Embedded Linux

Author : Yasser JEMLI

Dive into the world of embedded Linux

In this presentation, we will delve into the realm of embedded Linux, exploring several fundamental concepts. These concepts include:

1. Understanding embedded Linux: We will explore the definition and characteristics of embedded Linux.
2. Decoding cross compilation: We will explain the significance of cross compilation and what it entails.
3. Kernel basics: We will examine the role of the kernel, its purpose, and its status as an open-source entity.
4. Modifying the kernel: We will address the feasibility and procedure of obtaining the kernel and incorporating a module or syscall.
5. Introducing Buildroot and Yocto projects: We will shed light on the functionalities and features of Buildroot and Yocto projects.

Additionally, we will provide insights on building a custom Linux image, as well as creating a specific image tailored for a particular platform like the Raspberry Pi.

To ensure comprehensive coverage of these questions and concepts, we have structured the presentation into five labs. Each lab will focus on a distinct topic, guiding you through the practical implementation and highlighting key considerations.

Enjoy the presentation and the valuable information it has to offer!

"Linux is not just an operating system, it's a philosophy of openness, collaboration, and limitless possibilities."

TABLE OF CONTENTS

Linus Torvalds

"The kernel is really the core of the operating system, and the rest of the system is built around it. Everything else is just application software."- Linus Torvalds

01

Cross-Compiler

You can describe the topic of the section here

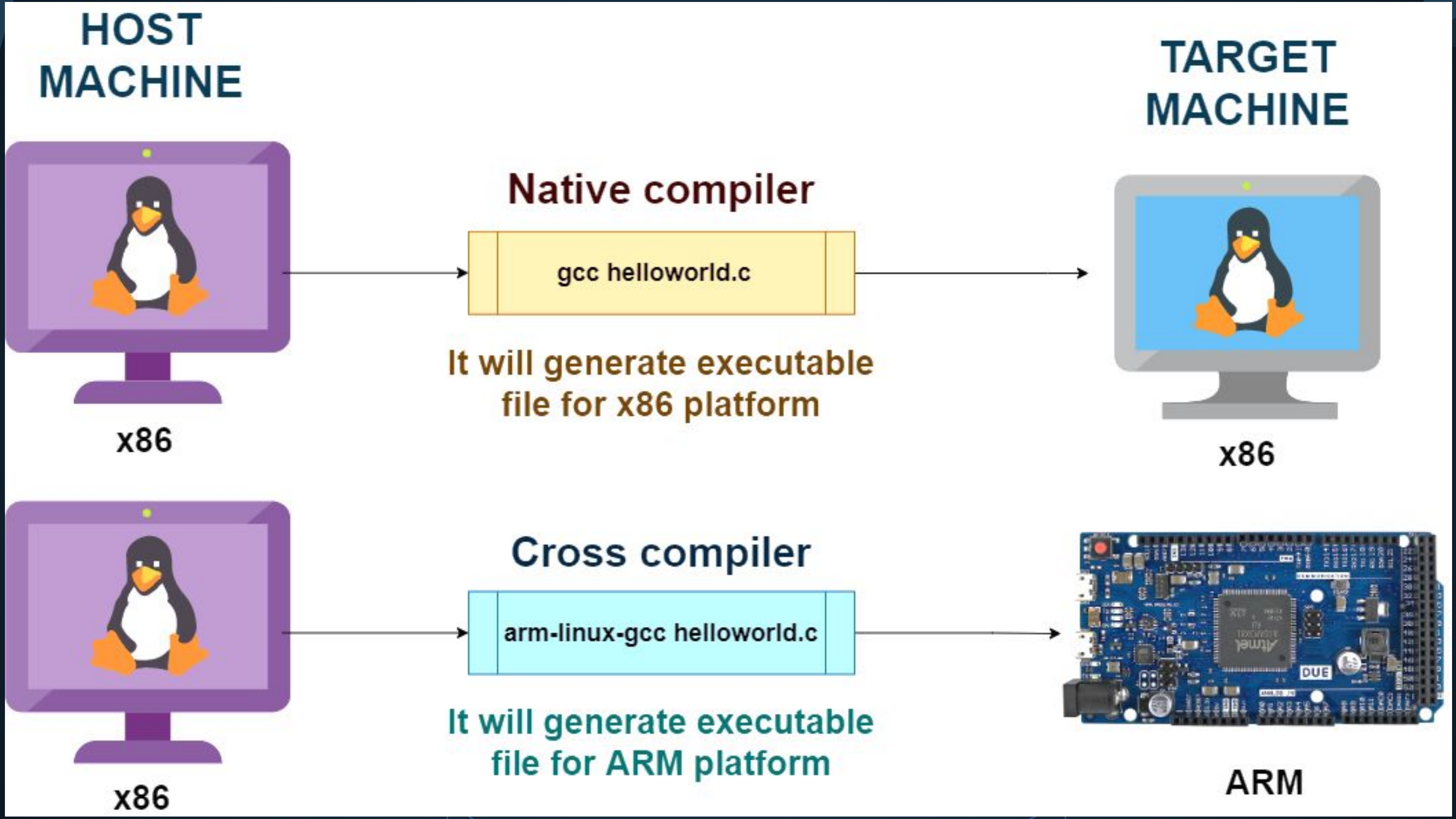
Let’s see what Cross-compiling means

Cross-compiling refers to the process of compiling code on one platform (the host system) in order to generate executable binaries that can run on a different platform (the target system). In other words, it involves developing or building software on one machine for it to be executed on another machine with a different architecture, operating system, or environment.

Typically, cross-compiling is used when the development environment and the target platform differ significantly. For example, you might develop software on a powerful desktop computer (host system) but intend to run it on an embedded device or a different architecture (target system) like an ARM-based microcontroller or a mobile device.

Cross-compiling involves setting up a suitable toolchain that includes a compiler, linker, and necessary libraries specific to the target platform. The toolchain enables the host system to understand and generate binaries compatible with the target system. By using cross-compilation, developers can save time and resources by avoiding the need to compile code directly on the target system.

Overall, cross-compiling is a valuable technique for software development, particularly when working with embedded systems, different architectures, or resource-constrained devices.


HOST MACHINE
TARGET MACHINE
Native compiler
gcc helloworld.c
It will generate executable file for x86 platform
x86
x86
Cross compiler
arm-linux-gcc helloworld.c
It will generate executable file for ARM platform
x86
ARM

To Simplify

Host Machine

Processeur : i9
RAM : 32 Go

Cross-Compiler

arm-gcc

Target Machine

Processeur : ARM V8
RAM : 2 GO

Let’s Start With the First Lab

Now, in this Lab, we will explore an overview of the process involved in installing an ARM cross compiler. This compiler is essential for building software that specifically targets ARM-based embedded systems.

The Lab :

- sudo apt-get update
- sudo apt-get install build-essential gcc-arm-linux-gnueabihf
- Create a SImple C code and save the file under the name “hello.c”
- arm-linux-gnueabihf-gcc hello.c -o helloarm
- sudo apt-get install qemu-user
- qemu-arm helloarm
- sudo ln -s /usr/arm-linux-gnueabihf/lib/ld-linux-armhf.so.3 /lib/
- export LD_LIBRARY_PATH=/usr/arm-linux-gnueabihf/lib

What does Those Commands ?

First let’s see what the command “Sudo apt-get update “ do :

- The role of "sudo apt-get update" is to refresh the local package database with the latest information about packages and their versions. By doing so, it ensures that you have the most up-to-date package lists before performing any software installations or upgrades.
- This command is typically executed before installing new software or upgrading existing packages using package managers like apt-get or apt. It allows you to fetch the latest package information from the repositories, providing an accurate representation of the software packages available for installation or upgrade on your system.
- In summary, "sudo apt-get update" ensures that your system is aware of the latest package versions and updates, enabling you to make informed decisions when installing or upgrading software.

What does Those Commands ?

Now let’s see this one “sudo apt-get install build-essential gcc-arm-linux-gnueabihf”
We will install two Packages :

- build-essential: This is the name of a meta-package that includes essential development tools and libraries required for building software on the system. It typically includes packages like gcc (the GNU Compiler Collection), make, libc-dev, and others.
- gcc-arm-linux-gnueabihf: This is the package name for the GNU Embedded Toolchain for the ARM architecture. It provides a set of compilers, libraries, and tools specifically tailored for cross-compiling C, C++, and Assembly code for ARM-based systems.

What does Those Commands ?

The command "arm-linux-gnueabihf-gcc hello.c -o helloarm" is used to compile a C source code file named "hello.c" into an executable file called "helloarm" specifically for the ARM architecture.

Let's break down the command:

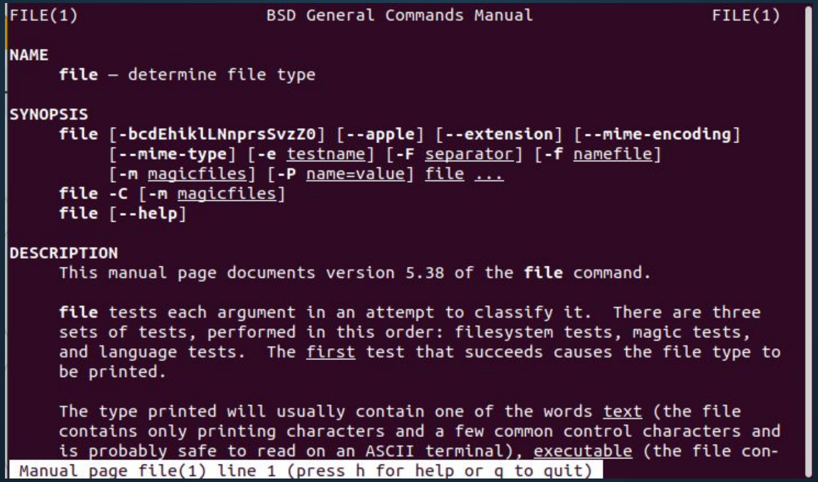
- "arm-linux-gnueabihf-gcc": This is the name of the compiler being used. It indicates that it is a cross-compiler targeting the ARM architecture. Cross-compilation allows you to compile code on one platform (in this case, likely a different architecture, such as x86) for another platform (ARM).
- "hello.c": This is the source code file you want to compile. It should contain the C code for your "hello" program.
- "-o helloarm": This option specifies the output file name. In this case, the compiled executable will be named "helloarm". You can change the name to whatever you prefer.

When you run this command, the compiler will read the source code file "hello.c", compile it for the ARM architecture, and generate an executable file named "helloarm". The resulting executable can then be executed on an ARM-based device or emulator.

Explore the Power of the File Command

Now, let's consider a scenario where you have two executables: one compiled for your host machine and the other for your target machine. How can you determine which executable is intended for your host machine and which one is intended for the target machine? The answer lies in using the powerful utility command called "file."

To access comprehensive information about this command, simply enter "man file" in your terminal. This will provide you with all the necessary details and usage instructions.



```
FILE(1)                  BSD General Commands Manual                 FILE(1)

NAME
     file — determine file type

SYNOPSIS
     file [-bcdEhiklLNnprsSvzZ0] [--apple] [--extension] [--mime-encoding]
          [--mime-type] [-e testname] [-F separator] [-f namefile]
          [-m magicfiles] [-P name=value] file ...
     file -C [-m magicfiles]
     file [--help]

DESCRIPTION
     This manual page documents version 5.38 of the file command.

     file tests each argument in an attempt to classify it.  There are three
     sets of tests, performed in this order: filesystem tests, magic tests,
     and language tests.  The first test that succeeds causes the file type to
     be printed.

     The type printed will usually contain one of the words text (the file
     contains only printing characters and a few common control characters and
     is probably safe to read on an ASCII terminal), executable (the file con-
 Manual page file(1) line 1 (press h for help or q to quit)
```

Explore the Power of the File Command

Now Let’s see the output for the two executables

```
yasser@yasser-Lenovo-IdeaPad-S145-15IWL:~$ file a.out
a.out: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamica
lly linked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=39b1
e2d691815e877fbc4d9ca331cf99b87e3a4c, for GNU/Linux 3.2.0, not stripped
```

```
yasser@yasser-Lenovo-IdeaPad-S145-15IWL:~/Tache1$ file a.out
a.out: ELF 32-bit LSB shared object, ARM, EABI5 version 1 (SYSV), dynam
ically linked, interpreter /lib/ld-linux-armhf.so.3, BuildID[sha1]=cb5d
b9f1c5e5881e0f5069d1f8c410a4357b6469, for GNU/Linux 3.2.0, not stripped
```

Explore the Power of the File Command

```
yasser@yasser-Lenovo-IdeaPad-S145-15IWL:~/Tache1$ file a.out
a.out: ELF 32-bit LSB shared object, ARM, EABI5 version 1 (SYSV), dynam
ically linked, interpreter /lib/ld-linux-armhf.so.3, BuildID[sha1]=cb5d
b9f1c5e5881e0f5069d1f8c410a4357b6469, for GNU/Linux 3.2.0, not stripped
```

The file named "a.out" is an ELF (Executable and Linkable Format) 32-bit LSB (Least Significant Byte) shared object. It is specifically designed for the ARM architecture. The version of the EABI (Embedded Application Binary Interface) used is EABI5, conforming to the SYSV (System V) standard.

The file is dynamically linked, meaning it relies on external libraries during runtime. The interpreter for this executable is located at "/lib/ld-linux-armhf.so.3", which is responsible for loading and executing the program.

The BuildID, represented by the SHA-1 hash "cb5db9f1c5e5881e0f5069d1f8c410a4357b6469," helps identify the specific build or version of the executable.

It is intended to run on GNU/Linux 3.2.0 or a compatible version. Finally, the executable has not been stripped, indicating that debug symbols and additional information are present in the file.

Explore the Power of the File Command

```
yasser@yasser-Lenovo-IdeaPad-S145-15IWL:~$ file a.out
a.out: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamica
lly linked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=39b1
e2d691815e877fbc4d9ca331cf99b87e3a4c, for GNU/Linux 3.2.0, not stripped
```

The file named "a.out" is an ELF (Executable and Linkable Format) 64-bit LSB (Least Significant Byte) shared object. It is specifically compiled for the x86-64 architecture, commonly known as 64-bit Intel or AMD processors. The version of the SYSV (System V) standard used is version 1.

Similar to the previous example, the file is dynamically linked, meaning it depends on external libraries during runtime. The interpreter for this executable is located at "/lib64/ld-linux-x86-64.so.2", responsible for loading and executing the program.

The BuildID, represented by the SHA-1 hash "39b1e2d691815e877fbc4d9ca331cf99b87e3a4c," helps identify the specific build or version of the executable.

This executable is intended to run on GNU/Linux 3.2.0 or a compatible version. Lastly, it has not been stripped, implying that debug symbols and additional information are present in the file.

Qemu Emulator

Now before jumping the next commands let's discuss the qemu emulator. The QEMU emulator (short for Quick Emulator) is an open-source virtualization and emulation software that allows you to run operating systems and programs designed for one architecture on another. It supports emulating various CPU architectures, such as x86, ARM, MIPS, PowerPC, and more.

QEMU can be used in different scenarios, including:

1. System emulation: In this mode, QEMU can emulate a complete computer system, including the CPU, memory, storage devices, and other peripherals. It allows you to run an entire operating system, such as Linux or Windows, on a different architecture or platform.
2. User-mode emulation: QEMU can also work in user-mode emulation, which enables running programs compiled for a specific architecture on a different architecture without the need for full system emulation. This mode is useful when you want to execute binaries compiled for ARM on an x86 system, for example.
3. Virtualization: QEMU can also operate as a hypervisor, providing virtualization capabilities similar to other hypervisors like KVM (Kernel-based Virtual Machine). This allows you to create and manage virtual machines (VMs) running different operating systems simultaneously on a single physical host machine.

QEMU is a versatile and widely used emulator in the virtualization and development communities. It provides a flexible and convenient way to test and run software on different architectures without requiring dedicated hardware.

Qemu Emulator

The command "sudo apt-get install qemu-user" :

The "qemu-user" package specifically provides the user-mode emulation binaries for QEMU.

When you run the command with "sudo" (superuser do), it grants you administrative privileges to install the package. "apt-get" is a command-line package management tool used in Debian-based systems to handle software installations and updates.

So, the overall purpose of the command is to install the QEMU user-mode emulation package on your system, enabling you to run programs and operating systems designed for different architectures than your own.

What does Those Commands ?

If we attempt to execute our file using qemu-user by running the command "qemu-user helloarm," an error occurs. The error message states that the interpreter of our executable lacks a required library. To resolve this issue, we can create a symbolic link of the library in question within the /lib/ directory. By doing this, when qemu-user searches for the file, it will locate it in the /lib/ directory. To accomplish this, you can use the following command: "sudo ln -s /usr/arm-linux-gnueabihf/lib/ld-linux-armhf.so.3 /lib/."

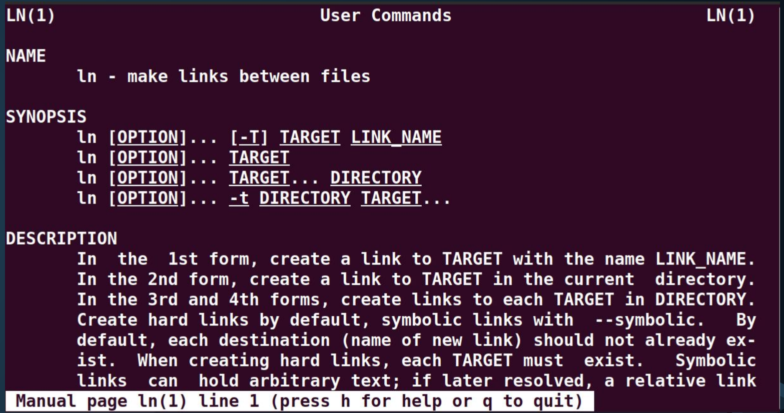
When you install qemu-user, it typically searches for the required libraries in the system's default library directories. These directories are specified in the system's library search path.

In Linux systems, the library search path is defined by the environment variable LD_LIBRARY_PATH and a set of default directories configured by the system. The default directories include /lib, /usr/lib, and additional architecture-specific directories like /usr/lib32 or /usr/lib64.

When qemu-user is installed, it registers its own set of library directories specific to the architecture it supports. For example, if you install qemu-user for ARM emulation, it may add /usr/arm-linux-gnueabihf/lib or similar directories to the library search path.

By adding these directories to the search path, qemu-user ensures that it can find the required libraries for executing programs designed for different architectures.

It's worth noting that the library search path can be modified and customized by system administrators or users by modifying the LD_LIBRARY_PATH environment variable or updating system configuration files.



```
LN(1)                          User Commands                          LN(1)

NAME
       ln - make links between files

SYNOPSIS
       ln [OPTION]... [-T] TARGET LINK_NAME
       ln [OPTION]... TARGET
       ln [OPTION]... TARGET... DIRECTORY
       ln [OPTION]... -t DIRECTORY TARGET...

DESCRIPTION
       In  the  1st form, create a link to TARGET with the name LINK_NAME.
       In the 2nd form, create a link to TARGET in the current  directory.
       In the 3rd and 4th forms, create links to each TARGET in DIRECTORY.
       Create hard links by default, symbolic links with  --symbolic.   By
       default, each destination (name of new link) should not already ex-
       ist.  When creating hard links, each TARGET must  exist.   Symbolic
       links  can  hold arbitrary text; if later resolved, a relative link
 Manual page ln(1) line 1 (press h for help or q to quit)
```

What does Those Commands ?

If we execute the file again using qemu-user, we will encounter a new error related to a different library. To resolve this issue permanently, we can set the LD_LIBRARY_PATH environment variable, which specifies the paths to the required libraries. We need to add the path of the ARM library to this variable.

To make the changes persistent, we should add this variable to the .bashrc file. Afterward, we can source the .bashrc file to apply the modifications. With these adjustments in place, when we run qemu-user, it will be aware of the library paths it needs and will successfully execute the file.

```
      *) return;;
esac
export LD_LIBRARY_PATH="/usr/arm-linux-gnueabihf/lib"

# don't put duplicate lines or lines starting with space in the history.
```

Bashrc File !

The .bashrc file is a script file used by the Bash (Bourne Again SHell) environment in Unix-like operating systems. It is executed every time a new interactive Bash shell session is started.

The primary purpose of the .bashrc file is to define and configure various settings and behaviors for the Bash shell. It allows you to customize your shell environment according to your preferences. Some common configurations found in the .bashrc file include:

1. Setting environment variables: You can define and export environment variables that will be available to all shell sessions. These variables can control various aspects of the shell's behavior, such as the PATH variable for executable search paths or the PS1 variable for the shell prompt.
2. Defining aliases: Aliases are custom shorthand commands that expand to longer commands. You can define aliases in the .bashrc file to create shortcuts for frequently used commands or to modify the behavior of existing commands.
3. Configuring command prompt: You can customize the appearance of your shell prompt by modifying the PS1 variable in the .bashrc file. This allows you to display useful information like username, hostname, current directory, or git branch.
4. Setting shell options: The .bashrc file can be used to enable or disable various shell options. For example, you can enable case-insensitive tab completion, set the history size, or configure command line editing options.

By modifying the .bashrc file, you can tailor your shell environment to your specific needs and preferences, making your command line experience more efficient and comfortable.

Please take pleasure in watching our YouTube video showcasing our First lab.