東京 2025

LINUX PLUMBERS CONFERENCE

TOKYO, JAPAN / DECEMBER 11-13, 2025

Multikernel: Kernel-to-Kernel Isolation with Elastic Resource Management

Cong Wang <cwang@multikernel.io>
Founder and CEO at Multikernel Technologies

東京 2025
LINUX PLUMBERS CONFERENCE
TOKYO, JAPAN / DEC. 11-13, 2025

A Common Assumption

- A machine can only run a single Linux kernel
- Running multiple kernels traditionally requires either:
    - Specialized asymmetric multiprocessing (AMP) hardware
    - Or virtualization (KVM, Xen, etc.)
- Rooted in decades of SMP design:
    - All CPUs are managed symmetrically
- But this "one-kernel" model is not a hardware requirement

Introducing Multikernel

- Run multiple Linux kernels concurrently on the same machine
  - No virtualization, no hypervisor
  - No specialized hardware required
- Each kernel controls its own hardware partition
- Kernels communicate via explicit messaging

Previous Multikernel Research

Barrelfish (ETH Zurich, 2009)

- Replica-based single system image design
- Not Linux

Popcorn Linux (Virginia Tech, 2013)

- Process migration across kernels, still maintains a single system image
- Focuses on heterogeneous ISA

McKernel (RIKEN, 2015)

- Custom lightweight kernel from scratch (~200 syscalls)
- Two-tier design: a performance kernel plus a service kernel

Our Design: Kernel-to-Kernel Isolation

Design Principles:

- Isolation rather than replication
- Kernels coexist as equal peers
- Host/spawn kernel distinguished only for management duties
- Each application runs on its own tailored Linux kernel
- With dedicated yet elastic hardware resources

Design Overview

- Host kernel
  - Manages hardware resources
  - Manages spawn kernels
  - Coordinates cross-kernel state
- Spawn kernels
  - Tailored upstream Linux kernels
  - Run applications independently
  - Isolated via CPU, memory, I/O
- Inter-kernel communication uses IPI and shared memory

Tailoring Linux Kernel

McKernel's burden:

- Custom kernel from scratch
- Implements ~200 syscalls; delegates the rest via IPC
- Linux kernel is more than just syscalls (eBPF + kernel modules)

Our approach:

- A tailored and optimized Linux kernel per application
- Fully compatible upstream Linux
- Use kexec to load Linux kernels

Reusing Kexec

- Loads and unloads kernel images via kexec_file_load() syscall
- Inherits existing kernel-signing infrastructure
- Reuses Kexec HandOver for cross-kernel resource passing
- Separates resource management out

Our Philosophy: Don't reinvent the wheel, reuse what exists

Reusing Everything

- Reuse IPI for inter-kernel communication
- Reuse genpool for physical memory management
- Reuse the kernel suspend/resume mechanisms for freezing
- Reuse hibernation mechanisms for checkpointing and kernel switching
- Reuse vsock for user-space inter-kernel communication

Our Philosophy: Embrace Linux with deep integration

Static vs Dynamic

Our perspective:

- Static partitioning is simply the default state of dynamic resource allocation
- The reverse is not true

For cloud and HPC:

- Dynamic resource management is not optional, but essential
- Challenge: achieve dynamism while maintaining isolation
- Flexibility ultimately prevails

Resource Management

- **Key Insight**: Resource management is critical for Multikernel architecture
- Our innovation is using Device Tree for Multikernel resource management
- Device Tree is the established standard for hardware resource description in embedded systems
- Device Tree Overlays enable dynamic hardware configuration changes at runtime
- This also avoids fragile kernel command-line manipulations

Device Tree Example

```
Device Tree:
/dts-v1/;
/web-server {
    compatible = "multikernel-v1";
    id = <0x1>;
    resources {
        memory-base = <0x0 0x40000000>;
        memory-bytes = <0x0 0x20000000>;
        cpus = <0x1>;
        devices {
            enp9s0 {
                device-type = "pci";
                pci-id = "0000:09:00.0";
                vendor-id = <0x1af4>;
                device-id = <0x1041>;
            };
        };
    };
};
```

Device Tree Overlay Example

```
/ {
    compatible = "linux,multikernel-overlay";
    fragment@0 {
        __overlay__ {
            cpu-add {
                mk,instance = "web-server";
                cpu@2 {
                    reg = <0x2>;
                };
            };
        };
    };
};
```

Dynamic Resource Allocation

Key Innovation: Leverages existing Linux CPU/memory/PCI hotplug

1. Userspace generates a device-tree overlay describing changes
2. Host kernel translates to resource update messages
3. Host kernel sends updates via IPI and shared memory
4. Target kernel applies the changes via standard hotplug mechanisms

Hardware Device Sharing Challenge

Key Challenge: I/O devices are typically fewer compared to CPU cores

SR-IOV Limitations:

- Hardware-level isolation requires virtualization
- Limited VF (Virtual Function) count
- Inflexible resource allocation

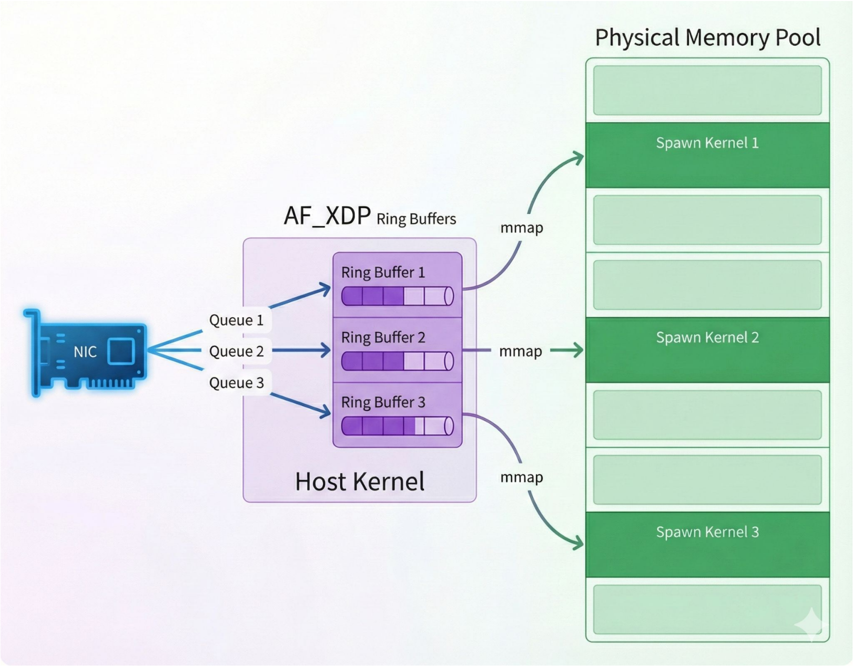
We prefer hardware-level isolation without using virtualization stack

Our Solution: Hardware Queue Isolation

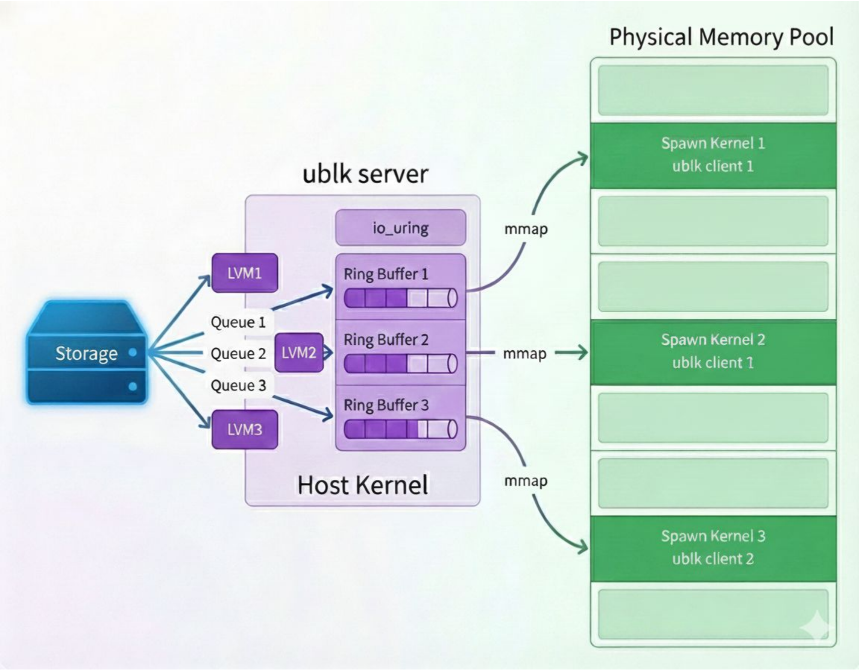
Key Insight: Modern I/O devices support multiple hardware queues

- Each kernel gets exclusive access to specific queues
- Leverages AF_XDP and io_uring for ring-buffer–based zero-copy I/O
- UAPI stability provides even stronger compatibility guarantees
- Dynamic queue assignment is more elastic than hotplug

Isolating Networking Queues With AF_XDP



Isolating Storage Queues With ublk



Use Case 1: Cloud Computing - Container

- Kernel-level isolation vs resource-level isolation
- Performance isolation: eliminates noisy neighbors
- A tailored lightweight Linux kernel, reduces OS noises
- A tuned, optimized, and highly customized per-application Linux kernel

Use Case 1: Cloud Computing – VM

- No hypervisor, no virtualization overhead
- No single point of failure
- Much simpler stack, reduced attack surface
- More elastic resource management
- Fast startup: no OS, boot into init=/your/app directly

Use Case 2: Live Kernel Update

Traditional kernel update:

- Node reboot required
- All applications are interrupted
- Cluster-wide scheduling updates; requires redundancy

Live Update Orchestrator (LUO):

- Reduces but doesn't eliminate downtime
- All processes pause during kexec transition
- All-or-nothing state transfer

Our Approach: Parallel Kernel Execution

Idea: Run old and new kernels simultaneously

1. Boot the new kernel alongside the old one
2. Gradually migrate processes from old to new kernel one by one
3. Retire the old kernel once all processes have moved to the new one
4. Can rollback at any checkpoint

Result: Zero application interruption, **truly** zero downtime

Use Case 3: Kernel Crash Auto Healing

Traditional approach:

- Kdump kernel captures vmcore, cannot run production services
- Must reboot **twice** (~minutes of downtime)

Our approach: maintain a backup kernel running in parallel

- Backup kernel detects crash via IPI heartbeat timeout
- Takes over all hardware devices, restores process state from preserved memory
- Resumes service in a **few seconds**

Use Case 4: Kernel Live Debugging

Traditional kernel debugging:

- kdump for post-crash debugging
- /proc/kcore of the running kernel
- Impossible to pause a running kernel without a debugger

Multikernel approach:

- Cross-kernel debugging: the host kernel debugs spawn kernels
- Obtain the kcore through kernfs
- Pause and resume spawn kernel execution at any time

Security Model

Implications:

- Each kernel is trusted to honor its resource boundaries
- Reduced attack surface through kernel/OS customization

Suitable for:

- Multi-tenant systems within same organization
- HPC centers with controlled environments
- Dedicated devices with strict kernel control

Trade-off: Performance over absolute security isolation

Call for Hardware Enhancements

1. CHERI (Capability Hardware Enhanced RISC Instructions)
    a. Fine-grained memory protection via hardware capabilities
    b. Bounded pointers with enforced ranges
2. Hardware-Filtered IPI
    a. Hardware-enforced IPI access control without software overhead
3. Confidential-computing-friendly hardware support
4. NVMe Controller Partitioning

Multikernel Linux

Linux kernel implementation for upstream

https://github.com/multikernel/linux

Beyond Kernel - kerf

kerf simplifies multikernel management and orchestration

https://github.com/multikernel/kerf

Beyond Kernel - kmorph

kmorph transforms kernels without requiring reboots

https://github.com/multikernel/kmorph

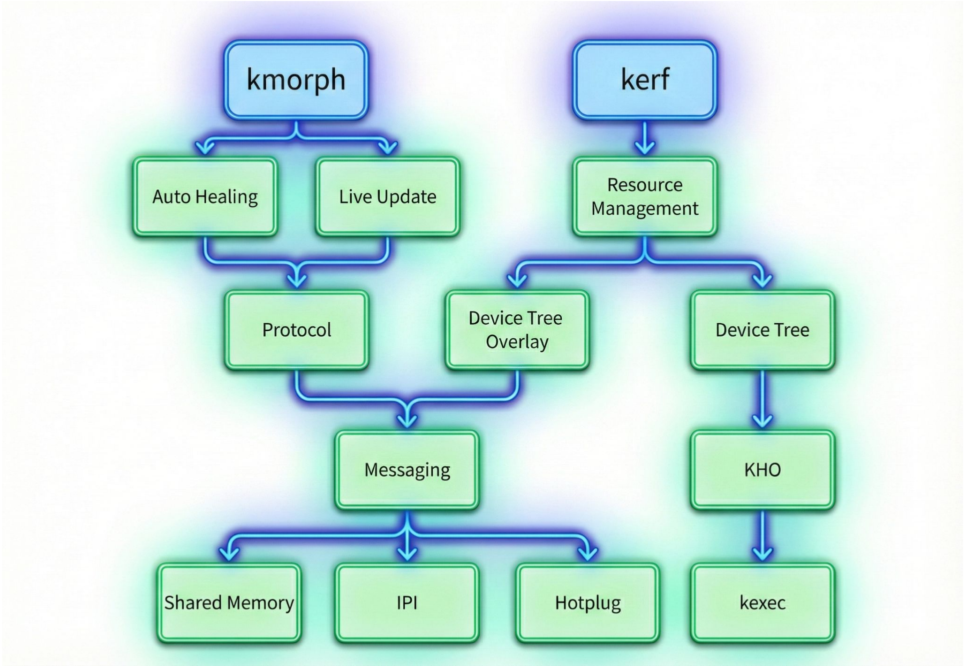
Beyond Kernel - KernelScript

KernelScript is a programming language unifying eBPF, kfunc, and user-space

KernelScript

https://github.com/multikernel/kernelscript

Multikernel Landscape



More Questions than Answers

1. How to *automatically* tailor a Linux kernel down to the minimum?
2. How to *automatically* optimize a Linux kernel for a specific application?
3. Could we use machine learning for kernel optimization?
4. Could we use LLM for kernel optimization?
5. How to implement zero-down kernel live update?
6. How to implement kernel switching for overcommitment?
7. How to pack the kernel together with the application?
8. How to support confidential computing?
9. Could Multikernel help evolve Linux into an Agentic OS?

Questions, Feedback and Collaboration

- Contact: cwang@multikernel.io
- Open Source Projects: https://github.com/multikernel
- Video Demo: https://www.youtube.com/@multikernel-tech
- Discord: https://discord.gg/32VtCfqd
- Join the mailing list: multikernel@lists.linux.dev

Credits

- Eric OKALA
- Yecan Zhu
- Ray Huang
- Yusheng Zheng
- Songtao Xue

東京 2025

LINUX PLUMBERS CONFERENCE

TOKYO, JAPAN / DECEMBER 11-13, 2025

Appendix - Agentic OS Workshop

- Co-located with ASPLOS 2026
- Call for Papers: https://os-for-agent.github.io/

Appendix - Related Papers

- *The Multikernel: A new OS architecture for scalable multicore systems*: https://www.sigops.org/s/conferences/sosp/2009/papers/baumann-sosp09.pdf
- *Factored Operating Systems (fos)*: https://www.princeton.edu/~wentzlaf/documents/Wentzlaff.2009.OSR.fos.pdf
- *Popcorn: a replicated-kernel OS based on Linux*: https://www.kernel.org/doc/ols/2014/ols2014-barbalace.pdf
- *Stramash: A Fused-Kernel Design Operating System for Cache-Coherent, Heterogeneous-ISA Platforms*: https://www.ssrg.ece.vt.edu/papers/asplos25.pdf

Appendix - Related Papers

- *Decoupling Cores, Kernels, and Operating Systems:* https://www.usenix.org/system/files/conference/osdi14/osdi14-paper-zellweger.pdf
- *On the Scalability, Performance Isolation and Device Driver Transparency of the IHK/McKernel Hybrid Lightweight Kernel:* https://bgerofi.github.io/papers/bgerofi-ipdps16.pdf
- *Linux vs. lightweight multi-kernels for high performance computing: experiences at pre-exascale*: https://dl.acm.org/doi/10.1145/3458817.3476162
- *Toward Full Specialization of the HPC Soware Stack*: https://bgerofi.github.io/papers/bgerofi-ross2017.pdf

Appendix - Related Papers

- *On Horizontal Decomposition of the Operating System*:
  https://openreview.net/forum?id=z50YPIcVKL
- *mOS: an architecture for extreme-scale operating systems*:
  https://dl.acm.org/doi/10.1145/2612262.2612263
- *K2: A Mobile Operating System for Heterogeneous Coherence Domains*:
  https://dl.acm.org/doi/abs/10.1145/2699676
- *The Composite Component Based Operating System*:
  https://courses.cs.umbc.edu/421/Spring12/02/slides/composite.pdf

Appendix - Related Papers

- *Achieving Performance Isolation with Lightweight Co-Kernels*: https://people.cs.pitt.edu/~jacklange/pubs/hpdc-2015-pisces.pdf
- *Covirt: Lightweight Fault Isolation and Resource Protection for Co-Kernels*: https://people.cs.pitt.edu/~jacklange/pubs/ipdps21-covirt.pdf
- *A Case for Transforming Parallel Runtimes Into Operating System Kernels*: https://dl.acm.org/doi/pdf/10.1145/2749246.2749264
- *Nested Kernel: An Operating System Architecture for Intra-Kernel Privilege Separation*: https://nathandautenhahn.com/downloads/publications/asplos200-dautenhahn.pdf

Appendix - Related Papers

- *Twin-Linux: Running independent Linux Kernels simultaneously on separate cores of a multicore system:* https://www.kernel.org/doc/ols/2010/ols2010-pages-101-108.pdf
- *Directvisor: Virtualization for Bare-metal Cloud*: https://dl.acm.org/doi/pdf/10.1145/3381052.3381317
- *Breaking the System Noise Barrier at Exascale:* https://dl.acm.org/doi/pdf/10.1145/3712285.3759793
- *Reproducible Performance Evaluation of OpenMP and SYCL Workloads under Noise Injection*: https://dl.acm.org/doi/pdf/10.1145/3731599.3767538