CS 35L Week 4 Worksheet

JavaScript & Client-Server Applications Worksheet

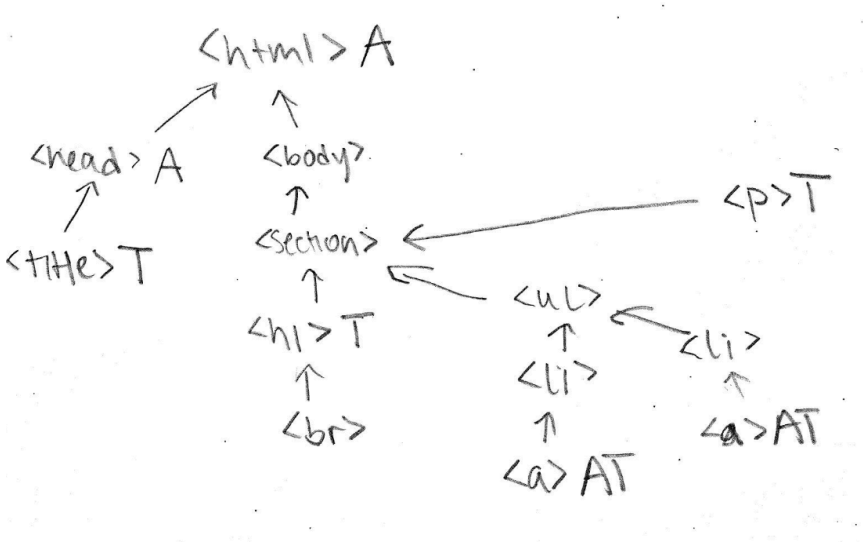
HTML & JavaScript

HTML & JavaScript are the ubiquitous languages used on the web. JavaScript has a robust series of APIs and is natively supported in all browsers. HTML is used by all webpages to define the markup for the page. We will be exploring the features of HTML & JavaScript through these next set of questions.

Spring 2022 Midterm. Question 5.
(10 minutes) Consider the following HTML taken from the class web page. Diagram the elements of the document tree that it represents. Draw arrows from each element to its parent element, if any. Label each element with its tag, along with the letter A if the element has attributes and the letter T if the element contains text that is not part of any sub-element.

```
<html lang='en'>
  <head>
    <title>UCLA Computer Science 35L, spring 2022</title>
  <head>
  <body>
    <section>
      <h1>UCLA Computer Science 35L, spring 2022. <br>Software
Construction</h1>
      <ul>
        <li><a href='news.html'>News</a></li>
        <li><a href='syllabus .html'>Syllabus</a>/li>
      </ul>
      <p>Lecture, 4 hours; laboratory, 2 hours; outside study, 6 hours.</p>
    </section>
  </body>
</html>
```

Solution:



Spring 2022 Midterm. Question 10.
(6 minutes) Suppose React were written in C++ rather than in JavaScript. Explain how this would affect how you'd do Assignment 3 in React. In particular, how would you debug your program, compared to debugging your program with React as it is?

Solution: C++ is a compiled language, while JavaScript is interpreted. With debugging, for C++ you will have to compile. Since JavaScript is interpreted, errors will only show up when they are encountered while testing. But since C++ is compiled, the compiler will give compile-time warnings/errors

- Overall conclusion
    - Compiled languages are more “safe” than interpreted languages, with regards to reliability, as the compiler can catch buggy code by examining the code itself, while the interpreted languages can run smoothly as long as the input you give it doesn’t encounter the buggy code (meaning you don’t know where your bugs are unless you hit one!)

Spring 2020 Final.
The Electron software framework <https://www.electronjs.org> combines Node.js with the Chromium rendering engine to let developers build standalone apps, as opposed to the client-server model used in class. Several desktop apps are built using Electron, including Slack, Skype, and Visual Studio Code.

In Electron, a single program runs both the browser and Node.js code. Instead of splitting the app between client and server, the Electron architecture splits your app between a single

main process that runs the overall app, and a separate renderer process per web page; see <https://www.electronjs.org/docs/latest/tutorial/process-model>.

Node.js is based on an event loop in a single thread, so how can it possibly work in an Electron environment which is based on multiple processes? Briefly explain. Hint: see <https://www.electronjs.org/docs/latest/tutorial/multithreading>.

Solution:

- Electron can spawn multiple processes
- Instead of network connectivity you can maybe use interprocess communication (IPC) instead.
- Event driven programming has a main “while loop” where you can see if an event is triggered by the client (maybe a button is clicked or something similar)
- You can spawn a process to handle each event async from the main loop to handle it asynchronously

React Coding.

1. Write a simple React component with the following features (you can use https://playcode.io/react for an online editor that doesn't need a login):

- There is a counter that is initialized to 0.
- There is text that displays "Counter is: x", where x is the value of the counter.
- There is a button that says "Increment" which will increment the counter.
- There is a button that says "Reset" which will reset the counter to 0.

Solution:

```
import { useState } from "react";

export function App() {
  const [counter, setCounter] = useState(0);
  return (
    <>
      <p>Counter is: {counter}</p>
      <button onClick={() => setCounter(counter + 1)}>
        Increment
      </button>
      <button onClick={() => setCounter(0)}>
        Reset
      </button>
```

```
    </>
  );
}
```

2. Write a React component to play a simplified version of blackjack:

- There is a button that says "Hit" that draws a randomly generated card from 1 to 13
- There is a button that says "Reset" that clears the drawn cards
- There is text that displays "Total is: x" where x is the sum of the drawn cards
- If the total is greater than or equal to 21, the hit button is disabled
- Each drawn card has a text element displaying "Card: x", where x is the value of the card

Solution:

```jsx
import { useState } from "react";

export function App() {
  const [cards, setCards] = useState([]);
  // easy way to sum a list in JS
  // for more info: https://stackoverflow.com/q/1230233
  const total = cards.reduce((a, b) => a + b, 0);
  const onHit = () => {
    // generate a random integer from 1-13
    // for more info: https://stackoverflow.com/q/1527803
    const val = Math.floor(Math.random() * 12) + 1;
    setCards([...cards, val]);
  };
  const onReset = () => setCards([]);
  return (
    <>
      <p>Total is: {total}</p>
      <button onClick={onHit} disabled={total >= 21}>
        Hit
       </button>
      <button onClick={onReset}>
        Reset
      </button>
      {cards.map((val, index) => (
        // when rendering lists like this, elements need keys
        // for more info: https://react.dev/learn/rendering-lists
        <p key={index}>Card: {val}</p>
      ))}
    </>
```

```
  );
}
```

Spring 2022 Final.

13a (6 minutes). Suppose the following source code is in the file yourproject/src/app/lib/utilities/idSanitizer/index.js. What does it do and how would you use it?

```
const pc = /[\]\]\.,\/?¿!$'"%^&*;:{}=-_`~(),«»@#¥""'|]/g;

const sp = /\s+/g;


const idSanitizer = text =>

 text.replace(pc, '').replace(sp, '-');


export default idSanitizer;
```

13b (6 minutes). A typo in the code causes it to mishandle upper-case ASCII letters, '>', and '\'. These characters are mistakenly deleted. Identify and fix the bug.

Solution:

- A.
  - This code generates a function, idSanitizer that takes in input text, removes special characters specified by pc, and replaces spaces with hyphens.
- B.
  - The typo here is in pc. The hyphen between = and _ means that we match characters in the range ASCII 61-95 (this includes all capital letters, >, and \). This means that in the idSanitizer function, we would delete capital letters. To fix the bug, we would either move the hyphen to be the last character between the brackets in the regex, or escape it with a backslash.
  - `pc = /[\]\]\.,\/?¿!$'"%^&*;:{}=_`~(),«»@#¥""'|-]/g;`
  - `pc = /[\]\]\.,\/?¿!$'"%^&*;:{}=\-_`~(),«»@#¥""'|]/g;`

(8 mins.) Consider the following pseudocode for an event-based shell program written in JavaScript:

```
class Event { handle() { ... } }
class KeyboardEvent extends Event { handle() { ... } }
class OutputEvent extends Event { handle() { ... } }

function wait_for_event() {
  while (true) {
    if (keyboard_input_waiting) {
      // A keyboard button was pressed by the user
      keyboard_input_waiting = false;
      return KeyboardEvent();
    } else if (output_waiting) {
      // We ran a shell command and have output that needs
      // to be displayed back to the user
      output_waiting = false;

      return OutputEvent();
    }
  }
}

// Main
while (true) {
  const e = wait_for_event();
  e.handle();
}
```

(1 min) What code corresponds to the event loop?

```
while (true) {
  const e = wait_for_event();
  e.handle();
}
```

a. (1 min.) What code corresponds to the event handler(s)?

e.handle();

b. (6 min.) Discuss the performance of this particular implementation. Is it a good

implementation of event-based programming? Why or why not?

This is not a good implementation. One of the key benefits of event-based programming is that the CPU is able to do other work while the process is waiting for events to happen. However, in this implementation, that does not happen. In wait_for_event, we simply have a while loop that spins until an event is ready. The process does not use any special low-level function to release control of the CPU. While the OS will probably interrupt the process itself, it could get much more work done if our program explicitly slept while waiting.

Past CS 35L Exam (Fall 22)

6 (6 minutes). Suppose you were supposed to use Python as much as possible when doing assignment 3, thus minimizing the use of JavaScript. Explain how this would affect how you'd develop your solution to Assignment 3. In particular, how would you debug your program, compared to debugging your program with Node.js as it is?

Solution:

Pdb for Python. For JS, you can debug the code in the browser. Both Python and JS are interpreted languages, so debugging would be similar. However, while most browsers have an in-built V8 JS engine in them, they don't have anything for Python, meaning the Python code would have to be transpiled to JS to show on the browser.

Client-Server Applications

Probably the most widely implemented class of programs are networked applications connected to the **internet**. A popular structure of these applications follow the **client-server model** which is a large focus of CS 35L. In order to understand how **client-server applications** work, we must understand **computer networking**.

Activity - A Request's Journey

To get a better understanding of how networking works within client-server applications, we will be exploring the path of a network request being made to www.google.com. We will be exploring common Linux utilities to get a better understanding of what this one request does. You can follow along by running these commands on the SEASNet servers.

1. **HTTP**. A common protocol that is used to communicate between web servers is the **Hypertext Transfer Protocol (HTTP)**. This is known as an **Application Layer Protocol** and is the most popular protocol used in client-server applications. We will be exploring the journey of an HTTP request at different layers of the TCP/IP stack.
    a. What does the command, **curl https://www.google.com,** do?
    Make an HTTP request to www.google.com.
    b. Many internet protocols have implicit/default port numbers that they use on a host machine. What is the default port number used by **HTTP?** You can run the following commands to get a more **verbose** output from curl.
        i. HTTP: **curl -v http://www.google.com**
        HTTP defaults to port 80.
    c. An extension to HTTP is HTTPS which uses a protocol known as **TLS** for security. We will explore this in a later part but HTTPS uses a different default port than HTTP. What is the default port number used by **HTTPS**?
        i. HTTPS: **curl -v https://www.google.com**
        HTTPS defaults to port 443.
2. **DNS**. When your browser first tries to visit a website, it first needs to know the address of where that website is. These addresses are known as **IP Addresses**. These are

usually not memorable/human-friendly which is why assigning names to them is often convenient. The protocol that helps figure out how to assign names to addresses is called **Domain Name Service (DNS)**. This is another **Application Layer Protocol**.

   a. What does the command, dig www.google.com, do? Try running it.
      It makes a DNS request to see what records resolve for www.google.com.
   b. www.google.com should have an **A Record** corresponding which maps a domain name to an IP address. What is the IP address for www.google.com?
      Depends on which resolver you are located near but it is the 4 byte IPv4 address on the right hand side of the A Record.
   c. DNS requests are made to a special host called a **DNS Server** which contains **DNS Records**. What is the IP address of the server that your DNS request was made to? **Hint**: located near the part of the dig output labeled **Server**.
      Depends on where you make your request from.

3. **TCP**. After knowing where the host address of the server the HTTP request is being made to, your computer needs to establish direct connection between the two hosts. This is commonly done using a protocol known **Transport Control Protocol (TCP)** which is a **Transport Layer Protocol**.
   a. What does the command, nc www.google.com 80, do?
      Establishes a TCP connection to www.google.com at port 80.
   b. Application Layer protocols like HTTP build on top of Transport Layer Protocols like TCP. By setting up just a TCP connection, you can make an HTTP request! What does the following command do?
      echo -e "GET / HTTP/1.1\r\n\r\n" | nc www.google.com 80
      Manually sends an HTTP request over TCP!
   c. What other Application Layer Protocols use TCP? **Hint**: How do you connect to the SEASNet servers?
      SSH, SMTP, etc
4. **IP**. The **Internet Protocol (IP)** is a **Network Layer Protocol** used to uniquely identify hosts using **IP addresses**. Historically, IP addresses were represented in a format known as **IPv4** which consisted of 4 bytes (e.g. 127.0.0.1), however, with the growing popularity of the internet there is a growing demand to use **IPv6** (e.g. 2001:db8:3333:4444:5555:6666:7777:8888).
   a. What does the command, tracepath www.google.com, do?
      Shows the hops between routers each packet takes to get to www.google.com.
   b. Network packets don't go directly between hosts and are routed using **packet switching** between hosts called **routers**. Which path do your packets visit before reaching www.google.com? If traceroute is being finicky use, mtr

www.google.com instead.
Depends on your location but each line in the mtr/traceroute output is a router.

c. The final destination of your packet in the traceroute/mtr may not be www.google.com and may be to a **Content Delivery Network (CDN)** instead? Does the destination of your packet match what is expected in your **DNS Query**? How can you use dig to verify this?
Your route most likely will resolve to a CDN but its DNS records should resolve to the same IP address as www.google.com.

5. **TLS**. Many client-server applications employ security to prevent attacks from adversaries such as **man-in-the-middle** attacks and **replay attacks** to prevent adversaries from violating the confidentiality and integrity of data across untrusted channels like the internet. The most widely used security protocol on the internet is the **Transport Layer Security (TLS)** protocol.
   a. What does the command, openssl s_client -connect www.google.com:443, do?
   Establishes a TLS connection to www.google.com.
   b. An important part of TLS is verifying that a host claims to be who they are. This is done through **Certificate Authorities (CAs)**. What is the **Organization** for the root of the certificate chain for www.google.com:443**? Hint**: Check the "Certificate chain" part of the output and look for the largest number in the chain for which Organization to put as the answer.
   Most likely Google Trust Services LLC.
   c. TLS applies encryption on the Transport Layer Protocol (typically TCP). Once an encrypted Transport Layer is established, you can then send Application Layer data! What does the following command do?
   echo -e "GET / HTTP/1.1\r\n\r\n" | openssl s_client -connect www.google.com:443
   Manually makes an HTTPS request!

Past Exam Question (Quarter Unknown)

11. Suppose your client-server app is based on Node and React, your Node-based server is in Tokyo, your React-based client in Los Angeles caches the price of Takara Bio stock, and every ten seconds your client refreshes its cache by sending a single UDP request packet to the server and getting a single UDP response packet back containing the stock price. Suppose also (somewhat unrealistically) that the elapsed time between sending a request and receiving a corresponding response is always 128 ms, that this delay is entirely due to network latency, and that packets head east to Japan just as quickly as they head west to California.

11a (5 minutes). If your client received a response exactly one second ago and immediately updated its React-based browser cache, what's the range for the staleness of that cache entry? That is, what's the best case (least stale) for the cache entry, and what's the worst case (most stale) for the cache entry? Briefly explain.

Solution:

- Best case 0 seconds; this happens if no change between the server and the browser cache
- Worst case 1.064s; if there is a change, then our cache is 1s + 128ms/2 old. (128/2 is half of round trip time, and the one second is the time since we received the information.

11b (5 minutes). More generally, what's the worst case (most stale) for the cache entry, during the entire time your application is operating? Briefly explain.

Solution:

- Worst case is infinity/no cache entry (due to packet loss), since UDP allows packet loss; thus the cache can be arbitrarily out of date.

5 (8 minutes). HTTP/3 is often touted as a major performance win for web applications. Explain why your solution to Assignment 3 would not materially benefit from switching to HTTP/3.

Solution:

- HTTP/3 uses UDP. Assignment 3 doesn't have a large datastream and wouldn't benefit from the lightweightness of UDP. TCP would be better, because we would just want reliable transfer of data, since the data is so small (only 6 pieces!).
- Also, assignment 3 is very simple; the assignment as it is doesn't have a client-server structure; all the logic can just be implemented on the browser without needing data to be streamed back and forth. (i.e. if it wasn't multiplayer)

CS 35L Spring 2024 Midterm

Q4
10 Points

Suppose your browser sees an element , which is supposed to cause the browser to fetch and execute bigprog.js immediately before continuing to parse the current web page.

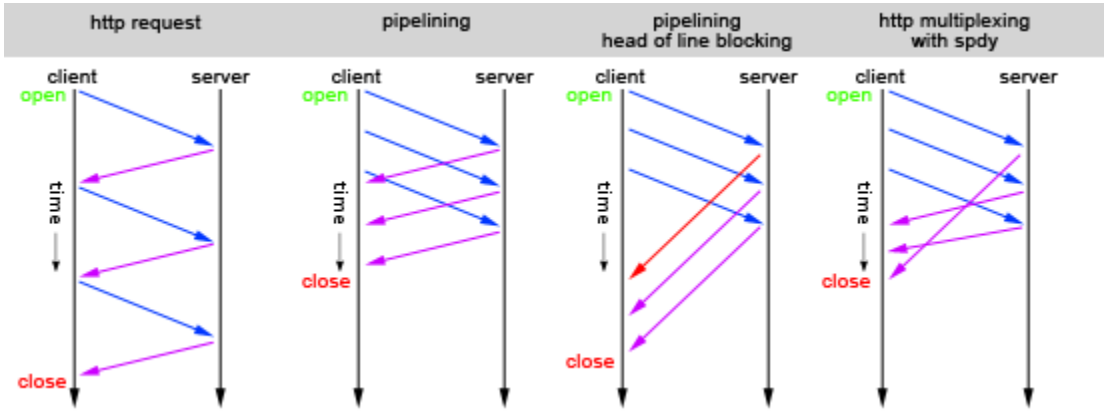
What are the advantages using HTTP/3 instead of HTTP/2 to fetch bigprog.js? How about using HTTP/2 instead of HTTP/1? Briefly describe the pros and cons of using the three major HTTP variants.

If a packet is lost under HTTP3 only the specified stream is blocked (due to QUIC over UDP) rather than the entire connection under previous HTTPs (TCP-based). For large programs with a lot of packets, HTTP3 can therefore be a major performance improvement. HTTP2 is also a performance improvement over HTTP1 because it introduces multiplexing etc which would also be helpful with fetching bigprog.js.

Q9

5 Points

Explain why in ordinary web applications, HTTP multiplexing is more likely to increase the total amount of network traffic than HTTP pipelining is.



Multiplexing breaks the head of the line requirement so that packets can be delivered as they arrive (whereas under pipelining they need to be delivered in order, even though we can have multiple requests out). Therefore HTTP multiplexing will increase the total amount of network traffic since we no longer spend time polling for correctly ordered packets.