CS 4414/5416: Systems Programming

Network and Web Programming

David Bindel

2026-11-05

Logistics

Upcoming

- P5 (Tetris) posted! **Groups of 3-4** preferred
  - Check-in is Nov 11
  - Final is Nov 18
- Lab 11 is Friday: multithreaded web service
- Next week: Start asynchronous programming

Where We Are

This module: Concurrent programming with threads

- Message-based concurrent programming
- Shared memory concurrent programming
- Synchronization issues
- Concurrent network programming (today)

Reminder: Today will be a little shorter (4 PM)

Poll Everywhere!

An example of deadlock prevention (vs avoidance or recovery) is:

(a) Acquiring locks according to a total ordering

(b) Running the banker's algorithm of Dijkstra

(c) Aborting a thread on a timeout during wait for a lock

Application Protocol

Network Models

- Application layers (e.g. HTTP)
  - Full OSI has application, presentation, session
- Transport layer (e.g. TCP, UDP)
- Media layers: network, link, physical

How do we think about application layer questions?

Example Protocol: HTTP

- Stateless protocol, simple request/response pair
  - Client establishes TCP connection
  - Server responds on same (then usually close)
- HTTP/1 and /2 are TCP/IP; /3 is QUIC over UDP
- We'll stick to HTTP/1.1

See also Rust Book, Ch 21

Request

Example from Wikipedia:

```
GET / HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9
Accept-Language: en-GB,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
```

Message Structure

- Header (ASCII) with CR/LF terminators (`\r\n`)
  - Start line (different for request/response)
  - Header fields (`key: value`)
  - Empty line ends header
- Then body data (need not be ASCII text)

Message Structure

- Request start line specifies
  - Method/verb (`GET`, `POST`, `PUT`, `DELETE`, etc)
  - Path argument
  - Protocol version
- Response start line specifies
  - Protocol version
  - Three-digit status code
  - Reason phrase

Leveling Up

- Usually wrap HTTP in libraries
  - Client side: `reqwest`
  - Server side: `axum` and `actix-web`
- Can build higher-level APIs on top

RPC

Remote Procedure Call interfaces:

- “Look like” a function call at both ends
  - This is a leaky abstraction
  - Need to be careful about error handling!
- Middleware serializes and sends arguments/returns
  - Ex: JSON RPC

RESTful APIs

REST is Representational State Transfer design style

- Uniform interface via unique URIs
- Stateless
- Client-server separation
- Cacheability
- Layered

CRUD

Four HTTP methods (“verbs”):

- Create via POST
- Read via GET
- Update via PUT (or PATCH)
- Delete via DELETE

Getting Notifications

Issue: “Don’t call us, we’ll call you”

- HTTP doesn’t support anything like a callback!
- But have some mechanisms as needed
  - Polling, short and long
  - Web sockets
  - Server-sent events

Polling

"Do you have something to tell me?"

- Short polling: "Are we there yet?" ad nauseum
  - Relatively expensive
- Long polling: Open an event request and wait
  - `Connection: Keep-Alive` and `Keep-Alive: ...` help

Web Sockets

- Full duplex communication for the web!
- Use URL (vs IP and port) to identify server
- Handshake is over HTTP
- Subsequent message-based communication over TCP

Server-Sent Events

- Around since 2011, uptake took a while
- Two components to standard
  - `EventSource` interface in the browser
  - Event stream data format
- Send events over initial TCP stream
  - Keep session open until close or idle timeout

Live-Coding: Two Dots

Wrap-Up

Up Next

- P5 (Tetris) posted! **Groups of 3-4** preferred
  - Check-in is Nov 11
  - Final is Nov 18
- Lab 11 is Friday: multithreaded web service
- Next week: Start asynchronous programming