Recall: Distributed Consensus Making

• Consensus problem
  – All nodes propose a value
  – Some nodes might crash and stop responding
  – Eventually all nodes must decide on a value from set of proposed values

• Distributed Decision Making
  – Choose between "true" and "false"
  – Or Choose between "commit" and "abort"

• Equally important (but often forgotten!): make it durable!
  – How do we make sure that decisions cannot be forgotten?
  » This is the "D" of "ACID" in a regular database
  » In a global-scale system?
  » What about erasure coding or massive replication?
  » Like Blockchain applications!

Recall: Two-Phase Commit Protocol

• Persistent stable log on each machine: keep track of whether commit has happened
  – If a machine crashes, when it wakes up it first checks its log to recover state of world at time of crash

• Prepare Phase:
  – The global coordinator requests that all participants will promise to commit or rollback the transaction
  – Participants record promise in log, then acknowledge
  – If anyone votes to abort, coordinator writes "Abort" in its log and tells everyone to abort; each records "Abort" in log

• Commit Phase:
  – After all participants respond that they are prepared, then the coordinator writes "Commit" to its log
  – Then asks all nodes to commit; they respond with ACK
  – After receive ACKs, coordinator writes "Got Commit" to log

Log used to guarantee that all machines enter commit or don’t

Discussion (1/2)

• Why is distributed decision making desirable?
  – Fault Tolerance!
  – A group of machines can come to a decision even if one or more of them fail during the process
  – Why is distributed decision making desirable?

• Why is 2PC not subject to the General’s paradox?
  – Because 2PC is about all nodes eventually coming to the same decision — not necessary that they do so at the same time!
  – Allow us to reboot and continue allows time for collecting and collating decisions
  – After receive ACKs, coordinator writes "Got Commit" to log
  – Then asks all nodes to commit; they respond with ACK
Undesirable feature of Two-Phase Commit: Blocking

- One machine can be stalled until another site recovers:
  - Site B writes "prepared to commit" record to its log, sends a "yes" vote to the coordinator (site A) and crashes
  - Site A crashes
  - Site B wakes up, check its log, and realizes that it has voted "yes" on the update. It sends a message to site A asking what happened. At this point, B cannot decide to abort, because update may have committed
  - B is blocked until A comes back

- A blocked site holds resources (locks on updated items, pages pinned in memory, etc) until learns fate of update

Alternatives to 2PC

- Three-Phase Commit: One more phase, allows nodes to fail or block and still make progress.
- Paxos: An alternative used by Google and others that does not have 2PC blocking

General's Problem

- Byzantine General's Problem (n players):
  - One General and n-1 Lieutenants
  - Some number of these (f) can be insane or malicious

- The commanding general must send an order to his n-1 lieutenants such that the following Integrity Constraints apply:
  - IC1: All loyal lieutenants obey the same order
  - IC2: If the commanding general is loyal, then all loyal lieutenants obey the order he sends

- General: Attack!
  - Retract!
  - Attack!

- Lieutenant: Attack!
  - Retract!
  - Attack!

- Malicious!

Impossible Results:

- Cannot solve Byzantine General's problem with n=3 because one malicious player can mess things up
- With f faults, need n > 3f to solve problem

- Various algorithms exist to solve problem
- Original algorithm has #messages exponential in n
- Newer algorithms have message complexity O(n^2)
- One from MIT, for instance (Castro and Liskov, 1999)
- Use of BFT (Byzantine Fault Tolerance) algorithm
  - Allow multiple machines to make a coordinated decision even if some subset of them (< n/3) are malicious
Networking protocols: many levels
- Physical level: mechanical and electrical network (e.g., how are 0 and 1 represented)
- Link level: packet formats/error control (for instance, the CSMA/CD protocol)
- Network level: network routing, addressing
- Transport level: reliable message delivery

Protocols on today's Internet:
- Ethernet
- WiFi
- LTE
- IP
- UDP
- TCP
- RPC
- NFS
- WWW
- e-mail
- ssh

Broadcast Networks Details
- Media access control (MAC) address:
  - 48-bit physical address for hardware interface
  - Every device (in the world!?) has a unique address
- Delivery:
  - When you broadcast a packet, how does a receiver know who it is for?
  - Put header on front of packet: [Destination MAC Address | Packet]
  - Everyone gets packet, discards if not the target
  - In Ethernet, this check is done in hardware
    - No OS interrupt if not for particular destination

Point-to-point networks
- Why have a shared bus at all? Why not simply and only have point-to-point links?

Broadcast Networks
- Why have a shared bus at all? Why not simply and only have point-to-point links?
The Internet Protocol (IP)

- **Internet Protocol**: Internet’s network layer
- **Service it provides**: “Best-Effort” Packet Delivery – Tries its “best” to deliver packet to its destination
  - Packets may be lost
  - Packets may be corrupted
  - Packets may be delivered out of order
- **IP Is a Datagram service!**
  - Routes across many physical switching domains (subnets)

IPv4 Address Space

- **IP Address**: a 32-bit integer used as destination of IP packet
  - Often written as four dot-separated integers, with each integer from 0—255 (thus representing 8x4=32 bits)
  - Example CS file server is: 169.229.60.83
  - `0xA9E53C53`

- **Internet Host**: a computer connected to the Internet
  - Host has one or more IP addresses used for routing
    - Some of these may be private and unavailable for routing
    - Not every computer has a unique IP address
      - Groups of machines may share a single IP address
      - In this case, machines have private addresses behind a “Network Address Translation” (NAT) gateway

- **Subnet**: network connecting hosts with related IP addresses
  - A subnet is identified by a 32-bit value, with the bits which differ set to zero, followed by a slash and a mask
  - Example: 128.32.131.0/24 designates a subnet in which all the addresses look like 128.32.131.XX
  - Same subnet: 128.32.131.0/255.255.255.0

- **Mask**: The number of matching prefix bits
  - Expressed as a single value (e.g., 24) or a set of ones in a 32-bit value (e.g., 255.255.255.0)
  - Often routing within subnet is by MAC address (smart switches)

IPv4 Packet Format

- **IP Packet Format**: IP Datagram:
  - an unreliable, unordered, packet sent from source to destination
  - Function of network – deliver datagrams!
  - **Header**: 20 bytes
  - **Fields**:
    - `IP Version`: 4 (IP version 4)
    - `Header Length` (in 4-byte units): 5
    - `Type of Service`: 8
    - `Time to Live`: 8
    - `Identification`: 16
    - `Flags`: 3
      - `Bit 0`: Don’t Fragment
      - `Bit 1`: More Fragments
      - `Bit 2`: Don’t Cache
    - `Fragment Offset`: 13
      - Value of fragment offset is used to compute relative offset of fragments
      - Each fragment has a unique ID
      - ID is used with the offset to reassemble packets
    - `Protocol`: 8
      - Either TCP or UDP
    - `Header Checksum`: 16
    - `Source IP Address`: 32
    - `Destination IP Address`: 32
    - `Options` (if any): 0
    - `Data`: 0

Wide Area Network

- **Wide Area Network (WAN)**: network that covers a broad area (e.g., city, state, country, entire world)
- **Wide Area Network Example**: Internet is a WAN
- **WAN connects multiple physical layer networks (LANs)**
  - **Datalink layer networks are connected by routers**
  - Different LANs can use different communication technology (e.g., wireless, cellular, optical, wired)

Internet Protocol Layer

- **IP layer**: Network layer
  - Provides an end-to-end service between two hosts
  - Each packet must have a unique source and destination address
  - **Transport layer** services (e.g., TCP, UDP) are used above IP
  - Transport layer provides end-to-end services
  - **Routing** is responsible for determining the path a packet takes from source to destination
  - **Network Address Translation (NAT)** is used to map private to public IP addresses
  - NAT is used to enable private network access to the Internet

TCP/IP Reference Model

- **Network layer (Layer 3)**: Responsible for packet delivery
  - **IP** is the main protocol used at this layer
  - **DNS** is used to translate domain names to IP addresses
  - **ARP** is used to translate IP addresses to MAC addresses
Routers

- Forward each packet received on an incoming link to an outgoing link based on packet's destination IP address (towards its destination)
- Store & forward: packets are buffered before being forwarded
- Forwarding table: mapping between IP address and the output link

Packet Forwarding

- Upon receiving a packet, a router reads the IP destination address of the packet
- Consult its forwarding table
- Forward packet to corresponding output port

IP Addresses vs. MAC Addresses

- Why not use MAC addresses for routing?
  - Doesn't scale
  - Analogy: MAC address à SSN, IP address à home address
  - MAC address: uniquely associated with device for the entire lifetime of the device
  - IP address: changes as the device location changes

Why does packet forwarding using IP address scale?

- Because IP addresses can be aggregated
- E.g., all IP addresses at UC Berkeley start with 0xA9E5, i.e., any address of form 0xA9E5**** belongs to Berkeley
- Thus, a router in NY needs to keep a single entry for all hosts at Berkeley

IP Addresses vs. MAC Addresses

- Analogy: give this letter to person with SSN: 123-45-6789 vs. give this letter to "John Smith, 123 First Street, LA, US"
Midterm 3: Thursday 4/28, 7-9PM

Naming in the Internet

• How to map human-readable names to IP addresses?

Mechanism: Domain Name System (DNS)
- CS 896/898 – hands-on exercises on DNS
- CS 896/898 – hands-on exercises on DNS
- CS 896/898 – hands-on exercises on DNS
- CS 896/898 – hands-on exercises on DNS
- How to map human-readable names to IP addresses?

How Important is Correct Resolution?

• If attacker manages to give incorrect mapping:
  - Can get someone to route to server, thinking that they are talking to a different server
  - Give up username and password

• Is DNS Secure?
  - Definitely a weak link
  - What if “response” returned from different server than original query?
  - Get person to use incorrect IP address!
  - Attempt to avoid substitution attacks:
    - Query includes random number which must be returned
  - In July 2008, hole in DNS security located!
  - Dan Kaminsky (security researcher) discovered an attack that broke DNS security
  - High profile, highly advertised need for patching DNS
  - Big press release, lots of mystery
  - Security researchers told no speculation until patches applied

DNS is a hierarchical mechanism for naming
- Name divided in domains, right to left: www.eecs.berkeley.edu
- Each domain owned by a particular organization
- Top level handled by ICANN (Internet Corporation for Assigned Numbers and Names)
- Subsequent levels owned by organizations
- Resolution: series of queries to successive servers
- Caching: queries take time, so results cached for period of time
Layering: building complex services from simpler ones

- Each layer provides services needed by higher layers by utilizing services provided by lower layers.

The physical/link layer is pretty limited:

- Packets are of limited size (called the "Maximum Transfer Unit" or MTU: often 200–1500 bytes in size).
- Routing is limited to within a physical link (wire) or perhaps through a switch.

Our goal in the following is to show how to construct a secure, ordered, message service routed anywhere:

<table>
<thead>
<tr>
<th>Physical Reality: Packets</th>
<th>Abstraction: Messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Limited Size (MTU)</td>
<td>Arbitrary Size</td>
</tr>
<tr>
<td>Unordered (sometimes)</td>
<td>Ordered</td>
</tr>
<tr>
<td>Unreliable</td>
<td>Reliable</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Machine-to-Machine</th>
<th>Process-to-Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>Only on local area net</td>
<td>Routed anywhere</td>
</tr>
<tr>
<td>Asynchronous</td>
<td>Synchronous</td>
</tr>
<tr>
<td>Insecure</td>
<td>Secure</td>
</tr>
</tbody>
</table>

Recall: IPv4 Packet Format

- IP Datagram:
  - An unreliable, unordered, packet sent from source to destination.
  - Function of network.
  - Deliver datagrams!

- IPv4 header:
  - 20 bytes
  - Length (16 bits): Size of datagram (header+data)
  - Flags & Fragmentation:
    - to split large messages
  - Time to Live (hops)
  - Protocol
  - Flags
  - Options (if any)
  - Data

Building a messaging service on IP:

- Process to process communication.
  - Basic routing gets packets from one machine to another.
  - What we really want is routing from one process to another.

- Add "ports", which are 16-bit identifiers.
  - A communication channel (connection) defined by 5 items:
    - [source addr, source port, dest addr, dest port, protocol]

- For example: The Unreliable Datagram Protocol (UDP)
  - Layered on top of basic IP (IP Protocol 17)
  - Datagram:
    - An unreliable, unordered, packet sent from source user to destination user (Call it UDP/IP)
  - Important aspect: low overhead!
  - Often used for high-bandwidth video streams.
  - Many uses of UDP considered "antisocial".

- Internet Architecture: Five Layers

<table>
<thead>
<tr>
<th>Layer</th>
<th>Implemented at Hosts</th>
<th>Implemented at Hosts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Physical</td>
<td>Lower three layers</td>
<td>Lower three layers</td>
</tr>
<tr>
<td>Datalink</td>
<td>Top two layers</td>
<td>Top two layers</td>
</tr>
<tr>
<td>Transport</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Network</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Host A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Router</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Host B</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Communication goes down to physical network
Then from network peer to peer
Then up to relevant layer

Transport
Network
Datalink
Physical

Application
Host A
Host B
Router

Layering Analogy: Packets in Envelopes

Internet Transport Protocols
- Datagram service (UDP): IP Protocol 17
  - No-frills extension of "best-effort" IP
- Reliable, in-order delivery (TCP): IP Protocol 6
  - Connection set up & tear down
  - Discarding corrupted packets (segments)
  - Retransmission of lost packets (segments)
  - Flow control
  - Congestion control
- Other examples:
  - DCCP (33), Datagram Congestion Control Protocol
  - RDP (26), Reliable Data Protocol
  - SCTP (132), Stream Control Transmission Protocol

Recall: Sockets in concept
- Create Server Socket
  - Bind it to an Address (host:port)
  - Listen for Connection
- Create Client Socket
  - Connect it to server (host:port)
- Accept
  - syscall
- write request
- Accept
  - syscall
- read request
- Close Client Socket
- Close Server Socket
- Close Connection Socket
Reliable Message Delivery: the Problem

- All physical networks can garble and/or drop packets
  - Physical media: packet not transmitted/received
    - If transmit close to maximum rate, get more throughput
    - Even if some packets get lost
    - If transmit at lowest voltage such that error correction just starts correcting errors, get best power/bit
  - Congestion: no place to put incoming packet
    - Point-to-point network: insufficient queue at switch/router
    - Broadcast link: two host try to use same link
    - In any network: insufficient buffer space at destination
    - Rate mismatch: what if sender send faster than receiver can process?

Reliable Message Delivery on top of Unreliable Packets

- Need some way to make sure that packets actually make it to receiver
  - Every packet received at least once
  - Every packet received at most once
- Can combine with ordering: every packet received by process at destination exactly once and in order

Transmission Control Protocol (TCP)

- TCP (IP Protocol 6) layered on top of IP
- Reliable byte stream between two processes on different machines over Internet (read, write, flush)

TCP Details

- Fragments byte stream into packets, hands packets to IP
  - IP may also fragment by itself
- Uses window-based acknowledgement protocol (to minimize state at sender and receiver)
  - "Window" reflects storage at receiver
    - Sender shouldn't overrun receiver's buffer space
    - Also, window should reflect speed/capacity of network
      - Sender shouldn't overload network
- Automatically retransmits lost packets
  - Adjusts rate of transmission to avoid congestion

Using Acknowledgements

- How to ensure transmission of packets?
  - Detect garbling at receiver via checksum, discard if bad
  - Receiver acknowledges (by sending "ACK") when packet received properly at destination
  - Timeout at sender: if no ACK, retransmit

Some questions:

- If the sender doesn't get an ACK, does that mean the receiver didn't get the original message?
  - No
- What if ACK gets dropped? Or if message gets delayed?
  - Sender doesn't get ACK, retransmits, Receiver gets message twice, ACK each

Transmission Control Protocol (TCP)

- Stream in:
  - Stream out:

  A

  B

  Packet

  B

  ACK

  Timeout

  A

  B

  Packet

  B

  ACK

  Packet

  Timeout

Using Acknowledgements

Transmission Control Protocol (TCP)
Stop-and-Wait (No Packet Loss)

- Loss recovery relies on timeouts.
- How to choose a good timeout:
  - Too short: lots of duplication
  - Too long: packet loss is really disruptive!

- How to deal with duplication:
  - Retransmission certainly opens up the possibility for packet loss.

- Equation for one-way latency:
  \[ \text{One-way latency} = \frac{\text{RTT}}{2} \]

- Throughput for 0% packet loss:
  \[ \text{Throughput} = \frac{\text{Bandwidth}}{1 + \text{Packet Loss}} \]

- 200 Mbps link and packet size of 1000 bytes:
  \[ \text{Throughput} = \frac{200 \text{ Mbps}}{2} = 100 \text{ Mbps} \]

- For symmetric latency:
  \[ \text{Throughput} = \frac{\text{Bandwidth}}{2} \]

- Calculate the round trip time (RTT) for a packet to travel from sender to receiver and back:

- Suppose RTT = 100 ms and Bandwidth is 1 packet per RTT:
  \[ \text{Throughput} = \frac{1 \text{ packet}}{100 \text{ ms}} = 10 \text{ Mbps} \]

- If the data transfer is 1500 bytes, then the throughput is:
  \[ \frac{1500 \text{ bytes}}{10 \text{ Mbps}} = 150 \text{ packets per RTT} \]

- Throughput is limited by the timeout (\( T \)), which is the round trip time (RTT) for the packet to travel from sender to receiver and back:

- How fast can you send data?

- Little's Law applied to the network:
  \[ \lambda = \frac{\text{Packet Arrival Rate}}{\text{RTT}} \]

- For Stop-and-Wait:
  \[ \lambda = 1 \text{ packet per RTT} \]

- So bandwidth is 1 packet per RTT:
  \[ \text{Bandwidth} = \lambda \cdot \text{RTT} \]

- Very inefficient if we have a 100 Mbps link:
  \[ \text{Throughput} = \frac{100 \text{ Mbps}}{2} = 50 \text{ Mbps} \]

- Suppose RTT = 100 ms and Bandwidth is 1 packet per RTT:
  \[ \text{Throughput} = \frac{1 \text{ packet}}{100 \text{ ms}} = 1 \text{ Mbps} \]

- So bandwidth is 1 packet per RTT:
  \[ \text{Bandwidth} = \lambda \cdot \text{RTT} \]

- How to deal with packet loss:
  - Loss recovery relies on timeouts.

- How to choose a good timeout:
  - Too short: lots of duplication
  - Too long: packet loss is really disruptive!

- How to deal with duplication:
  - Retransmission certainly opens up the possibility for packet loss.

- Little's Law applied to the network:
  \[ \lambda = \frac{\text{Packet Arrival Rate}}{\text{RTT}} \]

- For Stop-and-Wait, assume 1 packet:
  \[ \lambda = 1 \text{ packet per RTT} \]

- So bandwidth is 1 packet per RTT:
  \[ \text{Bandwidth} = \lambda \cdot \text{RTT} \]

- Very inefficient if we have a 100 Mbps link!
Solution: put sequence number in message to identify re-transmitted packets

- Receiver checks for duplicate number's; Discard if detected

Requirements:
- Sender keeps copy of unACK'd messages
  - Easy: only need to buffer messages
  - Hard: when ok to forget about received message?

Alternating-bit protocol:
- Send one message at a time; don't send next message until ACK received
- Sender keeps last message; receiver tracks sequence number of last message received

Pros: simple, small overhead

Cons: doesn't work if network can delay or duplicate messages arbitrarily

Recall: Communication Between Processes

- Data written by A is held in memory until B reads it
- Writing to the queue blocks if the queue is full
- Reading from the queue blocks if the queue is empty
- POSIX provides this abstraction in the form of pipes

Buffering in a TCP Connection

- A single TCP connection needs "in-memory" queues

Advantages of Moving Away From Stop-and-Wait
Host 1
Window Size: Space in Receive Queue

A host's window size for a TCP connection is how much remaining space it has in its receive queue.

A host advertises its window size in every TCP packet it sends!

Sender never sends more than receiver's advertised window size.

Sliding Window Protocol

TCP sender knows receiver's window size, and aims never to exceed it.

Sliding Window (No Packet Loss)

Example: Window size = 3 packets

Sliding Window Protocol: Per Byte!

TCP sender and receiver have three regions:
- Sent and ACK'd
- Sent and not ACK'd
- Not yet sent

Sender has three regions:
- Window in receiver's view
- Out-of-order packets
- Unacked packets

TCP Windows and Sequence Numbers:

TCP sender ensures that:
Number of sent but unacknowledged bytes < receiver's advertised window size.

Rule: TCP sender ensures that:
- All packets that have been sent may arrive before the window size.
- TCP sender knows receiver's window size and aims never to exceed it.

Window Size: Space in Receive Queue
**Congestion**

- Too much data trying to flow through some part of the network.
- TCP artificially restricts the window size to avoid congestion.
- The network can accommodate only a certain amount of data at a time.

**Congestion Avoidance**

- How long should the timeout be for retransmitting messages?
  - Too long: wastes time if the message is lost.
  - Too short: unnecessary timeout.

- The sender's window size gets chosen to match the rate of sending packets with the rate that the link can accommodate.
  - Closely related to the window size at the sender: too big means putting too much data into the network.
  - Goal: fill the network between sender and receiver.

**TCP Solution: Slow Start**

- If no timeout, slowly increase the window size (throughput) by 1 for each ACK received.
- If a timeout occurs, cut the window size in half.

**Additive Increase, Multiplicative Decrease**

- These two strategies help TCP avoid congestion by acknowledging the data that flows.
Recall: Connection Setup over TCP/IP

1. Open connection: 3-way handshake
   - Client (initiator)
     - SYN, SeqNum = x
   - Server
     - SYN and ACK, SeqNum = y and Ack = x + 1
     - Send response
   - Client
     - ACK, Ack = y + 1

2. Reliable byte stream transfer from (IPa, TCP_Port1) to (IPb, TCP_Port2)

3. Close (tear-down) connection
   - Indicate if connection is reset

Other elements:
- Source IP Address and Port Number
- Sequence Number (one for each sender)
- Acknowledgment Number
- 3-way handshake
- Reset packet
- Connect call()
- New connection to wait
- Sockets in concept
- Establishing TCP Service

- Well-known ports (0-1023)
- Lower 16 bits reserved
- Destination Port Number
- Source Port Number
- Destination IP Address
- Source IP Address
- Open, Close ports
- New connection
- Send/Receive
- 3-way handshake
- Accept socket
- Connection Server
- New connection
- Close connection
- Close sockets
- Close connection sockets
- Close server sockets
Question: Data Representation

Recall: Distributed Applications Build With Messages

Data Representation

- An object in memory has a machine-specific binary representation
  - Threads within a single process have the same view of what’s in memory

If data is not memory, a machine-specific binary representation

- Easy to compute offsets into fields below pointers, etc.

Outcome: Distributed applications build with messages

Question: How do you actually program a distributed application?

- Need to synchronize multiple threads, running on different machines
  - No shared memory, so cannot use test&set

- One abstraction: send/receive messages

- Interface:
  - Mailbox (mbox): temporary holding area for messages
    - Includes both destination location and queue
  - Send(message, mbox)
  - Send message to remote mailbox identified by mbox
  - Receive(buffer, mbox)
    - Wait until mbox has message, copy into buffer, and return

Network

Send

Receive

Recall: Distributed Applications Build With Messages

Data Representation

- An object in memory has a machine-specific binary representation
  - Threads within a single process have the same view of what’s in memory

- In the absence of shared memory, externalizing an object requires us to turn it into a
  - Sequence of sharable memory
  - Expressing an object as a sequence of sharable memory

- Easy to compute offsets into fields below pointers, etc.

- Any call to read() returns below pointers, etc.

- An object in memory has a machine-specific binary representation

Question: Close Sockets: 4-Way Teardown

Close Sockets: 4-Way Teardown

- Close Server Socket
  - Close Connection Socket
    - Send FIN
    - Close socket
    - Close connection

- Connect to Receiver (option)
  - Bind to port
  - Listen for connection
    - Accept
    - Create socket
    - Close server socket
Simple Data Types

uint32_t x;

• Suppose I want to write a
  x to a file

• First, open the file:
  
  ```
  FILE* f = fopen("foo.txt", "w");
  ```

• Then, I have two choices:
  1. `fprintf(f, "%lu", x);`
  2. `fwrite(&x, sizeof(uint32_t), 1, f);`

  » Or equivalently, `write(fd, &x, sizeof(uint32_t));`

• Neither one is "wrong" but sender and receiver should be consistent!

Machine Representation

• Consider using the machine representation:
  ```
  #include "stdio.h"
  
  int main(void) {
    int x;
    x = 1024;
    return x;
  }
  ```

• For a byte-address machine, which end of a machine-recognized object (e.g., int) does its byte-address refer to?

  • Big Endian: address is the most-significant byte
  • Little Endian: address is the least-significant byte

Endianness

• What Endian is the Internet?
  • Big Endian
  • Network byte order
  • Vs. "host byte order"
Dealing with Endianness

• Decide on an "on-wire" endianness
• Convert from native endianness to "on-wire" endianness before sending out data (serialization/marshalling)
  – `uint32_t htonl(uint32_t)` and `uint16_t htons(uint16_t)` convert from native endianness to network endianness (big endian)
• Convert from "on-wire" endianness to native endianness when receiving data (deserialization/unmarshalling)
  – `uint32_t ntohl(uint32_t)` and `uint16_t ntohs(uint16_t)` convert from network endianness to native endianness (big endian)

What About Richer Objects?

• Consider `word_count_t` of Homework 0 and 1...
  • Each element contains:
    – An `int`
    – A pointer to a string (of some length)
    – A pointer to the next element
• `fprintf_words` writes these as a sequence of lines (character strings with \\n) to a file
• What if you wanted to write the whole list as a binary object (and read it back as one)?
  • How do you represent the string?
  • What if you wanted to write the word list as a binary object (and read it back as one)?
  • How do you represent the string?
  • How do you represent the string?

Data Serialization Formats

• JSON and XML are commonly used in web applications
• Lots of ad-hoc formats

What About Richer Objects?

• Consider `word_count_t` of Homework 0 and 1...
  • Each element contains:
    – An `int`
    – A pointer to a string (of some length)
    – A pointer to the next element
• `fprintf_words` writes these as a sequence of lines (character strings with \\n) to a file
• What if you wanted to write the whole list as a binary object (and read it back as one)?
  • How do you represent the string?
  • What if you wanted to write the word list as a binary object (and read it back as one)?
  • How do you represent the string?
  • How do you represent the string?

Data Serialization Formats
Remote Procedure Call (RPC)

- Raw messaging is a bit too low-level for programming:
  - Must wrap up information into message at source
  - Must decide what to do with message at destination
  - May need to sit and wait for multiple messages to arrive
  - And must deal with machine representation by hand

- Another option: Remote Procedure Call (RPC)
  - Calls a procedure on a remote machine
  - Idea: Make communication look like an ordinary function call
  - Automate all of the complexity of translating between representations

Client (caller) $r = f(v1, v2)$;
Server (callee) $res_t = f(a1, a2)$

RPC Concept

RPC Implementation

- Request-response message passing (under covers!)
- “Stub” provides glue on client/server
- Marshalling involves (depending on system):
  - Converting values to a canonical form, serializing objects, copying arguments passed by value.
- Unmarshalling occurs (depending on system)
- Shown side is responsible for “marshalling” arguments and “unmarshalling” the return
- Client stub is responsible for “marshalling” arguments and “unmarshalling” the return
- Custom procedures on a remote machine

RPC Information Flow

- Need to resolve message passing (under covers)
RPC Details (1/3)

• Equivalence with regular procedure call
  – Parameters
  – Request Message
  – Result
  – Reply message
  – Name of Procedure: Passed in request message
  – Return Address: mbox2 (client return mail box)

• Stub generator: Compiler that generates stubs
  – Input: interface definitions in an “interface definition language (IDL)"
    » Contains, among other things, types of arguments/return
  – Output: stub code in the appropriate source language
    » Code for client to pack message, send it off, wait for result, unpack result and return to caller
    » Code for server to unpack message, call procedure, pack results, send them off

RPC Details (2/3)

• Cross-platform issues:
  – What if client/server machines are different architectures/languages?
    » Convert everything to/from some canonical form
    » Tag every item with an indication of how it is encoded (avoids unnecessary conversions)

• How does client know which mbox (destination queue) to send to?
  – Need to translate name of remote service into network endpoint (Remote machine, port, possibly other info)
  – Binding: the process of converting a user-visible name into a network endpoint
    » This is another word for “naming” at network level
    » Static: fixed at compile time
    » Dynamic: performed at runtime

RPC Details (3/3)

• Dynamic Binding
  – Most RPC systems use dynamic binding via name service
    » Name service provides dynamic translation of service

• What if there are multiple servers?
  – Could give flexibility at binding time
    » Choose unloaded server for each new client
    » Could provide same mbox (router level redirect)
    » Choose unloaded server for each new request
    » Only works if no state carried from one call to next

• What if multiple clients?
  – Pass pointer to defendant’s return mbox in request

Problems with RPC: Non.Atomic Failures

• Different failure modes in distributed system than on a single machine

• Different failure modes in det system than on a single machine

• Consider many different types of failures

• Dynamic Binding
Problems with RPC: Performance

- RPC is not performance transparent:
  - Cost of Procedure call « same-machine RPC « network RPC
  - Overheads: Marshalling, Stubs, Kernel-Crossing, Communication

Programmers must be aware that RPC is not free:
- Caching can help, but may make failure handling complex

How do address spaces communicate with one another?

- Shared Memory with Semaphores, monitors, etc...
- File System
- Pipes (1-way communication)
- "Remote" procedure call (2-way communication)

RPC's can be used to communicate between address spaces on different machines or the same machine:
- Example: split kernel into application-level servers.
- Fault isolation: bugs are more isolated (build a firewall)
- Enforces modularity: allows incremental upgrades of pieces of software (client or server)
- Location transparency: service can be local or remote

Microkernel operating systems

- Example: split kernel into application-level servers.
- File system looks remote, even though on same machine

Why split the OS into separate domains?
- Fault isolation: bugs are more isolated (build a firewall)
- Enforces modularity: allows incremental upgrades of pieces of software (client or server)
- Location transparent: service can be local or remote

Microkernel Structure

- File system looks remote, even though on same machine

Network-Attached Storage and the CAP Theorem

- Consistency:
  - Changes appear to everyone in the same serial order
- Availability:
  - Can get a result at any time
- Partition-Tolerance:
  - System continues to work even when network becomes partitioned

Consistency, Availability, Partition-Tolerance (CAP) Theorem:
- Cannot have all three at the same time
- Otherwise known as "Brewer's Theorem"
Summary

• TCP: Reliable byte stream between two processes on different machines over Internet
  – Uses window-based acknowledgement protocol
  – Congestion avoidance dynamically adapts sender window to account for congestion in network
• Remote Procedure Call (RPC): Call procedure on remote machine or in remote domain
  – Provides same interface as procedure
  – Automatic packing and unpacking of arguments without programmer intervention
  – Adapts automatically to different hardware and software architectures at remote end
• Distributed File System: Transparent access to files stored on a remote disk
  – Caching for performance
  – Transparent access to files stored on a remote disk

TCP: Reliable byte stream between two processes on different machines over Internet