Frequently asked questions from the previous class survey

- Publish/subscribe
  - Twitter
  - Does producer store the data?
  - Pub/sub vs P2P
  - Can a new subscriber receive an old event?
  - Are broker nodes connected to all publishers?
  - NDNs and publish/subscribe?
  - Event vs Topic based subscriptions?

Topics covered in this lecture

- RPC
  - Persistence/transience
  - Synchronous/asynchronous communications
  - Parameters in RPC settings

RPCs and Distributed Objects

- Why are we looking at this?
- What we will cover in the next few lectures:
  - Middleware based interactions
  - Remote Procedure Calls
  - How to design your own distributed objects framework
  - Java RMI

Middleware based interactions
Viewing the middleware as an additional service in client-server computing: E-mail

- Sender passes mail to mail-server
- Hope that server will eventually deliver to client
- Receiver connects to mail server to see if mail is available
  - If so, pull it

The middleware as an intermediary for application-level communications

- Sender passes mail to mail-server
- Hope that server will eventually deliver to client
- Receiver connects to mail server to see if mail is available
  - If so, pull it

Persistent communications

- Message handed-off to a middleware is held as long as it takes to deliver to receiver
- Sending application need not execute after message submission
- Receiver application need not execute when message is being submitted
- Time-decoupled interactions

Transient communications

- Message stored only as long as the sender and receiver are executing
  - At the same time
- Transport level communications are transient
  - E.g., if a router cannot deliver to a host, simply drop the message

Asynchronous communication

- Do not block
- Proceed to the next thing, once a message has been submitted

Synchronous communications

- Sender can block
- Until request is known to be accepted
Synchronization points

Client
Synchronize at request submission
Synchronize at request delivery
Synchronize after processing by server
Server
Storage/Transmission facility

Synchronicity and persistence options can be combined
- Persistence + synchronization?
  - Message queuing systems
- Transience + synchronization?
  - Remote procedure calls

Discrete and streaming communications
- Discrete
  - Each message is a complete unit of information
- Streaming
  - Multiple messages; one after another
  - Related by sending order or temporal relationships

Conventional procedure calls
- Consider a call:
  \[
  \text{count} = \text{read}(\text{fd}, \text{buf}, \text{nbytes});
  \]
- \( \text{fd} \) indicates a file
- \( \text{buf} \) is an array of characters
- \( \text{nbytes} \) tells us how many bytes to read

Remote Procedure Calls
Caller pushes parameters onto stack (last one first) in order
- Main program's local variables
  - nbytes
  - buf
  - return address
- read's local variables

BEFORE

AFTER

Main program's local variables

Main program's local variables

SLIDES CREATED BY: SHRIDEEP PALICKARA
Parameter passing in C:

- Call-by-value
- Call-by-reference

Parameter passing in C: Call-by-value

- Values parameters simply copied on to stack
  - Scalar types
  - In our example, fd and nbytes
  - In the called procedure?
    - Value parameters = initialized local variables
  - Changes made do not affect original value at the caller

Parameter passing in C: Call-by-reference

- Reference parameter
  - Pointer (address) to a variable
  - Not the value
  - When used to store something
    - Results in modifications at the caller
- Arrays are passed by reference
  - Address of our character array buf pushed on stack

Another parameter passing scheme: Call-by-copy/restore

- Caller will copy variable on to the stack
  - Similar to call-by-value
  - Copy back after call
    - Overwrite original value at the caller
  - In most settings has the same effect as call-by-reference

Distributed systems often have explicit message exchanges

- Rely on send and receive primitives
- Do not conceal communications
  - Needed to achieve access transparency

Origins of RPC

  - First use was at Xerox in 1981 (Crypt)
- 1984 paper by Birell and Nelson
  - Introduced a different way of handling communications
  - Allow programs to call procedures located on remote machines
Remote procedure call (RPC)

- Process on A calls procedure on machine B
- Calling process on A is suspended
- Execution of called procedure takes place on B
- Results are transported
- Message passing not visible to the programmer

Goal of RPC is to make a remote procedure call look like a local one

- Objective is transparency
- Calling procedure should not be aware that called procedure is remote
  - The other way around too

Reading data from a file on a single processor system

- Use the API's read in the code
- The read routine is extracted from library by a linker
  - Inserted into program
- The routine is implemented by an equivalent read system call
- The read call is an interface between user-code and the local OS

RPC transparency is similar

- If read is a remote procedure?
  - A different version, called a stub, is put into library
- When it is invoked:
  - Does not ask the OS for data, but …
  - Creates a request message to be sent to server
- Client stub does a send
  - Then calls receive (blocking itself) until reply comes

The server's OS passes message to a server stub

- Transforms network requests into local invocations
- Stub usually calls receive
  - Is blocked awaiting incoming messages
- Perform remote work to get results
  - Return result to the client

RPC Mechanism
What makes RPCs tick?

- Blissful ignorance at the clients
- Remote services accessed by making ordinary procedure calls
- No need for send and receive primitives

Parameter passing

- Stubs pack and unpack request parameters
- Packing parameters into a message
  - Parameter marshaling

Challenges in passing parameters in distributed systems

- Multiple machine types are present
- Each has its own representation for
  - Numbers, characters, etc.
- Examples
  - IBM mainframes use EBCDIC character codes
  - PCs use ASCII
  - Integer representations
  - One’s or Two’s complement

The endian issue

- Big endian: PARC
  - Number their bytes from left to right
- Little endian: INTEL
  - Number their bytes from right to left

An example of two parameters: 32 bits
(a) Integer - 5
(b) 4 character string JILL

<table>
<thead>
<tr>
<th>0</th>
<th>3</th>
<th>2</th>
<th>0</th>
<th>1</th>
<th>5</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>L</td>
<td>I</td>
<td>J</td>
<td>L</td>
<td>I</td>
<td></td>
</tr>
</tbody>
</table>

On INTEL: Little endian

How are messages sent?

- Messages are transferred byte-for-byte
- First byte sent is the first to arrive
- Big-endian is typically the convention in data networking (including IPv6)
  - Network byte order
Message sent from INTEL to SPARC:

<table>
<thead>
<tr>
<th>INTEL: Little endian</th>
<th>SPARC: Big endian</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 2 0 1 5 0</td>
<td>5 0 1 0 2 0 3</td>
</tr>
<tr>
<td>L7 L6 I5 J4</td>
<td>J4 I5 L6 L7</td>
</tr>
</tbody>
</table>

The integer is now $5 \times 2^4$
The String is correct "JILL"

On INTEL:
Little endian

On SPARC:
Big endian

October 24, 2017

Passing reference parameters:
Our read example:
- Copy array into message and send to server
- Server stub calls server with a pointer to the array
- Changes made by server using the pointer?
  - Affects the array
  - Send array back to the client
- Call-by-reference replaced by copy/restore

October 24, 2017

Passing reference parameters:
How about we invert the bytes of each word:

<table>
<thead>
<tr>
<th>INTEL: Little endian</th>
<th>SPARC: Big endian</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 3 0 2 1 5 0</td>
<td>5 0 1 0 2 0 3</td>
</tr>
<tr>
<td>L7 L6 I5 J4</td>
<td>J4 I5 L6 L7</td>
</tr>
</tbody>
</table>

integer is 5
String is "LILI"

October 24, 2017

Parameters in RPC settings:

- Pointers are meaningful only within a process
- Address 1000 in process A may be start of an array
- Address 1000 in process B is something else
- Forbid pointers and reference parameters?
  - These are highly important
  - Not having them is not an option

October 24, 2017

Optimizing the copy/restore:

- If the buffer is an input or output parameter?
  - One of the copies can be eliminated
- If it is input to server?
  - No need to copy back to client
- If it is output from the server?
  - No need to send from client in the first place

October 24, 2017
Parameter specification and stub generation

- Agree on the format of the messages
- Similar actions for passing complex data structures

Parameter agreements: An example

```plaintext
foobar(char x; float y; int z[5])
```

- Word = 4 bytes
- Character in rightmost byte of a word
- Leave the leftmost 3 bytes empty
- Float as a whole word
- Array as group of words
  - Precede this by word indicating the array length

Parameter agreements: An example

```
foobar(char x; float y; int z[5])
```

|--------------------------|---|---|------|------|------|------|------|

Besides message format, there needs to be agreement on representation

- Integers, characters, Boolean, etc
- For e.g.:  
  - Integers could be 2's complements  
  - Characters are 16-bit Unicode  
  - Floats in IEEE 754 standard  
  - Store everything in big-endian

The contents of this slide-set are based on the following references