

# 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:~/aesc/assignment-3-dwalkes-1/finder-app$ export PATH=$PATH:/usr/local/arm-cross-compiler/install/gcc-arm-10.2-2020.11-x86_64-aarch64-none-linux-gnu/bin
yocto@yocto-Latitude-E6540:~/aesc/assignment-3-dwalkes-1/finder-app$ aarch64-none-linux-gnu-gcc -g -Wall -c -o writer.o writer.c
yocto@yocto-Latitude-E6540:~/aesc/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:~/aesc/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:~/aesp/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