CS140E: embedded OS

Dawson Engler, prof
Holly Chiang, TA
Stanford winter 2020

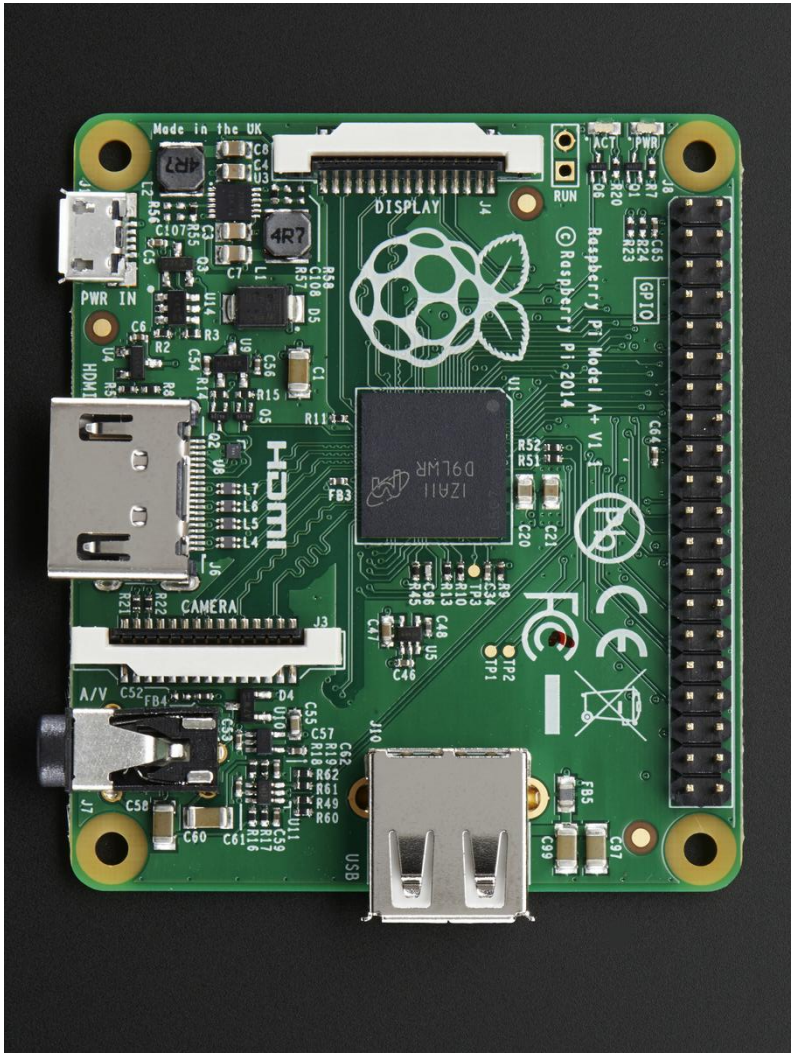
Outline

- What
- Why

What

Write small, clean OS on an r/pi A+:

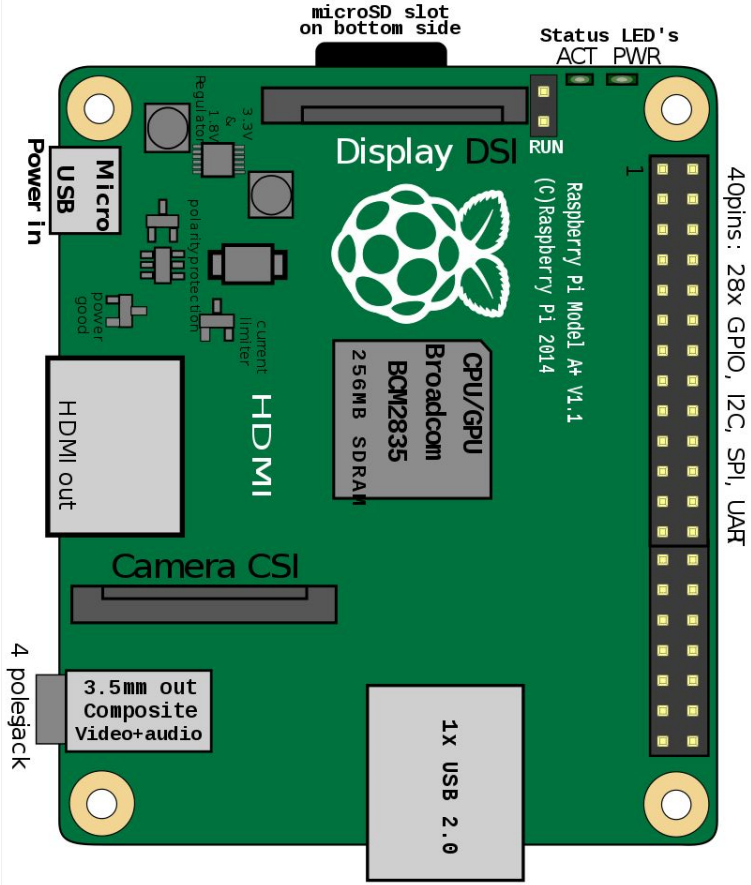
https://en.wikipedia.org/wiki/Raspberry_Pi



What

Write small, clean OS on an r/pi A+:

https://en.wikipedia.org/wiki/Raspberry_Pi



What you will build (tentative)

Bootloader (ship code from your laptop to the pi)

Device drivers

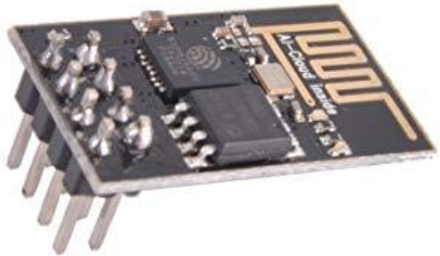
Threads + interrupts

Virtual Memory

Fuse file system to export pi to laptop

Simple file system on SD card.

Networking code that uses the cheap esp8266 chip



End result: simple, clean OS where you wrote (almost) every line.

Why OS?

If you can write a real OS, you can write almost anything (non-math-y).

- Once you get this, easy to delta to something else.

Classes are fake: real world is not a clean, textbook of systematized knowledge

- Difficult to understand documents
- Wrong
- Incomplete
- Not written to be used
- You will learn to operate in such a world without a lot of panic/drama.

Why R/PI A+?

Most OSes write code on a fake simulator

- Alot of work, not that cool at the end.

R/pi = real computer for about $20 and an ounce of weight.

- Many examples / blog posts of how to do various things.
- Unlike most machines, makes interacting with the real world easy.
- Can build many interesting systems b/c can use weird hardware easily (motion sensors, IR sensors, accelerometer, gyroscope, light sensor…)

Class philosophy

Write a complete, narrow OS all the way down to the bare metal.

- We cover less material than most OS classes (multi-level schedulers, multi-level page tables)
- However, you will understand much more thoroughly
- Hope: easy to do delta off of your knowledge to more fancy things.

Labs vs lectures:

- Always try to have you be writing code. You will actually understand what is going on.
- Common tragedy of OS: missing a key sentence, mistake in key document. We will use lab to fill this in, saving you many hours/days.
- Goal: you do pre-work to pre for lab, walk in, by the end of the lab, you have a complete working simple version of a key trick.

Goal: you will develop two super-powers

Power 1: Differential debugging.

> Efficiently answering “why doesn’t work” for complex things.
>
> Swap working pieces + Binary search

Power 2: Epsilon development.

> Foundational paradox: When building systems, the smaller the step you take, the faster you can run.

Differential debugging

You write code, it doesn’t work, the error could be:

The code you wrote

Hardware fault (bad manufacturing, smoked something)

Wiring mistake

Subtle cache issue

Compiler problem (more on this)

…

You will get good at breaking down problems by swapping pieces between a working system (yesterday’s code, your partners lab) and a non-working system (today’s code, your lab)

Example from next lab.

You get the following set of stuff:

To run you:

- Copy blink.bin to sd
- Wire up led
- Wire up serial device
- Plug into your laptop
- It doesn't work.
- But your partner's does

Example from next lab.

You get the following set of stuff:

To run you:

- Copy blink.bin to sd
- Wire up led
- Wire up serial device
- Plug into your laptop
- It doesn't work.
- But your partner's does

Example from next lab.

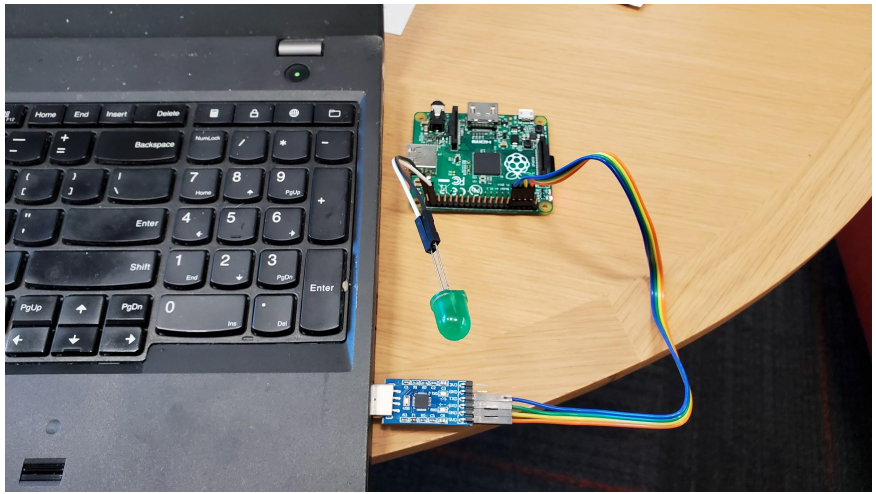
You get the following set of stuff:

To run you:

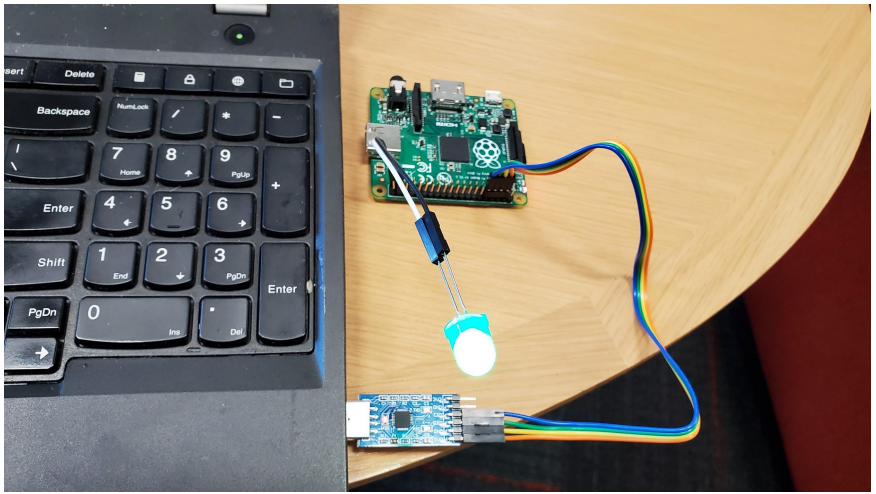
- Copy blink.bin to sd
- Wire up led
- Wire up serial device
- Plug into your laptop
- It doesn't work.
- But your partner's does

What is messed up?

Yours: Not working



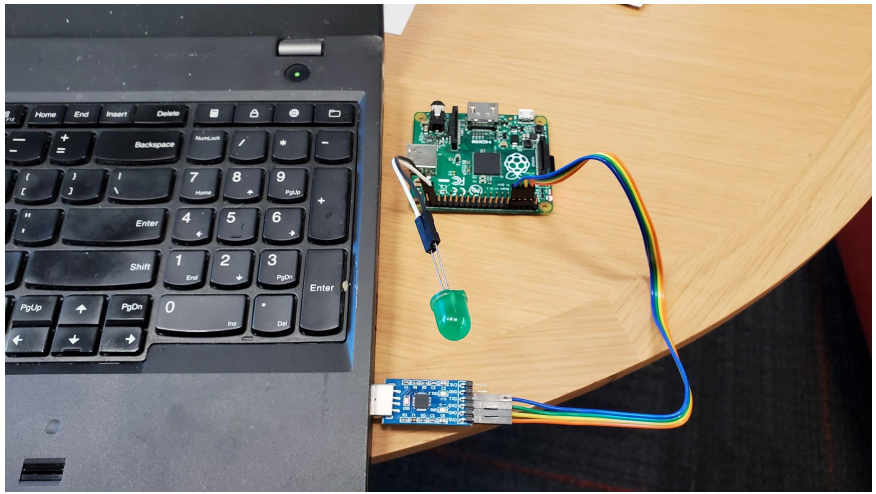
Partner's: Working



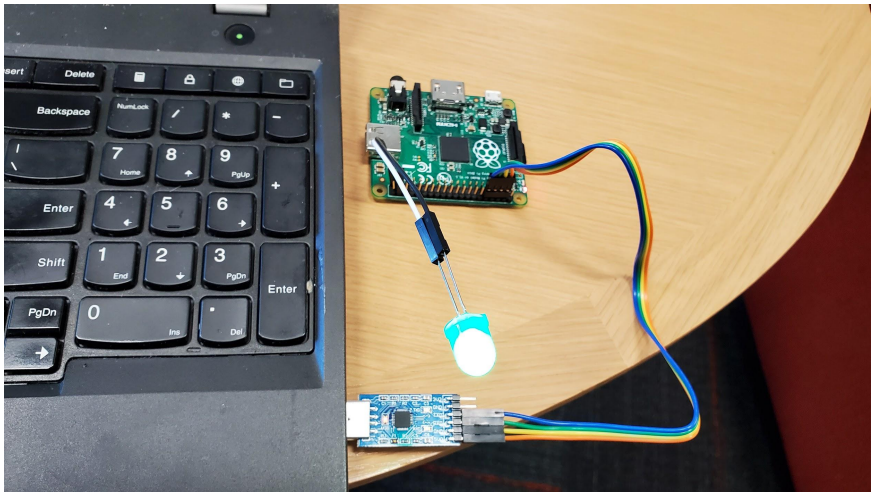
What to do first?

What is messed up?

Yours: Not working



Partner's: Working

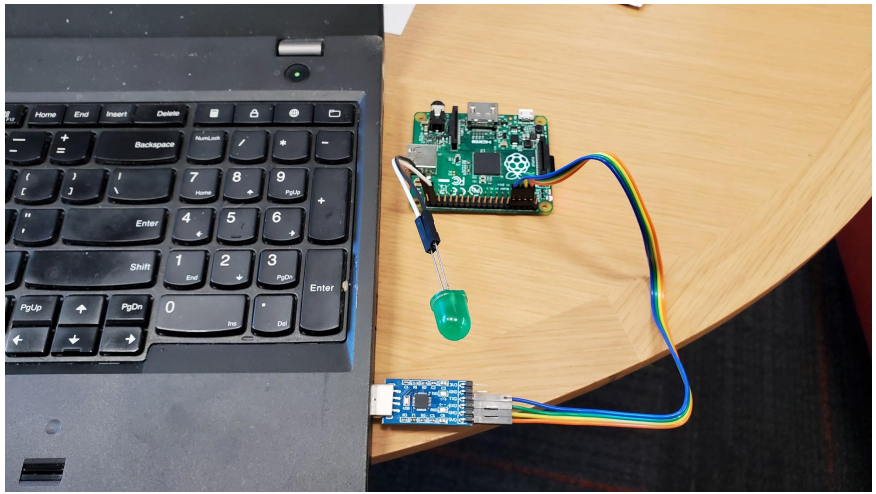


What does swapping tell you if doesn't work?

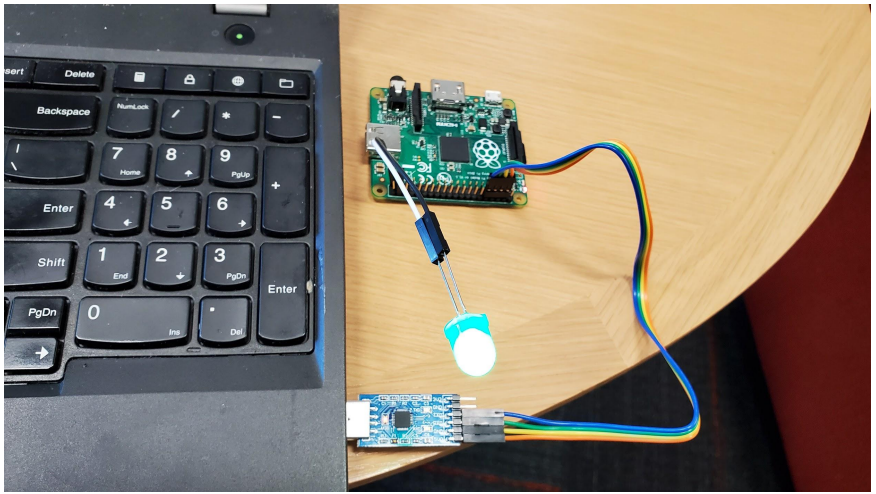
What does swapping tell you if works?

What is messed up?

Yours: Not working



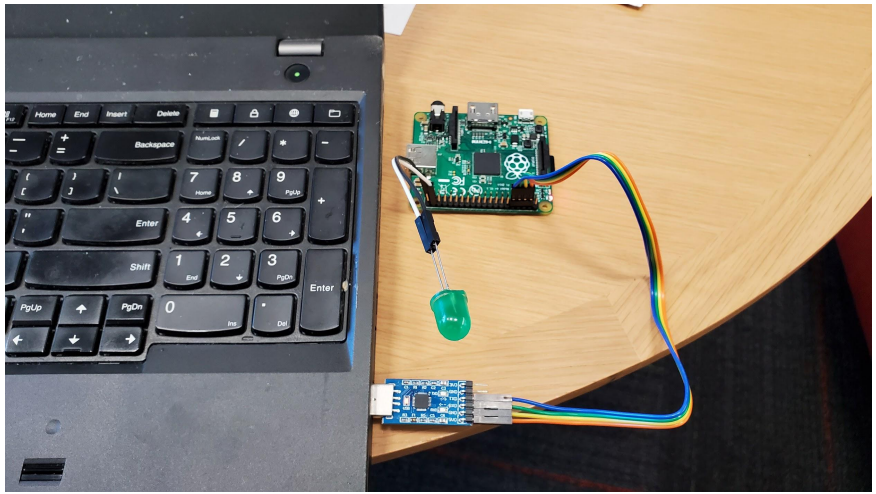
Partner's: Working



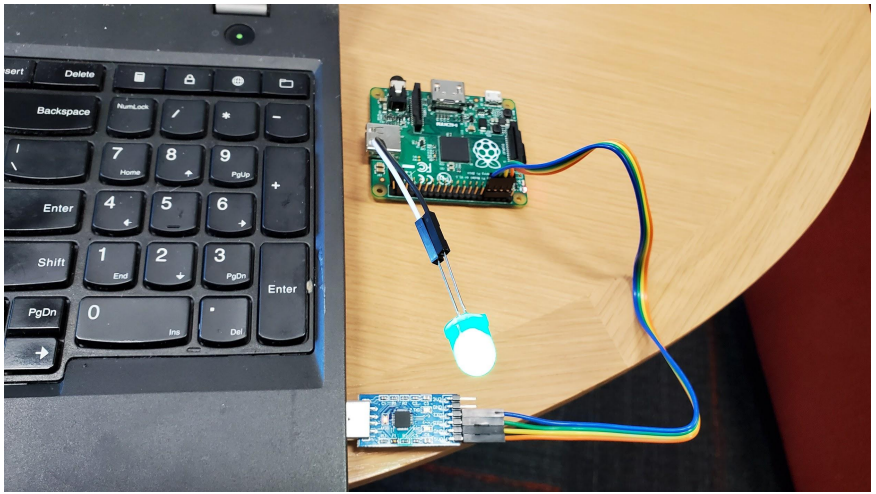
Swapping works: how to narrow down with least work?

What is messed up?

Yours: Not working



Partner's: Working



Entire class: whenever we control device, has some software component S (can be wrong) and some hardware component H (can be broken).

Doesn't work = linear equation solving with two variables. How to isolate?

Slow is fast.

What is wrong?

If I did X, it's X.

If I did X1+X2+...+Xn it could be any, or some combination.

Inverting crash / bug to root cause is much harder in the latter case.

The epsilon-sprint theorem:

Given a working system W□ and a change *C*, then as |C| $\Longmapsto \epsilon$, the time + computation it takes to figure out why { W□ + *C* } doesn't work goes to 0.

Related claim: the time it takes to debug why a change broke the system increases non-linearly with the size of the change.

Administrivia

We may or may not have a final project. Depends on how the class goes.

I am currently rewriting about 50% of the code / labs.

Grade breakdown if no final project:

Labs = 60%, HW = 30%, participation = 10%.

If final project:

Labs = 50%, Hw = 20%, project = 20%, participation = 10%.

Administrivia

Two labs each week.

- Each lab will have pre-lab work you should turn in before lab.
- Ideally finish during the lab period (I will stay til everyone is done).
- Must finish within a week of the lab, or start losing a letter grade each day.
- Must pre-arrange missed labs. It's a problem to miss more than a couple.
- ½ the time we will provide food.

There (tentatively) will be three "capstone" homework assignments that consolidate a chunk of labs together.

- If you've done the lab, this shouldn't be a big deal.

Administrivia

You can work with other people!

However, you *must* type and turn in everything yourself.

Please post to the newsgroup.

What to do now

Go to / clone the class git repository:

https://github.com/dddrrreee/cs140e-20win

Go to the newsgroup, or let me know if you don't have an invitation.

https://groups.google.com/forum/#!forum/cs140e-win20

For lab wednesday, make sure you:

have a way to write either a micro-SD or SD card.

Have a way to plug in a standard USB device

Read labs 0-blink and 1-gpio and do the pre-lab homework.