Introduction to Multikernel Linux

Cong Wang

Founder and CEO, Multikernel Technologies, Inc.

The Multikernel Idea

TL;DR History:

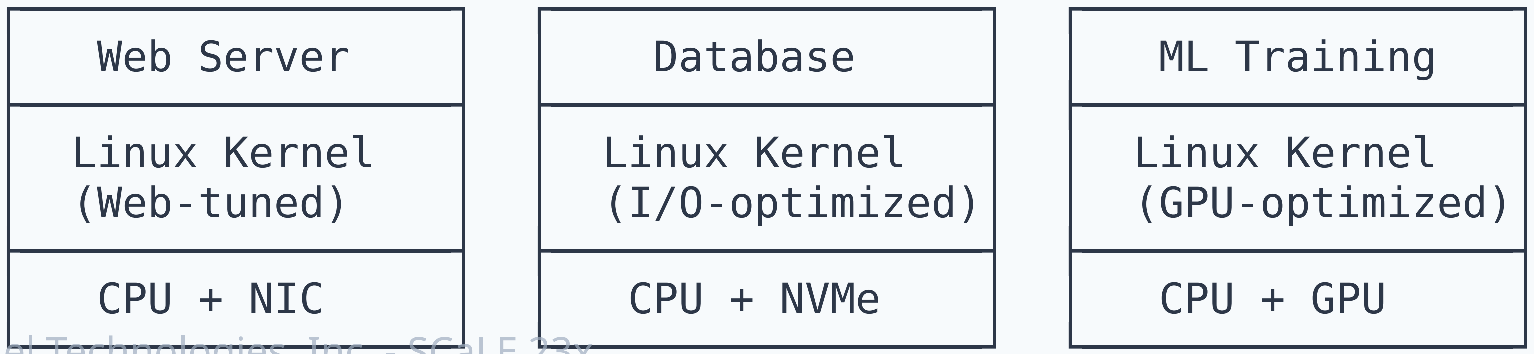
- **Barrelfish** (ETH Zurich, 2009) - New OS from scratch, never production-ready
- **Popcorn Linux** (Virginia Tech, 2013) - Research prototype for heterogeneous CPUs
- **McKernel** (RIKEN, 2015) - Custom kernel, only ~44% syscall coverage

What's different now? We use unmodified upstream Linux as every kernel instance - no new OS, no compatibility gaps, no rewriting your stack

Multikernel Linux

Core Concept: Run multiple Linux kernels simultaneously without virtualization

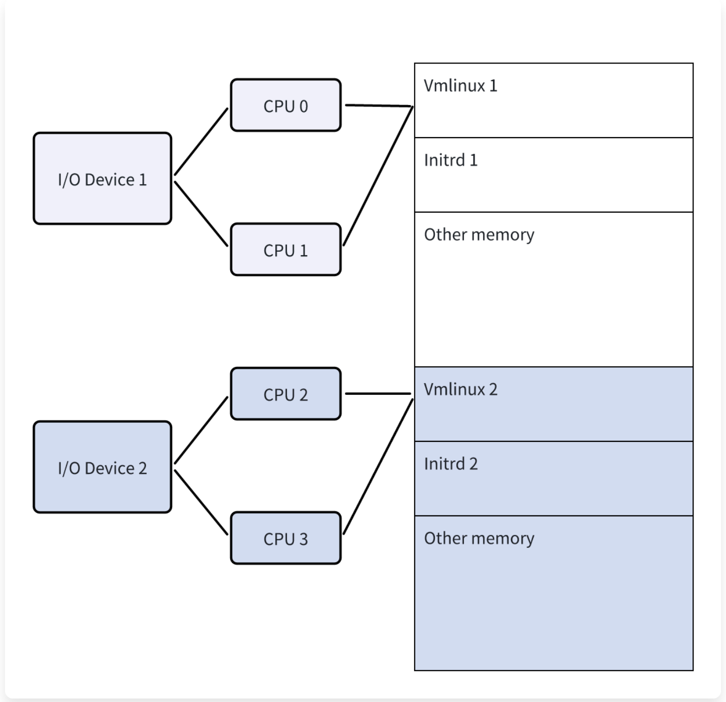
- **Isolation without VMs**: Each kernel runs independently on real hardware
- **Dedicated resources**: Each kernel gets its own CPUs, memory, and devices
- **Per-application kernels**: A tailored Linux kernel for each workload
- **Fully Linux**: No new OS to learn, fully compatible



How Does It Work?

- **Boots additional kernels**: Uses Linux's existing `kexec` mechanism to launch new kernels alongside the running one
- **No hypervisor needed**: Kernels run directly on hardware, not inside VMs
- **Resources are partitioned**: CPUs, memory, and devices are split between kernels
- **Dynamic rebalancing**: Resources can be moved between kernels at runtime via hotplug

Architecture Overview



Stronger Than Containers

Container limitations:

- All containers share one kernel, a kernel bug affects everyone
- "Noisy neighbor" problem: one container can slow down others

Multikernel advantage:

- **Kernel-level isolation**: Each workload gets its own kernel
- **No noisy neighbors**: Completely separate kernel schedulers
- **Tailored kernels**: Strip down Linux to only what each workload needs
- **Contained vulnerabilities**: A kernel exploit only affects one instance

Lighter Than VMs

VM limitations:

- Virtualization adds overhead (~5-20%)
- Running VMs inside cloud VMs (nested virtualization) is even worse
- Full OS per VM is heavyweight

Multikernel advantage:

- **Native performance**: No hypervisor layer, runs directly on hardware
- **Lightweight**: Only the kernel, no redundant OS layers
- **Simpler stack**: Direct hardware access, fewer things to go wrong

Native Docker Integration

Boot into Docker images directly

- Uses existing Docker images, no conversion needed
- DAXFS shares the container rootfs across kernel instances
- Boots directly into the Docker `ENTRYPOINT`, no OS init
- Similar workflow: pull an image, launch your service

Running AI Agents Safely

The problem: AI agents need isolation, GPU access, and fast snapshots - VMs are too heavy, containers are too leaky

Multikernel advantage:

- **Full GPU access**: No virtualization layer between your agent and the hardware
- **Strong isolation**: Each agent gets its own kernel, a rogue agent can't escape
- **Fast snapshots**: Lightweight kernel state means rapid checkpoint/restore
- **Works in cloud VMs**: No nested virtualization needed

Better Than Kexec

Traditional update: Reboot required, all applications interrupted

Multikernel approach:

1. Boot new kernel alongside the old one
2. Gradually migrate processes to the new kernel
3. Retire old kernel when done
4. Rollback possible at any point

Result: Zero application interruption - update your kernel without downtime

Better Than Kdump

Traditional approach: Kernel panic -> kexec reboot -> full reboot

Multikernel approach:

1. A backup kernel runs alongside your main kernel
2. It monitors the main kernel via heartbeat
3. If the main kernel crashes, the backup takes over its devices
4. Service resumes in **sub-second time**

Kmorph: Kernel Transformation

Kernel live update and auto-healing

- **Auto-healing**: Detects failures, triggers failover, reclaims resources, no reboot
- **Live kernel update**: Atomic resource handover with rollback
- **`kmorphd`** daemon monitors all kernel instances via heartbeat
- **`kmorphctl`** CLI for operations: `morph prepare`, `morph commit`, `morph rollback`

Multikernel vs Alternatives

	Linux	Containers	VMs	Multikernel
Isolation	None	Low	Strong	**Moderate**
Performance	Native	Near-native	-5-20%	**Near-native**
Kernel Customization	No	No	Yes	**Yes**
Dynamic Resources	N/A	Yes	Yes*	**Yes**
Zero-Downtime Updates	No	Yes*	Yes*	**Yes**
Attack Surface	Full	Full	Reduced	**Minimal**

*With orchestration overhead

Hardware Device Sharing

Challenge: You may have fewer devices (NICs, SSDs) than kernels

Our Solution: Share devices at the hardware queue level

- Modern NICs and SSDs have multiple hardware queues
- Each kernel gets exclusive access to specific queues
- Networking: Each kernel gets its own NIC queues and independent network stack
- Storage: Each kernel gets its own SSD queues and independent I/O path
- No software overhead, real hardware isolation

Filesystem Sharing: DAXFS

Challenge: Spawn kernels need access to files on the host

Our Solution: DAXFS - a writable filesystem built on shared memory

- Direct load/store on memory, no copying, no network
- Lock-free writes using hardware atomics
- Shared page cache, automatically visible to all kernel instances

DAXFS: Beyond Multikernel

Also works across CXL-connected hosts

- **LLM inference** - Share model weights across GPU instances; one copy serves all
- **CXL memory pooling** - Common filesystem across hosts with lock-free access
- **Container rootfs** - Shared base image, N containers = one copy in memory

Communication Between Kernels

Challenge: Applications running on different kernels need to talk to each other

Our Solution:

- Reuses `vsock` (Virtual Socket), a standard Linux socket type
- Kernels communicate over shared memory, no network needed
- Applications use familiar socket APIs, almost 100% compatible

Security Model

What you get:

- Kernel-enforced trust model
- Much smaller attack surface per kernel
- Kernel vulnerabilities are contained to a single instance

Best suited for:

- Running mixed workloads on the same machine
- Cloud environments where you need isolation without VM overhead
- Dedicated servers with strict security requirements

Kerf: Multikernel Orchestration

One tool to manage all your kernel instances

- Initialize resource pools (CPUs, memory, devices)
- Create, load, and run kernel instances
- Validates every operation, no invalid configurations allowed
- Topology-aware resource allocation

Multikernel
TECHNOLOGIES

Getting Started (1/2)

1. Clone and build the multikernel-enabled kernel:

```
git clone https://github.com/multikernel/linux.git
cd linux
make menuconfig  # Enable CONFIG_MULTIKERNEL
make -j$(nproc)
sudo make install
```

2. Boot with memory reserved for additional kernels:

```
mkkernel_pool=1023M@0x40000000
```

Add this to your bootloader (GRUB, etc.)

Multikernel
TECHNOLOGIES

Getting Started (2/2)

3. Install the kerf management tool:

```
git clone https://github.com/multikernel/kerf.git
cd kerf
pip install -e .
```

4. Launch your first kernel instance:

```
kerf init --cpus=4-31
kerf create web-server --cpus=4-7 --memory=2GB
kerf load web-server --kernel=... --initrd=...
kerf exec web-server
```

What's Next

- **Kernel switching**: Share CPUs across kernels via time-sharing
- **Checkpoint and restore**: Save and resume entire kernel states
- **Kernel live update**: Zero-downtime kernel upgrades with process migration
- **Kubernetes support**: Run multikernel instances as K8S pods
- **Confidential computing**: Hardware-backed security for multikernel

Get Involved

We didn't wait until it was "done" to open source

- Most projects open source after the design is set - ours is open from day one
- Your contributions shape the architecture, not just fix bugs
- Try it out, file issues, submit PRs
- Build it together with us

Summary

- **Multiple Linux kernels in parallel** - no VMs, no hypervisor
- **Stronger than containers** - kernel-level isolation, no noisy neighbors
- **Lighter than VMs** - native performance, no overhead
- **Zero-downtime updates** - better than kexec, better than kdump
- **Docker compatible** - use your existing images and workflow
- **100% upstream Linux** - full Linux compatibility

Feedback

Contact: cwang@multikernel.io

Open Source Projects: github.com/multikernel

Video Demo: youtube.com/@multikernel-tech

Discord: https://discord.gg/23f3EZHx

Join the mailing list: multikernel@lists.linux.dev