

15-410

*“Experience is what you get...
...when you don't get what you want.”*

Debugging
Sep. 17, 2004

Dave Eckhardt

What is “Debugging”?

Debugging is resolving a clash between stories

- Your hopeful story of achievement
- The world's sad tale of woe

The stories look alike!

- At the beginning, they both start with `main()`...
- Key step: finding the divergence

Stories are fractal

- You can zoom in on them and get more detail each time
- The divergence is typically a tiny detail
 - You will need to zoom in quite a lot

Move Beyond “Plot Summaries”

“When I install my keyboard handler it crashes”

- Insufficient detail
- This is a “plot summary”, not a tale of woe
- Don't ask us to look at your code yet!

Deepen your level of detail

- What was your story of hope, in detail?
- What parts of your story *already happened*?

Telling Your Story

“When I install my keyboard handler...”

- What do you really hope?
 - Hardware notices keyboard event
 - Hardware posts interrupt
 - CPU recognizes interrupt as keyboard interrupt
 - CPU responds to (vs. ignores) keyboard interrupt
 - CPU stores trap frame
 - CPU vectors through your IDT entry
 - Your wrapper is run
 - Wrapper calls C code
 - C code does ...

Pinpointing Depends on the Story

“...it crashes”

- Ok, that's generally what programs do
- Or, at least, that's when we start to pay attention to them...

The critical question

- How far did your story progress *before* the crash?

Pinpointing the problem

- How can you *measure* which steps worked ok?
 - “Keypress $\Rightarrow$ crash” tells you quite a bit!

Matching Phenomena to the Story

“Keypress $\Rightarrow$ crash” tells you quite a bit

- ✓ Hardware notices keyboard event
- ✓ Hardware posts interrupt
- ✓ CPU recognizes interrupt as keyboard interrupt
- ✓ CPU responds to (vs. ignores) keyboard interrupt
- ? CPU stores trap frame
- ? CPU vectors through your IDT entry
- ? Your wrapper is run
- ? Wrapper calls C code
- ? C code does ...

- 6 - **What now?**

Measuring

How can you *measure* the other steps?

- ? CPU stores trap frame
- ? CPU vectors through your IDT entry
- ? Your wrapper is run
- ? Wrapper calls C code
- ? C code does ...

Measurement Techniques

“Obvious”

- `printf()`
- single-step the program

Moving beyond the obvious

- Know your debugger
 - breakpoints, watchpoints
- Those pesky registers
 - `%esp`, `%eip` – these should *always* “make sense”
 - » You should always know what would be “sensible”!
 - `%CS`, `%DS`, `%SS` – not all that many legal values, right?
 - `%EFLAGS`, `%CR0` – “when the going gets tough...”

Measurement Techniques

Writing code

- Breakage of a complex data structure is, well complex
- Probably need code to check invariants
 - Doing it by hand is fun at most once

Asking for Help

“Plot summary” is not enough

- We probably have no idea what's wrong
 - Really!
 - Please see “triple fault” web page

You should always have a measurement plan

- What is the next thing to measure?
- How would I measure it?

You may reach the end of your rope

- Some things are genuinely tricky to debug
- Things in this class may occasionally qualify
 - This is a good learning experience

Asking for Help

When are you ready to ask for help?

- You have a long, detailed story – this is *critical*!!!
 - Based on lecture, handout, Intel docs
 - “Story” often needs one or two pictures
- Parts of the story are clearly happening
 - You have straightforward evidence, you are confident
- You have a measurement problem
 - Too many things to measure?
 - No idea how to measure one complicated thing?
 - Measurement results “make no sense”?

Summary

Debugging is about reconciling two stories

- “Plot summaries” aren't stories (you must zoom in)
- “If you don't know where you are going, you will wind up somewhere else.” — Yogi Berra

Measure multiple things, use multiple mechanisms

You should “always” have a next measurement target

When you see us, bring a long story

- ...which you will naturally be an expert on the first part of
- Try to know why each register has the value it does