I/O is faster than the OS

July 22, 2020

Nicolai Fischer

Friedrich-Alexander-Universität Erlangen-Nürnberg
Introduction
Introduction
Introduction

- I/O Devices are getting faster (200GbE NICs)
- CPU core counts increase, per core performance does not
- New smarter hardware can perform previous kernel tasks
- Legacy OS abstractions do no longer scale
Introduction

Legacy POSIX Abstractions
Portable Operating System Interface

- In development since over 30 years
- Common API between different UNIX/UNIX-like OSes
- Foundation for many software projects
Legacy POSIX Abstractions

[1]
Legacy POSIX Abstractions

POSIX usage in Ubuntu, Mac OS and Android

- API not fully implemented
- Lack of proper GPU and async I/O support
- System-call intensive API
- Security risks

Figure 1: POSIX function linkage (logscale both axis). Static analysis of (a) 374,463 Android apps with native libs and (b) 17,989 Ubuntu packages. Only a fraction of POSIX functions are ever linked.

[1]
Network Stack

Legacy Network Stack on Linux
- Kernel multiplexes devices
- Packets pass through kernel network stack
Legacy Network Stack on Linux

- Kernel multiplexes devices
- Packets pass through kernel network stack

**Problems**

- Per-packet memory allocation
- Copy between kernel and userspace
- Single CPU core is not fast enough
Network Stack

Kernel bypass with DPDK
Kernel bypass with DPDK

- Circumvent kernel entirely
- Exclusive access to hardware
- Dedicate CPU cores to network processing
- Low latency

Diagram:
- CPU 1
  - Process
  - Network Stack
  - Userspace
- CPU 0
  - Process
  - Network Stack
  - Userspace
- CPU 2
  - Process
  - Userspace

Connections:
- Queue
- NIC
- Kernel

Devices are unavailable to remaining system
Userspace drivers
Tighter coupling to specific hardware
Kernel bypass with DPDK

- Circumvent kernel entirely
- Exclusive access to hardware
- Dedicate CPU cores to network processing
- Low latency

**Downsides**

- Devices are unavailable to remaining system
- Userspace drivers
- Tighter coupling to specific hardware
Network Stack

eXpress Data Path
eBPF Programs

- Extended Berkely Packet Filter
- Access to the entire network packet and metadata
- Exit code determines route of packet
- Stateless between executions
XDP with Hardware Offload

- Kernel removed from the data-plane
- Zerocopy into userspace
- Device usable for legacy applications

![Diagram showing XDP with Hardware Offload]

- CPU 0
  - Process
  - Network Stack
  - Queue
  - eBPF

- CPU 1
  - Process
  - Network Stack

- Userspace
- Kernel
- NIC
Kernel removed from the data-plane
Zerocopy into userspace
Device usable for legacy applications

Caveats
Special hardware required (Smart NICs)
Driver support needed
Parakernel

Partition Devices
Partition Devices

- Eliminate kernel from data-plane
- Partition I/O devices that support it (eBPF)
- Only multiplex legacy hardware (SATA, Timers)
Parakernel

Multikernel Architecture
Multikernel Architecture

Monolithic Die

32C Die Cost
1.0X

EPYC MCM

4 x 8C Die Cost
0.59X

1. Based on AMD internal yield model using historical defect density data for mature technologies.

[3]
Multikernel Architecture

- Inspired by distributed systems
- Relatively independent OS instances on each CPU core
- Global state gets explicitly replicated
- Message passing instead of shared memory
- Highly scalable design
Parakernel

Eliminate Legacy Abstractions
Eliminate Legacy Abstractions

Asynchronous Kernel API

- No kernel threads, only Processes
- No blocking system-calls
- Application controlled concurrency (Coroutines, Fibers)
- POSIX compatibility through userspace libraries
Conclusion
Parakernel

- Very scalable
- Likely more secure
- Async design pattern supported by libraries, managed runtimes and modern languages
- POSIX compatibility in userspace
References
ATLIDAKIS, V., ANDRUS, J., GEAMBASU, R., MITROPOULOS, D., AND NIEH, J.
POSIX abstractions in modern operating systems: the old, the new, and the missing.

NVIDIA CORPORATION.

TIRAS RESEARCH, AND AMD.
AMD Optimizes EPYC Memory with NUMA.
Appendix

Zero-copy Architecture
Zero-copy Architecture

Motivation
- Avoid copying data between user and kernelspace

Zero-copy Architecture
- Share buffers between user and kernelspace
- Devices read/write data directly from/into them
- O_DIRECT flag in Linux

O_DIRECT
- Needs filesystem support
- Buffer alignment dependent on filesystem
- Circumvents filesystem cache
Appendix

io_uring
io_uring

Goals

- Zero-copy disk I/O
- Reduce context switches
- Asynchronous API

io_uring

- Ringbuffers shared between kernel and userspace
- Queue multiple I/O operations
- Use system-call to execute operations
- Alternative polling mode without system-calls
Appendix

Context-Switches are Expensive
Context-Switches are Expensive

**Meltdown**

- Trick CPU into executing specific instructions out-of-order/speculative
- Raise an exception
- CPU fails to wipe changed state correctly
- Use cache side-channels to extract arbitrary data

**Context Switches**

- More exploits: Spectre, Fallout, etc
- Software mitigation slow 2% to 11%
- Hardware solution partially available, but also expensive
- Context switches have to be avoided whenever possible
Appendix

Multi-Queue Devices
Multiplexing

I/O Devices

- Ring buffer of DMA-descriptors
- Write data into DRAM
- Interrupt informs OS of new data
- Kernel multiplexes device for applications
Multiplexing

Problems

- One CPU core is not fast enough
- Copying data in memory is too slow

Optimizations

- Write data directly into LLC
- Multi-Queue Devices
  - Up to 1535 queue pairs on Intel X710
  - Up to 65535 queue pairs in NVMe specification
Queues are processed by multiple CPU cores