RPC abstraction via stub functions on client and server
RPC stubs

- **Client stub:**
  - Constructs message with function name and parameters
  - Sends request message to server
  - Receives response from server
  - Returns response to client

- **Server stub:**
  - Receives request message
  - Invokes correct function with specified parameters
  - Constructs response message with return value
  - Sends response to client stub
Producer-consumer using RPC

- **Client stub**

  ```c
  int produce (int n) {
      int status;
      send (sock, &n, sizeof(n));
      recv (sock, &status, sizeof(status));
      return(status);
  }
  
  **Server stub**

  ```c
  void produce_stub () {
      int n;
      int status;
      recv (sock, &n, sizeof(n));
      status = produce(n);
      send (sock, &status, sizeof(status));
  }
  ```
Generation of stubs

- Stubs can be generated automatically
- What do we need to know to do this?

- Interface description:
  - Types of arguments and return value

- e.g. rpcgen on Linux
RPC Transparency

- RPC makes remote communication look like local procedure calls
  - Basis of CORBA, Thrift, SOAP, Java RMI, …

- What factors break illusion?
  - Failures – remote nodes/networks can fail
  - Performance – remote communication is inherently slower
  - Service discovery – client stub needs to bind to server stub on appropriate machine
RPC Arguments

- Can I have pointers as arguments?
- How to pass a pointer as argument?
  - Client stub transfers data at the pointer
  - Server stub stores received data and passes pointer
- Challenge:
  - Data representation should be same on either end
  - Example: I want to send a 4-byte integer:
    » 0xDE AD BE EF
    » Send byte 0, then byte 1, byte 2, byte 3
    » What is byte 0?
Endianness

- int x = 0xDE AD BE EF
- Little endian:
  - Byte 0 is 0xEF
- Big endian:
  - Byte 0 is 0xDE

- If a little endian machine sends to a big endian:
  - 0xDE AD BE EF will become 0xEF BE AD DE

April 3, 2019
EECS 482 – Lecture 21
No class on Monday!

Next Wednesday:

FORESHADOW
From operating systems to distributed systems

- Why build distributed systems?
- Performance
  - Aggregate performance of many computers can be faster than that of (even a fast) single computer
- Reliability
  - Try to provide continuous service, even if some computers fail
  - Try to preserve data, even if some storage fails
What is a distributed system?

“A distributed system is one in which the failure of a computer you didn’t even know existed can render your own computer unusable.”

Leslie Lamport
What is a distributed system?

- A collection of distinct processes that:
  - are spatially separated
  - communicate with each other by exchanging messages
  - have non-negligible communication delay
  - do not share fate
What does it mean for an event to “happen before” another event?
What is a distributed system?

- A collection of distinct processes that:
  - are spatially separated
  - communicate with each other by exchanging messages
  - have non-negligible communication delay
  - do not share fate
  - have separate physical clocks

(imperfect, unsynchronized)
**Single machine**

- A single clock
- Each event has a timestamp
- Compare timestamps to order events

**Distributed system**

- Each process has its own clock
- Each clock runs at a different speed
- Cannot directly compare clocks

An absolute temporal ordering is not what you want in a distributed system anyway

Leslie Lamport
an absolute temporal ordering is not what you want in a distributed system anyway

Leslie Lamport

Why not?

Because temporal ordering is not observable. You cannot read two separate clocks simultaneously!
Ordering events without physical clocks

Modeling a process:

- A set of instantaneous events with an a priori total ordering
- Events can be local, sends, or receives.
Ordering events without physical clocks

“Happened-before” relation, denoted: \( \rightarrow \)

Part 1

- If \( a \) and \( b \) are events on the same process and \( a \) comes before \( b \), then \( a \rightarrow b \)

\[
\begin{array}{c}
\bullet \quad a \\
\bullet \quad b
\end{array}
\]
"Happened-before" relation, denoted: $\rightarrow$

Part 2

- If $a$ is the sending of a message by one process and $b$ is the receipt of the same message by another process, then $a \rightarrow b$
Ordering events without physical clocks

“Happened-before” relation, denoted: →

Part 3

- If $a \rightarrow b$ and $b \rightarrow c$, then $a \rightarrow c$
Ordering events without physical clocks

Putting it all together

April 3, 2019

EECS 482 – Lecture 21
Ordering events without physical clocks

Can arrows go backwards? Yes
Can cycles be formed?

No, because the same event would happen at two different times
Ordering events without physical clocks

Are all events related by $\rightarrow$ ?

Diagram:

- Events: a, b, c, d, e, f, g, h, i, j
- Relations:
  - a → b
  - b → c
  - c → d
  - e → f
  - f → g
  - g → d
  - h → i
  - i → j
  - j → d
A partial order

The set of events $q$ such that $q \rightarrow p$ are the events that could have influenced $p$ in some way

\{a, b, e, f, h\}
A partial order

If two events could not have influenced each other, it doesn’t matter when they happened relatively to each other.

$h$ and $d$ are concurrent: $h \leftrightarrow d, d \leftrightarrow h$
Goal

- Generate a total order that is consistent with the happened-before partial order
  - E.g. $a \ b \ c \ d \ldots$
Lamport clocks

Define a function LC such that:

\[ p \rightarrow q \Rightarrow LC(p) < LC(q) \]

(the Clock condition)
Lamport clocks

Define a function LC such that:

\[ p \rightarrow q \Rightarrow LC(p) < LC(q) \]

(the Clock condition)

Implement LC by keeping a local \( LC_i \) at each process \( i \).
Lamport clocks

Single process

\[ p \]

\[ a \quad b \quad c \quad d \]

\[ 1 \quad 2 \quad 3 \quad 4 \]

\[ 6 \quad 37 \quad 1145 \]
Lamport clocks

Across processes

\[ b \rightarrow h \Rightarrow LC(b) < LC(h) \]
Lamport clocks

Across processes

\[ b \rightarrow h \Rightarrow LC(b) < LC(h) \]
\[ g \rightarrow h \Rightarrow LC(g) < LC(h) \]
Putting it all together
Is this correct?

No. $k$ should be 7.
Generating a total order

- Order messages by LC
- Ties are broken by unique process ID
Generating a total order

Total order: $a \ h \ b \ i \ c \ e \ f \ j \ d \ g$