Introduction to Embedded Linux and the Yocto Project

Four Elements of Embedded Linux

1. **Toolchain**
    - The compiler and other tools needed to create code for your target device.
    - Everything else depends on the toolchain.
2. **Bootloader**
    - The program that initializes the board and loads the Linux kernel.
3. **Kernel**
    - The heart of the system, managing system resources and interfacing with hardware.
4. **Root Filesystem**
    - Contains the libraries and programs that run once the kernel completes initialization.

Additional Element

- A collection of programs specific to your embedded application, making the device perform its intended function, such as:
    - Weighing groceries
    - Displaying movies
    - Controlling a robot
    - Flying a drone

What is Yocto?

- Yocto is the smallest SI metric system prefix.
- For example, 'm' stands for milli (10^-3), similarly 'y' (yocto) stands for 10^-24.

What is the Yocto Project?

- Provides open-source, high-quality infrastructure and tools to help developers create custom Linux distributions for any hardware architecture.

History

- Founded in 2010 to reduce work duplication and provide resources for new and experienced users.
- Collaboration of:
    - Many hardware manufacturers
    - Open-source operating system vendors
    - Electronics companies
- Yocto is a project working group of the Linux Foundation.

Advantages of the Yocto Project

1. **Widely Adopted Across the Industry**
    - Supported by companies like Intel, Facebook, ARM, Juniper Networks, LG, AMD, NXP, and DELL.
2. **Architecture Agnostic**
    - Supports various architectures (Intel, ARM, MIPS, AMD, PPC, etc.)
    - Supports custom silicon through BSP creation
    - Fully supports device emulation via QEMU
3. **Images and Code Transfer Easily**
    - Easily transferable across different architectures without new development environments.
4. **Flexibility**
    - Enables customization through layering.
5. **Ideal for Embedded and IoT Devices**
    - Creates minimal Linux distributions with only necessary components.
6. **Layer Model Usage**
    - Groups related functionality into separate bundles for easy customization.

Understanding the Yocto Project

Input:

- Data specifying the desired output (Kernel configuration, Hardware name, Packages/Binaries to be installed).

Output:

- Linux-based embedded product (Kernel, Root Filesystem, Bootloader, Device Tree, Toolchain).

Setting Up a Build Machine

Prerequisites

1. 50 GB of free disk space.
2. Supported Linux distribution (Fedora, openSUSE, CentOS, Debian, Ubuntu).
3. Required software:
   - Git 1.8.3.1 or greater
   - Tar 1.27 or greater
   - Python 3.4.0 or greater

Installing Required Packages

```
sudo apt-get install gawk wget git-core diffstat unzip texinfo gcc-multilib \
  build-essential chrpath socat cpio python python3 python3-pip python3-pexpect \
  xz-utils debianutils iputils-ping python3-git python3-jinja2 libegl1-mesa libsdl1.2-dev \
  pylint3 xterm
```

Poky

- Poky is the reference distribution of the Yocto Project.
- Contains components:
  - Bitbake
  - OpenEmbedded Core
  - meta-yocto-bsp
  - Documentation

- It does not contain binary files; it's an example for building custom Linux distributions.

Difference Between Poky and Yocto

- Yocto refers to the organization, while Poky refers to the downloadable build system.

Metadata in Yocto

- Metadata refers to the build instructions and consists of:
    - Configuration files (.conf)
    - Recipes (.bb, .bbappend)
    - Classes (.bbclass)
    - Includes (.inc)

OpenEmbedded Project

- Provides a cross-compile environment to create complete Linux distributions for embedded systems.

Differences Between OpenEmbedded and Yocto

- Yocto Project focuses on tools, metadata, and BSPs.
- OpenEmbedded provides metadata for various architectures and features.

Bitbake

- A core component of the Yocto Project, similar to make.
- Task scheduler that parses Python and shell script mixed code.

Building Yocto Project

Steps

1. **Download the Poky Source Code:**

git clone git://git.yoctoproject.org/poky

2. **Checkout the Latest Branch:**

git checkout scarthgap

3. **Prepare the Build Environment:**

source poky/oe-init-build-env

4. **Building Linux Distribution:**

5. bitbake core-image-minimal

Running the Image in QEMU

runqemu <machine> <zimage> <filesystem>

- Example:

runqemu qemuarm zimage-qemuarm.bin filesystem-qemuarm.ext2

Exiting QEMU

- Use Ctrl-C or shutdown via the GUI.

Generating ARM Image and Running in QEMU

1. Edit local.conf file and set:

MACHINE = "qemuarm"

2. Build the image:

3. source poky/oe-init-build-env

4. bitbake core-image-minimal

runqemu core-image-minimal

Adding Packages to Root Filesystem

1. Open local.conf and add:

IMAGE_INSTALL += "recipe-name"

2. Example:

IMAGE_INSTALL += "usbutils"

Challenge

Question:

- What changes would you make to generate an image for QEMU MIPS and run it in the QEMU emulator?
