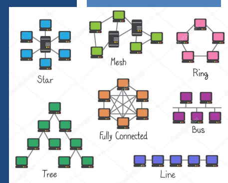
TCSS 558:
APPLIED DISTRIBUTED COMPUTING

System Architectures and Processes



Wes J. Lloyd
School of Engineering
& Technology (SET)
University of Washington - Tacoma

OBJECTIVES – 1/26

- Questions from 1/21
- Assignment 0: Cloud Computing Infrastructure Tutorial
  - New testFibService.sh script
- Chapter 2.3: System Architectures
  - Decentralized peer-to-peer architectures
  - Hybrid architectures
- Chapter 3: Processes
  - Chapter 3.1: Threads
    - Context Switches
    - Threading Models
    - Multithreaded clients/servers
  - Chapter 3.2: Virtualization

ONLINE DAILY FEEDBACK SURVEY

- Daily Feedback Quiz in Canvas – Available After Each Class
- Extra credit available for completing surveys ***ON TIME***
- Tuesday surveys: due by ~ Wed @ 10p
- Thursday surveys: due ~ Mon @ 10p

TCSS 558 A > Assignments

Winter 2021

Search for Assignment

Home
Announcements
Assignments
Zoom
Chat

▾ Upcoming Assignments

TCSS 558 - Online Daily Feedback Survey - 1/5
Not available until Jan 5 at 1:30pm | Due Jan 6 at 10pm | -/1 pts

---

TCSS 558 - Online Daily Feedback Survey - 1/5

Due Jan 6 at 10pm **Points** 1 **Questions** 4
Available Jan 5 at 1:30pm - Jan 6 at 11:59pm 1 day **Time Limit** None

Question 1 0.5 pts

On a scale of 1 to 10, please classify your perspective on material covered in today's class:

1 2 3 4 5 6 7 8 9 10
Mostly Review To Me — Equal New and Review — Mostly New to Me

Question 2 0.5 pts

Please rate the pace of today's class:

1 2 3 4 5 6 7 8 9 10
Slow — Just Right — Fast

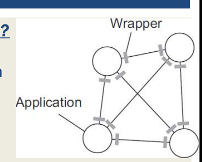
MATERIAL / PACE

- Please classify your perspective on material covered in today's class (22 respondents):
- 1-mostly review, 5-equal new/review, 10-mostly new
- **Average – 6.73** (↑ - *previous 6.65*)

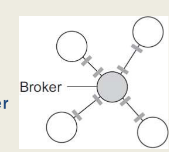
- Please rate the pace of today's class:
- 1-slow, 5-just right, 10-fast
- **Average – 5.50** (↓ - *previous 5.60*)

FEEDBACK FROM 1/21

- ***How is a broker represented in middleware?***
- Each application (circles) must provide unique interface for every other application
- Example: 4 applications → 12 wrappers
- N applications require ~(N^2-N) wrappers
- Every app directly talks to every other app



- **Broker model**
- All interfaces consolidated in the broker
- Broker provides middleware (layer of abstraction) between applications
- Every app only talks to broker
- 4 applications → just 4 wrappers to the broker
- Broker greatly improves maintainability
  - → less custom app-to-app interfaces to maintain



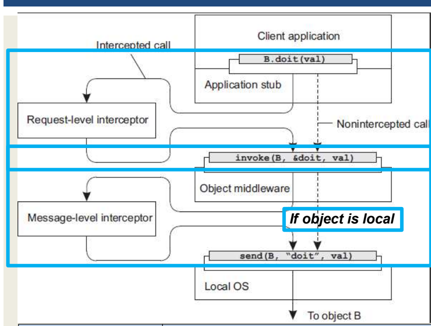
FEEDBACK - 2

- ***Could you explain the graph for interceptors again?***
- ***I am not clear about when to use wrappers and when to use interceptors.***
- Interceptors are used when leveraging RPC or Java RMI approaches
  - Many remote object facilities such as Java RMI and RPC will automate the creation of interceptors
  - Programs may not be required to write or interact with the interceptors directly
- The interceptors graph shows the role interceptors play to translate a remote method invocation.

REMOTE OBJECT INVOCATION BY INTERCEPTION

- Object A can call a method belonging to Object B, while Object B resides on a different computer
- Object A is offered a local interface that is exactly the same as the interface offered by object B
- Object A calls a method in this interface
- Call is transformed into a generic object invocation using the **request-level interceptor** offered by the **middleware** where A resides
  - Abstracts object replication
  - Requests can be sent to each replica
- This generic object invocation is transformed into a message by the **message-level interceptor** that is sent through the TCP-layer offered by object A's **local OS**.
  - Turns method invocation into network call

MIDDLEWARE: INTERCEPTORS - 2



Request-level interceptor transforms:
`B.doit(val)`
into generic call:
`invoke(B,&doit,val)`

Message-level interceptor in middleware sends message through OS (TCP/IP socket) to transfer data:
`send(B,"doit",val)`

Non-intercepted:

OBJECTIVES – 1/26

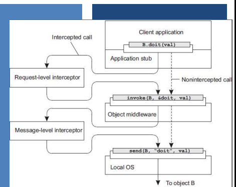
- Questions from 1/21
- **Assignment 0: Cloud Computing Infrastructure Tutorial**
  - **New testFibService.sh script**
- Chapter 2.3: System Architectures
  - Decentralized peer-to-peer architectures
  - Hybrid architectures
- Chapter 3: Processes
  - Chapter 3.1: Threads
    - Context Switches
    - Threading Models
    - Multithreaded clients/servers
  - Chapter 3.2: Virtualization

ASSIGNMENT 0

- ***Preparing for Assignment 0:***
  - Establish AWS Account
    - Standard account – ***** request cloud credits from instructor *****
      - Specify "AWS CREDIT REQUEST" as subject of email
      - Include email address of AWS account
    - **AWS Educate Starter account** – some account limitations
      - **https://awseducate-starter-account-services.s3.amazonaws.com/AWS_Educate_Starter_Account_Services_Supported.pdf**
  - Establish local Linux/Ubuntu environment
- Task 1 – AWS account setup
- Task 2 – Working w/ Docker, creating Dockerfile for Apache Tomcat
- Task 3 – Creating a Dockerfile for haproxy
- Task 4 – Working with Docker-Machine
- Task 5 – For Submission: Testing Alternate Server Configurations

TESTING CONNECTIVITY TO SERVER

- **testFibPar.sh** script is a parallel test script
- Orchestrates multiple threads on client to invoke server multiple times in parallel
- To simplify coordinate of parallel service calls in BASH, **testFibPar.sh** script ignores errors !!!
- To help test client-to-server connectivity, have created a new **testFibService.sh** script
- TEST 1: Network layer
  - Ping (ICMP)
- TEST 2: Transport layer
  - TCP: telnet (TCP Port 8080) – security group (firewall) test
- TEST 3: Application layer
  - HTTP REST – web service test



CH 2.3: SYSTEM ARCHITECTURES

OBJECTIVES – 1/26

- Questions from 1/21
- Assignment 0: Cloud Computing Infrastructure Tutorial
  - New testFibService.sh script
- Chapter 2.3: System Architectures
  - **Decentralized peer-to-peer architectures**
  - Hybrid architectures
- Chapter 3: Processes
  - Chapter 3.1: Threads
    - Context Switches
    - Threading Models
    - Multithreaded clients/servers
  - Chapter 3.2: Virtualization

TYPES OF SYSTEM ARCHITECTURES

- Centralized system architectures
  - Client-server
  - Multitiered
- Decentralized peer-to-peer architectures
  - Structured
  - Unstructured
  - Hierarchically organized
- Hybrid architectures

DECENTRALIZED PEER-TO-PEER ARCHITECTURES

- Client/server:
  - Nodes have specific roles
- Peer-to-peer:
  - Nodes are seen as ***all equal…***
- **How should nodes be organized for communication?**

STRUCTURED PEER-TO-PEER

- Nodes organized using specific ***topology*** (e.g. ring, binary-tree, grid, etc.)
  - Organization assists in data lookups
- Data indexed using "semantic-free" indexing
  - Key / value storage systems
  - Key used to look-up data
- Nodes store data associated with a subset of keys

DISTRIBUTED HASH TABLE (DHT)

- Distributed hash table (DHT) *(ch. 5)*
- Hash function

```
key(data item) = hash(data item's value)
```

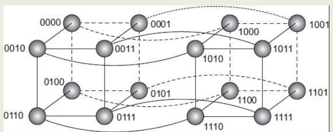
- Hash function "generates" a unique key based on the data
- No two data elements will have the same key (hash)
- System supports data lookup via key
- ***Any*** node can receive and resolve the request
- Lookup function determines which node stores the key

```
existing node = lookup(key)
```

- Node forwards request to node with the data

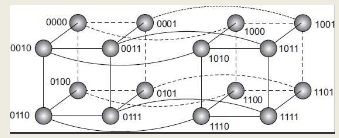
FIXED HYPERCUBE EXAMPLE

- Example where topology helps ***route*** data lookup request
- Statically sized 4-D hypercube, every node has 4 connectors
- 2 x 3-D cubes, 8 vertices, 12 edges
- Node IDs represented as 4-bit code (0000 to 1111)
- Hash data items to 4-bit key (1 of 16 slots)
- Distance (number of hops) determined by identifying number of varying bits between neighboring nodes and destination



FIXED HYPERCUBE EXAMPLE - 2

- **Example:** *fixed hypercube*
  node 0111 (7) retrieves data from node 1110 (14)
- Node 1110 is not a neighbor to 0111
- **Which connector leads to the shortest path?**

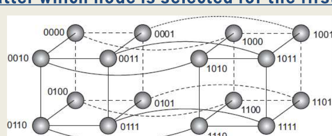


WHICH CONNECTOR LEADS TO THE SHORTEST PATH?

- **Example:** node 0111 (7) retrieves data from node 1110 (14)
- Node 1110 is not a neighbor to 0111

[0111] Neighbors:
1111 (1 bit different than 1110) 0011 (3 bits different– bad path)
0110 (1 bit different than 1110) 0101 (3 bits different– bad path)

- **Does it matter which node is selected for the first hop?**

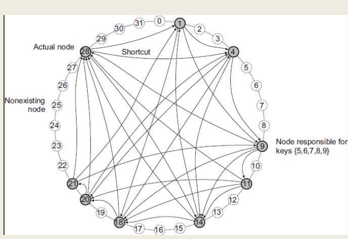


DYNAMIC TOPOLOGY

- Fixed hypercube requires static topology
  - Nodes cannot join or leave
- Relies on symmetry of number of nodes
- Can force the DHT to a certain size
- Chord system – DHT (again in ch.5)
  - Dynamic topology
  - Nodes organized in ring
  - Every node has unique ID
  - Each node connected with other nodes (shortcuts)
  - Shortest path between any pair of nodes is ~ order O(log N)
  - N is the total number of nodes

CHORD SYSTEM

- Data items have m-bit key
- Data item is stored at closest "successor" node with ID ≥ key k
- Each node maintains finger table of successor nodes
- Client sends key/value lookup to ***any*** node
- Node forwards client request to node with m-bit ID closest to, but not greater than key k
- Nodes must continually refresh finger tables by communicating with adjacent nodes to incorporate node joins/departures



UNSTRUCTURED PEER-TO-PEER

- **No topology:** ***How do nodes find out about each other?***
- Each node maintains adhoc list of neighbors
- Facilitates nodes frequently joining, leaving, adhoc systems
- **Neighbor:** node reachable from another via a network path
- Neighbor lists constantly refreshed
  - Nodes query each other, remove unresponsive neighbors
- Forms a "random graph"
- Predetermining network routes not possible
  - How would you calculate the route algorithmically?
- Routes must be discovered

SEARCHING FOR DATA: UNSTRUCTURED PEER-TO-PEER SYSTEMS

- **Flooding**
- [Node u] sends request for data item to all neighbors
- [Node v]
  - Searches locally, responds to u (or forwarder) if having data
  - Forwards request to **ALL** neighbors
  - Ignores repeated requests
- Features
  - High network traffic
  - Fast search results by saturating the network with requests
  - Variable # of hops
  - Max number of hops or time-to-live (TTL) often specified
  - Requests can "retry" by gradually increasing TTL/max hops until data is found

SEARCHING FOR DATA - 2

- **Random walks**
- [Node u] asks a randomly chosen neighbor [node v]
- If [node v] does not have data, forwards request to a random neighbor
- Features
  - Low network traffic
  - Akin to sequential search
  - Longer search time
  - [node u] can start "n" random walks simultaneously to reduce search time
  - As few as n=16..64 random walks sufficient to reduce search time (LV et al. 2002)
  - Timeout required - need to coordinate stopping network-wide walk when data is found...

SEARCHING FOR DATA - 3

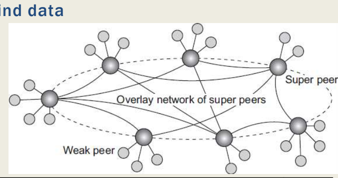
- **Policy-based search methods**
- Incorporate history and knowledge about the adhoc network ***at the node-level*** to enhance effectiveness of queries
- Nodes maintain lists of preferred neighbors which often succeed at resolving queries
- Favor neighbors having highest number of neighbors
  - Can help minimize hops

HIERARCHICAL PEER-TO-PEER NETWORKS

- **Problem:**
  Adhoc system search performance does not scale well as system grows
- Allow nodes to assume **ROLES** to improve search
- Content delivery networks (CDNs) *(video streaming)*
  - Store (cache) data at nodes local to the requester (client)
  - Broker node – tracks resource usage and node availability
    - Track where data is needed
    - Track which nodes have capacity (disk/CPU resources) to host data
- Node roles
  - Super peer –Broker node, routes client requests to storage nodes
  - Weak peer – Store data

HIERARCHICAL PEER-TO-PEER NETWORKS - 2

- Super peers
  - Head node of local centralized network
  - Interconnected via overlay network with other super peers
  - May have replicas for fault tolerance
- Weak peers
  - Rely on super peers to find data
- Leader-election problem:
  - Who can become a super peer?
  - What requirements must be met to become a super peer?



OBJECTIVES – 1/26

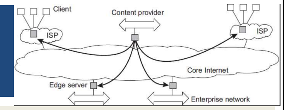
- Questions from 1/21
- Assignment 0: Cloud Computing Infrastructure Tutorial
  - New testFibService.sh script
- Chapter 2.3: System Architectures
  - Decentralized peer-to-peer architectures
  - **Hybrid architectures**
- Chapter 3: Processes
  - Chapter 3.1: Threads
    - Context Switches
    - Threading Models
    - Multithreaded clients/servers
  - Chapter 3.2: Virtualization

TYPES OF SYSTEM ARCHITECTURES

- Centralized system architectures
  - Client-server
  - Multitiered
- Decentralized peer-to-peer architectures
  - Structured
  - Unstructured
  - Hierarchically organized
- Hybrid architectures

January 26, 2021 | TCSS558: Applied Distributed Computing [Winter 2021] School of Engineering and Technology, University of Washington - Tacoma | L7.31

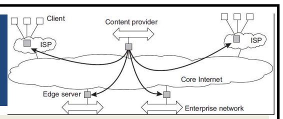
HYBRID ARCHITECTURES



- Combine centralized server concepts with decentralized peer-to-peer models
- **Edge-server systems:**
- Adhoc peer-to-peer devices connect to the internet through an edge server (origin server)
- Edge servers (provided by an ISP) can optimize content and application distribution by storing assets near the edge
- **Example:**
- AWS Lambda@Edge: Enables Node.js Lambda Functions to execute "at the edge" harnessing existing CloudFront Content Delivery Network (CDN) servers
- **https://www.infoq.com/news/2017/07/aws-lambda-at-edge**

January 26, 2021 | TCSS558: Applied Distributed Computing [Winter 2021] School of Engineering and Technology, University of Washington - Tacoma | L7.32

HYBRID ARCHITECTURES - 2



- **Fog computing:**
- Extend the scope of managed resources beyond the cloud to leverage compute and storage capacity of end-user devices
- End-user devices become part of the overall system
- Middleware extended to incorporate managing edge devices as participants in the distributed system
- Cloud → in the sky
  - *compute/resource capacity is huge, but far away…*
- Fog → (devices) on the ground
  - *compute/resource capacity is constrained and local…*

January 26, 2021 | TCSS558: Applied Distributed Computing [Winter 2021] School of Engineering and Technology, University of Washington - Tacoma | L7.33

COLLABORATIVE DISTRIBUTED SYSTEM EXAMPLE

- **BitTorrent Example:** File sharing system – users must contribute as a file host to be eligible to download file resources
- Original implementation features hybrid architecture
- Leverages idle client network capacity in the background
- User joins the system by interacting with a central server
- Client accesses global directory from a ***tracker*** server at well known address to access torrent file
- Torrent file tracks nodes having chunks of requested file
- Client begins downloading file chunks and immediately then participates to reserve downloaded content **or network bandwidth is reduced!!**
- Chunks can be downloaded in parallel from distributed nodes

January 26, 2021 | TCSS558: Applied Distributed Computing [Winter 2021] School of Engineering and Technology, University of Washington - Tacoma | L7.34

WE WILL RETURN AT 2:40PM

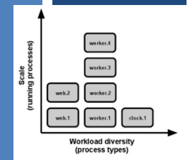
REVIEW QUESTIONS

- What is difference in finding/disseminating data in unstructured vs. structured peer-to-peer networks?
  - Spreading/finding data
  - Flooding, Random walk
- What are some advantages of a decentralized structured peer-to-peer architecture?
- What are some disadvantages?
- What are some advantages of a decentralized unstructured peer-to-peer architecture?
- What are some disadvantages?

January 26, 2021 | TCSS558: Applied Distributed Computing [Winter 2021] School of Engineering and Technology, University of Washington - Tacoma | L7.36

OBJECTIVES – 1/26

- Questions from 1/21
- Assignment 0: Cloud Computing Infrastructure Tutorial
  - New testFibService.sh script
- Chapter 2.3: System Architectures
  - Decentralized peer-to-peer architectures
  - Hybrid architectures
- **Chapter 3: Processes**
  - Chapter 3.1: Threads
    - Context Switches
    - Threading Models
    - Multithreaded clients/servers
  - Chapter 3.2: Virtualization



CH. 3: PROCESSES
CH. 3.1: THREADS

CHAPTER 3

- Chapter 3 titled "processes"
- Covers variety of distributed system implementation details
- "Grab bag" of topics

- Processes/threads
- Virtualization
- Clients
- Servers
- Code migration

OBJECTIVES – 1/26

- Questions from 1/21
- Assignment 0: Cloud Computing Infrastructure Tutorial
  - New testFibService.sh script
- Chapter 2.3: System Architectures
  - Decentralized peer-to-peer architectures
  - Hybrid architectures
- Chapter 3: Processes
  - **Chapter 3.1: Threads**
    - Context Switches
    - Threading Models
    - Multithreaded clients/servers
  - Chapter 3.2: Virtualization

CH. 3.1 - THREADS

- For implementing a server (or client) threads offer many advantages vs. heavy weight processes
- What is the difference between a process and a thread?
  - (*review?*) from Operating Systems
- ***Key difference***: what do threads share amongst each other that processes do not…. ?
- What are the segments of a program stored in memory?
  - Heap segment (dynamic shared memory)
  - Code segment
  - Stack segment
  - Data segment (global variables)

THREADS - 2

- **Do several processes on an operating system share…**
  - **Heap segment?**
  - **Stack segment?**
  - **Code segment?**
- **Can we run multiple copies of the same code?**
- These may be managed as shared pages (across processes) in memory

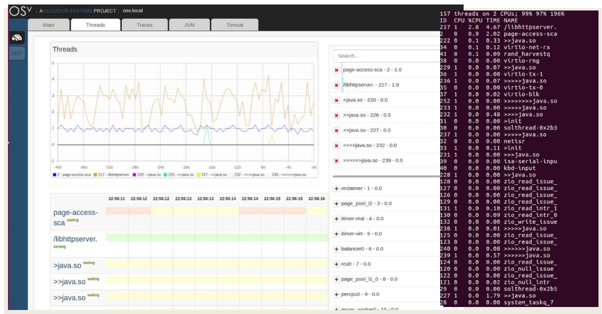
- Processes are isolated from each other by the OS
  - Each has a separate heap, stack, code segment

THREADS - 3

- Threads avoid the overhead of process creation
- No new heap or code segments required
- **What is a context switch?**
- Context switching among threads is considered to be more efficient than context switching processes
- Less elements to swap-in and swap-out
- Unikernel: specialized single process OS for the cloud
- Example: Osv, Clive, MirageOS (see: http://unikernel.org/projects/)
- Single process operating system with many threads
- Developed for the cloud to run only one application at a time

January 26, 2021 | TCSS558: Applied Distributed Computing [Winter 2021] School of Engineering and Technology, University of Washington - Tacoma | L7.43

OSV: ONE PROCESS, MANY THREADS



January 26, 2021 | TCSS558: Applied Distributed Computing [Winter 2021] School of Engineering and Technology, University of Washington - Tacoma | L7.44

THREADS - 4

- Important implications with threads:
- (1) multi-threading should lead to performance gains
- (2) thread programming requires additional effort when threads share memory
  - Known as thread **synchronization**, or enabling **concurrency**
- Access to **critical sections** of code which modify shared variables must be **mutually exclusive**
  - No more than one thread can execute at any given time
  - Critical sections must run **atomically** on the CPU

January 26, 2021 | TCSS558: Applied Distributed Computing [Winter 2021] School of Engineering and Technology, University of Washington - Tacoma | L7.45

BLOCKING THREADS

- Example: spreadsheet with formula to compute sum of column
- User modifies values in column
- Multiple threads:

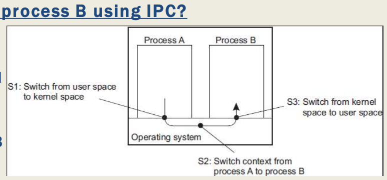
1. Supports interaction (UI) activity with user
2. Updates spreadsheet calculations in parallel
3. Continually backs up spreadsheet changes to disk

- Single core CPU
  - Tasks appear as if they are performed simultaneously
- Multi core CPU
  - Tasks ***execute*** simultaneously

January 26, 2021 | TCSS558: Applied Distributed Computing [Winter 2021] School of Engineering and Technology, University of Washington - Tacoma | L7.46

INTERPROCESS COMMUNICATION

- IPC – mechanism using pipes, message queues, and shared memory segments
- IPC mechanisms incur context switching
  - Process I/O must execute in kernel mode
- **How many context switches are required for process A to send a message to process B using IPC?**
- **#1 C/S:** Proc A→kernel thread
- **#2 C/S:** Kernel thread→Proc B



January 26, 2021 | TCSS558: Applied Distributed Computing [Winter 2021] School of Engineering and Technology, University of Washington - Tacoma | L7.47

OBJECTIVES – 1/26

- Questions from 1/21
- Assignment 0: Cloud Computing Infrastructure Tutorial
  - New testFibService.sh script
- Chapter 2.3: System Architectures
  - Decentralized peer-to-peer architectures
  - Hybrid architectures
- Chapter 3: Processes
  - Chapter 3.1: Threads
    - **Context Switches**
    - Threading Models
    - Multithreaded clients/servers
  - Chapter 3.2: Virtualization

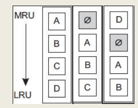
January 26, 2021 | TCSS558: Applied Distributed Computing [Winter 2021] School of Engineering and Technology, University of Washington - Tacoma | L7.48

CONTEXT SWITCHING

- **Direct overhead**
  - Time spent not executing program code (user or kernel)
  - Time spent executing interrupt routines to swap memory segments of different processes (or threads) in the CPU
  - Stack, code, heap, registers, code pointers, stack pointers
  - Memory page cache invalidation
- **Indirect overhead**
  - Overhead not directly attributed to the physical actions of the context switch
  - Captures performance degradation related to the side effects of context switching (e.g. rewriting of memory caches, etc.)
  - ***Primarily cache perturbation***

CONTEXT SWITCH – CACHE PERTURBATION

- Refers to cache reorganization that occurs as a result of a context switch
- Cache is not clear, but elements from cache are removed as a result of another program running in the CPU
- 80% performance overhead from context switching results from this **"cache perturbation"**



OBJECTIVES – 1/26

- Questions from 1/21
- Assignment 0: Cloud Computing Infrastructure Tutorial
  - New testFibService.sh script
- Chapter 2.3: System Architectures
  - Decentralized peer-to-peer architectures
  - Hybrid architectures
- Chapter 3: Processes
  - Chapter 3.1: Threads
    - Context Switches
    - **Threading Models**
    - Multithreaded clients/servers
  - Chapter 3.2: Virtualization

THREADING MODELS

- **Many-to-one threading:** multiple user-level threads per process
- Thread operations (create, delete, locks) run in user mode
- Multithreaded process mapped to single schedulable entity
- Only run thread per process runs at any given time
- Key take-away: thread management handled by user processes
- **What are some advantages of many-to-one threading?**
- **What are some disadvantages?**

THREADING MODELS - 2

- **One-to-one threading**: use of separate kernel threads for each user process - also called ***kernel-level threads***
- The kernel API calls (e.g. I/O, locking) are farmed out to an existing kernel level thread
- Thread operations (create, delete, locks) run in kernel mode
- Threads scheduled individually by the OS
- System calls required, context switches as expensive as process context switching
- Idea is to have preinitialized kernel threads for user processes
- Linux uses this model…
- **What are some advantages of one-to-one threading?**
- **What are some disadvantages?**

APPLICATION EXAMPLES

- Google chrome: processes
- Apache tomcat webserver: threads
- Multiprocess programming avoids synchronization of concurrent access to shared data, by providing coordination and data sharing via interprocess communication (IPC)
- Each process maintains its own private memory
- **While this approach avoids synchronizing concurrent access to shared memory, what is the tradeoff(s) ??**
  - Replication instead of synchronization – must synchronize multiple copies of the data
- **Do distributed objects share memory?**

OBJECTIVES – 1/26

- Questions from 1/21
- Assignment 0: Cloud Computing Infrastructure Tutorial
  - New testFibService.sh script
- Chapter 2.3: System Architectures
  - Decentralized peer-to-peer architectures
  - Hybrid architectures
- Chapter 3: Processes
  - Chapter 3.1: Threads
    - Context Switches
    - Threading Models
    - **Multithreaded clients/servers**
  - Chapter 3.2: Virtualization

MULTITHREADED CLIENTS

- **Web browser**
- Uses threads to load and render portions of a web page to the user in parallel
- A client could have dozens of concurrent connections all loading in parallel

- **testFibPar.sh**
- Assignment 0 client script (GNU parallel)

- **Important benefits:**
- Several connections can be opened simultaneously
- Client: dozens of concurrent connections to the webserver all loading data in parallel

MULTIPLE THREADS

- In Linux, threads also receive a process ID (PID)
- To display threads of a process in Linux:

- Identify parent process explicitly:

- top –H –p <pid>
- htop –p <pid>
- ps –iT <pid>

- Virtualbox process ~ 44 threads
- No mapping to guest # of processes/threads

PROCESS METRICS

CPU
- cpuUsr: CPU time in user mode
- cpuKrn: CPU time in kernel mode
- cpuIdle: CPU idle time
- cpuIoWait: CPU time waiting for I/O
- cpuIntSrvc: CPU time serving interrupts
- cpuSftIntSrvc: CPU time serving soft interrupts
- cpuNice: CPU time executing prioritized processes
- cpuSteal: CPU ticks lost to virtualized guests
- contextsw: # of context switches
- loadavg: (avg # proc / 60 secs)

Disk
- dsr: disk sector reads
- dsreads: disk sector reads completed
- drm: merged adjacent disk reads
- readtime: time spent reading from disk
- dsw: disk sector writes
- dswrites: disk sector writes completed
- dwm: merged adjacent disk writes
- writetime: time spent writing to disk

Network
- nbs: network bytes sent
- nbr: network bytes received

LOAD AVERAGE

- Reported by: `top`, `htop`, `w`, `uptime`, and `/proc/loadavg`
- Updated every 5 seconds
- Average number of processes using or waiting for the CPU
- Three numbers show exponentially decaying usage for 1 minute, 5 minutes, and 15 minutes
- One minute average: exponentially decaying average
- Load average = 1 • (avg last minute load) – 1/e • (avg load since boot)

- 1.0 = 1-CPU core fully loaded
- 2.0 = 2-CPU cores
- 3.0 = 3-CPU cores . . .

THREAD-LEVEL PARALLELISM

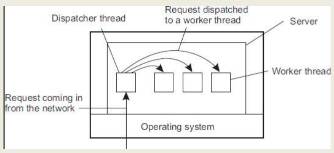
- Metric – measures degree of parallelism realized by running system, by calculating average utilization:

$$TLP = \frac{\sum_{i=1}^{N} i \cdot c_i}{1 - c_0}$$

- Ci – fraction of time that exactly I threads are executed
- N – maximum threads that can execute at any one time
- Web browsers found to have TLP from 1.5 to 2.5
- Clients for web browsing can utilize from 2 to 3 CPU cores
- Any more cores are redundant, and potentially wasteful
- **Measure TLP to understand how many CPUs to provision**

MULTITHREADED SERVERS

- Multiple threads essential for servers in distributed systems
- Even on single-core machines greatly improves performance
- Take advantage of idle/blocking time
- Two designs:
  - Generate new thread for every request
  - Thread pool – pre-initialize set of threads to service requests



SINGLE THREAD & FSM SERVERS

- Single thread server
  - A single thread handles all client requests
  - BLOCKS for I/O
  - All waiting requests are queued until thread is available
- Finite state machine
  - Server has a single thread of execution
  - I/O performing asynchronously (non-BLOCKing)
  - Server handles other requests while waiting for I/O
  - Interrupt fired with I/O completes
  - Single thread "jumps" back into context to finish request

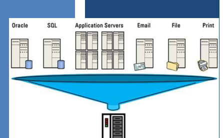
SERVER DESIGN ALTERNATIVES

- A blocking system call implies that a thread servicing a request synchronously performs I/O
- The thread BLOCKS to wait on disk/network I/O before proceeding with request processing
- Consider the implications of these designs for responsiveness, availability, scalability. . .

Model	Characteristics
Multithreading	Parallelism, blocking I/O
Single-thread	No parallelism, blocking I/O
Finite-state machine	Parallelism, non-blocking I/O

OBJECTIVES – 1/26

- Questions from 1/21
- Assignment 0: Cloud Computing Infrastructure Tutorial
  - New testFibService.sh script
- Chapter 2.3: System Architectures
  - Decentralized peer-to-peer architectures
  - Hybrid architectures
- Chapter 3: Processes
  - Chapter 3.1: Threads
    - Context Switches
    - Threading Models
    - Multithreaded clients/servers
  - **Chapter 3.2: Virtualization**



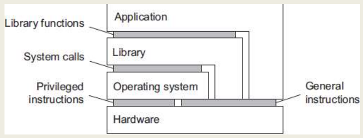
CH. 3.2: VIRTUALIZATION

VIRTUALIZATION

- Initially introduced in the 1970s on IBM mainframe computers
- Legacy operating systems run in mainframe-based VMs
- Legacy software could be sustained by virtualizing legacy OSes
- 1970s virtualization went away as desktop/rack-based hardware became inexpensive
- Virtualization reappears in 2000s to leverage multi-core, multi-CPU processor systems
- VM-Ware virtual machines enable companies to host many virtual servers with mixed OSes on private clusters
- Cloud computing: Amazon offers VMs as-a-service (IaaS)

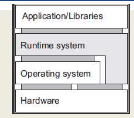
TYPES OF VIRTUALIZATION

- **Levels of instructions:**
- **Hardware**: CPU
  - Privileged instructions KERNEL MODE
  - General instructions USER MODE
- **Operating system**: system calls
- **Library**: programming APIs: e.g. C/C++,C#, Java libraries
- **Application**:
- **Goal of virtualization:** mimic these interface to provide a virtual computer

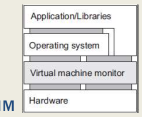


TYPES OF VIRTUALIZATION - 2

- **Process virtual machine**
  - Interpret instructions: (interpreters) (JavaVM) byte code → HW instructions
  - Emulate instructions: (emulators) (Wine) windows code → Linux code

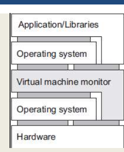


- **Native virtual machine monitor (VMM)**
  - Hypervisor (XEN): small OS with its own kernel
  - Provides an interface for multiple guest OSes
  - Facilitates sharing/scheduling of CPU, device I/O among many guests
  - Guest OSes require special kernel to interface w/ VMM
  - Supports **Paravirtualization** for performance boost to run code directly on the CPU
  - Type 1 hypervisor



TYPES OF VIRTUALIZATION - 3

- **Hosted virtual machine monitor (VMM)**
  - Runs atop of hosted operating system
  - Uses host OS facilities for CPU scheduling, I/O
  - Full virtualization
  - Type 2 hypervisor
  - **Virtualbox**



- *Textbook: note 3.5–good explanation of full vs. paravirtualization*
- **GOAL**: run all user mode instructions directly on the CPU
- x86 instruction set has ~17 privileged user mode instructions
- **Full virtualization**: scan the EXE, insert code around privileged instructions to divert control to the VMM
- **Paravirtualization**: special OS kernel eliminates side effects of privileged instructions

EVOLUTION OF AWS VIRTUALIZATION

From http://www.brendangregg.com/blog/2017-11-29/aws-ec2-virtualization-2017.html

VS: Virtualization in software

P: Paravirtual

VH: Virtualization in Hardware

H: Hardware

AWS EC2 Virtualization Types

Bare-metal performance (grey) / Near-metal performance (green) / Optimized performance (blue) / Poor performance (red)

Importance: Most → Least

	#	Tech	Type	With	CPU, Memory	Network I/O	Local Storage I/O	Remote Storage I/O	Interrupts, Timers	Motherboard, Boot
	1	VM	Fully Emulated		VS	VS	VS	VS	VS	VS
Old	2	VM	Xen PV 3.0	PV drivers	P	P	P	P	VS	VS
	3	VM	Xen HVM 3.0	PV drivers	VH	P	P	P	VS	VS
	4	VM	Xen HVM 4.0.1	PVHVM drivers	VH	P	P	P	P	VS
	5	VM	Xen AWS 2013	PVHVM + SR-IOV(net)	VH	VH	P	P	P	VS
	6	VM	Xen AWS 2017	PVHVM + SR-IOV(net, stor.)	VH	VH	VH	P	P	VS
	7	VM	AWS Nitro 2017		VH	VH	VH	VH	VH	VS
New	8	HW	AWS Bare Metal 2017		H	H	H	H	H	H
			Bare Metal		H	H	H	H	H	H

VM: Virtual Machine. HW: Hardware.
VS: Virt. in software. VH: Virt. in hardware. P: Paravirt. Not all combinations shown.
SR-IOV(net): ixgbe/ena driver. SR-IOV(storage): nvme driver.

AWS VIRTUALIZATION - 2

- **Full Virtualization - Fully Emulated**
  - Never used on EC2, before CPU extensions for virtualization
  - Can boot any unmodified OS
  - Support via slow emulation, performance 2x-10x slower
- **Paravirtualization: Xen PV 3.0**
  - Software: Interrupts, timers
  - Paravirtual: CPU, Network I/O, Local+Network Storage
  - Requires special OS kernels, interfaces with hypervisor for I/O
  - Performance 1.1x – 1.5x slower than "bare metal"
  - Instance store instances: 1[ST] & 2[nd] generation- m1.large, m2.xlarge
- **Xen HVM 3.0**
  - Hardware virtualization: **CPU**, **memory** **(CPU VT-x required)**
  - Paravirtual: network, storage
  - Software: interrupts, timers
  - EBS backed instances
  - m1, c1 instances

AWS VIRTUALIZATION - 3

- **XEN HVM 4.0.1**
  - Hardware virtualization: CPU, memory **(CPU VT-x required)**
  - Paravirtual: network, storage, **interrupts**, **timers**
- **XEN AWS 2013** *(diverges from opensource XEN)*
  - Provides hardware virtualization for CPU, memory, **network**
  - Paravirtual: storage, **interrupts**, **timers**
  - Called Single root I/O Virtualization (SR-IOV)
  - Allows sharing single physical PCI Express device (i.e. network adapter) with multiple VMs
  - Improves VM network performance
  - 3[rd] & 4[th] generation instances (c3 family)
  - Network speeds up to 10 Gbps and 25 Gbps
- **XEN AWS 2017**
  - Provides hardware virtualization for CPU, memory, network, **local disk**
  - Paravirtual: remote storage, **interrupts**, **timers**
  - Introduces hardware virtualization for EBS volumes (c4 instances)
  - Instance storage hardware virtualization (x1.32xlarge, i3 family)

AWS VIRTUALIZATION - 4

- **AWS Nitro 2017**
  - Provides hardware virtualization for CPU, memory, network, **local disk, remote disk, interrupts, timers**
  - All aspects of virtualization enhanced with HW-level support
  - November 2017
  - Goal: provide performance indistinguishable from “bare metal”
  - 5th generation instances – c5 instances (also c5d, c5n)
  - Based on KVM hypervisor
  - Overhead around ~1%

January 26, 2021 | TCSS558: Applied Distributed Computing [Winter 2021] School of Engineering and Technology, University of Washington - Tacoma | L7.73

QUESTIONS

January 26, 2021 | TCSS558: Applied Distributed Computing [Winter 2021] School of Engineering and Technology, University of Washington - Tacoma | L7.120