CSC301

Distributed Applications

Midterm Information

- Next week - **March 19 @ 18:10 OR March 20 @ 12:10**
  - Duration: 90 minutes
  - **Attend the section where you are officially registered!**
- Questions
  - Git/GitHub
    - Given a scenario, write a series of commands to accomplish a certain task
  - Java Programming
    - Single domain
    - Given: interfaces, helper functions, stubs, test cases
    - To-Do: complete implementations of classes/methods using design patterns studied in the course
  - **No Aids**

Guest Lectures

- Planned for **Monday/Tuesday March 26/27**
- Speakers from the industry (aiming for 2)
- Open discussion format
  - Their experience working in development and managing development projects
  - Tools, processes, techniques they use
  - How development is happening in their organizations - e.g., a feature journey
  - Hiring practices
- Other topics?

Today’s goal

The goal of this lecture is to connect (some of) the dots between

- What you see in your programming courses, and
- The technology you use in your daily life.

Working With Multiple Processes

- Up until now we have been discussing objects that are in the same Java Virtual Machine (same process, same machine).
- However, almost any significant modern application spans several processes and/or machines.
- How do we communicate between processes?

Network communication

Remember CSC209? How did you communicate between processes?

- Pipes & local files - between processes on the same machine.
- Sockets - Allow you to communicate between machines.

Network communication

- Sockets
  - Essentially a “pipe between machines”
  - Two processes open a socket between them, and read/write data (aka send/receive messages).
- Protocols
  - Sometimes (usually) the data needs context.
  - The format of the data on the wire.

What does foo.f() mean?

- When we see
  - **`foo.f();`**
- We read
  - “*calling*” a function `f` on object `foo`
- In some OO languages, we say:
  - “*Sending the message*” `f` to object `foo`

Sometimes, it’s easier to imagine “sending a message” between machines than “calling a function” across machines.

Distributed Objects

- How should we think about an app that sends data between machines?
  - How about a network of “distributed” objects that “send messages” to each other?
- How should we think of the messages?
  - The receiver (`self` or `this`, depending on language)
  - The name of the method
  - Arguments to the method

Remote Procedure Call (RPC)

- A remote procedure call is a method call where the callee is on one machine and the caller is on another.
  - Developers don't explicitly code the details of these interactions - the code is generally very similar to a local procedure call.
- How do the arguments travel from callee to caller?

Remote Procedure Call (RPC)

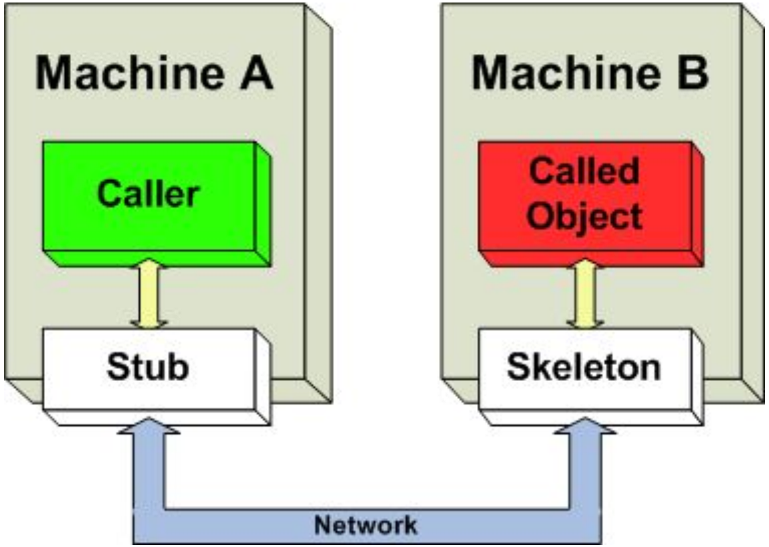
- A remote procedure call is a method call where the callee is on one machine and the caller is on another.
- How do the arguments travel from callee to caller?
  - Serialization/deserialization to convert arguments (objects) to/from data (bytes)
  - Send/receive the resulting data through a socket!

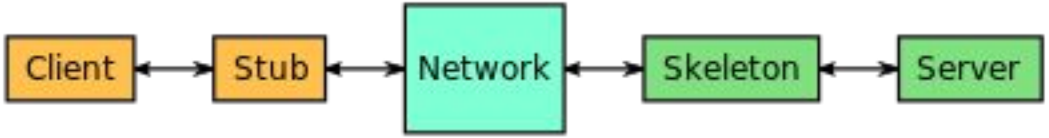
Examples

- Sending bytes over a socket
- Sending a Java-serialized object over the socket
- Java RMI - Calling a method on a remote object, as if it was local.
  - See this hello world example, where the RMI library
    - Allows you to bind/lookup objects by name
    - Provides you with proxy objects that abstract away all communication details
  - RMI depends on agreement/convention between the participants (e.g., client and server)
  - *Note:* RMI is a bit old and is not so popular these days.
    - But the same concepts/challenges apply to current libraries/technologies as well.

Java RMI

- Remote Method Invocation -
  OOP version of RPC
  - Simplifies interaction between remote Java Applications
  - Caller delegates request to its stub (proxy object), a local object representing a remote object
  - Stub passes caller arguments over the network to the server skeleton
  - Skeleton passes data to the called object, waits for response and returns the results
  - Stub/skeleton responsible for marshalling/unmarshalling messages





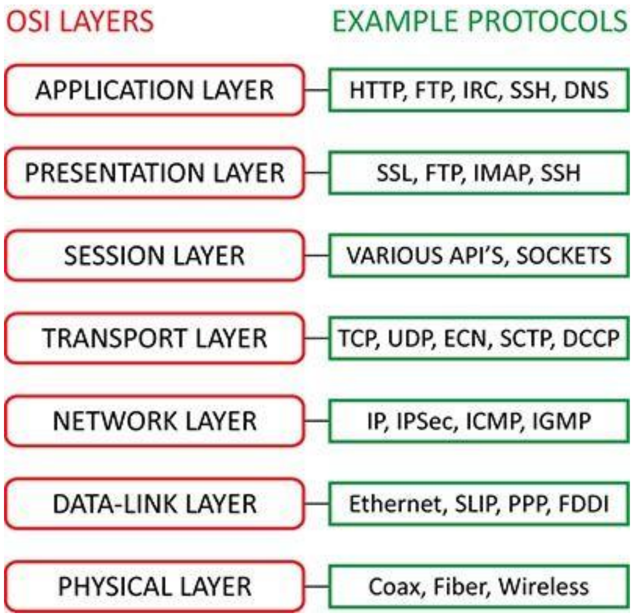
[Image credit: Wikipedia]

Distributed Applications

- Processes on different machines need to communicate
- At the lowest level, use sockets
  - Send/receive bytes
  - Control every bit that goes on the wire
- Q: How to get from pushing bytes through a socket to the Internet as we know it?

Protocols

- *Hierarchy* of protocols on top of sockets
  - Agreed-upon standards/conventions
  - Enable communication
  - Each protocol *abstracts a level of details*
- Participants (i.e., processes) know
  - When they should send data
  - What data to send
  - When they should wait for incoming data
  - What incoming data is expected to look like



Basic Communication Protocol

- IP - Network Layer
  - The principal communication protocol of the Internet
  - All about routing - “Getting data from A to B”
- TCP - Transport Layer
  - Transport-layer protocol that complements IP
  - Reliable data transfer using error detection
    - Alternative - UDP (User Datagram Protocol), unreliable replacement for TCP

Basic Communication Protocol

- TCP/IP is a *transport-layer* protocol
  - Low-level protocol(s), on top of sockets
  - Not much semantics, all about transferring data reliably across the wire.
- If we want to build the Internet, we need …
  - To think a bit more high-level
  - To abstract away details of moving bytes on a wire
- We need an *application-layer* protocol

Modern Web Applications

- Java-only protocols are no longer an option
- For example, think of Facebook’s system…
    - iOS app (objective C/Swift)
    - Android App (Java)
    - Web client (JavaScript)
    - Web server (PHP)
    - And many (many!) more internal pieces that make up their extremely complex and large system.

Modern Web Applications

- Need a communication protocol that is
  - Application-layer
    - Allow us to focus on building applications, not arranging bits on the wire.
  - Language-agnostic
    - Exchange data between applications written in different programming languages
  - Adopted by the industry as a standard!

HTTP

- Application layer protocol
  - On top of reliable transport layer (e.g., TCP/IP).
- Client-server protocol
  - Client sends a request
  - Server sends back a response
- One of the most commonly-used protocols on the web

HTTP Request

- Request line
  - Indicates which resource the client is requesting
  - E.g.: `GET /index.html HTTP/1.1`
- Request headers
  - Contain metadata
  - E.g.: `Accept: application/json`
- An empty line
- Optional message body (i.e., data)

HTTP Response

- Response line
  - Includes a status code
  - E.g.: `HTTP/1.1 200 OK`
- Response headers
  - Contain metadata
  - E.g.: `Content-Length: 348`
- An empty line
- Optional message body (i.e., data)

Live Demo

1. Open your browser at http://example.com

2. `telnet www.example.com 80`, and type

```
GET /index.html HTTP/1.1
Host: www.example.com

```

(With an empty line at the end)

Example Response

Request

```
telnet www.example.com 80
Trying 93.184.216.34...
Connected to www.example.com.
Escape character is '^]'.
GET /index.html HTTP/1.1
Host: www.example.com
```

Metadata

```
HTTP/1.1 200 OK
Cache-Control: max-age=604800
Content-Type: text/html
Date: Mon, 12 Mar 2018 14:57:41 GMT
Etag: "1541025663+ident"
Expires: Mon, 19 Mar 2018 14:57:41 GMT
Last-Modified: Fri, 09 Aug 2013 23:54:35 GMT
Server: ECS (lga/1383)
Vary: Accept-Encoding
X-Cache: HIT
Content-Length: 1270
```

Response Body

```
<!doctype html>
<html>
<head>
        <title>Example Domain</title>

        <meta charset="utf-8" />
        <meta http-equiv="Content-type" content="text/html; charset=utf-8" />
        <meta name="viewport" content="width=device-width, initial-scale=1" />
        <style type="text/css">
        body {
        background-color: #f0f0f2;
        margin: 0;
        padding: 0;
        font-family: "Open Sans", "Helvetica Neue", Helvetica, Arial, sans-serif;

        }
        div {
....
```

Live Demo - Cont’d

- In both cases, you sent HTTP request to a server, and got a response.
- Let’s look at the telnet window
  - Save the response body (without the headers) to a local file, test.html.
  - And open the file in your web browser.

Another Demo

- Every modern browser has *developer tools*
  - Allow you to inspect what the browser is doing under the hood
  - Different browsers = different menu/shortcut
- Let's try something ...
  - Open the developer tools in your browser
  - Go to http://google.com

Another Demo - Cont’d

- We can see all HTTP requests:

- We only asked for one website, how come there are 25 requests?

Another Demo - Cont’d

- In your browser, keep the *developer tools* open, and start typing a search query in Google.
  - The browser sends HTTP request(s),
  - Gets a response containing JSON data,
  - And updates the screen (i.e., display auto-complete options)
- Notice … All network communication happened in the background

Sync vs. Async

- Synchronous flow
  - Call a function, wait for the result
- Asynchronous flow
  - Call a function
  - Go do something else (without waiting for the result)
  - When the function returns (at some point), do something useful with its result.

Very Quick “History Lesson”

- Mid-90’s
  - Browsers were working synchronously
  - User-experience was inferior (in today’s standards)
- Late-90’s
  - Browser vendors gradually added async support
- Early 2000’s
  - Developers realized the potential
  - Emerging standards, conventions, tools and hypes.

AJAX

- Asynchronous JavaScript And XML
  - Decouples data interchange and presentation layers, allowing for dynamic content change (no need to reload the entire page)
- Huge hype around 2004
  - GMail and Yahoo Mail were some of the early notable web apps to offer desktop-like UX
- In time, JSON replaced XML as the standard serialization format
  - Less data to transfer
  - Easier to parse in web browsers (that have an extremely optimized JavaScript engine)

Async Function Call - Review

- When calling `f`, provide a callback
  - Like Lambda expressions or function pointers
  - Possibly multiple callbacks (e.g. success & error)
- Run `f` in a background thread
  - Other threads don’t have to wait for `f`
  - Extremely useful if `f` makes a network request
- When `f` returns a result, call the callback function (with the result as an argument)

Software As A Service

- Instead of providing users with software that runs on their machine, provide them with a service that runs on yours.
  - Easier to maintain
  - For many businesses, solves piracy issues
- Client talks to your software over the web
- More standards and conventions ⇒ More powerful tools & libraries

SaaS

- This is how modern web/mobile apps work
- Use remote service(s) to
  - Show weather info
  - Manage comments on articles
  - Display ads
  - etc.

SaaS

- Each service has its own API
    - Similar to interfaces in Java
- The API specifies
    - Format
      E.g.: JSON over HTTP
    - Available resources
      E.g.: `GET /weather?city=Toronto,CA HTTP/1.1`
    - Expected request/response data
      E.g.: `{“text” : “hot”, “degrees” : “33c”}`

Summary

- Sockets
  - Let us transfer bytes *reliably* between machines
- Communication protocols
  - Abstract low-level communication details, and used for communicating between applications
- Serialization
  - Convert in-memory objects to/from data that can be sent to/from different machines
- Threads
  - Allow us to build asynchronous flow
  - Async flow enables us to use rich web-services while providing quality UX
  - Lambdas (i.e., functions) and promises make async code more elegant and maintainable
- Conventions & Standards
  - E.g.: By following REST, one can auto-generate server and client code from documentation.