ECE 471 – Embedded Systems
Lecture 5

Vince Weaver

http://web.eece.maine.edu/~vweaver

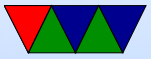
vincent.weaver@maine.edu

13 September 2019

Announcements

- Any questions on HW#1?
  Will go over answers next class.

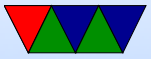
- HW#2 will be posted today



Homework #2 background

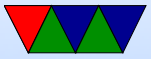
- It's mostly about getting your pi up and running, a small C coding assignment, and some short-answer questions.

- The directions will have you copy a file to your pi. It's a .tar.gz file. What is that?
  Sort of the Linux equivalent of a zip file
  tar = tape archive (ancient history) that runs lots of files together
  gz = gzip, which compresses it (makes it smaller)
  you may see other (Z, bz2, xz). What are the differences?



Mostly in compressed size vs compress/uncompress resources
gzip good enough for what we are doing.

- Coding homework is to take some existing C code and modify it. Can use the editor of your choice. Many on Linux. “nano” is easy. “vim” if you are serious about Linux. Also various graphical ones, and if you want you can even code it up on your desktop/laptop, but you probably want to copy it over to test before submitting.

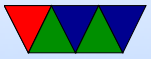


Why C?

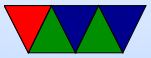
- Portability (sort of)
- Higher than assembly (barely)
- Why over Java or C++?

  They can hide what is actually going on. C statements map more or less directly to assembly. With things like operator overload, and exceptions, garbage collection, extra layers are added. This can matter for both size, speed, determinism, and real time.

  Might be restricted to a subset of C++

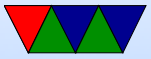


- Why over python?
  Mostly speed. (although you can JIT) Also if accessing low level hardware, in general you are calling libraries from python that are written in C anyway.
- What about Rust and Go? Don't overlook momentum of an old platform, sample code, libraries, etc.



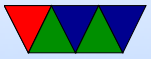
Downsides of C?

- Undefined behavior. Compiler is allowed to do anything it wants (including dropping code) if it encounters something undefined by the standard. This can be something as simple as just overflowing an integer or shifting by more than 32.
- "Enough rope to shoot yourself in the foot". C gives a lot of power, especially with pointers. It assumes you know what you are doing though. With great power comes great responsibility.



- Buffer overflows

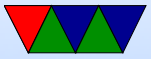
```
int a[5];
a[0]=1;              // fine
a[10000000]=1;       // obviously bad
a[5]=1;              // subtly bad
```



C compiling, Makefiles

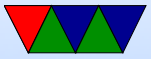
- C compiling on Linux
  We will use gcc (what others exist. clang?)
  Typical command line is something like:
  `gcc -O2 -Wall -o hello_world hello_world.c`
  `-O2` is optimization, `-Wall` is show all warnings
  A lot more options, see man page

- We use a Makefile to automate the process. What is make?
  You give it a list of dependencies, then it automatically



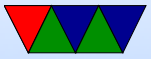
sees what files have changed and then runs commands to build things

Feel free to play with it, but a warning, tabs are significant so weird errors if you use spaces instead.



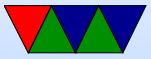
Cross compiling

- Can compile for a different architecture, for example x86 to ARM
- Why do it? Faster. Target doesn't have enough resources. Want to target multiple devices.
- To test would need an emulator (like qemu)



Comment your Code!

- Comment your code!!!!!
  Why?
  I will take points off it you don't.
  Also helps other people looking at your code figure out what's going on. Including me the graded. Including you trying to re-use some code a year from now.
  Having your name and a description of what the overall file and each function does doesn't hurt.
  Even fancier commenting conventions companies will



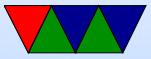
have for automated tools.

Mostly comment non-obvious stuff.

So `for(i=0;i<10;i++)` not so much.

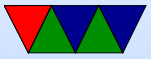
But something like `i=4.3+10*j;` yes.

You can't really over-comment (well you can, but it's harder to over-comment than under-comment)



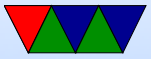
Using git

- Not using gitlab like ECE271, was huge hassle
- Still idea to use some sort of source control management (SCM)
- There are actually worse than git out there



Documentation on Linux commands

- Use `man command` where command is what you are interested in
- Use `man ls` to see how to use ls
- Also useful for functions `man -a printf` or random stuff `man ascii`



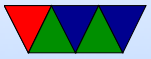
C Review

In past years sometimes the reason a HW assignment didn't work was due to using C poorly rather than misunderstandings of the desired algorithm.

- Loops in C

```
for(i=0;i<10;i++) {}
while(i<10) { i++}
do {} while(i<10);
```

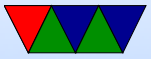
- printf



See the man page

How print an integer? `printf("%d",i);`. Character? String? floating point? More advanced formatting stuff

Escape characters like percent and quotes.



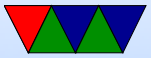
Common C Pitfalls

- Out of bounds in memory (see the `a[5]` example earlier. Also a problem with malloc() memory, Valgrind can help with that.

- Missing braces

```
if  (a==0)
        b=2;

if  (a==0)
```

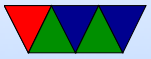


```
b=2;
c=3;
```

- = vs ==

```
if (a=0) do_something_important()
```

- Never ignore warnings from the compiler!



Debugging – when things go wrong

- Use a debugger like gdb
  - Compile your code with `-g` for debug symbols
  - Run `gdb ./hello`
  - `bt` backtrace, `info regis` gives register, `disassem` disassembles, etc.
- Sprinkle printf calls