Multiple Processor Systems

Today

- Basic hardware and OS issues in
- Multi-processors
- Multi-computers
- Distributed systems
Multiple processor systems

- We always need more computation power
- Making clocks run faster won’t do it for long
  - Special theory of relativity limits signals’ speed to ~20cm/nsec on wire
    - 1GHz computer – signals cannot travel more than 20cm within one cycle (10⁹ cycles/second, 10⁹ nanoseconds/sec)
    - 10GHz – 2cm, 100GHz – 2mm, …
  - Even if we can make them that small, temperature/heat dissipation is another issue

- Alternative – bring multiple machines together either as massively parallel machines or, over the Internet
  - Different organization for multiple computers – in the time, distance scales and logical organization
Shared-memory systems

- Shared-memory – Nodes have equal access to memory, typically managed as paged virtual address space
- Uniform or non-uniform memory access (UMA and NUMA)
  - How to interconnect more machines given hardware and budget limitations
  - UMA – Bus-based (a few 10s), cross-bar or multistage switching (still need quite a few switches)
  - NUMA – Add local memory and forget uniform access time!
    - With/without caching to hide differences; which must be somehow kept consistent
  - Multicore chips – sometimes called chip-level multiprocessors (CMPs)
Multiprocessor OS types

- Each CPU with its own OS
  - Simplest, but poor performance
    - E.g. Disk blocks? Each OS has its own cache – inconsistency;
      Nor caches – good for consistency, bad for performance

- Master-slave
  - Solves synchronization issue – a master holds the one copy
    of OS and tables, and distributes tasks among slaves
  - Problem: With a few slaves, master becomes a bottleneck

- Symmetric multiprocessing (SMP)
  - One copy of the OS, but any CPU can run it
    - On a sys call, the CPU where it was made traps to the kernel
      and processes it
  - Problem – need to synchronize access to shared OS state
Multiprocessor OS issues – Synchronization

- CPUs in MPs frequently need to synchronize
  - E.g. when accessing kernel critical regions
- Disabling interrupts won’t do
- TSL – you need to lock the bus to implement it
  - But spin-lock blocks everybody trying to use the bus
  - And cache invalidation generates a lot of traffic between holder and requesters of the lock
  - Test (read) before TSL?
- Spinning or switching
  - Testing repeatedly is not productive and CPU switching wastes cycles – can you learn from past behavior (how long did you spin last)?
Multiprocessor OS issues – Scheduling

- Now bi-dimensional – which thread *and* where?
  - Timesharing – Simplest, a single ready queue
    - Contention for shared data structure
    - Support affinity to preserve state?
  - Space sharing
    - Split CPUs dynamically to run related threads at the same time in different slices

- Gang scheduling – Time and space
  - Group of related threads in gangs, scheduled as a unit
    - Gang members start and end their time slices together
  - All members of a gang run simultaneously, on different timeshared CPUs
    - So, communicating members can request/reply almost immediately
Message passing multicomputer

- CPU-Mem pairs interconnected by a high-speed network (aka, clusters or COWS)
- Local memory can be accessed by local CPU
- Between CPUs, a high-speed interconnect
  - Different organizations – single switch, ring, mesh, double torus, hypercube …
  - Store-and-forward or circuit switching
  - Various types of network interfaces – RAM, DMA, network processors, two of them …
  - Key issue – avoid unnecessary copying of messages
Multicomputer OS issues – Communication

- No shared memory, communication via msg passing
  - Easier to build, harder to program
- Send & receive, at the barest minimum
  - How to address a target node? cpu# and port#?
  - Send/receive blocks thread until msg is sent/received

Sender running

Sender blocked

Sender running

Trap to kernel
sender blocked

Msg transfer

Return from kernel
sender released
Multicomputer OS issues – Communication

- Send & receive, at the barest minimum
  - Non-blocking sender

Sender running → Sender blocked → Sender running

- Trap to kernel
- Sender blocked
- Msg copied to kernel buffer?
- Return from kernel
- Sender released
- Msg transfer

- Sender can go on with its work, but what about the unsent msg? Copy to kernel space? $$, COW?

- Non-blocking receive
  - Interrupts – costly, slow and complicated
  - Poll for arrived messages – how often?
  - Pop-up threads and active messages
Remote Procedure Calls

- Msg passing is I/O, for many, the wrong programming model
- RPC (Birrell and Nelson ‘84) – simple idea with subtle implications
  - Allow a process to call procedures in another machine
- To make it look like a procedure call - client and server stubs that pack and unpack the parameters and deal w/ the call

Some problems

- Pointers, weakly typed languages (array length?), ...
- Global variables (now not longer shared)
Multicomputer OS issues – Memory

- Many programmers still prefer shared memory
  - Distributed shared memory without physically sharing it

- Basic idea
  - Each page is located in one of the memories
  - Each machine has its own virtual mem and page tables
  - When a page fault occurs, OS locates the page and requests it from the CPU holding it
    - Basically going to remote memory rather than disk.

- Improving performance
  - Replication of read only pages – easy
  - Replicating read/write pages – special handling to ensure consistency when pages are modified
  - False sharing
  - Failing machines …
Multicomputer OS issues – Scheduling

• Scheduling done per node, each has its own set of processes – easier
  – But allocation of processes to nodes is key

• Allocation can be done in a number of ways
  – A graph-theoretic approach – split graph (of processes/traffic exchanged) into tightly coupled clusters and them to nodes
  – More distributed models
    • Sender-initiated – at process creation, if origin node is overloaded, pick a random node, if load is bellow threshold, move it there
    • Receiver-initiated – if a node loads is too low, pick another one at random and ask for extra work
    • At runtime – load balancing can be done at run-time if you expect a process to last for long enough to justify migration
And now a short break ...

I've got a bunch of virtual Windows machines networked together, hooked up to an incoming pipe from the net. They execute email attachments, share files, and have no security patches. Between them, they have practically every virus.

There are mail trojans, Warhol worms, and all sorts of exotic polymorphics. A monitoring system adds and wipes machines at random. The display shows the viruses as they move through the network, growing and struggling.

You know, normal people just have aquariums.

Good morning, Blaster. Are you and W32.Welchia getting along?

Who's a good virus? You are! Yes, you are!
Distributed systems

- Very broad definition
  - Collection of independent, interconnected processors that communicate and coordinate through message exchanges
  - A collection of independent computers that appears to its users as a single coherent system

- Loosely-coupled - each system is completely autonomous
  - Each host runs an independent autonomous OS
  - Computers don’t really trust each other
  - Some resources are shared, but most are not
  - The system may look differently from different hosts
  - Typically, communication times are long
Distributed systems – what for?

- Resource sharing – both, physical resources and information
- Computation speedup – to solve large problems, we will need many cooperating machines
- Reliability – machines fail frequently
- Communication – people collaborating from remote sites
- Many applications are by their nature distributed (ATMs, airline ticket reservation, etc)
Distributed systems challenges

- Making resources available
  - The main goal of DS – making convenient to share resources

- Scalability
  - In numbers (users and resources), geographic span and administration complexity

- Providing transparency
  - Hide the fact that the system is distributed
  - Some types of transparency
    - Location – Where’s the resource located?
    - Replication – Are there multiple copies?
    - Concurrency – Is there anybody else accessing the resource now?
    - Failure – Has it been working all along?
  - How much transparency?
Distributed systems challenges

- **Openness**
  - Services should follow agreed-upon rules on component syntax & semantics

- **Security**
  - Sharing, as always, introduces security issues

- **Adding to the challenges, common false assumptions**
  - The network is reliable / secure / homogenous
  - The topology does not change
  - Latency is zero / Bandwidth is infinite / Transport cost is zero
  - There is one administrator
The network underneath

- Connecting hosts in a distributed system – a network
  - A hierarchical system (used to be) organized by geographic proximity

- Lowest level – Local Area Network (LAN)
  - Most popular LAN technology – Ethernet
    - Ethernet segment – hosts connected through adapters and wires to a hub; segments can be bridge
    - Packetized – fixed packets (frames) include header + payload
    - Broadcast network – Carrier Sense Multiple Access with Collision Detection (CSMA-CD)
    - Each adapters has a globally unique address (MAC)
The network underneath

- Higher in the hierarchy, LANs connected through specialized computers *routers* to form an internet
- Internet Protocol (IP) – network layer
  - Routes packets across multiple networks, from source to destination
  - Every computer has a unique Internet address (IP address)
  - Individual networks are connected by routers that have physical addresses (and interfaces) on each network
From names to IP addresses

- Large integers (IP) are hard to remember
- To make it easier, use a set of human friendly domain names
  zappa.cs.northwestern.edu (165.124.180.8)
- Originally mapping was kept in a text file (HOSTS.TXT)
- Since 1998 through a distributed database – Domain Name System (DNS)
From source to destination

- Internet is made of autonomous systems (AS)
- Routing within and between ASes
  - Within, potentially different routing algorithms but commonly OSPF – Open Shortest Path First
  - Between, BGP – Border Gateway Protocol
- Collectively let a node find physical address on the network of a router that can get a packet one step closer to destination
The network is unreliable

- Packets get lost or corrupted on the way
  - Flipped bits, links down, buffers overflow, …

- One approach, ignore it – the application can handle it
  - UDP/IP

- For reliability
  - How does the sender knows the receiver got it?
    - ACK
  - What if the ACK gets lost?
    - Timeout
  - What if the ACK just gets delayed?
    - Counters
  - …
Reliable transport on an unreliable network

- Transmission Control Protocol (TCP)
  - Reliable multi-packet messages from unreliable single-packet datagrams

- With TCP/IP and lower layers we get
  - Multi-packet messages
  - Reliable delivered between
  - Two address spaces in two different machines over heterogeneous networks
  - All without caring for the details

- Higher protocol layers facilitate specific services
  - SMTP for email, HTTP for Web, FTP for file transfer, …
Now that you can communicate …

- How do you order events happening in different machines that don’t share a clock?
  - Keeping clocks somewhat synchronized
  - Ignore real-time clocks, just logical ordering could be enough

- How do you ensure mutual exclusion?
  - E.g., Token passing, voting, …

- Replication for fault tolerance or performance
  - Where do you place replicas?
  - How do you keep multiple replicas consistent?

- …

- A whole class just to get you started …
More? Take EECS 340, 345 or 3/495s

- 340 – Introduction to networking
  - The nitty-gritty details of allowing machines to talk to each other

- 345 – Distributed systems
  - An introduction – naming and finding things, time and the logical order of events, self destructing data for increased privacy …

- 3/495 – Distributed systems in challenging environments
  - Hot and crazy ideas when pushing distributed systems to uncomfortable spaces (outer-space, under-water, on a volcano, …)

- 3/495 – Internet system experimentation
  - Designing, building and measuring large-scale distributed systems and their underlying network