CSE 291: Operating Systems in Datacenters

Amy Ousterhout

Oct. 24, 2023
Agenda for Today

• Reminders
• Introduction to CPU scheduling
• Shenango discussion
Reminders

- Project check ins
  - Sign up if you have not already
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

```c
int var1 = *addr;  // cache miss!
var1++;            // out-of-order execution
var2++;            
var3++;            
var4++;            
```

with hyperthreads: thread 1 thread 2

w/o hyperthreads: cache miss
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**
  - 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**

- 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 queuing, slow context switches, load imbalance, interference
Shenango Discussion