CS140E: embedded OS

Stanford win 2025

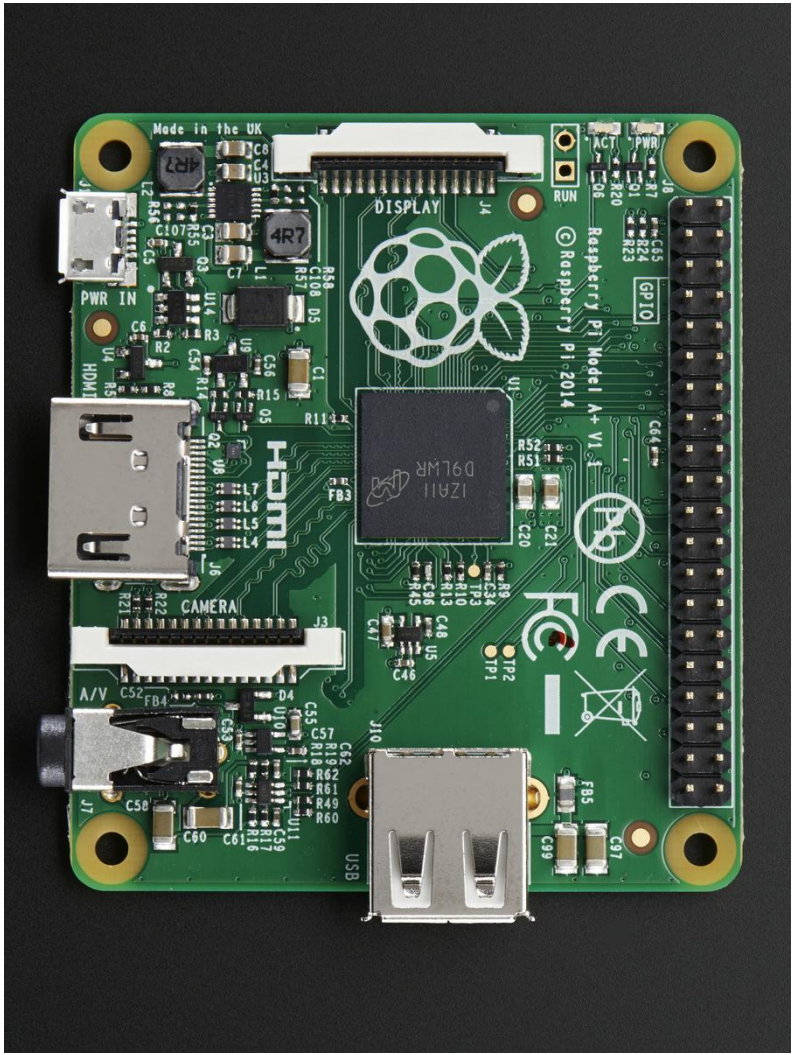
Outline

- Staff:
  - Dawson Engler (“Dawson” is fine: if stuff is broken it’s my fault)
  - CAs: Joe Tan, Matthew Sotoudeh (paid for outside class stuff)
  - SLs: Joseph Shetaye, Arjun Vikram (not paid for outside class stuff)
  - Volunteers: Ammar Ali Ratnani, Aaryan Singhal (not paid at all 🫡
- What
- Why

What

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

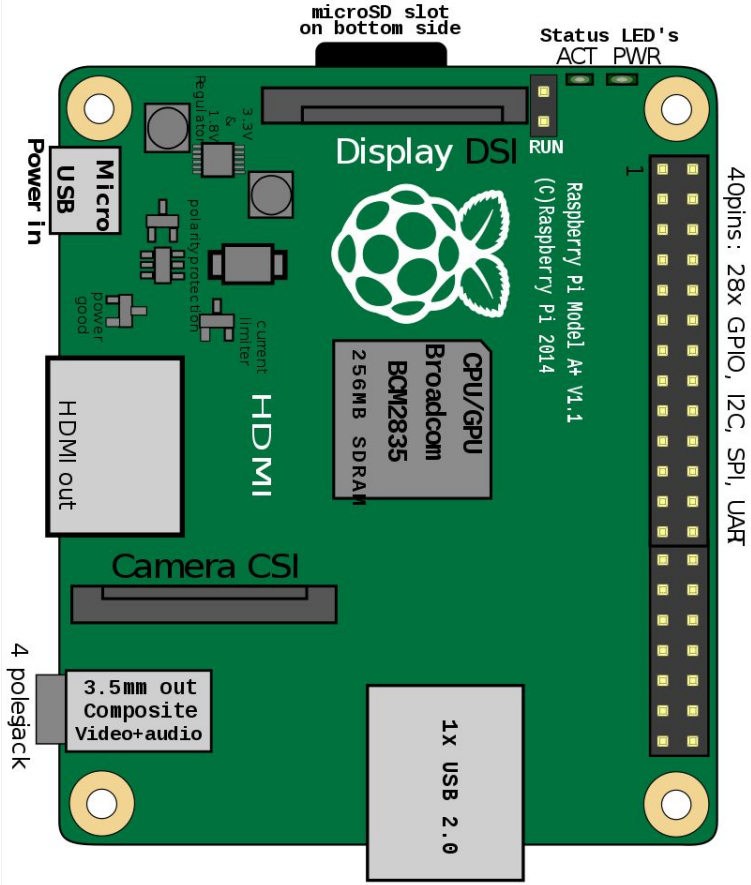
https://en.wikipedia.org/wiki/Raspberry_Pi



What

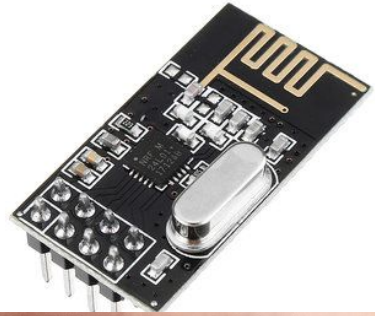
Write small, clean OS on an r/pi A+:

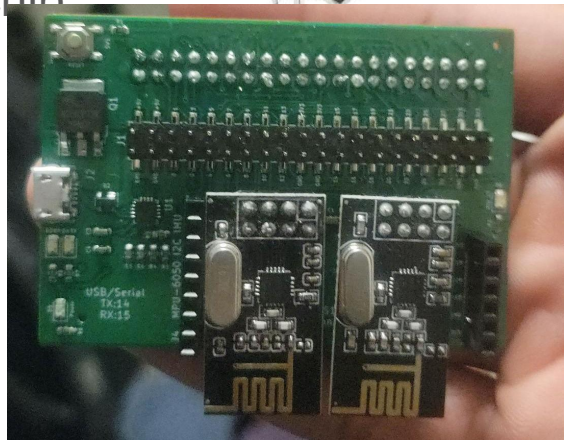
https://en.wikipedia.org/wiki/Raspberry_Pi



What you will build (different than last year)

- Bootloader (ship code from your laptop to the pi)
- GPIO, UART, SPI Device drivers
- Threads + interrupts
- Virtual Memory
- Simple fat32 file system on SD card.
- Networking code that uses the cheap nrf24L01 chip
    - Will use a pcb that Parthiv designed as 340lx project





- End result: simple, small unix OS
    - Mostly your code

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.

Your OS

You will write (almost) all code.

Small + lightweight = you can do things impossible on modern OSes.

- Nanosecond latencies for messages
- Real time guarantees faster than expensive digital tools
- Exception tricks that let you build valgrind in < 1KLOC.
- ….

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.

Class philosophy, subtext

- This class is for people that are super interested in low level
  - If you're on the spectrum (hi): this class is designed for you.
  - Our goal: move fast with a bunch of people that are very interested.
  - Only take the class if (1) stanford is easy or (2) very interested and have an open quarter.
  - Our hope: we don't waste your time, and you cover adult stuff in a way and level haven't seen.
  - One result: you'll find the hardcore people at stanford you want to work with in the future.
- Warning:
  - Don't take this class thinking it is an "easier version of 112"
  - It is not.
  - Making harder: we don't have enough staff!

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.

> Epsilon sprint 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.

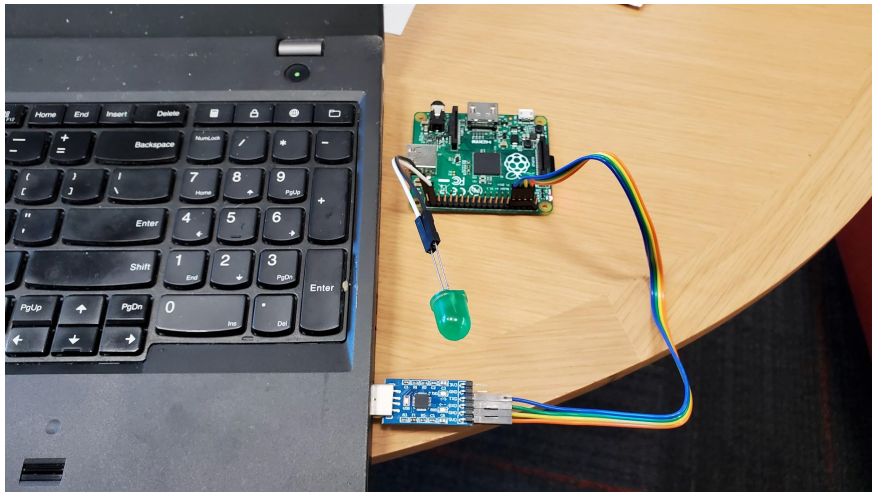
You get the following set of stuff:

To run you:

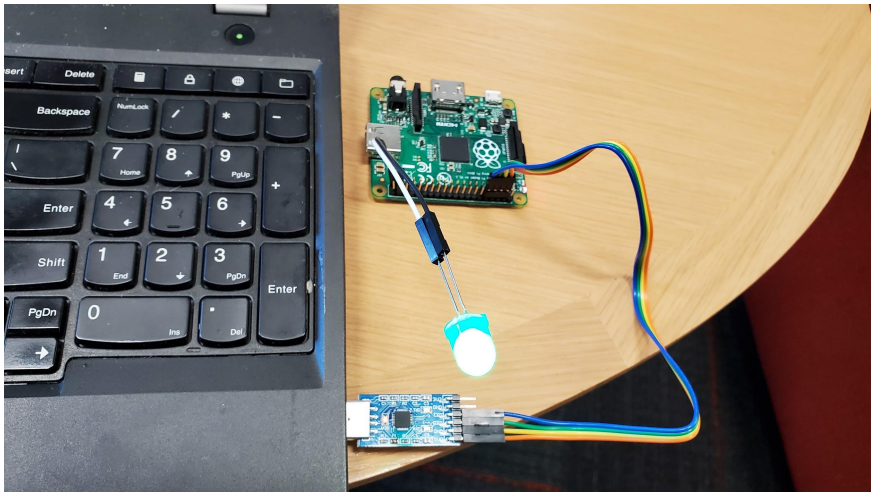
- Copy blink.bin to sd
- Wire up led
- Wire up serial device
- Plug into your laptop
- It doesn't work.
- But your partner's does

What is messed up?

Yours: Not working



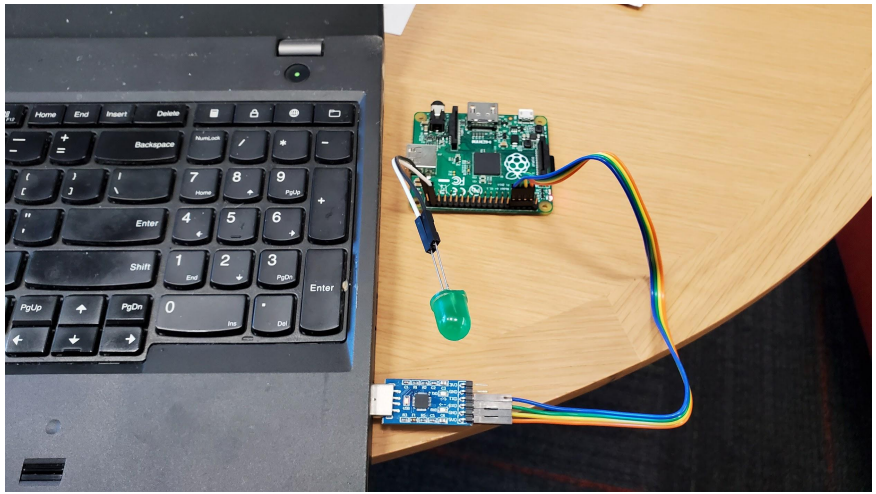
Partner's: Working



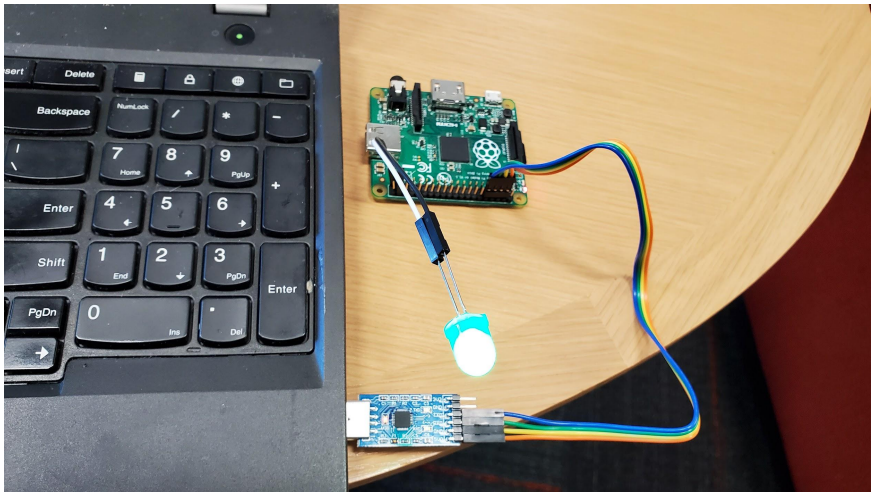
What to do first?

What is messed up?

Yours: Not working



Partner's: Working

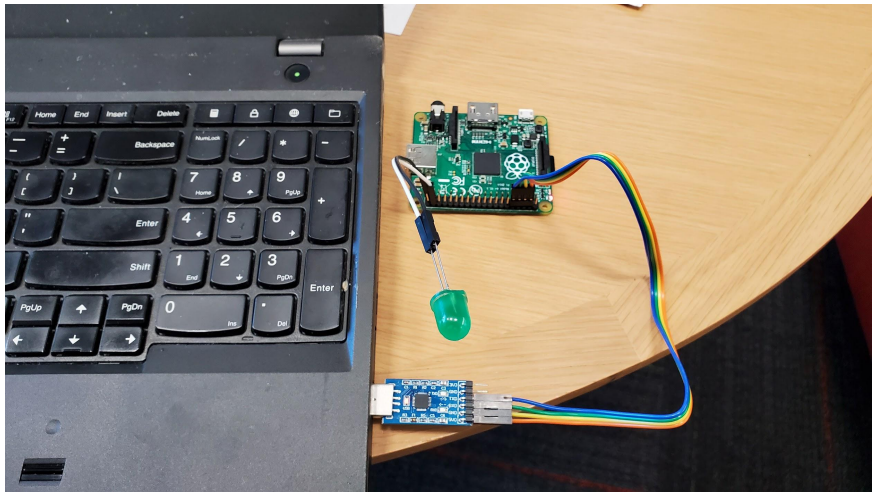


What does swapping tell you if doesn't work?

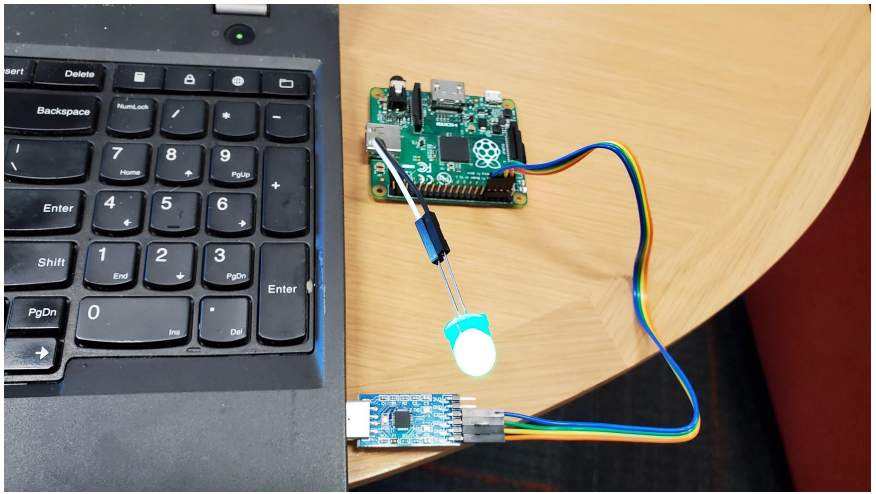
What does swapping tell you if works?

What is messed up?

Yours: Not working



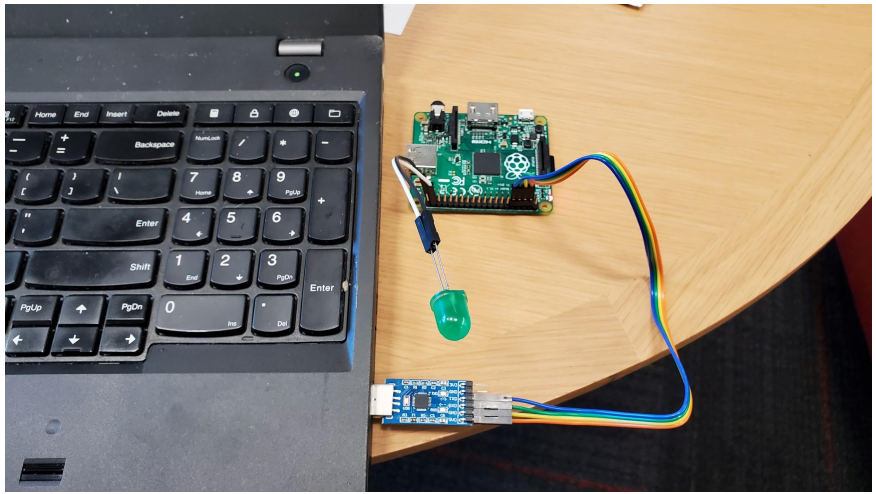
Partner's: Working



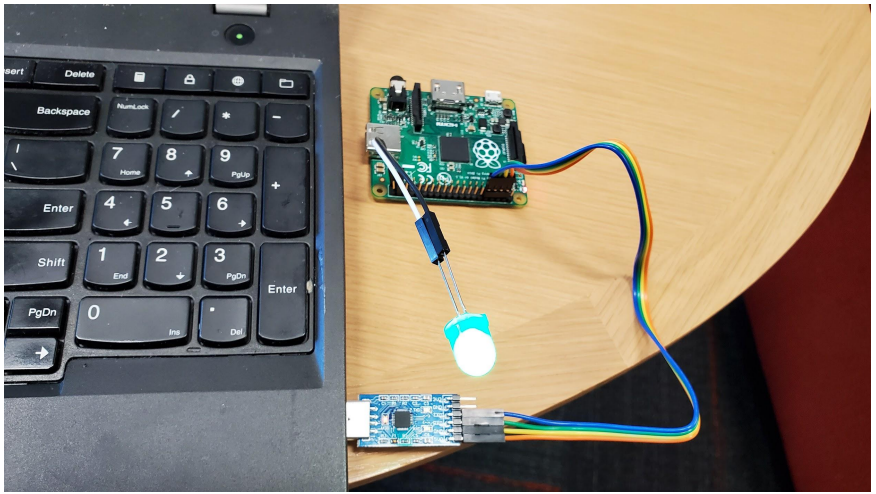
Swapping works: how to narrow down with least work?

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?

Epsilon sprinting: 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.

My epsilon-sprint theorem:

Given a working system W□ and a change *C*, then as |C| $\Longmapsto \epsilon$, the time + computation (IQ) 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

- Rules:
  - Feel free to leave early, but don’t be late (makes us run O(n))
  - Don’t use co-pilot or other people’s code. You have to write.
  - Don’t miss more than 2 labs (even that is very tough).
- Grade:
  - “A” requires 3+ hard extensions
  - Absolutely must turn in lab w/in 7 days. No exceptions!
  - Usually prelab due before lab. Won’t accept after.
  - Will have a final project (roughly 3+ labs worth of work)
  - Will have 2 homeworks.
- Plus side:
  - We pay for food.
  - We pay for equipment
  - We stay til 11:30pm or so.

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.

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.

If the rules on in-person meetings lift, it’s great if you can do the labs in groups.

(We will pay for food you order during lab if this happens.)

What to do now

- clone the class git repository:
  - https://github.com/dddrrreee/cs140e-25win
  - git clone git@github.com:dddrrreee/cs140e-25win.git

- For lab next tues, 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
  -
  - Do PRELAB for lab 2-trust