Summer, 2005

Day 8
Multiple Processor Systems

Instructor: Neil Rhodes
Why?

Limits on improving CPU speed
- Speed of light: 1 foot/nanosecond

Add more CPUs working together

Some problems can’t be split
- If it takes 9 months for 1 woman to have a baby, can adding more women to the job speed it up?
  - Some problems inherently sequential
Overview

**Multiprocessors:** Multiple CPUs with shared memory
- Hardware
- Operating Systems
- Synchronization
- Scheduling

**Multicomputer:** Tightly-coupled CPUs with local memory
- Interconnection
- Communication software
- Distributed Shared Memory (DSM)
- Scheduling

**Distributed Systems:** Loosely coupled computers
- Networking
- Remote File Systems
- Shared Objects
- Coordination
Multiprocessors

Model:
- Multiple processors sharing a common memory
- Each processor can read/write to any memory location
  - One oddity: CPU can write a value to memory location, and read back a different value

Speed of accessing memory
- **Uniform Memory Access** (UMA): every memory word can be read as fast as every other memory word by any CPU
- **Nonuniform Memory Access** (NUMA): some memory is closer to some CPUs than others
UMA Multiprocessors (1)

**Bus**

- Local cache
  - Cache not just a single word, but a block of 32 or 64
  - Cache marked dirty or clean
  - When writing, must check with other caches to see whether they have same block in cache
    - If yes and dirty, other CPU must write dirty block to writing CPU
    - If yes and clean, other CPU must release block from cache

- Local memory
  - Used only for read-only data
    - Code, constants
  - Plus local variables
    - Data that need not be shared

- Problems: bandwidth of the single bus can be the bottleneck
  - Each CPU access memory every $X$ instructions.
UMA Multiprocessors (2)

Crossbar switches

- $n^2$ switches between memory and CPU
- Non-blocking: CPU 1 can access memory block 1 while CPU 3 is accessing memory block 2
- Downside: large number of switches
UMA Multiprocessors (3)

Switching network
- Can build up a network out of 2x2 switches
Switching network

- Omega network:
  - Usually use low-order bits of memory address as module index (spreads out memory access across memory modules)
NUMA Multiprocessors

Characteristics

- Single address space visible to all CPUs
- Access to remote memory is via 2 instructions
  - Load
  - Store
- Access to remote memory is slower than to local memory

![Diagram of NUMA multiprocessor architecture]

![Diagram of directory structure]

Directory at node 36
Multiprocessor Operating System Types (1)

Each CPU has its own copy of the OS
Multiprocessor Operating System Types (2)

Master-slave multiprocessors
Multiprocessor Operating System Types (3)

Symmetric multiprocessors (SMP)
Multiprocessor Synchronization

Implementing mutex

- Can’t disable interrupts
  - only affects current CPU
- Test-and-set requires two bus cycles
  - Another CPU may use bus in the middle
- Solution:
  - Lock bus
  - Do 2 memory accesses
  - Release bus
Multiprocessor Synchronization

Problem with Test and Set

- Waiting CPU constantly testing
- Potential solution: cache
  - Problem: waiting CPU does write; gets cache block containing lock
  - using CPU uses memory in cache block. When it reads, it gets cache block
- Better solution: read first
  - Only do Test and Set if it appears to be free
  - All reads can be cached
- Another solution: exponential backoff
  - If Test and Set fails to acquire lock, delay before trying again
  - Increase delay by a factor of two after each failure (up to some maximum)
- Yet another solution: private locks
Multiprocessor Synchronization

Spinning vs. switching

- When waiting for a lock, can either:
  - spin, waiting for lock
  - switch to another thread/process

- Tradeoffs
  - Spinning: wastes CPU time waiting for lock
  - Switching: wastes time:
    - saving current state
    - acquiring ready list lock
    - loading new state
    - Also, cache misses

- Best choice: whichever one wastes less time
  - How to know?
  - Spin for a while, then switch?
  - Look at history (how long have recent spin times on this lock been)
Multiprocessor Scheduling

Uniprocessor
- What process to run?

Multiprocessor
- What process to run?
- Where to run it?

Processes
- Related/unrelated
Multiprocessor Scheduling

Problems:

- Contention for single data structure
- Interrupted process may still hold spin lock
  - Smart scheduling: process sets a flag showing it holds spin lock
  - System doesn’t interrupt process holding spin lock, but wait a little more time to release the spin lock
- Caching
  - If process A ran on machine B last time, some of its data may still be in B’s cache
  - Prefer to rerun on same machine
- Two-level scheduling
  - When process is created, assign to a machine (stays there forever)
  - Each machine has its own ready queue
    - Good for caching
    - Contention for single ready list gone
    - If ready queue is empty, grab a process from another machine
Multiprocessor Scheduling

Space Sharing
- Related threads are scheduled together across multiple machines
  - Stay on machine until done
  - Machine idle if thread blocks on I/O

![Diagram showing CPU partitioning]

8-CPU partition
- 0 1 2 3 4 5 6 7
- 8 9 10 11 12 13 14 15
- 16 17 18 19 20 21 22 23
- 24 25 26 27 28 29 30 31

4-CPU partition
- 0 1 2 3
- 4 5 6 7

6-CPU partition
- 8 9 10 11
- 12 13 14 15

12-CPU partition
- 16 17 18 19
- 20 21 22 23

Unassigned CPU
- 24 25 26 27
- 28 29 30 31
Multiprocessor Scheduling

**Gang Scheduling**
- Groups of related threads scheduled as a *gang*
- All members of a gang run simultaneously (on different CPUs)
- All gang members start and stop their time slices together

![Gang Scheduling Diagram]

Thread $A_0$ running
### Multiprocessor Scheduling

#### Gang Scheduling

<table>
<thead>
<tr>
<th>Time slot</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$A_0$</td>
<td>$A_1$</td>
<td>$A_2$</td>
<td>$A_3$</td>
<td>$A_4$</td>
<td>$A_5$</td>
</tr>
<tr>
<td>1</td>
<td>$B_0$</td>
<td>$B_1$</td>
<td>$B_2$</td>
<td>$C_0$</td>
<td>$C_1$</td>
<td>$C_2$</td>
</tr>
<tr>
<td>2</td>
<td>$D_0$</td>
<td>$D_1$</td>
<td>$D_2$</td>
<td>$D_3$</td>
<td>$D_4$</td>
<td>$E_0$</td>
</tr>
<tr>
<td>3</td>
<td>$E_1$</td>
<td>$E_2$</td>
<td>$E_3$</td>
<td>$E_4$</td>
<td>$E_5$</td>
<td>$E_6$</td>
</tr>
<tr>
<td>4</td>
<td>$A_0$</td>
<td>$A_1$</td>
<td>$A_2$</td>
<td>$A_3$</td>
<td>$A_4$</td>
<td>$A_5$</td>
</tr>
<tr>
<td>5</td>
<td>$B_0$</td>
<td>$B_1$</td>
<td>$B_2$</td>
<td>$C_0$</td>
<td>$C_1$</td>
<td>$C_2$</td>
</tr>
<tr>
<td>6</td>
<td>$D_0$</td>
<td>$D_1$</td>
<td>$D_2$</td>
<td>$D_3$</td>
<td>$D_4$</td>
<td>$E_0$</td>
</tr>
<tr>
<td>7</td>
<td>$E_1$</td>
<td>$E_2$</td>
<td>$E_3$</td>
<td>$E_4$</td>
<td>$E_5$</td>
<td>$E_6$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CPU</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$A_0$</td>
<td>$A_1$</td>
<td>$A_2$</td>
<td>$A_3$</td>
<td>$A_4$</td>
<td>$A_5$</td>
</tr>
<tr>
<td>1</td>
<td>$B_0$</td>
<td>$B_1$</td>
<td>$B_2$</td>
<td>$C_0$</td>
<td>$C_1$</td>
<td>$C_2$</td>
</tr>
<tr>
<td>2</td>
<td>$D_0$</td>
<td>$D_1$</td>
<td>$D_2$</td>
<td>$D_3$</td>
<td>$D_4$</td>
<td>$E_0$</td>
</tr>
<tr>
<td>3</td>
<td>$E_1$</td>
<td>$E_2$</td>
<td>$E_3$</td>
<td>$E_4$</td>
<td>$E_5$</td>
<td>$E_6$</td>
</tr>
<tr>
<td>4</td>
<td>$A_0$</td>
<td>$A_1$</td>
<td>$A_2$</td>
<td>$A_3$</td>
<td>$A_4$</td>
<td>$A_5$</td>
</tr>
<tr>
<td>5</td>
<td>$B_0$</td>
<td>$B_1$</td>
<td>$B_2$</td>
<td>$C_0$</td>
<td>$C_1$</td>
<td>$C_2$</td>
</tr>
<tr>
<td>6</td>
<td>$D_0$</td>
<td>$D_1$</td>
<td>$D_2$</td>
<td>$D_3$</td>
<td>$D_4$</td>
<td>$E_0$</td>
</tr>
<tr>
<td>7</td>
<td>$E_1$</td>
<td>$E_2$</td>
<td>$E_3$</td>
<td>$E_4$</td>
<td>$E_5$</td>
<td>$E_6$</td>
</tr>
</tbody>
</table>
Multicomputers

Definition: tightly-coupled CPUs that do not share memory

- Also called:
  - Cluster Computers
  - Cluster of Workstations (COWS)

Topologies
Multicomputer Hardware

Store-and-forward packet switching:

Circuit switching:
- Establish circuit from CPU 1 to CPU 2, then send bits through circuit
Multicomputer Hardware

Network Interface Card (NIC)
Low-level Communication Software

If a user process needs to send packets
  ▪ Map network interface card to address space of process
    - (to avoid buffer copying)

If multiple user processes need to send packets
  ▪ Map NIC to address space of all processes
  ▪ Add mutex to mediate access to NIC buffers

If kernel also needs to send/receive packets
  ▪ Have a second NIC reserved for the kernel
User-Level Communication Software

Messaging
- Send(dest, &messagePtr)
- Receive(addr, &messagePtr)

Blocking vs. nonblocking

![Diagram showing blocking and nonblocking communication processes]

(a) Blocking process: Sender running, sender blocked, message sent, sender released.

(b) Nonblocking process: Sender running, trap, return, message copied to kernel buffer, message sent.
Remote Procedure Calls (RPC)

- Call a procedure on a remote machine

**Implementation issues:**
- Pointers/references (call-by-reference turns into copy-restore)
- Sizes of arrays, etc in weakly-typed languages
- Types of parameters (printf, for example)
- Global variables
  - copy
  - don’t use
Distributed Shared Memory

When referencing non-local page, trap causes DSM to retrieve page from machine that has it.
Pages distributed among 4 machines

After CPU 1 references page 10

If page 10 is read-only and replication is used
Multicomputer Scheduling

Load balancing: Assigning processors to nodes

- Given process that communicate with one another, find clusters of processes that communicate mostly within the cluster and not outside the cluster.
Load Balancing

Sender-initiated Distributed Heuristic
- Newly-created process starts on same node that created it
- But, if overloaded (by some metric)
  - Randomly ask another node its load
    - If low, send the process to it
    - If not, choose another node
    - Eventually, give up and keep the process

Receiver-initiated Distributed Heuristic
- When process finishes, system checks load
- If underloaded (by some metric)
  - Randomly ask another node its load
    - If high, receive a process from it
    - If not, choose another node
    - Eventually, give up and be underloaded

Bidding
- Processes buy CPU time
  - When creating process, get bids from nodes and choose best node
- Nodes auction off CPU time (specifying speed, features, etc).
Distributed Systems

Definition: loosely coupled computers that don’t share memory

<table>
<thead>
<tr>
<th>Item</th>
<th>Multiprocessor</th>
<th>Multicomputer</th>
<th>Distributed System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node configuration</td>
<td>CPU</td>
<td>CPU, RAM, net interface</td>
<td>Complete computer</td>
</tr>
<tr>
<td>Node peripherals</td>
<td>All shared</td>
<td>Shared exc. maybe disk</td>
<td>Full set per node</td>
</tr>
<tr>
<td>Location</td>
<td>Same rack</td>
<td>Same room</td>
<td>Possibly worldwide</td>
</tr>
<tr>
<td>Internode communication</td>
<td>Shared RAM</td>
<td>Dedicated interconnect</td>
<td>Traditional network</td>
</tr>
<tr>
<td>Operating systems</td>
<td>One, shared</td>
<td>Multiple, same</td>
<td>Possibly all different</td>
</tr>
<tr>
<td>File systems</td>
<td>One, shared</td>
<td>One, shared</td>
<td>Each node has own</td>
</tr>
<tr>
<td>Administration</td>
<td>One organization</td>
<td>One organization</td>
<td>Many organizations</td>
</tr>
</tbody>
</table>
The Internet
Network Hardware

Ethernet

- Signals propagate through the *ether*
- Carrier-Sense Multiple Access with Collision Detect (CSMA/CD)
- All NICs connected electrically
- When want to send, listen to see if carrier in use
- If not, start transmitting
  - If two machines transmit at once, each notices collision, waits random time (within range) and tries again
  - If still collision, exponential backoffs
## Network Services

### Connection-oriented
- Reliable message stream: Sequence of pages of a book
- Reliable byte stream: Remote login
- Unreliable connection: Digitized voice

### Connectionless
- Unreliable datagram: Network test packets
- Acknowledged datagram: Registered mail
- Request-reply: Database query

-
Distributed Systems

Middleware: a layer of software on top of the OS
- Provide consistency across multiple OS types

Diagram:
- Application
  - Middleware
  - Windows
  - Pentium
- Application
  - Middleware
  - Linux
  - Pentium
- Application
  - Middleware
  - Solaris
  - SPARC
- Application
  - Middleware
  - Mac OS
  - Power PC

Network
File System-based Middleware

Looks like great big filesystem

Transfer model
- upload/download
- Remote access
File System-based Middleware

**Naming**

- Machine + path naming
  - `/machine/path`
- Mounting remote file system on local file hierarchy
- Single name space looks same on all machines
Shared Object-based Middleware

Common Object Request Broker Architecture (CORBA)
Coordination-based Middleware

Linda

- processes communicate via *tuple-space*
- tuple, ordered list of typed data
  - ("abc", 2, 5)
  - ("matrix-1", 1, 6, 3.14)
  - ("family", "is-sister", "Stephany", "Roberta")

Operations:
- *out*: puts tuple into tuple space
- *in*: retrieves tuple from tuple space and removes it (atomic, can use wildcards)
  - block if required tuple not present
- *read*: like *in*, but doesn’t remove tuple from tuple space
- *eval*: used for creating parallel processes
Coordination-based Middleware

Jini

- Devices join network and register services
- Same idea as Linda of tuple-spaces: JavaSpaces
  - Write: put a new entry into the JavaSpace
  - Read: copy an entry that matches a template
  - Take: copy and remove entry that matches a template
  - Notify: notify caller when a matching entry is written
- Read and take can have timeout