ECE 471 – Embedded Systems
Lecture 25

Vince Weaver

https://www.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

6 November 2026

Announcements

- HW#8 is due
- Project topics are due
- Midterm #2 on Wednesday November 18th
- HW#9 will be posted, you can have two weeks as it's a bit more involved
- HW#5 finally graded

HW#9 – Summary

- Use a temperature probe (either SPI or 1-wire) and output the result to the i2c display
  - Re-use i2c display code from earlier homework
  - Re-use temp code (either TMP36 or the 1-wire)
  - Display the temperature on display
- When done can turn back in parts (assuming you aren't using them for the project)

HW#9 Notes – Modular Code

- In previous homeworks we put everything in one C file
- This isn't really practical for large projects
- By splitting things up into smaller files you can have some benefits:
  - Easier to organize/find code
  - Can re-use code easier
  - Less chance of merge conflicts when multiple people working on project in git
  - Can take common code and make libraries

HW#9 – Writing Modular Code

- In C you can compile each C file into its own object file, link together at end
- Each file does not need a `main()` function, you only need one per combined program.
- API defined in a header .h file

HW#9 – Header Files

- For example in the homework, we could put temperature read code into its own file with a `double get_temperature(void)` interface
- For other C files to see this, you need to export the definition. Usually this is done by putting the advance definition `double get_temperature(void);` in a .h header file and then including it in the other files
- Note: don't put full C functions in header files. I know this is a C++ thing but it's usually frowned upon when programming in C

HW#9 – Building Modular Code

- To link the various .o files together involves the "linker". However it's easier to just let gcc do it (gcc knows how to run the linker for you) `gcc -o display_temp display.o temperature.o`
- The linker merges the .o files into one big executable, and makes sure the placeholders to functions/variables in all of the files get the right addresses/pointers to where things live in the finished executable.
- How do you make sure when you change one C file that everything that uses it is also rebuilt? A well-crafted Makefile will have all these dependencies in place and will rebuild everything properly.

HW#9 – Code Libraries

- What if you want to make an official library?
- On Linux Static libraries are `.a`, dynamic are `.so`, (Windows `.dll`)
- Static libraries are just a bunch of .o files together and are linked to the executable at compile time
- Dynamic libraries are separate and get loaded when you run the program (the use a lot of magic to essentially live-patch your code to point to the library code on the fly)
- It's somewhat straightforwad to create libraries, just a few extra command line tools like `ar` or maybe even just using `-shared` to gcc

Aside about Shared vs Static Libraries

- Shared libraries, only need one copy of code on disk and in memory
  - Good for embedded system (less room needed)
  - Good for security updates (only need to update lib, not every program using it
- Static libraries, all libraries included
  - No dependencies
- These days maybe containers, docker, kubertenes
- Also Flatpack, Snap. Why? Stability, Know package will work on all distributions, Not have to install dependencies
- Can use `ldd` to view library usage

HW#9 – Converting Floating Point to Digits

- Use `sprintf()`

```
char string[128];
double temperature;
sprintf(string,"%.1lf",temperature);
/* Now string[0] has first digit, string[1] second, etc */
/* You might need to convert from ASCII as well */
```

- Use division/modulus

```
double temperature=23.4;
int hundreds,tens, ones,remainder;

hundreds=temperature/100;
remainder=temperature%100;
tens=remainder/10;
ones=remainder%10;
```

HW#9 – Specification (1/4)

- Temperatures from 0 to 99.9 degrees, inclusive. $0.0 \leq temp \leq 99.9$
- These should be displayed as two digits, a decimal point, another digit, and then a degree symbol (which is just a crude circle made of the top 4 segments on the display).
- For example: "24.5°"
- Leading zeros should be suppressed (i.e. display "2.4°" not "02.4°", 0 should be "0.0°")

HW#9 – Specification (2/4)

- Temperatures between -99.9 and 0 degrees
  $-99.9 \leq temp < 0$
- These should display a minus sign and then two digits of temperature, then the degree symbol.
- For example: "-25°"
- For temperatures between 0 and -9.9 be sure to print two digits of result (with decimal point). For example, "-2.5°"

HW#9 – Specification (3/4)

- Temperatures between 100 and 999 degrees.
  $100 \leq temp \leq 999$
- Should print three digits of temperature, then the degree symbol.
- For example: “245°”

HW#9 – Specification (4/4)

- Invalid temperatures that won't fit the display (and errors reading the thermometer) should be reported (via the display) in a method that isn't a valid temperature.
- It is your choice how to indicate this.

HW#9 – Writing Good Testcases

- Once you have written your nice modular code, how can you test it?
- Need to write some test cases that test a wide range of behaviors
- In the homework I have you think up some test cases

Computer Security:

and why it matters for embedded systems

- Most effective security is being unconnected from the world and locked away in a box. Until recently most embedded systems matched that.
- Modern embedded systems are increasingly connected to networks, etc. Embedded code is not necessarily prepared for this.
- Internet of Things: IoT (the S is for Security)

Computer Security – The Problem

- Untrusted inputs from user can be hostile.
- Users with physical access can bypass most software security.

What can an attacker gain?

- Fun / Mischief / Curiosity
  "Some people want to watch the world burn"
- Profit
- A network of servers that can be used for illicit purposes (SPAM, Warez, DDOS, bitcoin mining)
- Spying on others (companies, governments, etc)

Sources of Attack

- Local
  - Typing directly
  - USB Keys (CD-ROMs)
    Autorun/Autostart on Windows
    Scatter usb keys around parking lot, helpful people plug into machine.
- Remote/Network
  - cellphone modems
  - ethernet/internet
  - wireless/bluetooth

Ways to Break In

- Backdoors
  Debugging or Malicious, left in place
- Brute Force – trying all possible usernames/passwords

Types of Security Compromise

- Crash
  "ping of death"
- DoS (Denial of Service)
- User account compromise
- Root account compromise
- Privilege Escalation
- Rootkit (modify OS kernel to hide attacks)
- Re-write firmware? VM? Above OS?

Unsanitized Inputs

- Using values from users directly can be a problem if passed directly to another process
- If data (say from a web-form) directly passed to a UNIX shell script, then by including characters like ; can issue arbitrary commands:
  `system("rm %s\n",userdata);`

SQL Injection Attacks

- Escape characters can turn a command into two, letting user execute arbitrary SQL commands
- xkcd comic: `https://xkcd.com/327/`
- Name your child:
  `Robert '); DROP TABLE Students;--`