including igalia, Andre's employer).

The first controversial one was the addition of case-insensitive filesystems. This works with normalization at the lookup time. It was initially done for ext4, then reused for F2FS. There is work in progress for case-insensitiveness in tmpfs and bcachefs.

futex was the next one. A win32 API was used a lot for waiting on multiple locks. futex2 came out of this, as a completely new API, addressing this issue, but many other issues like NUMA-awareness.

Some games might use syscall directly in assembly, bypassing the Win32 API. To emulate those, userspace syscall dispatch was added. It took the form of a new prctl, and is even now used by CRIU as well.

For VR gaming, many input peripherals might generate a lot of events in parallel. So HID throughput was a bottleneck with a single mutex used to protect the HID device table. This was replaced by a rwlock, leading to a 4x perf improvement.

Split-lock across cachelines are possible on x86, and might be used to DoS the system, so Linux added a delay to mitigate potential attacks. But games might use this… so a new sysctl option was added to disable the mitigation.

A WIP to improve performance are adaptative spinlocks, which would allow using spinlocks in userspace by relying on rseq (restartable sequence); the lock would only spin if the process is scheduled.

Pagemap scanning was added to help anti-cheat support that needs to know about recently modified memory pages; dirty page mechanism isn't as accurate as Windows'.

On Steam Deck, HDR work was done for amd gpu. On Steam Deck, all device drivers are upstreams, or being upstreamed.

GPUs are complex and can crash; there is no standard way on Linux to recover and report a reset to userspace.

That's it for today ! See you tomorrow for the day 2 live blog !

Live blog day 2: morning

Sep 26, 2023

Welcome back for day 2 of the Kernel Recipes live blog ! You can also follow the live stream.

The unique way to maintain Ftrace Linux subsystem – Steven Rostedt

This is the third talk about maintaining, but it's only an accident. Great minds think alike.

As an introduction, Steven said it's only his personal approach. He uses it because he thinks it's efficient, and doesn't like change for the sake of change, but only if it works.

Steven showed pictures and config of his personal server and workstation and office desk. His workflow starts on his workstation, where he does most of the development. Then the server manages email, web, inside dedicated VMs before going to a firewall.

Email clients aren't the best things for managing patches, Steven says. He might miss patches, and find them 6 months later. Most of the time, maintainers ignoring patches is not intentional, but probably just missing patches.

He setup a patchwork instance on his server. He showed his procmail config; some patches are ignored directly (if they don't Cc the mailing list), some go to the inbox, and others go to patchwork.

Steven also subscribes to the git-commt-head mailing list, which sends an email for every accepted commit that goes into Linus' tree. This is used to remove patches that have been merged from the database of pending changes.

The workflow to look at a patch starts from a patchwork instance; a link to a series is copied and passed to a script to download it on the server; then lore lkml links are added with a script. If applying the patch with `git am` fails, then Steven switches to `quilt`. The conflict resolution is done in emacs, with manual diff editing. Then the patch diff is double checked in git.

With git, he gets too many branches, and it makes it hard to find things. This is why he created a custom tool, `git-ls` to see all of them. But for small changes, he uses quilt. The patch files are generated by `git diff`, then managed in quilt.

In order to build and test the kernel, Steven uses `ktest.pl`, an in-tree tool to automate building, installing and testing a kernel. It uses a config files, and there are example config files, for example to use VMs as test targets.

To test, Steven uses two VMs, one 64 bits, and one 32 bits. To build both against the same repo, he uses `git worktree`. The test setup takes a few hours for each config; no tree is sent to Linus Torvalds without passing this testsuite. But Steven will often find bugs in other parts of the code preventing sending his tree. The configs and test suites are both public on Steven's github.

To prevent testing twice the same thing and wait a 13 hours for nothing, in Steven's config, ktest verifies the previously tested tags and fails if a full testsuite was already ran for the git commit.

Sending emails to pull `for-linus` and `for-next` branches, this is automated with scripts as well.

sched_ext: pluggable scheduling in the Linux kernel – David Vernet

A CPU scheduler is used to share tasks across CPUs. Things get complicated pretty quickly though, and many constraints that get added depending on the OS, tasks, CPUs, etc.

CFS (Completely Fair Scheduler), the default Linux scheduler aims to have a fair sharing policy. It's great, David says, but it was built in a simpler times: smaller CPUs, homogenous topologies, closer caches. Nowadays, the hardware is more complex with CCD's (Core Complex Dies), aggregating CCX's (Core CompleXes). Heterogeneity is becoming the normal, with CPUs that might have 4 cores per CCX, with 2 CCX per CCD; then different cache hierarchies, shared L3s, etc.

CFS has other some drawbacks as well: experimentation is difficult, because one needs to recompile and reboot for tests. On boarding a scheduler developer can take years, David says. Since the scheduler is for everyone, it's general approach makes it hard to please everyone at the same time.

sched_ext enables dynamically-loaded BPF programs to set the scheduler policy. It works as a different scheduling class, in parallel to CFS, but at a lower priority. The interface is not stable and considered kernel-only, and GPLv2-only.

With BPF, experimentation is much simpler: no reboot needed, impossible to crash the host (eBPF verifier rejects bad programs). The API is simpler to use too, David says. And for safety, a new sysrq key has been added to go back to the default scheduler.

At Meta, HHVM has seen up to +3% requests per second with a custom scheduler, and up to +10% for ads ranking. At Google, they also have seen strong results on search, VM scheduling and ghOSt.

One of the goals is to move complexity to userspace.

sched_ext does not aim to replace CFS, there is always going to be a need for a general scheduler. David hopes that quick experimentation will even help improve CFS. The GPLv2 constraint is checked at load time by the BPF verifier; some example BPF schedulers are provided.

Currently, a goal keep the sched_ext API unstable, and not impose UAPI-like constraints on the kernel scheduler.

In order to build a scheduler, there's a basic set of callback functions that the BPF scheduler programs can call. One of the building blocks for that are the Dispatch Queues (DSQs). Every core has its own DSQ, and they are similar to the kernel "runqueue".

David showed an example Global FIFO scheduler that is very simple, combining a global DSQ with local DSQs, yet works very well on single CCX machines.

The strategy to select a core is also relatively simple, even as it takes SMT into account.

Another scheduler examples used a Per-CCX DSQ, which improves L3 cache locality (same CCX cores usually share the L3 cache). This on is a bit more complex, and needs a work stealing strategy once a CCX DSQ runs out.

David says that the first three example schedulers that are shipped as part of sched_ext are all considered production ready. `scx_rusty` is a multi-domain BPF / userspace hybrid scheduler. `scx_simple` is the one that was showed with the global FIFO. `scx_flatcg` flattens the cgroup hierarchy.

Then there are four other sample schedulers that have very different and creative strategies, but aren't considered production ready.

sched_ext is still out of tree (patchset at v4). It relies on clang for now, but gcc support for bpf programs is upcoming. Meta is following an upstream-first philosophy; and the company's kernel team top priority is upstreaming sched_ext.

The current version is considered complete enough to merged and testable; upcoming features could include power awareness or latency nice.

stress-ng: finding kernel bugs through stress testing – Colin Ian King

stress-ng is tool written by Colin to do stress-testing. There might be different reasons to do stress testing: from finding breakage points (in the kernel…), to check that the systems behaves well underload, scales well, to verify the failure modes, or even to test that the hardware works well.

Stress-ng has already been used to find 60+ kernel bugs; it has brought many kernel performance improvements. Many people use it for performance testing, even some silicon vendors that use it for bring-up of new hardware. Stress-ng has been cited in 80+ academic papers to do synthetic stress testing.

About 10 years ago, Colin was working on laptops, and found in some cases, the CPU might get quite hot, even leading to shutdowns. He started with the "stress" tool to verify the Intel thermal daemon was working properly. Then he needed to improve it, and that's how stress-ng was born.

Nowadays, stress-ng has more than 300 "stressors"; which are used to test various parts of the CPU, the kernel, memory, and even GPUs. They each have different goals.

A stressor has a very simple architecture: first initialization, then a loop that does the work, increase counters, and then it exits when a condition is reached, and runs the clean-up phase.

When running stress-ng, there are global options (for run duration for example), and stressor-specific options (number of instances for example).

Metrics might evolve between versions, so when comparing it should always be between the same versions of stress-ng.

Colin showed many examples. Starting with running different stressors in parallel with a common timeout.

Memory stressing is done from userspace, and it's possible to test the memory behaviour, its bandwidth, or even how the kernel behaves in OOM scenarios.

Network stressing has many modes, from udp, sockets, zerocopy etc. When testing filesystems, there are multiple workloads, and it's possible to verify if there are disk SMART errors reported during the test.

Kernel interfaces like sysfs and procfs have their own dedicated stressors. Syscalls (existing and non-existent) can be tested, and it has found real-world bugs.

Stressors are tagged by classes, so it's possible to run all stressors that are in a given class in a single run with various running modes: sequential, in parallel, with permutations. Stressors can have methods, which are different sub-stresses; by default, they are all used in a round-robin fashion when selecting a stressor, but it's possible to select a specific one.

Stress-ng can run perf directly to analyze counters during a test.

The update cycle is very fast, and the program is very simple to compile. Colin says he really likes testing systems, and that is what drives the development.

To release, stress-ng is tested on many different architectures, compilers, operating systems, with over 100 VMs used before release. Portability is a big goal. Different static analyzers are used as well.

That's it for this morning. Continue with the afternoon live blog.

Live blog day 2: afternoon

Sep 26, 2023

Welcome back on the live blog. You can also follow the live stream.

Demystifying the Linux kernel security process – Greg Kroah-Hartman

The upcoming EU regulation: Cyber Resilience Act (CRA). The main idea behind this regulation is good, but the way it is currently written brings huge issues to open source.

Often, companies ask to join the Linux Security team; but it doesn't work that way.

Linux is big, everyone runs only about 5% to 10% of the code base. The current release model has been stable since 2004; userspace ABI is stable, and won't ever be broken on purpose. Version numbers no longer have special meaning (major number or parity for example).

The stable kernel rules haven't evolved, it's Linus-tree first. Longterm kernels are maintained for at least 2 years. Patching older kernels is more work. The kernel release are considered stable, and people should not fear upgrades.

The world has changed, Greg says; 80% of the world of the world's server run non-commercial distribution kernels. Android is so big that it dwarfs everything in number of running Linux kernels. The "community" does not sign NDAs.

The Kernel security team is reactive, not proactive. Proactive work is left to other groups. It was started in 2005. As part of its declaration, it's not a formal body, and cannot enter NDAs. It uses the security@kernel.org alias. The individuals behind don't represent any company (not even their own); they mostly do triage, by bringing in responsible developers when a security issue is reported, and working in getting this fixed and merged in Linus' tree. If people are brought in enough times, they get added to the alias.

The goal of the team is to fix issues as soon as possible. Embargoes can't be longer than 7 days, and no announcement is done for fixes; they aren't even tagged specially.

Linux Kernel developers don't tag security issues specially because they don't want to alert anyone, and aren't confident that a bug is or isn't a security issue: any bug can have security impact, so all fixes should be applied. Greg invites contributors to do the analysis; many people tried to triage security or non-security patches, and have failed: it's too much work, and it's easy to miss patches. Better to take them all than missing a security issue that is lurking.

The kernel developers don't know the details of every user use case, what code is used in which context; which is why taking all fixes is better idea: even a known security bug might have varying kind of impact depending on the context.

Hardware bugs are a bit of the exception to the Kernel security policy. Greg says the hardware vendors aren't moving fast enough, and embargoes might last 15 months. Embargoes have been tolerated, but having such a long period is no longer tolerable, Greg says. The plan might be to only give 14 days to fix them.

When fixing issues, there is no early announcement; even to a limited audience. Greg considers all "early notice" lists to just be leaks and should be considered public. Otherwise, why would your government allow it to exist ? Greg thinks the `linux-distros` list might not be allowed to exist much longer. He does not want to play this game.

There is at least one security fixes, which is known about; and probably even more that aren't known. Mitre agrees that CVEs don't work for the kernel.

If you are not using an up-to-date stable / longterm kernel, your system is insecure, Greg says.

Panic Attack: Finding some order in the panic chaos – Guilherme G. Piccoli

SteamOS, a linux distribution, has an interest in having a kernel panic log collecting tool.

The goal is to have as much as possible, while being careful with size. kdump is an existing tool to collect crash information at kexec time. pstore can be used to store dmesg or information after a panic in one of the backend (RAM, UEFI, etc.), that resists reboots.

Both need a userspace component. kdumpst for example is in Arch Linux, and uses pstore for kdumps; it has been used by default on Steam Deck, submitting logs to Valve.

An issue, was that it was missing some information, because `panic_print()` was called to late, after kdump. Guilherme proposed to move it before, so that logs contain the appropriate information. But it causes other issues. In the audience, Thomas remarks that calling printk causes calls in serial drivers, and often framebuffer ones for virtual console. It might even touch panic pointers.

Panic notifiers are a mechanism to call multiple handlers (with a priority) in the kernel, when there is a panic. A proposal was to filter those notifiers, but doing that papers overs the main issue. A better proposal was to split the panic notifier list into multiple different lists according to use case.

Currently (6.6-rc2), there are 47 notifiers in the kernels (18 in arch). Before splitting in lists, analyzing those uses is necessary: decoupling multi-purpose ones, verifying priorities, fixing locks, etc. An example was given with pvpanic, which used a spin_lock, and it was changed to a spin_trylock.

The plan is to split the panic notifiers into four lists (from three initially): hypervisors, informational, pre-reboot, and post-reboot. Then a proposal was to modify `panic_print()` into a notifier itself.

An extensive discussion on the proposal exposed plenty of conflicting views. Consensus is at least to make the system as simple as possible. Many fixes will be integrated in the next iteration.

Guilherme gave a real-life example of an issue where a NIC would hang the system by creating an interrupt storm; running kdump in this case didn't work in the middle of the interrupt storm. A workaround was to clear all PCI MSI on boot. Thomas added that in general, the IOMMU configuration should always be cleared at boot.

There is work having graphics showing the panic, but none of the proposal are fully ready yet.

How not to submit a patchset – Frederic Weisbecker

Frederic said this was a tour of his failed patchsets. Starting the lazy RCU patchset. Read-Copy Update is a synchronization mechanism in the kernel used to have readers and writers do the work in parallel, with copies. An example use case was to free an object with an RCU callback when the old copy no longer has any user.

The callbacks could be offloaded to other CPUs. This has CPU isolation and powersaving advantages. The issue is that it has poorer performance. But not all callbacks are performance sensitive. Some, like memory release, can wait and be batched together.

Frederic thought that lazy callbacks could benefit non-offloaded RCUs, bringing even more power savings. And it worked; until it was discovered that performance measurements were buggy. So it didn't work. It showed only minor improvement on rare cases (<1%); it made things worse otherwise. After 3 weeks of work, and a v1 sent, Frederic dropped the idea. He still learned a lot about RCU internals in the process.

The big soft irq lock is next failed patchset. It starts with bottom-half IRQ code; those run with IRQ disabled in a non-preemptible context. A running vector blocks all the others. This is implemented with the "big hammer" local_bh_disable which disables all the IRQ vectors. Frederic had the idea of softening this, by having a specific local_bh_disable that only works on a subset of vectors. But the issue is with the vectors themeselves. So in another proposal, Frederic proposed re-enabling softirqs from vectors that are known to be safe against other vectors. But this had other issues.

After 2 months of works, the design still needs to be justified. He still learned a lot about softirq and lockdep internals in the process, triggering an RT debate.

Nohz_full cpusets interface is next failure. Frederic says he has been working on nohz for close to 10 years. nohz_full stops the timer tick completely on a given CPU. What is done in the timer callback is moved on another CPU. This is a tradeoff for extreme workloads that need full use of a given CPU. The way to enable nohz_full is only at boot time with the nohz_full= command line switch. The plan was to add a runtime way to enable it. After multiple years working on this, Frederic realized that their is no real user need for runtime enabling of nohz_full. So this work is now postponed permanently.

Why do those fail patchsets happen ? Frederic asks. The first reason is that kernel code has become very complex (audience: and the hardware as well). Usecases have broadened a lot; subsystems have grown in stability.

Those failed patchsets are still very good and efficient to start a discussion on a mailing list. There is actual code to discuss, and a better approach can be found. Frederic says reviews should happen in "RW" mode; hacking on a subsystem while reviewing.

7 years of cgroup v2: the future of Linux resource control – Chris Down

Chris is a systemd maintainer and working on the kernel at Meta. He started by saying that control of resource is a critical thing at Meta. cgroup v2 was declared stable a few years ago and brought the unified hierarchy.

In cgroupv1, each resource type (blkio, memory, pids…) was maintaining its own hierarchy. By contrast, cgroup v2 has a unified hierarchy, and resource controllers are enabled on a given part of the hierarchy.

Chris says that without this unified hierarchy, it's impossible to do resource control at scale. If memory starts to run out, this would cause reclaim, and the reclaim costs a lot more if the hierarchies are split. Reclaimable and unreclaimable are important, but not guaranteed. And RSS has been a focus while it provides a very unreliable (but easy to measure) view of process memory usage. Chris gave an example that was thought to use 150MB of memory, that was in fact closer to 2GB and that was found with cgroup v2. In cgroupv2, all types of memory are tracked together.

When cgroup v2 memory.max is reached, reclaim is called for the cgroup. A change is paradigm is to use memory.low, which is a different kind of limit: if a cgroup is below this limit, no reclaim will happen. Chris says that cgroup memory.current tells the truth, but the truth might be sometimes complicated. For example, applications will grow to the limit (with kernel caches for example) if the system is not under pressure.

The issue is to have the wrong, unactionable metrics. cgroup v2 added PSI pressure metrics, like memory.pressure to be able to solve this problem, and understand how to better tune applications.

In order to understand exactly how much memory an application needs to run without loss of performance, Meta wrote a tool called `senpai` which will reduce the memory.high limit of a repeatable task until it finds a stabilized minimum amount of memory.

Memory hierarchy is the way to offload memory to various types of storages, with various level of latency. zswap (compressed in-ram swap) can help a lot here, but the swapping strategy in the kernel was tuned to rotating disks, using it only as last resort, even with swappiness set to 100. A new swap algorithm was introduced in Linux 5.8. The effect was a decrease in heap memory, increase in cache memory, and lower disk I/O activity. Meta calls this Transparent Memory Offload.

Applying resource control on memory is not enough, io needs to be looked as well, otherwise issues might compound.

Nowadays, cgroup v2 is well deployed, on most container runtimes and distributions. It can be verified by booting with cgroup_no_v1=all on the kernel command line. Android is also using metrics from the PSI project.

That's it for today ! You can continue on to the last day !

Live blog day 3: morning

Sep 27, 2023

Welcome back for the last day of this 10th edition of the Kernel Recipes live blog ! You can also follow the video live stream.

Netconf 2023 Workshop – David Miller

This year, Netconf, the invitation-only Linux conference for network kernel developers was held in parallel of the first two days of Kernel Recipes. The agenda was packed.

During the first day, the location for netconf was discussed by Alexei. David is comfortable having it happen co-located with Kernel Recipes. Toke talked about XDP; it's a very important technology, David says.

Jesper discussed page pool, a buffer management tool. It has become more important to have a common mechanism like this for drivers.

Alexandre discussed GRO (receive offload) overhead; decreasing the overhead per packet is quite important for general performance.

Jesper also talked about the impact of ksoftirq on network processing. Networking would welcome important in this area of the kernel.

Paolo discussed evolving some terminology in the kernel to be more inclusive.

Jiri on devlink; devlink allows abstracting a network device when it has multiple ports to only use a subset.

Florian F talked about mDNS offload.

Jakub talked the development process, including the pw-bot. It had improved toil since it detects common issue on patch submissions. The impact of new tools on CI is quite important, and helps a lot. When they don't work, the developers miss them, which Dave compared to an addiction.

Willem talked about refactoring complex code. It's important, but impacts both rebases (making the refactoring harder), and backporting patches to older, pre-refactor version.

Florian W did a workshop on IPSEC; PK key deprecation took a good part of the discussion, as it's getting older, but not all applications have migrated to the new netlink-based API.

Daniel Borkmann discussed header/data split which is very important for zero copy of packets, since the data can be put in a separate page and sent to userspace. He also discussed Big TCP and zerocopy, as well as bpf_mprog to have dependencies and ordering between bpf programs, allowing more complex pipelines.

Eric Dumazet is trying to reorganize struct file to make sure the fast path work. The most important fields should be in one cacheline, and the rest in the next ones. He also discussed deferred wakeups to improve performance by having less wakeups. The idea of using UDP accept() was introduced recently: it makes more sense than it seems for userspace applications, that often have stateful protocols on top of UDP.

Florian W also talked about fixing CVEs in nftables. It's often related to the accumulation of technical debt, and tackling this is important, but the limited resources of the networking developers make this very hard.

David finished by thanking the Kernel Recipes conference organizers, because it stayed true to its roots and is a great environment to discuss technical ideas.

Faster and fewer page faults – Matthew Wilcox

This talk is about four projects. Matthew had a hard time deciding what he should work on; he picked ambitious projects, that made sense and came together.

Linked lists are an immoral data structure, Matthew says. If they are used for anything where you care about performance, it does not work. Your CPU is attempting to extract parallelism from sequential code. His laptop, is able to do up to 6 instructions / clock cycle; missing an L3 cache hit costs 1680 instructions; which is a lot of work. Linked lists bottleneck on fetching the next entry in the list. Arrays don't have this issue, and they can be prefetched. This compounds a lot in the kernel.

A page fault starts with a VMA (Virtual Memory Area) lookup for virtual address. Then the page tables are walked down. If the VMA is anonymous memory, a page needs to be allocated, zeroed, put in the page table, and returned to userspace. Otherwise, for a file-backed page, the VMA page fault handler is called to return a page or populate the page table directly.

VMAs were originally stored in singly-linked list (in 1992); then this was replaced by an AVL tree in 1995; then Red-Black tree in 2001. The RB tree was a bad idea because of imbalance vs the AVL tree, Matthew says. Lookups are penalized to favor insertions. In 2022, a new generic data structure called a Maple Tree was introduced by Liam Howlett to fix this issue, with Matthew advising.

A Maple Tree is an in-memory, RCU-safe B-tree. It changed the tradeoff: lookups are faster, but modification slowers. With RCU, readers no longer need to take a lock.

The next project, in per-VMA locking. VMAs used to be protected by a semaphore (1996); then this was changed to rw-semaphore in 2001. But this introduced its own set of problems, whether readers or writers have priority. In 2023, this was replaced by per-VMA RCU read locks, combined with a seqcount. The approach is complex, but conceptually simple, Matthew says.

Support for per-VMA locking need architecture support, and it was already added for Anonymous VMAs (the most common) to arm64, powerpc, s390, x86 and riscv. Other VMAs type like swap, userfaultd, page cache and DAX have been added later. There is still more support to add, Matthew says.

The past twos brought faster page faults. To get fewer of those, Large folios come into place. Some filesystems like XFS, AFS and EROFS already added fs support for larger than PAGE_SIZE buffer chunks. Matthew says he won't do any other filesystems, but is extending help to any filesystem developer that wants to add folio support.

Larger folios are now possible on file `write()`. Folio support for anonymous memory is being worked on by other people.

Another page-fault reduction strategy is brought by a new interface for PTE (Page Table Entries) manipulation APIs. For example, `set_pte_at()` can only insert one PTE at a time; `set_ptes()` was added to insert multiple consecutive PTEs pointing to consecutive pages. This allows exploiting arm64 and AMD hardware features for PTE entries. Other APIs were added to be able to flush multiple pages at a time.

Matthew says there are a lot of other projects he is thinking about or working on. Matthew is happy to work as an adviser to anyone who might be interested.

Coming soon – Thomas Gleixner

The placeholder title of this talk became the actual one.

It looks like there are four preemptions models in the kernel. Or is there ? Thomas said that in practice these can be reduced to two.

The first of the four is PREEMPT NONE. While userspace can be preempted at any point, in the kernel there is cooperative multitasking that uses `schedule()`. But it might cause issues of latencies and starvation because the NEED_RESCHED flag will only be used at the syscall interface. The workaround is manual placement of rescheduling capabilities using `cond_resched()` for points in a kernel task where preemption can happen. It's a manual and error-prone task, and it needs careful placement to prevent doing it with lock helds for example.

The next is PREEMPT VOLUNTARY. It's mostly the same as NONE, with additional scheduling opportunities with `might_sleep()`. Initially, `might_sleep` was a debug mechanism, and `cond_resched` was glued onto it, which bothers Thomas. It can result in redundant task switching and even lock contention for example. It does provides better latencies than NONE, but has the same issues, with even more contention possible.

FULL preemption model is for multitasking, based on timeslicing and priority. There are still non-preemptible sections in the kernel: spin and rw locks, any interrupt disabling or per-cpu accessors, or even simply calling `preempt_disable()`. In those sections, migration is prevented and blocking operations are forbidden. Migration prevention can be made preemptible with `migrate_disable`.

In this model, the scheduler knows when preemption is safe. But latencies can increase, and agressive preemption can cause contention.

The RT preempt model is the same as FULL. RT will further reduce the non-preemptible sections, by changing some locks to become sleeping locks. There are also further restrictions in RT for non-preemptible sections: no memory allocation, or call to any function that might itself use a now-sleepable lock. The tradeoffs are similar to RT, but with smaller worst case latencies in exchange for less throughput.

This throughput tradeoff for non-RT tasks can be mitigated with LAZY preemption.

On x86, REP MOV/STO can bring uge memcpy/memset performance improvement. This instruction can be interrupted, but not in the NONE and VOLUNTARY preempt modes. So in those cases large copies/clear will cause latencies. In this case, going back to a chunk-based loop (instead of single instruction), it looses the hardware benefit.

Bringing new semantics on NONE and VOLUNTARY is error-prone. Which is why there's the need to take a step back. The explicit points are moved regularly in those collaborative scheduling mode, and these are hacks. The preemption counter should be always enabled, Thomas says, since it's no longer as expensive (even if not free). This would allow all preemption models to know when it is safe to preempt.

This would also bring preemption model reduction. NONE, VOLUNTARY and FULL could be merged, giving more control to the scheduler. It would be aware of the voluntary semantics.
And `cond_resched()` can then be removed completely. Scheduler hints for lazy preemption can be added for the normal case.

All this work is still at PoC level, but it works and looks promising. RT would still be separate and selected at compile time. "Museum" architecture support is a challenge, but this can be worked on.

In response to a question from the audience, Thomas said that the mechanisms in the kernel shouldn't be convoluted with the policy, which comes from elsewhere.

That's it for this morning ! Continue with the afternoon live blog.

Live blog day 3: afternoon

Sep 27, 2023

This is the last live blog for this Kernel Recipes edition.

Hardware and its Concurrency Habits – Paul E. McKenney

Hardware has evolved a lot in the last 40 years: CPUs have more cache, deeper pipelines, are out of order, and mostly unpredictable. Paul showed a the simple block diagram of the 386 ALU. With the help of a logic analyzer it taught him concurrency.

At the time, instructions took multiple cycles. Pipelined and Superscalar execution changed that. Where does this hardware complexity come from ? Laws of phisics first: atoms are too big, and light is too slow. Transistor atom size determines switch speed. Light goes 1 width of A4 paper per nanosecond. In silicon, this is 100 times less. As a result, data is even slower.

This accounts for a part of the CPU complexity. But people want to write portable code as well, and this impacts performance. In addition, there are systems with a lot of CPUs.

In order to get the maximum performance, one needs perfect branch prediction. Branch misses are an obstacle. Memory references are another: anything not in cache loses hundreds of clock cycles. Atomic operations require locking cachelines and busses; even if the impact has lowered recently. Memory barriers are another obstacle. More recently, Thermal Throttling is another issue if the CPU is used efficiently enough.

Which obstacles to focus on? Paul wants to focus on cache misses, memory barriers and atomic operations.

Paul showed the time to do a compare-and-swap on a Xeon CPU. Depending on the use case, it can take from 14 to 2000 cycles (across cross-connect in multi-socket configurations). To get a sense of scale, Paul went in the audience with multiple rolls of toilet paper. A single sheet represented a cpu cycle; 2000 cycles would be at least 4 fully unrolled paper rolls.

Can hardware help ? There is research to improve the latency by improving the integration. Stacked transistors for example.

Hardware accelerators also show promise, too; depending on the usecase. They have been helping for quite some time.

Memory hierarchies help. L3s have been growing a lot recently. Hardware is not afraid to throw transistors at the problem.

To summarize, Paul says that modern hardware is highly optimized, most of the times. Incremental improvements have compounded.

Gaining bounds-checking on trailing arrays in the upstream Linux kernel – Gustavo A. R. Silva

The work presented comes from multiple contributors, Gustavo says.

Arrays in the C language can be declared simply, but the boundaries aren't enforced in the language. Trailing arrays are arrays found at the end of a structure. Flexible arrays are trailing arrays where the size aren't known at compile-time, but determined at run-time.

There are two types of fake flexible arrays: they can be 1-element or 0-length arrays. Both are used as flexible arrays, but don't use the "modern" C99 flexible array syntax.

1-element arrays are prone to off-by-one problems, and extra-memory might be allocated if not careful. They trigger -Warray-bounds false positives.

0-length arrays are a GNU extension; they don't contribute to the size of the struct, but they also trigger -Warray-bounds false positives. Another issue, might be undefined behaviour if someone adds a struct field after the 0-length array.

Another undefined behaviour can be triggered if the structure with a fake flexible array is embedded in another structure, and has fields after it. It should always be at the bottom; and even the embedding structure should be at the bottom if embedded in another structure. Luckily, there's a new warning in development for GCC 14 to detect those called -Wflex-array-member-not-at-end.

After testing this warning, Gustavo found 650 unique occurrences of it in the kernel.

Another issue, is that `sizeof()` of a (C99) flexible array member is a compile error. And that brought problems in the kernel. For fortified memcpy, this meant that __builtin_object_size returned -1, for fake or not flex arrays; which is inconsistent with `sizeof`. It even failed for any type of trailing array. This is for historical reasons: in struct sockaddr, the trailing 14 elements array in fact behaved like a flexible array. And this broke bounds-checking for any type of trailing array.

There's a new compiler option GCC 13 and clang 16 -fstrict-flex-arrays=<n> to be able to control that behaviour. If n is 3, all type of trailing arrays gain bounds-checking.

Starting with Linux 6.5, both fortified memcpy and -fstrict-flex-arrays=3 are enabled globally.

In order to properly sanity check, an attribute is being added in GCC 14 and clang 18 to point to the element count of the flexiible inside the structure. For bounds-checking, __builtin_dynamic_object_size replaced __builtin_object_size in the fortified memcpy; it's able to use the hint from the attribute.

In order not to break UAPI, the whole structs was duplicated in a union with both 1-element array for userspace, and flexible array for the kernel. There are now helpers to simplify this.

The impact of this work even had impact on improving user-space.

Getting the RK3588 SoC support upstream – Sebastian Reichel

The RK3588 is an SoC from Rockchip. He was asked to look at support for it last year; a few days later he had the evaluation board on his desk.

He had a look at the available source code from the vendor, and tried to create a minimal device tree from that. His goal was to only have the serial port and uboot. He tried it, and had nothing. After adding interrupts, he was able to get some kernel error messages on the serial port.

Slowly extending the device tree step-by-step, he was able to try and fix the error messages: first missing interrupts, then CPU properties, then other devices. There were many third party devices. Most of them depended on clocks, so he started with that.

The goal was to boot a Debian userspace. First step was clocks and resets, then pinctrl, eMMC and normal console.

Clokcs and reset are similar to previous generations. He needed to add support for lookup tables for different register offsets. Then he worked on the clock gates; the clock framework only supports one active parent, which didn't match the hardware reality, that had linked clocks. Initial solution was to mark linked clocks as critical, wasting power but making the device work; this is being fixed now.

The V2.1 GPIO controller is very similar to the V2.0 version already upstream. eMMC needed two changes compared to the previous generation; and there even was a regression in 6.4 breaking the controller on the RK3588; this is now fixed.

Once Debian booted far enough, Sebastian started to upstream the incomplete Device Tree. While this was being reviewed, he continued working on adding more features.

Network and power management domains were the next two ones. They worked, until a colleague received another Radxa board. So he fixed it. But the next Rockchip board was done even more differently, using PCIe ethernet.

It needed GIC-ITS (Interrupt Translation Service) enabling, but just enabling it broke the boot completely. The RK3588 has a design flow, making the cache non-coherent. Previous generation did not use the ITS either. An Errata was send from Rockchip, working around the issue, and Lorenzo Pieralisi started working on a generic solution, proposing a change to the ACPI standard.

The PMIC (Power Management IC), was new for the RK3588, with two different configurations using different chipsets. Having both work required fixing RK808 MFD subdrivers in a lot of subsystems.

Multiple hardware blocks were supported by simply adding a compatible string in the device tree, so not a lot of changes. For some devices, DT bindings were broken and had to be fixed. AV1 codec work started early.

Most of the basics are supported. Many persons are working on HDMI Out, In, USB3, GPU, Crypto and DFI. DisplayPort, DSI, CSI, ISP, Video Codecs, SPDIF, CAN, RNG are still at the TODO status; the audience said that someone had a demo for DSI and CSI.

In u-boot, Ethernet support was needed for Kernel CI support. But it didn't work in the downstream u-boot; fixing this was as simple as disabling an option.

Upstream u-boot support is being worked on by multiple people. It follows a milti-step plan. DFU support won't be worked on by Sebastian, he said.

Evolving ftrace on arm64 – Mark Rutland

`ftrace` is a framework to attach tracers on kernel functions at runtime. It is used for tracing, but also live-patching, fault-injection. It is used in production environments thanks to its minimal overhead. It needs architecture-specific code.

In order to hook a function entry or return point, ftrace needs architecture specific magic. This "magic" is related to how functions are called. On arm64 this happens with the `bl`, branch-and-link register, that jumps to an address, and stores the return address `lr`, the link register. In the function entry point, this lr might be stored on the stacked, and restored at the end. Using the `ret` instruction will consume the link register.

Another "magic" that ftrace relies on is compiler feature called `mcount`. It inserts at the beginning of every function calls to an `_mcount` function. The `_mcount` function is used as a trampoline to hook the entry point. But what about the return ? To do this, hooking the frame record (the part where the link register is placed on the stack) is required. By modifying the return address to instead a return tracer, that can then restore the link register that was saved in the entry point hook.

But when tracing is not being used, the call to mcount shouldn't stay here in every function of the kernel. Another bit of magic is to replace the calls to mcount by nop instructions, this way the overhead is minimal.

That was all simple, until Pointer Authentication came in the loop. It's a new arm64 CPU feature that protects against ROP/JOP attacks: at the beginning and end of every function, the compiler inserts two new instructions: `paciasp` and `autiascp`. And this breaks the mcount-based approach.

GCC 8+ added support for `-fpatchable-function-entry=N` which adds nop at the beginning of every function, but before the `paciasp` pointer authentication instruction; so the link register isn't signed yet. The net result of this new patchable-function-entry is also much better code generation, with less instructions.

The current approach only allows calling a single tracer; for tracing this might not be enough, so ftrace has common code to multiplex tracers. Some architectures can add multiple tracers trampoline dynamically to a callsite, but this isn't feasible cheaply enough on arm64, Mark says.

In order to make that work cheaply enough, a creative solution had to be found. Before each function entry, 64 bytes are added for custom trampolines. Combining that with patchable function entry, allows having per-callsite ops. The logic is simple enough to be maintainable and enable other features.

That's it for this edition of Kernel Recipes ! See you next year !