Using Open Source Software to Build an Industrial-grade Embedded Linux Platform from Scratch

SZ Lin (林上智)
Embedded Linux Development Center, Software R&D Engineer
07/19, 2019
About Me

SZ LIN (林上智)

- Software Engineer at Moxa
- Cybersecurity Fundamentals Specialist
  - ISA/ IEC 62443
- Debian Developer
- Blog - https://szlin.me
Industrial Embedded Linux Platforms

Application

Smart Rail  Smart Grid  Smart Oil Field  Smart Transportation  Smart Factory  Smart Marine

Edge Connectivity

Serial Connectivity  I/O Connectivity  Video Connectivity

Industrial Computing

Embedded Computers

Network Infrastructure

Industrial Ethernet  Industrial Wireless LAN  Industrial Routers

Device

3
Before Using Open Source Software

Something You Should Know
Copyright

Copyright is a legal right, that grants the creator of an original work exclusive rights to determine whether, and under what conditions, this original work may be used by others.

src: https://en.wikipedia.org/wiki/Copyright

Patent

A patent gives its owner the right to exclude others from making, using, selling, and importing an invention for a limited period of time, usually twenty years.

src: https://en.wikipedia.org/wiki/Patent
Identify key recommended processes for effective open source management [1].

It is a shared defensive patent pool with the mission to protect Linux [2].
The OpenChain Project builds trust in open source by making open source license compliance simpler and more consistent.

Software Package Data Exchange (SPDX) is a file format used to document information on the software licenses under which a given piece of computer software is distributed.

FOSSology is a open source license compliance software system and toolkit.
Industrial/ Harsh Environments

Including smart rail, smart grid, intelligent transportation, factory automation, oil & gas, marine, and more
Target Application

Longevity + Stability + Security

- Performance
- Real-time
- Resource Limited
- Safety
Target Application

Hardware

Software

Linux

Real-time Safety Resource Limited Software Hardware
Lifecycle of Industrial-grade Embedded Linux Platform

Development Phase

1. System Preparation
2. Build and Testing

Maintenance Phase

3. Long-term Test
4. Regular Update

MOXA
Development Phase

Design and development according to application
Choose Proper Bootloader
Bootloader behavior
(A case of ARM-based architecture)
<table>
<thead>
<tr>
<th>Category</th>
<th>License</th>
<th>Supported Platforms</th>
<th>Supported UEFI</th>
<th>Maintainer</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Das U-Boot</strong></td>
<td><strong>GPL-2+</strong></td>
<td>68k, ARM, Blackfin, MicroBlaze, MIPS, Nios, SuperH, PPC, RISC-V, x86 (on top of Coreboot)</td>
<td>Y</td>
<td>DENX Software Engineering</td>
</tr>
<tr>
<td><strong>Coreboot</strong></td>
<td><strong>GPL-2</strong></td>
<td>IA-32, x86-64, ARMv7, ARMv8, MIPS, RISC-V, POWER8</td>
<td>Y</td>
<td>coreboot.org</td>
</tr>
<tr>
<td><strong>GRUB</strong></td>
<td><strong>GPL-3</strong></td>
<td>IA-32, x86-64, IA-64, ARM, PowerPC, MIPS and SPARC</td>
<td>Y</td>
<td>GNU Project</td>
</tr>
<tr>
<td><strong>rEFInd</strong></td>
<td>GNU GPLv3, Modified BSD License (original program), additional components released under various licenses</td>
<td>x86, x86-64, or ARM64</td>
<td>Y</td>
<td>Roderick W. Smith</td>
</tr>
</tbody>
</table>
Kernel Space
Choose Proper Kernel

Based on the application requirement
# Linux Kernel Comparison Table

<table>
<thead>
<tr>
<th>Category</th>
<th>Latest version</th>
<th>Target Application</th>
<th>Maintainer</th>
</tr>
</thead>
</table>
| Linux kernel           | 5.2            | • Performance  
• Resource Limited [12][13]                   | Kernel.org                                      |
| Preempt RT kernel      | 5.2            | • Real-time  
• Functional safety  
• Resource Limited                   | Real Time Linux collaborative project           |

*Real-time application [14][15]
SoC Board Support Package Kernel

• Kernel version depends on SoC vendors
  – Well made but not well maintained

• Contain lots of in-house patches
  – Errata patches
  – Specific feature patches
  – ...

• Different SoC might use different versions of kernel

• The lifetime is unsure
LTS: Long Term Stable Kernel

Longterm release kernels

<table>
<thead>
<tr>
<th>Version</th>
<th>Maintainer</th>
<th>Released</th>
<th>Projected EOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.14</td>
<td>Greg Kroah-Hartman</td>
<td>2017-11-12</td>
<td>Jan, 2020</td>
</tr>
<tr>
<td>4.9</td>
<td>Greg Kroah-Hartman</td>
<td>2016-12-11</td>
<td>Jan, 2023</td>
</tr>
<tr>
<td>4.4</td>
<td>Greg Kroah-Hartman</td>
<td>2016-01-10</td>
<td>Feb, 2022</td>
</tr>
<tr>
<td>3.16</td>
<td>Ben Hutchings</td>
<td>2014-08-03</td>
<td>Apr, 2020</td>
</tr>
</tbody>
</table>

Extend software uptime for stable kernel

• Only accept bug fixes and security fixes

img: https://www.kernel.org/category/releases.html
LTSI: Long Term Support Initiative

• Linux Foundation collaborative project
  – Based on LTS
  – Add another chance to include further patches on top of LTS
  – Auto Test framework
  – Same lifetime with LTS (yearly release and 2 years life time)
CIP (Civil Infrastructure Platform) 

• Linux Foundation collaborative project
  – Support kernel and core package
  – Auto Test framework
  – Maintenance period
    • 10 years and more (10-20 years)
# Linux Kernel Source Comparison Table

<table>
<thead>
<tr>
<th>Version</th>
<th>Maintenance Period (years)</th>
<th>Features</th>
<th>Latest Version</th>
<th>Supported Realtime kernel</th>
<th>Maintainer</th>
</tr>
</thead>
<tbody>
<tr>
<td>SoC BSP kernel</td>
<td>?</td>
<td>Bug fixes</td>
<td>?</td>
<td>N</td>
<td>SoC vendor kernel team</td>
</tr>
<tr>
<td>LTS kernel</td>
<td>2 ~ ?</td>
<td>• Bug fixes</td>
<td>4.19</td>
<td>N</td>
<td>Kernel.org</td>
</tr>
<tr>
<td>LTSI kernel</td>
<td>2 ~ ?</td>
<td>• Bug fixes</td>
<td>4.14</td>
<td>N</td>
<td>LTSI</td>
</tr>
<tr>
<td>CIP kernel</td>
<td>10 +</td>
<td>• Bug fixes</td>
<td>4.19</td>
<td>Y</td>
<td>CIP</td>
</tr>
</tbody>
</table>
Longevity + Stability + Security

Mutually Exclusive?
Multiple Kernel In Single Platform

To fulfill multiple user scenarios
FIT (Flattened Image Tree)  
(A case of ARM-based architecture)

• Tree data structure
• Handle multiple types of image
  – kernel : kernel image
  – fdt : dtb file
  – ramdisk : root file system
• Image hashing
  – md5
  – sha1
• Image signing
• Each node in configurations has their image configuration in booting stage
More info.:
http://git.denx.de/?p=u-boot.git;a=blob_plain;f=doc/uImage.FIT/source_file_format.txt;hb=HEAD

Advancing Open Source Safety-Critical Systems

The mission of the Enabling Linux In Safety Applications (ELISA) project is to make it easier for companies to build and certify Linux-based safety-critical applications – systems whose failure could result in loss of human life, significant property damage or environmental damage. ELISA members are working together to define and maintain a common set of tools and processes that can help companies demonstrate that a Linux-based system meets the necessary safety requirements for certification.

- **Linux Foundation collaborative project**
  - Build and certify Linux-based safety-critical applications
  - Define and maintain a common set of tools and processes
    - SIL2LinuxMP [21] project and the Linux Foundation’s Real-Time Linux project
  - IEC 61508
Choose Proper C Library and Toolchain
## C Library and Toolchain Comparison Table

<table>
<thead>
<tr>
<th>Category</th>
<th>License</th>
<th>Features</th>
<th>Target Application</th>
<th>Maintainer User</th>
</tr>
</thead>
</table>
| **glibc** [25] | LGPL 2.1 | • Stable ABI  
• Backward compatibility  
• Fully symbol versioning  
• Stack smashing protection/ heap corruption detection  
• Profiling | • Performance  
• Security | GNU |
| **uClibc-ng** [26] | LGPL 2.1 | • No-MMU architecture support  
• Tiny size | • Resource Limited | uclibc-ng.org |
| **Musl** [28] | MIT | • Stable ABI  
• Backward compatibility  
• Stack smashing protection/ heap corruption detection | • Resource Limited  
• Security | musl-libc.org |

Other option [93]  
* Be aware of year 2038 problem [29]
Year 2038 Problem [92]

- The `time_t` datatype is a data type in the ISO C library and kernel structure defined for storing system time values.

- 32-bit system can represent dates from
  - Dec 13 1901
  - Jan 19th 2038

- It causes integer overflowing on 03:14:08 UTC 19 January 2038
Init System
## Init System Comparison Table

<table>
<thead>
<tr>
<th>Category</th>
<th>License</th>
<th>C Library</th>
<th>User</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>busybox</td>
<td>GPL 2.0</td>
<td>uClintex-ng Glibc musl</td>
<td>ProteanOS PiBox</td>
<td>Resource-limited application</td>
</tr>
<tr>
<td>sysvinit</td>
<td>GPL 2.0+</td>
<td>uClintex-ng glibc musl</td>
<td>Devuan</td>
<td></td>
</tr>
<tr>
<td>systemd</td>
<td>LGPL 2.1+</td>
<td>glibc</td>
<td>Arch, CentOS, CoreOS, Debian, Fedora, Mint, OpenSUSE, Redhat, Ubuntu</td>
<td>Linux only</td>
</tr>
<tr>
<td>openrc</td>
<td>2-clause BSD</td>
<td>musl glibc</td>
<td>Gentoo Alpine Linux</td>
<td></td>
</tr>
<tr>
<td>upstart</td>
<td>GPL 2.0</td>
<td>glibc</td>
<td>Chromium OS</td>
<td>Linux only</td>
</tr>
</tbody>
</table>
Choose proper RFS (Root filesystem)

Stable root filesystem
# Root filesystem Comparison Table

<table>
<thead>
<tr>
<th>Category</th>
<th>Maintenance Period (years)</th>
<th>Number of packages</th>
<th>C Library</th>
<th>Security Tracker</th>
<th>CI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Busybox</td>
<td>?</td>
<td>300 ~ 400 applets</td>
<td>• uClibc</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• glibc</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Yocto</td>
<td>Latest release the previous two releases</td>
<td>It depends on meta-*</td>
<td>• glibc</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• musl</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Buildroot</td>
<td>1</td>
<td>2000+ [42]</td>
<td>• glibc</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• musl</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• uClibc-ng</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Debian</td>
<td>3 + 2 (i386, amd64, armel, armhf and arm64)</td>
<td>51000+</td>
<td>• glibc</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• musl</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
System Development Tools
## System Development Tools Comparison Table

<table>
<thead>
<tr>
<th>Root filesystem</th>
<th>System Development Tools</th>
<th>Toolchain</th>
<th>System Development Tools License</th>
</tr>
</thead>
<tbody>
<tr>
<td>Busybox</td>
<td>Yocto</td>
<td>OE-Core</td>
<td>MIT</td>
</tr>
<tr>
<td>Yocto</td>
<td>Yocto</td>
<td>OE-Core</td>
<td>MIT</td>
</tr>
<tr>
<td>Buildroot</td>
<td>Buildroot</td>
<td>Buildroot</td>
<td>GPL 2.0+</td>
</tr>
<tr>
<td>Debian</td>
<td>ISAR</td>
<td>Debian toolchain</td>
<td>Metadata: MIT</td>
</tr>
<tr>
<td></td>
<td>ELBE</td>
<td>Debian toolchain</td>
<td>Others: GPL 2.0</td>
</tr>
<tr>
<td></td>
<td>Yocto Deby (meta-debian)</td>
<td>OE-Core</td>
<td>MIT</td>
</tr>
<tr>
<td></td>
<td>Live-build</td>
<td>Debian toolchain</td>
<td>GPL 3.0+</td>
</tr>
</tbody>
</table>
Why We Choose Debian

**Stability**
unstable → testing → stable

**Scalability**
Server, Desktop, Laptop, Embedded devices

**Good system security**
Everything is open
Usually, fixed packages are uploaded within a few days

**Long term support**
5 more years by Debian-LTS project (i386, amd64, armel, armhf and arm64)

**Multiple architectures**
alpha, amd64, armel, armhf, aarch64, hppa, i386, ia64, mips, mipsel, powerpc, s390, and spar

**Incredible amounts of software**
Debian comes with over 51000 different pieces of software with free
Development Phase

Build and Testing

1. System Preparation
2. Build and Testing
3. Long-term Test
4. Regular Update

More info: Building, Deploying and Testing an Industrial Linux Platform
Open Source Summit Japan 2017 [51]
CI/CD Automatic Release Pipeline

Continuous Integration

Code

Building

1

Continuous Delivery

Deploying

2

Testing

3

Continuous Deployment

Release

4
CI/CD Automatic Release Pipeline
Server

slave node

Static Program Analysis

- Coding style
- OWASP [52]
- Infer [53]
- Sonarqube [54]
- ...

Local Branch

Pull Request

Internal/External Developers

Webhooks

GitLab

Patches

Slave node

Slave node

Slave node

A tool to detect bugs in Java and C/C++/Objective-C code before it ships

Infer is a static analysis tool that can find bugs in Java or C/C++/Objective-C code. It produces a list of potential bugs. Anyone can use Infer to intercept critical bugs before they have shipped to users, and help prevent crashes or poor performance.

The leading platform for Continuous Code Quality

Download 6.3.1 Use Online

MOXA®
Static Testing Cases Management - Jenkins
The #1 programmer excuse for legitimately slacking off: “My code’s compiling.”

Hey! Get back to work!

Compiling!

Oh. Carry on.
Distributed Compiler

• **Software**
  – *Icecream/ IceCC was created by SUSE based on distcc* [55][56]
    • Improve performance of compile jobs in parallel
    • Add dynamic scheduler of the compilation jobs
    • Support multiple platform
    • Support cross compiling

• **Hardware - for each node**
  – SSD
  – Large capacity memory
  – Gigabit LAN
1. Send command

2. Get source

3. Send log & image

ICECC - Distributed Compiler

Webhooks

Pull Request

Local Branch

Patches

Internal/ External Developers

Client

Client

Client
CI/CD Automatic Release Pipeline

Continuous Integration

Code

Continuous Delivery

Building

Deploying

Testing

Release
Continuous Delivery – LAVA

Introduction to LAVA

Navigation
Use the navigation bar at the top of each page to quickly navigate between sections of the documentation; Index, Contents, Page and Next.

Index
The Help Index is often the quickest way to find specific sections of the documentation.

Contents
If you are new to LAVA, the Help Contents describes several useful starting points, depending on how you expect to use LAVA.

Page indices
Each page also has a Page menu for topics within the page as well as forward and back navigation to lead readers through in a logical manner.

About LAVA V2
LAVA V2 is the second major version of LAVA. The major user-visible features are:

- The Pipeline model for the dispatcher
- YAML job submissions
- Results
- Queries
- Charts
- Data export APIs

The architecture has been significantly improved since V1, bringing major changes in terms of how a distributed LAVA instance is installed, configured and used for running test jobs.
1. Send job file via XML-RPC

2. Dispatch job via ZMQ

3. Download image via curl or wget

4. Boot up via Ethernet remote I/O

5. Deployment via TFTP

6. Trigger test framework

7. Send test cases

8. Start testing

9. Send back testing result

Platform Test
- LTP [62]
- Security testing [63]
- Kselftest [67]
- ...

Dynamic Program Analysis
- gcov [59]
- valgrind [60]
- profiling tools [61]
- ...

Server

Worker

Worker

Worker

Master

DUT Clusters

(Test framework)
• **Test framework for testing embedded Linux**
  – *Official automated test framework for the LTSI project.*
    • BSD 3-Clause license in default
    • Over 100 pre-packaged tests
  – *Ability for 3rd parties to initiate or schedule tests on our hardware, and the ability to share our test results with others.*
Maintenance Phase

Long-term Testing and Regular Update
Maintenance Phase

Long-term Test

System Preparation
Build and Testing

Regular Update

More info: Building, Deploying and Testing an Industrial Linux Platform
Open Source Summit Japan 2017
* Test cases are managed by test framework
24/7 Long-term Platform Test

Endurance test
Compatibility test ...

- **Longevity**
  Long-term support at least **10 years life cycle** with bug fixes, new features and new hardware components

- **Robustness**
  Robustness is the ability of a computer system to cope with errors during execution and cope with erroneous input [71]

- **Reliability**
  Reliability is enhanced by features that help to avoid, detect and repair hardware faults [72]

- **Security**
  Quick response in resolving CVE/vulnerabilities and attacks in platform
24/7 Long-term Platform Test

- **Longevity**: Long-term support at least 10 years life cycle with bug fixes, new features and new hardware components.

- **Robustness**: Robustness is the ability of a computer system to cope with errors during execution and cope with erroneous input [71].

- **Reliability**: Reliability is enhanced by features that help to avoid, detect and repair hardware faults [72].

- **Security**: Quick response in resolving CVE/vulnerabilities and attacks in platform.

Fuzz testing [64][65][66] …
24/7 Long-term Platform Test

Power failure test
Reboot test
Regression test
...

Reliability
Reliability is enhanced by features that help to avoid, detect and repair hardware faults [72]

Security
Quick response in resolving CVE/ vulnerabilities and attacks in platform
24/7 Long-term Platform Test

Daily test for CVE [63] ...

Security

Quick response in resolving CVE/ vulnerabilities and attacks in platform
For Stable Kernel Maintenance

KernelCI

• Automated Linux Kernel Testing [73][74]
  – Detect, bisect, report and fix regressions on upstream Kernel trees before release
  – Short tests on many configurations
Reproducible Builds

- Create an independently-verifiable path from source to binary
  - Ensure builds have identical results
  - Act as part of a chain of trust
  - Prove the source code has not been tampered/modified
# Open Source Testing Tools

<table>
<thead>
<tr>
<th>Category</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Continuous Integration</td>
<td>Jenkins [78], Jenkins X [79]</td>
</tr>
<tr>
<td>Continuous Delivery/ Deployment</td>
<td>LAVA 2 [57]</td>
</tr>
<tr>
<td>Distributed compiler service</td>
<td>icecc [55], GOMA [80][81], distcc [82]</td>
</tr>
<tr>
<td>Test Case Management</td>
<td>Jenkins, LAVA 2, Fuego [68][69]</td>
</tr>
<tr>
<td>Version Control</td>
<td>Git with gitlab [83]</td>
</tr>
<tr>
<td>Static Program Analysis</td>
<td>Coding style, OWASP [52], Infer [53], Sonarqube [54]</td>
</tr>
<tr>
<td>Dynamic Program Analysis</td>
<td>Gcov [59], Valgrind [60], Profiling tools [61]</td>
</tr>
<tr>
<td>Security Testing</td>
<td>OpenVAS [63], Vuls [84]</td>
</tr>
<tr>
<td>Fuzzing Testing</td>
<td>Syzkaller [64], Trinity [65], OSS-fuzz [66]</td>
</tr>
</tbody>
</table>
CI/CD/LT are **concepts of software engineering** instead of tools or procedures.
Maintenance Phase

1. Hardware Survey
2. Kernel/ RFS Preparation
3. Long-term Test
4. Regular Update
Why We Need Software Update?

- Bug fixes
- Security fixes
- New userspace program
- New kernel features

- Over 10+ years

- Maintenance release
## The Components Might Be Updated

<table>
<thead>
<tr>
<th>Components</th>
<th>Size</th>
<th>Update frequency</th>
<th>Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peripheral devices firmware</td>
<td>&lt; 10 MB</td>
<td>Rarely</td>
<td>Mid</td>
</tr>
<tr>
<td>Bootloader (including SPL)</td>
<td>&lt; 1 MB</td>
<td>Rarely</td>
<td>High</td>
</tr>
<tr>
<td>Device tree</td>
<td>&lt;100 kB</td>
<td>Rarely</td>
<td>High</td>
</tr>
<tr>
<td>Linux kernel</td>
<td>&lt; 10 MB</td>
<td>Regularly</td>
<td>High</td>
</tr>
<tr>
<td>Root file system</td>
<td>Variant</td>
<td>Regularly</td>
<td>High</td>
</tr>
<tr>
<td>System configuration</td>
<td>&lt; 1 MB</td>
<td>Rarely</td>
<td>Low</td>
</tr>
<tr>
<td>Application</td>
<td>Variant</td>
<td>Often</td>
<td>Low</td>
</tr>
</tbody>
</table>
Characteristics of Industrial Embedded Linux Platform

1. **Harsh environment**
   - Unreliable network and power supply

2. **Middle of nowhere**
   - Human-less warehouse or site

3. **Bandwidth limited**
   - Wireless focus

4. **Multiple version supported**
   - Rollback version

5. **Multiple devices**
   - Remote management

6. **Longevity**
   - Long-term support at least 10 years life cycle
The Media for Software Update

- Wire cable
- OTA
- Portable storage
- On-site
## Software Update Requirements

<table>
<thead>
<tr>
<th>Basic Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fail-safe</td>
</tr>
<tr>
<td>Roll-back</td>
</tr>
<tr>
<td>Size reduction</td>
</tr>
<tr>
<td>Signatures</td>
</tr>
<tr>
<td>Multiple storage type support (e.g., NOR/NAND flash, eMMC)</td>
</tr>
<tr>
<td>Build system integration</td>
</tr>
<tr>
<td>Remote access (e.g., OTA)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Additional Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Online and offline updates</td>
</tr>
<tr>
<td>Encryption</td>
</tr>
<tr>
<td>Delta-updates</td>
</tr>
<tr>
<td>Successful update detection</td>
</tr>
<tr>
<td>Proactive updating</td>
</tr>
</tbody>
</table>
# Update Approaches

<table>
<thead>
<tr>
<th>Components</th>
<th>Size</th>
<th>Complexity</th>
<th>Time Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image/ block based</td>
<td>Large</td>
<td>Low</td>
<td>Very High</td>
</tr>
<tr>
<td>File based</td>
<td>Variant</td>
<td>Low</td>
<td>Variant</td>
</tr>
<tr>
<td>Package based (e.g., deb, rpm)</td>
<td>Variant</td>
<td>Low</td>
<td>Variant</td>
</tr>
<tr>
<td>Delta based</td>
<td>Low</td>
<td>Very High</td>
<td>Low</td>
</tr>
</tbody>
</table>
Partition Architecture

OS

Bootloader/ Firmware

Linux kernel, (device tree)

Middleware/ Libraries

Application Framework

Application

User Data/ Configuration

Root file system
Asymmetric/ Symmetric Firmware Updates

**Asymmetric Firmware Updates**
- Fail-safe
- Downtime

**Symmetric Firmware Updates**
- Seamless update
- Roll-back
- Fail-safe
- Double copy of OS
## Comparison - Features

<table>
<thead>
<tr>
<th>Category</th>
<th>Fail-Safe</th>
<th>Roll-Back</th>
<th>Delta-Updates</th>
<th>Signatures</th>
<th>Multiple Storage Type Support</th>
<th>Build System Integration</th>
</tr>
</thead>
<tbody>
<tr>
<td>SWUpdate</td>
<td>Y</td>
<td>Y</td>
<td>librsync</td>
<td>Y</td>
<td>• NOR NAND flashes</td>
<td>Yocto/Buildroot</td>
</tr>
<tr>
<td>RAUC</td>
<td>Y</td>
<td>Y</td>
<td>casync</td>
<td>Y</td>
<td>• NOR NAND flashes</td>
<td>Yocto/Buildroot</td>
</tr>
<tr>
<td>OSTree</td>
<td>N</td>
<td>Y</td>
<td>archive-z2</td>
<td>Y</td>
<td>?</td>
<td>Yocto</td>
</tr>
</tbody>
</table>
## Comparison - Others

<table>
<thead>
<tr>
<th>Method</th>
<th>Asymmetric/ Symmetric Image Updates</th>
<th>Type</th>
<th>Language</th>
<th>License</th>
</tr>
</thead>
<tbody>
<tr>
<td>SWUpdate</td>
<td>Both</td>
<td>Image-based</td>
<td>C99</td>
<td>GPLv2 With openssl exception</td>
</tr>
<tr>
<td>RAUC</td>
<td>Both</td>
<td>Image-based</td>
<td>C</td>
<td>LGPLv2.1</td>
</tr>
<tr>
<td>OSTree</td>
<td>Asymmetric</td>
<td>File-based</td>
<td>C/C++</td>
<td>MPL 2.0 /LGPLv2+</td>
</tr>
</tbody>
</table>
Conclusion

Preparedness Planning

Longevity, stability and security

Community Collaboration

Different approach for multiple target applications
Thank You
Q & A
References

[1] https://www.openchainproject.org
[10] https://www.kernel.org
[12] https://tiny.wiki.kernel.org/start
[14] https://xenomai.org/
References

[17] https://ltsi.linuxfoundation.org/
[19] https://www.cip-project.org/
[20] https://elisa.tech/
[23] https://lists.elisa.tech/login?r=%2Ftopics
[26] https://uclibc-ng.org/
[28] https://www.musl-libc.org/
[29] https://en.wikipedia.org/wiki/Year_2038_problem
References

[33] https://sysdfree.wordpress.com/2019/03/09/135/
[34] https://wiki.gentoo.org/wiki/Comparison_of_init_systems
[37] https://www.piboxproject.com/
[38] https://lists.debian.org/debian-devel/2016/02/msg00122.html
[39] https://busybox.net/FAQ.html#libc
[40] https://wiki.yoctoproject.org/wiki/Stable_branch_maintenance
[41] https://www.debian.org/social_contract#guidelines
References

[45] https://github.com/meta-debian/meta-debian
References

[52] https://wiki.jenkins-ci.org/display/JENKINS/Plugins
[54] https://www.sonarqube.org/
[55] https://github.com/icecc
[56] https://www.slideshare.net/szlin/distributed-compiler-icecc
[57] https://validation.linaro.org/static/docs/v2/
[61] https://perf.wiki.kernel.org/index.php/Main_Page
[64] https://github.com/google/syzkaller
[66] https://github.com/google/oss-fuzz
[67] https://kselftest.wiki.kernel.org
References

[68] https://elinux.org/Fuego
[70] https://elinux.org/Automated_Testing_Summit_2019
[71] https://en.wikipedia.org/wiki/Robustness_(computer_science)
[73] https://kernelci.org/
[75] https://reproducible-builds.org/
[77] https://wiki.debian.org/ReproducibleBuilds
[78] https://jenkins.io
[79] https://jenkins.io/projects/jenkins-x/
[80] https://chromium.googlesource.com/infra/goma/server/
[81] https://chromium.googlesource.com/infra/goma/client
References

[82] https://github.com/distcc/distcc
[83] https://about.gitlab.com/
[84] https://vuls.io/
[93] https://android.googlesource.com/platform/bionic