ECE 471 – Embedded Systems
Lecture 5

Vince Weaver

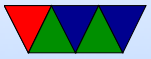
http://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

9 September 2022

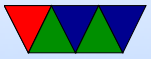
Announcements

- HW#2 will be posted
- Note you won't need to bring your Pi to class



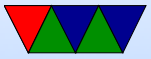
Why C?

- Portability (sort of) (i.e. how big is an `int`)
- Higher than assembly (barely)
  Pearce: "all the power of assembly with all the ease-of-use of assembly"
- 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,



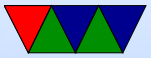
speed, determinism, and real time.
Might be restricted to a subset of C++

- 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.

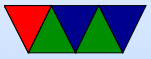


Downsides of C – Security

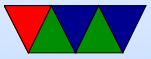
- Biggest issue is memory handling (lack thereof)
- Buffer overflows

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

- How can that go wrong? Crash? Corrupt Memory? Wrong results? Total system compromise?

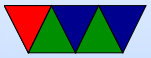


- Overwriting stack can be bad, as return address there
- Especially if user input going into the variable



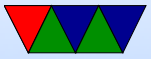
Homework #2 – Background

- It's mostly about getting everyone up to speed on the Pi as not everyone has used one before
  - Many ways to set up your Pi for use, everyone has a different preference
  - Be sure to change your password from default
- Also a small C coding assignment, and some short-answer questions.



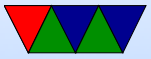
Homework #2 – Transferring Assignment

- 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.



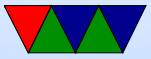
Homework #2 – Editing C on Pi

- 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.
  - "emacs" – I've known some emacs wizards
  - Also various graphical ones
  - Modern (MS VS Studio? Eclipse? Atom?) but can take more RAM than the Pi has



Using the Pi for this Class – Two Challenges

- Getting to the point you can log in
- Getting files onto and off of the board. (Definitely needed for homework)

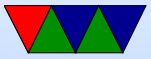


Installing Linux

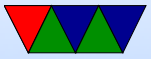
- Any Linux fine, I typically use Raspbian
  Using the same that I do is easiest and I can more easily help

- Easiest way is to buy SD card with image pre-installed
  Also can get NOOBS which will give you the option to select from a variety of images via menu (allowing to install Raspbian)

- If starting with a blank SD card,

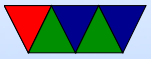


`https://www.raspberrypi.org/downloads/`
has good step by step instructions for getting an image and putting it on a card for a variety of operating systems.
Warning: it's a large download (900MB?) and takes a while to write to SD (which is slow)
dd on Linux, be sure **to get right partition**



Booting Linux

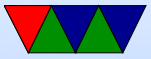
- Why called booting? Bootstrapping? Pull oneself up by own bootstraps? Meant to do something impossible
- Easiest if you have a USB keyboard and HDMI display connected.
  - Put SD card in
  - Hook up input/output (see later)
  - Plug in the USB power adapter; *NOTE* can also draw power over serial/usb and HDMI
  - Lights should come on and blink and should boot



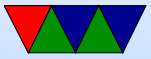
A number of raspberries should appear and some Linux boot messages

- Things can also go wrong in ways hard to troubleshoot

- First boot a menu comes up. You probably want to do a number of things:
  - Expand to fill disk.
  - Change password if you want
    pi/raspberry is default
  - Change locale— probably defaults to England giving pound char for #. en_US.UTF8, not GB
  - change hostname?



  - for this class, advanced options, enable i2c and spi
- You can get back to the original menu with `sudo raspi-config`
- Don't make fun of the text interface, once upon a time it's all we had.

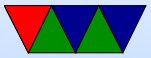


Other Optional things you can do

- Install updates
  sudo apt-get update
  sudo apt-get upgrade

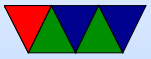
- Add a user account
  adduser vince

- Give new user sudo access
  involves text editing /etc/sudoers



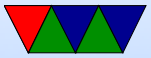
Connecting to the Pi

- Monitor/Keyboard (Easiest)
- Network Connection
- Serial Connection



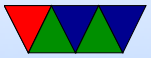
Monitor and Keyboard

- HDMI monitor, USB keyboard, USB mouse (optional unless using gui)
- Need HDMI cable.
- Used to be a nice setup in the Electronics Lab but I don't think that exists anymore unfortunately.

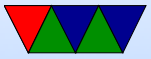


Network/Ethernet Connection

- Ethernet cable
- Either an Ethernet port, or connect direct to PC
- If something goes wrong on boot hard to fix
- Can also try this with a wireless connector
- Can hook it onto dorm network, but need to request a static IP. Can also direct connect between PC (configure pi with a local address like 192.168.1.2 and set your

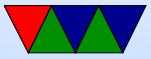


wired Ethernet on PC side to something like 192.168.1.1 and then use ssh to connect)



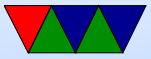
Network/netatalk

- Only works with MacOS (?)
- Some students in the past have used netatalk to connect to their Pi and copy files
- Look for info on Raspberry Pi and "netatalk"



Serial Connection

- Old fashioned, but very good skill to have.
- Need USB/serial adapter
- Need another machine to hook to, with a comm program minicom, putty
- Thankfully unlike old days don't need specific NULL modem cable. Still might need to set some obscure COM port settings (BAUD, stop bits, parity) and console TERM settings (ANSI, VT102).



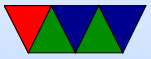
Transferring Files

- Easiest: Putting USB key in rasp-pi
  Easier on B+ (4 USB ports)
  In theory the Pi should auto-mount the drive for you
  May need to mount / umount by hand or be root

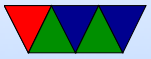
- Network: just use ssh/scp

- Serial: sz/rz ZMODEM

- Putting sd-card (after unpowering!) in another machine.

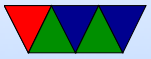


Challenge: Filesystem is in Linux format (ext4) so Windows and Macs can't read it by default.



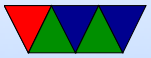
What you will do before starting HW2

- Get Linux installed
- Login with the default user/password (on Raspbian it is pi / raspberry)
  You can use `adduser` to add a new user and/or `passwd` to change a password.
- Learning a little bit of Linux. Most importantly compiling C/asm programs and transferring HW assignments in and out



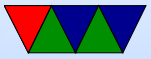
SD Card Digression

- Why are they so slow?
- BACK UP YOUR WORK. ALL THE TIME. SD cards corrupt easily. Why?
- SHUTDOWN CLEANLY
  menu or `shutdown -h now`
- Try to get things done a little before the deadlines, that way you have some time to recover if a hardware failure does happen.



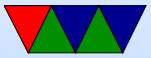
Using the Pi

- If using monitor/keyboard you can type `startx` after logging in and getting a nice GUI interface.
- You can do many things through that, but in this class we will use the command line for many things.
- You can select `lxterm` to get a terminal.
- Also if you log in over ssh or connect via serial port all you will get is the command line.



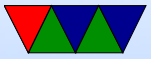
Homework #2 – Editing C on your own computer?

- 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.
- Be careful not to introduce errors if cutting and pasting



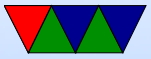
HW#2 Something Cool

- ANSI escape color/art – Demoscene!
- In old days: how could you do colors on screen?
- Over serial port, so couldn't directly write to video card
- Escape sequences: A pattern not normally typed by accident
- ESCAPE (ASCII 27) followed by [then some pattern of characters.
- Can move cursor, change colors, etc.
- Back in the day I used to make a lot of ANSI/ASCII art.



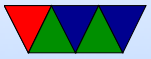
C compiler

- 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



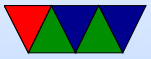
Makefiles

- 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.



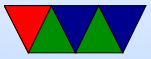
Cross compiling

- 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



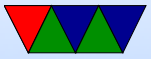
have for automated tools.

Mostly comment non-obvious stuff.

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

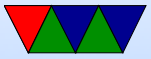
But something like `i=4.3+10*j;` yes.

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
- Who wrote git? Linus Torvalds.



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`