

Introduction to Multikernel

Cong Wang

Founder and CEO, Multikernel Technologies, Inc.

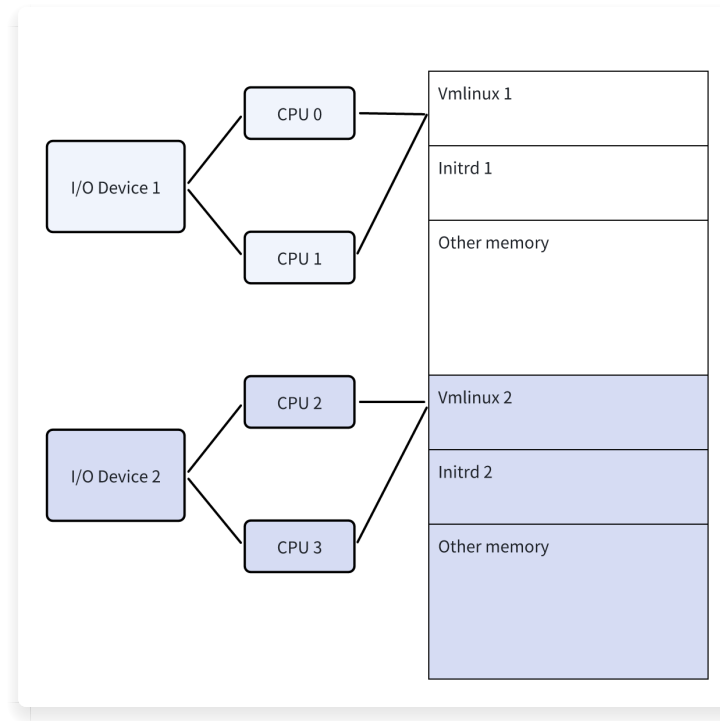
What is Multikernel?

Core Concept: Run multiple Linux kernels simultaneously without virtualization

- **Isolation over replication:** Each kernel is independent, not a replica
- **Dedicated resources:** Each kernel owns exclusive hardware partitions
- **Per-application kernels:** Tailored Linux kernel for each workload
- **Linux Compatible:** Full Linux compatibility

Web Server	Database	ML Training
Linux Kernel (Web-tuned)	Linux Kernel (I/O-optimized)	Linux Kernel (GPU-optimized)
CPU + NIC	CPU + NVMe	CPU + GPU

Architecture Overview



Spawning Kernels via `kexec`

- **New kimage type**: Introduced `KEXEC_TYPE_MULTIKERNEL` for spawning kernels without replacing the current one
- `kexec_file_load()` **based**: Leverages existing kexec infrastructure with kernel signing support
- **Direct 64-bit to 64-bit**: Uses vmlinux directly, no decompression or real-mode, no low 1M memory at all
- *Kexec HandOver*: Reuses KHO for cross-kernel resource passing

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

Why Device Tree?

- Established standard for hardware description
- Overlays enable dynamic runtime changes
- Kernel already provides solid Device Tree infrastructure

How Device Tree is used?

- Baseline: Available hardware resources
- Overlays: Dynamic instance changes
- Transactions: Rollback support

Kernel Filesystem Interface

```
/sys/fs/multikernel/  
├── device_tree           # Baseline resources (writable)  
├── overlays/            # Dynamic changes  
│   ├── new              # Write overlay here  
│   └── tx_101/          # Applied transaction  
│       ├── status       # applied | failed  
│       └── dtbo          # Overlay blob  
└── instances/           # Running instances  
    ├── web-server/  
    │   ├── device_tree  # Instance resources (read-only)  
    │   ├── id  
    │   └── status  
    └── database/
```

Dynamic Resource Allocation

Allocation Process:

1. Userspace generates overlay describing changes
2. Host kernel translates to resource update messages
3. Sends via IPI inter-kernel communication
4. Target kernel applies via hotplug operations

Leverages existing Linux infrastructure: CPU/memory/PCI hotplug

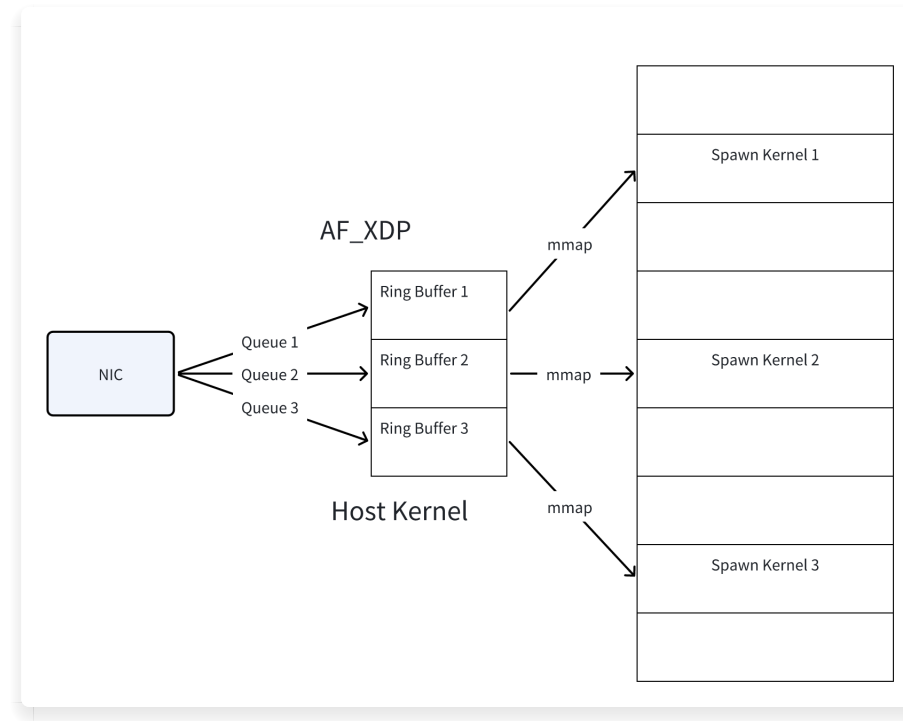
Hardware Device Sharing

Challenge: Devices are fewer than CPU cores

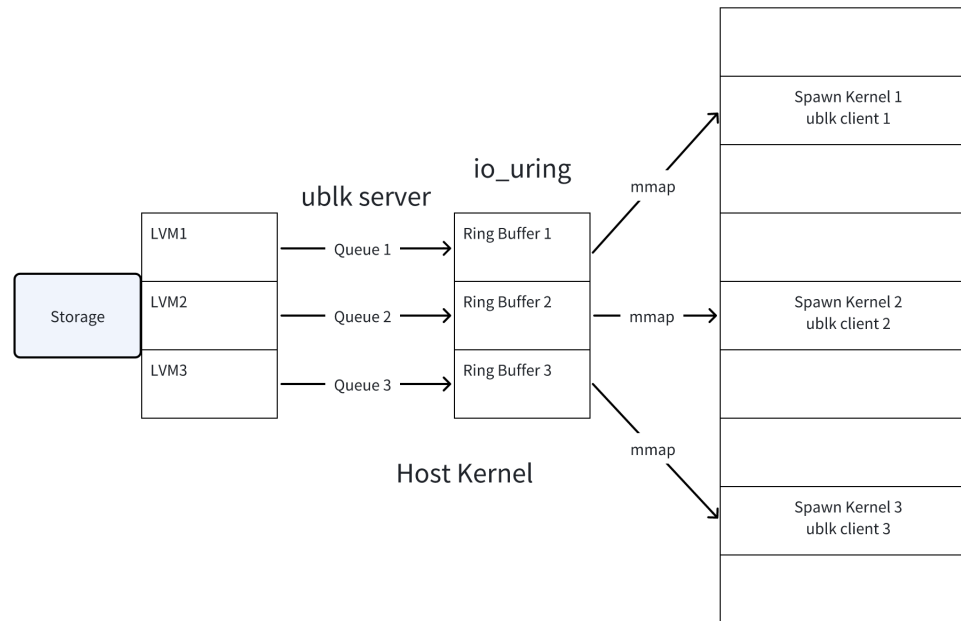
Our Solution: Hardware Queue Isolation

- Modern devices support multiple hardware queues
- Each kernel gets exclusive access to specific queues
- Leverages `AF_XDP` for networking, `ublk` for storage

Networking: Reuse **AF_XDP**



Storage: Reuse **ublk**



Filesystem Sharing: **DAXFS**

Challenge: Spawn kernels need access to host filesystem

Our Solution:

- **DAXFS**: shared filesystem on DAX memory, mounted concurrently by all kernel instances
- Zero-copy reads via direct load/store, lock-free CAS-based writes, no central coordinator
- Shared base image with per-instance COW overlay: reuse Docker container images directly
- Cooperative page cache: one copy in memory serves all kernels

Inter-kernel Communication: vsock

Challenge: Applications need to communicate across kernels

Our Solution:

- Reuse vsock (Virtual Socket) for inter-kernel communication
- New local vsock transport over IPI+shared memory
- Enables seamless service-to-service communication

Security Model

Implications:

- Each kernel trusted to respect its resource boundaries
- Reduced attack surface

Suitable for:

- Multi-tenant systems within same organization
- Cloud environments with controlled access
- Dedicated devices with strict kernel control

Trade-off: Performance over absolute security isolation

Use Case: Cloud - Virtual Machines

VM limitations:

- Modern virtualization overhead is minimal but not zero (~5%)
- Cloud instances are already VMs; further isolation requires nested VMs

Multikernel advantage:

- **Native performance:** No hypervisor, no nested virtualization
- **Lightweight:** Tailored kernels without full OS bloat
- **Simpler stack:** Direct hardware access, fewer layers

Use Case: AI Agent Sandbox

MicroVM limitations:

- Still requires virtualization layer
- Incompatible with cloud VMs due to nested virtualization constraints
- Hard to share memory

Multikernel advantage:

- **Native performance:** Continue using SR-IOV devices passed through from cloud VM
- **No nested virtualization:** Kernel-to-kernel isolation without hypervisor
- **Faster checkpoint/restore:** Lightweight kernel state for rapid agent snapshots

Use Case: Cloud - Containers

Container limitations:

- Containers share the same kernel (resource-level isolation)
- Noisy neighbor problem persists
- Kernel vulnerabilities affect all containers

Multikernel advantage:

- **Kernel-level isolation:** Each tenant gets its own kernel
- **Performance isolation:** Eliminates noisy neighbors completely
- **Tailored kernels:** Lightweight Linux optimized per workload
- **Security:** Kernel vulnerabilities contained to single tenant

Multikernel vs Alternatives

	Linux	Containers	VMs	McKernel	Multikernel
Linux Compatibility	Full	Full	Full	Partial	Full
Isolation	None	Low	Strong	Moderate	Moderate
Performance	Native	Native	-20-40%	Native	Native
Customization	No	No	Yes	No	Yes
Dynamic Resources	N/A	Yes	Yes*	Yes	Yes
Zero-Downtime Updates	No	Yes*	Yes*	No	Yes

*With orchestration overhead

Use Case: Live Kernel Update

Traditional update: Reboot required, applications interrupted

Multikernel approach:

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

Result: Zero application interruption

Use Case: Kernel Auto-Healing

Traditional approach: Kdump + double reboot

Multikernel approach:

1. Backup kernel monitors via heartbeat
2. Detects failure, takes over devices
3. Restores process state from saved memory
4. Resumes service in **sub-second time**

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 pool reserved for spawn kernels:

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

Add this to your bootloader (GRUB, etc.)

Getting Started (2/2)

3. Install kerf management tool:

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

4. Manage kernel instances:

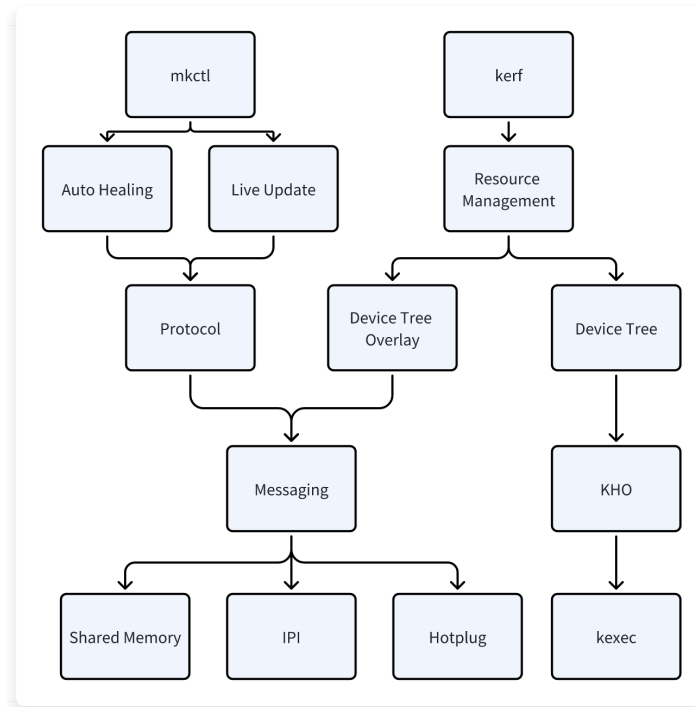
```
kerf init --cpus=4-31
kerf create web-server --cpus=4-7 --memory=2GB
kerf load web-server --kernel=... --initrd=...
kerf exec web-server
```

Kmorph Project

Transform between kernels without reboot

- **Live kernel update:** Migrate workloads from old kernel to new kernel seamlessly
- **Kernel auto-healing:** Detect failures and recover via backup kernel
- **Rollback support:** Return to previous kernel state at any point

Putting Together



Future Work

- **GPU Support:** Might already work via PCI hotplug
- **ARM/RISC-V Porting:** Targeting ARM and RISC-V servers
- **Checkpoint and restore:** Essential for AI agent state persistence and rapid recovery
- **Kubernetes support:** Implement `libkerf` to support K8S CRI
- **Kernel switching:** CPU overcommitment support via kernel time-sharing

Research Opportunities

Open questions we're exploring:

- Automatic kernel tailoring
- Automatic kernel optimizations based on profiles
- Machine learning based kernel optimizations
- Zero-downtime kernel upgrade
- Confidential computing for Multikernel

Call For Collaboration

What you get:

- Direct input on design decisions
- Co-authorship on research papers
- Free consulting during deployment
- Direct open source contributions
- Linux kernel community recognition

Questions and Feedback

Contact: cwang@multikernel.io

Open Source Projects: github.com/multikernel

Video Demo: youtube.com/@multikernel-tech

Discord: <https://discord.gg/KsGgrZMS>

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