Where’s the puck going?
Announcements

• Final: 12/11 @ 9AM, Barton Hall, 100 west
• Review session: 12/08, 2PM, zoom (post on Ed discussions)
• Lost sessions: thanks for using; makes me happy about my experiments
• Please fill out the course evaluations
  • Easy way to get 5%
  • Please be constructive (evaluations are for many eyes, not just me)
Taking 25 steps back!
What is an operating system, and what does it do?

A software layer designed with three goals:

- Enable applications to conveniently access hardware
- Manage all hardware resources
- Implement common services for applications
What does an OS do?

- Enables **convenient “abstractions”** for applications to access hardware
  - **CPU**: threads
  - **Memory**: virtual memory
  - **Storage devices**: files
  - **Network**: sockets
  - **Server**: collection of resources needed by an application
    - processes, VM, containers
What does an OS do?

• Enables **convenient “abstractions”** for applications to access hardware

• **Manages** hardware resources
  • Resource **allocation** to individual applications
  • Resource **sharing** across concurrently running applications
  • Resource **isolation** across concurrently running applications
What does an OS do?

- Enables **convenient “abstractions”** for applications to access hardware
- **Manages** hardware resources
- Implements **common services** for applications
  - Security, protection and authentication
  - Reliability
  - Communication
  - Input/output operations
  - Program execution
  - ....
Four Fundamental OS Concepts

• **Thread: Execution Context**
  • A single, sequential execution context

• **Address space (with translation)**
  • Program's view of memory is distinct from physical memory

• **Process: an instance of a running program**
  • Address Space + One or more Threads + ...

• **Protection/Isolation**
  • Only the “system” can access certain resources
  • Combined with translation, isolates programs from each other
Threads

• A single, sequential execution context

• A virtual core: provides illusion of infinite cores
  • Enables efficient multiplexing of physical cores...
    • ...across concurrently running applications

• Challenges in designing virtual cores
  • Scheduling, synchronization

• The OS provides protection/isolation at process granularity
  • Each thread has its own state
  • Can access other threads’ state (within the same process)
CPU scheduling

• Many different possible scheduling mechanisms

• FIFO, SJF, EDF, RR, SRTF
  • Some are preemptive, some are not preemptive
  • No one-size-fits-all solution

• Our focus: understanding tradeoffs (pros and cons) of each mechanism
  • And using the insights to build a near-ideal CPU scheduler
  • Very close to the Linux CFS scheduler

• Some conceptual takeaways that we studied
  • Priority scheduling can “emulate” most scheduling mechanisms
  • Priorities should be used to define physical core share
    • Rather than strictly preferential job scheduling
Synchronization

- Coordination between multiple...
  - ...threads within the same protection/isolation domain
  - ...processes and threads operating on shared data

- A hard problem
  - No “algorithm” to design a correct-by-design program

- Our focus:
  - Understanding the core challenges in synchronization
  - A suite of techniques that can be used
    - Locks, semaphores, condition variables, monitors
  - Hardware support for synchronization
Memory management

• Virtual address space: virtualizing physical memory address space
  • Enables efficient multiplexing of memory...
    • ...across concurrently running applications

• We focused on three aspects in memory management

• Efficient sharing of physical resources
  • Paging, and page replacement

• Space and time efficient address translation
  • Space efficiency: multi-level page tables
  • Time efficiency: TLB, small #levels in multi-level page tables

• Protection
  • Apps use virtual address, kernel handles physical addresses
Memory management

- Virtual memory: provides illusion of infinite memory
  - By swapping/paging data to secondary storage
  - Each program gets the illusion of having dedicated, infinite, memory

- Paging
  - Page faults
  - Page replacement mechanisms:
    - Optimal (Belady’s algorithm)
    - LRU
    - Approximating LRU: The clock algorithm
    - Working set page replacement
  - Local and global page replacement
Beyond threads, processes and memory

• The OS must handle all IO devices
  • Storage devices: HDD, SSD
  • Network devices: NIC
  • Peripheral devices: mouse, keyboards, ...
  • And all buses: memory bus, I/O bus, peripheral bus

• **Mechanisms**: Interrupt-driven I/O, DMA

• **Devices**: Mostly SSD, brief discussion on HDD
Beyond threads, processes and memory

• The OS must handle all IO devices
  • Storage devices: HDD, SSD
  • Network devices: NIC
  • Peripheral devices: mouse, keyboards, ...
  • And all buses: memory bus, I/O bus, peripheral bus

• OS support for handling storage devices
  • File systems
    • contiguous, linked list, tree-based multi-level index file storage
    • consistent updates
  • Block layer
  • Device drivers (minimal discussion)
Beyond threads, processes and memory

• The OS must handle all IO devices
  • Storage devices: HDD, SSD
  • Network devices: NIC
  • Peripheral devices: mouse, keyboards, ...
  • And all buses: memory bus, I/O bus, peripheral bus

• OS support for handling network devices
  • The entire “network stack"
  • End-to-end story
  • Various functionalities that interact with other layers
    • Sockets and ports
    • Packet steering, and tradeoffs
    • Packet aggregation, and tradeoffs
Taking 1 step forward!
Skate where the puck’s going, not where it’s been!

- Walter Gretzky
Where is the puck right now?

<table>
<thead>
<tr>
<th>Size (TB)</th>
<th>Random Access (us)</th>
<th>Seq. Access (GB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>0.1</td>
<td>80</td>
</tr>
<tr>
<td>1</td>
<td>25</td>
<td>1x</td>
</tr>
<tr>
<td>10</td>
<td>4000</td>
<td>0.1x</td>
</tr>
</tbody>
</table>
Where is the puck going?

- Memory bus (80 GB/s)
- PCIe (1x16 GB/s)
- SATA (0.05-0.1 GB/s)
- Ethernet (1.25 GB/s)
Where is the puck going? (CPU performance)
Where is the puck going?

- #Cores: +18-20%
- Per core: +10%

Diagram:
- CPU
- Memory bus
- PCIe
- SATA
- Ethernet

Text:
+30-32%
Where is the puck going? (DRAM capacity)
Where is the puck going?

Tape is dead, Disk is tape, SSD is disk, RAM is the king!
- Jim Gray
Where is the puck going? (Memory bus)
Where is the puck going?

Tape is dead, Disk is tape, SSD is disk, RAM is the king!

- Jim Gray
Where is the puck going? (PCIe)
Where is the puck going?

Tape is dead, Disk is tape, SSD is disk, RAM is the king!

- Jim Gray
Where is the puck going? (Ethernet)

+33-40%
Where is the puck going?

Tape is dead, Disk is tape, SSD is disk, RAM is the king!

- Jim Gray
Many powerful implications

- CPU is becoming the core bottleneck
  - Storage devices can achieve 10-100x higher throughput
  - NIC can transmit/receive 10-100x more packets
  - PCIe can transmit/receive 10-100x more data
  - But CPU capacity is mostly stagnant

- New devices are emerging
  - New hardware accelerators: for apps that require more compute
    - FPGAs, TPUs, SmartNICs
  - Non-volatile memory devices
    - byte addressable, but persistent
    - 10x slower than main memory, 10x faster than SSD
  - RDMA NICs
    - Can read/write to memory on other servers without CPU
CPU is becoming the bottleneck

• **Today, CPU involved in all steps**
  - Running applications
  - Kernel processing
  - I/O

• **Many of these are heavy-weight operations**
  - Thread and process state management
  - Context switches
  - Swapping and paging
  - Storage access
  - Network access

• Need to rethink design/optimization of each of these layers
Emergence of new devices [Compute]

• New hardware accelerators: for apps that require more compute
  • How should the OS enable sharing of accelerators?
  • How should the OS orchestrate traditional CPU and accelerator resources?
  • How should CPUs and accelerators share memory?

• Requires rethinking the abstractions developed over decades
  • Threads
  • Processes
  • Virtual address space, and virtual memory
  • Sockets
Emergence of new devices [Storage]

• Non-volatile memory devices
  • Byte-addressable—like main memory
  • Persistent—like SSDs
  • 10x slower than main memory, 10x faster than SSDs

• Requires rethinking the abstractions developed over decades
  • Virtual address space
  • Virtual memory
  • Page replacement
Remote Memory Faster than Local Storage

- **Under zero queueing:**
  - Remote memory access takes less than 6.3us
  - Local SSD access latency today is 25us (hardware, ignoring stack)
  - Remote Direct Memory Access (RDMA) becomes feasible
Emergence of new devices [Network]

• **Remote Direct Memory Access**
  • Enables accessing remote server memory....
    • ...without involving remote server CPU
  • “Kernel-bypass”: CPUs can read/write data to NIC without kernel

• **Requires rethinking the abstractions developed over decades**
  • Sockets
  • Protection/isolation
  • Virtual address space, and virtual memory
Operating Systems are the bottleneck again!

- Lot of research in "user space designs" and kernel-bypass
  - Minimize kernel involvement
  - Low-overhead CPU scheduling
  - Lots of interesting challenges

- Lot of research in low-overhead storage stack design
  - Revisiting File systems, virtual memory, block layer, ...
  - To minimize CPU utilization, to achieve low latency and high throughput
  - Extremely interesting challenges

- Lot of research in low-overhead network stack design
  - Revisiting the many layers within the network stack
  - To minimize CPU utilization, to achieve low latency and high throughput
  - Requires rethinking host architecture, and host network
  - One of the biggest challenges faced by the OS community
Closing Thoughts

• These are exciting times for operating systems
  • The first ever since the invention of SSDs!
  • You are witness to the transformation!!!!

• And, I am glad I got the chance to introduce you to this world :-)
  • You have made me a better teacher!!!!
  • Thank you.

• Wherever you end up:
  • Please remember me
  • Say hello if you see me
  • Remember, there is nothing more important than
    • Knowing the fundamentals!!!!
    • Being happy!!!!