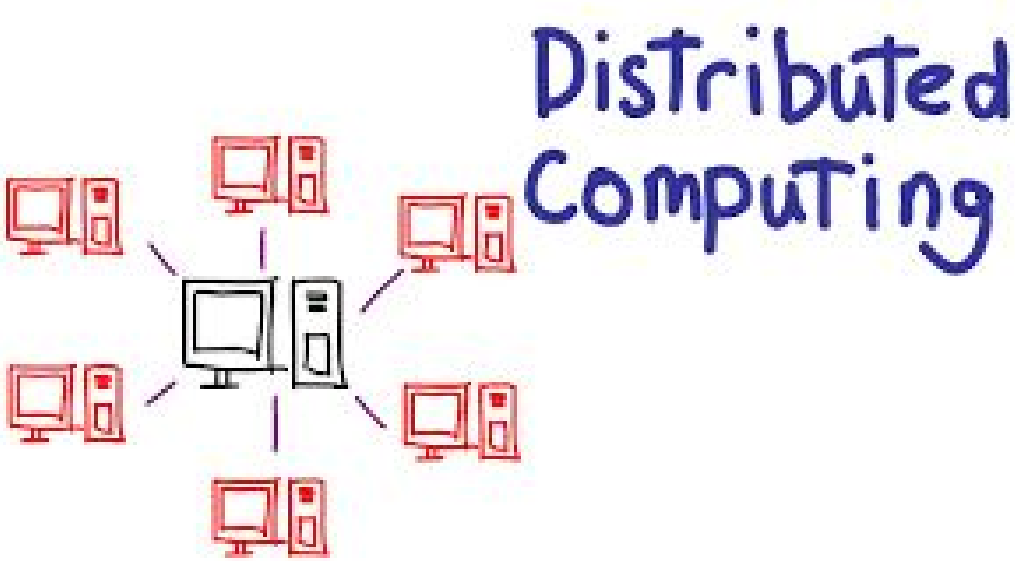


QUIZZES

Quiz 1:

1.The 4B/5B encoding scheme ensures that the translated codes (4-bit to 5-bit encodings) do not result in more than 1 leading 0 and no more than 2 trailing 0's. This is because **the 5 bit codes are transmitted using NRZI that performs transitions for 1s.**
2. The efficiency of the 4B/5B encoding scheme is **80% or ⅘**
3. The efficiency of the Manchester encoding scheme is **50% or ½**
4. Circuit switched networks require circuits and their accompanying state to be established before data can be transferred. **True**
5. Landline-based telephone networks **DID NOT** popularize packet-switched networks.
6. When multiplexing data onto a physical link, setting aside time quanta is not feasible because: Number of flows must be known ahead of time, the link may be idle if a particular flow is not active, and typical network usage patterns are bursty .
7. The Delay x Bandwidth product tells us how many bits fit in a network pipe. What is the maximum number of pipes that a sender can fill before it receives an acknowledgement from the receiver? **Two pipes**
8. The bandwidth and latency for a link **ARE NOT** tightly-coupled.
9. In layered architectures, services at higher layers **ARE** implemented in terms of lower layers
10. All communications in distributed systems are based on sending/receiving messages.

Quiz 2:

1. The OSI architecture implies a strict layering of the protocol stack
2. The Internet architecture **does not** imply a strict layering of the protocol stack.

3. The process-per-protocol model results in minimal overheads and highest possible efficiency when processing messages. **FALSE**
4. In the Internet architecture, every node (i.e. hosts, routers, switches, etc.) **needs** to include support for the IP protocol.
5. IP **does not** attempt to recover from reassembly failures at the receiver side by sending a retransmission request to the source
6. Packet formats **are** designed to align on 32-bit boundaries to simplify the task of processing them in software.
7. The demultiplexing key **is useful** for identifying the higher-level protocol that a message should be sent to.
8. The presence or absence of Options in the IP header cannot be determined solely from the length of the header (HLen) **FALSE**
9. Consider an IP packet whose checksum field indicates that the header has been corrupted in transit. The IP checksum **DOES NOT** double as an error-correcting code and can be used to recover from data corruptions.
10. Packets transmitted using IP **can** be duplicated i.e., a packet can never be received more than once

Quiz 3:

- IPv6 will **NOT** fragment a packet that is larger than MTU (Maximum Transmission Unit).
- IPv6 does **NOT** include a 32-bit checksum.
- IPv6 does **NOT** include a header length because it is fixed at 40 bytes.
- TCP and UDP are limited to 0-65535 because the port field i their headers is 16 bits.
- If a UDP port queue is full when a datagram packet arrives it will be discarded.
- Size of TCP sliding window is based on memory set aside by receiver of the particular connection.
- In TCP each byte has a sequence number associated with it.
- Flow control in TCP is when sender tries not to overrun capacity of receiver
- Threads have their own stack
- Threads within a process share the process heap.

Quiz 4:

- Consider a class A that has two synchronized methods s1() and s2(); this class also has two unsynchronized methods u1() and u2(). Class A was used to create two object instances, a1 and a2 , in a particular process P. Within the process P, there are N threads that are represented as T_1 , T_2 , …, T_N.
    - Threads T_1 , T_2 , …, T_N **CAN** all be active in instance a1 at the same time **AND** have different program counters.
    - Threads T_1 and T_2 **CAN NOT** be active inside method a1.s1() at the same time.
    - Thread T_1 **CAN NOT** be active in a1.s1() and Thread T_2 can be active in a1.s2() at the same time.

    - Thread T_1 **CAN** be active in a1.u1() and Thread T_2 can be active in a1.u2() at the same time.
    - Thread T_1 **CAN** be active in a1.s1() and Thread T_2 can be active in a2.s2() at the same time.
- Java **DOES NOT** use the same mutex lock to ensure thread safety for both static synchronized methods and non-static synchronized methods.
- A program that declares all its variables volatile **WILL NOT** execute faster than the same program where these variables were not declared volatile.
- The volatile keyword for a variable **IS NOT** used for performance reasons because it allows Threads to cache that variable in a register.
- Using the volatile keyword **DOES NOT** ensure that increment (++) and decrement (--) operations on an integer variable is atomic and thread safe.

Quiz 5:

- Heisenbugs are real.
- Protect your variables so that you can reason about thread-safety
- wait()/notify() has a race condition (use synchronized wait)
- Key to thread safe programming is how an object is used rather than what it does
- Volatile keyword ensures that all access to variable will be redirected to main memory
- Stateless objects are always thread safe
- The increment operator (++) of type long must be synchronized because it is not atomic
- We must synchronize reads as well as writes so that we don't read an inconsistent state
- If you are holding a lock, you can still acquire it to increment a count of how many releases you must make to relinquish the lock.
- Synchronizing all public methods of all classes within a program will **NOT** guarantee thread-safety

Quiz 6:

- Performing Deep-copies on structures and returning references of the copy is a way of coping with thread safety issue of returning a data strictures.
- A reference to an object should never escape in its constructor.
- Even synchronized collections don't provide thread-safe compound operations.
- Iterators of synchronized collections are not thread safe. And you must synchronize on the collection while you use its iterator.
- Synchronizers are objects that coordinate the flow of other threads based on the internal state of the synchronizer.
- Latches can not be reset once they reach their terminal state.

Quiz 7:

- A Semaphore can be initialized to the count of the resource
- You acquire a lock to grab it and release to relinquish a resource
- You can not skip the lock acquisition phase
- Streaming through disk accesses is faster than accesses dominated by seeks.
- It is possible that one of the mappers may not have generated intermediate outputs that need to be routed to Rx

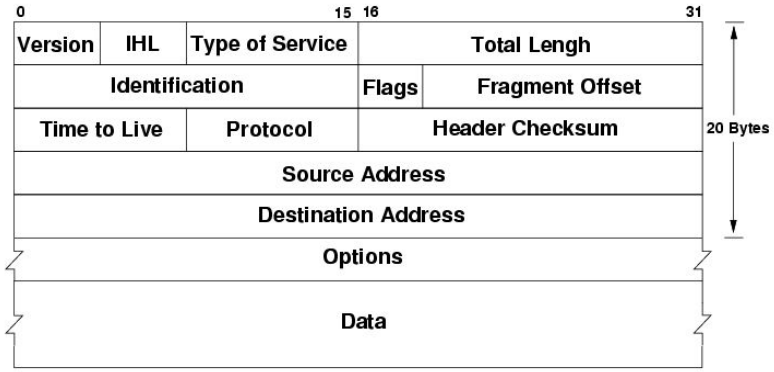
- Intermediate values are sorted by key and grouped by keys.

NETWORKING (25%) (Lectures 1-5)

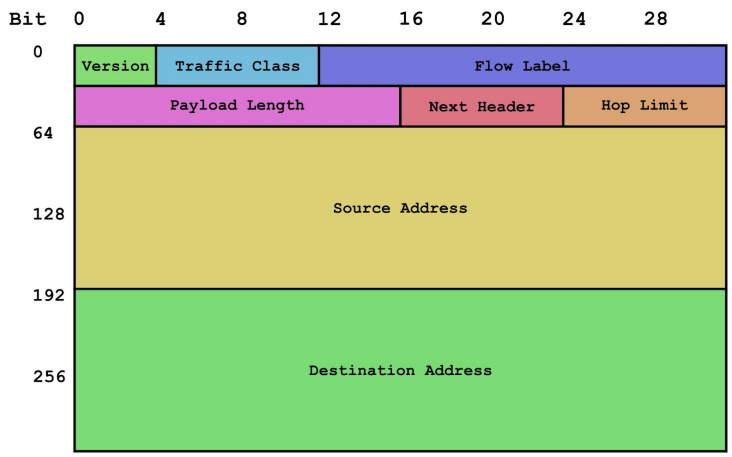
- Servers create a ServerSocket that listens on a port with the blocking call accept() which creates a Socket on new port
- Socket are bidirectional communication, write & read ends on both server and client.
- We are sending signals of 1's and 0's
- Data can flow in both directions: Yes → full duplex; No → half duplex
- Non-return to zero (NRZ)
    - Map 1 to a high signal and 0 to a low signal
    - Problem **baseline wander**:
        - Receiver keeps average of the signal
        - Lots of consecutive 1/0 will make it difficult to detect significant change
    - Problem **Clock recovery**
        - The clocks would need to be perfectly synchronized otherwise it's not possible to decode the signal
- Manchester encoding
    - 0 is a low to high transition
    - 1 is a high to low transition
    - 50% efficient
    - Doubles the rate at which signal transitions are made on the link
        - Receiver has half the time to detect each pulse
- NRZI: Make a transition from current signal to encode a 1, and stay at current signal to encode a 0, but this doesn't solve the problem with consecutive 0's
- 4B/5B encoding: 80% efficient, addresses the issues with the manchester encoding
    - Inserts an extra bit into the bitstream
        - 5-bit codes are carefully selected where no more than 1 leading 0 and no more than 2 trailing 0's
        - No pair of 5 bit codes results in more than 3 consecutive 0's being transmitted
- Two different types of networks:
    - Circuit switched and packet switched
    - Circuit switch
        - Establishes a dedicated circuit across a set of links that is used until termination, commonly used by the telephone system
        - Use a store and forward method
            - Receive complete packet over the link, store the packet in memory, and forward the packet to the next node
- Multiplexing

    - Sharing the CPU among multiple processes
    - Synchronous time division multiplexing STDM and Frequency division multiplexing FDM
    - Issues with STDM and FDM
        - Limited to specific situations, the max num of flows is fixed which needs to be known ahead of time
        - Idle links
    - Statistical multiplexing → data is transmitted from each flow **on demand**
    - An upper bound exists on the size of the packet
        - These can be fragmented and then reassembled on the client side
- Bandwidth and latency
    - Bandwidth is also known as **throughput** → num bits transmitted over the network in a given time
    - Latency is how long it takes for the message to go from one end of the network to another. Contributors: speed of light delay, amount of time to transmit data and queueing delays
    - Transcontinental channel:
        - 50 ms one-way latency with a bandwidth of 45 Mbps → this pipe can hold 280 KB
        - (50 * 10^-3 seconds) * (45 * 10^6 bits/sec) = 2.25 * 10^6 bits
    - Sending 1 MB data over a cross country link. Delay is 100 ms
        - Pipe: 100 * 10^-3 * 10^6 = 100 Kb = 0.1 Mb
        - Need 80 pipes to transmit 1 MB → 8 Mb / 0.1 Mb = 80
- Network Architecture
    - No shared memory we need to send messages in between address spaces
    - Messages utilize a layering protocol to wrap the message
    - Demux key is an 8-bit field that is used to decode the protocol TCP {6} UDP {17}
- OSI Network Architecture
    - Open Systems Interconnections → partitions network functionality into 7 layers
        - Physical Layer: handles transmission of **raw bits**
        - Data Link layer: Collects stream of bits into **frames**, compute checksum for the frame. These would be implemented on network adapters and device drivers
        - Network layer: the data that is exchanged is **packets**, handles routing among nodes in a packet-switched network
        - **The above three layers are implemented on all network nodes**
        - Transport layer: implements process-process channel, this will send **Messages**
        - Presentation layer: this is the format of data exchanged between peers
        - Session: this is the namespace to the different transport-streams that are part of the application
- Internet Architecture
    - 4 level model that **DOES NOT** imply strict layering

- Protocol implementation:
  - Process-per-protocol: inefficient due to the number of context-switches that would be required at each level of the protocol graph
  - Process-per-message: protocol graph traversed in sequence of procedure calls, when a message arrives dispatch a process to move the message up the protocol graph



  - Things that can go wrong when packets are in transit: Lost, corrupted, misdelivered, and out of order and duplicates.
  - IPv4 packet headers are represented in 32-bit aligned words
    - HLen: specifies header len in 32-bits, determines the presence of options.
    - TOS: meant to be how the datagram should be handled but was not widely implemented
    - Length: total datagram length in bytes. The maximum size is 2^16 bytes
    - Second word of the IP packet: information about fragmentation {Ident, Flags, Offset}
      - Ident is used to enable fragment reassembly
      - If all of the fragments don't arrive the receiver gives up and discards fragments that did not arrive
      - **IP DOES NOT** attempt to recover from the missing fragments
    - TTL: time to live, The hop count, not timer
    - Protocol field → this is the Demux key ex TCP {6} UDP {17}
    - Checksum
    - Source Address and the Destination address
- Every network has a maximum transmission unit MTU
  - This is the largest IP datagram that it can carry in its frame
  - This is smaller than the largest packet size of the network
    - IP datagram needs to fit in the payload of the link-layer frame
- IPv6
  - Source and destination addresses are 128 bit fields
  - Treats options as extension headers
  - This protocol did away with fragmentation: this protocol must perform path MTU discovery, perform end-to-end fragmentation,



or send packets no larger than the default MTU of 1280
    - Traffic class: defined for QoS
    - Flow Label: if it is non zero: serves as a hint to routers and switches with multiple outbound paths that these packets should stay on the same path so that they will not be re ordered
    - Payload Length: 16 bits that are dedicated to the size of the payload including the extension headers
    - Next Header: 8 bits that specifies the type of the next header
    - Hop limit: 8 bits that basically replaces TTL of IPv4
    - **NO CHECKSUM:** transport layer expected to provide error handling
    - [IPv6 Header][Extension Headers][Upper Layer Protocol Data Unit (PDU)]
        - Above would be an entire IPv6 packet, and the Extension headers and the PDU would consist of the payload
    - Extension header:
        - If it holds a non zero value it defines extension headers, these could include info for routers, route definitions, fragment handling, authentication, and encryption. These will come after original header and before the payload and will contain an addition Next header
- UDP → User Datagram Protocol
    - Processes indirectly identify each other, the abstract locator is called a port
    - Utilizes the demultiplexing key and is typically implemented as a message queue
    - If the queue is full then the message is lost
    - Provides a checksum field
- TCP
    - These packets contain Acknowledgements, and if after x amount of time, the packet will retransmit a new ACK. THe sender waits for ACK before sending next frame
    - Includes a flow control mechanism that throlles how fast TCP sends the data preventing from overloading the network. Don't exceed the capacity of the receiver
    - Sliding window frames:
        - TCP Send buffer: LastByteAcked <= LastByteSent, LastByteSent <= LastByteWritten
        - TCP Receive Buffer: LastByteRead < NextByteExpected, NextByteExpected <= LastByteReceived + 1
        - Advertised window: MaxReceiveBuffer - ((NextByteExpected - 1) - LastByteRead))
        - EffectiveWindow = AdvertisedWindow - (LastByteSent - LastByteAcked)
            - This value should be greater than 0 before source can send more data

THREADS (50%) (Lectures 5-11)

- **A thread is a discrete task that operates on data shared with other threads.**
- Threads have the ability to share the address space (and all of its data) among themselves.
- Each thread needs its own stack
    - Contains one frame for each procedure called but not returned from
    - A frame contains:
        - Local variables
        - Procedures return address.
- Each thread calls different procedures
    - Has a different execution history
- Threads that run within the same application process
    - Share the memory space of the process
    - However two diverse apps within the same machine may not communicate so well.
- Threads within a process can access and share any object on the heap
    - Each thread has space for its own local vars (on the stack)
- Lifecycle of a thread: Creation→Starting→Terminating→Pausing/Suspending/Resuming
- Thread Creation
    - Threads are instances of the thread class
    - Threads require 4 pieces of information to construct
        - Name: default is Thread-N; N is a unique number
        - Runnable Target: List of instructions; default run() of thread itself.
        - Group: a thread is assigned to the group of the thread that calls the constructor.
        - Stack size: store temporary variables during method execution.
- Starting Thread
    - Once constructed it exists but it's in a waiting state.
    - When ready for a thread to begin execution
    - Call the start() method, performs housekeeping and then calls run() method.
        - When start() returns
            - The original thread which just returned from calling start()
            - The newly started thread that is executing its run() method
    - The thread is now alive and can call isAlive() method returns the state of a thread
- Terminating Thread
    - Terminates when a threads run() method returns
    - Should NOT use the Thread class stop() method, contains a race condition.
- Pausing/Suspending/Resuming Thread
    - … L6 page 15 (gonna do other parts first)
- **A thread can hold multiple locks**

Synchronization

- Not possible to execute the same method in one thread while the sync method is running in another thread.
- Two or more synced methods of the same object can never run in parallel in separate threads.
    - But any number of unsynchronized methods of the same object can be running.
- A lock can be obtained for each class called the class lock
    - The class lock is the object lock of the class object that models the class
        - There is only 1 Class object per class
        - Allows for synchronization of static methods
    - The class lock can be grabbed independently of the object lock.
    - If a non-static sync method calls a static sync method, it acquires both locks
- Advantages:
    - Serializes accesses to sync methods in an obj
- Disadvantages:
    - Not suitable for controlling lock scope
    - Too primitive in some cases
- Synchronization policy specifies a combination of
    - Immutability
    - Thread confinement
    - Locking
- Sync Methods vs Sync Blocks
    - Possible to use only the sync block mechanism to sync whole method
        - Depends on implementation
    - Establish as small a lock scope as possible
-

Explicit Locks

- Two important methods: lock() and unlock().
- Using explicit locks like synchronized keyword
    - Call lock() at the start of the method, and call unlock() at the end.
- An actual object that represents the lock
    - Store, pass, or discard
- ReentrantLock
    - Implements Lock interface and allows locks to be granted fairly
- Advantages:
    - Grab and release whenever
    - Possible for two objects to share the same lock
    - Lock can be attached to data, or groups of data
    - Can adjust lock scope, scope specific to the problem, not just the obj
- Disadvantages:
    -

Mutex Locks

- Locks Mutual Exclusion

Semaphores
- Counting Lock

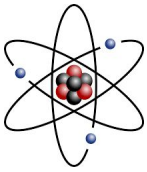
Wait()/notify()
- Thread can wait(timeout) until notified()
- Must be used in conjunction with synchronized lock to avoid race condition
    - Upon successfully waiting you will relinquish lock once you are in a state that you can be notified. (Also use synchronized notify)
- Race condition in wait-notify
        - First thread tests condition and begins to call wait
        - Second thread sets condition and notifies, which goes unheard since the first thread has not completely waited yet
        - The first thread finally calls wait
    - Using synchronized blocks and methods can solve this race condition
- If notify is called when no other thread is waiting the notification is lost
    - Wait-notify mechanism has no knowledge about the condition about which it notifies
    - If there are multiple threads waiting, then there is no way to determine which thread will be granted the notification
- notifyAll()
    - All threads that are waiting on an object are notified
    - When waiting threads receive a notifyAll they must:
        - Decide which thread should continue
        - Decide which threads should wait again
            - All threads wake but still have to re-acquire the object lock, must wait for the the lock to be freed

Latches
- Wait for events
    - Threads cannot pass until it is in its terminal state
    - Provides ability to change state
    - Once latch is open, a.k.a. In terminal state it cannot change state to another state

Barriers
- Wait for other threads
    - All threads must come together at barrier point before they can proceed

Atomic

- 



- Non compound operations, no ++ -- +=

Exchange

- Threads can trade Objects at an exchange.
    - Thread can access one of the two sides of the exchange via a blocking call that will wait until a thread meets it on the other side with an object to trade.

Volatile

- All access to a volatile variable a directed to main memory
    - Avoids old cached values to be used.

Thread Safe Object/Programs

- When deciding if an object should be thread safe, think how the object is **used**, not what it **does**
- To make a object thread safe, use synchronization to coordinate access to **mutable state**
- Don't share state vars across threads, make state vars immutable, use sync to coordinate access to the state var.
- Stateless objects are **always thread safe**

MapReduce (25%) (Lectures 11-13)

- MapReduce IS HERE
  - https://www.cs.colostate.edu/~cs455/lectures/CS455-L11-ThreadSafetyAndMapReduce.pdf
  - https://www.cs.colostate.edu/~cs455/lectures/CS455-L12-MapReduce.pdf
  - https://www.cs.colostate.edu/~cs455/lectures/CS455-L13-Hadoop.pdf

- Requires HDFS or GFS (hadoop distributed file system or google file system)
  - Data stored in 3 places
- Pushes computations to data.
  - Computation for a map task pushed to the node of the 3 nodes holding that tasks associated partition of input data that is the least busy.
- Map
  - Mappers take inputs as <Key, Value> pairs
  - Typically Keys of outputs are derived or contained from the values of inputs
  - Mappers produce intermediate <Key, Value> outputs that are sent to [key-space]%R reducer
  - Each mapper produces R files of intermediate outputs using **APPEND ONLY**
    - Best effort could be partially combined by the combiner acting on merging buffered intermediate values that have not yet been appended to the output file.
- Reduce
  - R reducers each receives a partition of the intermediate outputs to Merge.
    - Partition is decided by [key-space] % R
  - Reducers take inputs as <Key, Value> pairs
    - Input Key is Value of mapper's intermediate outputs
    - Inputs are sorted by Key and values are Grouped
      - <k0, [v0, v1, v2]>
      - <k1, [v0, v1,...vn]>
  - Reducers produce outputs as <Key, Value> outputs
- Type chain
  - map(k1, v1) -> list(k2, v2)
  - reduce(k2, list(v2)) -> list(v2)
- Fault Tolerance
  - If a map task fails (number of failed pings) it is reset to idle state and becomes eligible for rescheduling
  - Optional - Skipping bad records
    - Detect deterministic crashes & report to master
    - Skip records that have crashed upon re-invocations
- Task Granularity

- Subdivide map phase into M pieces
- Subdivide reduce phase into R pieces
- M + R >> (greater than typically) number of worker machines
- Each worker performs many tasks