Multiple Processor Systems

Today

- Multiprocessors, multi-computers and distributed systems
- Basic hardware and OS issues
- Distributed file systems

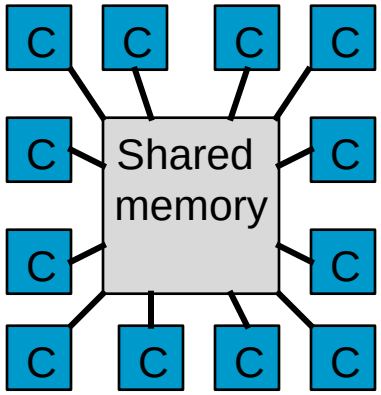
Multiple processor systems

- We always need more computation power
- Simply making the clock run faster won't do it for long
  - Einstein's special theory of relativity limits signals' speed to C in vacuum (30cm/nsec or 20cm/nsec on wire)
    - 10GHz computer, signals cannot travel more than 2cm within one cycle
    - 100GHz – 2mm
  - And that's not it – temperature/heat dissipation is another issue
- Alternative, bring multiple machines together either as massively parallel machines or, through the Internet

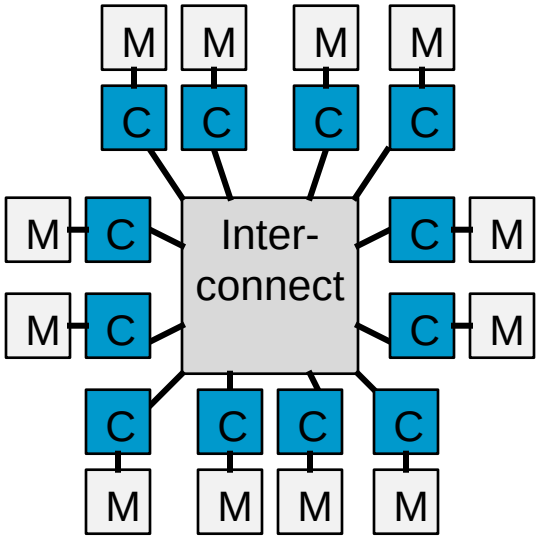
Multiple processor systems

- All communication between nodes comes down to message exchange
- Differences are in the time and distance scales and logical organization

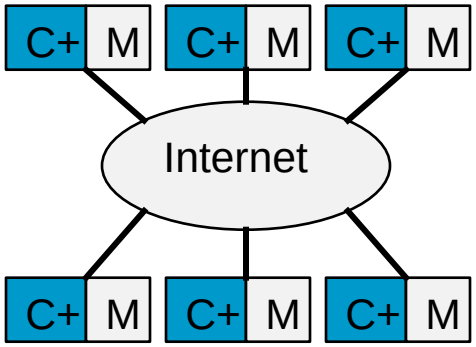
Shared memory multiprocessor



Message passing multicomputer

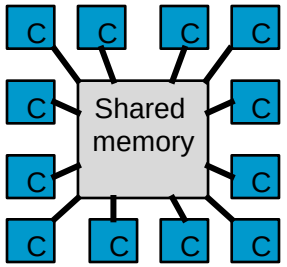


Distributed system



Shared memory multiprocessor

- Nodes have equal access to memory, typically managed as paged virtual address space
- Two main classes – Uniform or nonuniform memory access (UMA and NUMA)
  - How to interconnect more machines given hardware and budget limitations
  - UMA – bus-based (a few 10s), cross-bar switching (still need n^2 switches) or multistage switching (omega network – $n/2 * \log_2 n$ switches)
  - NUMA – forget uniform access time! With/without caching to hide differences
    - A popular model for Cache Coherent NUMA (CC-NUMA) is directory-based where each node keeps track of its cache lines
  - Multicore chips – sometimes called Chip-level MP



Multiprocessor OS types

- Each CPU with its own OS
  - Simplest, but poor performance
    - E.g. what do you to w/ disk blocks? If each OS has its own cache of recently used disk blocks – inconsistency; solution: eliminate buffer 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: can't handle too many slave processors before master becomes a bottleneck
- Symmetric multiprocessing
  - One copy of the OS, but any CPU can run it
    - When a sys call is made, the CPU where it was made traps to the kernel and processes the sys call
  - Problem - you need to synchronize access to shared OS state

Multiprocessor OS issues – Synchronization

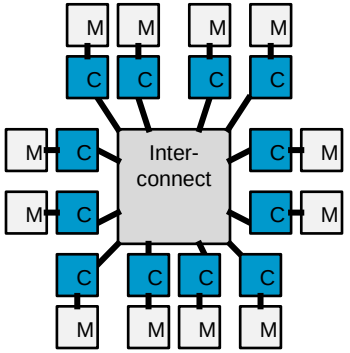
- Disabling interrupts won't do
- TSL – you need to lock the bus to implement it
- But now spin-lock blocks everybody trying to use the bus
  - Test before TSL?
  - Another option - chained list of waiting CPUs, each testing its own private lock (Mellor-Crummey & Scott)
- Spinning or switching
  - If you can choose, which one?
  - 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
  - Obvious problem of 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 related threads in gangs
  - Space shared within gang so all gang members run at once, timeshare between gangs

Message passing multicomputer

- Number of CPU-Mem pairs interconnected by a high-speed network
- Local memory can be access by local CPU
- Between CPUs a high-speed interconnect
  - Organized in a variety of ways – 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

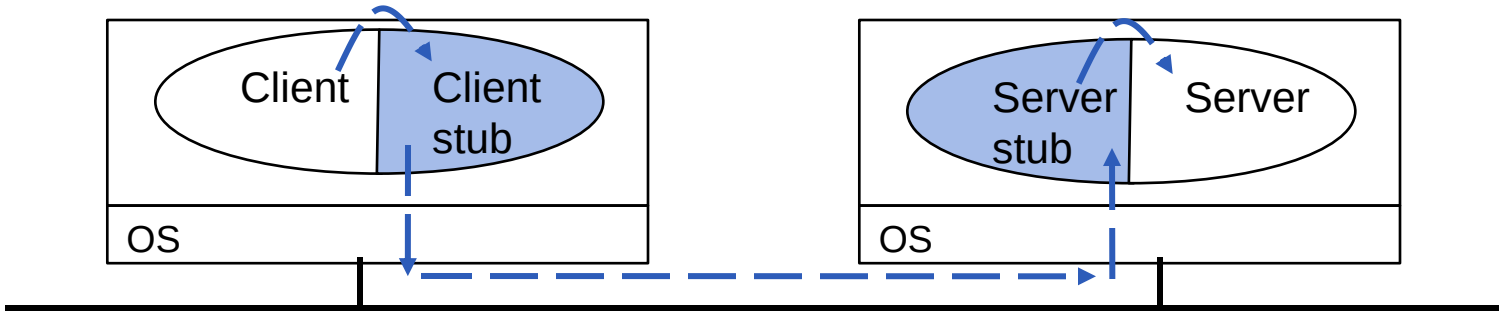


Multicomputer OS issues – Communication

- No shared memory, communication via msg passing
  - Easier to build but harder to program
- Send & receive
  - How to address a target node? cpu# + port# could be enough
  - Send/receive blocks caller/sender until msg is sent/received
  - Non-blocking calls – sender can go on with its work, but what about the unsent msg? Sender cannot modify the buffer
    - Copy into kernel space? $$
    - Interrupt to notify? Tricky programming
    - Copy on write? The copy may ended up being wasted
  - Nonblocking receive
    - Interrupts – costly, slow and complicated
    - Poll for arrived messages – how often?
    - Pop-up threads and active messages

Multicomputer OS issues – Communication

- 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

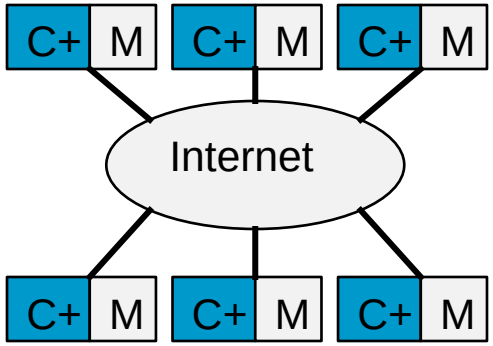
- Many programmers still prefer shared memory
  - Distributed shared memory without physically sharing it
- Basic idea
  - Each page is located in one of the memories
  - Each machine has its own virtual mem and page tables
  - When a page fault occurs, OS locates the page and requests it from the CPU 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
  - Maintaining consistency
  - False sharing

Multicomputer OS issues – Scheduling

- Scheduling 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.
  - Given a graph of processes and traffic – partition graph into tightly coupled clusters with little interaction with the rest; assign clusters to nodes
  - More distributed models
    - Sender-initiated – at process creation, if origin node is overloaded, pick another one at random and if load is bellow some threshold, move it there
    - Receiver-initiated – if a node loads is too low, pick another one at random and ask for extra work.
    - Load balancing does not have to be only at creation time if you expect the processes to last for long enough to justify migration

Multiple processor/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 CPU 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 – motivation

- 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
- Security
  - Sharing, as always, introduces security issues
- Providing transparency
  - Hide the fact that the system **is** distributed
  - Types of transparency
    - Access – What's data representation?
    - Location – Where's the resource located?
    - Migration – Have the resource moved?
    - Relocation – Is the resource being move?
    - Replication – Are there multiple copies?
    - Concurrency – Is there anybody else accessing the resource now?
    - Failure – Has it been working all along?
  - Do we **really** want transparency?

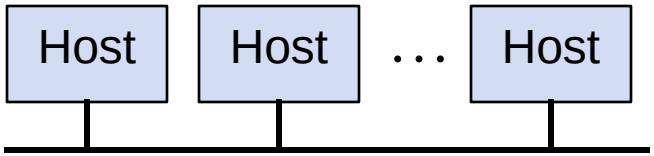
Distributed systems challenges

- Openness
  - Services should follow agreed-upon rules on component syntax & semantics
- Scalability
  - In numbers (users and resources), geographic span and administration complexity
  - Some useful techniques
    - Asynchronous communication, distribution, caching/replication
- 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

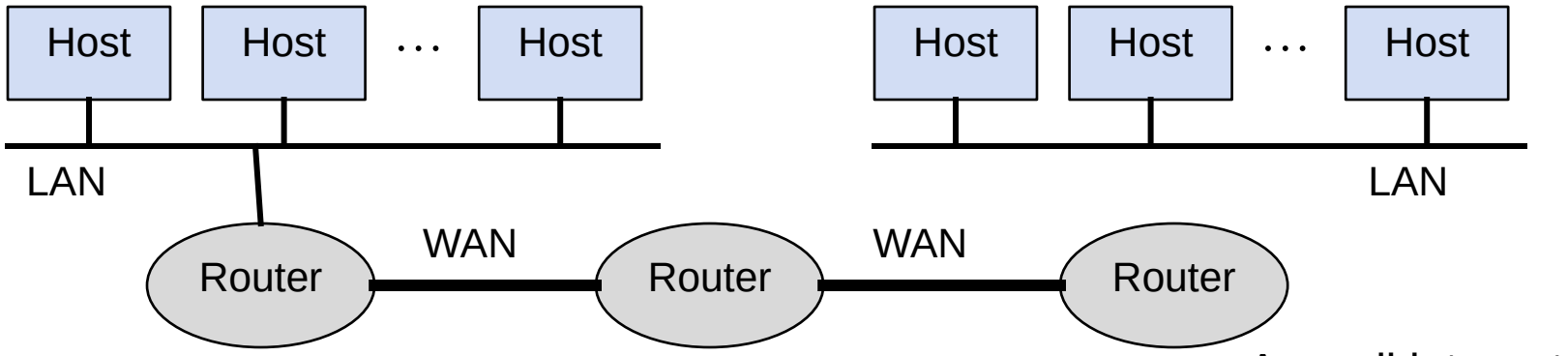
- A network is a hierarchical system organized by geographic proximity
- Lowest level – Local Area Network (LAN)
  - Most popular LAN technology – Ethernet
  - Ethernet segment – hosts connected through adapters and wires to a hub; segments can be bridge

Conceptual view



  - 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)
    - Interface listens for its address, interrupts OS when a packet is received

The network underneath



A small internet

- 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

The network underneath

- Getting 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
- Large integers (IP) are hard to remember
  - To make it easier, the Internet defines a set of human friendly domain names and the mechanism to map it to IPs
    - 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)

The network underneath

- Transmission Control Protocol (TCP) – transport layer
  - Manages to fabricate reliable multi-packet messages out of unreliable single-packet datagrams
  - Analogy: sending a book via postcards – what's required?
- Summary – 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, …

Distributed file systems

- A distributed file systems supports network-wide sharing of files and devices
- Typically presents clients with a “traditional” file system view
  - A single file system namespace that all clients see
  - Client can see side-effects of other clients’ file system activities
  - In many (but not all) ways, an ideal DFS provides clients with the illusion of a shared, local file system
- But …with a distributed implementation
  - Read blocks / files from remote machines across a network, instead of from a local disk

Distributed file systems issues

- What is the basic abstraction
  - A remote file system or a remote disk?
- Naming and transparency
  - How are files name? Location transparent and location independent names
- Caching and replication
  - Where should the cache exist? Client, server, both
  - Replication can exist for performance and/or availability
  - can there be multiple copies of a file in the network?
  - if multiple copies, how are updates handled?
  - what if there's a network partition?

Distributed file systems issues

- Hardware and OS heterogeneity
  - Services interfaces should be defined to enable multiple versions
- Fault tolerance
  - What invocation semantics? At most once, at least once and idempotent operations? Stateless severs?
- Sharing and consistency
  - What are the semantics for sharing and how do we ensure consistency?
- Security, efficiency, …
- Different points in the design space, different DFS

More? Take EECS 340, 345, 443 or 495s

- 340 – Introduction to networking
  - The nitty-gritty details of allowing machines to talk to each other
- 345 – Introductory course on distributed systems
  - Naming and finding things around Earth? Time and the logical order of events? Self destructing data for increased privacy?
- 443 – Advanced OS concepts and current research
  - How to take advantage of multicore? How does Google do file systems? How can we debug millions of lines of code? …
- 495 – Distributed systems in challenging environments
  - The Internet like a P2P system; How does the network look from home and why you care; Seriously large-scale gaming; Environmentally aware distributed systems …