# Linux Kernel on Reddit

- ebpfexplorer: An eBPF programs & maps explorer (2026/11/04 09:48)
  submitted by /u/noboruma [link] [comments]
- How to start writing device drivers? (2026/11/04 09:11)
  submitted by /u/Dangerous-Hamster272 [link] [comments]
- The Fourteen-Byte Cliff: Reconstructing an act_ife Short-Linear-Head Panic (2026/11/03 09:27)
  eth_type_trans() panicked because fourteen packet bytes were present in page fragments but absent from the linear head of an sk_buff. A valid 5,082-byte IFE frame triggered the fault. TC's IFE decoder advanced skb->data through two Ethernet headers and its metadata block. loopback_xmit() then treated the remaining data as another Ethernet frame and pulled fourteen bytes from a linear head whose length was zero. submitted by /u/lapzero [link] [comments]
- Offloading zram compression to GPU? (2026/11/03 02:17)
  Is there a way to offload zram compression & decompression to the GPU? I have a compilation server with 36 logical cores but only 16GB of RAM (I am unable to upgrade due to the global RAM crisis; DDR4 registered ECC costs a ton). In an effort to squeeze more "RAM" out of my RAM, I decided to start using zram compressed with zstd. I managed to get 48GB of zram to work (3:1 compression), however it resulted in one of my cores being pinned at 100%, desperately trying to compress the increasing amount of ram being used by 36 compile jobs. Is there a way to offload this to the GPU? I think it would greatly speed up compilation if possible, and would perhaps allow for a greater compression rate (zstd ultra maybe???) EDIT: The GPU is a GTX 1660 Super with the open-source kernel modules from NVIDIA. It basically never gets used and usage is at a constant 0-1% submitted by /u/Silent-Degree-6072 [link] [comments]
- How well does linux modules handle resource deallocation? (2026/11/01 06:24)
  Hello, I'm a CS undergraduate studying about linux kernel development. I'm currently on LDD3 book, emphasizing about importance of cleaning up allocated resources on exit. I assume that this is not likely to be handled properly in some linux modules as removal of modules is not something that is performed on a daily basis. I have been looking for places to start my contributions and was wondering whether this, identification of memory leaks and applying fixes, could be a valid entry point. The questions I want to ask is, Is this problem considered too trivial? Have others been working on it? (maybe there exists some linux group) Is this problem commonly found, or gets managed during review process and fixed before getting accepted? Thank you for your time. Any comments are welcome! *EDIT Thank you all for your kind responses. They have been very helpful ! submitted by /u/AccomplishedAir8352 [link] [comments]
- What do I add in ./security/ and ./crypto/ (2026/11/01 03:53)
  I'm building a monolithic kernel called sonix (Son I'm crine + Unix) and I just found out about encrypted filesystems and I don't know what to put in these directories submitted by /u/Totallynotnormalguy [link] [comments]
- Should we make an AI-free fork of the linux kernel? (2026/10/31 13:59)
  submitted by /u/Logogram_alt [link] [comments]

- Best book for learning Linux kernel Delevopemet and Device driver. (2026/10/31 04:19)
  submitted by /u/Commercial-Golf-4285 [link] [comments]
- Security Kernel Engineer (2026/10/30 13:02)
  What topics would you consider essential for a kernel security engineer? I am currently studying memory corruption, privilege escalation, race conditions/TOCTOU, exploit mitigations, secure boot, and hypervisor isolation. Which areas would you prioritise and are there any, that I am missing? Also what tools are useful to be familiar with, when dealing with the security of the kernel? submitted by /u/Serious_Ad_1942 [link] [comments]
- BrainixKernel — an alternative to XNU/macOS, built from scratch for AArch64 (2026/10/29 12:23)
  Hi r/kernel, I want to share BrainixKernel — an experimental AArch64 kernel I'm writing from scratch — and explain why it exists, what its architecture is, and where it's going. I'm posting early on purpose: I'd rather get architecture criticism now than after I've built more on top of a bad decision. Repo: https://github.com/Bindo-Lenzini/BrainixKernel (Caption: BrainixKernel OS (BKOS) v0.45 — my test harness running a custom window manager, top bar, and a user-space ksh terminal on a 2560x1600 framebuffer). Why it exists I started BrainixKernel as an alternative to XNU and macOS — not a drop-in replacement, but a different answer to the same question: what would a modern, auditable, capability-based kernel look like if it were designed from scratch today? Two motivations drove that: macOS is closed. You can't audit it, can't reason about it end-to-end, can't modify the parts that matter. XNU carries a lot of legacy. It's a hybrid of Mach and BSD with decades of compatibility layers, resulting in a large TCB that's hard to reason about as a whole. That's not a criticism of the people who built it — it's a consequence of 30+ years of evolution. But it means the design space without that legacy is still open, and that's the space I want to explore. So BrainixKernel is a research/hobby project testing one hypothesis: a kernel can be small enough to read end-to-end, capability-based at its core, and still useful — networking, filesystem, user space, graphics. Architecture & What works today AArch64-first, targeting QEMU virt. No x86 legacy to carry around. Monolithic kernel, single CPU for now. MMU, virtual memory management, preemptive scheduler, syscall layer. User space with its own libc and a small shell. bkfs — my own filesystem, backed by a Virtio block driver. Network stack from scratch: ARP, DHCP, IPv4, ICMP, UDP, TCP. wget and udp_echo work today. Virtio-GPU & Framebuffer graphics. (Caption: Boot log on QEMU virt*: ARP resolution, ICMP ping,* bkfs mount, Virtio-GPU initialization, and launching the user-space ELF shell.) The Test Harness (BKOS) To test the kernel without constantly building a full OS, I built BKOS (BrainixKernel OS) — a lightweight test environment running in user space. It features a custom window manager, a dock, "Liquid Glass" background, and a terminal. It's not the final OS (which I will build in the future), but a proving ground to test new kernel features like the ELF loader, per-process address spaces, and the graphics stack. On AI assistance — yes, I use it, and I see nothing wrong with that Let me address this head-on, because it's increasingly relevant in hobby kernel projects and I'd rather state my position openly than have it "exposed" in the comments. I use AI for routine code — boilerplate, repetitive driver scaffolding, build scripts, documentation, mechanical things where the design is already decided. I don't hide it, I don't apologize for it, and I don't think it makes the project less legitimate. Using the best available tools is normal engineering; nobody demands you hand-write your assembler before you're allowed to write a kernel. What AI does not do is architecture. The MMU layout, the scheduler design, the capability model, the bkfs on-disk format, the syscall ABI, the decision to go monolithic and AArch64-first — those are mine. I understand how this kernel works end-to-end, and I can explain and defend every subsystem in it. If something in the repo is wrong or inconsistent, that's my responsibility, not a generation artifact. If you want to test whether I actually understand the code: ask me about any subsystem in the repo and I'll walk through the design and the implementation. What does NOT work yet (important) Capability

enforcement is incomplete. bkfs doesn't store UID/GID or file permissions yet, and new tasks get UID 0 with all capabilities — so in practice every process is root. Enforcement on open()/exec()/IPC is the next major milestone. Single CPU only. No SMP, no locking, no IPIs. No mmap**,** no demand paging, no page cache. Build paths are currently hardcoded for macOS. Linux build support is coming. What I want from you Given the goal — small, auditable, capability-based — what's the biggest architectural mistake I'm making that will hurt in 6 months? Capability enforcement design: what should I study before committing? I know seL4 is the reference, but I want something lighter. Any smaller kernels with a clean capability model worth reading? SMP: what's the realistic order of operations so I don't rewrite the scheduler twice? For those who've reviewed AI-assisted kernel code before — what are the typical red flags I should audit for in my own repo? I'd rather find them myself. I'm aware that "alternative to XNU" is an ambitious framing for a hobby project — that's exactly why I want the criticism now. If the hypothesis is wrong, I'd rather find out from people who've built kernels than from my own bugs in two years. Thanks. Processing img vobvns9aheyh1... Processing img g7ba7u9aheyh1... submitted by /u/bindoLenz [link] [comments]

- Starting Linux kernel/driver dev: is this roadmap realistic, and what would you add or cut? (2026/10/29 11:03)
  submitted by /u/More-String6376 [link] [comments]
- How to intercept a physical webcam data stream and inject/substitute custom video at the driver/kernel level? (2026/10/28 05:46)
  submitted by /u/fkj83 [link] [comments]
- kernel compilation (2026/10/27 17:55)
  submitted by /u/milomak [link] [comments]
- How 2 fix my unstable CCF? (MSM8255 to 7.2.9) (2026/10/26 12:37)
  Hello everyone (or whoever is looking in here), i am currently trying to upstream an ancient 2011 board into mainline. Its been weeks and days ever since i decided to start translating 3.4 nAa kernel to 7.2.9 CCF because the limitations were terrible and i dont know whether its the arm-none-eabi and mkbootimg fault for not giving me a working screen and/or a normal dmesg, but the amounts of brain damage i may have experienced with a lot of sleep deprivations just made my day worse by ever so slightly feeling horrible that the compiler might throw another tantrum. Sure it's not throwing 7 thousand lines of errors like the first time but i have house trained it to not throw rocks again, i didnt change any critical files that break other phones or SoC's Device: Sony Ericsson Xperia Arc/Arc S Goal: make it have a working screen and make it spit out that it cant mount a system (because doing all the ramdisk wizardry is useless as now its gonna be a userspace problem whether its set correctly or not) if noone can help me then invite postmarketOS (or Nura) porters into this madness, maybe they can just throw in some overhacky things that might get accepted into lkml when i send it or just keep it locally. https://pastebin.com/3aCHzuhe the CCF. https://pastebin.com/BfMZrYZN the dt-binding clocks. submitted by /u/govnonabortike [link] [comments]
- Can a linux kernel engineer transition to an LLM inference/model perf engineer? (2026/10/26 07:34)
  submitted by /u/Different_Buyer2770 [link] [comments]
- J'ai découvert que mon scanner AF_XDP était plus lent que nmap. Le réparer m'a appris quelques choses sur le mode de copie. (2026/10/26 07:10)
  Suivi de mon post précédent ici. J'ai construit un scanner de réseau open-source, tcpcat (https://github.com/NycolazSec/tcpcat), avec un chemin rapide AF_XDP. Projet solo, écrit en Go. Certains d'entre vous m'ont poussé à réfléchir sur le natif vs générique et si le mode de copie en valait la peine. Cela s'est avéré être exactement la bonne chose à examiner. D'abord la partie embarrassante. J'avais un chiffre de "2,62 fois plus rapide

que nmap". C'était faux. Un bug faisait qu'il ne scannait qu'un des deux hôtes cibles, et le temps qu'il rapportait (4,47s) était en réalité en dessous du seuil théorique de ma propre limite de taux (131k probes à 25k pps ne peuvent pas finir en moins de ~5,2s). J'ai donc retiré tout ça et j'ai reconstruit le benchmark correctement : il vérifie désormais que chaque exécution a trouvé tous les ports ouverts et enregistre l'utilisation du CPU et les trames perdues. J'ai relancé sur un vrai serveur (x86_64, noyau 6.12, mais toujours veth donc mode de copie). Le moteur eBPF s'est avéré être le plus lent des outils rapides. 8,7s, contre nmap à 5,6s et mon propre scan SYN sur socket brut à 5,5s, avec une variance désagréable. Deux choses étaient fausses : Busy-poll + verrouillage CPU. Ça allait sur mon portable au repos, mais c'est activement nuisible sur une machine chargée/virtualisée : le fil RX épinglé se battait avec les autres charges de travail pour un vCPU. J'ai rendu le busy-poll et le verrouillage sans copie uniquement. En mode copie, la boucle RX bloque maintenant simplement dans poll(). 8,7 à 7,7s, la variance a beaucoup baissé. Le vrai problème. AF_XDP a une seule bague TX, et je faisais une transmission (un sendto) par sonde, sous verrou. À 25k pps, cet appel système par paquet était pratiquement tout le temps de mur sur un hôte lent. Regroupé jusqu'à 64 trames en une seule transmission. 7,7 à 5,65s. Maintenant, ça égalise nmap. Le bilan honnête : en mode copie, AF_XDP n'est pas miraculeusement plus rapide. Il égalise nmap en temps mais à environ un tiers du CPU du scan SYN brut (8,7s contre 25,8s de CPU). Purement, le scan SYN est en réalité le plus rapide, car il utilise plusieurs cœurs et la seule bague AF_XDP ne peut pas suivre. Le véritable potentiel de vitesse nécessite clairement un zéro-copie sur une NIC multi-queue, ce que je ne peux simplement pas tester (tout ce que j'ai est en mode copie, file unique). Je travaille sur la bibliothèque Go asavie/xdp, qui est assez basique (pas besoin-de réveil). J'ai pensé à déplacer la boucle chaude vers C/libxdp, mais le seul binaire statique vaut la peine d'être conservé et le regroupement m'a aidé à faire la plupart du chemin. Deux questions pour le sous-forum : - Quelqu'un a-t-il une NIC qui fait le zéro-copie AF_XDP pour exécuter un benchmark rapide ? Je suis vraiment curieux de savoir combien la modification d'écriture de la bague plus besoin-de réveil change les choses par rapport au mode copie. - En mode copie avec une bague TX, y a-t-il quelque chose de plus intelligent que de regrouper les sendto pour pousser le débit, en dehors du passage à multi-queue ? Le code et le script de benchmark (Docker, vérifie la correction) sont ici si vous voulez jeter un œil ou reproduire : https://github.com/NycolazSec/tcpcat . Les PR et les résultats de test de NIC sont très appréciés. submitted by /u/nycolazsec [link] [comments]

- What are the best books to learn Embedded Linux and Linux device driver development for SoCs? (2026/10/24 15:02)
  submitted by /u/CertainBaby9837 [link] [comments]
- ACPI regression on lts (mainline works). Worth reporting? (2026/10/23 15:25)
  submitted by /u/Strazza02 [link] [comments]
- All kernel paths lead to C. (2026/10/22 15:39)
  submitted by /u/Unique_Ebb_5802 [link] [comments]
- The programming languages of the independent kernel I have developed (2026/10/22 00:38)
  submitted by /u/Unique_Ebb_5802 [link] [comments]
- AI usage in private vs military sectors (2026/10/21 21:32)
  Hey all, I have been in the field for little over 2 years. One year working as an embedded firmware engineer and for more than a year more as a linux device driver/bsp engineer and for the last 3-4 months company that I work at has been pushing very heavily to agentic development. Basically a bug is found and my boss says just give it to claude and give the ssh access to the product that we are developing to claude too so it can debug this. And I really hate that and I feel like im not learning as much as I should be. So basically my question is: Is there a difference in

these agentic workflows between private sector companies and military companies/startups where in my opinion privacy is way more important than in private sector. The main reason why Im asking this question is because im seriously considering switching jobs and I just want to know what is the best sector to go to. Thank you and will be waiting for your opinions and answers. submitted by /u/reinars1 [link] [comments]

- Custom Dma linux driver (2026/10/21 10:29)
  submitted by /u/Frank-Issuer50 [link] [comments]
- Which new 7.3.0 default kernel module messed up my saturation and color? (2026/10/21 09:49)
  I've compiled the kernel as I've always done (i used the same modprobed.db I've always used) and selected all the default new modules by pressing enter a bunch, now even #000000 appears light gray and everything else is washed out. I'm using a 9070xt through HDMI, monitor doesn't have HDR and its 1080p@60Hz. I still have a boot entry for 7.2.11 pre compiled by gentoo, i tried running modprobed-db store on it once more and recompiling 7.3.0 but nothing changed. I also tried using the iGPU (7700x) and the issue persists. What can i do? I'd appreciate any input about the suspect causing this, i don't want to hunt down which of the 10-15 new modules is causing this by exclusion. Thanks in advance for any and all replies. (sorry if it's a stupid question, but this seemed like the best place to ask) submitted by /u/OkAlbatross9889 [link] [comments]
- Proposal: A Rust-based Secondary Microkernel for Lightweight, AI-Native OS Architecture, Local Eco-Sync, and Bare-Metal Resource Optimization (2026/10/20 09:22)
  submitted by /u/Careful-Rutabaga2162 [link] [comments]
- CPU_OUT_OF_SPEC in kernel 7.3.0 (2026/10/19 17:23)
  I built 7.3 this morning. When I booted I got that 219 times. When I halted I got it 173 more times. I switched back to 7.2.11, which hasn't so complained. What does it mean? https://pastebin.com/QdxJPaD2 has the full error. submitted by /u/ArthurPeabody [link] [comments]

From:
https://wiki.tromjaro.alexio.tf/ - **TROMjaro wiki**

Permanent link:
**https://wiki.tromjaro.alexio.tf/doku.php?id=news:reddit:kernel&rev=1584393415**

Last update: **2021/10/30 11:38**