Intro to Linux and gcc/g++

Fall 2017 - Aaron S. Crandall, PhD

Today’s Outline

- Announcements
- Thing of the Day
- What is Linux (actually, what is an OS?)
- What is g++ (actually, what is a compiler?)

Announcements

- Linux User’s Group is here to help you - tutoring and office hours

Thing of the Day: My old college buddy got a job! Moral of story: Don’t burn bridges

- https://www.tesla.com/blog/welcome-chris-lattner

Welcome Chris Lattner

The Tesla Team • January 10, 2017

We would like to welcome Chris Lattner, who will join Tesla as our Vice President of Autopilot Software. Chris's reputation for engineering excellence is well known. He comes to Tesla after 11 years at Apple where he was primarily responsible for creating Swift, the programming language for building apps on Apple platforms and one of the fastest growing languages for doing so on Linux. Prior to Apple, Chris was lead author of the LLVM Compiler Infrastructure, an open source umbrella project that is widely used in commercial products and academic research today.

What is Linux and why do I care so much?

- Today we’re going to do a bit more on:
  - Linux
  - Operating systems
  - Compilers
  - Philosophy
  - Tools
- The goal is to give you the basics to do the work for this course

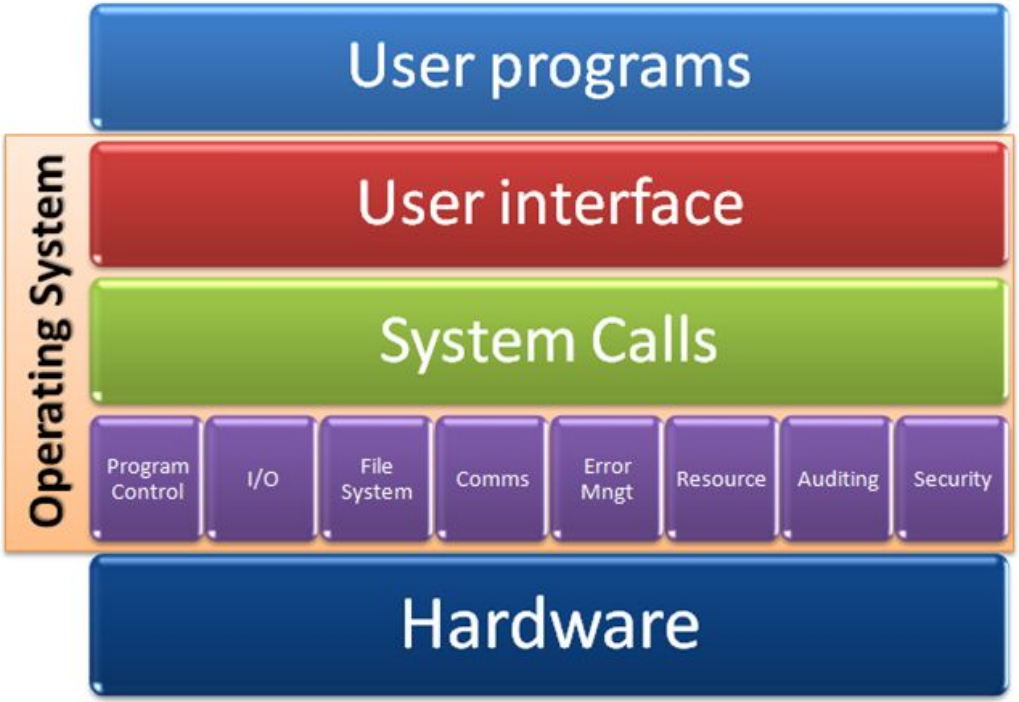
First off: What is an operating system?

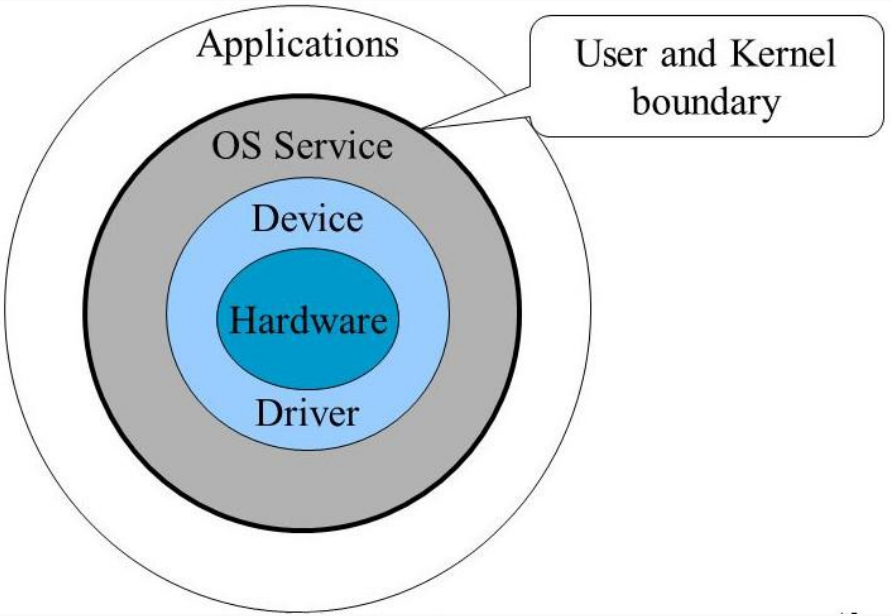
- I’ve been inured in tech for so long that this is something I forget people need to learn. It used to be a much more common point of discussion when it came to buying computers, but everything works so well now that operating systems have faded into the background.

First off: What is an operating system?

Short answer: It’s a program

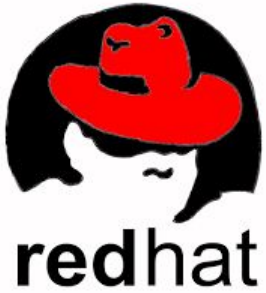
Some more detail about that “it’s a program thing” - Either a stack or an onion view





Officially “Linux” is just a kernel

- Linux is only the operating system kernel
- Colloquially, it’s used to refer to a whole distribution with user tools
- The result is the GNU/Linux name, since the user tools are (mostly) GNU
- Packaged together they become Linux distributions or “distros”
  - Debian, Ubuntu, Arch, SUSE, RedHat, etc.



There are LOTS of operating systems

- Linux
- WinNT
- Mach (OSX)
- BSD (freeBSD, OpenBSD, NetBSD)
- Plan9

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

People get attached to their OS environments, but it’s a false war

Just use the best tool for the job. I’ll mock various tools for their oddities and issues, but I’ve used every one of them as the situation warranted

Here’s the Dilbert take on UNIX

WAIT! Isn't Linux UNIX?

Ken Tompson & Dennie Ritchie

- No, well… kinda. They're cousins.
- UNIX was developed at AT&T Bell Labs 1971.11.03
  - Ken Thompson and Dennis Ritchie (invented C '69-'73 to rewrite UNIX)
- Linux was modeled after Minix (a stripped down UNIX clone for education)
  - Released by Linus Torvalds in October 1991
- Linux was inspired by UNIX, but doesn't share the same code
  - They do share many of the same philosophies, though: https://goo.gl/99Owcx
- There are still official UNIX OS releases:
  - FreeBSD, OSX, others: https://www.opengroup.org/openbrand/register/

Okay… so what's that UNIX philosophy?

- https://en.wikipedia.org/wiki/Unix_philosophy
  The Unix philosophy, originated by Ken Thompson, is a set of cultural norms and philosophical approaches to minimalist, modular software development.
- Emphasis on building simple, short, clear, modular, and extensible code that can be easily maintained and repurposed by developers other than its' creators.

Here’s one summary from 1978

1. Make each program do one thing well. To do a new job, build afresh rather than complicate old programs by adding new "features".
2. Expect the output of every program to become the input to another, as yet unknown, program. Don't clutter output with extraneous information. Avoid stringently columnar or binary input formats. Don't insist on interactive input.
3. Design and build software, even operating systems, to be tried early, ideally within weeks. Don't hesitate to throw away the clumsy parts and rebuild them.
4. Use tools in preference to unskilled help to lighten a programming task, even if you have to detour to build the tools and expect to throw some of them out after you've finished using them.
   (Write programs to help you work with data instead, even if you throw them away right afterwards)

GUI vs. Command Line

VT100 terminal

- The eternal question (since about 1973)
- Both are great! (I love me some GUIs)
- But… since all data are strings in the end, the command line has more power for the user to interact with the OS
- Read “In the Beginning Was the Command Line” by Neal Stephenson
  - http://faculty.georgetown.edu/irvinem/theory/Stephenson-CommandLine-1999.pdf
  - Highly recommended for understanding UNIX and Linux philosophy
  - Best chapter is on the Hole Hawg. I’d read it here if I have time.
  - BTW: Stephenson is one of the best cyberpunk/tech authors ever. Cryptonomicon FTW

Why Linux and not [Fill in the blank]

- Probably 2 primary reasons: Control & Utility
  - You have control over your system (at ALL levels)
  - It has a large suite of extant tools for most uses
    - BSD is awesome, but it just doesn't have the devs behind it
    - Mostly because the BSD license doesn't create the mandatory shared space that the GPL license does
- Linux can be used for almost computing environment
  - The Kernel scales very well, and you can use the source to suit your needs
- Linux/UNIX was designed for remote access by default
  - Windows kind of backed into the Internet. GUIs aren't great over the network.
  - UNIX was built for timesharing mainframes and Linux inherited that powerful structure

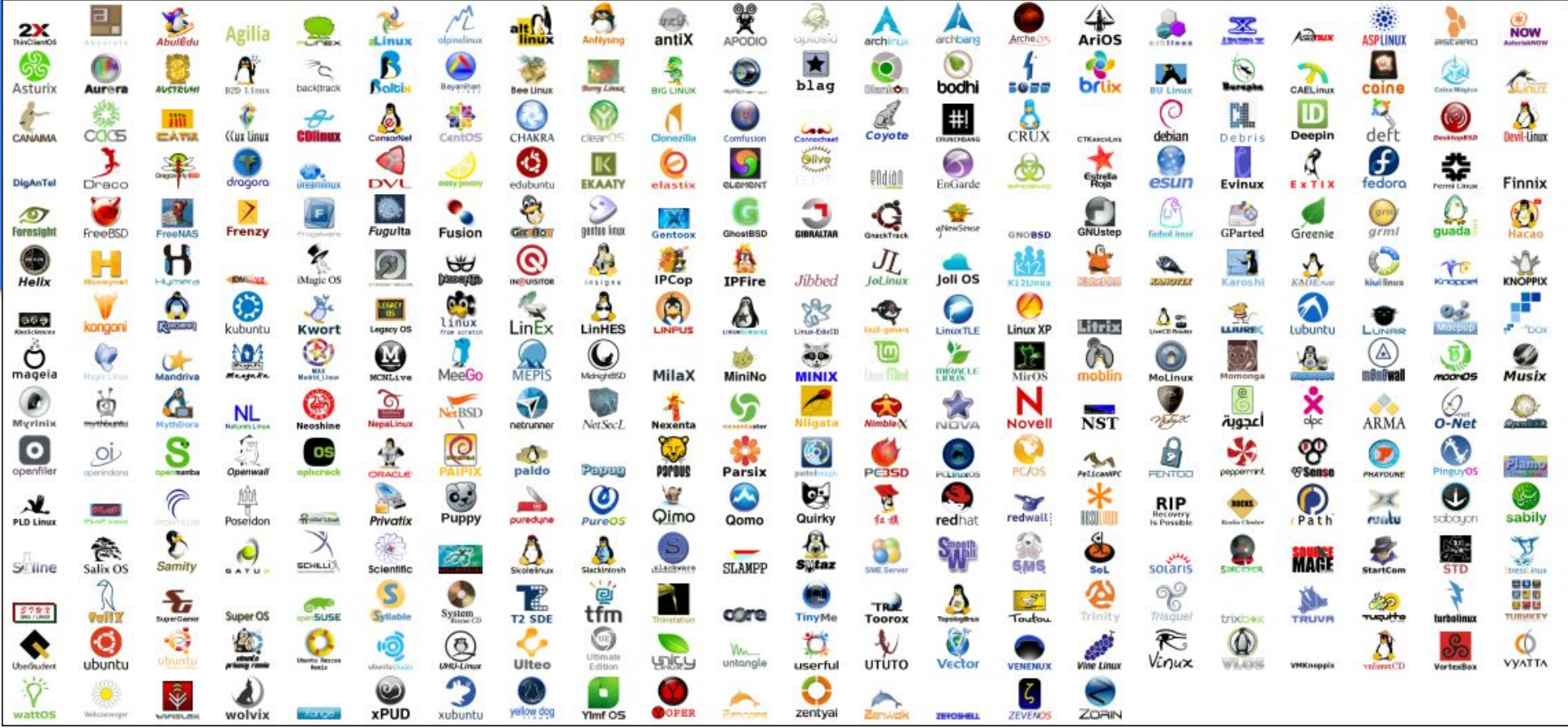
Plus… Why is Linux free?

FSF GNU Linux Penguin (Tux)

- We'll talk about Open Source and Free Software over time
  - Basically: if you don't have the source to your programs you have nothing
  - This actually does matter
  - The Free Software Foundation (FSF) are the makers of GNU (GNU's not UNIX)
    - They also released the GPL license to make copyright work for us
  - Open Source Software (OSS) is the idea that software should be shared
    - OSS created a huge field of people developing code and releasing it to others
- Linux is released under the GPL license, as are most OSS tools and distros
  - https://en.wikipedia.org/wiki/GNU_General_Public_License
  - https://github.com/torvalds/linux

So, which Linux distro should I use?

- Actually, all of them have the GNU gcc/g++ compiler
  - That's what they use to build the Linux kernel itself, so this is a requirement
  - I'd suggest one of the big names to start, but probably Debian or Mint
    - Debian, Mint, Ubuntu, RedHat/Fedora/CentOS
  - If you feel like reaching:
    - Arch, ElementaryOS, geez 100+ options: https://goo.gl/mltHMA



There’s a whole lotta Linux distros out there

Now… to sum up OSes:

- An operating system manages and mediates the hardware in your computer.
- An operating system launches other programs and schedules them
- An operating system manages memory and disk use
- The user starts in a shell (GUI or command line), which launches other applications as needed
- Pick the right one for the job! But, I’m hoping that we can convince you of why Linux has taken over all serious lifting in the computer world

Now, onto compilers

- What is a compiler?
  - It's a program! (or several programs that work together)
  - It's a very special program - it has to take itself as input!
  - It takes a string of text and converts it to a different string of text
    - This is actually all a program can ever do, but I digress
  - It takes a string in one language and converts it to another language:
    - C -> Assembly -> Machine code
    - Python -> Assembly -> Machine code
    - Java -> Byte Code
- Visual Studio has a compiler within the IDE called Visual C++

There’s LOTS of compilers out there

- Visual C++
- GNU gcc/g++
- Intel C Compiler
- Python interpreter
- Ugh. It’s a huge list: https://en.wikipedia.org/wiki/List_of_compilers

To make a programming language useful (beyond a spec), you’ll need to build a compiler for it.

So, the process of programming is:

1) Create a string in a given language
2) Pass that string to a compiler
3) Take results from compiler and execute those

You've been doing this all along inside of VS' IDE, but here it's going to be more explicit:

1) Edit a text file (or more text files)
2) Pass that text file to g++
3) Run resulting file as a program (Do an example, here, Crandall)

Options for Linux access

- SSH to EECS servers
- Install VirtualBox (or something else) and make a Virtual Linux machine
- Install Linux on a computer, either solely or dual boot

- You'll definitely need to do the SSH one, but the others can make development *much* easier since the GUI is more available.

SSH to EECS from a Windows computer

- SSH - Secure SHell
  - RSA-based encrypted network connection to a terminal (pseudo console / command line) on a computer
  - My favorite program for this is PuTTY (google & download it)
- SCP - Secure CoPy
  - Copies files over the SSH protocol to another computer
  - Used to copy your work/assignments to the EECS servers
  - My favorite program for this is WinSCP (google & download it)
    - Gives you drag & drop of files to the server

WinSCP

Quick VirtualBox intro

- VirtualBox is a program to run operating systems on other operating systems!
- These are called virtual machines
  - actually, the hardware is virtual, not the OS
- I use it for testing OSes, virtual networks, building evil tools, etc.

Some primary command line programs

- ls - List files in directory
- cd - change directory
- rm - remove file
- cp - copy file
- mkdir - make directory
- rmdir - remove directory
- nano / vi / emacs - edit a file
- ssh - use ssh to connect to server
- scp - copy file over ssh to server
- man - manual page for tools

- g++ - use GNU C++ compiler
- make - run make to build a program
- ps - list running programs
- kill - kill a running program
- top - watch running programs

Tons of Linux tutorials out there:
* https://ryanstutorials.net/linuxtutorial/
* http://linuxcommand.org/index.php
*https://www.codecademy.com/learn/learn-the-command-line
* http://www.ee.surrey.ac.uk/Teaching/Unix/

Next Class: More on Linux & development
Will also talk git, start reading the book, etc.

- How to edit code & files
- What are STDIN & STDOUT -> argc & argv!
- Command line options
- What does main(){return(1);} mean?
- Touching on the STL
- Starting debugging
- What is make?
- Yeah, it’ll go on for a bit, but we’ll get there