UC San Diego

CSE 291: Operating Systems in Datacenters

Amy Ousterhout

Oct. 13, 2022

Agenda for Today

- Reminders
- Introduction to CPU scheduling
- Shenango discussion

Reminders

- Projects
  - More project ideas on Canvas tonight
  - Proposals due on 10/20
  - Talk to us if you want help brainstorming ideas

CPU Scheduling

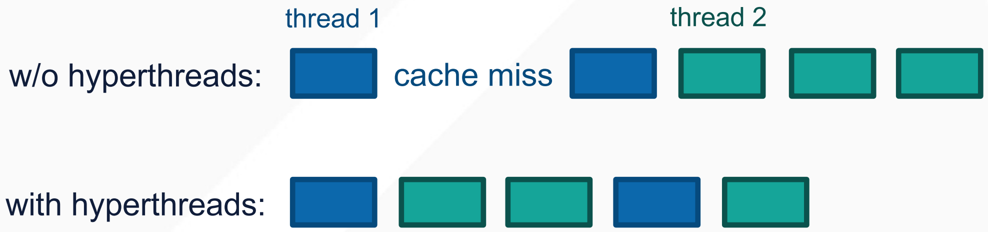
Handling Nanosecond-Scale Events

- 2017: “Attack of the Killer Microseconds”
- Hardware can efficiently handle nanosecond-scale events (e.g., cache misses, ~100 ns)
  - Out-of-order execution
  - Hyperthreads (simultaneous multithreading (SMT))
  - Prefetching
- Programmers don’t have to think about this

```
int var1 = *addr;
var1++;
var2++;
var3++;
var4++;
```

cache miss!

out-of-order
execution



Handling Millisecond-scale Events

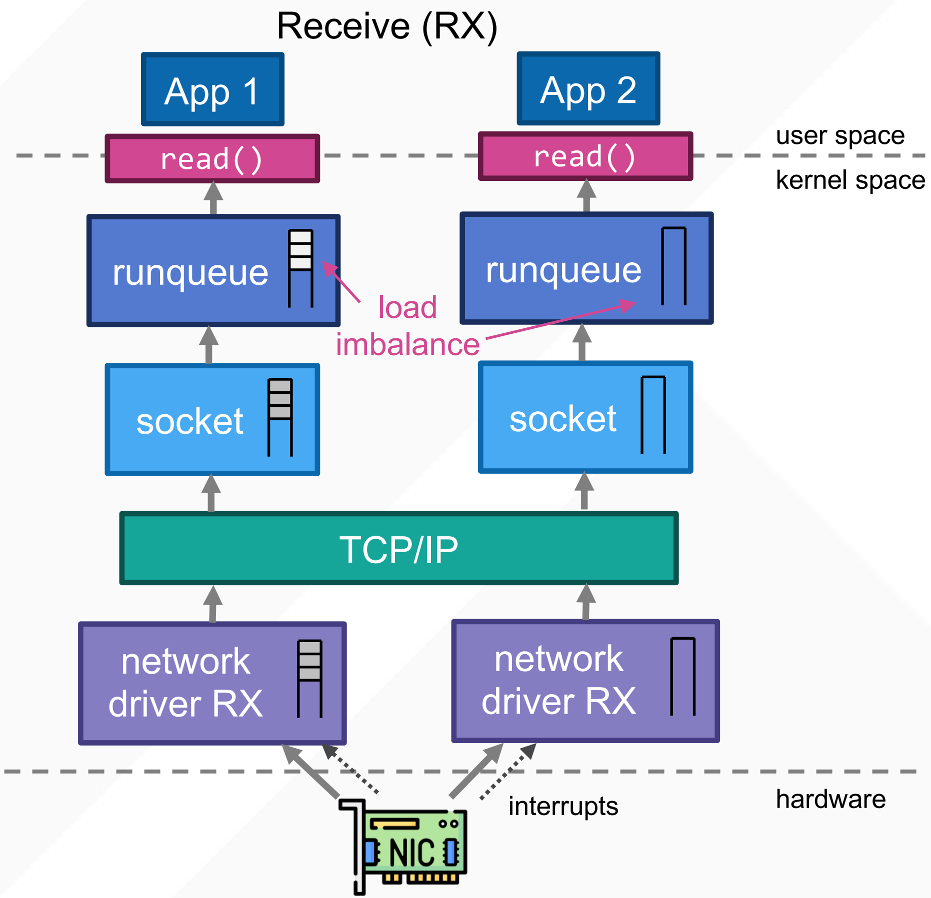
- Millisecond-scale events
  - Disk reads – 10s of ms
  - Wide-area network traffic – 10s of ms
  - Low-end flash – a few ms
- Software can efficiently mask these
  - OS can context switch to a different thread (microseconds)
- Programmers can use convenient synchronous (blocking) programming models

The Challenges of Microsecond-Scale Events

- But microsecond-scale events remain challenging
  - Datacenter RTT - a few µs
  - High-end flash – tens of µs
  - GPU/accelerator - tens of µs
- Hardware techniques do not scale well
  - Not enough independent instructions to fill µs
  - Not enough hyperthreads to hide µs
- Software techniques have too high of overhead
  - These are the killer microseconds!
- Asynchronous programming can reduce overheads but is inconvenient

How Does the OS Add So Much Overhead?

- Focus on the receive path
- Multicore example
- Sources of overhead:
  - Context switches
  - Lots of queueing
  - Load imbalance (balances runqueues every 4 ms)
  - Packets can arrive at the wrong core
  - Applications can interrupt each other
  - Hard to enforce policies



Research on CPU Scheduling

theoretical ⟷ practical

Theory

- Prioritization
- First come first served (FCFS)
- Shortest remaining processing time (SRPT)
- Process sharing (PS)
- Etc.

Kernel Bypass Scheduling

- ZygOS (SOSP ‘17)
- Arachne (OSDI ‘18)
- **Shenango (NSDI ‘19)**
- Caladan (OSDI ‘20)
- Scheduling Policies (NSDI ‘22)

Improve Linux’s Scheduling

- Snap (SOSP ‘19)
- **ghOSt (SOSP ‘21)**
- Syrup (SOSP ‘21)

Linux’s Scheduler (CFS)

Limitations

Theory	Kernel Bypass Scheduling	Improve Linux’s Scheduling	Linux’s Scheduler (CFS)
Assumes known task service times, no overheads, centralized queues	Require app changes, don’t support many policies or support multitenancy	Worse performance than kernel-bypass approaches	Lots of queueing, slow context switches, load imbalance, interference

Shenango Discussion