Async Programming

Juan Pablo Zendejas

ACM@SDSU

September 25th, 2024

whoami

- 3rd year student
- Secretary
- Math and CS Major
- Coding since 13

Coding experience

- JavaScript was my first love
- Over-engineered solutions to problems
- Love functional programming
- Open source contributor

Solutions to my own problems on Linux

What is an async?

- Perform operations non-synchronously
- Out of order, simultaneously, ...
- Stop *waiting* and do more stuff

The proceedings

- JavaScript (web dev)
- Rust (low-level, performant)
- Deno (deno.land)
- Async by default

Use of async

- Modern web APIs use async

JavaScript

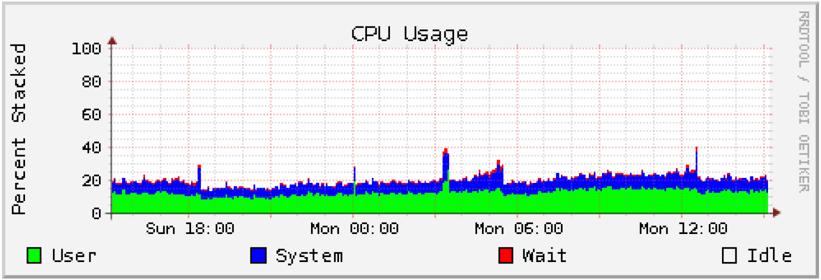
```javascript
let response = await fetch("https://sdsu.edu");
console.log(await response.text());
```

- Why design APIs this way?

Use of async

Many operations on computers are *not* bound by CPU speed.

- Lot of time is spent waiting
- Organize! Compute more!
- Switch contexts



Concept of event loop

- Out-of-order with only a single thread
- No parallelism (in most cases...)

Loop

1. Wait for messages
2. Dispatch message to method
3. Method now has control
4. Return control to loop

in JavaScript

- Interaction creates a message
- Method is executed
- During the method run, nothing else can happen
- Cause of lag in older sites

Taking advantage of the event loop makes web interaction smooth.

Now, a magic spell

- Stop thinking of functions as different from data
- Functions ARE data
- In JS, they are even objects with properties

Example

```
console.log.name
console.log.toString()
```

Functions of functions

- Pass around functions
- Return functions
- Compose functions

JavaScript

```javascript
function createAdder(n) {
    return (x) => x+n;
}

const addtwo = createAdder(2);
const result = addtwo(6);
console.log(result); // 8
```

Promises

JavaScript uses an abstraction called a `Promise` for asynchronous programming.

- Represents an operation to be completed
- We call this fulfilled
- Or, can be rejected

Promises

JavaScript

```
const myPromise = new Promise((resolve, reject) => {




});
```

Promises

JavaScript

```
const myPromise = new Promise((resolve, reject) => {
    setTimeout(() => {
        resolve("I'm done!");
    }, 300);
});
```

Promises

JavaScript

```
const myPromise = new Promise((resolve, reject) => {
    setTimeout(() => {
        resolve("I'm done!");
    }, 300);
});

myPromise
    .then(console.log)
    .then(handlePromise)
```

Promises

JavaScript

```javascript
const myPromise = new Promise((resolve, reject) => {
    setTimeout(() => {
        resolve("I'm done!");
    }, 300);
});

myPromise
    .then(console.log)
    .then(handlePromise)
    .catch(console.error);
```

Promises

JavaScript

```
const myPromise = new Promise((resolve, reject) => {
    setTimeout(() => {
        resolve("I'm done!");
    }, 300);
});

myPromise
    .then(console.log)
    .then(handlePromise)
    .catch(console.error);
```

- We use chaining

The await keyword

- .then chaining is messy
- Abstract it away using *syntax sugar*

JavaScript (cont.)

```
async function handlePromise() {
    try {
        console.log(await myPromise);
    } catch (e) {
        console.error(e);
    }
}
```

When to use await

- We don't always want to wait
- Other functions can manipulate Promises

JavaScript

```
const p1 = fetch("https://sdsu.edu");
const p2 = fetch("https://acm.sdsu.edu");

const both = Promise.all([p1, p2]);
const [res1, res2] = await both;
```

API design

- Deno and Bun are async-first
- No need to enter an async context
- `Deno.open`, `Deno.connect`

Make I/O operations non-blocking so more requests can be handled.

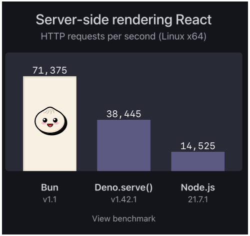


Figure: Bun and Deno perform better

Asynchronous APIs

- New Web APIs are almost all asynchronous
- Open files, download webpages
- Ancient history: XMLHttpRequest
- Horrid to use, but set the foundation

Futures

- The concept of a Promise is very powerful
- Rust borrowed the idea, called it “Futures”
- rust-lang.github.io/async-book

Futures

Rust

```rust
async fn get_two_sites_async() {
    // Create two different "futures" which,
    // when run to completion,
    // will asynchronously download the webpages.
    let future_one = download_async("https://foo.com");
    let future_two = download_async("https://bar.com");

    // Run both futures to completion at the same time.
    join!(future_one, future_two);
}
```

(from the Rust book)

Tasks

- In C#, we call them Tasks
- Probably work the closest to Promises

C#

```
List<Task> myTasks;
while (myTasks.Count > 0) {
    Task finishedTask = await Task.WhenAny(myTasks);
    var value = await finishedTask;
    Console.WriteLine("Task finished: " + value);
    myTasks.Remove(finishedTask);
}
```

(adapted from MSDN)

Why is async important

- Create faster programs
- Servers can handle multiple requests
- Break free from sequential programming

Further exploration

- Write a web server in JS (Deno, Bun)
- Use threading in Rust to perform complex calculations
- Find asynchronous principles in your favorite language

Thank you!