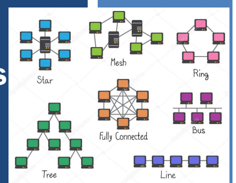
TCSS 558: APPLIED DISTRIBUTED COMPUTING

Chapter 3 - Processes

Wes J. Lloyd
School of Engineering and Technology
University of Washington - Tacoma



1

OBJECTIVES

- Assignment 0 – questions
- Assignment 1 – questions
- Feedback from 1/30
- Chapter 3.1: Threads – cont'd
- Chapter 3.2: Virtualization
- Chapter 3.3: Clients
- Chapter 3.4: Servers

2

MIDTERM SCHEDULING SURVEY

- **TCSS 558B**
- Tuesday February 11 – 6 respondents (32%)
- Thursday February 13 – 7 respondents (37%) *(Internship fair @UW Seattle)*
- **Tuesday February 18 – 12 respondents (63%)** √
- No Preference – 2 respondents (11%)

- **Midterm Plan:**
- Content coverage - through 1st half of Lecture 11 on Feb 11th
- Practice midterm - 2nd half of Lecture 11 on Feb 11th
- February 13th – Will cover new material not on midterm
- Midterm Exam – Tuesday February 18th
- Exams returned no later than Tuesday February 25th

3

MATERIAL / PACE

- Please classify your perspective on material covered in today's class (9 respondents):
- 1-mostly review, 5-equal new/review, 10-mostly new
- **Average – 6.11 (down from 7.09)**

- Please rate the pace of today's class:
- 1-slow, 5-just right, 10-fast
- **Average – 5.22 (up from 4.91)**

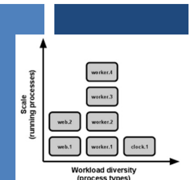
4

FEEDBACK FROM 1/30

- What is the purpose of the many-to-one and one-to-one threading models? *(question edited to correct terms)*

- Thank you for question
- Have revised slides L8 for clarity

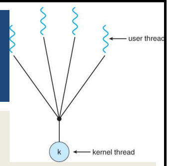
5



CH. 3: PROCESSES
CH. 3.1: THREADS

6

MANY-TO-ONE THREADING MODEL

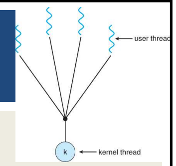


- **Many-to-one threading model:**
- Multiple user-level threads per process
- All threads mapped to single schedulable process in the OS
- Program appears as single process to the system
- Thread operations (create, delete, locks) run in user mode
- Any blocking system call by one thread blocks entire process
- User manages scheduling of threads, not OS
- One kernel thread per process: process restricted to 1 CPU
- Key take-away: thread management handled by user processes
- **What are some advantages of many-to-one threading?**
- **What are some disadvantages?**

February 4, 2020 | TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma | L9.7

7

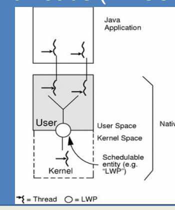
MANY-TO-ONE THREADING MODEL



- **Many-to-one threading model:**
- Multiple user-level threads per process

Initial implementation of Java threads (~ 1995?) used many-to-one threading model

This threading model is now seldomly used

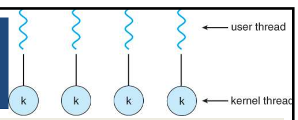


- **What are some disadvantages?**

February 4, 2020 | TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma | L9.8

8

ONE-TO-ONE THREADING MODEL



- Threads operations managed by the OS (create, delete, lock)
- Thread ops run in kernel mode using separate kernel threads
  - Kernel API calls farmed out to preinitialized kernel level theads
  - Requires system calls and context switch from user to kernel thread
- One user thread to one kernel thread
- User process can use many kernel threads
- Also called ***kernel-level threads***
- All threads scheduled individually by the OS
- Enables running single process across multiple CPUs
- Now commonly used… (used in Linux)
- **What are some advantages of one-to-one threading?**
- **What are some disadvantages?**

February 4, 2020 | TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma | L9.9

9

APPLICATION EXAMPLES

- Alternative: **Collection of concurrent processes**
- Google chrome: tabs backed by processes
- Apache http server: Apache Multi-Processing-Module (MPM prefork)
- 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?**

February 4, 2020 | TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma | L9.10

10

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

February 4, 2020 | TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma | L9.11

11

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

February 4, 2020 | TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma | L9.12

12

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

13

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 . . .

February 4, 2020 | TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma | L9.14

14

THREAD-LEVEL PARALLELISM

- TLP Metric: ***Is multithreading effectively exploited?***
- ***Is the bottleneck the CODE or the system?***
- TLP measures degree of parallelism realized by running system, by calculating average utilization:

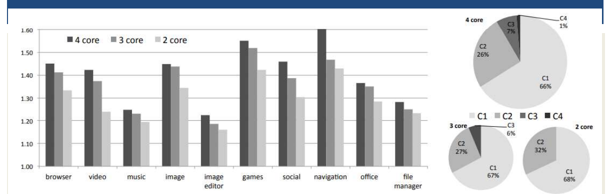
$$TLP = \frac{\sum_{i=1}^{N} i \cdot c_i}{1 - c_0}$$

- Ci – fraction of time where exactly i threads are executed
- C_0 – idle time of the system
- 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**

February 4, 2020 | TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma | L9.15

15

HOW PARALLEL IS YOUR PROGRAM?



(a) TLP for different application categories
(b) Average CPU time breakdown

From: Gao et al., *A Study of Thread Level Parallelism on Mobile Devices*, 2014 IEEE International Symposium on Performance Analysis of Systems and Software

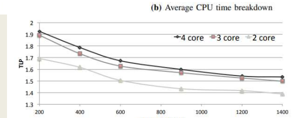


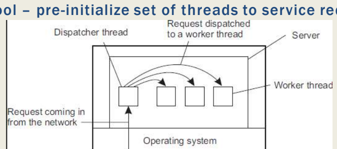
Fig. 2: TLP under different frequencies (BBench)

February 4, 2020 | TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma | L9.16

16

MULTITHREADED SERVERS

- Common & essential for TCP/IP servers and distributed systems
- **Example**: Apache tomcat webserver: threads
- Even on single-core machines greatly improves performance
- Take advantage of idle/blocking time
- Thread management approaches:
  - Generate new thread for every request
  - Thread pool – pre-initialize set of threads to service requests



February 4, 2020 | TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma | L9.17

17

SINGLE THREAD SERVERS

- Single thread server
  - A single thread handles all client requests
  - BLOCKS for I/O
  - All waiting requests are queued until thread is available
  - Concurrent users all share a single thread
  - Must wait until it is available
  - No data corruption or synchronization challenges
  - No debugging of race conditions, deadlocks
  - Slow, not scalable

February 4, 2020 | TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma | L9.18

18

FINITE STATE MACHINE SERVERS

- Finite state machine
  - Server has just single thread for executing client requests
  - DIFFERENCE: I/O performed asynchronously (i.e. non-BLOCKing)
    - Presumably farmed out to kernel threads
  - Server handles other request while waiting for I/O
  - Interrupt fires when I/O completes
  - Single thread "jumps" back into context to finish request
  - Server design avoids synchronization and concurrency issues

19

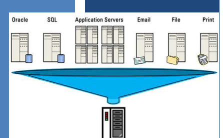
SERVER DESIGN COMPARISON

- 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

20



CH. 3.2: VIRTUALIZATION

21

VIRTUALIZATION

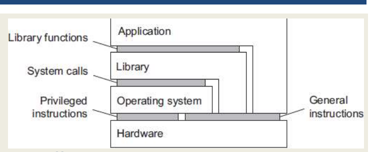
- Initially introduced in the 1970s on IBM mainframe computers
- Legacy operating systems run in mainframe-based VMs
- Legacy software could be sustained by virtualizing legacy OSes
- 1970s virtualization went away as desktop/rack-based hardware became inexpensive

- 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)

22

INSTRUCTION LEVELS REQUIRING VIRTUALIZATION

- **Hardware**: CPU
  - Privileged instructions KERNEL MODE
  - General instructions USER MODE

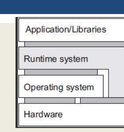


- **Operating system**: system calls
- **Library**: programming APIs: e.g. C/C++,C#, Java libraries
- **Application**: user program code
- **Goal of virtualization:** mimic these interfaces to provide a virtual computer

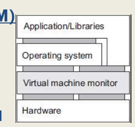
23

TYPES OF VIRTUALIZATION

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



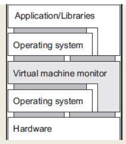
- **NATIVE (bare-metal) virtual machine monitor (VMM)**
  - Hypervisor (XEN): small OS with its own kernel
  - Provides an interface for multiple guest OSes
  - Facilitates sharing/scheduling of CPU, device I/O among many guests
  - Guest OSes require special kernel to interface w/ VMM
  - Supports **Paravirtualization**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



24

TYPES OF VIRTUALIZATION - 2

- **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**

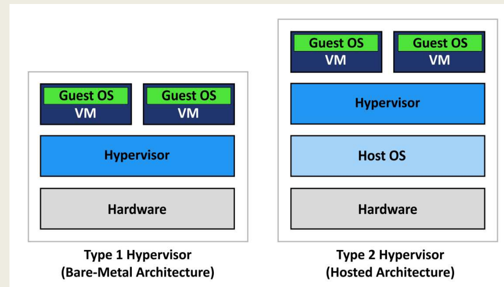


- *Textbook: note 3.5–good explanation of full vs. paravirtualization*
- **GOAL**: run all user mode instructions directly on the CPU
- x86 instruction set has ~17 privileged user mode instructions
- **Full virtualization**: scan the EXE, insert code around privileged instructions to divert control to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

February 4, 2020 | TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma | L9.25

25

BARE METAL VS. HOSTED HYPERVISOR



February 4, 2020 | TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma | L9.26

26

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 / Near-metal performance / Optimized performance / Poor performance

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.

February 4, 2020 | TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma | L9.27

27

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: 1ST & 2nd 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

February 4, 2020 | TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma | L9.28

28

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
  - 3rd & 4th 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)

February 4, 2020 | TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma | L9.29

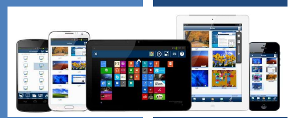
29

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, m5, r5)
  - Based on KVM hypervisor
  - Overhead around ~1%

February 4, 2020 | TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma | L9.30

30



CH. 3.3: CLIENTS

L9.31

31

TYPES OF CLIENTS

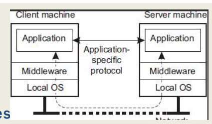
- Thick clients
  - Web browsers
    - Client-side scripting
  - Mobile apps
  - Multi-tier MVC apps
- Thin clients
  - Remote desktops/GUIs (very thin)

February 4, 2020 | TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma | L9.32

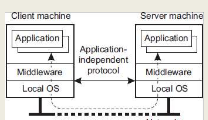
32

CLIENTS

- Application specific protocol
  - Thick clients
  - Clients maintain local data
  - Middleware (APIs)
  - Clients synchronize data with remote nodes
  - Example: shared calendar application



- Application independent
  - Thin clients
  - Client acts as a remote terminal
  - Provides interface to user (GUI / UI)
  - Server houses entire application stack



February 4, 2020 | TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma | L9.33

33

X WINDOWS

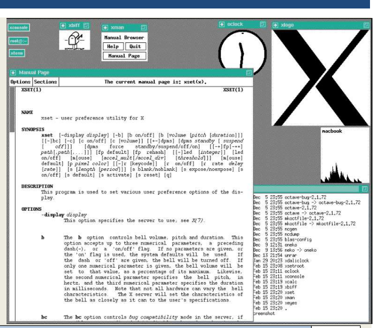
- Layered architecture to transport UI over network
- Remote desktop functionality for Linux/Unix systems
- X kernel acts as a server
  - Provides the **X protocol**: application level protocol
  - Xlib instances (client applications) exchange data and events with X kernels (servers)
  - Clients and servers on single machine → Linux GUI
  - Client and server communication transported over the network → remote Linux GUI

February 4, 2020 | TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma | L9.34

34

X WINDOWS - 2

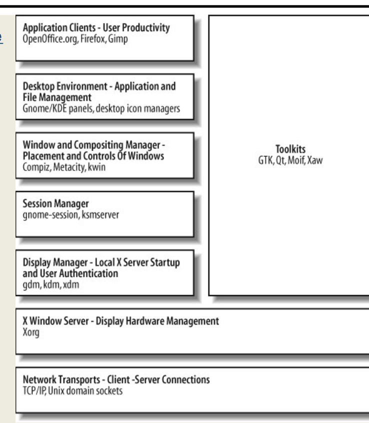
- Window manager:
  - Application running atop of X-windows which provides flair
  - Many variants
  - Without X windows is quite bland



February 4, 2020 | TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma | L9.35

35

- Layered architecture
- X-kernel: low level interface/APIs for controlling screen, capturing keyboard and mouse events (X window Server)
- Provided on Linux as Xlib
- Provides network enabled GUI
- Layering allows for use for custom window managers



February 4, 2020 | TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma | L9.36

36

EXAMPLE: VNC SERVER

- How to Install VNC server on Ubuntu EC2 instance VM:
- `sudo apt-get update`
- `# ubuntu 16.04`
- `sudo apt-get install ubuntu-desktop`
- `sudo apt-get install gnome-panel gnome-settings-daemon metacity nautilus gnome-terminal`
- `# on ubuntu 18.04`
- `sudo apt install xfce4 xfce4-goodies`
- `sudo apt-get install tightvncserver # both`
- Start VNC server to create initial config file
- `vncserver :1`

February 4, 2020 | TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma | L9.37

37

EXAMPLE: VNC SERVER – UBUNTU 16.04

- On the VM: edit config file: `nano ~/.vnc/xstartup`
- Replace contents as below (Ubuntu 16.04):

```
#!/bin/sh

export XKL_XMODMAP_DISABLE=1
unset SESSION_MANAGER
unset DBUS_SESSION_BUS_ADDRESS

[ -x /etc/vnc/xstartup ] && exec /etc/vnc/xstartup
[ -r $HOME/.Xresources ] && xrdb $HOME/.Xresources
xsetroot -solid grey

vncconfig -iconic &
gnome-panel &
gnome-settings-daemon &
metacity &
nautilus &
gnome-terminal &
```

February 4, 2020 | TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma | L9.38

38

EXAMPLE: VNC SERVER – UBUNTU 18.04

- On the VM:
- Edit config file: `nano ~/.vnc/xstartup`
- Replace contents as below (Ubuntu 18.04):

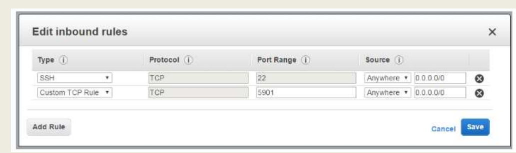
```
#!/bin/bash
xrdb $HOME/.Xresources
startxfce4 &
```

February 4, 2020 | TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma | L9.39

39

EXAMPLE: VNC SERVER - 3

- On the VM: reload config by restarting server
- `vncserver -kill :1`
- `vncserver :1`
- Open port 22 & 5901 in EC2 security group:



February 4, 2020 | TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma | L9.40

40

EXAMPLE: VNC CLIENT

- On the client (e.g. laptop):
- Create SSH connection to securely forward port 5901 on the EC2 instance to your localhost port 5901
- This way your VNC client doesn't need an SSH key

```
ssh -i <ssh-keyfile> -L 5901:127.0.0.1:5901 -N
-f -l <username> <EC2-instance ip_address>
```

- For example:

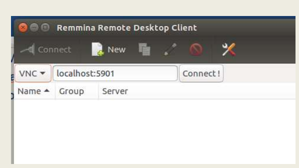
```
ssh -i mykey.pem -L 5901:127.0.0.1:5901 -N -f -
l ubuntu 52.111.202.44
```

February 4, 2020 | TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma | L9.41

41

EXAMPLE: VNC CLIENT - 2

- On the client (e.g. laptop):
- Use a VNC Client to connect
- Remmina is provided by default on Ubuntu 16.04
- Can "google" for many others
- Remmina login:
- Chose "VNC" protocol
- Log into "localhost:5901"

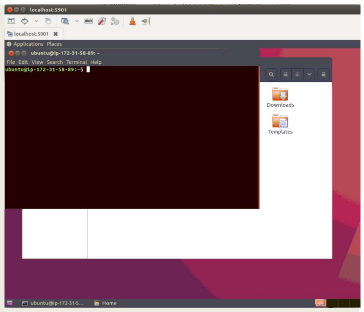


February 4, 2020 | TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma | L9.42

42

REMOTE COMPUTER IN THE CLOUD

- EC2 instance with a GUI. . .!!!



43

ALTERNATIVE: XRDP

- Open source implementation of Microsoft RDP (remote desktop protocol)
- Open source version supports fully functional RDP-compatible remote desktop experience
- Original implementation relied on vncserver, but no longer…

- #Ubuntu installation:
- sudo apt-get install xrdp
- sudo service xrdp start

- Point RDP client to PUBLIC IP: 3389
- Open port 3389 in security group
- Must create user to log in

44

RDP VS. VNC

- VNC sends picture of desktop across network
- Minimal optimizations are employed
  - Send only parts of screen which have changed
  - Limit colors, resolution
- VNC requires more data transfer
- RDP sends instructions on how to draw screen to client
- Client renders image based on instructions and displays it
- Transferring instructions requires much less network bandwidth
- Client computer "understands" image it has created
- Client performs simple operations locally
  - Move windows without sending mouse input to host computer
  - No need to wait for host computer to render moved window
  - No need to wait for response from server
  - Client just calculates and draws results locally

45

THIN CLIENTS

- Thin clients
  - X windows protocol
  - A variety of other remote desktop protocols exist:

Remote desktop protocols include the following:

- Apple Remote Desktop Protocol (ARD) – Original protocol for Apple Remote Desktop on macOS machines.
- Appliance Link Protocol (ALP) – a Sun Microsystems-specific protocol featuring audio (play and record), remote printing, remote USB, accelerated video
- HP Remote Graphics Software (RGS) – a proprietary protocol designed by Hewlett-Packard specifically for high end workstation remoting and collaboration.
- Independent Computing Architecture (ICA) – a proprietary protocol designed by Citrix Systems
- NX technology (NoMachine NX) – Cross platform protocol featuring audio, video, remote printing, remote USB, H264-enabled.
- PC-over-IP (PCoIP) – a proprietary protocol used by VMware (licensed from Teradici)[2]
- Remote Desktop Protocol (RDP) – a Windows-specific protocol featuring audio and remote printing
- Remote Frame Buffer Protocol (RFB) – A framebuffer level cross-platform protocol that VNC is based on.
- SPICE (Simple Protocol for Independent Computing Environments) – remote-display system built for virtual environments by Qumranet, now Red Hat
- Splashtop – a high performance remote desktop protocol developed by Splashtop, fully optimized for hardware (H.264) including Intel / AMD chipsets, NVIDIA of media codecs, Splashtop can deliver high frame rates with low latency, and also low power consumption.
- X Window System (X11) – a well-established cross-platform protocol mainly used for displaying local applications; X11 is network transparent

46

THIN CLIENTS - 2

- Applications should separate application logic from UI
- When application logic and UI interaction are tightly coupled many requests get sent to X kernel
- Client must wait for response
- Synchronous behavior and app-to-UI coupling adverselt affects performance of WAN / Internet

- Protocol optimizations: reduce bandwidth by shrinking size of X protocol messages
- Send only differences between messages with same identifier
- Optimizations enable connections with 9600 kbps

47

THIN CLIENTS - 3

- Virtual network computing (VNC)
- Send display over the network at the pixel level (instead of X lib events)
- Reduce pixel encodings to save bandwidth – fewer colors
- Pixel-based approaches loose application semantics
- Can transport any GUI this way

- THINC- hybrid approach
- Send video device driver commands over network
- More powerful than pixel based operations
- Less powerful compared to protocols such as X

48

TRADEOFFS: ABSTRACTION OF REMOTE DISPLAY PROTOCOLS

- Tradeoff space: abstraction level of remote display protocols

Pixel-level VNC ⟷ **Graphics lib** X11 / RDP

49

TRADEOFFS: ABSTRACTION OF REMOTE DISPLAY PROTOCOLS

- Tradeoff space: abstraction level of remote display protocols

Pixel-level VNC ⟷ **Graphics lib** X11 / RDP

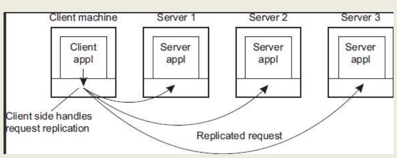
- Generic – no app context
- Graphics data
- Higher network bandwidth
- Fewer colors
- Utilize graphics compression
- More network traffic
- Server more processing

- Application context is available
- UI data/operations
- Lower network bandwidth
- More colors
- Client more processing

50

CLIENT ROLES IN PROVIDING DISTRIBUTION TRANSPARENCY

- Clients help enable distribution transparency of servers
- Replication transparency
  - Client aggregates responses from multiple servers
  - Only the client knows of replicas



51

CLIENT ROLES IN PROVIDING DISTRIBUTION TRANSPARENCY - 2

- Location/relocation/migration transparency
  - Harness convenient naming system to allow client to infer new locations
  - Server inform client of moves / Client reconnects to new endpoint
  - Client hides network address of server, and reconnects as needed
  - May involve temporary loss in performance
- Replication transparency
  - Client aggregates responses from multiple servers
- Failure transparency
  - Client retries, or maps to another server, or uses cached data
- Concurrency transparency
  - Transaction servers abstract coordination of multithreading

52

CH. 3.4: SERVERS

53

SERVERS

- Cloud & Distributed Systems – rely on **Linux**
- http://www.zdnet.com/article/it-runs-on-the-cloud-and-the-cloud-runs-on-linux-any-questions/
- IT is moving to the cloud. And, what powers the cloud?
  - **Linux**
- Uptime Institute survey - 1,000 IT executives (2016)
  - 50% of IT executives – plan to migrate majority of IT workloads to off-premise to cloud or colocation sites
  - 23% expect the shift in 2017, 70% by 2020…
- Docker on Windows / Mac OS X
  - Based on **Linux**
  - Mac: Hyperkit Linux VM
  - Windows: Hyper-V Linux VM

54

SERVERS - 2

- Servers implement a specific service for a collection of clients
- Servers wait for incoming requests, and respond accordingly
- **Server types**
- **Iterative**: immediately handle client requests
- **Concurrent**: Pass client request to separate thread
- Multithreaded servers are concurrent servers
  - E.g. Apache Tomcat
- ***Alternative***: fork a new process for each incoming request
- ***Hybrid***: mix the use of multiple processes with thread pools

55

END POINTS

- Clients connect to servers via: **IP Address** and **Port Number**
- How do ports get assigned?
  - Many protocols support "default" port numbers
  - Client must find IP address(es) of servers
  - A single server often hosts multiple end points (servers/services)
  - When designing new TCP client/servers must be careful not to repurpose ports already commonly used by others

56

COMMON PORTS

packetlife.net

TCP/UDP Port Numbers

7	Echo	554	RTSP	2745	Bagle.H	6891-6901	Windows Live
19	Chargen	546-547	DHCPv6	2967	Symantec AV	6970	Quicktime
20-21	FTP	560	rmonitor	3050	Interbase DB	7212	GhostSurf
22	SSH/SCP	563	NNTP over SSL	3074	XBOX Live	7648-7649	CU-SeeMe
23	Telnet	587	SMTP	3124	HTTP Proxy	8000	Internet Radio
25	SMTP	591	FileMaker	3127	MyDoom	8080	HTTP Proxy
42	WINS Replication	593	Microsoft DCOM	3128	HTTP Proxy	8086-8087	Kaspersky AV
43	WHOIS	631	Internet Printing	3222	GLBP	8118	Privoxy
49	TACACS	636	LDAP over SSL	3260	iSCSI Target	8200	VMware Server
53	DNS	639	MSDP (PIM)	3306	MySQL	8500	Adobe ColdFusion
67-68	DHCP/BOOTP	646	LDP (MPLS)	3389	Terminal Server	8767	TeamSpeak
69	TFTP	691	MS Exchange	3689	iTunes	8866	Bagle.B
70	Gopher	860	iSCSI	3690	Subversion	9100	HP JetDirect
79	Finger	873	rsync	3724	World of Warcraft	9101-9103	Bacula
80	HTTP	902	VMware Server	3784-3785	Ventrilo	9119	MXit
88	Kerberos	989-990	FTP over SSL	4333	mSQL	9800	WebDAV
102	MS Exchange	993	IMAP4 over SSL	4444	Blaster	9898	Dabber
110	POP3	995	POP3 over SSL	4664	Google Desktop	9988	Rbot/Spybot
113	Ident	1025	Microsoft RPC	4672	eMule	9999	Urchin
119	NNTP (Usenet)	1026-1029	Windows Messenger	4899	Radmin	10000	Webmin
123	NTP	1080	SOCKS Proxy	5000	UPnP	10000	BackupExec
135	Microsoft RPC	1080	MyDoom	5001	Slingbox	10113-10116	NetIQ
137-139	NetBIOS	1194	OpenVPN	5001	iperf	11371	OpenPGP
143	IMAP4	1214	Kazaa	5004-5005	RTP	12035-12036	Second Life
161-162	SNMP	1241	Nessus	5050	Yahoo! Messenger	12345	NetBus
177	XDMCP	1311	Dell OpenManage	5060	SIP	13720-13721	NetBackup
179	BGP	1337	WASTE	5190	AIM/ICQ	14567	Battlefield

57

TYPES OF SERVERS

- Daemon server
  - Example: NTP server
- Superserver
- Stateless server
  - Example: Apache server
- Stateful server
- Object servers
- EJB servers

58

NTP EXAMPLE

- Daemon servers
  - Run locally on Linux
  - Track current server end points (outside servers)
  - Example: network time protocol (ntp) daemon
    - Listen locally on specific port (ntp is 123)
    - Daemons routes local client traffic to the configured endpoint servers
    - University of Washington: time.u.washington.edu
    - Example "`ntpq -p`"
      - Queries local ntp daemon, routes traffic to configured server(s)

59

SUPERSERVER

- Linux inetd / xinetd
  - Single superserver
  - Extended internet service daemon
  - Not installed by default on Ubuntu
  - Intended for use on server machines
  - Used to configure box as a server for multiple internet services
    - E.g. ftp, pop, telnet
  - inetd daemon responds to multiple endpoints for multiple services
  - Requests fork a process to run required executable program
- Check what ports you're listening on:
  - `sudo netstat -tap | grep LISTEN`

60

INTERRUPTING A SERVER

- Server design issue:
  - Active client/server communication is taking place over a port
  - How can the server / data transfer protocol support interruption?
- Consider transferring a 1 GB image, how do you pass a unrelated message in this stream?
  1. **Out-of-band** data: special messages sent in-stream to support interrupting the server (*TCP urgent data*)
  2. Use a separate connection (different port) for admin control info
- Example: sftp secure file transfer protocol
  - Once a file transfer is started, can't be stopped easily
  - Must kill the client and/or server

February 4, 2020 | TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma | L9.61

61

STATELESS SERVERS

- Data about state of clients is not stored
- Example: web application servers are typically stateless
  - Also function-as-a-service (FaaS) platforms
- Many servers maintain information on clients (e.g. log files)
- Loss of stateless data doesn't disrupt server availability
  - Loosing log files typically has minimal consequences
- **Soft state**: server maintains state on the client for a limited time (*to support sessions*)
- Soft state information expires and is deleted

February 4, 2020 | TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma | L9.62

62

STATEFUL SERVERS

- Maintain persistent information about clients
- Information must be explicitly deleted by the server
- Example:
  File server - allows clients to keep local file copies for RW
- Server tracks client file permissions and most recent versions
  - Table of (client, file) entries
- If server crashes data must be recovered
- Entire state before a crash must be restored
- Fault tolerance - *Ch. 8*

February 4, 2020 | TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma | L9.63

63

STATEFUL SERVERS - 2

- Session state
  - Tracks series of operations by a single user
  - Maintained temporarily, not indefinitely
  - Often retained for multi-tier client server applications
  - Minimal consequence if session state is lost
  - Clients must start over, reinitialize sessions
- Permanent state
  - Customer information, software keys
- Client-side cookies
  - When servers don't maintain client state, clients can store state locally in "cookies"
  - Cookies are not executable, simply client-side data

February 4, 2020 | TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma | L9.64

64

OBJECT SERVERS

- **OBJECTIVE:** Host objects and enable remote client access
- Do not provide a specific service
  - Do nothing if there are no objects to host
- Support adding/removing hosted objects
- Provide a home where objects live
- Objects, *themselves*, provide "services"
- Object parts
  - State data
  - Code (methods, etc.)
- **Transient object(s)**
  - Objects with limited lifetime (< server)
  - Created at first invocation, destroyed when no longer used (i.e. no clients remain "bound").
  - Disadvantage: initialization may be expensive
  - Alternative: preinitialize and retain objects on server start-up

February 4, 2020 | TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma | L9.65

65

OBJECT SERVERS - 2

- **Should object servers isolate memory for object instances?**
  - Share neither code nor data
  - May be necessary if objects couple data and implementation
- Object server threading designs:
  - Single thread of control for object server
  - One thread for each object
  - Servers use separate thread for client requests
- Threads created on demand ***vs.*** Server maintains pool of threads
- **What are the tradeoffs for creating server threads on demand vs. using a thread pool?**

February 4, 2020 | TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma | L9.66

66

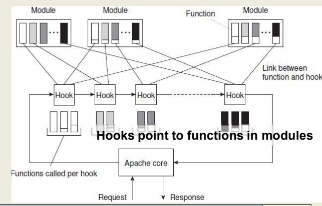
EJB – ENTERPRISE JAVA BEANS

- EJB- specialized Java object hosted by a EJB web container
- 4 types: stateless, stateful, entity, and message-driven beans
- Provides "middleware" standard (framework) for implementing back-ends of enterprise applications
- EJB web application containers integrate support for:
  - Transaction processing
  - Persistence
  - Concurrency
  - Event-driven programming
  - Asynchronous method invocation
  - Job scheduling
  - Naming and discovery services (JNDI)
  - Interprocess communication
  - Security
  - Software component deployment to an application server

67

APACHE WEB SERVER

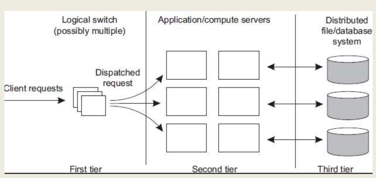
- Highly configurable, extensible, platform independent
- Supports TCP HTTP protocol communication
- Uses hooks – placeholders for group of functions
- Requests processed in phases by hooks
- Many hooks:
  - Translate a URL
  - Write info to log
  - Check client ID
  - Check access rights
- Hooks processed in order enforcing flow-of-control
- Functions in replaceable modules



68

SERVER CLUSTERS

- Hosted across an LAN or WAN
- Collection of interconnected machines
- Can be organized in tiers:
  - Web server → app server → DB server
  - App and DB server sometimes integrated



69

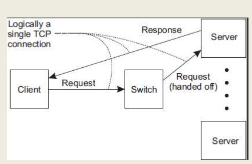
LAN REQUEST DISPATCHING

- Front end of three tier architecture (logical switch) provides distribution transparency – hides multiple servers
- Transport-layer switches: switch accepts TCP connection requests, hands off to a server
  - Example: hardware load balancer (F5 networks – Seattle)
  - HW Load balancer - OSI layers 4-7
- Network-address-translation (NAT) approach:
  - All requests pass through switch
  - Switch sits in the middle of the client/server TCP connection
  - Maps (rewrites) source and destination addresses
- Connection hand-off approach:
  - **TCP Handoff**: switch hands of connection to a selected server

70

LAN REQUEST DISPATCHING - 2

- Who is the best server to handle the request?
- Switch plays important role in distributing requests
- Implements load balancing
- **Round-robin** – routes client requests to servers in a looping fashion
- **Transport-level** – route client requests based on TCP port number
- **Content-aware request distribution** – route requests based on inspecting data payload and determining which server node should process the request



71

WIDE AREA CLUSTERS

- Deployed across the internet
- Leverage resource/infrastructure from Internet Service Providers (ISPs)
- Cloud computing simplifies building WAN clusters
- Resource from a single cloud provider can be combined to form a cluster
- **For deploying a cloud-based cluster (WAN), what are the implications of deploying nodes to:**
- (1) a single availability zone (e.g. us-east-1e)?
- (2) across multiple availability zones?

72

WAN REQUEST DISPATCHING

- Goal: minimize network latency using WANs (e.g. Internet)
- Send requests to nearby servers
- Request dispatcher: routes requests to nearby server
- **Example**: Domain Name System
  - Hierarchical decentralized naming system
- Linux: find your DNS servers:

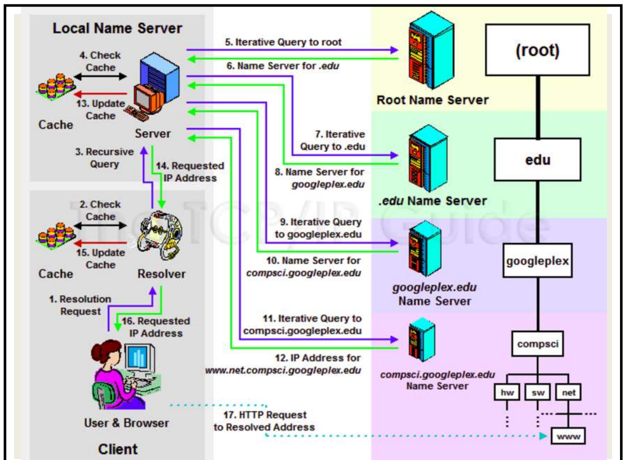
```
# Find you device name of interest
nmcli dev
# Show device configuration
nmcli device show <device name>
```


73

DNS LOOKUP

- First query local server(s) for address
- Typically there are (2) local DNS servers
  - One is backup
- Hostname may be cached at local DNS server
  - E.g. www.google.com
- If not found, local DNS server routes to other servers
- Routing based on components of the hostname
- DNS servers down the chain mask the client IP, and use the originating DNS server IP to identify a local host
- ***Weakness:*** *client may be far from DNS server used. Resolved hostname is close to DNS server, but not necessarily close to the client*

74



75

DNS: LINUX COMMANDS

- `nslookup <ip addr / hostname>`
- Name server lookup – translates hostname or IP to the inverse
- `traceroute <ip addr / hostname>`
- Traces network path to destination
- By default, output is limited to 30 hops, can be increased

76

DNS EXAMPLE – WAN DISPATCHING

- Ping www.google.com in WA from wireless network:
  - nslookup: 6 alternate addresses returned, choose (74.125.28.147)
  - Ping 74.125.28.147: Average RTT = **22.458 ms (11 attempts, 22 hops)**
- Ping www.google.com in VA (us-east-1) from EC2 instance:
  - nslookup: 1 address returned, choose 172.217.9.196
  - Ping 172.217.9.196: Average RTT = 1.278 ms (11 attempts, 13 hops)
- From VA EC2 instance, ping WA *www.google* server
- Ping 74.125.28.147: Average RTT 62.349ms (11 attempts, 27 hops)
- Pinging the WA-local server is ~60x slower from VA
- From local wireless network, ping VA us-east-1 google :
- Ping 172.217.9.196: Average RTT=81.637ms (11 attempts, 15 hops)

77

DNS EXAMPLE – WAN DISPATCHING

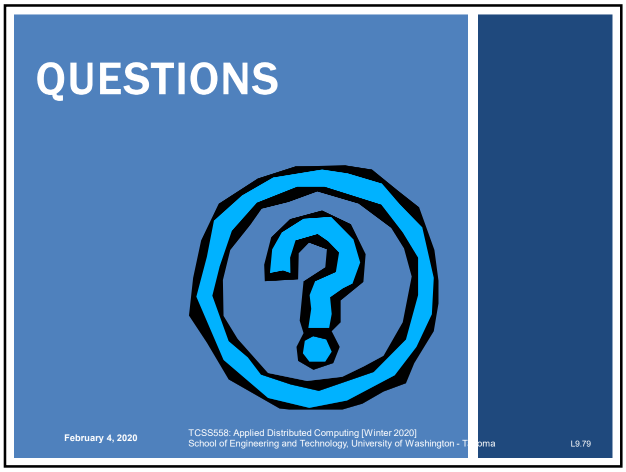
- Ping www.google.com in WA from wireless network:
  - nslookup: 6 alternate addresses returned, choose (74.125.28.147)

Latency to ping VA server in WA: ~3.63x
WA client: local-google 22.458ms to VA-google 81.637ms

Latency to ping WA server in VA: ~48.7x
VA client: local-google 1.278ms to WA-google 62.349!

- From local wireless network, ping VA us-east-1 google :
- Ping 172.217.9.196: Average RTT=81.637ms (11 attempts, 15 hops)

78


QUESTIONS
February 4, 2020
TCSS558: Applied Distributed Computing [Winter 2020]
School of Engineering and Technology, University of Washington - Tacoma
L9.79

79