Concurrency

This stuff is hideously complicated...

Concurrency is hard

Concurrent

Parallel

Asynchronous

Reactive

Multi-Process

Multi-Threaded

Concurrency

Composition of independently executing processes. - Rob Pike

Dealing with multiple processes at once.

Think of Concurrency as a Design and/or Architecture

Parallelism

Simultaneous execution of multiple processes.

Performing multiple processes at once.

Concurrency != Parallelism

Parallelism implies Concurrency

Concurrency does not imply Parallelism

Simple example: A single core CPU can run a concurrent algorithm, but it cannot run it in parallel.

Mechanisms of Achieving Concurrency

Asynchronous Programming

Reactive Programming

Multi Threaded Programming

Multi Process Programming

Only Parallelized if the CPU, OS, Compiler, etc support it, expose it, and enable it.

How a computer works...

How a Computer works

The CPU allocates special areas in memory to interact with external devices (network, io, hdd, etc).

The CPU starts execution at memory location 0

Each clock tick, the “program counter” moves forward by 1, and executes the next instruction (first 0, then 1, then 2, etc.)

“jump” instructions set the program counter to an arbitrary value

How a Computer Works

Special instructions manipulate the “IRQ jump table”, which by default is empty.

Each clock tick, prior to executing the next instruction, the CPU checks to see if the interrupt lines are tripped. If so, it triggers the matching IRQ jump instruction.

The IRQ jump instruction moves the program counter to an arbitrary value and “remembers” the previous program counter value

The expectation is that at the end of the instruction list for the IRQ jump, the last instruction is a jump back to the previous program counter value, thereby resuming normal execution.

Take aways

One CPU core can only execute one thing at a time

* modern CPU cores have “vector” instructions that in effect do multiple things at once, but conceptually it is useful to think of them as executing one thing at a time.

CPU sends instructions to external adapters and they execute asynchronously and flag the CPU when they are done. The CPU addresses the interruption in a synchronous fashion (i.e. it has to drop what it is doing to address the interruption)

At no point, is the CPU doing more than one thing at a time.

How an OS works

Abstracts IRQs behind a common API

Multiplexes multiple programs together so that they can run in a time shared fashion.

Each program is called a Process

By default, each Process has a single Thread of execution.

An OS will *schedule* the execution of Threads across the Execution Resources available to it.

Provides a special *sleep* API call allowing one Thread to yield execution to another

Multi Process Programming

What if we broke up an application into multiple smaller applications?

Multi Threaded Programming

What if each process had multiple threads?

Asynchronous Programming

```
window.setTimeout(function() { console.log(‘yay, async!’); }, 1000);
```

```
fetch('flowers.jpg')
.then(function(response) {
  return response.blob();
})
.then(function(myBlob) {
  var objectURL = URL.createObjectURL(myBlob);
  myImage.src = objectURL;
});
```

Reactive Programming

Think of Spreadsheets

A=B+C

When B or C changes, the value of A automatically changes.

How is Async Programming implemented?

Typically through a mechanism called an Event Loop

Example of event loop...

How is Reactive Programming Implemented?

Reactive Instructions are converted to a DAG

Whenever leaf nodes of the DAG are manipulated, the impacted portions of the DAG are recomputed

Since changes are typically invoked via “reactions” to user input/external events, the entry points for these changes is typically the Event Loop