Linux Overview

Advanced Embedded Linux Development

with **Dan Walkes**

University of Colorado **Boulder**

Learning objectives:

Linux History

Linux Requirements

Linux contributors

Starting Out with Linux

- Embedded Linux started with the TiVo in 1999
- 2 billion devices running Linux as of 2017
- Why run Linux?
  - Moore's Law

Mastering Embedded Linux Programming Chapter 1
http://www.iretron.com/blog/posts/technology-flashback-tivo-1999/

Points Driving Linux Adoption

- Functionality
  - Support built in for scheduler, network stack, USB, WiFi, Bluetooth, storage, etc
- Ported to wide range of architectures.
- Open source, modifiable
- Active community - answer questions.
- No vendor lock-in

Linux Considerations

- Needs a 32 bit processor and ~16+MB of RAM, ~8MB of flash
  - Needs Memory Management Unit for all practical purposes
- Needs skill set of engineers (IE you!)
- May not be appropriate for some real-time applications.

The Players

- Open Source Community
  - Alliance of developers, not for profit, academic, commercial
  - Group for each set of applications
- CPU Architects/SOC Vendors/Board Vendors
  - Create reference hardware/ board support packages (BSP)s

Linux Command Line

Advanced Embedded Linux Development

with **Dan Walkes**

University of Colorado **Boulder**

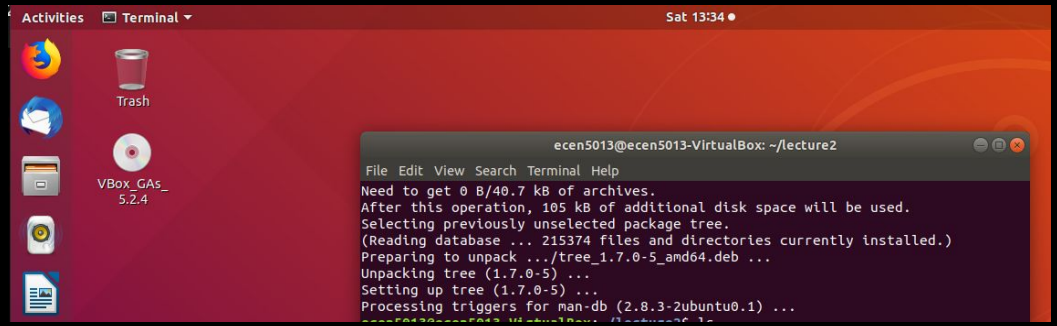
Learning objectives:

Overview of Linux Command Line

Common Linux Utilities

Why use the command line?

- Ubuntu has a full graphical UI… can I use that?



- Absolutely!
- However, our assignments are going to focus on automation and scripting - which require the terminal

Why use the command line?

- The command line is the only thing <every> linux system has.
  - This includes your embedded device.
- It’s often the only thing you have when troubleshooting problems.
- It’s often the easiest, fastest, and most reliable way to remotely access a system.

Common Linux Utilities - I/O

- echo
  - echo a command to the terminal

```
ecen5013@ecen5013-VirtualBox:~/lecture2$ echo "Hello World!"
Hello World!
```

- cat
  - concatenate (print out) a file to the terminal

```
ecen5013@ecen5013-VirtualBox:~/lecture2$ cat textfile.txt
ECEN_5013_IS_AWESOME!
```

https://nsdl.oercommons.org/courses/an-introduction-to-linux-2/view

Common Linux Utilities - Directory Navigation

- pwd
  - Print Working Directory

```
ecen5013@ecen5013-VirtualBox:~$ pwd
/home/ecen5013
```

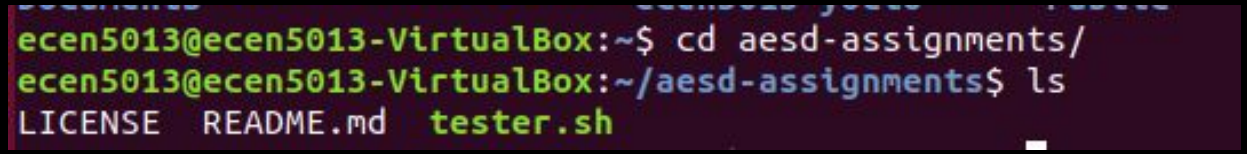
- ls
  - List contents of a directory

```
ecen5013@ecen5013-VirtualBox:~$ ls
aesd-assignments              Downloads           examples.desktop  Templates
assignments-complete-private  ecen5013-buildroot  Music             Videos
Desktop                       ecen5013-source     Pictures
Documents                     ecen5013-yocto      Public
```

https://nsdl.oercommons.org/courses/an-introduction-to-linux-2/view

Common Linux Utilities - Directory Navigation

- cd
  - Change directory



- mkdir
  - Make a new directory

https://nsdl.oercommons.org/courses/an-introduction-to-linux-2/view

Common Linux Utilities - Directory Navigation

- mv
  - Move files or directories to a new location
- cp
  - Copy files or directories to a new location
- rm
  - Remove a directory

https://nsdl.oercommons.org/courses/an-introduction-to-linux-2/view

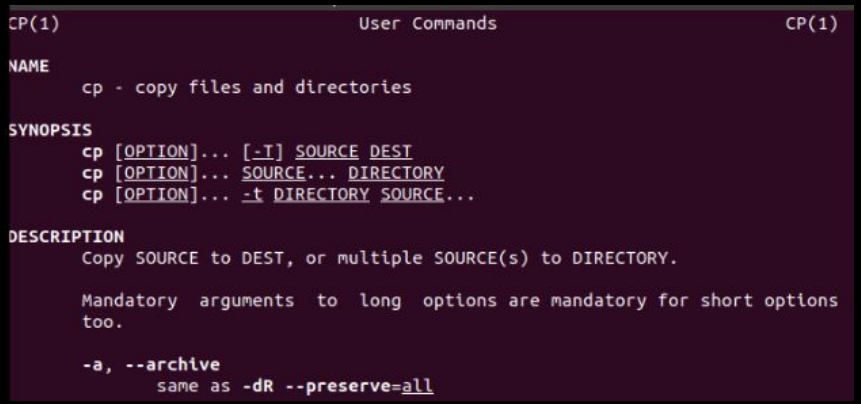
Common Linux Utilities

- touch
  - Update a file’s timestamp without editing if the file exists.
  - Create an empty file if the file does not exist.

https://nsdl.oercommons.org/courses/an-introduction-to-linux-2/view

Common Linux Utilities - man

- man < program or command>
  - “man pages” (short for manual pages)
  - Dates back to the Unix Programmer's Manual, 1971

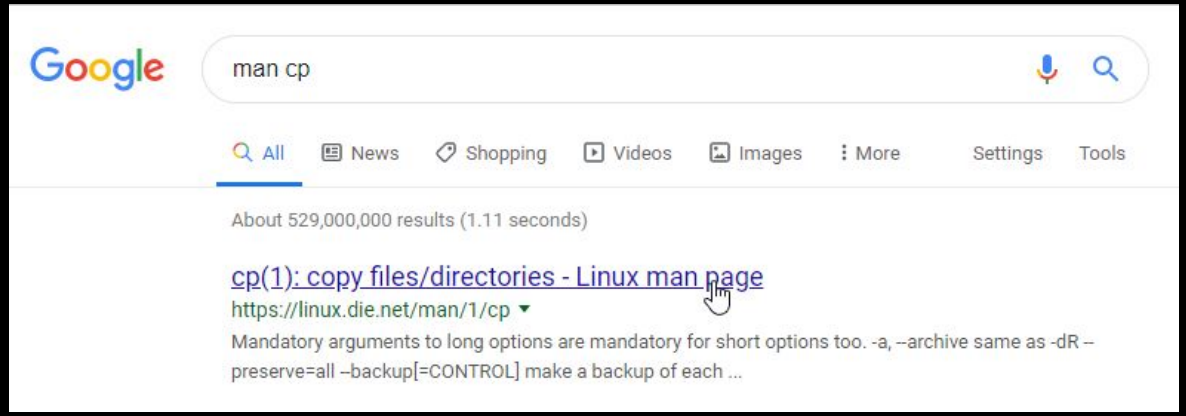


```
CP(1)                          User Commands                          CP(1)

NAME
       cp - copy files and directories

SYNOPSIS
       cp [OPTION]... [-T] SOURCE DEST
       cp [OPTION]... SOURCE... DIRECTORY
       cp [OPTION]... -t DIRECTORY SOURCE...

DESCRIPTION
       Copy SOURCE to DEST, or multiple SOURCE(s) to DIRECTORY.

       Mandatory  arguments  to  long  options are mandatory for short options
       too.

       -a, --archive
              same as -dR --preserve=all
```

https://nsdl.oercommons.org/courses/an-introduction-to-linux-2/view
https://www.bell-labs.com/usr/dmr/www/manintro.html

Common Linux Utilities - man

- Google knows about man pages too!



https://nsdl.oercommons.org/courses/an-introduction-to-linux-2/view

Common Linux Utilities - man

- Example of common copy command issue

```
dan@DESKTOP-BQMVP69:~/CU/aesd-lectures/lecture2$ mkdir a_directory
dan@DESKTOP-BQMVP69:~/CU/aesd-lectures/lecture2$ mkdir a_directory/a_subdir
dan@DESKTOP-BQMVP69:~/CU/aesd-lectures/lecture2$ mkdir a_copy_target
dan@DESKTOP-BQMVP69:~/CU/aesd-lectures/lecture2$ cp a_directory/ a_copy_target/
cp: -r not specified; omitting directory 'a_directory/'
```

cp(1) - Linux man page

Name

cp - copy files and directories

-R, -r, --recursive
copy directories recursively

```
dan@DESKTOP-BQMVP69:~/CU/aesd-lectures/lecture2$ cp -r a_directory/ a_copy_target/
```

https://nsdl.oercommons.org/courses/an-introduction-to-linux-2/view

cp -r

- Fix to use cp -r, validate with tree

```
dan@DESKTOP-BQMVP69:~/CU/aesd-lectures/lecture2$ cp -r a_directory/ a_copy_target/
```

```
dan@DESKTOP-BQMVP69:~/CU/aesd-lectures/lecture2$ tree a_copy_target/
a_copy_target/
└── a_directory
    └── a_subdir

2 directories, 0 files
dan@DESKTOP-BQMVP69:~/CU/aesd-lectures/lecture2$ tree a_directory/
a_directory/
└── a_subdir

1 directory, 0 files
```

https://nsdl.oercommons.org/courses/an-introduction-to-linux-2/view

Advanced Command Line

Advanced Embedded Linux Development

with **Dan Walkes**

University of Colorado **Boulder**

Learning objectives:

Editing from command line

Using the shell

Searching from command line

Wildcards, pipes, redirection

Permissions

Remote access to the command line

Editing - vi

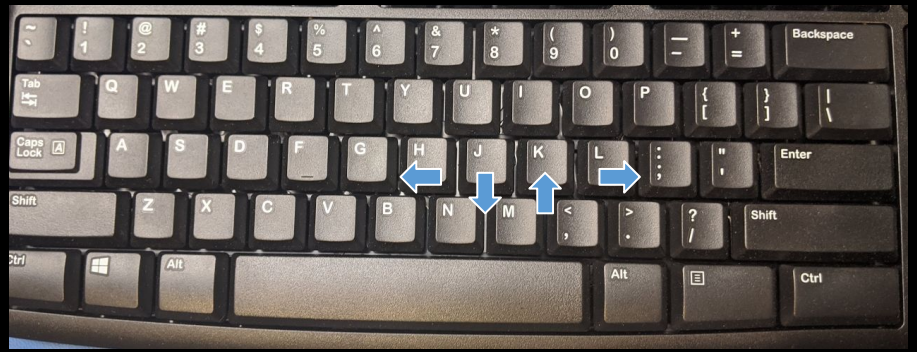
- Linux has lots of text editors to choose from.
- “vi” is one classic utility
  - Short reference for the “visual” mode of an early text editor utility on UNIX
  - Written in 1976
  - Often replaced by “vim” “VI improved”
- I suggest getting at least some familiarity with “vi”
  - Why? - It’s everywhere. In my experience it’s the only editor you can always count on being available.

http://ex-vi.sourceforge.net/
https://www.vim.org/

Editing - vi

- Basic commands:
  - vi <name> (or vim <name>)
- Entering “normal mode” for commands - use Esc key
  - write - Esc -> :w Esc -> :q - quit
  - Insert: i, Delete d
  - Navigation keys

Move	Key
Left	h
Down	j
Up	k
Right	l



https://nsdl.oercommons.org/courses/an-introduction-to-linux-2/view

Shell Interpreter Options

- A shell interpreter handles commands from the terminal (or from a script file).
- Several have been developed, many have similar command support.
- Bourne shell (sh) was one of the originals, released in 1979
- Bash shell (Bourne Again SHell) was created as a replacement for Bourne with features from ksh and csh.

https://www.gnu.org/software/bash/

Searching for Files

- Locate files/directories with find
  - Find files/directories below the current directory “.” with name a_directory

```
dan@DESKTOP-BQMVP69:~/CU/aesd-lectures/lecture2$ find . -name a_directory
./a_copy_target/a_directory
./a_directory
```

https://nsdl.oercommons.org/courses/an-introduction-to-linux-2/view

Searching for Content

- Searching for content with grep

```
dan@DESKTOP-BQMVP69:~/CU/aesd-lectures/lecture2$ cat textfile.txt
AESD_IS_AWESOME!
dan@DESKTOP-BQMVP69:~/CU/aesd-lectures/lecture2$ cat textfile2.txt
CU_IS_AWESOME!
dan@DESKTOP-BQMVP69:~/CU/aesd-lectures/lecture2$ grep -r "IS_AWESOME" *
textfile.txt:AESD_IS_AWESOME!
textfile2.txt:CU_IS_AWESOME!
```

https://nsdl.oercommons.org/courses/an-introduction-to-linux-2/view

Wildcards

```
dan@DESKTOP-BQMVP69:~/CU/aesd-lectures/lecture2$ cat textfile.txt
AESD_IS_AWESOME!
dan@DESKTOP-BQMVP69:~/CU/aesd-lectures/lecture2$ cat textfile2.txt
CU_IS_AWESOME!
dan@DESKTOP-BQMVP69:~/CU/aesd-lectures/lecture2$ grep -r "IS_AWESOME" *
textfile.txt:AESD_IS_AWESOME!
textfile2.txt:CU_IS_AWESOME!
```

- * in means files with any character in the name
- textfile*txt would have meant any file starting with textfile and ending with txt
- expanded by the shell to
  - grep -r “IS_AWESOME” textfile.txt textfile1.txt textfie2.txt textfile3.txt textfile4.txt

https://ryanstutorials.net/linuxtutorial/wildcards.php

Pipes

- The Pipe Character ‘|’ sends the output of one command to the input of another command
  - Chain commands together based on standard input/output streams.

```
dan@DESKTOP-BQMVP69:~/CU/aesd-lectures/lecture2$ cat textfile.txt  | grep "IS_AWESOME"
AESD_IS_AWESOME!
```

https://nsdl.oercommons.org/courses/an-introduction-to-linux-2/view

Redirection

- Send the output of a command to a new file using >

```
dan@DESKTOP-BQMVP69:~/CU/aesd-lectures/lecture2$ grep -r "IS_AWESOME" * > searchresult.txt
dan@DESKTOP-BQMVP69:~/CU/aesd-lectures/lecture2$ cat searchresult.txt
textfile.txt:AESD_IS_AWESOME!
textfile2.txt:CU_IS_AWESOME!
```

- Append the output of a command into an existing file using >>

```
dan@DESKTOP-BQMVP69:~/CU/aesd-lectures/lecture2$ echo "hello" >> searchresult.txt
dan@DESKTOP-BQMVP69:~/CU/aesd-lectures/lecture2$ cat searchresult.txt
textfile.txt:AESD_IS_AWESOME!
textfile2.txt:CU_IS_AWESOME!
hello
```

https://nsdl.oercommons.org/courses/an-introduction-to-linux-2/view

File Permissions

- Control how a file may be used and by whom
- 3 levels of permission - User, Group and World (or everyone

  Use ls -l to show permission info

User (owner)			Group			Others (everyone)		
Read (r)	Write (w)	Execute (x)	Read (r)	Write (w)	Execute (x)	Read (r)	Write (w)	Execute (x)

```
dan@DESKTOP-BQMVP69:~/CU/aesd-lectures/lecture2$ ls -l
total 20
drwxr-xr-x 3 dan dan 4096 Jun 21 15:28 a_copy_target
drwxr-xr-x 3 dan dan 4096 Jun 21 15:28 a_directory
-rw-r--r-- 1 dan dan   65 Jun 21 15:35 searchresult.txt
-rw-r--r-- 1 dan dan   17 Jun 21 15:31 textfile.txt
-rw-r--r-- 1 dan dan   15 Jun 21 15:31 textfile2.txt
```

https://nsdl.oercommons.org/courses/an-introduction-to-linux-2/view

File Permissions

- Execute permissions typically not granted by default

```
dan@DESKTOP-BQMVP69:~/CU/aesd-lectures/lecture2$ cat helloworld.sh
#!/bin/bash
echo "hello world!"
dan@DESKTOP-BQMVP69:~/CU/aesd-lectures/lecture2$ ls -l helloworld.sh
-rw-r--r-- 1 dan dan 32 Jun 21 15:38 helloworld.sh
dan@DESKTOP-BQMVP69:~/CU/aesd-lectures/lecture2$ ./helloworld.sh
-bash: ./helloworld.sh: Permission denied
```

- Use chmod to change permissions

```
dan@DESKTOP-BQMVP69:~/CU/aesd-lectures/lecture2$ chmod u+x helloworld.sh
dan@DESKTOP-BQMVP69:~/CU/aesd-lectures/lecture2$ ls -l helloworld.sh
-rwxr--r-- 1 dan dan 32 Jun 21 15:38 helloworld.sh
dan@DESKTOP-BQMVP69:~/CU/aesd-lectures/lecture2$ ./helloworld.sh
hello world!
```

https://nsdl.oercommons.org/courses/an-introduction-to-linux-2/view

File Permissions

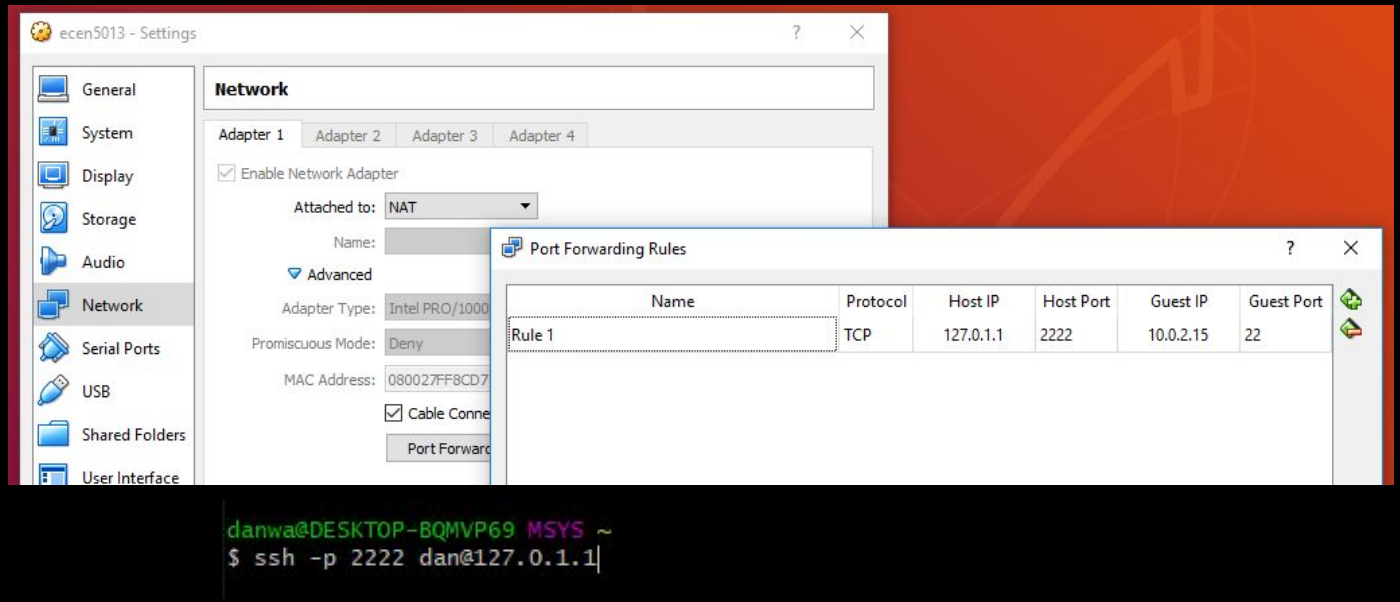
- Permissions can be specified by octal digits in numeric mode.
  - 4 = read
  - 2 = write
  - 1 = execute

```
dan@DESKTOP-BQMVP69:~/CU/aesd-lectures/lecture2$ ls -l helloworld.sh
-rwxr--r-- 1 dan dan 32 Jun 21 15:38 helloworld.sh
dan@DESKTOP-BQMVP69:~/CU/aesd-lectures/lecture2$ chmod 766 helloworld.sh
dan@DESKTOP-BQMVP69:~/CU/aesd-lectures/lecture2$ ls -l helloworld.sh
-rwxrw-rw- 1 dan dan 32 Jun 21 15:38 helloworld.sh
```

https://nsdl.oercommons.org/courses/an-introduction-to-linux-2/view

Remote Access

- You can use Secure Shell (ssh) for remote access to machines (including your VM)



https://nomanualentry.org/post/174407221318/ssh-into-your-virtualbox-guest

Remote Access

- Use a unix environment WSL or msys2 on Windows
  - https://docs.microsoft.com/en-us/windows/wsl/install-win10
  - https://www.msys2.org/
- Use scp for file transfer into/out of your VM

Transfer from windows to your virtual machine home directory

```
danwa@DESKTOP-BQMVP69 MSYS ~
$ scp -P 2222 /c/Users/danwa/Downloads/somefile.txt dan@127.0.1.1:~
```

SSH into your virtual machine

```
danwa@DESKTOP-BQMVP69 MSYS ~
$ ssh -p 2222 dan@127.0.1.1
```

Verify your virtual machine contains somefile.txt

```
dan@DESKTOP-BQMVP69:~$ ls -la | grep somefile
-rw-r--r--   1 dan  dan      0 Jun 21 15:45 somefile.txt
```

Transfer from virtual machine into your Windows system

```
danwa@DESKTOP-BQMVP69 MSYS ~
$ scp -P 2222 dan@127.0.1.1:~/somefile.txt /c/Users/danwa/Downloads/from_vm.txt
```

https://linuxize.com/post/how-to-use-scp-command-to-securely-transfer-files/

Scripting

Advanced Embedded Linux Development

with **Dan Walkes**

University of Colorado **Boulder**

Learning objectives:

Scripts Overview
Script Variables
Script Arguments
Script Conditionals

Scripts

- Scripting allows us to:
  - Automate frequently used commands
  - Customize commands for different scenarios using arguments and variables
- Think of scripts as documentation!
  - If you find yourself listing commands used to perform a task, consider scripting instead.
- Often are in files ending with .sh (short for shell)

Simple Script Example

- #!/bin/sh is a comment which tells the shell which shell interpreter to use
  - Also known as “shebang”
- Also common to see #!/bin/bash

```
dan@DESKTOP-BQMVP69:~/CU/aesd-lectures/lecture2$ cat helloworld.sh
#!/bin/bash
greeting=hello
echo "${greeting} world!"
dan@DESKTOP-BQMVP69:~/CU/aesd-lectures/lecture2$ ./helloworld.sh
hello world!
```

http://manpages.ubuntu.com/manpages/disco/en/man1/bash.1.html
https://linuxize.com/post/bash-shebang/

Script Variables

- Script Variables include:
  - Variables we define ourselves

```
dan@DESKTOP-BQMVP69:~/CU/aesd-lectures/lecture2$ cat helloworld.sh
#!/bin/bash
greeting=hello
echo "${greeting} world!"
dan@DESKTOP-BQMVP69:~/CU/aesd-lectures/lecture2$ ./helloworld.sh
hello world!
```

  - Variables passed into the script as arguments

```
dan@DESKTOP-BQMVP69:~/CU/aesd-lectures/lecture2$ cat helloworldargs.sh
#!/bin/bash
greeting=$1
echo "${greeting} world!"
dan@DESKTOP-BQMVP69:~/CU/aesd-lectures/lecture2$ ./helloworldargs.sh hola
hola world!
```

https://ryanstutorials.net/bash-scripting-tutorial/bash-variables.php

Script Argument

- Script special variables
  - $0 - the invoked name of the bash script
  - $1 - $9 - first 9 arguments to a script
  - $# - Number of arguments passed to the script
  - $? - The exit status of the most recent process

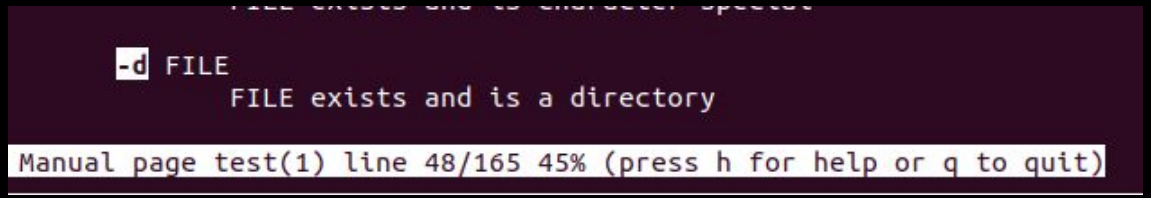
https://ryanstutorials.net/bash-scripting-tutorial/bash-variables.php

Bash Conditional if/else

- Conditional syntax for bash
- [ ] is shorthand for the “test” command

```
if [ conditional ]
then
    action
else
    action
fi
```

```
if [ -d "$WRITEDIR" ]
then
        echo "$WRITEDIR created"
else
        exit 1
fi
```



https://ryanstutorials.net/bash-scripting-tutorial/bash-if-statements.php

Script Failure Exit Codes

- Scripts can communicate success/failure through exit status and the exit command

```
if [ $? -eq 0 ]; then
        echo "success"
        exit 0
else
        echo "failed: expected  ${MATCHSTR} in ${OUTPUTSTRING} but instead found"
        exit 1
fi
```

```
dan@DESKTOP-BQMVP69:~/CU/assignments-complete-private$ ./finder.sh
ERROR: Invalid Number of Arguments.
Total number of arguments should be 2.
The order of the arguments should be:
   1)File Directory Path.
   2)String to be searched in the specified directory path.
```

https://bash.cyberciti.biz/guide/Exit_command

Course Introduction

Advanced Embedded Linux Development

with **Dan Walkes**

University of Colorado **Boulder**

Learning objectives:

Introduction to the course

Why learn Linux?

What this course is and isn’t

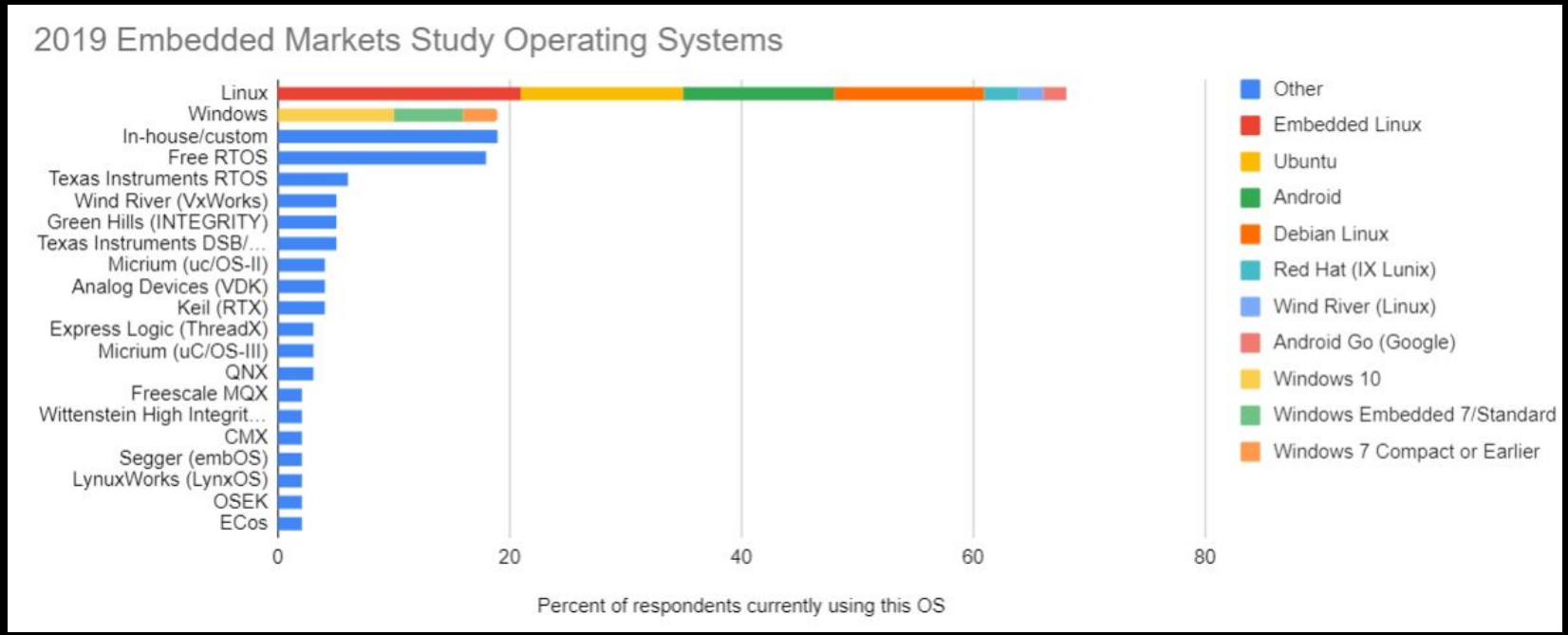
Who Is This Guy?

- Dan Walkes
- Work full time in Industry and teach part time
  - VP of Embedded Engineering at Sighthound
- Began teaching Spring 2019 semester
- Former CU Boulder ESE student in the early 2000s
  - Former TA for ECEN 5623 Real Time Embedded

Who Is This Guy?

- 20th year of Industry experience in 2019
  - Most of the past 10 has been Linux embedded systems related.
- Not a Dr. or Professor. Please call me Dan.
  - Think of me as a team lead on your development team.

Why Learn Linux?



- Based on 958 responses Jan-March 2019

EETimes Embedded Market Study 2019
https://www.embedded.com/wp-content/uploads/2019/11/EETimes_Embedded_2019_Embedded_Markets_Study.pdf

Course Objectives

- Understand how to configure and deploy a Linux based Embedded System.
- Gain experience with common Linux system build tools and components:
  - BASH and shell scripting
  - Buildroot and Yocto
  - QEMU

Course Objectives

- Gain experience with Linux Driver development
- Improve your resume/interview skills in relation to Embedded Linux!
- Cover topics I wish I would have had as a part of my graduate studies.
  - In many cases they didn’t exist yet.

What This Class is NOT

- Not a data structures or algorithms course.
  - We will cover these only from the standpoint of how to use existing libraries/software.
- Not a low resource microcontroller/FreeRTOS course.
  - Not covering event loops or protocols like Bluetooth
  - Not specifically covering sensors
  - Not covering low resource embedded libraries.

Course Format

Advanced Embedded Linux Development

with **Dan Walkes**

University of Colorado **Boulder**

Learning objectives:

Prerequisites and expectations

Assignments and build environments

Books

Prerequisites and Expectations

- I expect some working knowledge of:
  - C Programming
  - Makefiles
- Ideally you'd have some knowledge of:
  - Linux command line
  - Shell scripting

Assignments and Build Environment

- A Linux build environment will be required for all assignments.
  - This can be a virtual machine or actual Linux hardware.
  - See Assignment 1 instructions for version in use.
- Option 1: Use VirtualBox on Windows or MacOS
  - Install a Ubuntu VM with at least 4GB of RAM and 100GB of disk space
  - Requires ~8GB of RAM and 100GB of host disk space
  - Some students use a USB 3.0 HDD/SSD for VM images

Assignments and Build Environment

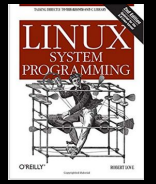
- Option 2: Use a dedicated Linux laptop
  - Find one with at least 8GB of RAM and 100GB of disk space.
  - Doesn’t need to be expensive: think pawn shops and ebay.
  - It’s a good idea to google <hardware model> Ubuntu to see if you can find people stating it’s supported.
- Option 3: Use a cloud hosted runner

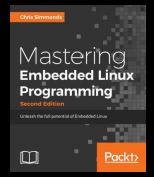
Assignments / Project and git

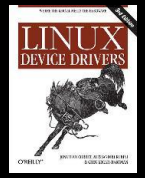
- We will make extensive use of git in this course.
  - You will use git to turn in all assignments through Github Classroom.
- Assignments may start with git merge.
- You may see merge issues/errors. When you do:
  - Use an internet search! Lots of help is available online
  - Use appropriate help channels

Books

- Linux System Programming, 2nd Edition (LSP)
  - How your software programs will interact with the kernel.
  - Processes, Threading, POSIX
- Mastering Embedded Linux Programming, 2nd Edition (MELP)
  - Practical aspects of building Embedded Systems
- Linux Device Drivers 3rd Edition (LDD)
  - Theory and practice related to Linux Device Driver development.







Environment Setup

Advanced Embedded Software Development

with **Dan Walkes**

University of Colorado **Boulder**

Learning objectives:

Introduction to Linux Workshop
Installing a Linux Build VM
VM Snapshots
Distributions and Packages
Ubuntu and root access

Lecture Backup Material

- Introduction to Linux Workshop
  https://nsdl.oercommons.org/courses/an-introduction-to-linux-2/view
  - Shells, Text Editors (vim section), Remote Connections, Filesystem, File Permissions, Optional Topics
- Bash Scripting Tutorials:
  https://linuxconfig.org/bash-scripting-tutorial-for-beginners
  https://ryanstutorials.net/bash-scripting-tutorial/bash-variables.php

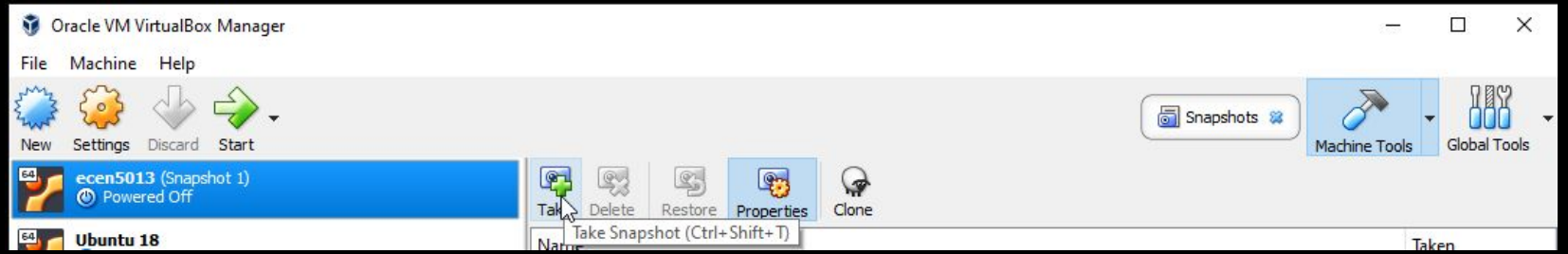
Installing a Linux Build VM

- If you are running Windows or MacOS, we will use VirtualBox for our Linux based build environment.
  - See https://www.virtualbox.org/
- We will use VirtualBox + Ubuntu.
  - See the Environment Setup document for details.

https://www.virtualbox.org/
http://releases.ubuntu.com/

Virtual Machine Snapshots

- Use these in case your VM crashes, to recover your last known working configuration



https://www.virtualbox.org/manual/UserManual.html#snapshots

Using a Physical Linux Machine

- If your laptop is not powerful enough to run VirtualBox you can also run on dedicated Linux hardware.
  - Does not need to be expensive - lots on ebay or in pawn shops for < $200
- You may also consider dual booting

Distributions and Packages

- Linux is available in many distributions.
  - Distributions are collections of preconfigured software maintained by the community or companies.
- Software version and configuration is managed in Packages.
- Ubuntu is a popular Linux distribution.
- Each distribution has its own mechanism of adding software packages through the command line.

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

Packages vs Binaries

- Why doesn’t Linux just use binary installers like Windows does? Why use package management?
  - Packages were essentially the precursor to “app stores”
  - Package managers provide a centralized location for trusted software sources.
  - Also solves fragmentation problems
    - Software packages may need slight customizations for different distributions.

What is Ubuntu?

- Pronounced “uu-boonto”
  - African word meaning “humanity to others”
- First release in 2004
- Based on another distribution known as “Debian”
  - Compared to Debian it’s (arguably) less stable but more up to date.
  - Ubuntu is more focused on usability than licensing.

https://ubuntu.com/about
https://linuxconfig.org/debian-vs-ubuntu

Ubuntu Package Manager

- Ubuntu's package management was traditionally performed by the "apt" utility
- Recent Ubuntu releases also support the "snap" utility
- Ubuntu will often suggest packages

```
ecen5013@ecen5013-VirtualBox:~/lecture2$ tree a_copy_target/

Command 'tree' not found, but can be installed with:

sudo snap install tree  # version 1.8.0+pkg-3fd6, or
sudo apt  install tree

See 'snap info tree' for additional versions.
```

https://help.ubuntu.com/community/AptGet/Howto
https://snapcraft.io/

Linux Root user

- A default user account has access limitations.
  - Prevents you from accidentally deleting/modifying things you didn’t intend.
    - example:
      - rm -rf /${undefined_variable}
- The “root” user can access/modify anything on the system
- Different permission levels allow the system administrator to configure appropriate permissions per user

Ubuntu Root Access Philosophy

- By default your account is able to run commands as root using the “sudo” (short for super user do) command.
- sudo <command name>
  - Asks you for your password
  - Runs the command you specified as root

https://www.howtoforge.com/tutorial/sudo-beginners-guide/
https://searchsecurity.techtarget.com/definition/sudo-superuser-do

System Programming Overview

Advanced Embedded Linux Development

with **Dan Walkes**

University of Colorado **Boulder**

Learning objectives:

Introduce System Programming

Understand APIs and ABIs

Introduce POSIX

System Programming Overview

- What is System Software?
  - Interfacing with the kernel and C library.
  - GUI, compiler, debugger, web server, database.
- Differences relative to “Application Software”:
  - Doesn’t use higher level libraries.
  - Less OS/hardware details abstracted.

Cornerstones of System Programming

- System Calls (syscalls)
- C Library (libc)
- C Compiler/linker (toolchain)

Cornerstones of System Programming

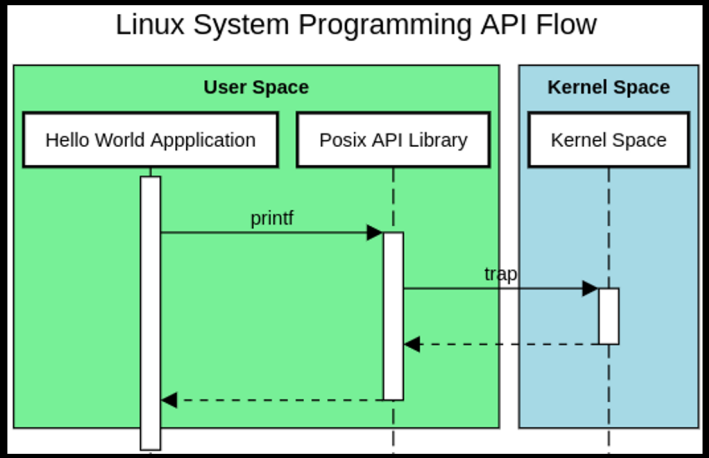
- System Calls (syscalls)
  - Kernel function invocations from user space (read() write()) Roughly 300 total.
  - Shared subset of ~90% implemented by all architectures on Linux.

Cornerstones of System Programming

- C Library (libc)
  - GNU libc (glibc) (gee-lib-see) Wrappers for system calls, threading support and applications
- C Compiler/linker (toolchain)
  - GCC (GNU C compiler)

Invoking System Calls

- Not possible to link your user space application with the Linux kernel.
  - Why not?
    - Not allowed for security/reliability reasons
- How do you invoke system calls?
  - Use a “trap” - typically a software interrupt



Linux System Programming, Robert Love Chapter 1

https://medium.com/the-tech-house/dual-mode-multimode-operation-bcdc93077f7a

API

- Application Programming Interface.
- API is a definition.
- Software which provides an API is the implementation.
- Ensures **source** compatibility.
  - Source can be compiled for different platforms, works in a specific way.

API

- Source code remains portable across different hardware platforms/revisions (ideally)
- Example: C library functions like printf(), strcpy(), etc.
- Exceptions: syscall ~10% architecture differences mentioned in previous slide

ABI

- Application Binary Interface
- Calling conventions, byte ordering, register use
- Ensures **binary** compatibility
- Defined/implemented by Kernel and toolchain
- Defines application interaction with itself, libraries, the kernel.

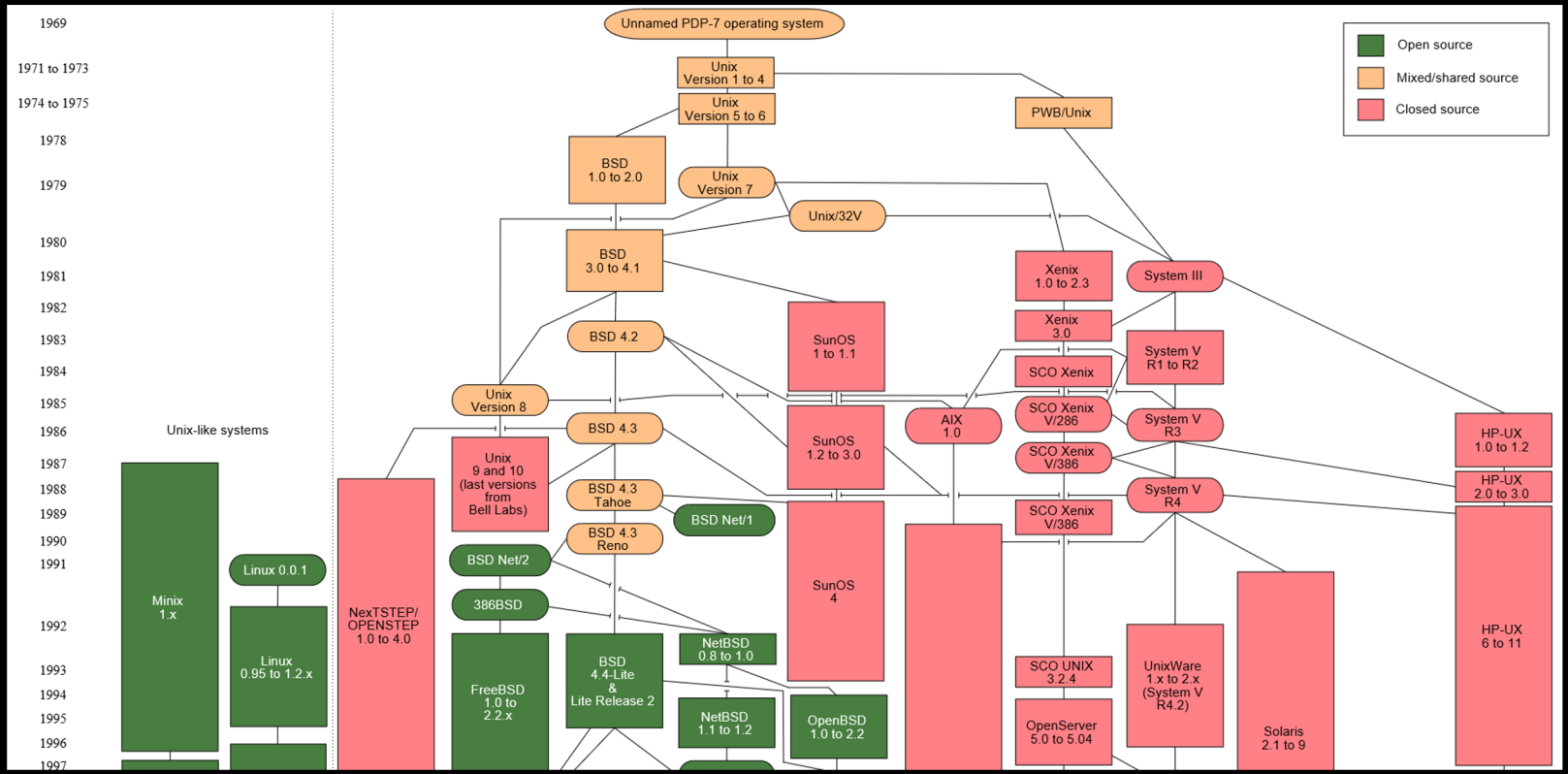
ABI

- Binary generated after compilation/link (by toolchain).
- Byte code specific hardware types, software/compiler/library revisions.
  Requires match:
  - hardware
  - software libraries (especially glibc)
  - compiler

POSIX

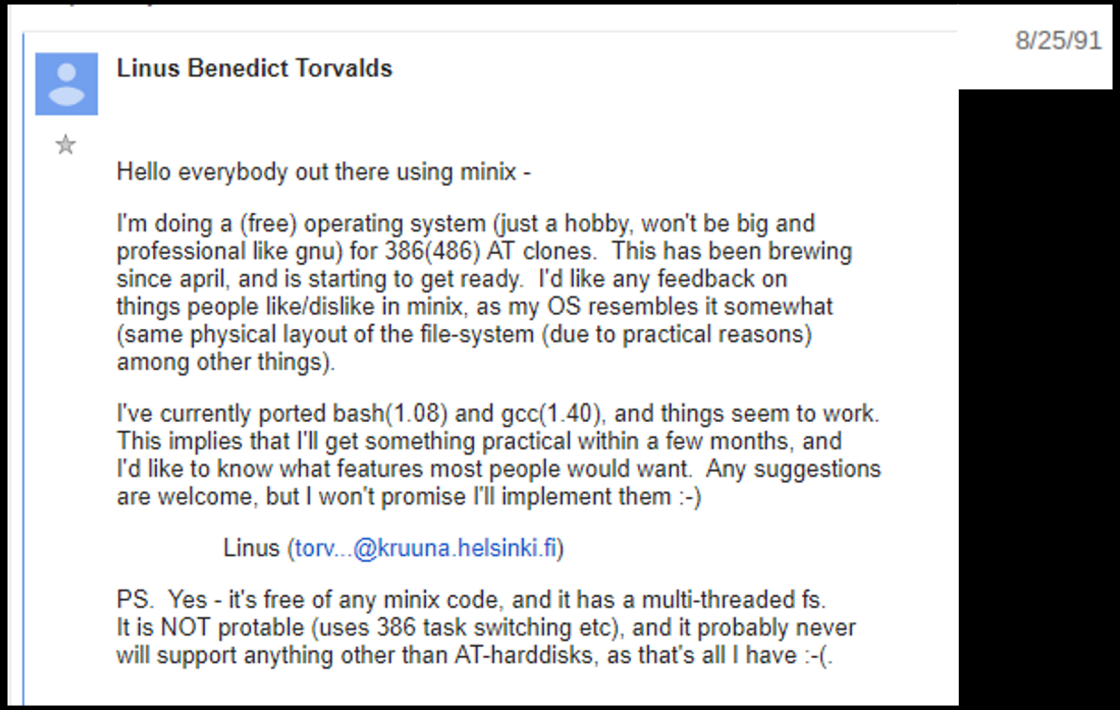
- C APIs for Unix-like operating systems
- Stands for Portable Operating System Interface
  - Pronounced “pahz-icks”
- Started by IEEE in the late 1980s, as a way to coalesce the “Unix Wars”

UNIX Wars



By Eraserhead1, Infinity0, Sav_vas - Levenez Unix History Diagram, Information on the history of IBM's AIX on ibm.com, CC BY-SA 3.0,

Enter Linux



8/25/91

Linus Benedict Torvalds

Hello everybody out there using minix -

I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486) AT clones. This has been brewing since april, and is starting to get ready. I'd like any feedback on things people like/dislike in minix, as my OS resembles it somewhat (same physical layout of the file-system (due to practical reasons) among other things).

I've currently ported bash(1.08) and gcc(1.40), and things seem to work. This implies that I'll get something practical within a few months, and I'd like to know what features most people would want. Any suggestions are welcome, but I won't promise I'll implement them :-)

Linus (torv...@kruuna.helsinki.fi)

PS. Yes - it's free of any minix code, and it has a multi-threaded fs. It is NOT protable (uses 386 task switching etc), and it probably never will support anything other than AT-harddisks, as that's all I have :-(.

https://groups.google.com/forum/#!msg/comp.os.minix/dlNtH7RRrGA/SwRavCzVE7gJ

POSIX

- Open Software Foundation created the Single Unix Specification (SUS)
  - Specification was free
  - Eventually incorporated the POSIX standard.

POSIX

- POSIX defines C code API for many concepts we'll cover this semester
  - File and directory operations
  - Clocks and timers
  - Semaphores
  - Shared memory
  - Threads

POSIX

- Official list of headers: http://pubs.opengroup.org/onlinepubs/9699919799/idx/head.html
  - Some familiar ones:
  - <stdio.h>
  - <stdint.h>

Linux Filesystems

Advanced Embedded Linux Development

with **Dan Walkes**

University of Colorado **Boulder**

Learning objectives:

Understand use of files in Linux

Introduction to Inodes

Introduction to Links

Linux filesystems

“Everything Is A File”

- Much of the interaction with the kernel occurs via reading/writing files.
- Devices are accessed similar to files
  - Example /dev/ttyUSB0 for USB serial port.
- Common open, manipulate, close paradigm shared with files and devices.

Files in Linux

- Represented by paths.
  - Absolute “/path/to/file” - from root of filesystem.
  - Relative “path/to/file” - from working directory.
    - No leading “/” = relative path.

Linux Regular File Properties

- Bytes of data in a byte stream.
  - No structure enforced at the system level.
- File position/offset tracks location.
- Can be truncated to size smaller or larger than original.

Linux Regular File Properties

- Can be opened more than once, by different or same processes.
- Referenced and accessed by inode in the filesystem.

Linux Inode (Index Node)

- Metadata used to track files on disk.
- Includes timestamp, owner, size, mode (access permission), location.
- Fixed small (128 byte) size.
  - Filename is not included.

Inode					
Accessed Time	Size	UID	GID	Permission	Location On Disk

https://premaseem.wordpress.com/2016/02/14/what-is-inode-in-linux-unit/

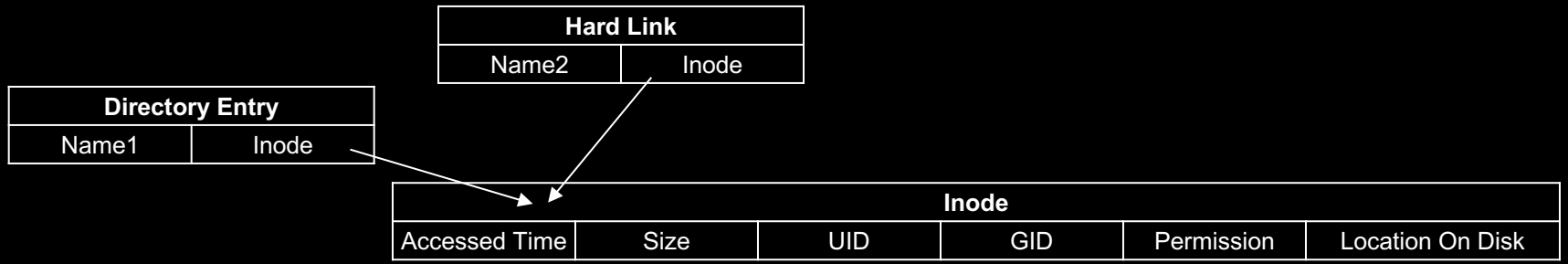
Linux Directories

- Map human readable names to Inode numbers.
- Actually just files (with their own inodes) containing mapping of names to Inodes.
- Start at the root directory (“/”)

Linux System Programming, Robert Love Chapter 1
https://premaseem.wordpress.com/2016/02/14/what-is-inode-in-linux-unit/

Linux Directories

- Why not include filenames in Inodes?
  - Allows different file names to share the same content without duplicating Inode content.



Linux System Programming, Robert Love Chapter 1
https://premaseem.wordpress.com/2016/02/14/what-is-inode-in-linux-unit/

Linux Links

- Flexibility of Inode design means multiple names can resolve to the same Inode.
- Links redirect two file/directory paths to the same Inode

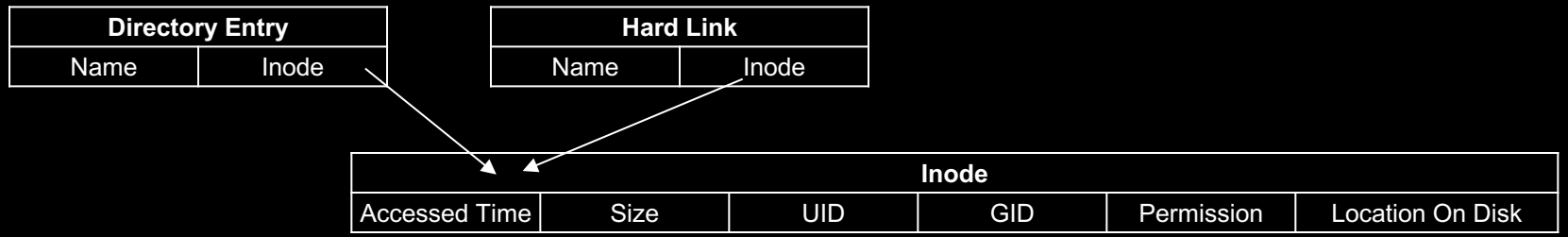
```
user@myhost:~/myfolder$ ls -l
total 4
-rw-r--r-- 1 user user 29 May  8 17:09 myfile.txt
lrwxrwxrwx 1 user user 10 May  9 05:55 mylink -> myfile.txt
```

Linux Links

- Two types of Links, Hard and Symbolic (symlinks)
  - Hard Links map directly to inodes, only allowed on the same filesystem.
  - Soft links map to filenames, work across filesystems, can be broken.

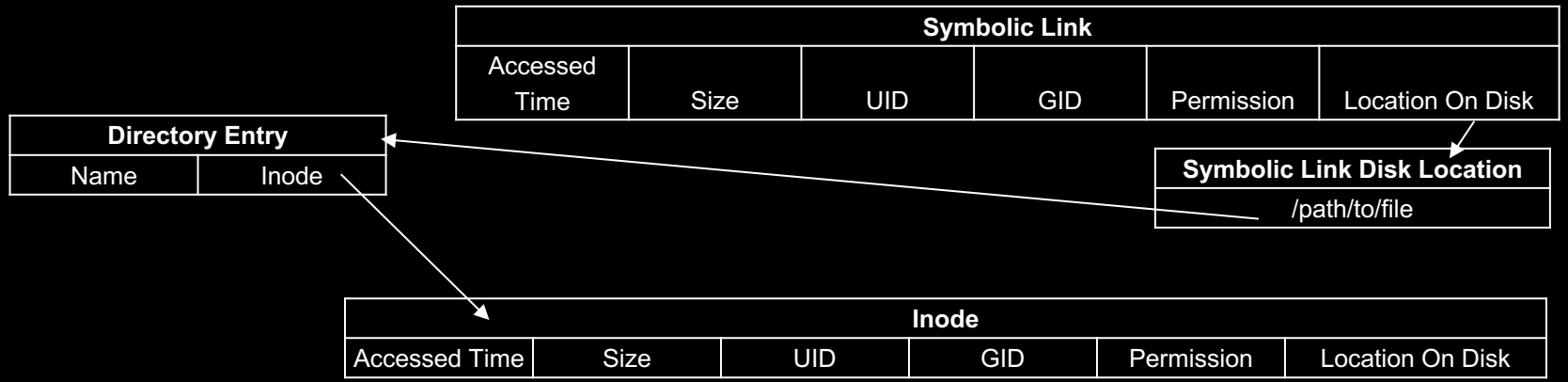
Hard Links

- Can’t span filesystems (Inode references a specific filesystem.)
- Conceptually the same as a Directory Entry.
- File deletes aren’t allowed until all references are deleted (preventing broken Hard Links).



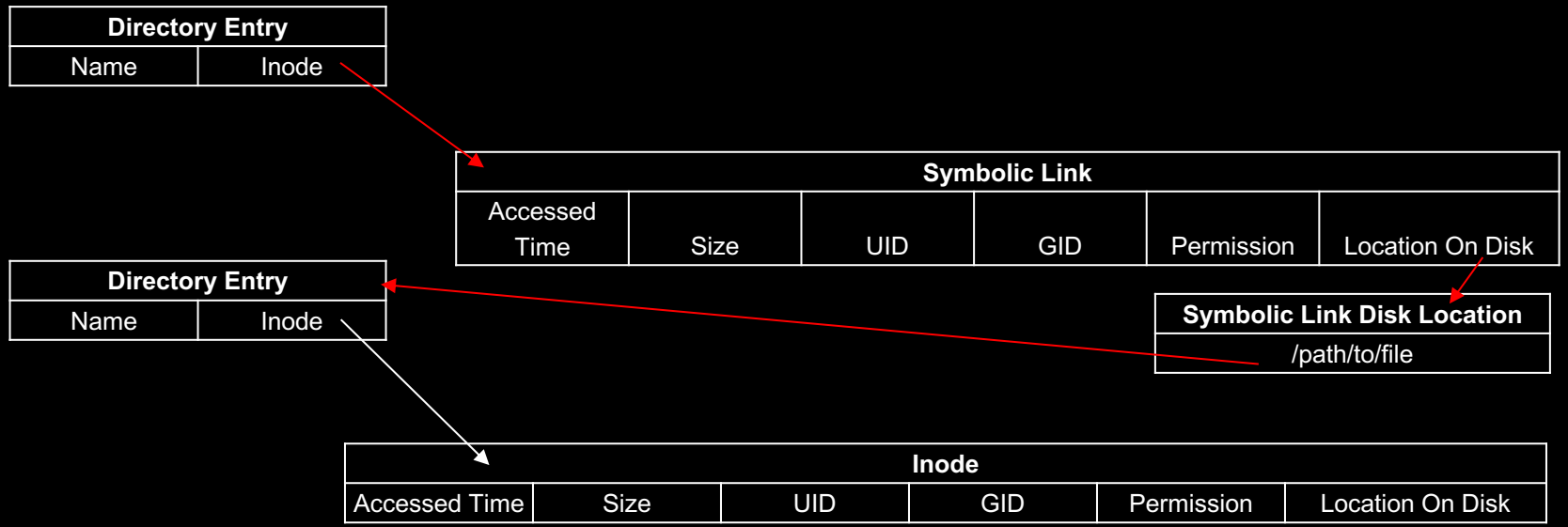
Symbolic Links (symlinks)

- Regular file with complete path in content
- Can point anywhere, including other filesystems, can break



Symbolic Links

- Requires extra steps to resolve



Linux Special Files

- Map hardware devices to the “everything is a file” paradigm.
- Kernel objects represented as files.
  - Character Device
  - Block Device
  - Named Pipes
  - Sockets

Linux Special Files

- Character Device
  - Linear queue of bytes
  - Example: Keyboard
- Block Device
  - Array of bytes, addressable in a sector
  - IE Hard Disk
- Named Pipes/Sockets
  - Interprocess Communication

Linux Filesystem

- Collection of files in a hierarchy
- Specific types supported, tied to storage types
  - NFS - network file storage
  - ext4 - block device storage
  - fat - Microsoft defined storage format for disks

Linux Filesystem

- Mounted/Unmounted to add/remove from the root filesystem.
- Smallest unit addressable is a block
  - block is a power of two multiple of sector size
  - Typical/historical sector size is 512 bytes

Linux Blocks/Sectors

- Consider a 1TB Filesystem.
  - 1,000,000,000,000 bytes.
  - 0x3B98CCA00 bytes.
  - 34 bits required to address individually.

Linux Blocks/Sectors

- 34 bits to address each byte.
- With 512 byte blocks:
  - Only 1,953,125,000 (0x746a5288) blocks.
  - 30 bits required to access each block.
- With 4K blocks:
  - Only 244,140,625 (0xE8D4A51) blocks.
  - 28 bits required to access each block.

Processes and Threads

Advanced Embedded Linux Development

with **Dan Walkes**

University of Colorado **Boulder**

Learning objectives:

Understand Linux Processes

Understand Linux Threads

Introduce Interprocess Communication (IPC)

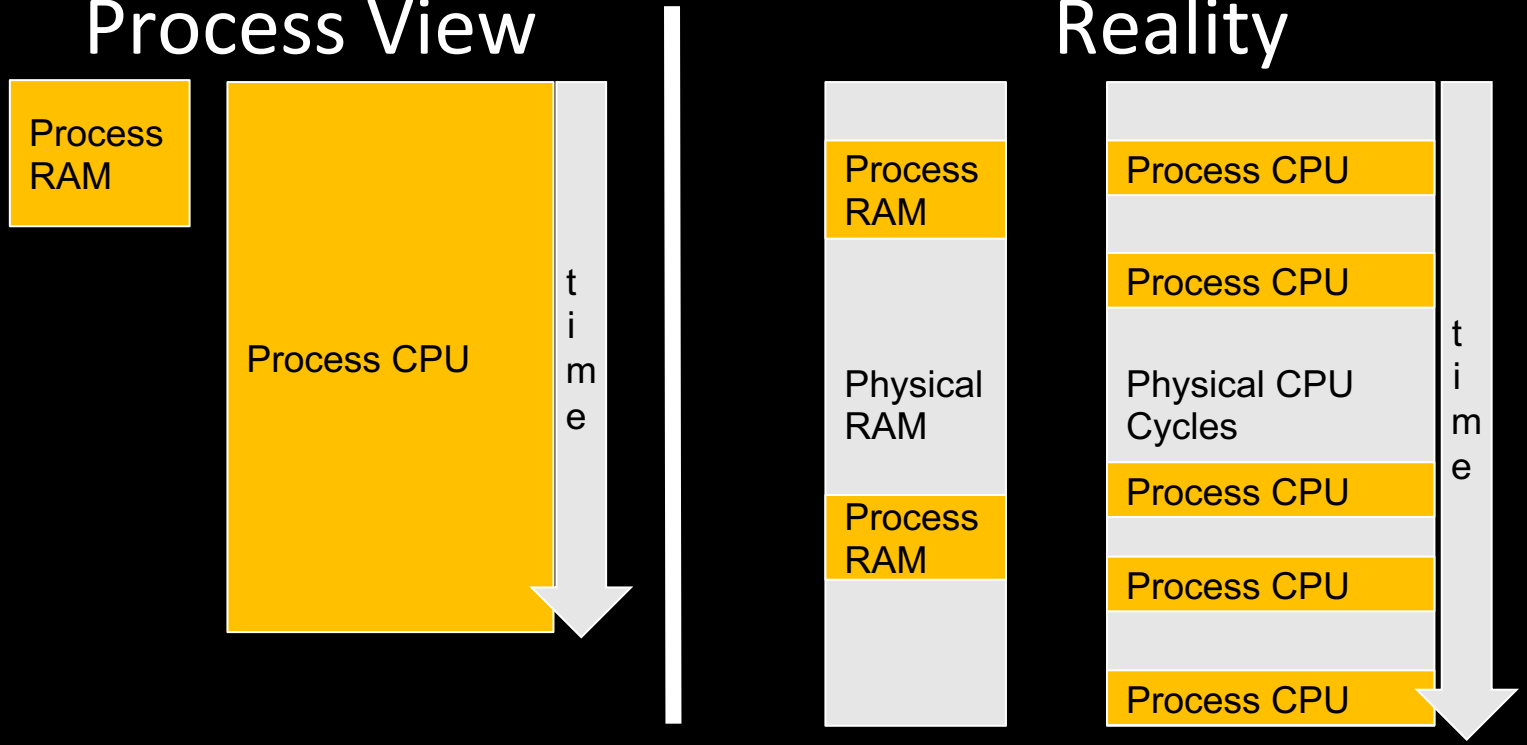
Linux Processes

- Executable object code, running on hardware.
  - ELF (Executable and Linkable Format).
- Necessary resources to run are allocated and managed by the kernel through system calls.
  - Timers
  - Files
  - Hardware access

Linux Processes

- Referenced by Process ID (pid).
- Run on virtualized processor and virtualized view of memory.
  - Appears to software as its own dedicated RAM/CPU.

Virtualized Processor/Memory



Linux Threads

- Unit of activity within a process
- A process may be single-threaded or multithreaded
- Each thread has
  - Stack - stores local variables
  - Processor state/current location
- Memory address space is shared between threads

Linux Threads/Processes and Memory

- Each Process has its own virtual memory.
- Each Thread shares process virtual memory.
- Sharing memory access between threads?
  - Access directly (use synchronization).
- Sharing memory access between processes?
  - Use Inter-Process Communication (IPC).

Linux Signals

- One-way asynchronous notifications sent from:
  - Kernel to process.
  - One process to another process (IPC).
  - A process to itself
- Processes setup signal handlers to control how to respond to a signal
  - Example Ctrl->C or SIGINT to stop a process.

Linux Signals

- Signal handlers must use signal safe functions which are safe to call asynchronously.
  - Use of global variables can introduce unsafe scenarios.
  - Process code can be interrupted at any time by signals.

Linux Interprocess Communication (IPC)

Allows process to exchange information without using a common global memory space.

- Pipes
- Semaphores
- Message Queues
- Shared Memory

Users and Groups

Advanced Embedded Linux Development

with **Dan Walkes**

University of Colorado **Boulder**

Learning objectives:

Understand Linux Users and Groups

Understand Linux Permissions

Linux Users and Groups

- User logs in with a username and password
  - Authentication
- Each user is associated with a User ID (uid)
  - Each process is associated with the UID of the user running the process.
  - /etc/passwd maps usernames uids

Linux Users and Groups

- UID 0 is the root user
  - Can do almost anything on the system
- Each user belongs to one or more groups, with corresponding Group IDs (gid)
  - /etc/group maps group names to gids

Linux Permissions

Associates a file with owning user, owning group, and permission bits

	User (owner)			Group			Others (everyone)		
	Read (r)	Write (w)	Execute (x)	Read (r)	Write (w)	Execute (x)	Read (r)	Write (w)	Execute (x)
Bit	8	7	6	5	4	3	2	1	0
Octal Value	0400	0200	0100	040	020	010	04	02	01

```
dan@DESKTOP-BQMVP69:~/CU/aesd-lectures$ ls -l
total 24
-rw-r--r-- 1 dan dan 1083 Jun 21 15:27 LICENSE
-rw-r--r-- 1 dan dan  149 Jun 21 15:27 README.md
drwxr-xr-x 4 dan dan 4096 Jun 21 17:08 lecture2
drwxr-xr-x 2 dan dan 4096 Jun 21 15:27 lecture5
drwxr-xr-x 2 dan dan 4096 Jun 21 15:27 lecture7
drwxr-xr-x 4 dan dan 4096 Jun 21 15:27 lecture9
```

Linux System Programming, Robert Love Chapter 1

System Programming and Error Handling

Advanced Embedded Linux Development

with **Dan Walkes**

University of Colorado **Boulder**

Learning objectives:

Understand errno.

Understand error handling strategies for System Programming.

Errno and error handling

- C Library mechanism for reporting errors is errno <errno.h>

```
FOPEN(3)                                                    Linux Programmer's Manual

NAME
       fopen, fdopen, freopen - stream open functions

DESCRIPTION
       The fopen() function opens the file whose name is the string pointed to by pathname and associates a stream with it.

RETURN VALUE
       Upon successful completion fopen(), fdopen() and freopen() return a
       FILE  pointer.   Otherwise,  NULL  is  returned and errno is set to
       indicate the error.
```

http://www.cplusplus.com/reference/cstdio/fopen/

Errno and error handling

- See https://pubs.opengroup.org/onlinepubs/009695399/basedefs/errno.h.html for a list of POSIX defined errors - ENOENT means “No such file or directory”
- Use `errno -l` from the command line to dump error values and names

```
dan@DESKTOP-BQMVP69:~/CU/aesd-lectures/lecture2$ errno -l
EPERM 1 Operation not permitted
ENOENT 2 No such file or directory
ESRCH 3 No such process
EINTR 4 Interrupted system call
EIO 5 Input/output error
ENXIO 6 No such device or address
E2BIG 7 Argument list too long
ENOEXEC 8 Exec format error
```

Errno and error handling

```c
int main () {
   const char *filename = "non-existing-file.txt";
   FILE *file = fopen (filename, "rb");
   if (file == NULL) {
        fprintf(stderr, "Value of errno attempting to open file %s: %d\n", filename, errno);
        perror("perror returned");
        fprintf(stderr, "Error opening file %s: %s\n",filename, strerror( errno ));
   } else {
        fclose(file);
   }
   return 0;
}
```

```
Value of errno attempting to open file non-existing-file.txt: 2
perror returned: No such file or directory
Error opening file non-existing-file.txt: No such file or directory
```

Errno and error handling

```
PERROR(3)                Linux Programmer's Manual                PERROR(3)

NAME
       perror - print a system error message

DESCRIPTION
       The perror() function produces a message on standard error describ-
       ing the last error encountered during a call to a system or library
       function.
```

```c
int main () {
   const char *file1name = "first-non-existing-file.txt";
   FILE *file1 = fopen (file1name, "rb");
   const char *filename = "non-existing-file.txt";
   FILE *file = fopen (filename, "rb");
   if (file == NULL) {
        if( file1 ) {
           fclose(file1);
        }
        fprintf(stderr, "Value of errno attempting to open file %s: %d\n", filename, errno);
        perror("perror returned");
        fprintf(stderr, "Error opening file %s: %s\n",filename, strerror( errno ));
   } else {
        fclose(file);
   }
   return 0;
}
```

- What's wrong with this code?
  - fprintf may modify errno

Errno and error handling

- How could you fix it?
  - Move fprintf after perror
  - Save errno to local var

```
int main () {
   const char *file1name = "first-non-existing-file.txt";
   FILE *file1 = fopen (file1name, "rb");
   const char *filename = "non-existing-file.txt";
   FILE *file = fopen (filename, "rb");
   if (file == NULL) {
        if( file1 ) {
           fclose(file1);
        }
        fprintf(stderr, "Value of errno attempting to open file %s: %d\n", filename, errno);
        perror("perror returned");
        fprintf(stderr, "Error opening file %s: %s\n",filename, strerror( errno ));
   } else {
        fclose(file);
   }
   return 0;
}
```

Errno and error handling

- Given threads use the same memory space
- Wouldn't an error in one thread override an error in a second thread?
  - This is handled for us by POSIX

Redefinition of errno

In POSIX.1, *errno* is defined as an external global variable. But this definition is unacceptable in a multithreaded environment, because its use can result in nondeterministic results. The problem is that two or more threads can encounter errors, all causing the same errno to be set. Under these circumstances, a thread might end up checking errno after it has already been updated by another thread.

To circumvent the resulting nondeterminism, POSIX.1c redefines errno as a service that can access the per-thread error number as follows (ISO/IEC 9945:1-1996, §2.4):

> Some functions may provide the error number in a variable accessed through the symbol errno. The symbol errno is defined by including the header *<errno.h>*, as specified by the C Standard ... For each thread of a process, the value of errno shall not be affected by function calls or assignments to errno by other threads.

http://www.unix.org/whitepapers/reentrant.html

Embedded Linux Toolchain Overview

Advanced Embedded Linux Development

with **Dan Walkes**

University of Colorado **Boulder**

Learning objectives:

Understand the components of an Embedded Linux toolchain.

Understand how to setup an Embedded Linux toolchain.

Understand the toolchain sysroot.

Toolchains

- Compiler
- Linker
- Run-time libraries
- GCC or Clang are the most likely toolchain options.

GCC Toolchain Components

- Binutils - binary utilities including assembler and linker.
- GCC (Gnu Compiler Collection)
  - Compilers for programming languages (C in our case.)
- C Library - API based on POSIX definition.

Setting Up a Toolchain

- Option 1: Do it manually by downloading/building/installing components yourself.
- Option 2: Use a build system to generate (for instance Buildroot or Yocto.)

Setting Up a Toolchain

- Why hand download/build your own Toolchain?
  - You probably shouldn't.
  - We are going to use this with Assignment 2 just to understand how it can be done.
    - Demystify the process, help with build system troubleshooting later.

Types of Toolchains

- Native toolchain
  - Runs on the same system as the program it generates.
- Cross toolchain
  - Runs on a different architecture than the host.
  - “cross compiling”
    - Creating output for a different hardware architecture.

Types of Toolchains

- Why use a cross toolchain?
  - Your host is probably more powerful than the target, builds are faster.
  - You probably don't want to include development tools in your target image.
    - Might not be possible to fit these in your target image.

Specifying Toolchain Targets

- For GNU, a prefix specifies the toolchain target
- Example for QEMU: aarch64-none-linux-gnu-gcc
  - CPU is ARM 64 bit
  - Vendor is “none” (support a common set of ARM CPUs)
  - Kernel is “linux”
  - Operating system is GNU GCC

Building a toolchain with ARM cross compiler

- See toolchain install instructions at https://github.com/cu-ecen-aeld/aesd-assignments/wiki/Installing-an-ARM-aarch64-developer-toolchain

Example Cross Compile Steps

```
yocto@yocto-Latitude-E6540:~/aesd/assignment-3-dwalkes-1/finder-app$ export PATH=$PATH:/usr/local/arm-cross-compiler/install/gcc-arm-
10.2-2020.11-x86_64-aarch64-none-linux-gnu/bin
yocto@yocto-Latitude-E6540:~/aesd/assignment-3-dwalkes-1/finder-app$ aarch64-none-linux-gnu-gcc -g -Wall   -c -o writer.o writer.c
yocto@yocto-Latitude-E6540:~/aesd/assignment-3-dwalkes-1/finder-app$ file writer.o
writer.o: ELF 64-bit LSB relocatable, ARM aarch64, version 1 (SYSV), with debug_info, not stripped
```

Sysroot, library and header files

```
yocto@yocto-Latitude-E6540:~/aesd/assignment-3-dwalkes-1/finder-app$ aarch64-none-linux-gnu-gcc -print-sysroot
/usr/local/arm-cross-compiler/install/gcc-arm-10.2-2020.11-x86_64-aarch64-none-linux-gnu/bin/../aarch64-none-linux-gnu/libc
```

- Sysroot is the root filesystem of your (cross) toolchain.
- Consists of files specific to the *target* type.
  - Mirrors files on your host root filesystem.
- Some files are needed to compile programs.
- Others are (also) needed on the target at runtime.

Sysroot Directories

- lib - Shared objects for C library (on target.)
- usr/lib - Static library archive files for the C library.
- usr/include - Headers for libraries (for instance <stdio.h>.)
- usr/(s)bin: Utility programs for the cross toolchain.

Sysroot library files

- What do we mean by runtime target files?

```
yocto@yocto-Latitude-E6540:~/aesd/assignment-3-dwalkes-1/finder-app$ file /usr/local/arm-cross-compiler/install/gcc-arm-10.2-2020.11-x86_64-aarch64-none-linux-gnu/bin/../aarch64-none-linux-gnu/libc/lib64/ld-2.31.so
/usr/local/arm-cross-compiler/install/gcc-arm-10.2-2020.11-x86_64-aarch64-none-linux-gnu/bin/../aarch64-none-linux-gnu/libc/lib64/ld-2.31.so: ELF 64-bit LSB shared object, ARM aarch64, version 1 (SYSV), dynamically linked, with debug_info, not stripped
```

- sysroot/lib contains library files specific to the toolchain which will be installed on the target (in this case aarch64) even when built on a different architecture.

Toolchain Sysroot files

- Will the aarch64-none-linux-gnu-gcc compiler be in the toolchain sysroot?
  - No, because we are cross compiling
  - The aarch64-none-linux-gnu-gcc compiler is in your host system filesystem (and run by your host system), not your toolchain system root

Other tools in the toolchain

- Use all cross toolchain components with the same prefix referenced for gcc (aarch64-none-linux-gnu-XXXX)
- gcc, g++ - compiler
- gdb - debugger
- ld - linker

Other tools in the toolchain

- addr2line : Converts program addresses into filenames/numbers for debug.
- objdump - Disassemble object files.
- strip - Remove debug tables, make binary files smaller.
- readelf - Additional information about executables (object code, location in memory map, etc).

Static vs Dynamic Linking

- gcc or g++ always links with glibc, the C library
- Static Linkage
  - All library functions and dependencies are pulled from archive and placed in your executable
- Dynamic Linkage
  - Linking is done dynamically at runtime

Static vs Dynamic Linking

- When to use Static Linkage?
  - When you have relatively few applications (or only a single application)
    - Busybox
  - You need to run an application before the root filesystem is available
    - When would that happen?
      - At boot, loading storage drivers.

Shared Libraries (Dynamic Link)

```
yocto@yocto-Latitude-E6540:~/aesd/assignment-3-dwalkes-1/finder-app$ make CROSS_COMPILE=aarch64-none-linux-gnu- clean
rm -f *.o writer *.elf *.map
yocto@yocto-Latitude-E6540:~/aesd/assignment-3-dwalkes-1/finder-app$ make CROSS_COMPILE=aarch64-none-linux-gnu- all
aarch64-none-linux-gnu-gcc -g -Wall   -c -o writer.o writer.c
aarch64-none-linux-gnu-gcc -g -Wall -I/ writer.o -o writer
yocto@yocto-Latitude-E6540:~/aesd/assignment-3-dwalkes-1/finder-app$ aarch64-none-linux-gnu-readelf -a writer | grep "Shared library"
 0x0000000000000001 (NEEDED)             Shared library: [libc.so.6]
yocto@yocto-Latitude-E6540:~/aesd/assignment-3-dwalkes-1/finder-app$
```

```
yocto@yocto-Latitude-E6540:~/aesd/assignment-3-dwalkes-1/finder-app$ aarch64-none-linux-gnu-readelf -a writer | grep "program interpreter"
      [Requesting program interpreter: /lib/ld-linux-aarch64.so.1]
```

- Why is libc/ld-linux-aarch64 listed here?
  - This is a shared libraries referenced by “writer”
- What does it mean that libc is listed here?
  - This file need to be available on the root filesystem to run “writer” successfully.

Shared Library Locations

- Linker checks for shared libraries in:
  - /lib, /lib64
  - /usr/lib, /usr/lib64
  - content of LD_LIBRARY_PATH

Logging and Syslog

Advanced Embedded Linux Development

with **Dan Walkes**

University of Colorado **Boulder**

Learning objectives:
Logging and Syslog Overview.

Printf Debugging

- Print output is one of the simplest ways to troubleshoot and debug your application.
- The printf call works great for interactive programs.
- What about headless embedded systems?

https://linux.die.net/man/8/syslogd

Logging

- What about headless embedded systems?
  - Need some type of logging framework to store information in log files.
  - Syslog and syslogd (rsyslogd) is one such framework.

https://linux.die.net/man/8/syslogd

Logging and Syslog

- syslogd is a daemon which
  - Uses configuration file to configure logging (usually writing to files in /var/log)
  - Handles log messages from applications using syslog() API calls.

https://linux.die.net/man/8/syslogd

Syslog Usage

```
SYNOPSIS
       #include <syslog.h>

       void openlog(const char *ident, int option, int facility);
       void syslog(int priority, const char *format, ...);
```

- Facilities
  - LOG_USER
  - LOG_DAEMON
  - LOG_LOCAL0 to LOG_LOCAL7
- Priorities
  - LOG_ERR
  - LOG_WARNING
  - LOG_INFO
  - LOG_DEBUG

https://linux.die.net/man/8/syslogd
https://linux.die.net/man/3/syslog

Syslog Usage

```
openlog(NULL,0,LOG_USER);
```

```
syslog(LOG_ERR,"Invalid Number of arguments: %d",argc);
```

- Results in message in /var/log/syslog:

```
Sep  2 08:36:01 ecen5013-VirtualBox anacron[825]: Normal exit (3 jobs run)
Sep  2 08:45:41 ecen5013-VirtualBox writer: Invalid Number of arguments: 1
```

https://linux.die.net/man/8/syslogd
https://linux.die.net/man/3/syslog

Syslog Usage

- Log redirection in an Ubuntu VM happens based on rules in /etc/rsyslog.d/*default.conf

```
#  Default rules for rsyslog.
#
#                       For more information see rsyslog.conf(5) and /etc/rsyslog.conf

#
# First some standard log files.  Log by facility.
#
auth,authpriv.*                 /var/log/auth.log
*.*;auth,authpriv.none          -/var/log/syslog
#cron.*                         /var/log/cron.log
#daemon.*                       -/var/log/daemon.log
kern.*                          -/var/log/kern.log
#lpr.*                          -/var/log/lpr.log
mail.*                          -/var/log/mail.log
#user.*                         -/var/log/user.log

~
```

https://linux.die.net/man/8/syslogd
https://linux.die.net/man/3/syslog